Osborne/McGraw-Hill

OSBORNE CP/M® USER GUIDE

Second Edition

BY THOM HOGAN

FOR ALL CP/M USERS

OSBORNE
CP/M®
USER GUIDE
Second Edition

OSBORNE CP/M® USER GUIDE

Second Edition

By Thom Hogan

Osborne/ McGraw-Hill
Berkeley, California

Published by
Osborne/McGraw-Hill
630 Bancroft Way
Berkeley, California 94710
U.S.A.

For information on translations and book distributors outside of the U.S.A., please write to Osborne/McGraw-Hill at the above address.

OSBORNE CP/M® USER GUIDE, SECOND EDITION

1234567890 DODO 8765432

ISBN 0-931988-82-9

Cover design and art by Timothy Sullivan

Text design by K.L.T van Genderen

Photos by Harvey Schwartz

The italicized names are trademarks of the following companies (with registered trademarks denoted by ®):

Apple® Apple Computer, Inc.

CBM, PET® Commodore Business Machines Inc.

CP/M®, CP/NET®, CP/M-80, CP/M-86, MP/M II Digital Research Corp., Inc.

IBM® IBM

Smartmodem D.C. Hayes Corp.

Softcard Microsoft Inc.

The Source (Servicemark) Source Telecomputing Corporation

WordStar MicroPro International Corporation

Z80® Zilog, Inc.

This book is dedicated to Lore Harp, Carole Ely, Steve Jobs, Steven Wozniak, Gary Kildall, and Seymour Rubenstein, who gave me the tools to write it.

Contents

Introduction

Your computer is not a single unit but an interrelated system of devices and programs. You must direct these components to carry out any program you wish to run. CP/M-80 and CP/M-86 are operating systems which do much of this job for you. CP/M-80 or CP/M-86 directs the activities of your computer's components and manages files which contain computer instructions or data.

Although CP/M-80 and CP/M-86 are complex computer programs, you can learn to use them without any prior computer experience. This book introduces the novice user to the microcomputer system and examines CP/M's function within that system.

Chapter 1 provides the basic, practical information you need to get started. Chapters 2 and 3 detail the CP/M-80 and CP/M-86 commands. This is information you will use every day, and we recommend that you study the examples carefully. The beginning three chapters of this book provide a solid foundation for understanding what CP/M is and how to use it.

Chapter 6 explains the functions of two CP/M relatives—MP/M and CP/NET—and examines the commands unique to those operating systems. The major differences between Cromemco's CDOS and Digital Research's CP/M-80 are highlighted to enable Cromemco operators to use this book.

Chapters 4 and 7 are written for assembly language programmers who wish to modify CP/M-80 or CP/M-86 or use them for program development. This information is not essential to most CP/M users, but it is provided here to give a more complete discussion of CP/M. We hope that the more courageous readers will be stimulated to make full use of CP/M utilities described in these chapters.

A final section, Chapter 8, distills the author's experience with CP/M-80 and offers a number of helpful hints.

An annotated bibliography provides directions for additional reading, and several appendices offer practical consumer information about CP/M-compatible programs, languages, and products.

Some special introductory comments regarding this revised edition of the *Osborne CP/M® User Guide* are in order.

The primary concerns in rewriting, restructuring, and adding to the text of this second edition were to make the book more accurate and more complete. Since the book was originally written, CP/M-86, MP/M-86, and MP/M II have been introduced and a third revision of CP/M-80 has been designed (CP/M 3.0). And, as happens with many works, the author had second thoughts about how to make some sections more understandable. The CP/M-86 commands and information have been integrated into the text in a manner that allows owners of either CP/M-80 or CP/M-86 to make equal use of this book.

A lot of effort has been expended to make this revision the most complete and accurate manual for CP/M users. If you have a working computer system, read this book while seated in front of your computer. Try the commands and examples presented here; do not just read about them. You will be comfortable with CP/M much sooner than you expected.

One last comment: the terms CP/M, CP/M-80, and CP/M-86 are used in this book to mean specific (and different) versions of the operating system. Whenever we refer to CP/M, we are writing about all versions of CP/M. Whenever we refer to CP/M-80 or CP/M-86, we are referring to specific versions of the operating system.

Books, like computer programs, are never completely free of errors. The author and publisher invite your comments and criticisms.

CHAPTER

1 CP/M and Operating Systems

CP/M is a disk operating system for microcomputers produced by a company named Digital Research. CP/M stands for "Control Program/Monitor." Versions of CP/M are available for a wide variety of microcomputers from a number of different sources. CP/M-80 can be used on almost any microcomputer which uses the 8080 or Z80 central processor unit and has 8-inch or 5¼-inch floppy disk drives. CP/M-86 can be used with almost any microcomputer which uses the 8086 or 8088 central processor unit and has floppy disk drives.

History of CP/M

CP/M was developed in 1973 by Dr. Gary Kildall, who was, at that time, a software consultant for Intel. The earliest version was written for Kildall's own experimental system, which included one of the first 8-inch disk drives built by Shugart Associates—a shopworn drive that had been used for equipment life tests before being passed on to Kildall.

Kildall showed his earliest versions of CP/M to Intel, but that company declined to market or further develop the project. This was not surprising, because in 1973 and 1974 microcomputers were a rarity, and those that had them were not exactly sure what they wanted to do with them.

By 1975 quite a few small companies were marketing microcomputers to curious hobbyists. When most of these companies developed computers that incorporated disk drives, they usually also built their own operating systems. Had these pioneers—Altair, Polymorphic, and Processor Technology, for example—been able to get their products to consumers quickly, CP/M might not have become the "quasi-standard" operating system it is today.

Instead, several small microcomputer manufacturers decided to eliminate costly research and development and adopted Kildall's CP/M operating system for their products. Most notable among these smaller companies were Tarbell Electronics and Digital Microsystems. These two firms were among the first to ship working disk systems. Because these firms manufactured "add-on" components—ones which could be used on virtually any system—owners of Altairs, Vectors, Polys, and other systems did not have to wait for the manufacturer of their computers to produce drives. In addition, IMSAI, another microcomputer pioneer, had been shipping disk systems without any software to run them, yet they promised to ship an operating system as soon as it was ready. This operating system turned out to be IMDOS, which was really a disguised version of CP/M.

Another important element in CP/M history is the enthusiasm of its first users. These true hobbyists tackled normally insurmountable problems in their pursuit of new knowledge and experience. Theoretically, CP/M-80 could link any 8080- or Z80-based microcomputer with any disk system, and a group of hobbyists with "mix and match" systems emerged to test Kildall's product. These hobbyists developed a number of refinements and, more important, a strong and visible users' group.

The support of a strong users' group cannot be underestimated. During the infant years of the microcomputer industry, accurate product information was not readily available. Manufacturers often released products with incomplete documentation; computer stores were still relatively unknown; and in some cases, users' groups were more stable than the companies that developed the product the users' group was formed to support.

After manufacturers began delivering reliable disk drives, software developers launched the next vital phase of CP/M's evolution. The key to making software development financially feasible is to write programs that run on as many different microcomputers as possible. CP/M-80 was one of the few operating systems that could run on just about any 8080- or Z80-based microcomputer, and it was not restricted to only one type of disk drive.

Fortunately for CP/M's development, the first programs that became available were development tools—programs that help programmers generate other programs. Among the development tools that helped establish CP/M as the leading operating system for microcomputers were CBASIC (and its predecessor, EBASIC), Microsoft BASIC, and several special assembly language programs. These tools, in turn, were used to write application programs, such as general ledgers, database and inventory programs, and word processing programs.

The popularity of the CP/M operating system became part of an escalating pattern: CP/M spawned programming languages and development tools, which in turn gave birth to application programs. These CP/M-dependent application programs increased CP/M sales, which again led to an increased number of development tools being introduced. This upward spiral of sales leading to more tools leading to more application programs leading to more sales has continued unabated for several years now, and shows no signs of stopping.

In fact, 1981 was an important year for CP/M. With the introduction of new microcomputers with CP/M operating systems as a standard or as an option by computer giants like IBM, Hewlett-Packard, and Xerox, the number of users of CP/M passed the quarter-million mark during the year. It is estimated that over 300 computer manufacturers now offer CP/M-80 or CP/M-86 with their equipment, and owners of microcomputers which do not have an 8080, 8086, 8088, or Z80 central processor are now able to make use of CP/M with the recent introduction of the Microsoft SoftCard for the Apple and the announcement of an add-on processing board for the Commodore CBM and PET line of computers.

From CP/M's modest beginnings, it has become the most widely used operating system for microcomputers (and possibly for all computers if the number of installations are counted instead of the number of users). Many changes have been made to Kildall's original operating system, but despite its simplicity, there is much to learn about CP/M.

CP/M Manuals

Now *you* come into the picture. You probably purchased this book because you need CP/M to run an application program. The program may be a simple word processing program or a sophisticated accounting system, but both require an understanding of CP/M.

Digital Research's manuals for CP/M were not written for you; they were written for professional programmers. This book attempts to bridge the gap between Digital Research's manuals and your knowledge of computers.

The CP/M manuals you have will depend upon your version of CP/M and where you got it. Digital Research is constantly upgrading CP/M, so new versions are released from time to time. The most recently released CP/M-80 is version 2.2. Before version 2.2, version 1.4 was commonly distributed. In addition, IBM and others are now distributing a special version of CP/M named CP/M-86 (because it was designed for the 8086 central processor instead of the 8080 or Z80).

If you purchased a CP/M-80 version 2.0 or newer (a higher number), you should have the following manuals:

- *An Introduction to CP/M Features and Facilities*
- *CP/M 2.0 User's Guide*
- *ED: A Context Editor for the CP/M Disk System*
- *CP/M Assembler (ASM)*
- *CP/M Dynamic Debugging Tool (DDT)*
- *CP/M 2.0 Alteration Guide*
- *CP/M 2.0 Interface Guide*

Users of CP/M-80 version 1.4 or earlier will be missing the *CP/M 2.0 User's Guide*, the *CP/M 2.0 Alteration Guide*, and the *CP/M 2.0 Interface Guide* and will instead have an *Alteration Guide* and *Interface Guide* for version 1.4.

Some computer manufacturers, like Microsoft, CompuPro, and Morrow Designs, now reprint and bind these seven manuals into one. Others, like Osborne Computer Corporation and Xerox, do not use Digital Research manuals but instead rely upon manuals they have written specifically for their computers.

CP/M-86 owners receive four manuals: *CP/M-86 Operating System User's Guide, CP/M-86 Operating System Programmer's Guide, CP/M-86 Operating System Guide,* and *The CP/M-86 Operating System Command Summary.* Owners of the IBM Personal Computer receive a specially produced manual set that incorporates the aforementioned manuals.

However, you probably found whatever manuals you received with your copy of CP/M confusing or problematic, and thus you have purchased this book to help sort out the information you need to use CP/M. This book explains the first five of the CP/M-80 2.2 manuals and the *CP/M-86 System Guide* in detail, and it also summarizes the information in the remaining manuals.

The Function of CP/M Within a Microcomputer System

Before going any further, it is important to understand the functions served by CP/M within a microcomputer system. If you know what is going on and why, you are less likely to make mistakes.

We will describe the function of CP/M (or any operating system) within a computer system. This description assumes an elementary understanding of microcomputers and how they function.*

A microcomputer system is illustrated in Figure 1-1. The system illustrated is typical of configurations that you may encounter. It includes the microcomputer, a terminal that combines keyboard and video display, a pair of disk drives, and a printer.

You could make numerous changes to this system. Instead of the single, integrated terminal you could have a separate display and keyboard. The keyboard could be part of the microcomputer with the display as a separate unit, or the keyboard, display, and microcomputer could be packaged together.

Small systems may use cassette tape and cassette tape drives instead of floppy diskettes and floppy disk drives. In the early days microcomputer systems used paper tape to store information and read it back and required a paper tape reader and a paper tape punch. The use of cassettes and paper tape with CP/M is uncommon because disks are faster and more reliable.

Microcomputers spend a lot of time transferring information between the microprocessor and various other components of the system. Microcomputers must also control the operations of these other components. Microcomputers

* If you need more information about microcomputer systems, see *An Introduction to Microcomputers: Volume 0—The Beginner's Book,* 3rd. ed., by Adam Osborne and Dave Bunnell. Berkeley: Osborne/McGraw-Hill, 1982.

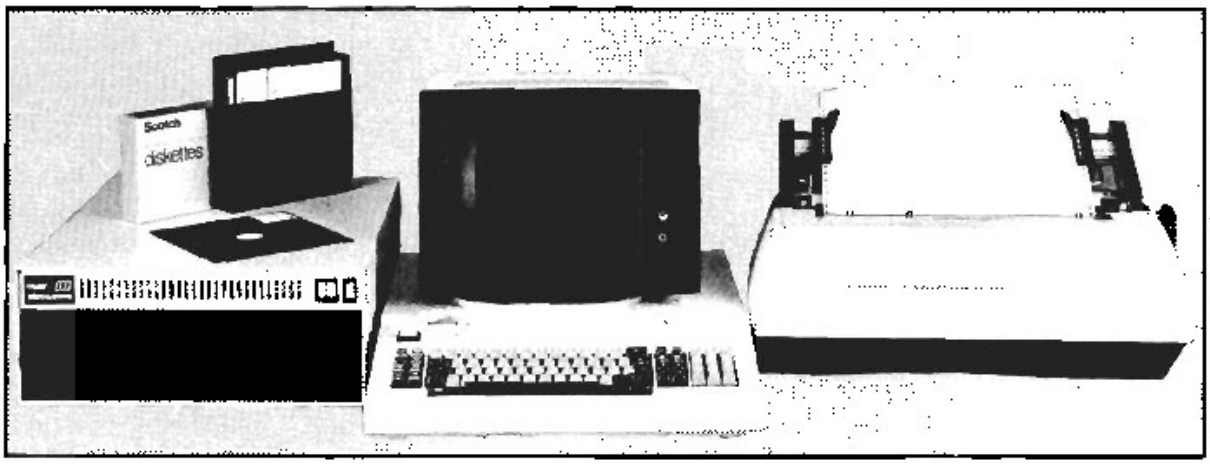


PHOTO BY HARVEY SCHWARTZ

FIGURE 1-1. A complete microcomputer system

perform these operations by executing programs that are referred to collectively as an *operating system.* CP/M is such an operating system. By using appropriate CP/M commands you can transfer data from a diskette to the microcomputer, print data at a printer, or perform any operation which the microcomputer system is physically capable of handling.

In order to perform these microcomputer system functions for a wide variety of different configurations, CP/M (and most other operating systems) ignores the physical units that comprise the microcomputer system and deals instead with logical units. In other words, rather than addressing a printer, the operating system assumes a listing device is present. Likewise, rather than reading directly from a paper tape reader, the operating system assumes the input comes from a reader device.

The manufacturer of your microcomputer system usually ensures that the system's actual physical units connect properly to the logical units CP/M uses. If you put together a system by "mixing and matching" components from several manufacturers, you may have to make some changes to CP/M yourself. These program modifications are invisible if made correctly, but if the proper changes are not made, CP/M may work incorrectly, if at all.

As an operator of a microcomputer system you may occasionally be concerned with physical and logical units. For example, you may have the option of sending output to a printer or a display. Likewise, you may have the option of typing input at a keyboard or receiving input over a telephone line from a remote terminal. You can make such physical unit choices easily by using the appropriate CP/M commands we describe later in this book. For the moment, you only need to understand the general function of CP/M. You do not need to understand the operating system's specific activities in order to use CP/M.

An operating system such as CP/M is itself a computer program which must be executed by a microcomputer. Because it is a program, CP/M must be written in a programming language. The programming language that a microcomputer under-

stands is determined by the microprocessor (sometimes called a central processing unit or CPU) that the microcomputer contains.

A microprocessor is a very small and unassuming device; Figure 1-2 illustrates a microprocessor. The microprocessor is a microcomputer's most important component; it actually translates the instructions which constitute a program and causes the action associated with the instruction. Some microprocessors can execute CP/M, but others cannot.

CP/M was initially written for the 8080A microprocessor manufactured by Intel. Since the 8085 and Z80 microprocessors also execute 8080A instructions, CP/M-80 will run on microcomputers containing either of these two microprocessors. In addition, in early 1981 Digital Research introduced CP/M-86 for the 8086 and 8088 microprocessors.

In order to distinguish between the 8080 and the 8086 versions of CP/M, we use the same nomenclature that Digital Research uses: CP/M-80 refers to the current version of CP/M for 8080, 8085, and Z80 microprocessors; CP/M-86 refers to the current version of CP/M for the 8086 and 8088 microprocessors.

Some Useful Terms

In order to be as precise and clear as possible throughout this book, we present here a short glossary of the more common computer terminology that we will use. Specifically, we want to make sure that you understand the "units" of information we will be talking about. When one talks about language, one uses the terms words,

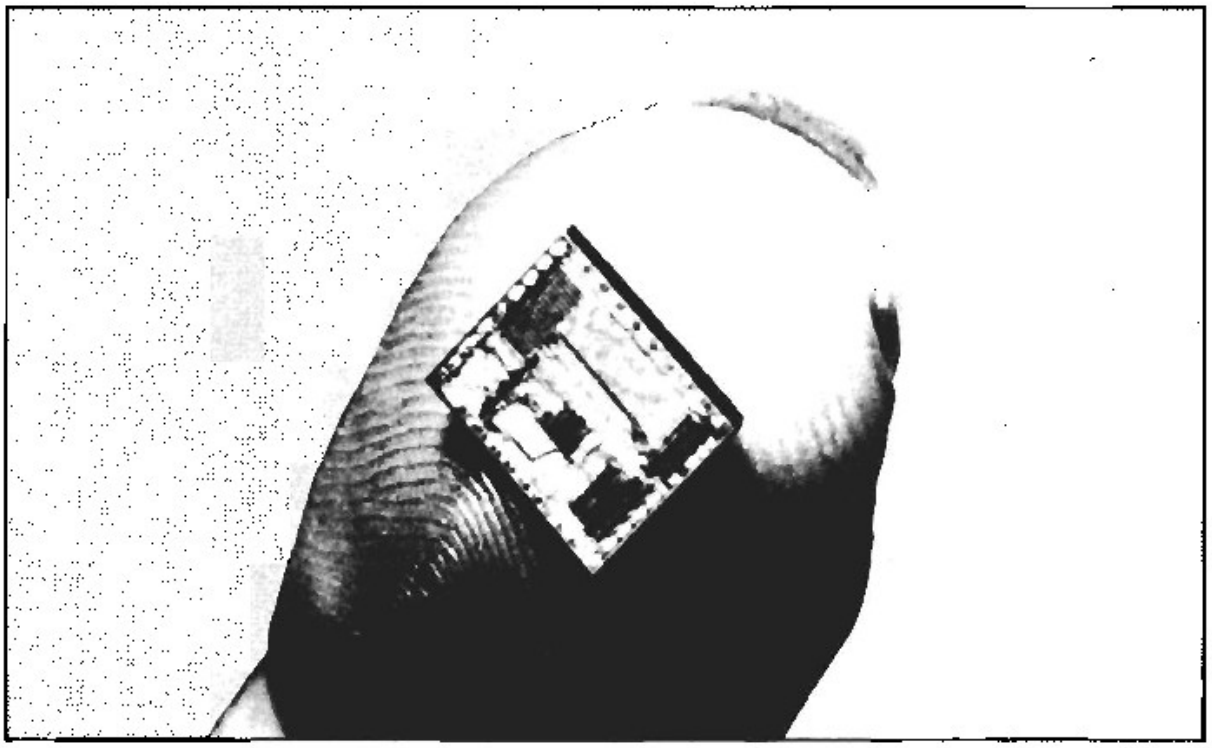


PHOTO COURTESY OF INTEL CORPORATION

FIGURE 1-2. A microprocessor chip

sentences, phrases, and paragraphs. Similarly, when one talks about computers, one talks about bits, bytes, sectors, and tracks.

If you are already familiar with computers and the difference between bytes and bits, you might wish to skip ahead to the next section. Newcomers to computing should study the following information carefully. As with most disciplines, computing has a vocabulary all its own.

BIT

A bit is the smallest piece of information a computer can maintain. All of a computer's data is stored internally as a series of 1's and 0's. A single bit is a single 1 or 0. You will encounter this term later in this book when the technical details of CP/M are discussed.

BYTE

The storage capacity of a microcomputer's memory, or its disks, is always described as some number of bytes. A byte is a memory unit capable of storing a single character. For example, the letter "A" or the digit "1" could be stored in one byte of memory. Numbers without decimal points (termed "integers") are usually stored in two consecutive bytes of memory, while numbers with decimal points (termed "floating point" numbers) are stored in four or more consecutive bytes of memory per number.

Memory size is usually expressed, not as thousands of bytes, but as some number of "K" bytes. 1K equals 1024. All computers are binary machines; in other words, they count in twos. You will get the number 1024 if you double 2 to give 4, then double 4 to give 8, and keep on doubling in this fashion ten times.

A byte consists of eight consecutive bits. Since a bit can only contain a 1 or a 0 value, a little binary arithmetic will soon show you that a byte has 256 possible values. In most microcomputers, each possible byte value is assigned to one symbol, letter, or digit. A "1", for instance, is stored by the computer as a byte value of 49, while an "A" is stored by the computer as a byte value of 65. The "code" which determines which value is assigned to which letter, digit, or symbol is called the ASCII code (ASCII stands for American Standard Code for Information Interchange).

TRACK

When information is stored on a cassette tape, it is stored as a single track of data down the length of the tape. When information is stored on a diskette, the surface of the diskette is considered to be a series of concentric circles, called "tracks." The outermost circle is referred to as track 0, while on standard 8-inch CP/M disk systems, the innermost concentric track is called track 76.

SECTOR

Each concentric track of information on a diskette is further subdivided into units called "sectors." On standard 8-inch CP/M disk systems each sector stores 128 bytes (characters) of information, and there are 26 sectors

on each track. The smallest unit of information that CP/M manipulates on a diskette's surface is one sector. Manipulations of a single byte at a time are done in the computer's memory and not directly on the diskette.

Figure 1-3 illustrates the concept of sectors and tracks.

Types of CP/M

CP/M may be a "quasi-standard" operating system in the microcomputing field at present, but all CP/Ms are not equal. CP/M-80 and CP/M-86 vary with the input and output (I/O) instructions to each machine and may vary in other ways as well.

In addition, computer technology is not standing still. The machine you now own does far more far faster than the vacuum tube computers of the 1950s. The recent introduction of hard and fixed disk drives may become one of the most significant developments. These new disk units can store far more information than the current diskette technology. While drives which use floppy diskettes may store up to 2 megabytes (2,048,000 characters of information), hard or fixed disks may store up to 300 megabytes.

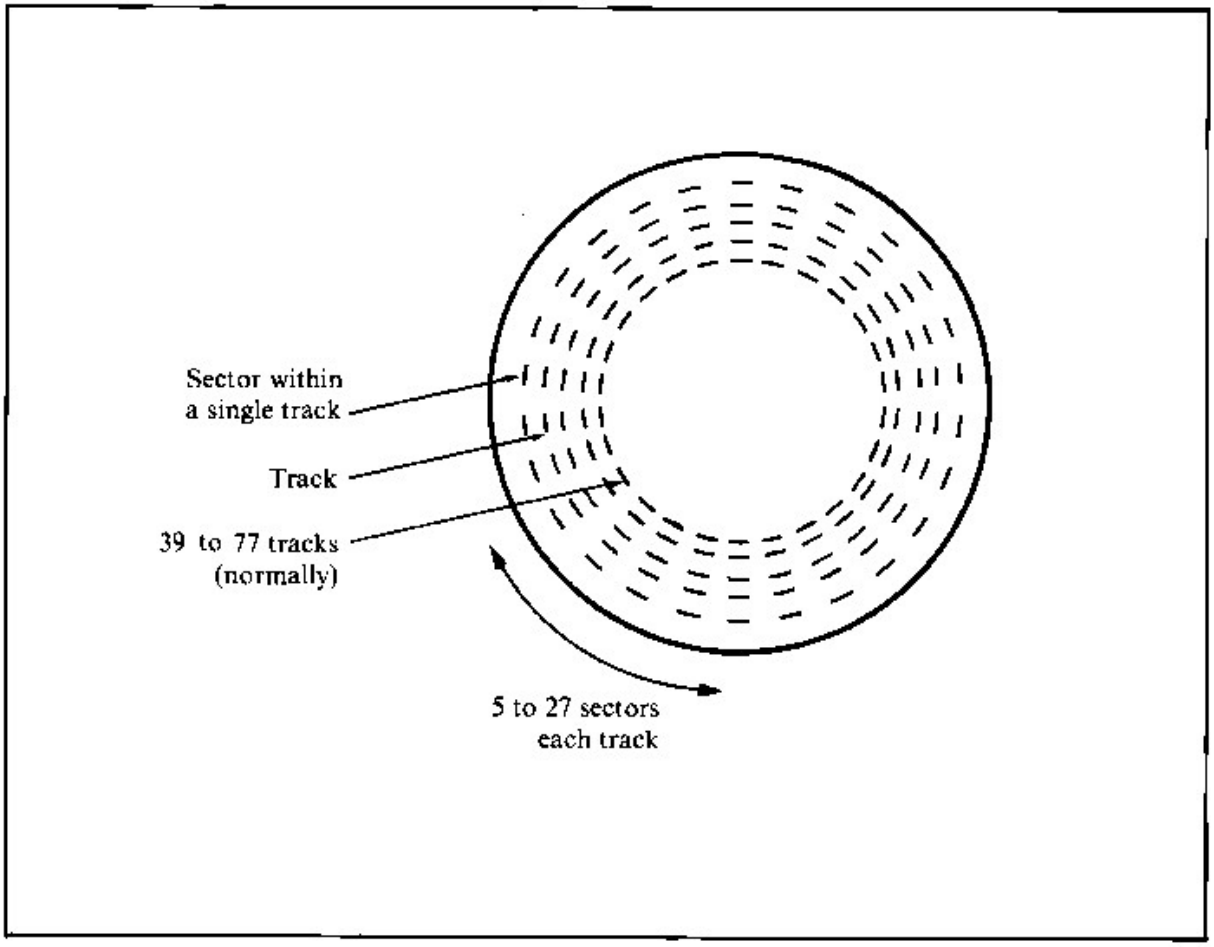


FIGURE 1-3. A diskette's recorded surface

Since CP/M-80 was designed originally for floppy diskette systems, it requires changes to use hard disk units. This is the primary difference between versions 1.4 and 2.2 of CP/M-80 (*see* Appendix C for a complete list of differences between versions).

The different types of CP/M-80 and CP/M-86 also reflect the large number of manufacturers using CP/M. Each may add utility programs or refinements to CP/M to improve performance on a particular machine.

CP/M Compatibility

Unfortunately, not all CP/Ms are compatible, and even the degree of compatibility can vary. The primary ways CP/M varies are

- Version number (1.3, 1.4, 2.0, 2.1, or 2.2)
- Location of CP/M within memory (64K CP/M, 48K CP/M)
- Type of diskettes used (single density, double density, 5¼-inch, 8-inch)
- Logical layout on the diskette (the way information is stored on the diskette, usually differences in the number of sectors and/or tracks used)
- Vendor (normally a combination of the four preceding variations)
- Processor type (CP/M-80 versus CP/M-86).

To list every different type of CP/M would be futile; the list would be out of date by the time you read this book. But despite the possibilities for variation, CP/M still remains more machine-independent than most operating systems. By linking computers using two disparate versions of CP/M, program interchanges or data transfers can usually be accomplished.

CP/M Version Numbers

As stated earlier, there are several versions of CP/M. In general, this book is applicable to CP/M-80 version 2.2, but owners of earlier or subsequent versions will find this book useful as well.

CP/M-80 versions are identified by one number to the left of the decimal point, which refers to the overall version number, and one number to the right of the decimal point, which refers to a revision within a version.

A second number to the right of the decimal point, as in 1.42, identifies subtle or machine-dependent modifications. If your version of CP/M-80 is not 1.4x or 2.2x (where "x" represents any machine-dependent number), ask your supplier for a newer release. Versions of CP/M-80 include

1.3 The original version of CP/M-80

1.4 A more error-free version of CP/M-80 release 1

2.0 The original release version of CP/M-80 release 2

2.1 A field update of version 2.0

2.2 The current revision of CP/M-80 version 2.

If you are using version 1.3 or 2.0 of CP/M-80, you should immediately see about

getting your version updated because significant problems may result from using it.

Version numbers can vary from vendor to vendor. Some CP/M-80 vendors released several different CP/M-80 revisions as version 1.4, while others indicated their own updates by numbers 1.41 and 1.42. In general, only the first two numbers in a version number indicate changes to CP/M-80 by Digital Research.

Digital Research CP/M Products

Since Digital Research wrote CP/M and supports it, we will compare other products with theirs. Currently, Digital Research supports the following CP/M products:

Single Density 8-inch Diskettes.
: This diskette uses a predefined (IBM 3740) format for storing information. CP/M-80 was originally designed for this format. Versions 1.4 and 2.2 are available and differ primarily in the number of disk drives and capacities each can address. As supplied by Digital Research, CP/M-80 comes ready to use on an Intel MDS development system microcomputer; if you own any other microcomputer you will have to make modifications to portions of CP/M-80 in order to get it to work properly.

Double Density 8-inch Diskettes.
: More information can be stored on this diskette than on a single density 8-inch.

> CAUTION: Most double density implementations of CP/M-80 are not directly compatible; a diskette created on one manufacturer's drive usually cannot be inserted and directly read by another manufacturer's drives. As with single density, Digital Research's implementation is based upon IBM specifications; however, unlike single density, very few microcomputer manufacturers support the double density standard.

In addition, Digital Research supports the following operating systems:

MP/M.
: A multiuser form of CP/M. Instead of supporting just one terminal, MP/M is capable of supporting several. See Chapter 6 for more information about MP/M. The current version supplied by Digital Research is known as MP/M II.

CP/NET.
: A multicomputer form of CP/M which allows one computer to use the resources of another (printers, disk drives, and so on). In order to use CP/NET, at least one of the computers must be equipped with MP/M, and all others must use CP/M. Again, see Chapter 6 for more details on this operating system.

CP/M-86.
: A special implementation of CP/M for computers that utilize the Intel 8086 or 8088 microprocessors. In addition to supplying a single density

8-inch version of CP/M-86, Digital Research also supports a version on 5¼-inch diskettes for the IBM Personal Computer system.

Concurrent CP/M-86

Like CP/M-86, Concurrent CP/M-86 is designed to be used with computers which utilize 8088 or 8086 central processing units. The "concurrency" indicated by the title refers to the fact that multiple processes may be performed by a single user on a Concurrent CP/M-86 system. Digital Research supplies Concurrent CP/M-86 for IBM Displaywriter and personal computer systems.

> NOTE: Digital Research provides a ready-to-run CP/M implementation for specific equipment only. For microcomputer systems that Digital Research does not support, special instructions for the terminal, modem, printer, and disk drives are necessary. Only a skilled assembly language programmer should buy CP/M directly from Digital Research. Consultants can be hired to install CP/M if ready-made implementation is not available for your hardware.

Lifeboat Associates CP/M-80 Products

In addition to selling Digital Research CP/M products, Lifeboat Associates has altered the part of CP/M-80 that contains instructions to the disk drives and other devices (the BIOS section, described in Chapter 7). Lifeboat Associates' CP/M-80 products are available for the Radio Shack TRS-80, Micropolis, North Star, Polymorphic, Altair, Zenith/Heathkit, and a number of other systems.

While written programs using Lifeboat Associates CP/M-80 products are almost universally compatible with other CP/M-80s, the physical program storage media are not. Thus, the end result may disappoint you. Since the physical media, the diskettes, may differ from one machine to the next, copying programs or data from one microcomputer to another can be difficult. On the other hand, if both systems have modems, you could send programs via phone lines to a different system.

The importance of having equipment capable of reading and writing information in the IBM 3740 8-inch single density diskette format cannot be underestimated. At present, this is the only *physical* means of interchanging information between computers. Short of having 8-inch single density disk drives, your computer system should have some means of communicating with other equipment. Normally a "modem port," or perhaps merely an extra "serial port" will enable you to link your computer to another.

Because Lifeboat Associates is only a distributor of CP/M-80, their products are released after Digital Research updates. For instance, Lifeboat Associates released products for disk drives other than standard 8-inch floppies one year after Digital Research first introduced CP/M-80 version 2.2.

Furthermore, all diskette systems do not universally use all CP/M-80 utility programs. For example, formatting a diskette—the process of labeling or identifying the sections of a previously unused diskette so that the operating system recognizes where data lies—is not provided for under Lifeboat's North Star single density CP/M-80.

Manufacturers' CP/M Products

Many microcomputer manufacturers provide CP/M-80 or CP/M-86 as a standard operating system on the computers they sell. Such manufacturers include Altos, CompuPro, Digital Microsystems, Dynabyte, Exidy, IMS International, Micromation, Morrow Designs, North Star, Onyx, Osborne, and Vector Graphic.

Manufacturers' CP/M-80 products differ slightly from Lifeboat Associates' CP/M-80 products—manufacturers all reconfigure portions of CP/M for their own equipment. Some, however, add subtle traps for the unwary. These traps are often labeled "features." For example, Vector Graphic CP/M-80 version 2.2 includes some complicated checking for the computer model in use. They also added a routine to check for the depression of certain keys. In the Vector Graphic product, these additional routines make some otherwise standard CP/M-80-compatible programs incompatible with their implementation of CP/M-80.

At least one manufacturer has also made modifications to the internal structure of CP/M-80 to make it work faster. While this is certainly a benefit, it reduces its compatibility with systems that use a standard version of CP/M-80.

Yet other manufacturers, like TEI, have produced operating systems for their own equipment which work similarly to CP/M. Again, there may be significant advantages to using such operating systems, but they are simply not CP/M, and users of these "look-alikes" should be aware of this.

With the advent of CP/M-86, some manufacturers, notably CompuPro and IBM, have begun providing CP/M-86 as an operating system with their computers. While data on CP/M-86 storage diskettes may be compatible with data on CP/M-80 storage diskettes, programs are not compatible between the two versions, at least not without some conversion.

Yet another type of hybrid of the Digital Research CP/M operating system appeared in early 1982. CompuPro and G & G Engineering introduced a CP/M-MP/M version of the operating system —one that is able to run programs created under both of the original versions (CP/M-80 or MP/M II). Digital Equipment Corporation (DEC) went one step further with the introduction of their Rainbow 100 system, which features both Z80 and 8088 central processing units. The DEC CP/M can execute programs and commands for both CP/M-80 and CP/M-86.

Needless to say, if you receive any non-standard CP/M operating system, you should read the manual that accompanies it to see how it differs from the versions of CP/M described here.

CP/M Look-alikes

In addition to computer manufacturers who have produced CP/M look-alikes, a number of software firms have also done the same. Like CP/M, most of these look-alike products can be used on a wide range of microcomputers. Some, however, are more restrictive than CP/M-80 and require that the microprocessor be a Z80. TP/M from Computer Design Labs is one such product.

Other popular CP/M-80 look-alikes are SDOS from SD Systems, developed originally for the SD computer and then marketed separately; I/OS (also offered under the name TSA/OS) from InfoSoft; and TurboDos, a special look-alike which emphasizes speed of memory-to-disk transfers. In addition, CDOS, the Cromemco Disk Operating System, claims compatibility with CP/M-80 version 1.3. The MULTI/OS, also from InfoSoft, is an MP/M look-alike.

For the most part, these look-alikes will run standard CP/M-80 software. For those of you who might have purchased a CP/M-80 substitute, Chapter 6 is devoted to these look-alikes.

There is now even a CP/M-86 "work-alike," known variously as 86-DOS, Microsoft DOS, and IBM DOS. Originally designed to allow Seattle Computer Products' system users to quickly convert existing CP/M-80 programs to the 8086 environment, 86-DOS is similar to CP/M-86. 86-DOS is briefly covered in Chapter 6.

Diskettes

Throughout this book we will be referring to "floppy diskettes" and "disk drives." A floppy diskette is the flimsy removable media used by a disk drive, which is the mechanical portion. Disk drives which utilize polished metal "platters" instead of removable floppy diskettes will be referred to in this book as "hard disk drives" (because the media is hard). For the sake of conciseness, we will refer to floppy diskettes as "diskettes," disk drives as "drives," and hard disk drives as "hard disks."

The backbone of any disk operating system is, of course, the diskette itself. It may be difficult to comprehend that 150 single-spaced typed pages (the equivalent of 30 hours of typing) can be stored on one flimsy magnetic diskette. Unfortunately, we can easily forget to take the proper steps to protect the information stored on the diskette. Before discussing detailed information about CP/M, we will pause for a quick course on types, care, and usage of diskettes.

Comparing Diskettes

Walk into a computer store, ask for a diskette, and the salesperson will ask you what kind you need. For an idea of the possibilities, consider the following features:

- 8-inch versus 5¼-inch diskettes
- Single-sided versus double-sided diskettes
- Single-density versus double-density diskettes
- Soft-sectored versus hard-sectored diskettes
- 10-sector versus 16-sector hard-sectored diskettes
- Write-protect notch versus no write-protect notch.

This is a confusing array of choices, and there are as many brands as there are types. We could not possibly list all of the combinations. In fact, diskette manufacturers publish long lists of compatible diskettes, computers, and disk drives. Check with any reputable computer store to learn which diskettes to use with your microcomputer. Better still, check with the vendor who sold you the computer.

A brief summary of the more popular types of diskettes and microcomputers is provided in Appendix E.

Describing Diskettes

Diskettes, as we mentioned previously, are flimsy, and that is why they are sometimes called floppies. If you have an extra diskette around, go get it right now, as we are about to take a "diskette tour."

You notice first that the diskette is accompanied by a heavy paper envelope. This envelope protects about two-thirds of the diskette from such data killers as dirt, liquids, and thumbprints. Since it is a very thin envelope, it provides limited protection—so be careful. Many diskette manufacturers print handling tips on the back of this envelope. Read any information printed on your diskette; someday it may mean the difference between retyping for hours or spending a relaxing evening at home.

Carefully pull the diskette out of the envelope (it slides right out). If it looks like you must cut something open to get inside, you are mistaking the diskette "sleeve" for the envelope. The diskette has a square vinyl sleeve which protects a thin circular disk. The sleeve has a circular hole in the center (as does the disk surface inside), and there is an oblong cutout at one edge of the sleeve (*see* Figure 1-4). Also, there is a smaller hole just to one side of the central hole.

These parts are identified as follows:

Centering Hole.
: The disk drive mechanism locks onto this hole to spin the diskette inside the sleeve.

Indexing Hole.
: The disk drive looks here to find the starting sector (and in the case of

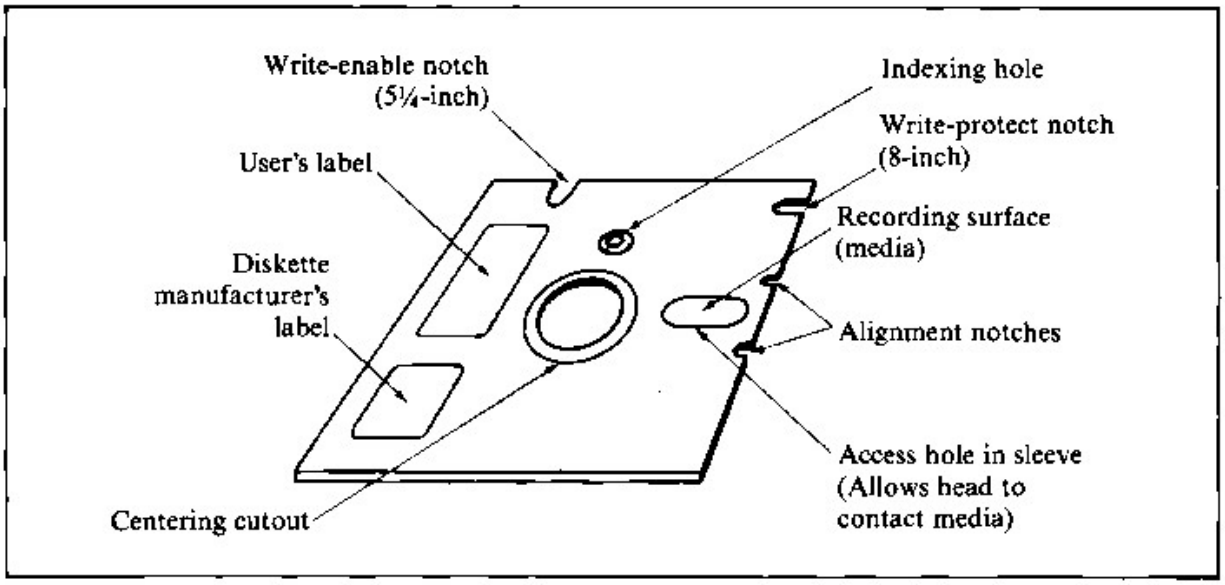


FIGURE 1-4. A typical diskette

hard-sectored diskettes, each individual sector) for each track on the diskette. Imagine a line drawn across the diskette surface at this point; the drive waits for this starting line, then counts characters of information from there.

Access Hole.

The head of the disk drive comes in contact with the magnetic surface through this cutout. The head moves back and forth in this opening, from track to track. Note that there are access holes on each side of the diskette.

Notch.

This is a write-protect notch. Writing on the diskette means adding information to the diskette. A point of confusion arises here: on 8-inch diskettes, if the write-protect notch is covered up, you can write on the diskette, and if the notch is left uncovered, you cannot write on the diskette. On 5¼-inch diskettes the opposite is true. If the notch is uncovered on the smaller diskettes you can write on the diskette; if the notch is covered, the diskette is protected against writing. (You might want to circle the appropriate section of this paragraph for future reference.)

To compound the problem further, the 8-inch diskette write-protect notch is located near the access hole while the 5¼-inch diskettes have their write-protect notch on the side of the diskette (*see* Figure 1-4).

As if this were not enough, some manufacturers buy diskettes which have no write-protect notch at all. Software vendors who sell programs on 5¼-inch media usually buy such diskettes so that you can never make any changes to the original diskette you receive.

Handling Diskettes

Now that you are familiar with the parts of the diskette, it is time to learn how to handle it.

First, the don'ts.

Never touch the diskette surface. You may handle the vinyl sleeve when necessary, but do not touch the actual magnetic surface of the diskette. No matter how clean your hands are, even a slight residue might prohibit your computer from reading some data.

Keep diskettes away from magnets. Silly instruction, you say? You think there are no magnets where you work? Think again. If your computer has a video terminal (one with a display tube), you have a magnet. Strong electrical fields can also act as magnets. The speakers in any sound system almost certainly contain magnets. Actually, the magnetic field most appliances generate is not enough to erase the information on your diskettes except through direct contact. Even so, it is a good idea to keep your diskettes at least a foot away from anything magnetic. Also, contrary to popular belief, it is best to store magnetic media in a metal box, not a plastic one. Plastic does not intercept stray magnetic fields, while metal does.

Do not bend your diskettes. The information on a diskette is packed into a very

small area. Think about it for a moment: 2,000,000 characters (on some diskettes) in an area of two surfaces eight inches in diameter. Any crease in the diskette can make the read head of the disk drive lose contact with the disk surface, and a lot of information could be lost in that crease.

Do not keep diskettes in a dirty environment. If you allow computer users to eat, drink, or smoke while they use the computer, you are asking for trouble. Film-makers routinely use cola to erase noises from magnetic sound tracks. If you should accidentally spill a cola drink on your diskette, you would probably destroy all its information.

Never leave a diskette in the computer when you power down. Remove the diskettes before you turn off the computer and/or the disk drives. Likewise, turn the power on before inserting the diskettes. Most of the time, if you do not follow this advice, nothing will happen. However, a "spike" of power could reach the magnetic head of the disk drive and write spurious information. This can prove disastrous.

Do not let your diskettes get too full. Many programs use a diskette to temporarily store data or to generate data. If this temporary or new data does not fit, you could lose it.

Two factors limit diskette capacity: the number of files and the number of characters. In some implementations CP/M allows as few as 32 files, while some versions of CP/M-80 version 2.2 allow as many as 8192 files to be present on a hard disk. Most versions of CP/M-80 will allow 64 or 128 files to be stored on a floppy diskette and 256, 512, or 1024 files to be stored on a hard disk.

The number of characters on a diskette is determined by the drive and ranges between 80,000 and two million characters for floppy diskettes. Hard disk drives can generally store a minimum of five million characters and range in size to 30 or more megabytes of storage.

Now for some do's.

Always insert diskettes slowly and carefully. Many of the 5¼-inch disk drives have a very small tolerance for alignment between the diskette and the magnetic head which reads and writes information. In particular, Micropolis drives can be made to miscenter a hastily entered diskette because the drives actually move the diskette before it it positioned for reading. Diskettes are fragile; they should be handled slowly and deliberately. Saving one second by rushing to get the diskette into the drive may waste hours when you must reenter the lost information.

Label all your diskettes. Nothing is more frustrating than having fifty identical diskettes and not knowing their contents. Diskettes, especially if you follow the backup recommendations given later in this chapter, have a tendency to multiply like rabbits. Consider this book. The text barely fits onto one high-capacity diskette. However, copies of the last three revisions, plus backups, are stored. In addition, the text editor program occupies most of another diskette. Add the programming diskettes, another set of diskettes to keep important records and data, plus a few for games and recreational purposes, and you can imagine the stack of paper envelopes you might accumulate.

So take heed. Label your diskettes. Develop procedures to distinguish older

versions or copies of your data from the current (or "use") ones. The label should contain your name and the date, the name and version number of the operating system, and some description of the diskette's contents.

Keep backup and rarely used diskettes away from the computer. Limit those diskettes you keep at the computer to those you constantly use. Since you will generate a number of diskettes, why complicate locating the diskette you need?

Make sure your diskettes are stored correctly. Just like phonograph records, diskettes will be damaged if stored for long periods of time in other than a horizontal or perfectly vertical manner. If diskettes are allowed to slant diagonally, gravity will in time subtly bend the diskette. While diskettes do not warp like records, when that bend gets to a certain point it will crease the diskette. If you store diskettes horizontally, do not pile large numbers of diskettes one on top of another. Diskette manufacturers recommend that you stack diskettes no more than ten deep.

Maintain your equipment. Disk drives are not as difficult to maintain as cars; they do not require oil changes every 10,000 miles. Actually, diskette equipment can be overmaintained. Constant adjustment of the mechanism may erode the tolerances built into the drives. Constant cleaning of the magnetic heads may be more abusive to the heads than normal wear. Use a cleaning diskette (available from diskette suppliers) once every three months or whenever you suspect disk drive errors. The latest disk drives need maintenance only once a year, and in the case of the newer Winchester technology hard disk drives, there is nothing to adjust—the unit is a sealed mechanism.

Buy quality diskettes. Mail-order bargains for diskettes are priced below what most dealers pay. At that price the diskettes have not been verified or checked for their ability to store and retrieve data. You can hear the difference in quality between diskettes; a poor quality diskette will make raspy rubbing noises. You do not keep your personal and business records on napkins and paper towels, so do not entrust your valuable information to their computer equivalents.

Inserting and Removing Diskettes

Consult the manual of your microcomputer to learn how the diskette goes into the drive. For horizontally mounted drives, hold the diskette (remember not to touch the magnetic surface) so it is also on a horizontal plane. The label side should be up. Insert the end with the head access hole (the oblong one) into the drive first (*see* Figure 1-5). The writing on the label usually faces away from you as you insert the diskette. Examples of computers for which the above process is correct are the Apple, IBM, and Osborne computers.

There are exceptions even to this simple process. Altos, for instance, mounts their disk drives upside down in the chassis for ease of maintenance. The label side of the diskette therefore faces down when you insert a diskette into an Altos system.

Vertically mounted drives are a bit more difficult to describe because they may be mounted so that the label either goes in on the left-hand or right-hand side as you insert a diskette. If you do insert a diskette upside down, there is a possibility that you may cause damage to the diskette; if you are in doubt about the process of

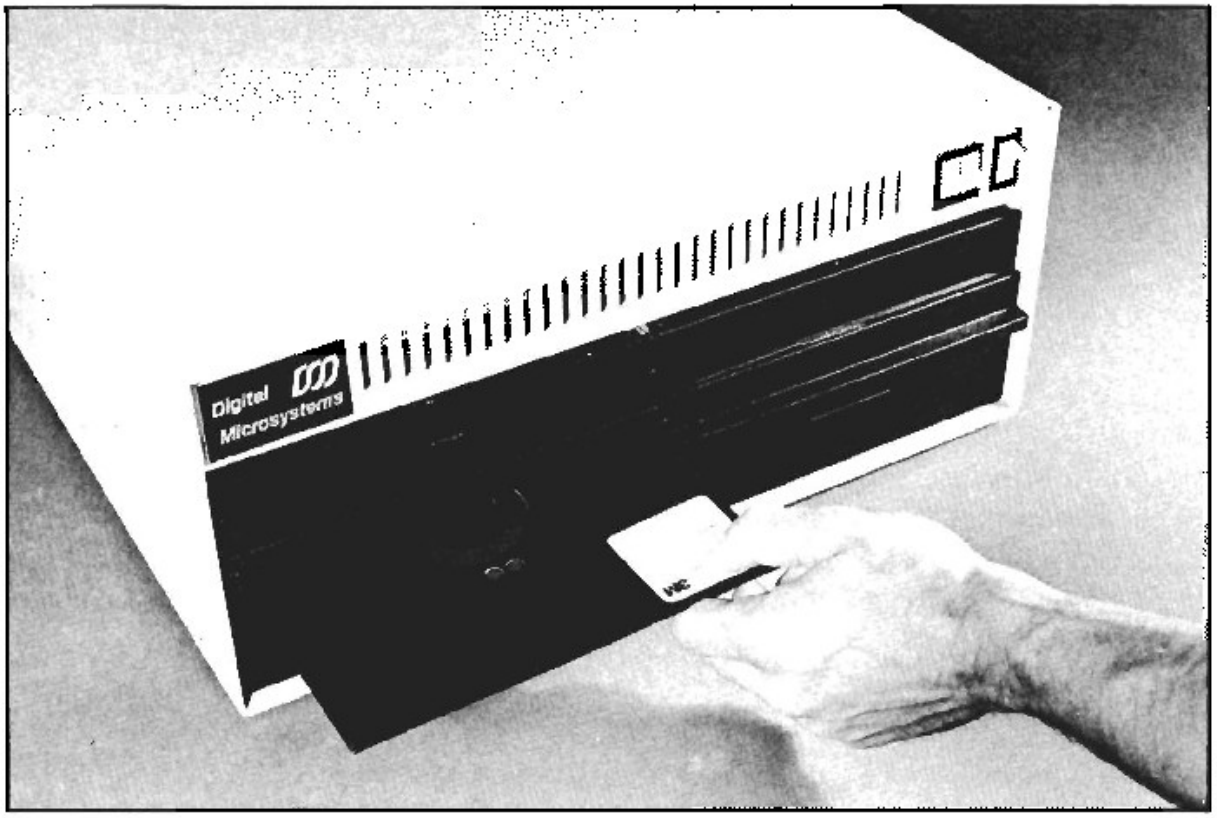


PHOTO BY HARVEY SCHWARTZ

FIGURE 1-5. Inserting a diskette in a horizontally mounted drive

inserting a diskette, make sure to consult the manual for your microcomputer.

Almost all disk drives tell the user when the drive is using a diskette (a process oftentimes called "accessing" a diskette). A small red light, the disk activity light, comes on each time a diskette is accessed. Do not put a diskette into or take a diskette from a drive when the disk activity light is on unless the manual for your computer says you can do so. On many systems, taking a diskette out while the activity light is on may cause information to be written randomly across the diskette surface. You stand more of a chance of damaging the integrity of the data on a diskette if you remove it from the drive with the disk activity light on when the computer is trying to write information onto the diskette (as opposed to merely reading from it).

Rational handling of diskettes is applied common sense. Although diskettes are new to the computer novice, there is no excuse for ignoring the implications of misuse. CP/M can only be as good as the environment you create for it. Properly maintain the equipment and carefully handle your data on the diskettes.

Starting Up CP/M

When any version of CP/M is loaded into your computer, several things happen. A cold start loader moves into your computer memory from the diskette. This loader varies from machine to machine and may differ between CP/M versions, but

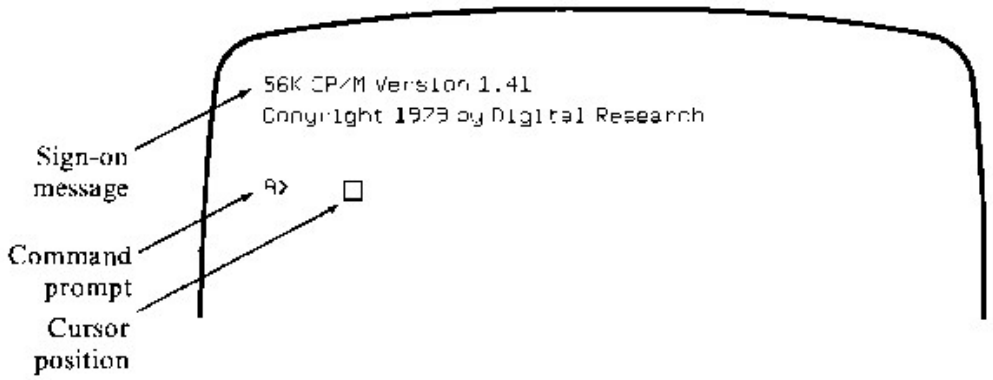
its function is always to load CP/M. The cold start loader is a distinct program stored with CP/M on the diskette.

Why load something into your computer in order to load something else? The reasons are complex (and are discussed in Chapter 7), but primarily it facilitates machine independence, which means that completely different computers can use the same disk drives and the same copy of CP/M.

After the cold start loader brings in CP/M, a number of things happen. In order, they are

- CP/M is loaded into your computer's memory
- Control of the computer is passed to CP/M
- CP/M performs various initialization operations
- CP/M places a "sign-on" message on your screen
- The prompt A> appears on your screen
- Finally, CP/M waits for you to type a command.

At this point, your display looks something like the following:



As you can see, CP/M does nothing particularly mysterious when you start your computer. The whirring and clacking you hear from the computer's disk drives indicate everything is working properly, and the information needed to get CP/M running is being transferred from diskette into memory.

Next, consider the physical process you undertake to initialize CP/M. Since every computer system is different, we will talk in general terms. If you are not sure of the correct procedure, read the manuals that accompanied your computer.

Some peripheral devices must first be turned on. If you have a hard disk drive, turn it on first, as some hard disk drives take several minutes to "get up to speed" before they can be used.

If you have floppy disk drives, do not insert any diskette yet (unless your computer manuals tell you to do so). If you insert the diskettes and then turn the computer on or off, a small transient voltage may be applied to the disk drive head and accidentally erase or change information on the diskette.

Turn the computer on. In some cases, the computer immediately attempts to load programs from the diskette when you turn the power on. Do not insert a diskette into a drive while the drive is attempting to access a diskette.

Most manufacturers use a special routine to get things started. When you "power on" such machines, they display a message on the screen and wait for your instruction. Some wait for you to press the RETURN key several times to determine the speed at which the keyboard sends characters. Usually the instruction is a single letter (like "B," which stands for "boot"). On these systems, you turn on the power, place your diskette in the first drive, and press the "B" key or the instruction your system requires.

Here is a summary of the start-up steps we have discussed.

1. Switch on peripherals' power
2. Insert diskette (some systems)
3. Power on computer
4. Insert diskette (remaining systems)
5. Press the CARRIAGE RETURN key (some systems)
6. Type start instruction (some systems—usually "B")
7. Wait for CP/M to "sign on."

Many newer computer systems automatically look for a CP/M diskette when first turned on. For these systems, the normal start-up process is abbreviated as follows:

1. Switch on peripherals' power
2. Insert diskette
3. Turn on computer
4. Wait for CP/M to "sign on."

See your computer manual for more specific start-up procedures.

Day-to-Day Routines

There are some day-to-day operating routines you need to know.

RESET the Computer: Pressing the Panic Button

There are a number of "fatal" errors that can endanger the data on your diskettes. Fatal errors (errors from which there is no recovery) are guaranteed if you turn off the power or press the RESET button while the disk drives are writing information on the diskette. Unfortunately, there is no steadfast rule that says you should avoid turning off the power or pressing RESET. You may resort to pressing the RESET button to stop an "endless loop" of continuously executed instructions. Thoroughly tested programs rarely get stuck in such a loop, but newly developed programs can, and often do.

If one of these endless loops includes an instruction to write information onto the diskette, the diskette activity light could come on and stay on. After a few minutes of

waiting for something to happen, you may panic. Be certain the computer is in an endless loop before proceeding. Good computer programs will give periodic messages like

I'M WORKING...

or

THIS MAY TAKE A WHILE...

While these messages are not exactly foolproof, they are somewhat reassuring when the diskette activity light comes on and stays on.

If you are entirely convinced that your computer is in an endless loop, press the RESET button on the computer. Almost all computers have one, but some of them are hidden on the back. Pressing the RESET button on most computers has the same effect as turning the power off and then on. However, the power is not interrupted to any component. Some computers even retain the program or data that was in memory at the time RESET was pressed, but this is not usually true.

If your manuals are not clear about what to do after pressing RESET, call your computer store before doing anything else. If important data still remains in the computer and you want to preserve it, reloading CP/M-80 or another program will probably destroy it. Reloading CP/M-86 may or may not destroy the program, depending upon your computer. Even so, it is best to assume the worst and not automatically assume that you can recover information in memory. Instead, be pleasantly surprised if you manage to recover information from memory after pressing RESET. The chance of doing so may be slim, but remember, it is the only chance you will get. Do not do anything else after pressing RESET until you are sure that the information in memory is not recoverable.

Backup the Diskettes

Almost every implementation of CP/M has a copy program (*see* Chapter 5). Before ending a session at the computer, make copies of any diskettes you have changed or updated. This process is called "backup."

Here are our suggestions on backing up diskettes.

1. Label all diskettes. Distinguish the original diskette from the copy. One way is the father-son method. The original diskette is labeled "father," and the copy is labeled "son." If you maintain an extra copy of the diskette for archival purposes, this is the "grandfather" (original), and the "father" and "son" diskettes are subsequent copies. This may not be the best way to maintain copies, because it is far too easy to slip into the habit of making infrequent backups.

   A far better method is the rotating backup procedure. You have one data diskette for each day you use the computer (for example, Monday through Friday). At the end of Monday's processing, you copy the Monday diskette onto the Tuesday diskette. At the end of Tuesday's processing, you copy the information onto Wednesday's diskette. Using this method (assuming

that you labeled each diskette carefully), you know immediately if you are using the right diskette, and there is a natural backup sequence.

2. Diskettes are not permanent. No diskette will last forever. The reason for this is simple: there is a great deal of physical contact between the diskette surface and the inside of its sleeve. Also, a poorly aligned drive will create excessive head wear.

   Estimates of the life of a diskette have an extraordinary range. One manufacturer claims its diskettes will endure a year of continuous use. In any case, do not expect diskettes to last forever. If you use the rotating backup procedure, retire diskettes sometime after six or twelve months of use. This is a conservative practice since each diskette has been used only 27 to 54 days. But considering the price of a diskette against the cost of reentering data, it seems a wise practice.

3. Never use programs on their original diskettes. When you receive a new program (or CP/M update) on diskette, make a copy of it, label the original "master," and safely store this master diskette. Using an original diskette is foolhardy. Someday your computer may sit idle while you wait for a replacement diskette. Most software vendors provide an update only on return of the original diskette, so be sure to store it in a safe place.

Shutting Off the System

Earlier you learned how to turn on your computer system. The procedure for shutting off the computer system is not the reverse of the power on procedure.

First, use a normal exit from any program you might be running. This step is extremely important because many programs do not correctly finish writing information to the diskette until you tell the program that you are done.

Second, remove any diskettes from their drives. Then turn the power off on the disk drives, terminal, printer, and any other peripherals. Turn power off at the microcomputer last. If you have hard disk drives on your system, turn off the power to those drives before turning off the microcomputer's power. Failure to do so may result in loss of information on the hard disk. If your manual tells you to do the opposite, of course you should follow its instructions.

CHAPTER

2 CP/M Built-in Commands

We described how to start up CP/M in Chapter 1. Now it is time to make CP/M work for you. This chapter investigates the built-in commands in CP/M-80 and CP/M-86.

COMMANDS ARE INSTRUCTIONS

Commands are instructions to CP/M. When CP/M is ready to receive a command, it displays its prompt (usually A>). To get CP/M to perform a command, type the command and then press the CARRIAGE RETURN key.

All versions of CP/M have two types of commands: *built-in* commands and so-called *transient* commands.

The distinction is a subtle one. The short programs which carry out built-in commands are always present in memory with CP/M. The programs which carry out transient commands are not automatically loaded into memory when CP/M is started up. A transient command causes CP/M to get a program from the diskette, load it into computer memory, and execute the program. The program, called a *transient program,* would not otherwise be present in memory.

To make things a little clearer, we will distinguish between built-in commands and transient commands as follows:

- A *built-in command* is immediately executed by CP/M without consulting further instructions on diskette.
- A *transient command* requires a set of instructions stored on diskette to be brought into memory before each use. It is important to note that CP/M executes all commands. Both built-in and transient command programs are invoked by typing a command such as LOAD or DIR in response to the CP/M prompt.

Commands Operate on Disk Files

CP/M commands access and manipulate information stored on diskettes in files. A *file* is any information stored as a single entity with a unique name. The length of a file may vary from no characters to the maximum capacity of the diskette. The following are some examples of file length:

- A single program
- All of the data used by a program
- An entire mailing list
- A single list
- A large group of standard form letters
- A chapter of this book
- The entire contents of this book.

No rule determines the information you may store in a file. You define a file's contents when you create the file.

All files consist of *fields:* single words, numbers, or any other convenient small unit of information. You can also divide files into *records* to represent another division of information larger than fields.

How files are divided into records and fields again depends on how you create the file. To illustrate records and fields, consider a mailing list file. All the names and addresses may constitute a single file, but each name and address could be designated a single record, while the name and each line of the address might each become a single field.

```
   FILE: [============NAME AND ADDRESS FILE============ ]
RECORDS: [--name/address/city--] [--name/address/city--]
 FIELDS: [name] [address] [city] [name] [address] [city]
```

BUILT-IN COMMANDS SUMMARY

Only six built-in commands are recognized by CP/M-80 version 1.4 and seven are recognized by CP/M-80 version 2.2. CP/M-86 also recognizes six built-in commands.

CP/M-80 Version 1.4	CP/M-80 Version 2.2	CP/M-86
DIR	DIR	DIR
TYPE	TYPE	TYPE
ERA	ERA	ERA
REN	REN	REN
SAVE	SAVE	
d:	d:	d:
	USER	USER

In addition to the basic built-in commands there are several immediately interpreted line editing commands. They are

^C ^E ^H ^J ^M ^P ^R ^S ^U ^X

We will describe each of these commands in detail later in this chapter, but as you can see, you need to learn only a few built-in commands.

Entering the Commands

Before we tell you what each command does, you need to know the conventions we use to denote commands and keystrokes.

Press means to press a single key

Type means to press a sequence of keys

<*cr*> means to press CARRIAGE RETURN (or ENTER on some machines)

^*X* means to enter the control character X

Underlines are used to distinguish user input from computer output (when necessary); anything you type is underlined in the examples we provide.

Note that the CARRIAGE RETURN key will be represented by the symbol <cr>. A typical example of command entry would be

```
DIR <cr>
```

This means type the letters "D," "I," and "R," and then press the CARRIAGE RETURN key. The CARRIAGE RETURN key can be labeled RETURN, CARR RET, CR, ENTER, or other abbreviations.

Control characters are subject to special interpretation by a computer. Just as you hold down the SHIFT key to type a capital letter on a typewriter, you hold down the CONTROL key to type a CONTROL letter (or CONTROL character, in computer jargon). This is illustrated in Figure 2-1. Therefore, ^C means press and hold down the CONTROL key, type a letter "C," then release the CONTROL key.

The CONTROL key is often abbreviated as CTRL or CTL or sometimes ALT; it is almost always found in the lower left-hand corner of the keyboard near the left-hand SHIFT key.

Your console displays both what you type and the computer's response. This book uses the following convention to differentiate between output from the computer and data typed at the keyboard: operator input is underlined, while all computer-generated characters are not underlined.

Here is an example:

```
A>DIR B:<cr>
A:STAT  COM| MBASIC4 COM| BACKUP  COM| MBASICS  COM
A:CBAS2 COM|  CRUN2  COM|  XREF   COM|   PIP    COM
A>MBASICS <cr>
```

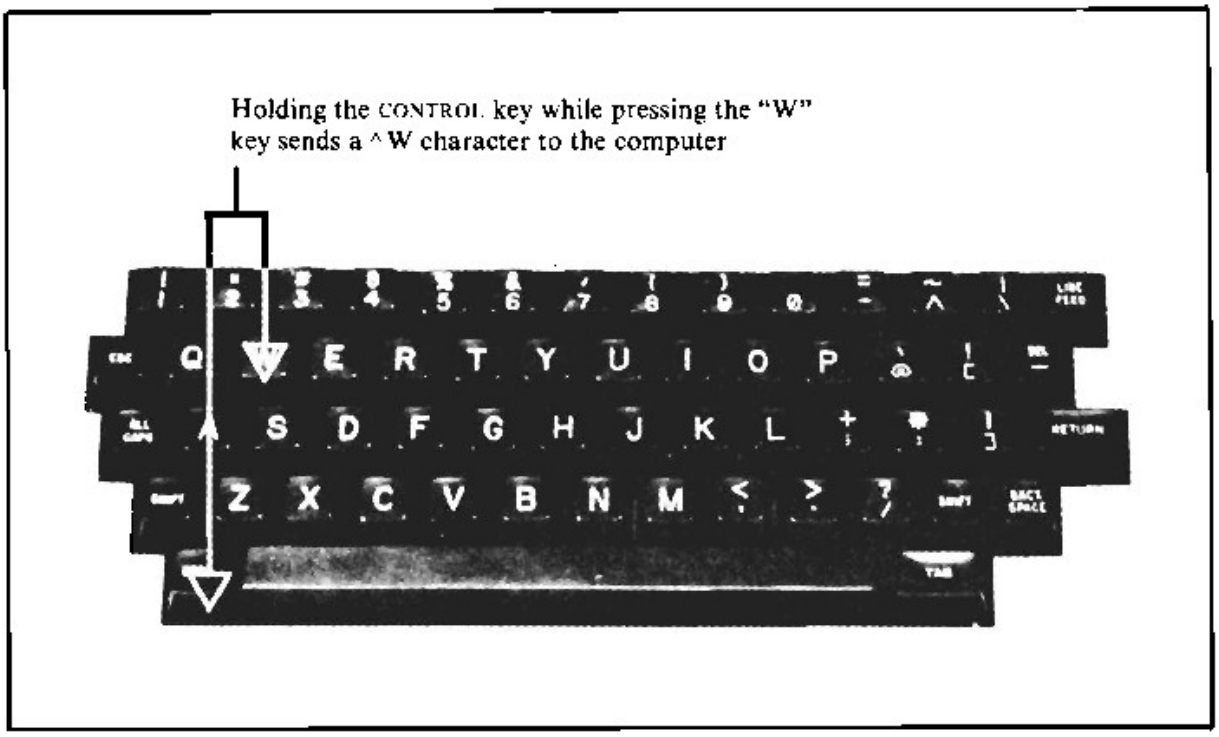


FIGURE 2-1. Typing a CONTROL character

```
Microsoft BASIC version 5.3
Copyright 1977, 1978, 1979, 1980
by Microsoft, Inc.
14,568 bytes free
Ok
```

In this example, the computer places the prompt, A>, on the screen (or printer). You type DIR <cr>. The computer replies with a listing of eight file names and another A>. Then you type MBASIC5 <cr>, and a five-line message indicates Microsoft BASIC has been loaded and is ready to use.

The space character, which is typed by pressing the SPACE BAR, is frequently required in commands. Even though it appears to be "nothing" or "blank space" when typed on a typewriter, within a computer system the space character is just as important as any other character. In fact, early versions of CP/M-80 are quite finicky about spaces, although this is less true of version 2.2. Generally, when the examples in this book show a space, you should press the SPACE BAR, and where no spaces are shown, do not press the SPACE BAR.

Upper-case and lower-case letters are used in command line and other user-input examples to differentiate the fixed and variable portions of the input. You must type the upper-case portion exactly as it appears; this unvarying part of your input is usually the command name. In place of the lower-case portions, you must substitute information of your choosing; this is the variable part of your input. For example, one form of the directory command is

```
DIR *.typ <cr>
```

You must type the "DIR" portion of the above command, but you must replace the "typ" portion with a file type of your choosing when you type the command. Here are some specific examples that would be acceptable forms of the above command.

```
DIR *.ASM <cr>
DIR *.BAS <cr>
DIR *.COM <cr>
```

CP/M FILE CONVENTIONS

Each CP/M-80 or CP/M-86 file is identified by a *file name* and a *file type*. These identifiers are discussed in detail below.

File Names

Each CP/M-80 or CP/M-86 file has a unique name consisting of one to eight characters. Normally, only upper-case letters are used for file names, although it is possible—using Microsoft BASIC, for instance—to create a file with a lower-case name, or even with no name at all.

Both CP/M-80 and CP/M-86 automatically convert characters you type in a command line (following the A>) from lower-case to upper-case and will not accept a blank file name in a command line. Several characters may not be used in a file name. These characters are

< > . , ; : = ? * [ ] ( ) / or <TAB>

Since CP/M uses each of the above characters in special ways, they are invalid as file name characters.

Nor may you use CONTROL characters (or any other characters that do not appear on your screen) in a file name. Here are some samples of valid CP/M file names.

FILENAME	
IS-FILE	
OH!	
87654321	
A+B+C	This combination is not recommended because the plus sign can be misinterpreted in some instances.

The following are all invalid CP/M file names:

ISFILENAME	Too many characters
IS,FILE	Contains illegal character
<FILE>	Contains illegal characters
IS)FILE	Contains illegal character
FILE^C	Contains control character
WORD*	Contains illegal character

When selecting a name for a file, it is best to pick a name which is meaningful and obvious. A file containing the instructions for your payroll program might be called PAYROLL, for instance. Naming your payroll program PR102 might make perfectly good sense to you, but it is more likely you will forget what is in a file so named faster than you will forget what is in a file called PAYROLL.

A common practice we suggest you avoid is shortening the file name to save entering extra keystrokes. Naming your payroll program "P" just to save typing six letters may seem like a good idea at first, but imagine looking at the contents listing of a diskette and seeing nothing but one-letter file names. If you have a lot of files, it may take you longer to figure out what files are on the diskette than it does to type the extra six letters in PAYROLL.

File Types

It is also a good idea to identify files in a manner that lets you know what a file is going to be used for, since different types of information are handled in different ways. For example, certain program file operations could destroy the contents of a data file, were it accessed by mistake.

To keep such disasters from happening, CP/M-80 and CP/M-86 rely upon a file type to express the file function. The file type follows the file name and consists of up to three characters chosen from the list of valid file name characters. The type is separated from the file name by a period; that is why a period cannot be imbedded within a file name. When CP/M sees a period in the file identification, it assumes what follows is the file type.

The following is a comprehensive listing of the file types commonly used by CP/M and MP/M users:

.ASC	File containing ASCII text
.ASM	Assembly language source program file (CP/M-80)
.A86	Assembly language source program file (CP/M-86)
.BAK	Backup file
.BAS	BASIC source program file
.BRS	Banked-resident system process (MP/M)
.C	C source program file
.CAL	SuperCalc data file
.CMD	Directly executable transient program (CP/M-86)
.COB	COBOL source program file
.COM	Directly executable transient program (CP/M-80)
.DAT	Data file
.DOC	Document file (text file)
.FOR	FORTRAN source program file
.HEX	Intel HEX format object code file (CP/M-80)

.H86	HEX format object code file (CP/M-86)
.INT	BASIC intermediate code program file (CBASIC)
.IRL	Indexed library file
.LIB	Library file
.LST	Assembly language print listing file (CP/M-86)
.MAC	Macro assembly language source program file
.OBJ	Machine code (object code) file
.OVR	Overlay file (MicroPro, Sorcim use this)
.PAS	Pascal source program file
.PCO	Pascal run-time program file (Sorcim Pascal/M)
.PLI	PL/I source program file
.PRL	Page-relocatable file (MP/M)
.PRN	Assembly language print listing file
.REL	Relocatable machine code (object code) file
.RSP	Resident-system process file (MP/M)
.SRC	Source file (used by CP/M Users' Group)
.SPR	System process file (MP/M)
.SUB	Command file for SUBMIT program
.SYM	Symbol table file (CP/M-86 and others)
.SYS	System file for CP/M-86 and MP/M
.TEX	Document file (TEX file)
.TXT	Document file (text file)
.$$$	Temporary file, or improperly saved, unusable file

While there are other types in use, this list represents about 95% of those you will encounter. As you can see, some types help to identify particular programs. In addition, they aid in selecting files to use or to print.

You can also invent your own file types. For example, you might use the version number of a program during the development stages. If you were working on a BASIC program called THEBEST, your list of files on a disk might look like the following:

```
THEBEST.001
THEBEST.002
THEBEST.003
THEBEST.004
THEBEST.BAS
```

THEBEST.BAS represents the finished product, and each of the other types represents the program at a different stage of development.

File types can be one, two, or three characters in length, or non-existent. That is,

the file type may sometimes—but not always, depending upon the program you are using—be omitted when you name a file.

While CP/M-80 and CP/M-86, for the most part, ignore the file type, many programs demand specific file types. For example, the CBASIC compiler requires source program files to have the type ".BAS", while a CBASIC compiled program needs the type ".INT". But these types are meaningful to the CBASIC program only; CP/M-80 and CP/M-86 know nothing about the CBASIC program needs, nor do they take such needs into account in any way.

In early Digital Research documentation (and for that matter, the first revision of this book), you will often find file types described as file extensions. This nomenclature derives from the fact that the three-letter file type was originally considered an extension of the file name. The word "type," in context, is more meaningful than "extension," so we will use file type exclusively in this book. If you are reading other documentation which describes "CP/M extensions," simply substitute "types" for the word "extensions."

Combining File Name and File Type

Sometimes in a command you can specify a file by typing just the file name (the eight-character label). Other times you must type both the file name and the file type. The file name and file type are always separated by a period; valid examples of the combined file name and type are

```
WATNOWMY.LUV
ROBOT.1
MICRO.SFT
THX1138.PIK
KINGAND.EYE
```

Notice that neither are eight characters required in the file name, nor are three characters required in the file type. A file name must have at least one character, although files need not have a type. In other words, you may have file names of one to eight characters in length, and file types of zero to three characters. Both CP/M-80 and CP/M-86 ignore any characters after the eighth one in a file name and the third one in a file type; thus, if you type

```
CPMUSERGUIDE.BOOK
```

when requested for a file name and type, CP/M-80 and CP/M-86 interpret that as

```
CPMUSERG.BOO
```

In short, CP/M lops off any character after the eighth one in the name and the third one in the type.

Now for a caveat: computer language is not standardized enough for universal definitions. For some programmers "file name" represents the *entire* combination of file name, period, and file type. Terms like "file reference," "file extension," "primary name," and "secondary name" are often used to refer to the various parts

and combinations of the file identification. In this book, *file name* describes the label (one to eight characters in length) used to refer to the file, and *file type* refers to the function of the file. In examples where you are to fill in a specific file identification, we will use the form

```
TYPE filename.typ <cr>
```

which would mean to enter the command TYPE, a SPACE, the file name, a period, the file type, and then CARRIAGE RETURN. Remember, words in upper-case letters indicate information that must be typed exactly as depicted, while lower-case letters indicate places where you must substitute your data of the type indicated.

Disk Drive Identifiers

CP/M refers to available disk drives with the letters "A," "B," "C," "D," and so on. The first, or *primary,* drive of your system is the "A" drive. Other drives follow in alphabetical order (the second drive is the "B" drive). The only exception occurs when hard disk and floppy disk drives are intermixed on a computer system. Some manufacturers use later letters ("M," "N," "O," or "P") to indicate the hard disk drives and early letters ("A," "B," "C," "D,") to indicate floppy drives. Check your manuals if your system uses mixed drives.

Since file names also consist of letters, CP/M-80 and CP/M-86 must distinguish between when the letter "A" refers to a drive and when it is part of a command or a file name. To denote a disk drive, enter a colon (:) after the drive letter; for example,

```
A:
```

In this book and in all CP/M manuals, A:, B:, C:, and so forth always pertain to disk drives. When we provide examples in this book which require you to enter a disk drive identifier, we will do so as follows:

```
TYPE d:filename.typ <cr>
```

This will indicate you are to enter the command TYPE, a SPACE, a valid disk drive identifier, a colon, a file name, a period, a file type, and then a CARRIAGE RETURN.

If you have used other disk operating systems, you may have trouble with the disk drive indentifiers; many operating systems use numbers instead of letters to identify drives. Since most manufacturers do not label the disk drives, it makes sense for you to place a label on each drive in your system with its proper CP/M identifier.

When you start CP/M-80 or CP/M-86 you will almost always use the "A" drive. Unless you specify another drive, the "A" drive will be the default—or currently logged drive—for all commands and programs you invoke. This means that to use the "A" (*default*) drive, you need not type its identifier.

Wildcard References

Sometimes you may not know the name or type of a file you wish to use, or you may wish to specify more than one file at a time. Digital Research has provided for such occurrences with a set of wildcard references, sometimes called *ambiguous file references.* The asterisk (*), when used in a name or type, replaces the *entire*

remaining portion of a file name or file type or both. The question mark (?), when used in a name or type, replaces a single character *within* the name or type.

In Digital Research manuals, ambiguous file references are abbreviated *afn* (for ambiguous file name), while the complete file name will be referred to as the *ufn* (for unambiguous file name).

Many CP/M commands require you to enter unambiguous file references in the command line (the complete file name). This means that you must precisely specify the complete file name and file type to invoke the command properly.

The asterisk can represent the entire set of characters on either side of the period in a complete file name. For instance, all of the following are valid ambiguous file specifications in CP/M-80 and CP/M-86:

`*.BAS`	All files with type of BAS
`THEBEST.*`	All files with the name of THEBEST
`*.*`	All files

Now wait a minute, you ask, what good is that? Consider the directory command. To learn the names of all the BASIC programs on drive "A," type the following command line:

`DIR *.BAS <cr>`	DIR stands for directory

The computer displays all files with the type "BAS." To erase all BASIC source program files on the diskette in drive "A," type

`ERA *.BAS <cr>`	ERA stands for erase

You may have already guessed that we can use the question mark like the asterisk. For instance, if PROGRAM1.ASM is the only file with the type "ASM," the following commands will be treated identically by CP/M:

`ERA *.ASM <cr>`	Erase all files with type ASM
`ERA PROGRAM1.ASM <cr>`	Erase specific file PROGRAM1.ASM
`ERA ???????1.ASM <cr>`	Erase all files with type ASM and last letter of file name of 1

Notice how seven question marks precede the "1" in the last example. This is because the question marks are considered to occupy one character position only, not several. Typing ?1.ASM and ???????1.ASM is not the same thing to CP/M.

Most CP/M systems have two or more drives. Remember that CP/M identifies a disk drive with a letter followed by a colon. The command

```
DIR B:*.* <cr>
```

displays *all* files on drive "B." (The prompt A> indicates drive "A" is the default drive. If we did not type B: in front of the above command, all files on drive "A" would be displayed by default.)

BUILT-IN COMMANDS

Built-in commands were briefly mentioned earlier in this chapter. In the following section each built-in command is described in detail. Differences between usage with CP/M-80 version 1.4, CP/M-80 version 2.2, and CP/M-86 will be explicitly described.

DIR—Display the File Directory

The DIR command displays a directory of files on a diskette. Several forms of the command are allowed. All file names can be displayed, diskette directories may be searched using the wildcard characters to select a group of similarly named files, or the user may specifically search for a single file.

Early versions of CP/M-80 (versions 1.3 and 1.4) provide directories in a single column with one file name and type on each line preceded by the drive specification, as in the following example:

```
A>DIR B:<cr>
B:DOCUMENT MAR
B:DOCUMENT APR
B:DOCUMENT JUN
A>
```

Version 2.2 of CP/M-80 and CP/M-86 display directories with as many as four files on a line, each separated with a vertical line as follows:

```
A>DIR B:<cr>
B:DOCUMENT MAR| DOCUMENT APR| DOCUMENT JUN| DOCUMENT JUL
B:DOCUMENT AUG| DOCUMENT SEP
A>
```

Some CP/M-80 version 2.2 directories display only two columns because of reduced screen width (notably, the Osborne 1).

The following are the valid forms of the directory command:

DIR d:<cr>

Displays the directory of all files present on the drive specified (d:). The drive specification may be omitted, in which case the default drive is used for the search.

DIR d:filename.typ <cr>

Displays the directory of all files present on the drive specified (d:) that match the file name and type specified. You may use the wildcard specifiers "*" and "?" to search for groups of files.

The following are sample uses of the CP/M DIR command and the resulting directories:

```
A>DIR B:*.BAS<cr>
B:FRESHWTR.BAS
B:SALTWTR.BAS
A>
```

The above example shows a directory in CP/M-80 version 1.4 format. The command used is DIR B:*.BAS (display a directory of all files with the type of "BAS" that are on drive "B").

```
A>DIR FI??.*<cr>
A:FISH   TRT | FISH   BAS | FISH   CRP | FISH   ING
A:FISH   EEE | FILE   BAS
A>
```

The above example shows a directory in CP/M-80 version 2.2 format. The command used is DIR FI??.* (display all files whose names start with "FI" and have any file type, all on the default drive).

Error Messages

You can make several different errors in using the DIR command. The primary error messages CP/M might display are

NO FILE, NOT FOUND, or FILE NOT FOUND

The diskette does not contain the file(s) you specified. Make sure that you typed the command correctly; if you did type the command correctly, then the file(s) does not exist.

BDOS ERR ON d:

(Where d: is the drive identifier.)

CP/M could not find a diskette in the selected drive, the diskette is improperly formatted, the power to the drives is off, or the drive door has not been closed. Check to make sure that a diskette is properly inserted into the drive. If the above message is followed by a BAD SECTOR message, in all probability you either have a bad diskette or you inserted the diskette upside down.

DIT?

Any error message ending in a question mark indicates that CP/M could not find the command you typed. In this particular instance, you typed DIT instead of DIR. Check carefully for typing errors when you see an error message ending with a question mark.

ERA—Erase a File (CP/M-80 and CP/M-86)

ERAQ—Erase a File with Query (MP/M)

ERA stands for erase. To use the command, you must specify the file or files to be erased immediately after typing the command. Since it is extremely rare to remove all files from a diskette, CP/M-80 and CP/M-86 will check with you to make sure if

you requested erasing all files from a diskette (this is not true of CP/M-80 version 1.3).

It is a good habit to develop to check the directory before invoking the erase command. First, verify that the file or files you wish to remove from the diskette actually exist by using the DIR command. Next, use the ERA command to actually perform the erasure. Last, use DIR again to make sure that CP/M correctly erased the selected file(s). If you do not use this procedure, you will never have any verification of when and what CP/M erases.

Users of MP/M can use a special version of the ERA command, ERAQ, to force MP/M to query before each file erasure. While ERA is also available under MP/M, if you use this CP/M cousin you should get into the habit of using ERAQ instead.

One form of the ERA command is allowed in CP/M; it is

ERA d:filename.typ <cr>

Erases all files which match the file name and type specified on the diskette specified (d:). The wildcard characters "*" and "?" may be used to specify more than one file at a time.

In addition to the above form, MP/M users may also use the following form:

ERAQ d:filename.typ <cr>

Erases all files which match the file name and type specified on the diskette specified (d:), but erases each file only after a user confirmation is entered.

Here are some examples of use of the ERA command.

```
A>DIR C:<cr>
C: QUALITY CTL | MIND  CTL | WEIGHT CTL | THOUGHT CTL
A>ERA C:QUALITY.CTL<cr>
A>DIR C:<cr>
C: MIND  CTL | WEIGHT CTL | THOUGHT CTL
```

In the above example we first ask for a directory of files on drive "C." Four files are listed. We then ask CP/M to erase the file named QUALITY.CTL on drive "C." No message is presented (other than the A>) to confirm this action, so we ask for another directory to confirm the deletion.

```
A>DIR B:*.BAS<cr>
B: NOW   BAS | THEN  BAS | ALWAYS  BAS
A>ERA B:*.BAS<cr>
A>DIR B:*.BAS<cr>
NOT FOUND
A>
```

In this example we find three files on the diskette in drive "B" with the type of "BAS," and then we ask CP/M to erase all files on drive "B" with the type of "BAS." Another directory command confirms that no files with the type of "BAS" are left on drive "B."

```
B>DIR <cr>
B: SOHO    NY| CHELSEA   NY| UPPREAST NY| UPPRWEST NY
B>ERA *.*<cr>
ALL FILES (Y/N)?Y <cr>
B>DIR <cr>
NO FILE
B>
```

The above example demonstrates what happens when you ask CP/M-80 or CP/M-86 to erase all files from a diskette. Note the message ALL FILES (Y/N)? and the user response that follows it. If you reply with anything other than an "N" to the query, no files are erased from the diskette. In this case, we replied with a "Y" to tell CP/M to proceed.

The following example is applicable to MP/M only:

```
A0>ERAQ SNWAKEEN.*<cr>
A: SNWAKEEN VLY?y
A: SNWAKEEN RVR?n
A: SNWAKEEN CTY?n
A0>DIR SNWAKEEN.*<cr>
A: SNWAKEEN RVR| SNWAKEEN CTY
A0>
```

Notice that in this MP/M example we have used the ERAQ (erase with query) option. Each file whose name and type match the one we specified is presented one at a time for our consideration. We decided to erase the first one, but not the next two that were presented us. A directory command confirms our actions.

The error messages which you might encounter using the ERA command are basically the same as those described with the directory command. They are

NO FILE, NOT FOUND, or FILE NOT FOUND

CP/M or MP/M could not find the file name and type you specified. If you type ERA KNOW instead of ERA NOW, for instance, this message might appear. But if you did have a file named KNOW, it would be erased. Do not count on this error message to catch your typing mistakes.

BDOS ERR ON d:

(Where d: is the drive identifier.)

CP/M could not find a diskette in the drive you specified, the drive door is open, no diskette has been inserted or one has been inserted improperly, or the power to the drive has not been turned on.

ERAQ?

Again, any error message ending in a question mark indicates that CP/M was unable to find the command you typed. If this particular error message appears, either you attempted to use an MP/M command with CP/M-80 or CP/M-86, or if you are using MP/M, the file ERAQ.PRL is not on the default disk drive.

REN—Rename a File

Files may be given new names with the REN, or rename, command. You must explicitly state the old name of the file and its new name; you cannot use the wildcard identifiers "*" and "?" to rename a group of files at once.

Almost universally, whenever the computer deciphers an equivalence statement, as in the rename command, the new formation or result is to the left of an equal sign, and the old formation or result appears to the right. This can be expressed as follows:

NEW = OLD

A disk drive specifier can be included with either one or both file names when you use the rename command. If it is included with one name, it applies to both. If it is included with both file names, the same drive specifier must be used with both. If the disk drive specifier is omitted, the default disk drive is used.

Here is the one form of the REN command.

REN d:newfilename.typ=d:oldfilename.typ<cr>

The rename command finds the file of name and type matching "oldfilename.typ" on the drive specified (d:), then gives it the new name and type matching "newfilename.typ."

Here is an example of the use of the REN command.

```
A>DIR<cr>
A: HOLDEN  ONE | PHOEBE  TWO | 19EIGHTY FOR
A>REN NOVEL.ONE=HOLDEN.ONE<cr>
A>DIR <cr>
A: NOVEL   ONE | PHOEBE  TWO | 19EIGHTY FOR
A>
```

In this example we ask to change the file named HOLDEN.ONE to a file named NOVEL.ONE.

The rename command has several error messages associated with it; they are

FILENAME?

You incorrectly used an ambiguous file reference in the rename command line. For instance, if you type REN TALL.CAR=SHORT???.CAR, the error message SHORT???.CAR? appears.

NO FILE

The file you specified does not exist. Check your typing.

FILE EXISTS

The new name you specified is the name of a file which already exists. You cannot use the REN command to rename a file to a name and type that already exist on a diskette; CP/M-80 and CP/M-86 consider this to be an error. If you wish to replace an existing file with a newer version of the same file, either rename or erase the existing file first, or use the PIP utility described in Chapter 3.

BDOS ERR ON d:

(Where d: is the disk identifier.)

CP/M could not find the diskette or activate the disk drive you specified. Check to make sure the diskette is inserted properly and that the disk drive is powered on with the door correctly closed.

SAVE—Save Memory Contents in a Disk File

One crude method of saving memory contents in CP/M-80 is the SAVE command. This command is not applicable to CP/M-86.

SAVE places the contents of what is known as the Transient Program Area into a file on diskette, using a name and type you choose. You must tell CP/M-80 how many pages (256-character blocks) of memory you wish saved and the name and type of the file you wish that information stored in. Select the file name carefully. SAVE erases any existing file of the same name before creating the new one.

You will rarely use the SAVE command if you only use CP/M-80 to run ready-made or canned business application programs. On the other hand, assembly language programmers will use this command frequently. For continuity's sake, the SAVE command is presented at this point in the book; if you are not familiar with assembly language, or you want more background before tackling with the particulars of SAVE, you should read Chapter 4 before proceeding with the rest of this description. Since DDT, the debugger utility program, is normally used in conjunction with the SAVE command, we have used DDT in our example. DDT is also examined in Chapter 4, Assembly Language Utilities.

The one form allowed for the SAVE command is

SAVE ## d:filename.typ <cr>

Where ## is the number of 256-character blocks of memory you wish to store on diskette.

In order to perform a SAVE, you must know the number of "pages of memory" to be saved. A page of memory is a block consisting of 256 characters. The first page saved always begins at memory location 0100 hexadecimal. The number of pages to be saved must be expressed as a decimal number.

Beginners often encounter problems with the "page of memory" concept and do not understand how 256 decimal is equivalent to 0100 hexadecimal. The number conversion between decimal and hexadecimal is not an easy one for anyone with "math anxiety."

The hexadecimal numbering system (usually abbreviated as "hex") is equivalent to base 16. In other words, the one's place can have 16 digits, as opposed to the ten digits in base 10. In base 10 we count

0 1 2 3 4 5 6 7 8 9

In base 16 (hex math) we count

0 1 2 3 4 5 6 7 8 9 A B C D E F

The letter "A" represents our normal number 10, "B" represents 11, and so on. In everyday math we would interpret 111 to mean

```
  1 in the 1's place =   1
 1 in the 10's place =  10
1 in the 100's place = 100
     for a total of    111
```

In hex math 111 means

```
  1 in the 1's place =   1
 1 in the sixteen's place = 16
  1 in the 256's place = 256
      for a total of    273 (in base 10)
```

But, you may ask, what does this have to do with the SAVE command?

Remember that 0100 hex equals one page of memory (256 decimal). Does 0200 hex equal two pages of memory? Yes. Since we are only concerned about the number of pages of memory to save, we can forget about the last two digits in converting computer memory addresses into the necessary information to fill out the SAVE command.

Let's calculate the number of pages of memory if we wish to save information which resides between 0100 hex and 2785 hex in our computer.

1. First, forget the first address, 0100; the SAVE command always starts at this address.
2. Next, drop the last two digits from the higher address. In our example 2785 thus becomes 27.
3. Convert the remaining number to decimal. For our example

```
  7 in the 1's place =  7
 2 in the 16's place = 32
     for a total of   39 (in decimal)
```

4. Step 3 always gives the correct result except for one special case. If the last two digits of the higher address in step 2 were 00 hex, you would subtract 1 from the result in step 3. For example, if the higher address were 2700 hex instead of 2785 hex, the correct result would be 39 minus 1, or 38 decimal pages to SAVE.

Now that you know how to perform number conversions from hexadecimal to decimal, let's look at an example of SAVE's use.

```
A>DDT CURSEAND.HEX<cr>
DDT VERS 2.2
NEXT PC
1100 0100
-
```

In this part of the example we use DDT (discussed in Chapter 4) to load a file named CURSEAND.HEX into memory. As you will eventually learn, the numbers under "NEXT" and "PC" are significant: the first indicates one more than the last memory address used by the file, while the second indicates the starting address of the file. If we use the rules given above for converting these numbers into the number of pages of information to save, we come up with the answer 16. (We arrived at the answer because 1 in the 1's place equals 1, plus 1 in the 16's place equals 17, subtracting 1 because the last two digits of the higher address are 00 hex.)

Therefore, we would use the following SAVE command:

```
A>SAVE  16  CURSEAND.COM<cr>
A>
```

Error Messages

FILENAME?

You failed to specify the number of pages to save, used a wildcard identifier, or misspelled the word SAVE. Check your typing for accuracy before going on. Unfortunately, due to a quirk in the way some CP/M-80 systems use memory during the SAVE process, you cannot assume that the data you wished to save is still valid. In short, you must first reload the memory with the information you wish to save before reattempting the command.

NO SPACE or DISK FULL

Too many files are already on the diskette, or no room is left on the diskette to save all the information you specified. Either erase unnecessary files on the diskette before proceeding, or use a different diskette.

TYPE—Display a File Containing ASCII-Coded Information

If you have a file which consists of printable characters (that is, data, as opposed to computer instructions), you may ask CP/M to display the file's contents on your console display by using the TYPE command.

TYPE usually works for files with types of "BAS," "ASM," "BAK," "DAT," "HEX," "DOC," "TXT," or any other file which contains ASCII text or data. ASCII is a character recognition and storage scheme for computers; each character has a unique representation that can be saved or used by the computer.

You could also use a text editor or word processing program to look at the contents of a file, but not everyone using CP/M has such a program available. If you merely want to take a peek at your program or data, why go to the trouble of using the text editor? The TYPE command is built into all versions of CP/M and is always available.

There is only one form of the TYPE command.

TYPE d:filename.typ <cr>

Displays the contents of the file specified.

An example of a use of TYPE would be the following:

```
A>TYPE O.POS<cr>
Had Dick Seeker been present, and had he been able to see
inside the NASCOM Cray-1 computer, he would have seen a
string of bits spelling out his name, computer style. But
the National Security Computer Center was 1500 miles away;
in fact, Dick wasn't even aware of its existence.
```

In the above example, the user asked to see a text display of the file O.POS, and the computer complied. If this had been a long file—longer than the console display could show at one time—a ^S could have been typed to "pause" the display temporarily. To terminate a text display of a file, press any key, or enter a ^C.

MP/M users may use an optional two-digit number (as shown in the following example) with the TYPE command to specify how many lines of text to show on the console at one time.

TYPE d:filename.typ P## <cr>

The number signs (##) represent the number of lines to display at a time.

When the MP/M form of the TYPE command is used as described above, the computer displays the specified number of lines following the "P" in the command and then waits for the user to press the CARRIAGE RETURN key. In this fashion, an MP/M user may display a text file without worrying about text running off the screen before it can be examined (the normal specification would be "P23" for 24-line terminals).

Error Messages

FILENAME?

This error message appears when the file you named does not exist or you misspell the TYPE command. Check your work and try again.

BDOS ERR ON d:

(Where d: is the disk identifier.)

CP/M could not find the diskette or disk drive you specified.

Meaningless display (sometimes accompanied by a bell)

You tried to use TYPE with a file which does not contain ASCII text or which includes machine language code with the text. If you accidentally displayed a file with machine language code in it, restart CP/M to make sure that you did not also accidentally change the contents of your computer's memory.

USER—Change the Currently Logged User Number

CP/M-80 version 2.2, CP/M-86, and MP/M all include a special command named USER. This command allows you to specify a number between 0 and 15,

inclusive, which is to be kept with any files you create.

When you cold start CP/M-80 or CP/M-86, user number 0 is assumed; when you cold start MP/M, the user number is initialized to the console number. Your disk operations will reference files in user area 0 only (or another number if you are using a different console number in MP/M). In other words, if you save a new file called JUNK.AGN after a cold start, it will always appear in directories assigned to user area 0. Type USER 2<cr> before you save the file, and it will appear only in the user area 2 directory.

"User areas" are imaginary. Every file on a diskette has a user number associated with it, stored on the diskette as an additional piece of information about that file. Thus, it is not necessary to set aside room on the diskette for each user area.

The USER command is of minimal value in running canned programs. However, when several users share disk drives, as in MP/M, one user can save files in user area 1 and another in user area 2, and both could share files in user area 0.

The only form of the USER command allowed is

USER ##<cr>

Where ## is a number between 0 and 15, inclusive.

When you utilize different user areas the ERA command functions differently. For example, ERA *.*<cr> only erases all files in the currently chosen user number. There is no way to erase all files on the diskette with a single command, unless they are all in the same user area.

Almost without exception, any diskette you receive from software vendors will have all files in user area 0.

Error Messages

? or #?

You forgot to specify a number or specified a number greater than 15, or you misspelled the command name USER. Check your work and try again.

FILENAME? or NO FILE

These messages may appear if you change user areas and attempt to access programs or data not in the current user area.

d:—Change Default Disk Drive

On a CP/M system with two or more drives, change the currently active (default) drive by typing the letter representing the drive being logged into, followed by a colon and a CARRIAGE RETURN.

```
B:<cr>
C:<cr>
L:<cr>
```

To revert to drive "A," type

```
A:<cr>
```

When you change the default drive, CP/M changes its prompt letter. Following B:<cr>, CP/M-80 and CP/M-86 return with a prompt of B>; MP/M would return with the prompt #B>, where the number sign represents the user area.

When selecting a file on the default drive, you do not have to type the drive letter with the file name, but in any reference to files on drives other than the default, you must specify the desired drive.

Error Messages

BDOS ERR ON d: SELECT

The above error message will appear if you attempt to change to a drive which does not exist or which CP/M cannot find (perhaps because the power is off or the door left open).

LINE EDITING COMMANDS

Line editing commands let you correct typing errors while entering command lines and give you some control over the console display (output). The line editing commands are grouped by function in Table 2-1.

The CONTROL-P and CONTROL-S commands are useful at any time, as is usually the CONTROL-H command. The other line editing commands must normally be typed within a CP/M command line, that is, after the CP/M prompt appears and before you press the CARRIAGE RETURN key to process the command.

The line editing commands can often be used when typing input requested by transient programs. Their usefulness in this case will depend on the program.

CONTROL-C—Restart CP/M-80

A *warm start* restores CP/M-80's internal information to a predefined state without destroying programs or data stored in memory. A *cold start* starts a system from scratch, destroying programs which were in memory before the cold start.

A cold start is often called a *cold boot*, and a warm start is often called a *warm boot*. In CP/M-80, the warm start command has two primary uses. They are

- To "log in" a diskette when you insert a different diskette into one or more drives
- To interrupt the current transient program and return to the CP/M-80 command level.

You can create problems if you do not "close a file" (tell CP/M-80 that you are through with a file) at the proper time. You can avoid such problems by always allowing a program to end normally (if there is an option in the program to QUIT, use it rather than pressing RESET or ^C).

If you change diskettes without telling CP/M-80 or CP/M-86, strange things happen. Always press CONTROL-C after you insert a different diskette into a disk drive when you are using CP/M-80, unless the diskette is specifically *requested* by

the program. In this latter instance, you have to assume that the programmer knew that you were changing diskettes and that he or she included instructions to inform CP/M-80 of the requested change.

CP/M-86 works a little differently than CP/M-80 when a CONTROL-C is typed. CP/M-86 allows multiple programs to reside in memory. CP/M-86 keeps track of the order in which programs are started. Each time you type a CONTROL-C in CP/M-86, the program last invoked is stopped. When all programs have been stopped, a CONTROL-C restarts CP/M-86, much as CONTROL-C restarts CP/M-80. It is important to note, however, that diskettes cannot be changed in CP/M-86 unless a program requests such a change or you type a CONTROL-C with no program running.

MP/M also works differently. CONTROL-C in MP/M only stops execution of the current program. Because multiple users may be logged onto a computer with MP/M, a special command is necessary to reset specific diskettes, that is, to restart one user, not all users. This command is DSKRESET.

`DSKRESET <cr>`	Resets all diskettes
`DSKRESET d: <cr>`	Resets disk drive d:
`DSKRESET d:, d: <cr>`	Resets multiple drives (each separated by a comma)

Disk resets may be denied if other users are using the same diskette. In such a case you will see a message indicating which console and program are using that diskette. DSKRESET should be executed before removing a diskette from a drive, otherwise you may invalidate another user's data.

Error Messages

BDOS ERR ON d: R/O

(Where d: is the drive identifier.)

CP/M can usually detect a switched diskette. When it does, it sets the read-only attribute for that diskette. If any program subsequently tries to write information onto that diskette, this error message appears. The only way to recover from this error and not destroy information on your diskette is to type CONTROL-C (with no program running if you are using CP/M-86).

The experienced user notices that the error message received when CP/M detects a switch in diskettes mentions the diskette is "read-only." This error message implies that you can switch diskettes at any time as long as you only read from the diskette. If you use your system like most users, however, it is rare for a diskette to be read from, but not written to. Thus, this is a potentially dangerous suggestion.

TABLE 2-1. Line-Editing Commands Grouped by Function

Function	Commands
Terminate command line	CARRIAGE RETURN (<cr>) LINE FEED (<lf>) CONTROL-J CONTROL-M
Cancel command line	CONTROL-C CONTROL-X
Cancel one character of command line	BACKSPACE (<bs>) DELETE (<del>) RUBOUT (<rub>) CONTROL-H
Display control	CONTROL-E CONTROL-R CONTROL-S
Printer control	CONTROL-P
Restart CP/M-80 Terminate function CP/M-86	CONTROL-C

CONTROL-E—Continue Typing On Next Line

To continue typing a long command on the next line of the console display, type CONTROL-E. This moves the cursor to the beginning of the next line. When you ultimately press the CARRIAGE RETURN key, the entire command line is used, even though it may appear as several lines on the display. The CONTROL-E character itself is not considered part of the resulting command line.

CONTROL-H or BACKSPACE—Delete Last Character

Version 2.2 of CP/M-80 and CP/M-86 allow you to use CONTROL-H or BACKSPACE to correct simple typing errors before pressing the CARRIAGE RETURN key.

If your keyboard has a key labeled "BACKSPACE" or "BS" you can use it; in any case, you can type CONTROL-H. The two are just different names for the same function.

CONTROL-H is similar to RUBOUT or DELETE; it differs in that it erases the unwanted character from the screen, while RUBOUT or DELETE leaves the unwanted character on the screen and repeats it.

CONTROL-H or BACKSPACE is intended for video displays rather than printers. But CP/M-80 or CP/M-86 will still respond correctly to CONTROL-H even if you are using a printer; the printer may respond strangely, however.

DELETE or RUBOUT—Cancel One Character and Echo it

The DELETE key cancels the last uncanceled character in the command line and echoes (repeats) it. Press the key marked "DELETE," "DEL," "RUBOUT," or "RUB" to do this.

Versions 1.3 and 1.4 of CP/M-80 usually lack the true BACKSPACING DELETE as we described in the section on CONTROL-H. Instead, these versions almost always perform the DELETE function which echoes, or repeats, each deleted character. Type **BLA** and then DELETE or RUBOUT; you will see either

```
BLAA      Deleted characters are echoed
```

or

```
BL        Deleted characters are erased (rubbed out)
```

Echoed characters may seem a little strange to computing newcomers. When CP/M-80 was first designed, the primary console device on most microcomputers was a Teletype or similar printer. These printers cannot backspace. In order to show a character was erased, Digital Research used echoing in the earliest versions of CP/M-80.

Times have changed, and today most computers use high-speed video display consoles. Such consoles have the ability to backspace and erase characters. Digital Research will provide a patch to any user of CP/M-80 version 1.3 or 1.4 who wants true backspacing deletions.

CONTROL-J and CONTROL-M—LINE FEED and CARRIAGE RETURN

If you are using CP/M-80 version 2.2 or CP/M-86, you can substitute CONTROL-J or the LINE FEED key (if your keyboard has one) for the CARRIAGE RETURN key. If your keyboard has a LINE FEED key, it might be marked "LF."

Typing CONTROL-M is exactly the same as pressing the CARRIAGE RETURN key when typing CP/M-80 and CP/M-86 commands.

CONTROL-P—Assigning Output To the Printer

To turn on the printer, type a CONTROL-P. If your version of CP/M has been configured for a printer, all output going to the console display is now also sent to the printer.

CONTROL-P works like a push-on, push-off switch: type it once to turn the printer on, type it a second time to turn it off.

Remember, when you use CONTROL-P to turn the printer on, the printer mimics the screen. Unless someone added a printer interface to your version of CP/M, you will most likely have to tolerate the following minor inconveniences:

1. Your printer will not paginate; it simply types line after line, oblivious to the end of the page.
2. Any control code imbedded in the text (a clear screen command, for example) may adversely affect the printer. Many computers use the ASCII FORM FEED character to clear the screen; thus, every time your screen clears, the printer may skip to the next page.
3. Unless you have a high-speed printer, output to your console display slows down. Every time the computer wants to send a character and finds the printer busy, it waits to send that character.

Some programs disable the printer automatically, whether or not you have selected it. These programs, such as WordStar, have other commands for sending information to the printer.

Likewise, if you purchased your programs as complete, prewritten (canned) packages, they may turn the printer on and off as necessary. Do not type a CONTROL-P before executing such a program unless directed by the manual; let the computer do this work whenever possible.

Error Messages

If you type a CONTROL-P and double characters appear on your screen but none appear on your printer, contact the firm that sold you the system.

```
A>DIR^P<cr>
AA:: SSTTUUTTTTEERR..TTXXTT || RREEPPEEAATT..DDOOCC
AA>>
```

Repeating characters indicate that CP/M has not been told where to send characters destined for the list device (usually the printer). Instructions must be added to the BIOS section of CP/M to get information to the printer properly.

Another problem that can occur is that, immediately after typing a CONTROL-P, the system "locks up," in other words, stops accepting characters from you. If this happens, it usually means that the printer is either not connected properly (check to see if it is unplugged) or not turned on.

CONTROL-R—Repeat Current Command Line

If you are a bad typist and have an early CP/M-80 system that echoes deleted characters, you might find that the command

```
DIR B:BASIC???.*
```

looks more like:

```
DIBBR BAA:BASIXXC?????.*
```

If you have to read command lines like that, you may come to hate your computer. CONTROL-R comes to the rescue. Type a CONTROL-R to display a new, correct line below the original command line. This new line deletes all the characters that were echoed instead of being erased. Here is an example.

```
DIBBR BAA:BASIXXC?????.*^R
DIR B:BASIC???.*
```

The CONTROL-R command does not work with CP/M-80 version 1.3. Users of CP/M-80 version 2.2 and CP/M-86 will probably not find the CONTROL-R command useful.

CONTROL-S—Pause the Display

CP/M-80 and CP/M-86 can *pause* the console display. Like a freeze frame in film, everything stops until you tell it to begin again. Pause the screen by typing a CONTROL-S. Another CONTROL-S resumes the output to your display (actually, typing any character except CONTROL-C will resume the output to the display). CONTROL-S really pauses the computer, and as a result the display pauses.

However, this is not the best possible answer to handling overflow on the screen. Generally, you are surprised by the overflow, or respond too slowly to pause the frame exactly where you want. "Good" programs use formatted screen output and never attempt to put more information on the screen than it can handle. "Good" programs also include an automatic stop at the end of each screen of information and wait for you to press the CARRIAGE RETURN (or some other key) before resuming output. In short, if you find yourself using the CONTROL-S key while executing a program, you might consider other programs with better human interfacing, or ask your vendor to modify your program.

CONTROL-U/CONTROL-X—Cancel Current Command

If you start typing gibberish, do not worry, it happens to all of us. Perhaps you typed

```
DIRG A:THOMKJLSK.SAB = SLIUF
```

That does not make any sense whatsoever. Instead of using the BACKSPACE or DELETE key to get back to the first mistake, it may be easier to start over by typing CONTROL-U. CP/M will ignore what you have already typed and move to the beginning of the next line. CP/M places a number sign (#) at the end of the canceled line.

With version 1.3 of CP/M-80, use CONTROL-U. With all other versions of CP/M, use CONTROL-X if you want to erase the canceled line or CONTROL-S if you want to keep the canceled line on the screen.

CHAPTER

3 CP/M Transient Commands

In the last chapter we introduced CP/M's built-in commands. CP/M-80 and CP/M-86 also allow you to expand the basic set of commands by having any number of additional programs that act like commands. This chapter describes some of these additional programs, specifically those normally supplied with CP/M-80 and CP/M-86.

WHAT A PROGRAM IS

You will recall from Chapter 2 that built-in commands are those words which CP/M can interpret using only the instructions already present in memory, while transient commands (usually referred to as programs) are invoked by typing a phrase that acts like a command and tells CP/M to get further instructions from the diskette.

How does CP/M know what other instructions are available on the diskette?

If you type something in response to the CP/M-80 command prompt and CP/M-80 does not recognize it as a command, the diskette directory is searched for a file with the ".COM" file type that matches the first word you typed. Suppose you type

BUY <cr>

After realizing that BUY is not a built-in command, CP/M-80 looks for a file entitled BUY.COM on the current disk drive. If there is no such file, CP/M-80 issues the error message BUY?. If such a file does exist, CP/M-80 loads the contents of that file into memory and transfers control to the first instruction in it.

CP/M-86 performs almost exactly as CP/M-80—the only difference is that CP/M-86 uses the ".CMD" file type. In the above example, CP/M-86 would look for the file named BUY.CMD. There is a valid reason for the difference in file types between CP/M-80 and CP/M-86. Since both versions of CP/M use the same diskette storage method, it is possible— and in some cases, advisable—to store both CP/M-86 and CP/M-80 programs on the same diskette. Since the 8086 chip that CP/M-86 is written for cannot use programs written for the 8080 family that CP/M-80 supports, to avoid confusion Digital Research modified the file type of command files when they designed CP/M-86.

You should immediately recognize the value of the transient command facility in CP/M. If the file that matched your "command" contained instructions for the computer, it would seem just like you had typed a valid CP/M command (although it might take a bit more time to execute since the diskette must be accessed before the program starts working).

What is this program that acts like a command? For the time being we will define a *program* in the context of CP/M as follows: a program is a set of instructions stored on diskette in a file with the ".COM" type for CP/M-80 and the ".CMD" type for CP/M-86. It is invoked by typing its file name. This program is often called a *transient program* or *transient command*.

This means that once a program is loaded from the diskette into the memory of your computer, the computer temporarily ignores CP/M and obeys the instructions in the program. A hypothetical DISPLAY.COM (or .CMD) program might instruct the computer to take the following steps:

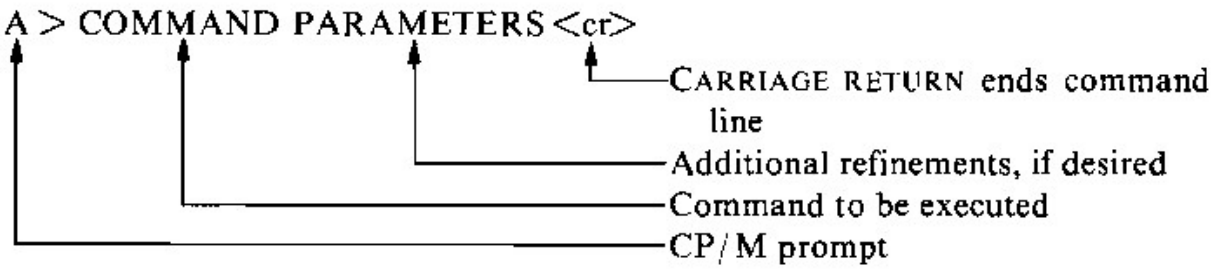
1. Ask you for the name of the file you wish to display
2. Go to the diskette and find the file you requested
3. Read a character from the file
4. Display the character on the console display
5. If there were more characters left, go back to Step 3
6. Return to CP/M.

Invoking a Program

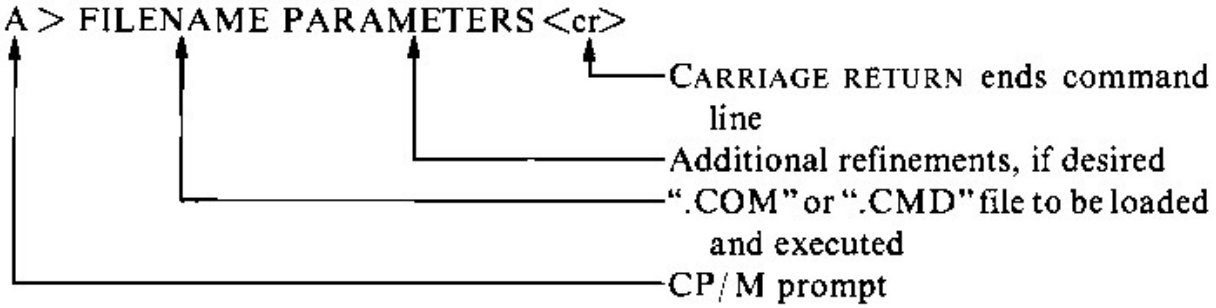
The command line is all that you type after the CP/M prompt (A >), up to but not including the CARRIAGE RETURN. CP/M allows a command line to be up to 127 bytes long; you could type 127 characters for a single command. Does it strike you as strange that the file name is, at most, eight characters long, but CP/M would accept 119 more?

Just as some commands can have extra refinements added by typing further information, so too can programs be invoked with additional information. With DIR *.BAS, for instance, the command is DIR, but B:*.BAS refines the command. Because of the lack of a better name, or even an established standard, we call this additional information parameters. There can be more than one parameter in a command line.

Let us briefly review. A generalization of the command line format looks like as follows:



Likewise, we can generalize the way a transient command, or program, is invoked as follows:



Now, what are the parameters we give the program? There is no simple answer; some programs use no additional parameters, some have optional parameters, and some require parameters. The number of parameters passed to the program may be as few as one or as many as the command line can hold.

This chapter describes a number of transient commands that you might use with CP/M. The term "transient" conveys the notion that the commands (programs, actually) are not permanently stored in memory but are kept on a diskette. We will use the term "transient command" to conform with the Digital Research description of the programs it supplies with CP/M-80 and CP/M-86.

Before moving on, we need to redefine one term we have been using throughout the first portion of this book.

There is a trend to include faster and larger capacity disk drives in microcomputer systems. These newer drives are thick, rigid, metal platters. They do not have the thin, flexible plastic surface of the diskette. These newer disk drives are called hard disks, rigid disks, Winchester disks (a nickname given to them after IBM introduced the model 3030 hard disk), or just disks. So far in this book, we used the term "diskette" exclusively. Since CP/M-80 version 2.2 and CP/M-86 were designed to accommodate hard disks, we will change our terminology slightly. Throughout the rest of this book, the term "disk" will be used frequently and is meant to include both floppy diskettes and hard disks. References to "diskettes" earlier in this book are generally applicable to hard disks also.

You now have all the information you need to proceed. While we have exhausted the list of commands that CP/M recognizes, we have barely hinted at the number of programs that may be loaded and executed as transient commands. In the remainder of this chapter, we will deal with some important transient commands

that Digital Research provides with CP/M-80 and CP/M-86. Chapter 4 deals with the remaining Digital Research programs that may be important to you.

FORMAT FOR TRANSIENT PROGRAM DESCRIPTIONS

Just as we used a structured format to describe each command introduced in Chapter 2, we will structure transient programs in this chapter as follows:

- *Program name*
  Brief description of the program and what it does, followed by the uses for the command.
- *Command line summary*

command line format...	...description
.	.
.	.
.	.
command line format...	...description

- *Purpose of the command line*
- *Example of use of the command with computer display*
- *Inputs other than command line.*

> NOTE: We have used examples from a variety of CP/M implementations although we have restricted the examples to common ones. Your version of CP/M may not display messages or information exactly as we show it here, but this is nothing to worry about. However, if the information you receive is so radically different that it does not seem to be related to our discussion, or if you receive apparently meaningless or unusual messages that do not match our examples, consult with your dealer to try to resolve the discrepancy before proceeding.

At the end of this chapter you will also find a section which describes most of the CP/M error messages you might receive while trying any of the commands introduced in this chapter.

HOUSEKEEPING UTILITIES

For computer users, *housekeeping* refers to the maintenance of the environment in which a computer program and its data operate. A well maintained environment ensures proper storage and retrieval of information; we place information so we can easily find it later. Consider a file cabinet. In this cabinet we arrange files for easy access. But what happens when the file cabinet becomes full? If we are in a hurry, do we still maintain everything in the proper order? How do we retrieve an item which temporarily resides on our desk? The answers to these questions involve any routine

housekeeping we undertake to maintain the integrity of the information kept in the cabinet.

Now consider the disk used to store information (and instructions) as an electronic file cabinet. Just like a regular file cabinet, this electronic one can become full, get out of order, or lack information that is stored elsewhere.

Clearly, we need a few housekeeping routines to maintain that electronic cabinet, our disks. These routines must perform the following housekeeping tasks:

- Ascertain the current status of a disk
- Rearrange the information on the disk
- Expand the filing capabilities.

But you may remember we already have some commands that perform these tasks. Thus, we could categorize the commands in Chapter 2 by function as shown in the following table:

Command	Use		
	Status	Rearranging	Expansion
DIR	x		
ERA		x	
REN		x	
SAVE		x	x
TYPE	x		
USER		x	

However, these commands are not sufficient to alter a disk or to do major reordering. For instance, while you can see which files are on the diskette, the built-in CP/M commands cannot determine the size of those files.

Fortunately, Digital Research provides a number of housekeeping utilities to help you keep your diskettes in order. These housekeeping utilities include the following transient programs:

Program	CP/M-80	CP/M-86
STAT	STAT.COM	STAT.CMD
PIP	PIP.COM	PIP.CMD
ED	ED.COM	ED.CMD
DUMP	DUMP.COM	DUMP.CMD
SYSGEN	SYSGEN.COM	LDCOPY.CMD
MOVCPM	MOVCPM.COM	n/a

As with the built-in commands, the transient housekeeping commands can be categorized by function.

Command	Use		
	Status	Rearranging	Expansion
STAT	x	x	x
PIP		x	x
ED		x	x
DUMP	x		
SYSGEN		x	x
LDCOPY		x	x
MOVCPM		x	x

SYSGEN, LDCOPY, and MOVCPM are described in Chapter 5. STAT, PIP, ED, and DUMP are described in this chapter.

The description of STAT is divided into two sections because of the two distinct uses of this program.

The Statistics on Files section tells how STAT is used in the housekeeping of files. The Statistics on Devices section tells how STAT is used in the housekeeping of devices. Devices were introduced in Chapter 1 and are further explained in the second STAT section.

As with STAT, PIP coverage is divided into separate sections. These three sections reflect the natural differences in the data sources and destinations that can be connected by using PIP.

STAT—STATISTICS ON FILES

STAT, short for STATISTICS, provides information about a file or a group of files on a disk. Uses of STAT relating to devices other than the disk drive are described following this section on files.

STAT commands either provide information on the size and attributes of a file or files on a disk, or change the attributes of a file or files on a disk. The command line parameters specify which files are to be included. *File size* refers to the amount of space (in bytes) occupied by a file. *Free space* refers to the amount of unused space (in bytes) still available on a disk for storage of files. *File* or *disk attributes* are a specific set of characteristics we can assign to a file or a diskette.

Introduction to STAT And Its Terminology

The size of a file or the amount of storage that is unused on a disk is always reported by STAT in bytes. As we discussed in Chapter 1, a byte is a memory unit capable of storing a single character. Also recall that 1K byte equals 1024 bytes, a quantity which is universally, but incorrectly, called a kilobyte.

If STAT tells us we have 34K of space remaining on our disk, we can store an additional 34,816 (1024 times 34) bytes of information on that disk.

Experience is the best guide for interpreting what STAT tells you about your files. If you use STAT often and compare its reports, you will develop a sense of what 34K represents. As a reference, 34K bytes is equivalent to 10 pages of single-spaced, typewritten material (assuming 65 characters per line and 54 lines per page).

Users of CP/M-80 version 2.2 and CP/M-86 may assign two pairs of file attributes: R/O or R/W, and DIR or SYS.

R/O Read-only file.
You may not update an R/O file or erase it by using the ERA command. You may not write to or enter information on an R/O file. R/O status protects against accidental erasures of valuable files.

R/W Read/Write file.
You may update or erase an R/W file; it is not protected unless the diskette is write-protected. This is the normal attribute of a file and is the default attribute (you do not have to set it explicitly).

SYS System file.
A SYS file does not appear in the display of a disk directory. You may retrieve information from a SYS file, store new information in it, erase it, or use it in any other normal manner, but it will not appear in any directory. However, it will appear when you use STAT to review the status of files. SYS files are available to all user areas in CP/M-80 version 2.2, but only SYS files in user area 0 will be available to other user areas in later CP/M-80 versions.

DIR Directory file.
A DIR file appears in all directory displays for the current user area. This is the normal attribute given to a file.

The members of each pair of attributes are mutually exclusive; files cannot be both R/W and R/O, nor can they be both SYS and DIR.

If you try to save information on a R/O disk or write on an R/O file, the error message BDOS ERR ON d: R/O will appear (where d: is the drive identifier). Even if a disk is not write-protected, it still appears as an R/O type using STAT. This means that you exchanged diskettes without telling CP/M, but CP/M recognized the swap. To restore the new diskette to R/W status, simply type a CONTROL-C.

If a diskette is protected with the write-protect notch covered, it still appears as R/W in the STAT report because CP/M has no way of detecting the physical status of the write-protect notch until you actually attempt to write on the diskette.

You use the information STAT gives you on file size, file attributes, and disk space to perform the following functions:

- Examine how much space is left on a diskette
- Examine how much space is occupied by a group of files
- Examine how much space is occupied by a single file

- Assign R/O or R/W attributes to a group of files
- Assign R/O or R/W attributes to a single file
- Assign DIR or SYS attributes to a group of files
- Assign DIR or SYS attributes to a single file.

To determine which STAT command line to enter for each of the uses listed above, read the next few pages carefully. Examine each command line, its purpose, and the computer display results. Try these examples at your computer. Try to generate the error messages. Consider how you might use the information displayed, given the uses of STAT we have just listed.

Using STAT on Files

STAT <cr>

Display the attribute and the amount of free space left on the disks accessed since the last cold or warm start.

Examples

```
A>STAT<cr>                Command line
A: R/W, SPACE: 2K         STAT's report

A>STAT<cr>                Command line
A: R/W, SPACE: 2K         STAT's report
B: R/O, SPACE: 120K
A>
```

(CP/M-80 version 1.3 only reports the status of the currently logged disk drive.)
Note that in the above examples the write-protection status of the disk is reported first, then the amount of free space remaining is reported.

Special Input

None

STAT d: <cr>

Display the amount of free space available on the disk in drive d:.

Example

```
A>STAT B:<cr>             Command line
BYTES REMAINING ON
B: 170K                   STAT's report
A>
```

You ask for the statistics on the second drive, drive B:.

Special Input

None

STAT d:filename.typ <cr>

Display the amount of space occupied by the file(s) filename.typ on drive d:. The file name and extension may contain the CP/M ambiguous characters (* and ?). The drive identifier is optional; if omitted, the current drive is assumed.

Examples

```
A>STAT JOAN.ARC<cr>

 RECS  BYTS  EX  ACC   D:FILENAME.TYP
   5    2K   1  R/W   A:JOAN.ARC


A>STAT B:JOAN.ARC<cr>

 RECS  BYTES EX  ACC   D:FILENAME.TYP
  10    4K   1  R/W   B:JOAN.ARC


A>STAT *.COM<cr>

 RECS  BYTES EX  ACC   D:FILENAME.TYP
   4    2K   1  R/W   A:DUMP.COM
  48    6K   1  R/W   A:(ED.COM)
  56    8K   1  R/O   A:PIP.COM
  24    4K   1  R/W   A:STAT.COM
  10    2K   1  R/W   A:SUBMIT.COM
BYTES REMAINING ON A: 218K


A>STAT B:EXAMPLE.?X?<cr>

 RECS  BYTES EX  ACC   D:FILENAME.TYP
  48    6K   1  R/W   B:EXAMPLE.TXT
  24    4K   1  R/W   B:EXAMPLE.EXT

 A>
```

Special Input

None

In the above examples some new terminology was introduced in the headings STAT printed out.

RECS
: The number of 128-byte records used by the file. CP/M stores information in 128-byte units.

BYTES
: The length of the file in bytes. 2K represents 2048 characters of information; a kilobyte is 1024 bytes since computers use the binary numbering system.

EX
: The number of physical extents occupied by the file. Extents are another part of the way CP/M maintains files, but most users can ignore this.

ACC
: The file access attribute, R/W or R/O. CP/M-80 versions 1.4 and earlier did not include this function.

D:FILENAME.TYP
: The heading for the drive and file name column on the display. The heading is absent for CP/M-80 version 2.2 and newer.

A:(ED.COM)
: Parentheses surrounding a file indicate that the SYS file attribute has been set. The SYS file attribute is only available on CP/M-80 version 2.2 and newer.

STAT d:filename.typ $atr <cr>

Assign the attribute atr (which can be R/O, R/W, DIR, or SYS and must be preceded by a dollar sign) to the file(s) filename.typ on drive d:. Filename.typ may use the wildcard specifiers, * and ?. The d: is optional; if omitted, the current logged drive is assumed.

Examples

```
A>STAT BROTHER.AND $R/O<cr>
BROTHER.AND SET TO R/O
A>

A>STAT B:BROTHER.AND $SYS<cr>
B:BROTHER.AND SET TO SYS
A>

A>STAT *.* $R/W<cr>
BROTHER.AND SET TO R/W
SISTER.TOO SET TO R/W
A>
```

Special Input

None

STAT—STATISTIC ON DEVICES

STAT also provides information about CP/M's physical and logical devices. In Chapter 1 we briefly introduced physical units as the devices you may choose to make up your microcomputer system. A logical device denotes a general function of your microcomputer, while a physical device is the specific piece of equipment you choose to perform that function. CP/M requires us to select a physical device to perform the function of each logical device. We convey our choices to the computer by using the STAT (on devices) command.

There are four logical devices in CP/M.

CON:
Enter commands and display information; operator console function
RDR:
Receive information; paper tape reader function
PUN:
Send information; paper tape punch function
LST:
List (print) information; list function.

There are 12 physical devices possible.

TTY:
Slow console display (teletypewriter)
CRT:
Fast console display (cathode ray tube)
BAT:
Batch processor
UC1:
User defined console
PTR:
Paper tape reader
PTP:
Paper tape punch
UR1:
User reader #1
UR2:
User reader #2
UP1:
User punch #1
UP2:
User punch #2
LPT:
Line printer
UL1:
User list device

NOTE: The colon in each of the logical and physical device names is required in all references to these devices. In other words, RDR: is the proper device name for the logical reader device.

Sixteen physical-to-logical device assignments are possible.

Logical		Physical
CON:	May be performed by	TTY:, CRT:, BAT:, or UC1:
RDR:	May be performed by	TTY:, PTR:, UR1:, or UR2:

PUN:	May be performed by	TTY:, PTP:, UP1:, or UP2:
LST:	May be performed by	TTY:, CRT:, LPT:, or UL1:

Your microcomputer might be programmed for physical devices other than the default TTY:. This programming was most likely done by the vendor who sold you your CP/M. If you have two printers, one might be programmed to be the LPT: device and the other as the UL1: device. Once made, such arrangements are generally constant; they reflect the distinct programming required to connect each device to the microcomputer. In our example, both LPT: and UL1: are physical devices that can perform the function of the logical device LST:.

Unfortunately, most computer manufacturers have not followed the original recommendations of Digital Research in assigning physical devices, and Digital Research has steadfastly maintained the original device names, despite major changes in the technology of terminals, modems, and printers. Here is the normal, and confusing, setup of a typical microcomputer.

Logical	Physical	Actual Device
CON:	TTY:	High-speed video terminal
RDR:	PTR:	Modem receive line
PUN:	PTP:	Modem send line
LST:	LPT:	Printer

Notice that the information in this table does not match the descriptions we gave for each device above (for example, TTY: is supposed to be a slow console device, not a high-speed video terminal). Paper tape readers and punch devices are not widely used anymore, and the abbreviations PUN:, RDR:, PTR:, and PTP: reflect an earlier time when paper tape was commonly used. Do not be disturbed if your device assignments do not exactly match the CP/M definitions.

When your computer is first started, the default values of the device assignments are made. These default assignments vary from system to system and are programmed into your CP/M by your vendor. The default values should be the assignments you most often use. A cold start changes the device assignments to their default value, but a warm start does not affect the assignments in any way.

We use STAT on devices

- To learn the current device assignments
- To learn the possible device assignments
- To assign physical devices to logical devices.

A few STAT on devices command lines request information on the disk drives. Specifically, we use these special command lines

- To learn the current status of a drive
- To learn the current status of user areas on the disk
- To protect a disk from accidental "writes."

To determine which STAT on devices command line to enter for each of the uses listed above, read the following pages carefully. Examine each command line, its purpose, and the computer display results. Consider how you might use the information displayed, given the uses of STAT on devices we have just listed.

STAT DEV: <cr>

Display the current device assignments.

Example

```
A>STAT DEV:<cr>
CON: IS CRT:
RDR: IS UR1:
PUN: IS UP2:
LST: IS UL1:
A>
```

Notice that the left-hand entry reflects the logical device; the right-hand entry is the current physical device assigned to it.

Special Input

None

STAT VAL: <cr>

Display the possible assignments of physical-to-logical devices and an abbreviated STAT command line summary.

Example

```
A>STAT VAL:<cr>
TEMP R/O DISK: D: = R/O
SET INDICATOR: D:FILENAME.TYP $R/O $R/W $SYS $DIR
DISK STATUS  : DSK: D:DSK:
USER STATUS  : USR:
IOBYTE ASSIGN:
CON: = TTY:CRT:BAT:UC1:
RDR: = TTY:PTR:UR1:UR2:
PUN: = TTY:PTP:UP1:UP2:
LST: = TTY:CRT:LPT:UL1:
```

NOTE: Only the last four lines appear if you are using CP/M-80 version 1.4 or version 1.3.

The first several lines STAT presents tell you the possible STAT device commands. They are

`STAT D: = R/O`	To make a temporary receive-only disk
`STAT D:FILENAME.TYP R $ATR`	To set an attribute

STAT D:DSK:	To see the statistics on a disk drive
STAT D:USR:	To see the statistics on user area use

Special Input

None

STAT log: = phy: <cr>

Assign the physical device specified to the logical device specified.

Example

```
A>STAT CON:=CRT:,LST:=UL1:<cr>
A>
```

Notice that several device settings can be made by separating each with a comma. In addition, note that CP/M does not issue any message to inform you that the assignment has been made. If your system no longer works after you make a new device assignment, it is possible you have assigned the console function to a non-existent device.

Special Input

None

STAT USR: <cr>

Display the current user number and list all user numbers for which there are files.

Example

```
A>STAT USR:<cr>
ACTIVE USER: 0
ACTIVE FILES: 0 1 3
A>
```

The above message indicates that you are currently in user area 0, but you have files on your disk in areas 0, 1, and 3. This function is not available on CP/M-80 version 1.3 or 1.4.

Special Input

None

STAT d:DSK: <cr>

Display information on how data is stored on the disk in drive d:.

Example

```
A>STAT B:DSK:<cr>
B: DRIVE CHARACTERISTICS
    4096: 128 BYTE RECORD CAPACITY
     512: KILOBYTE DRIVE CAPACITY
```

```
      128: 32 BYTE DIRECTORY ENTRIES
      128: CHECKED DIRECTORY ENTRIES
      128: RECORDS/EXTENT
       16: RECORDS/BLOCK
       58: SECTORS/TRACK
        2: RESERVED TRACKS
A>
```

The d: (drive identifier) is optional; if omitted, information is displayed for all disks accessed since the last cold or warm start.

The most relevant information in the display is "KILOBYTE DRIVE CAPACITY," which is the amount of free space on an empty disk, and "32 BYTE DIRECTORY ENTRIES," which is the maximum number of files you can store on the disk (this number will be reduced if you have any files which are larger than one extent).

The information in the STAT d:DSK: display will not change unless you modify CP/M. This static information reflects design decisions about how to arrange data on the disks.

The following definitions explain each of the STAT entries presented to you on the CRT:

128 BYTE RECORD CAPACITY
: The maximum number of 128-byte records you may store on the disk.

KILOBYTE DRIVE CAPACITY
: The maximum number of Kbytes you may store on the disk.

32 BYTE DIRECTORY ENTRIES
: The maximum number of files you may store on the disk.

CHECKED DIRECTORY ENTRIES
: Usually the same as "32 BYTE DIRECTORY ENTRIES" for drives with removable media (diskettes); usually "0" for drives with non-removable media (sealed hard disks).

RECORDS/EXTENT
: The maximum number of records per directory entry.

RECORDS/BLOCK
: The minimum amount of disk space which may be allocated to a file.

SECTORS/TRACK
: The number of sectors into which a track is divided (*see* Chapter 1).

RESERVED TRACKS
: The number of disk tracks not available for storage of files.

Special Input

None

STAT d: = R/O <cr>

Assign a temporary write-protect (read-only) status to the disk in drive d:.

Example

```
A>STAT B:=R/O<cr>
A>SAVE 2 B:JUNK.FIL
BDOS ERR ON B: R/O
```

In the above example we first set drive B: to read-only status using STAT. Note that STAT does not provide any confirmation of our request. Next, we attempted to save a new file on the disk and received a BDOS ERR message indicating that CP/M considers the disk to be write-protected. A warm or cold start cancels the temporary R/O status.

Special Input

Any subsequent warm or cold start cancels the temporary R/O status.

PIP—COPYING INFORMATION

PIP, Peripheral Interchange Program, copies information from one place to another. The names of the files you wish to copy can be ambiguous, and a number of optional parameters are available for use. PIP can also copy information from and to devices.

PIP is invoked in one of two ways. If you want to perform only one operation with PIP and it will not be necessary to swap the diskette in A: use the following:

```
A>PIP pipcommandline<cr>
```

If you have several operations to perform, use the following second method to invoke PIP:

```
A>PIP<cr>                    CP/M command line
*pipcommandline<cr>          PIP command line
  .
  .                          Any number of PIP commands
  .
*pipcommandline<cr>          PIP command line
*^C                          ^C returns you to CP/M
A>
```

No matter which method you use, there is no significant difference in the manner in which PIP performs the operation requested by pipcommandline. A minor difference is explained under the "Q" parameter later in this PIP section. Another minor difference is in PIP's response to errors. If the first invocation method is used, errors cause a return to CP/M. If the second method is used, errors cause a return to the PIP prompt.

The PIP command line ("pipcommandline" in our examples) usually takes the following form:

destination = source [parameters]

where "destination" is the name of the new file, "source" is the name of the old file, and "parameters" are the optional parameters you select for the copying process.

If there is already a file with the same name as your destination file, the existing file is erased after successful completion of the copy. The previous contents of the destination file are lost.

PIP command lines and special parameters specify the transfer made between two files or two groups of files. In every instance, the original file remains intact; we can duplicate an entire disk or only a portion of a file without changing the original. We can specify that certain characteristics of the new file be changed from the original, without changing the original. The following are some uses of PIP (files):

- Copy a file from one disk to another
- Create an identical file with a different name
- Copy several files from one disk to another
- Copy all files from one disk to another
- Create one file from the concatenation of several others
- Copy a portion of a file
- Copy from a system file
- Copy onto a R/O file
- Display the contents of the copied file in progress.

PIP also copies data from a file to a device, from a device to a file, or between devices; you simply substitute a device name for a file name in either the destination or source position, or both.

When you read from or write to a device instead of a disk file, make sure you can terminate the copying process. Some devices do not emit an end-of-file marker (^Z) as CP/M does at the end of every disk file. The following are some uses of PIP (devices):

- Send the contents of a file to a device, such as a printer
- Pass the data from a device, such as a paper tape reader or modem, into a file
- Send data from one device to another
- Print the contents of a file, formatting it for a special printer or paper size
- Display on the console selected data arriving at an input device
- Convert upper-case letters to lower-case ones, or vice versa
- Save in a file all the data arriving at an input device.

PIP has five "special" device names in addition to the ones mentioned in the section on STAT.

NUL:
A source device which sends 40 nulls ("do-nothing" characters, for example, 00 hex) to the destination specified.

EOF:
A source device which sends an end-of-file marker (^Z, or 1A hex) to the destination.

OUT:
A user-created custom destination device. PIP must be modified to include this device.

INP:
A user-created custom source device. PIP must be modified to include this device.

PRN:
A special form of the LST: device which expands tabs, numbers lines, and paginates the copy (the same as LST: [NPT8]).

As with the devices discussed earlier, these device names are used as sources and destinations in the command line.

Examine the following pages. Each PIP command line parameter and option is illustrated with an explanation of the computer display. Explore the possible combination of PIP command lines, parameters, and options and the resulting variations on the uses we have listed.

PIP <cr>

Load PIP into memory.

Example

```
A>PIP<cr>          Command line
*                  PIP prompt
```

When you see the PIP prompt (*) you may enter a valid PIP command line, or press CARRIAGE RETURN or ^C to return to CP/M.

Special Input

A CARRIAGE RETURN or ^C cancels PIP and returns you to CP/M.

PIP d:new.typ=d:old.typ[p]<cr>

Copy the file old.typ on the drive specified to the file new.typ on the drive specified using parameters ([p]).

Examples

```
B>PIP LETTER2.DOC=LETTER1.DOC<cr>
B>
```

Simple use of the PIP command to copy a file into a new file with a new name.

```
B>PIP A:LETTER2.DOC=B:LETTER1.DOC<cr>
B>
```

An example of using PIP to copy between disks A: and B:.

```
B>PIP LETTER2.DOC=LETTER1.DOC[V]<cr>
B>
```

An example of the use of a parameter, in this case, the "V" (for Verify) parameter.

```
A>PIP B:=A:DOCUMENT.LET[V]<cr>
A>

A>PIP B:DOCUMENT.LET=A:[V]<cr>
A>
```

Two examples of using shorthand methods of specifying a file name when the same name will be used on both drives. Either side of the equal sign may be abbreviated to just the drive identifier.

```
A>PIP B:=A:*.*[V]<cr>
COPYING-
DOC.ONE
DOC.TWO
LET.ONE
A>
```

An example which uses both the abbreviation (B:=) method just shown plus the use of ambiguous file references (A:*.*). When ambiguous file references are used, PIP reports the names of the files which match your request as it copies them.

Special Input

None

PIP d:new.typ=d:old1.fil[p],d:old2.fil[p]<cr>

Create a new file on the drive specified, the new file consisting of both files old1.typ and old2.typ. In other words, a comma on the PIP command line indicates concatenation.

Example

```
A>PIP B:DOC=A:JUNE[V],A:JULY<cr>
A>
```

This example would combine the data in JUNE and JULY (in that order) into a new file named DOC.

Special Input

None

PIP dev:=d:filename.typ[p]<cr>

Send the contents of a file on the drive specified to a device.

Example

```
B>PIP LST:=B:LAFFERTY.RT<cr>
B>
```

The above example would send a copy of the file LAFFERTY.RT to the LST: device.

NOTE: You may not use ambiguous file references to send a series of files to the LST: device.

Acceptable destination devices are

CON: PUN: LST:	Logical
TTY: PTP: LPT: CRT:	Physical
UP1: UP2: UL1: UL2:	
OUT: PRN:	Special

Special Input

You may stop the PIP to a device process by pressing any key while the copying is in progress.

PIP d:filename.typ=dev:[p]<cr>

Copy input from a device to the file name specified.

Example

```
B>PIP  B:DOCUMENT.MAY=CON: <cr>
I AM NOW TYPING FROM CONSOLE.^Z
B>
```

The above example creates a file from information typed at the console device (up to and including the terminating ^Z).

Acceptable source devices are

CON: RDR:	Logical
TTY: PTR: CRT: UR1:	Physical
UR2: UC1:	
NUL: EOF: INP:	Special

Special Input

Both a CARRIAGE RETURN (^M) and a LINE FEED (^J) must be at the end of each line. ^Z terminates the transfer and signifies the end the file.

PIP destinationdev:=sourcedev:[p]<cr>

Copy data from one device to another.

Example

```
A>PIP CON:=PTR:[U]<cr>
THIS IS A PAPER TAPE READING DATA
INTO THE SYSTEM, WHICH DISPLAYS IT
ON THE DESTINATION DEVICE, THE CONSOLE.
A>
```

In the above example, we asked that the data being read from the paper tape reader be copied to the console display. In addition, we specified the optional parameter of converting information to all upper-case characters ([U]).

PIP Parameters

This section describes each of PIP's optional parameters. You probably will only use one or two of these parameters frequently, and you may never use some of them. You may combine two or more parameters between the bracket characters (for example, [BEU]).

[B]

Block transfer parameter. The [B] parameter designates the "block mode transfer." In block mode, PIP transfers information from the source to a buffer until the ASCII character DC3 (53 hex, or ^S, sometimes called XOFF or reader off) is received. Once this character is received, PIP takes the information it has received so far and sends it to the destination device specified. Then PIP returns to the source for more data. We use [B] for those source devices that transfer data continuously and would otherwise fill up PIP's buffer area (where it stores information before processing to the specified file or device) if the buffer were not periodically purged. For normal file-to-file operations, [B] is not necessary.

The [B] parameter is normally used to read a paper tape into a file. Whatever the source device, it must send the DC3 character often enough to avoid filling the buffer and then waiting for PIP to empty the buffer.

The [B] parameter is not used on CP/M-86 system.

[D#]

Delete all characters after the #th column. The [D#] parameter tells PIP to delete any character received that extends past a column number. This number must be specified immediately following the "D," as in [D20], for example. This type of transfer will work only for line-oriented data (data that contains periodic CARRIAGE RETURN characters). After each CARRIAGE RETURN is detected, PIP counts and processes characters up to and including the column number specified and then ignores all remaining characters sent until the next CARRIAGE RETURN is received.

(NOTE: Line feed characters are interpreted the same as CARRIAGE RETURN for this operation.)

The [D#] parameter is primarily used to send wide-lined output to a device that handles only narrow lines (like a printer or CRT terminal). For file-to-file transfers, this option is rarely used.

The value "#" is a decimal integer in the range 1 through 255, inclusive. Input lines longer than 255 characters are not truncated properly by the [D#] option.

[E]

Echo the copying to the console as it is performed. The characters sent to the destination device are echoed to the console display during the copying process. This is a handy method of reviewing exactly what is being copied.

[F]

Filter form feeds from the original file. The [F] option "filters" form feed characters (0C hex) from the data flow; PIP ignores the ASCII-defined character for form feeds and copies without it. Form feeds are often embedded in files to correctly paginate the printed output. You may wish to eliminate form feeds to conserve paper, to display a file on your console screen, or to use a printer that improperly interprets the form feed character.

The ASCII form feed character is called "FF" and is equivalent to hexadecimal 0C and CONTROL-L.

The form feed character controls the layout or positioning of information in printing or display devices. Thus, a printer would normally respond to this character not by printing it but by advancing the paper to the top of the next form or page.

Use [F] if your printer does not respond properly to form feed characters or if you are sending a file to a device other than a printer. See also the [P#] parameter.

[G#]

Direct PIP to copy files from other user areas. The [G#] option allows PIP to copy files from another user area to the current one. A decimal number (between 0 and 15, inclusive) represents the source user area and should immediately follow the "G" in the command line. This option is helpful in setting up a user area with files that may be needed later. The [G#] parameter is not available in CP/M-80 version 1.3 or 1.4.

[H]

Check data transfer for proper Intel hex format. The [H] parameter specifies that the data to be transferred be in the special Intel hex format rather than in the normal ASCII or binary formats. The Intel format is usually used with paper tape punch and reader devices. Most users do not utilize this option.

When PIP detects errors in the hex format, a prompt will be displayed asking what corrective action is to be taken. See also the [I] parameter.

[I]

Ignore any null records in Intel hex format transfers. The [I] option also applies to Intel hex format records (see [H]). The [I] parameter tells the PIP program to ignore any data that appears in a null record ("00:" in Intel format). Again, this option is rarely of use to end users, as it applies primarily to paper tape reader and punch devices.

If the [I] option is specified, then the [H] option is automatically set by PIP. Thus, the PIP command line of

```
PUN:=PROGRAM.HEX[I]
```

is equivalent to

```
PUN:=PROGRAM.HEX[HI]
```

[L]

Convert upper-case letters to lower-case. The [L] option allows you to change the letters "A" through "Z" to their lower-case equivalents during the copying process.

Be extremely careful in using this option. It should never be used with files which contain instructions to the computer. If you use it on a program file, you may change important instructions which PIP recognizes as upper-case letters only.

[N]

Add line numbers to each line during the data transfer. The [N] option adds a line number to each line of data transferred. Each time a CARRIAGE RETURN (or LINE FEED) is detected by PIP, the line number counter is incremented. [N] is of particular use when transferring a file to the printer. If a "2" is appended to the [N] option ([N2]), leading zeroes are added and the number is printed in a six-character field followed by a three-character blank buffer. Here is the difference between files transferred using each of these options.

Original	**File Using [N]**	**File Using [N2]**	
Now is the	1: Now is the	000001	Now is the
time for all	2: time for all	000002	time for all
good persons	3: good persons	000003	good persons
to come to	4: to come to	000004	to come to
the aid of	5: the aid of	000005	the aid of
their party.	6: their party.	000006	their party.

[O]

Transfer object code files, or other non-ASCII files. Use the [O] parameter when copying from non-ASCII files (such as program, object code, or binary data files) or from devices sending non-ASCII data.

Specifying the [O] parameter is not necessary when copying from files with the ".COM" type, because PIP assumes that ".COM" files are non-ASCII (non-text) files.

Use of [O] tells PIP to treat CONTROL-Z (1A hex) like any other character it receives from a source device or file; PIP would otherwise interpret CONTROL-Z as signaling the end of the transfer from that source.

The following paragraphs contain more detailed information about ASCII files, non-ASCII files, and end-of-file markers. Read on if you are interested in these details.

PIP assumes that a CONTROL-Z character received from an ASCII source (device or file) is an end-of-file marker or data terminator which signals that all the data has been transferred.

PIP sends a CONTROL-Z character to an ASCII destination (file or device) to indicate that all the data has been sent.

The difference in handling non-ASCII data is necessary because CP/M uses different methods for marking (and detecting) the end of ASCII and non-ASCII files.

CP/M marks the end of an ASCII file by placing a CONTROL-Z character in the file after the last data character. If the file contains an exact multiple of 128 characters, in which case adding the CONTROL-Z would waste 127 characters, CP/M does not do so. Use of the CONTROL-Z character as the end-of-file marker is possible because CONTROL-Z is seldom used as data in ASCII files.

In a non-ASCII file, however, CONTROL-Z is just as likely to occur as any other character. Therefore, it cannot be used as the end-of-file marker. CP/M uses a different method to mark the end of a non-ASCII file. CP/M assumes it has reached the end of the file when it has read the last record (basic unit of disk space) allocated to the file. The disk directory entry for each file contains a list of the disk records allocated to that file. This method relies on the size of the file, rather than its content, to locate the end of the file.

[P#]

Issue a form feed after the #th line. Use the [P#] parameter when the destination is the LST: device and the current LST: device (usually a printer) does not print the source data in the following format:

1. A blank top margin
2. A blank bottom margin
3. Top and bottom margins totaling six lines
4. Text completely filling the space between top and bottom margins.

Deviation from these conditions usually occurs when the current LST: device

1. Does not understand the form feed character (see [F]),
2. Does not insert top and bottom margins, or
3. Is printing on a different length page than that assumed by the source data.

The [P#] option tells PIP you want it to insert a form feed after each number of lines of data. Enter the desired number of lines in place of "#" in the PIP command

line parameter [P#]. The character "#" represents a decimal number in the range 1 through 255, inclusive. If you do not enter a number or if the number is 1, PIP assumes you want a page eject after each 60 lines.

If [F] is also present in the parameter field, all form feeds detected during the transfer will be removed and PIP will then insert new form feeds as specified by [P#]. In other words, [F] and [P#] used together will override the pagination implied by the presence of form feed characters in the source data.

A printer that understands the form feed character responds to it by immediately advancing the paper to the top of the next page, whether or not the current page is full. Printers which do not understand the form feed character may perform erratically.

See the section on the PIP PRN: device.

[Qstring^Z]

Copy a portion of the file up to the string listed. Use of the [Q] parameter allows you to give PIP a set of characters to look for and terminate the copying of data when they are encountered. Select the terminating string of characters carefully; they must be unique to ensure you are copying the exact portion of the file desired. "Q" marks the beginning of the string, and ^Z marks the end of the string: [Qstring^Z].

The string following the "Q" parameter is converted to upper-case if the command line is typed after the CP/M prompt (A>). If the command is typed after the PIP prompt (*), then the automatic conversion does not occur. To include lower-case letters in the string, your command line must follow the PIP prompt. See also the [S] parameter.

[R]

Allow PIP to copy a system file. Only by appending the command line with [R] can we copy system files. The system file attribute, if present, is copied. This parameter does not apply to CP/M-80 versions 1.3 and 1.4.

[Sstring^Z]

Copy the portion of data beginning with string. Just like the [Q] option, the [S] option specifies that you wish to copy only a portion of a file using PIP, beginning at the point where PIP detects the string of characters specified immediately following the "S," where "string" is the sequence of characters to search for and ^Z is a CONTROL-Z used to terminate the string. Both the [S] and [Q] options may be present on the same command line.

See the important explanation of automatic lower-case to upper-case conversion under the [Q] option.

[T#]

Set tab stops every #th column. The [T#] parameter tells PIP to expand any TAB character (09 hex or CONTROL-I) it detects to the number of spaces necessary to

reach the next tab stop. This is useful because some editor programs do not save TABs as spaces, but as TAB characters, and not all printers or terminals are set up to detect TAB characters and expand them to the number of spaces they represent. Use [T#] to change the standard tabbing from eight characters (or whatever number your console or printer defaults to) to a number you define.

[U]

Translate lower-case characters to upper-case. The [U] option allows you to change the letters "A" through "Z" to their upper-case equivalents during the copying process.

[V]

Verify the copy is correct by comparing the memory buffer with the newly created file(s). The destination must be a disk file; if it is not, [V] is ignored. After the transfer is complete, the destination is reread and compared to the data in PIP's memory buffer. This means that the [V] parameter will catch errors in writing the file but not in reading it.

When concatenating several sources, the [V] parameter must follow the *first* source name; only one [V] is necessary.

> NOTE: Gary Kildall, creator of CP/M, never uses the [V] parameter, and members of the Digital Research staff indicate that in four years they have yet to see a verify error that is not accompanied by a BDOS ERR ON d: BAD SECTOR error. Therefore, it is doubtful whether the [V] parameter will prove effective in catching otherwise unseen errors.

[W]

Allow PIP to copy into a file with the R/O attribute. Since [W] permits you to write to a R/O file, verifying this attribute is not required before proceeding.

If you attempt to write to a R/O file without using the [W] attribute, you would receive the message

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

The [W] parameter is not available in CP/M-80 versions 1.3 and 1.4.

[Z]

Zero the parity bit during the transfer. Use the [Z] parameter to set the unused eighth bit of a character to zero when receiving ASCII characters from a device.

Each ASCII character uses seven of the eight bits which are processed by the computer. The eighth bit is sometimes called the parity bit. To avoid mysterious problems when processing ASCII data, it is usually wise to set the unused bit to zero.

Use of the [Z] parameter with files created by WordStar results in all "soft spaces" and "soft carriage returns" (spaces and carriage returns inserted by WordStar to format text) being converted to real spaces and returns. Depending upon your use,

using [Z] on a WordStar-created file can either have beneficial or detrimental effects.

Special PIP Devices

EOF:

Send an end-of-file marker to the destination device. When copying between two ASCII-type disk files, an end-of-file marker is automatically sent to the destination device. In some special instances, EOF: can be used to terminate the transfer. When sending files between two computers it is often necessary to terminate the transfer by sending the EOF: device character.

NUL:

Send 40 null (do-nothing) characters to the destination device. NUL: was originally used to terminate punched output with a blank section of paper tape; it provided a header or trailer to the actual information, which was useful for threading the tape. NUL: will produce a four-inch length of blank paper tape.

The ASCII null character is called "Null" and is equivalent to 00 hex, or CONTROL-@.

PRN:

Send data to LST: device with special instructions. The PRN: device specifies additional instructions to the LST: device. It can contain instructions that expand tabs, number lines, or paginate the copy.

Tab stops are assumed at every eighth column. Every time a TAB character is detected, PIP inserts spaces instead. All lines are numbered (beginning at 1 and continuous throughout the transfer), and page ejects are sent to the printer every 60 lines (giving you a margin of three lines at the top and bottom of a normal 11-inch piece of paper). PRN: is used for program development and for listing the contents of an ASCII-type file; the line numbers provide recognizable reference points within the file.

PRN: is equivalent to [NPT8].

INP: and OUT:

Special device drivers for PIP. The device INP: retrieves information from a special user-created PIP input source. OUT: sends information to a special user-created output destination.

You can add special input and output routines (in machine language) to PIP. Hexadecimal locations 103, 104, and 105 are reserved for a jump instruction to your special input routine. PIP calls location 103 hex to input a character and, upon return, expects to find the character in location 109 hex. Hexadecimal locations 106, 107, and 108 are reserved for the jump to your special output routine. PIP loads Register C with the character to transmit and then calls location 106 hex. In

addition, the area from 10A to 1FF hex is free for you to insert your routines. It is rare for a user to use this option.

ED—CONTEXT EDITOR

ED is a program with a number of built-in commands used to edit text files.

You may have wondered how you put something new onto a disk. The answer is that you use an editor. CP/M includes an editor which you will find saved in the file ED.COM.

The editor is a program that takes characters from the keyboard and puts them in a disk file. Since you might make entry errors or wish to make changes, the editor also includes a number of built-in commands that display, modify, delete, and add to the information you have typed.

The CP/M editor (called Context Editor by Digital Research) is both character- and line-oriented; commands operate on text either a character at a time or a line at a time. A line is a block of characters ending in a CARRIAGE RETURN.

To edit the file filename.typ, type the following:

```
A>ED filename.typ<cr>
```

Once the editor program is loaded into memory, it creates a new file with the name filename and the type "$$$" and waits for your command. You may now use the built-in commands of ED to

- Delete or change any part of the existing file, or
- Insert new information into the file from the keyboard or from another file.

For example, suppose the file DOCUMENT.MAY requires editing. To invoke ED type

```
A>ED DOCUMENT.MAY<cr>
```

ED creates the temporary file DOCUMENT.$$$ on the disk. This file contains no information, only a Directory entry for it.

In order to make changes to the original file DOCUMENT.MAY, you must be able to see the text and manipulate it. ED's commands permit you to move text from the original file into memory, to view the text that you have moved into memory, to make the required changes, and to move the text from memory back to the file. You can only edit a file while it is in memory. The area of memory ED sets aside to hold text is called the edit buffer.

As you progress in your editing, you will move more text from the original file into the edit buffer. The edit buffer holds a limited amount of text; as it becomes full you must move the editing text to the temporary file, DOCUMENT.$$$.

When you complete your editing session, some text may remain in the original file DOCUMENT.MAY, some in the edit buffer, and some in the temporary file DOCUMENT.$$$. To end the edit, ED first moves the edit buffer contents to the temporary file, DOCUMENT.$$$. Then ED moves the remaining text from the

original file to the temporary file. Finally the original file, DOCUMENT.MAY, is renamed DOCUMENT.BAK, and DOCUMENT.$$$ is renamed DOCUMENT.MAY.

You now have a copy of the original input in DOCUMENT.BAK, and your newly edited version exists on your disk in the file DOCUMENT.MAY.

Since movement through the entire text may require shifting bodies of text in and out of the edit buffer, you basically move forward through the text (the H command allows you to restart from the head of the file). This will be explained in more detail later. Examine the ED commands on the following pages to learn how to perform the tasks we have described.

ED Commands

To use ED commands, invoke ED, and follow ED's prompt (*) with the proper command and a CARRIAGE RETURN (<cr>), as follows:

```
A>ED d:filename.typ<cr>
*edcommand<cr>
```

We will use several abbreviations in this section. Whenever a plus or minus sign can be typed, you will see the symbol "~", which stands for plus (+) or minus (−); if you type neither "+" or "−", ED assumes you mean "+". Whenever a number can be typed to give the command further information, you will see the symbol "n." This "n" can be any decimal integer number in the range 0 through 65535, inclusive. Do not use a comma between the digits. If you omit the number, ED assumes a value of 1. Typing a zero (0) has a special meaning for some commands. If you type a number sign (#), ED substitutes 65535.

For purposes of ED, a line of text is a sequence of zero or more characters followed by a CARRIAGE RETURN character and a LINE FEED character. We will represent this character pair by the symbol CRLF. When you type a line of text for ED and then press CARRIAGE RETURN, ED adds the LINE FEED character so that the line ends with CRLF.

We can separate ED commands into four functions: transfer text, work with the text in the edit buffer, search and change the text, and combine commands.

Transferring Text.

You can transfer text by line or by blocks

- From the original file into the edit buffer
- From the edit buffer into the temporary file
- From the original file to the temporary file
- From the edit buffer into another file (not the original or the temporary file but a distinct file on a disk)
- From another file (not original or temporary but a distinct file on the disk) into the edit buffer.

Working in the Edit Buffer.

An imaginary character pointer (CP) locates the text in the edit buffer. These editing commands

- Insert text into the edit buffer
- Manipulate text within the edit buffer in relation to the CP
- Direct the movement of the CP.

Searching and Changing Text.

With this function of ED you can work within the edit buffer to

- Search for a particular set of characters
- Substitute a set of characters
- Switch placement between sets of characters
- Move text from the original file through the edit buffer to the temporary file automatically while searching.

Combining Commands.

A string of ED commands can be entered in one command line ending with a <cr> to permit a variety of sequential editing functions and to repeat a command string a specified number of times.

Commands for Transferring Text

#A

Append (copy) a number of lines from the original file to the edit buffer. To move a single line, enter "A" only; 1 is the default number of lines to move. To move the maximum number of lines into the edit buffer, include the number (#A). The #A command moves lines until the original file is exhausted or the edit buffer is full. To move a portion of the text into the edit buffer, enter 0A; this moves text from the original file until the edit buffer is at least half full. Once appended, lines in the original file are ignored by subsequent commands that read from the original file.

nW

Write a number of lines from the edit buffer into a temporary file.

W	writes a single line
#W	writes the entire buffer
0W	writes until the edit buffer is at least half empty

"W" always begins with the first line in the edit buffer and writes it in the temporary file after the line most recently placed there. Lines are deleted from the edit buffer as they are written to the temporary file.

E

End the editing session. Text is transferred and files are renamed as follows:

1. All text remaining in the edit buffer moves to the temporary file, as with the

W command.

2. All text remaining in the original file is then appended to the temporary file.
3. The type of the original file becomes ".BAK."
4. The type of the temporary file is changed to the type of the original file.
5. The block move file X$$$$$$$$LIB, if present, is erased.
6. The CP/M prompt returns.

Use the E command as the normal ending of an editing session. In order to work properly, "E" must be the only command on a line.

H

Move to the beginning of the file being edited (move to Head). Text is transferred and files are renamed as follows:

1. All text remaining in the edit buffer moves to the temporary file, as with the W command.
2. All text remaining in the original file is moved to the temporary file.
3. The type of the original file becomes ".BAK."
4. The type of the temporary file is changed to the type of the original file.
5. A new, empty temporary file is created.
6. You are now ready to edit the new original file.

The H command has two uses.

1. Since the A, N, and W commands can move only forward, not backward, through the original and temporary files, you use the H command to save the editing done so far and return to the beginning of the file for more editing.
2. Use the H command every few minutes during editing of critical files to save your results on the diskette. Text left in the edit buffer is generally lost if an operator error or equipment malfunction occurs, but text saved in the temporary file is easily recovered.

"H" must be the only command on a line. Frequent use of "H" is especially important when inserting a lot of text or making numerous changes. However, remember that the H command makes a new backup file. This means that after using the H command twice the original file as it was before beginning the editing session is lost. For this reason, if you anticipate using the H command often, use PIP to make your own second copy of the original file (do not call it type ".BAK," use ".OLD" or place it on a different drive to be safe).

O

Erase the edited file. Text is transferred as follows:

1. The contents of the edit buffer and the temporary file are deleted.
2. You are returned to the beginning of the original file.

ED asks "O-(Y/N)?" before it proceeds since all changes made to the text are erased when the O command executes. Press "Y" to return to the original file or "N" to continue editing.

"O" must be the only command on the line.

Q

Quit editing; do not institute any changes. ED asks "Q-(Y/N)?" before it proceeds since all changes made to the text are erased when the Q command executes. Press "Y" to quit editing or an "N" to continue the editing session.

"Q" must be the only command on the line.

R<cr>

Reads the file created by the X command into the edit buffer. "R" inserts the entire contents of the transfer file X$$$$$$$.LIB into the edit buffer immediately after the character pointer (CP). The transfer file is not affected; you can read it as many times as you want during an editing session. The transfer file is automatically erased when you leave ED with the E, Q, or C commands.

Rfilename<cr>

Reads the library file filename.LIB into the edit buffer. This command inserts the entire contents of the specified file into the edit buffer immediately after the character pointer (CP). The library file is not affected.

nX

Writes (Xfer, or transfer) lines from the edit buffer to a temporary file named X$$$$$$$.LIB. You may later transfer these lines back to the edit buffer by using the R<cr> command. With the X, R<cr> command combination you can move blocks of information. The file X$$$$$$$.LIB is erased when editing is completed with a E, Q, or C command. The X command copies the "n" lines that follow the character pointer in the edit buffer into the transfer file. The lines are not deleted from the edit buffer, and they are added to any previous contents of the transfer file.

Commands for Working in The Edit Buffer

~B

Moves the character pointer to the beginning (use a "+" or nothing before the "B") or to the end (use a "−" before the "B") of the edit buffer. Notice that the direction implied by the sign in front of the B command is the opposite of that for all other commands.

~nC

Move the character pointer by plus or minus "n" characters. The CARRIAGE RETURN/LINE FEED combination that terminates a line in ED is counted as two

separate characters. Type "+5C" to move the character pointer five characters toward the end of the edit buffer.

~nD

Deletes "n" characters immediately before (−) or after (+) the character pointer.

```
100: NOW IS THIME FOR ACTION          Before
                ^
100:*-3D<cr>                          Command

100: NOW IS THE TIME FOR ACTION       After
                ^
```

(The ^ indicates character pointer.)

I<cr>

Enter the insert mode. ED accepts all characters you type and inserts them into the edit buffer after the character pointer. With certain exceptions, every character you type goes into the buffer until you press CONTROL-Z. CONTROL-Z terminates the insert mode and returns the ED prompt. If you enter an upper-case "I", all inserted text is automatically translated to upper-case characters only.

There are some characters that perform differently in the insert mode than you may expect.

^H or BACKSPACE deletes the last character typed in current CP/M-80 and CP/M-86 versions

^L inserts a CARRIAGE RETURN/LINE FEED

^M or RETURN inserts a CARRIAGE RETURN/LINE FEED

^R redisplays the current line

^U deletes the current line

^X deletes the current line

RUBOUT or DEL deletes the last character typed and redisplays it.

The "current line" consists of the characters which follow the last CARRIAGE RETURN typed while in the insert mode.

Use the insert mode to type in a new file or to add several lines to an existing file. Remember to exit the insert mode and use the save command H often.

Istring^Z

Insert a string. This command inserts the character sequence "string" into the edit buffer following the character pointer. Use this command to insert short strings.

Istring<cr>

Insert a line. This command inserts the character sequence "string" into the edit buffer following the character pointer. A CARRIAGE RETURN/LINE FEED combination is inserted after the string. The CARRIAGE RETURN/LINE FEED is the difference between this command and the previous one. Use this command to insert a single line.

~nK

Kills (deletes) those lines from the edit buffer that you do not wish to appear in the the final edited version. Lines referenced by the nK command are not transferred to temporary file but remain in the original file, which later becomes the backup file. You may remove any number of lines from the edit buffer in either direction from the character pointer. "+K" or "K" erases all characters after the character pointer up to and including the current line's CARRIAGE RETURN/LINE FEED. "−K" erases all characters in the line before the character pointer, back to the first CARRIAGE RETURN/LINE FEED encountered, but not including it. If you still have some characters remaining in a line that you thought you had deleted with the K command, check to see if the character pointer was at the start of the line when you issued the command.

~nL

Moves the character pointer forward or backward "n" lines in the buffer. The character pointer moves to the beginning of the current or next line if it is not currently at the beginning of a line. Subsequent moves encompass full lines; the character pointer moves "n" lines forward or backward. 0L moves the character pointer to the beginning of the current line.

~nP

Moves the character pointer one page and displays the page following the character pointer. This command is repeated for a total of "n" pages. "P" is a convenient method of scrolling through the text in the edit buffer. A page is a fixed number of lines that varies with the version of CP/M, but usually it is one screen full. "+nP" moves you forward through the edit buffer and "−nP" moves you backward. "P" operates by first moving the character pointer backward or forward one page and then displaying the page that follows the character pointer. "0P" displays a page without moving the character pointer.

~nT

To see lines of text that are in the edit buffer use the "T" (type) command. To see the three lines before the character pointer, you would type "−3T." To see the three lines after the character pointer, type "+3T", or just "3T". If the character pointer is positioned at the beginning of a line, type "T" to see that line. If the character pointer is positioned within a line, type "0T" to see the part of the line before the character pointer, type "T" to see the part of the line after the character pointer, or type "0TT" to see the whole line.

The T command does not move the character pointer; it displays lines of text in relation to the character pointer.

~U

To translate lower-case letters to upper-case. Type "+U" to begin translation and "−U" to end translation.

Ordinarily ED does not translate characters. After being given the command "U" or "+U", however, ED translates characters which enter the edit buffer either from the keyboard or from the original file. Lower-case "a" through "z" are translated to upper-case "A" through "Z." This translation continues until ED receives the "−U" command or the editing session is concluded.

NOTE: You can create a file of all lower-case characters by using the PIP [L] parameter.

~V

When you type "V" or "+V," line numbers are displayed. Type "−V" to suppress display of line numbers. Line numbers are useful in identifying text and moving the character pointer. ("V" stands for verify line numbers.)

0V

A special function of the V command indicates how much of the edit buffer is in use and how much is still available for use. Type a "0" before the "V."

```
*0V<cr>
33706/33719
*
```

The first number ED reports is the amount of available memory in the edit buffer; the second is the maximum possible size of the edit buffer. Subtracting the first from the second, we find that 13 characters of the file being edited are currently in the buffer.

n:

This command moves you to line number "n." When line numbers are being displayed (*see* ~V), you can use this command to move the character pointer directly to the beginning of a specific line. The n: command can be used as a command prefix, making commands start at a particular line number (75:0P will make ED move to line 75 and display a page of text, for example).

:m

Start at the character pointer and continue through line number "m." This is not really a command by itself, but a prefix to a command. Suppose you type the command line

```
55: *25T<cr>
```

ED types lines (T) beginning at the character pointer (currently in line 55) and ending at line 75.

Notice that you can use this command together with the previous one to perform an operation on a specified range of lines. For example, to delete lines 20 through 30, no matter where the character pointer is, type the command line

```
87: *20::30K<cr>
```

~n

Move forward or backward and display one line. This is an abbreviated form of the command ~nLT, where the character pointer is moved forward or backward by "n" lines, whereupon the line then at the character pointer is displayed. The simplest form of this command is just a CARRIAGE RETURN, for example

```
11: *<cr>
12: LINE OF TEXT
12: *
```

This is equivalent to "LT," which moves the character pointer to the next line and displays it.

Commands for Searching and Changing Text

In this section on searching and changing text, the following rule applies: you must use lower-case commands to search for lower-case matches. An upper-case command means that ED will search for upper-case matches to your string only.

nFstring^Z

Find a particular unique sequence (string) of characters. If the search is successful (the string is found the specified number of times), the character pointer is placed immediately following the "n"th matching string. If you were searching for the first chorus of "Row, Row, Row Your Boat" with the command 3fRow<cr>, the character pointer would come to rest between "w" in the last "Row" and the space preceding "Your" if its original position was before the first letter of the first "Row." If the search cannot satisfy the command given, the character pointer does not move.

```
*Row, Row, Row Your Boat        Before

*3fRow<cr>                      Command

*Row, Row, Row Your Boat        After
```

The search begins at the location of the character pointer, so be sure it is positioned well before the string to be found when using the F command.

Also, remember that the F command searches only within the text in the edit buffer; it cannot look through text still in the original file or already saved to diskette. To do so, see the N command, below.

nNstring^Z

Looks for the string in the edit buffer and on the diskette. It automatically loads the next portion of the original file into the edit buffer and writes lines from the edit buffer to the temporary file as necessary. While "F" looks only in the edit buffer for the string, the N command (called autoscan) acts on the entire document, beginning

at the character pointer, even if portions of it are still in the original file.

Remember to position the character pointer properly before issuing the N command.

See the F command above.

Use CONTROL-L to represent the CARRIAGE RETURN / LINE FEED pair if it is part of the string.

nSfindstring^Zreplacestring^Z

Substitute information in the edit buffer. The substitute command finds one string and replaces it with another. As shown, the command would search for "findstring" and replace it with "replacestring." The searching begins at the character pointer and ends at the last character in the edit buffer. The substitution is performed a total of "n" times.

As in all substitution commands, be sure the strings you want to replace are unique.

Use CONTROL-L to represent the CARRIAGE RETURN / LINE FEED pair in the string.

nJfindstring^Zinsertstring^Zendstring^Z

Juxtapose two or more unique strings, one or more times. The command combines the insertion and deletion operations.

This command works in the following way: First find "findstring." Immediately following "findstring" insert "insertstring," then delete all characters from the end of "insertstring" to the beginning of "endstring." Repeat a total of "n" times. Assuming that the operation was successful, the character pointer is placed at the end of the final "insertstring."

For example, suppose the following lines appear in a document:

```
*WHEN IN ROME DO AS THE ROMANS DO,


*AND BE ROMANTIC
```

Using the juxtaposition command JROM ^ZDON'T ^Z AS^Z would result in the following:

```
*WHEN IN ROME DON'T DO AS THE ROMANS DO,


*AND BE ROMANTIC
```

Be careful when using multiple repetitions of the juxtaposition command because you can often change information you did not mean to change.

Combining Commands

ED lets you group some commands in one command line to save typing time. You can cascade ED commands one after the other and follow the last command with

the CARRIAGE RETURN. For example

```
1: *0A<cr>
1: *B<cr>
1: *T<cr>
1: LINE 1
1: *
```

can be typed as

```
1: *0ABT<cr>
1: LINE 1
1: *
```

There are a few simple rules to follow when typing several command on one line.

1. Some commands must be typed alone in a command line; these commands are E, H, O, and Q. This is done to prevent the disastrous consequences of certain typographical errors.
2. When typing commands which use strings, use CONTROL-Z, rather than <cr>, to end the strings. Commands that use strings are F, I, J, N, and S. Use <cr> only at the end of the command line.

You can create a command sequence for frequently used commands. The format is

*nMcommand1command2command3<cr>

For example, type:

```
*MFROM^Z-3DIRAM^Z0TT<cr>
```

The command line says to repeatedly perform the following sequence of steps:

1. Find ROM (the character pointer is placed after the M)
2. Delete the three previous characters (ROM)
3. Insert RAM
4. Show each line as it is changed (0TT).

The macro command will change all occurrences of ROM to RAM. If used on the example in the J commands, the resulting line will read: WHEN IN RAME DO AS THE RAMANS DO, AND BE RAMANTIC... If "n" is absent, or if "n" is equal to 0 or 1, the command sequence repeats until an error condition develops. Striking any key aborts the macro command.

DUMP—DISPLAYING THE CONTENTS OF A FILE

DUMP presents the contents of a file in hexadecimal form.
DUMP operates like the TYPE command discussed earlier. Instead of presenting

the ASCII representations of the file, DUMP presents the contents of the file in hexadecimal form. Assembly language programmers use the DUMP command to check the contents of a program file, a binary data file, or any non-ASCII file.

DUMP d:filename.typ <cr>

Displays the hex representation of each byte stored in the file with the name filename.typ on drive d:.

DUMP d:*.* <cr>

Displays the hex representation of the first file that matches the *.* (or other ambiguous) file name.

```
A>DUMP B:PROGRAM.COM<cr>
00 00 3A 07 00 FE C8 DA AC 03 21 00 00 39 22 25 07 31
00 10 00 C8 3E 11 D3 FD 21 27 07 7D D3 FD 7C D3 FD CD
00 20 3B 02 11 13 04 CD 28 02 CD 3B 02 11 55 04 CD 28
A>
```

The four-digit number which appears at the start of each line is the relative address of the first byte on that line. The pairs of hexadecimal digits represent each byte of the stored file, one pair for each byte in the file.

Special Input

^S pauses DUMP; press any key to interrupt DUMP and return to A>.

Batch Processing Utilities

Your computer is far more sophisticated than the mammoth beasts used in the 1960s. Almost all of the original computers operated primarily in the batch mode. A batch is a group of things; in the case of computers, it's a sequence of commands or data.

Most microcomputers in use today are interactive machines. This means that you input something, the computer responds, you input something else, the computer responds again, and so on. Interactive processing is appropriate for small business accounting, word processing, and other tasks for which microcomputers are used.

However, it is sometimes more efficient to submit a group of commands to be processed sequentially without your presence. The SUBMIT and XSUB commands fulfill these functions for CP/M.

SUBMIT—COMMAND LINE AUTOMATION

SUBMIT directs the sequential entry and execution of a number of CP/M commands without additional operator response. To utilize SUBMIT, you must first create a file with the ".SUB" file type, using the CP/M editor or another editor.

The file you create should contain the list of CP/M commands to be executed in the order you wish them performed, one to a line.

SUBMIT can provide a linkage to the BACKUP or DISKCOPY program every time you finish using a program. First, create a SUBMIT file named ORREPENT.SUB that has only two commands in it.

```
RUN YOURPROG.RAM<cr>
BACKUP<cr>
```

(The RUN YOURPROG.RAM can be any valid CP/M command, and you should substitute the name of your copying program for BACKUP.)

Next, run YOURPROG.RAM by typing SUBMIT ORREPENT<cr> instead of the usual RUN YOURPROG.RAM<cr>; when you finish using your program and attempt to return to CP/M, the SUBMIT facility will next whisk you to the BACKUP program, forcing you to duplicate your diskette every time you run YOURPROG.RAM.

Program developers often use a compiler rather than an interpreter to write programs. A compiled program requires an extra process: the actual compilation. In small business packages there may be five, ten, or more related program modules, all of which need compiling. The compilation process could take as much as an hour or more; the SUBMIT facility performs this function and frees the programmer from typing each compilation command individually.

SUBMIT sequences can be chained (linked together) by including a normal SUBMIT command line as the last line in a ".SUB" type file.

SUBMIT filename<cr>

Creates a file $$$.SUB on the current drive that contains the commands listed in filename.SUB and executes commands from this file rather than the keyboard.

NOTE: CP/M always looks for the file $$$.SUB on the A: drive, so if you are logged into a drive other than "A", the SUBMIT function will not work. Digital Research makes available a patch to correct this problem.

If you have created a file named NUCLEAR.SUB that contains the following:

```
STAT *.BAS<cr>
ERA *.BAS<cr>
DIR *.BAS<cr>
```

when you submit this file for execution the dialog will look like the following:

```
A>SUBMIT NUCLEAR<cr>
A>STAT *.BAS

RECS BYTS EX D:FILENAME.TYP

2    4K   1  A:PORGY.BAS
4    8K   1  A:PORKY.BAS
8    17K  2  A:PORTLY.BAS
BYTES REMAINING ON A: 151K
```

```
A>ERA *.BAS
A>DIR *.BAS
NO FILE

A>
```

The commands following SUBMIT are not underlined, since CP/M is doing the typing, not you. The only thing you typed in the above sequence is SUBMIT NUCLEAR<cr>.

Special Input

Press any key to stop the execution in between submit file commands; that is, just before the prompt is displayed.

SUBMIT filename A B C<cr>

Creates a file $$$.SUB that contains the commands listed in filename.SUB and executes commands from this file rather than from the keyboard. This form of the SUBMIT command differs from the previous one because you may include incomplete CP/M command lines in the file filename.SUB when you create it. SUBMIT fills in the missing information using the "A," "B", "C," and so on from the command line you type when you start SUBMIT. These parameters can be file names or any other information needed by the commands in filename.SUB. The symbols $1, $2, $3, and so on, should be used in your submit file to hold the place for the missing parameters. Up to nine parameters may be used.

Suppose that you are editing a file on drive "A" and you want to copy it to drive "B," then check disk space each time you finish an editing session. Without SUBMIT, you would enter the following command lines in the course of your work (intervening ED commands and other dialog are omitted for clarity):

```
A>ED MYFILE.DOC<cr>
A>PIP B:=A:MYFILE.DOC[V]<cr>
A>STAT B:MYFILE.*<cr>
```

Suppose also that you use this method with every file you edit, not just with MYFILE.DOC. To make SUBMIT do the typing for you, first create a "SUB" type file named WORKON (for example) containing the incomplete commands listed below:

```
ED $1.$2<cr>
PIP B:=A:$1.$2[v]<cr>
STAT B:1$.*<cr>
```

Now, to use this "SUB" file as outlined above, type the command line below:

```
A>SUBMIT WORKON MYFILE DOC<cr>
```

SUBMIT will create $$$.SUB from WORKON.SUB by substituting the first parameter, MYFILE, for $1 each time it appears, and the second parameter, DOC,

for $2 each time it appears. SUBMIT then initiates a warm start, and CP/M looks for the file $$$.SUB. It then executes the commands listed in $$$.SUB.

XSUB—USER INPUT AUTOMATION: A SUBSET OF SUBMIT

It is possible to put more than commands in a "SUB" type file. In fact, it can respond to questions a program might ask or add other variables when you invoke SUBMIT. This is done using the XSUB command. XSUB is not available in CP/M-80 versions 1.3 and 1.4, or in CP/M-86.

XSUB must precede the program name and program response in the submit file you create; it is a command which is not typed when you see the CP/M prompt, but instead is only used within a "SUB" type file as a command to SUBMIT.

When SUBMIT encounters the XSUB command, a special set of instructions is loaded into CP/M's memory space, and whenever a program requests console information, the SUBMIT file is used to input it.

As discussed earlier, you may use $1, $2, and so forth, to indicate last minute submissions of information to SUBMIT, which will pass them on to XSUB, if needed. The symbols in your "SUB" type file are replaced by the parameters you type in the actual SUBMIT command.

You should note that XSUB is a subset of SUBMIT; you do not ever type XSUB in response to a CP/M prompt. XSUB may appear only in a "SUB" type file.

XSUB is most often used in program and system development. But you might find that programs you purchase use XSUB for a portion of their input. You might also use XSUB to provide an addition to the automatic backup system we discussed in the section on SUBMIT. For example, if &BEFREE.SUB contains

```
RUN YOURPROG.RAM<cr>
XSUB<cr>
BACKUP<cr>
A<cr>
B<cr>
```

the following process occurs when you SUBMIT this file:

```
A>SUBMIT &BEFREE<cr>
A>RUN YOURPROG.RAM<cr>
    .
    .      Program runs until completion
    .
A>XSUB<cr>
(XSUB ACTIVE)                    Message from CP/M
A>BACKUP<cr>
DRIVE FOR SOURCE DISKETTE? A<cr>
```

```
DRIVE FOR DESTINATION DISKETTE? B<cr>

PUT BLANK DISKETTE IN B AND PRESS RETURN WHEN READY<cr>
```

In this example, everything that happens from the time you input your original SUBMIT command until you are prompted to put a blank diskette in drive "B" and press RETURN is done automatically by the computer. XSUB supplies the A<cr> and B<cr> in response to the BACKUP program's request for information.

The overriding implication of the use of SUBMIT and XSUB together is that the process of computing can be standardized to the point where actions are repeated in exactly the order you specify, without having to have a computer operator type in each individual command. For some computing jobs, such automated standardization makes sense; for others, especially those that have unpredictable or changing input needs, SUBMIT and XSUB offer no help.

Error Messages

We have deferred a discussion of error messages—messages displayed by the computer when it cannot perform a task as requested or does not understand your request—to the end of this chapter because many of the error messages provided in CP/M are general in nature and the same for several programs. Table 3-1 show the general error messages you might receive.

Each program within CP/M also has some error messages that are only associated with it. Table 3-2 shows is a brief list of those messages with some notations about what to do or to look for to recover from the error.

In general, error messages are printed by the system because there is some action that is expected of you to correct the condition, or because CP/M has terminated the task it was working on because it could not continue because of the error.

Most error messages you will see using CP/M are short but clear enough to convey a general idea of what problem was encountered. Some programs you might receive from firms other than Digital Research occasionally use numbers to represent error messages, such as "Error #1." This is done to save internal computer memory space, but you must look up the error in the manual to find out what happened.

If you encounter an error message you do not understand, first try to logically eliminate or narrow the possibilities. If you are still stumped as to what to do next, consult the vendor who sold you your computer. If all else fails, write to the vendor who created the software that issued the error message, explaining exactly what you did and what the computer reported back to you.

TABLE 3-1. General Error Messages

Error Message	What it Means and What to Do About it
command?	When the command you type is repeated by CP/M and displayed with a question mark following it, CP/M did not recognize it as a built-in command and could not find a file with the type "COM" (CP/M-80) or "CMD" (CP/M-86). Check your typing carefully for misspellings. If you are in doubt about the command, use the DIR *.COM (or DIR *.CMD) command to see a directory of the command files on diskette.
BDOS ERR ON d:	The term BDOS stands for Basic Disk Operating System, ERR stands for error. This message appears when CP/M attempts to do something with a diskette and cannot complete the task. Three primary types of BDOS errors exist.
BAD SECTOR	BDOS ERR ON d:. A BAD SECTOR message indicates that you are either using a unformatted diskette, have a suspect section on that diskette, or have possibly placed the diskette upside down in the drive.
SELECT	BDOS ERR ON d:. The SELECT message means that CP/M could not find the drive you specified. Either you specified a non-existent drive, or the drive has not been powered up, or you have not closed the door on the drive.
R/O	BDOS ERR ON d:. The R/O message indicates that you have either changed diskettes without telling CP/M or you have placed a write-protect tab on the diskette (for 5 ¼-inch diskettes). For 8-inch diskettes, you have removed the write-protect tab. Pressing RETURN forces CP/M to retry, but if the problem that caused the error has not been corrected, you probably will get another error message. A CONTROL-C causes CP/M to reboot; this is the normal method of reacting to a BDOS error.

TABLE 3-2. CP/M Program Error Messages

Program	Error Message	Comments
STAT	** ABORTED **	The STAT command is terminated.
	Bad Delimiter	Check placement of punctuation.
	Invalid Assignment	You cannot assign that device in the manner you specified.
	Invalid File Indicator	You cannot assign the file names to be used in the manner you specified.
	TOO MANY FILES	STAT has an upper limit on how many files it can sort; try to use wildcard file references.
	Invalid Disk Assignment	You cannot assign the disk in the manner you specified.
	Wrong CP/M Version	STAT is CP/M version dependent. Therefore, you may not use a STAT 1.4 with CP/M 2.2.
PIP	DISK READ ERROR	Obvious.
	DISK WRITE ERROR	Obvious.
	VERIFY ERROR	Dr. Kildall never got one of these to appear without a BDOS error also appearing.
	NOT A CHARACTER SINK	You cannot send characters there.
	READER STOPPED	Obvious.
	NOT A CHARACTER SOURCE	You cannot get characters from there.
	ABORTED	The process is terminated.
	BAD PARAMETER	The [ ] parameter(s) is incorrect.
	INVALID USER NUMBER	Obvious.
	RECORD TOO LONG	Obvious.
	INVALID DIGIT	Hex format does not match.
	END OF FILE, CTL-Z?	PIP thinks the end of file has been detected, but wants a confirming CONTROL-Z sent to be sure.
	CHECKSUM ERROR	The Intel format checksum did not match the record read.
	CORRECT ERROR	Obvious.

TABLE 3-2. CP/M Program Error Messages (continued)

Program	Error Message	Comments
	INVALID FORMAT	Obvious.
	NO DIRECTORY SPACE	CP/M has run out of room to keep track of files.
	NO FILE	PIP could not find the file you specified.
	START NOT FOUND	The start string you specified was not found when end-of-file was reached.
	QUIT NOT FOUND	The quit string you specified was not found when end-of-file was reached.
	CANNOT CLOSE FILE	Check diskette.
	DESTINATION IS R/O	The file you specified for information to be sent to is read-only. Press "Y" to delete it and write the new file into it.
	NOT DELETED	Confirmation that you pressed "N" in response to above error.
	NOT FOUND	Same as NO FILE.
	REQUIRES CP/M 2.0	PIP version must match CP/M version.
	UNRECOGNIZED DESTINATION	The device you specified does not exist.
	CANNOT WRITE	Obvious.
	INVALID PIP FORMAT	Check punctuation.
	CANNOT READ	Obvious.
	INVALID SEPARATOR	Check punctuation.
SUBMIT	Error on line	Improper command line in SUB file.
	No SUB file present	SUBMIT could not find SUB file.
	Disk Write Error	Diskette must not be write-protected for SUBMIT to work correctly, and there must be enough room on the diskette for $$$.SUB.
	Command Buffer Overflow	Command too long.

TABLE 3-2. CP/M Program Error Messages (continued)

Program	Error Message	Comments
XSUB	Command Too Long	Command with more than 127 characters.
	Parameter Error	The $1, $2, and so forth, are not specified correctly or do not match SUB file requests.
	Invalid Control Character	You have a control character in your file that SUBMIT cannot recognize.
	Directory Full	Obvious.
	Cannot Close, R/O?	Cannot close the SUBMIT file, is it read-only?
	XSUB Already Present	You have an unnecessary XSUB command in the SUB file.
	Requires CP/M 2.0 or Later	XSUB does not work with CP/M 1.3 or 1.4.
ED	Break "X" at "C" The "C" refers to the command that caused the error message.	"X" refers to one of six symbols: # Search failure ? Unrecognized command 0 The file could not be found > The buffer is full E The command was aborted F The disk is full

CHAPTER

4 Assembly Language Utilities

Despite CP/M's simplicity, the computer novice will still find a lot to learn—certainly more than most users require. The information in this chapter has limited application for users who primarily use packaged programs. If you are not developing or modifying programs in assembly language, skip this chapter.

This chapter covers the CP/M assembly language programming aids, ASM, DDT, LOAD, ASM-86, DDT-86, and GENCMD. ASM and ASM-86 are assemblers. They convert a source code program written in assembly language into an object program in hex format. This product is an intermediate step to a true machine language program which the computer can use. LOAD and GENCMD perform the final conversion step by creating a machine language program file from the hex format file. DDT and DDT-86 are debugging programs used to locate and correct errors in machine language programs. The DUMP program described in the last chapter is another useful assembly language programming aid. DUMP displays the contents of a file as hexadecimal numbers.

ASSEMBLY LANGUAGE

Assembly language is two steps removed from the action in the computer. Remember, the computer uses 1's and 0's as the basic elements of data. These *bits* can be either data or instructions to the computer, depending upon the context in which the CPU receives them.

Since it is inconvenient to think in terms of numbers like 10001010 and 10010010,

we use hexadecimal notation to refer to eight bits at a time. We call these groups of eight bits *bytes* of information. Some hexadecimal representations of bytes are 0C3, 45, and 0D. Remember that the computer itself always uses binary notation (1's and 0's), but we use the hexadecimal representation to help us recognize the patterns.

Since the hexadecimal representations are also awkward, we assign names to the machine instructions and the data they represent. The names are the components of the computer language known as *assembly language*. We call the names that represent instructions *mnemonics*. A mnemonic is the assembly language representation of one CPU machine instruction. Other names are used to represent other elements of a program; these names are all described in the section on ASM and ASM-86.

Generally, any machine language program for the 8080 CPU will run correctly on an 8085 CPU or a Z80 CPU. Since CP/M-80 was written for the 8080, it also runs on the 8085 and Z80. This book uses only the 8080 instruction mnemonics designated by Intel when referring to CP/M-80 assembly language, as they represent the assembly language instructions available to all users of CP/M-80.

The assembly language mnemonics for the 8085 CPU are nearly identical to those of the 8080. Those of the Z80 CPU are completely different however, even though the same machine language instructions are often intended. ASM and DDT understand only 8080 mnemonics, assembly language, and machine language.

CP/M-86 was written for use on the 8086 and 8088 CPUs manufactured by Intel. The differences between these two central processing units primarily lie in how they relate to the other components in the system (the 8088 has only an 8-bit data bus while the 8086 has a 16-bit data bus). Therefore, the mnemonics, assembly language, and machine language for CP/M-86 are consistent no matter what CPU is being used.

This chapter assumes you understand assembly language programming. If you do not, but want to learn, see the bibliography in Appendix F for an appropriate book on the subject.

ASM AND ASM-86: ASSEMBLE A PROGRAM

The CP/M assemblers, ASM and ASM-86, both convert assembly language source programs into object code, executable by the computer. ASM generates 8080 object code, while ASM-86 generates 8086 object code. Each assembler also provides a listing showing each line from the source program together with the corresponding object code it creates, if desired. ASM-86 also generates a symbol table, a cross-referencing tool that allows a programmer to quickly find the location of routines within his or her source code.

Assembly language source code is generated by typing it into a disk file using an editor. CP/M's ED may be used, or more sophisticated editors, such as WordStar, PMATE, or VEDIT may be used.

ASM and ASM-86 Command Line

Once you have obtained or created the source file, you invoke ASM and ASM-86 as follows:

```
A>ASM filename.opt<cr>          CP/M-80
A>ASM86 filename $option(s)<cr>  CP/M-86
```

The file name must be a valid CP/M file with the "ASM" file type for CP/M-80 and the "A86" file type for CP/M-86. If you are wondering why the two assemblers require different file types, it is because source code for either assembler can be resident on the same disk. Therefore, some way of distinguishing the two is necessary.

The *opt* in the CP/M-80 command line refers to a series of three options you may specify and not an "OPT" file type. The three letters following the period represent

- The drive that contains the *source* (original) file (FILENAME.ASM)
- The drive that should receive the *hex* (assembled program) file (FILENAME.HEX)
- The drive that should receive the *print* (listing of the program with error messages) file (FILENAME.PRN).

You can use the standard CP/M drive specifiers "A" through "P" as option letters, subject to the existence of those drives on your computer. Also, the option letters "X" and "Z" have special meanings; they do not mean drives "X" and "Z."

Z Can be used for either the HEX or PRN file option to skip generation of that file.

X Can be used as the PRN file option to display the listing on the console instead of storing it in a file.

If you type

```
A>ASM BSM.AAZ<cr>
```

you are telling CP/M-80's assembler to assemble a file named BSM.ASM, found on drive "A," into a hex (object program) file named BSM.HEX, also on drive "A," and to skip generation of the print file. The assembled program file always has the "HEX" file type; the print file always has the "PRN" file type. If all files are to be placed on the currently logged drive, you need not specify anything after the file name, that is, no options or period. After assembling the program BSM.ASM in such an assembly, you have the following files:

```
BSM.ASM   Original (source) file
BSM.HEX   Assembled program file
BSM.PRN   Listing file
```

ASM-86 operates in a slightly different manner. If you type a period followed by three letters after the file name, CP/M-86 thinks this is the file type for that source code program file. Options are specified after a space and dollar sign.

A~	Specifies source code input location
H*	Specifies hex file output location
P*	Specifies print file output location
S*	Specifies symbol file output location.
F@	Specifies format of hex output file.

Each option must be separated from the others by a space, and all are optional; any that are not specified will be assumed to default to the currently logged disk drive or normal setting.

The characters "~", "*", and "@" following each of the letters above stands for a second letter, as follows:

~ Can be any valid disk drive specifier (A-P)

* Can be any valid disk drive specifier (A-P), or
- X to specify the console device, or
- Y to specify the printer device, or
- Z to suppress output

@ Can be "I" for Intel format, or "D" for Digital Research format.

The options may be entered in any order following the dollar sign. Thus, the following is a valid ASM-86 command line:

```
A>ASM 86 FILE-19.86 $FI HB PY SB<cr>
```

The above line would assemble the contents of FILE-19.86, found on drive "A" (the default), in Intel hex format (the "FI") on drive "B" (the "HB"), print the listing ("PY"), and save the symbol table file on drive "B" ("SB"). When the above command line finished, the following files would exist:

FILE-19.86	Original (source) file
B:FILE-19.H86	Assembled program file
B:FILE-19.SYM	Symbol table listing file

The listing file, FILE-19.LST, would not exist since we sent the listing to the printer device. Note that the hex format assembled program file has the file type of "H86" in CP/M-86 instead of "HEX," and that the listing file has the file type of "LST" in CP/M-86 and "PRN" in CP/M-80. Also, the default source code file type is "A86" in CP/M-86, instead of CP/M-80's "ASM" file type.

Files Used by ASM and ASM-86

What does the multiplicity of files we just discussed do for you? The "ASM" or "A86" file type contains the assembly language source program and is created by you with ED or another text editor.

The assembler creates a "HEX" type file. It contains the hexadecimal representations of the instructions in the "ASM" or "A86" file. The format for the "HEX" file in CP/M-80 is known as Intel Hex Format. Intel first used the format for storing the

hexadecimal characters on paper tape devices, and the disk format is equivalent. Here is an example of a short "HEX" file.

```
:020100003E00BF
:00000000000
```

Not particularly enlightening, is it? Here is the 8080 assembly language program that created that information.

```
        ORG  0100h   ;program starts at address 0100 hex
        MVI  A,0     ;move a 00 hex into Register A
        END          ;end of program
```

The listing file, SAMPLE.PRN, combines both of the above sets of information. We see both the program we originally typed and what the assembler created.

```
0100           ORG  0100h   ;program starts at address 0100 hex
0101  3E 00    MVI  A,0     ;move a 00 hex into Register A
0102           END          ;end of program
```

This type of listing is particularly useful when you use many *labels* and *symbols.*

Of what use are the "HEX" and "PRN" type files? The "HEX" file is used to create an executable program. The "PRN" file is useful in the process of debugging a program (ridding it of errors). The listing in the "PRN" file is also an important part of the program documentation.

CP/M-86 naming conventions differ from CP/M-80. Here are the equivalent files.

CP/M-80	CP/M-86	
filename.ASM	filename.A86	Source code file
filename.HEX	filename.H86	Hex format program code file
filename.PRN	filename.LST	Program listing file
	filename.SYM	Symbol table listing file

In addition to the "HEX" and "PRN" files created by the CP/M-80 assembler, the CP/M-86 assembler creates one additional file, the symbol table file, with the "SYM" file type. A typical symbol table listing file for a short 8086 assembly language program might look like the following:

```
0000 VARIABLES
0080 NAMEN 0090 STAT

0000 NUMBERS
0123 SHORTSTACK
0000 LABELS
0073 LAST  0042 MIDDLE  0000 FIRST
```

VARIABLES lists, in alphabetical order, each of the data statements referenced by the assembler, such as NAMEN DB 0. NUMBERS lists, in alphabetical order, each of the equates or calculated numbers referenced by the assembler. LABELS

lists, in alphabetical order, all of the statement labels found by the assembler. The numbers to the left of each entry are the location at which they were found.

SOURCE PROGRAM FORMAT

The assembler expects the source file to be in a very specific form. The source file is a sequence of ASCII-coded (text) statements or lines. Each line ends with a CARRIAGE RETURN and LINE FEED.

Assembly Language Statements

Each assembly language statement is composed of between one and five *fields*. A field is a group of characters, and the fields are separated from one another by spaces or TAB characters. TAB characters create a more readable source program because the fields are aligned at the TAB STOPs. TAB STOPs are set at columns 1, 9, 17, 25, 33, and so on; that is, at every eight columns. ASM and ASM-86 differ only slightly in which fields they use. The general format of the assembly language statement is

```
line#     label     mnemonic   operand(s) ;comment     CP/M-80
label:    prefix    mnemonic   operand(s) ;comment     CP/M-86
```

Line Numbers (line#)

The line number is an optional integer decimal number at the beginning of an 8080 source code line. Some editors insert these line numbers automatically; ED does not. ASM ignores the line number.

Labels

A label is an identifier used to represent an address or value. A label is one to 16 characters long for ASM, and any length up to the length of the physical line for ASM-86. The first character must be a letter; the others can be letters or numbers. Lower-case letters are treated as if they were upper-case ones. A colon may follow a label when it is used in the label field for ASM, but it *must* follow a label when it is used in the label field for ASM-86. ASM ignores the "$" character while ASM-86 ignores the characters "@" and "_". The label is optional for all statements except those which use it in the operand field, such as with the EQU and SET directives. Generally, a particular label should appear in the label field of only one statement, but it can appear in the operand field of many statements.

There are certain reserved words that must not be used in the label field because they have predetermined meanings to ASM and ASM-86. The reserved words are

- All the 8080 and 8086 instruction mnemonics
- All the ASM and ASM-86 directive names

- All the 8080 and 8086 register names
- Special ASM-86 keywords, BYTE, WORD, and DWORD.

Prefix

Several 8086 instructions, notably those of the REP family and LOCK, perform a special prefixing action to the rest of the instructions on an assembly language statement. To work correctly, such prefixes must precede the mnemonic on the line.

Mnemonics

The mnemonic field is the only field that is not optional; it must be included in every statement. This field contains either an 8080/8086 instruction mnemonic or the name of an assembler directive. Most of the assembly language programming books listed in Appendix F contain a list of the 8080 or 8086 instruction mnemonics. Since the mnemonics are the heart of assembly language, it is wise to make sure that you are familiar with the entire set before attempting to write assembly language code.

Operands

Many assembly language instructions require one or more operands, while some require none at all. Assembler directives generally require one or more operands. An operand can be a constant, a label, or an expression. Constants and expressions are described below.

Comments

The comment field begins with a semicolon. This field is optional and is ignored by the assembler. However, you should always include comments because they are essential program documentation.

Constants and Expressions

The operand field of an assembly language instruction or directive can be occupied by a label, a constant, or an expression. Labels were explained above. Constants and expressions, and the rules for forming them, are described here.

Constants

A *numeric constant* is a fixed number in one of four number bases. The four number bases are binary, octal, decimal, and hexadecimal.

A *binary constant* is a sequence of the digits 0 and 1 followed by the letter "B" to signify that the number is binary.

An *octal constant* is a sequence of the digits 0 through 7 followed by the letters "Q" or "O" to signify that the number is octal.

A *decimal constant* is a sequence of digits 0 through 9 optionally followed by the letter "D" to signify that the number is decimal. If no letter suffix follows a constant, ASM and ASM-86 assume that the constant is a decimal number.

A *hexadecimal constant* is a sequence of the digits 0 through 9 and letters "A"

through "F", followed by the letter "H" to signify that the number is hexadecimal. A hexadecimal constant must begin with a number; this requirement can be satisfied by always preceding the contant with the digit 0.

You can insert the dollar sign within a numeric constant to be used with ASM to make it easier to read. ASM ignores the dollar signs. The following three constants are equal:

11010101 = 1101$0101 = 11$010$101

A *string constant* is a sequence of characters that is enclosed between apostrophe (') symbols. A string constant is limited to 64 characters for ASM and 255 characters for ASM-86. Only printable characters are allowed within such strings. Lower-case letters are *not* converted to upper-case. You can include an apostrophe within a string by typing two apostrophes in a row. ASM and ASM-86 compute the value of a string by adding a high-order bit of 0 to the seven-bit ASCII code for each character.

Expressions

You can use expressions in place of operands for many instructions. An *expression* is a combination of constants, labels, arithmetic operators, logical operators, and parentheses. During assembly, each expression is evaluated and reduced to a single value.

Arithmetic Operators

The assembler can perform simple arithmetic when it evaluates an expression to determine its value. You can use the following arithmetic operators to join labels and constants into expressions:

A+B
: Is the sum of "A" and "B"

A−B
: Is "B" subtracted from "A"

+B
: is the same as "B"

−B
: Is the same as "B" subtracted from zero

A*B
: Is "A" multiplied by "B" (unsigned)

A/B
: Is the quotient of "A" divided by "B" (unsigned)

A MOD B
: Is the remainder of A divided by B

A SHL B
: Is "A" shifted left "B" bit positions; shifted out high-order bits are discarded, and vacated low-order bits are replaced with zeros

A SHR B
Is "A" shifted right "B" bit positions; shifted out low-order bits are discarded, and vacated high-order bits are replaced with zeros.

The assembler performs the operation on 16-bit unsigned values and produces 16-bit unsigned results, modulo 2 to the 16th power.

Logical Operators

The assembler can perform Boolean (logical) operations as well as arithmetic operations. The following operators can be used:

NOT B
Is the bit-by-bit complement of "B"

A AND B
Is the bit-by-bit logical AND of "A" and "B"

A OR B
Is the bit-by-bit logical OR of "A" and "B"

A XOR B
Is the bit-by-bit logical exclusive OR of "A" and "B."

Logical operations are performed on 16-bit unsigned values and result in a 16-bit unsigned value.

Other Operators

Another frequently used operator is the dollar sign, which, when used as the operand, creates a value equal to the current value of the location counter.

In addition, ASM-86 recognizes the following other operations:

SEG A
Is the segment value of "A"

OFFSET A
Is the offset value of "A"

TYPE A
Is 1, 2, or 4, depending upon whether "A" is of TYPE BYTE, WORD, or DWORD

LENGTH A
Is the number of bytes associated with "A"

LAST A
Is LENGTH A−1 unless LENGTH=0, in which case LAST A equals zero

A PTR B
Is the virtual variable created with the type of "A" and the attributes of "B."

Relational Operators (8086 Only)

ASM-86 also allows the following relational operators:

A EQ B
  True if "A" equals "B"

A LT B
  True if "A" is less than "B"

A LE B
  True if "A" is less than or equal to "B"

A GT B
  True if "A" is greater than "B"

A GE B
  True if "A" is greater than or equal to "B"

A NE B
  True if "A" is not equal to "B."

Precedence of Operators

When the assembler evaluates an expression containing several operators, it does not simply proceed from left to right applying each operator in sequence. Instead, it applies certain operators before others. This hierarchy is called the *precedence* of the operators. The hierarchy used by ASM and ASM-86 is shown below. Operators shown on the first line are always used first if they exist in an expression. Those on the fifth line are used last. Within a line below, the operators have equal precedence and are used left to right as they are encountered in an expression. The precedence is as follows:

$ (highest)
the operation in the innermost parentheses
(8086: SEG, OFFSET, PTR, TYPE, LENGTH, LAST)
* / MOD SHL SHR
− +
(8086: EQ, LT, LE, GT, GE, NE)
NOT
AND
OR XOR (lowest)

You use sets of parentheses in an expression to override this hierarchy or to make the expression easier to read.

ASSEMBLER DIRECTIVES

You can include a number of special instructions to the assembler that are not part of the 8080 or 8086 assembly language set. These assembler directives control the assembly process and affect the resulting machine code. You place assembler

directive statements in the source program in roughly the same form as assembly language statements. The directive name goes in the mnemonic field of the statement.

DB, DW, and DS Assembler Directives

Use these three directives to initialize storage areas in memory.

DB Define byte.
Initializes an area byte by byte

DW Define word.
Initializes an area two bytes at a time

DS (8080) Define storage.
Reserves an area of a specified size

DD (8086) Define double word.
Initializes an area four bytes at a time

RS (8086) Define storage.
Reserves an area of a specified size

RB (8086) Define storage.
Similar to RS, above

RW (8086) Define word storage.
Reserves an area of a specified size.

Operand expressions in the DB statements are evaluated and stored as 8-bit values in successive memory locations. Expressions in the DW statement are evaluated and stored as 16-bit values in successive pairs of memory locations; within a pair, the low-order byte is stored first, followed by the high-order byte. The expressions in the DS, RS, RB, and RW directives are evaluated, and then the number of memory locations given by the resulting 16-bit value is reserved. These reserved locations are not filled.

A label is optional in these storage directives. If a label is used, ASM and ASM-86 assign it the value of the address of the first byte defined or reserved by the directive. Any number of expressions, each separated by commas, can be used with the DB, DW, and DD storage directives. Only one expression can be used with the DS, RS, RB, and RW directives, however.

ORG, END, EQU, CSEG, DSEG, SSEG, And ESEG Assembler Directives

A number of location directives are available with ASM and ASM-86. Both assemblers include the following directives:

ORG

This directive gives the assembler the memory address to use for the sequence of

statements that follows it. The form of the ORG statement is

```
label       ORG expression  ;comment
```

The value of the expression is used by the assembler as the memory address of the next program instruction or define directive; this address is fixed for 8080 code, relative for 8086 code. Comments are optional. You can use more than one ORG in a program.

END

The END directive tells the assembler that it has reached the end of the source statements. The 8080 form of the directive is

```
label       END expression  ;comment
```

The END directive is optional; if present, it should be the last statement in the source program.

The optional label is assigned the value of the assembler's location counter at that point in the program. The optional expression is evaluated and used as the program starting address in the Intel Hex Format file. If the expression is omitted, a starting address of "0000" is used. The comment field is optional.

The 8086 form of the directive is

```
            END             ;comment
```

EQU

The EQU (equate) directive assigns values or expressions to a label. The format is

```
label       EQU expression  ;comment
```

In addition, the 8086 assembler recognizes

```
label       EQU register    ;comment
label       EQU mnemonic    ;comment
```

The expression may be any valid number, address, constant, or expression, and in the case of the 8086 assembler, a register or mnemonic as well. The label and expression are required in an equate statement. You may use other labels—if previously defined—within the expression.

A variant of the EQU directive for 8080 code is the SET directive, which, unlike the EQU directive, allows you to reassign the value to a label. The format of the SET directive is the same as for EQU.

CSEG, DSEG, ESEG, SSEG

The CSEG (code segment), DSEG (data segment), ESEG (extra segment), and SSEG (stack segment) may be specified using these 8086 directives. All 8086 source code statements must be assigned to one of the segments in order for the CPU to correctly reference them. Instruction statements are legal in the CSEG only. Directive statements are legal in any of the segments. Data storage statements are valid in the DSEG or CSEG segments.

These directives form an integral part of the 8086 assembly language source code. To understand them properly, you must first understand how the 8086 CPU operates. Consult a reference work on the 8086 to find out more about what these important directives do.

IF and ENDIF Assembler Directives

Define a section of assembly language code that will be assembled only if the condition listed in the IF statement is true. The form of these directives is

```
label     IF      expression                ;comment
                    .
                    .
                    .
                  subsequent   statements
                    .
                    .
                    .
label     ENDIF                             ;comment
```

The assembler evaluates the expression in the IF statement. If the value of the expression is zero, the assembler ignores the statements between the IF and ENDIF. If the value of the expression is non-zero, the statements are assembled.

Using IF statements is called *conditional assembly;* the assembly takes place only if certain conditions are met.

The IF statement may be used in assembly language programming in a number of ways. Here is one of the most frequent.

```
HAVE$TERMINAL    EQU    0FFFFh                ;value = true
NO$TERMINAL      EQU    NOT HAVE$TERMINAL     ;value = false
;
                 IF     HAVE$TERMINAL
                          .
                          .
                          .
                        Assembly language statements to
                        be used when there is a terminal
                          .
                          .
                          .
                 ENDIF
                 IF     NO$TERMINAL
                          .
```

```
                    .
                    .
                    Assembly language statements to
                    be used if there is no terminal
                    .
                    .
                    .
ENDIF
```

In the 8080 example shown, you can change the program to reflect the presence of a terminal by changing only the HAVE$TERMINAL equate. Note that HAVE$TERMINAL (a label) is assigned a value that is all ones; NO$TERMINAL (another label) is assigned the complement value of HAVE$TERMINAL. The operator NOT changes each one to a zero.

Special 8086 Assembler Directives

ASM-86 also recognizes a few additional directives.

INCLUDE

The INCLUDE directive allows an assembly language program to wholly contain another assembly language program without having the actual text to the included program present. The form of the directive is

INCLUDE filename.typ

A file that is included during assembly cannot contain an INCLUDE directive; no nested INCLUDEs are allowed.

TITLE

The listing file prints a title at the top of each page as defined by the TITLE directive.

TITLE 'string'

PAGESIZE

The listing file normally prints with a page length of 66 lines. This can be changed by the PAGESIZE directive.

PAGESIZE expression

PAGEWIDTH

The listing file normally prints with a page width of 120 characters if it sends information to a disk file or printer; a page width of 79 is used with the display terminal. This can be changed by the PAGEWIDTH directive.

PAGEWIDTH expression

EJECT

The listing file can have page breaks forced by using the EJECT directive. When an EJECT directive is encountered, printing resumes with the top of the next page.

EJECT

SIMFORM

The listing device is assumed to respond to form feed characters. If it cannot, the SIMFORM directive should be used to substitute the proper number of line feeds for each form feed character.

SIMFORM

NOLIST

This directive blocks the creation of the list file for all subsequent assembly.

NOLIST

LIST

This directive restores the creation of the list file for all subsequent assembly.

ASSEMBLER PROGRESS MESSAGES

The assembler displays messages when it starts and when it finishes. Since the operations of ASM and ASM-86 are slightly different, we will discuss each separately.

ASM

After you enter the ASM command line, you will see a message such as

```
CP/M ASSEMBLER - VER 2.0
```

This may be followed by error messages if there were any errors.
When ASM has finished its job, it displays a three-line message.

```
xxxx
yyyH USE FACTOR
END OF ASSEMBLY
```

Interpret this as follows: "xxxx" is the hexadecimal address of the first free (unused) location following the assembly language program. "yyyH" is a rather perplexing indication of the portion of the symbol table area that has been used; the assembler has a finite amount of room in which to store the values of labels. "yyy" is a hexadecimal number between 000 and 0FF. This number divided by 0FF hex is the fraction of the symbol table used. If "yyy" is 080, for example, then about one-half (80/0FF hex = 128/255 decimal) of the symbol table has been used. The

END OF ASSEMBLY message means that ASM is finished; it does not mean that the assembly language program was necessarily created successfully.

ASM-86

Like its 8080 counterpart, ASM-86 displays a message when it starts assembly.

```
CP/M 8086 ASSEMBLER VER 1.1
```

During assembly, the following appears:

```
END OF PASS 1
END OF PASS 2
```

These messages indicate ASM-86's progress in assembly. Upon completion, the following message appears:

```
END OF ASSEMBLY. NUMBER OF ERRORS: 0
```

This message is self-explanatory, but it does not indicate that an assembly language program was *successfully* assembled, only that no coding errors were detected during assembly. ASM-86 may be stopped by pressing any key while it is assembling.

ASSEMBLER ERROR MESSAGES

The assemblers display two kinds of error messages. The first is a terminal error message, which indicates that conditions prevented the assembler from completing its job.

Problems with disks or files typically cause terminal error messages. The second kind of error message arises when ASM or ASM-86 cannot properly assemble a source code statement but can continue the assembly despite the error.

Terminal Error Messages

NO SOURCE FILE PRESENT

or

NO FILE

The assembler could not find the "ASM" or "A86" type file with the source code.

NO DIRECTORY SPACE

or

DIRECTORY FULL

CP/M has no more room to keep track of file names. To eliminate this message, erase a few files from your diskette.

SOURCE FILE NAME ERROR

You cannot use "*" and "?" in a file name to be assembled.

SOURCE FILE READ ERROR
or
DISK READ ERROR

The file containing your source file could not be understood or is damaged.

OUTPUT FILE WRITE ERROR
or
CANNOT CLOSE
or
CANNOT CLOSE FILES

Check for a write-protected diskette or a full disk condition.

SYMBOL TABLE OVERFLOW

You have too many labels and symbols in your program for the assembler to keep track of.

PARAMETER ERROR

The options following the file name in the command line are incorrect.

Source Program Error Messages

The assembler provides the second set of messages during program assembly. If no messages appear, the assembler is proceeding and has encountered nothing it cannot understand. (This is not saying the program is correct.) If ASM or ASM-86 encounter something they do not understand, they display a line in the following format:

```
a bbbb cccc  label  mnemonic  operand  ;comment
```

The "a" is a letter representing the type of error. The "bbbb" is the hexadecimal address of the statement at which the error was encountered, and the "cccc" is the hexadecimal representation of the machine language created by the assembler. When an error occurs, some or all of the machine code is set to zeros because the assembler did not know what to put there.

The following are the error codes ASM displays:

D Data error.
The value of the expression does not fit into the data area you indicated; it may be too long.

E Expression error.
You formed the expression improperly or its value cannot be computed at assembly time because it is too complex.

L Label error.
You used the label incorrectly. This normally occurs when you use the same label in the label field of more than one statement in a program.

N Feature not implemented.
Digital Research also has an assembler known as MAC, which accepts directives that ASM cannot handle. When ASM encounters something that only MAC can recognize, the "N" error message is given.

O Overflow error.
Your expression is so complicated that the assembler cannot handle it in its present form. Separate the expression into smaller pieces or simplify it by reducing the number of operators.

P Phase error.
The label changes value during assembly. If you must reassign a value, use the SET directive. This error can also be caused by a duplicate label.

R Register error.
You specified a register that is not compatible with the mnemonic given. For instance, POP B is a valid assembly language statement, but POP A is not. POP A would trigger the "R" error message.

S Syntax error.
This is a catchall error that can occur by using an invalid mnemonic or having a typographical error in the source file.

U Undefined symbol.
You used a label in an expression without assigning a value to it. For example,

```
J0102 06 00     MVI     B,ONE
```

appears if ONE is not defined in the program.

V Value error.
The operand (expression) encountered is not correct. This usually occurs with typing errors, where you forget to include a comma or certain letter the assembler is expecting.

ASM-86's messages use a number system that is similar to that of ASM. ASM-86's messages are more completely defined than ASM's, and most of them are obvious as to their meaning. The following are the error messages ASM-86 presents:

0 Illegal first item.
The first item on the statement line is not a valid label or mnemonic.

1 Missing pseudo instruction.

2 Illegal pseudo instruction.

3 Double-defined variable.
The variable was defined twice.

4 Double-defined label.
The label was defined twice.

5 Undefined instruction.

6 Garbage at end of line—ignored.
Usually encountered when control characters are embedded in file.
7 Operand(s) mismatch instruction.
8 Illegal instruction operands.
9 Missing instruction.
10 Undefined element of expression.
11 Illegal pseudo operand.
12 Nested IF illegal—IF ignored.
You have an IF statement embedded in a section of code that already is being evaluated with an IF directive.
13 Illegal IF operand—IF ignored.
14 No matching IF for ENDIF.
15 Symbol illegally forward referenced—neglected.
16 Double defined symbol—treated as undefined.
17 Instruction not in code segment.
18 File name syntax error.
19 Nested INCLUDE not allowed.
20 Illegal expression element.
21 Missing TYPE information in operand(s).
BYTE, WORD, or DWORD must be assigned first.
22 Label out of range.
23 Missing segment information in operand.
24 Error in code macro building.

DYNAMIC DEBUGGING TOOL (DDT)

The Dynamic Debugging Tool (DDT) is used to test and debug machine language programs. CP/M-80 comes with DDT, while CP/M-86 comes with DDT-86. You can use DDT to

- Load an assembled program into memory
- Make simple changes to a machine language program
- Help locate errors in machine language programs
- Make corrections or updates to your software
- Install special driver routines (programs that drive a peripheral, such as a printer)
- Change disk parameters with CP/M-80 version 2.2 or CP/M-86
- Examine and modify the contents of memory
- Enter assembly language code one line at a time

- Disassemble a section of a program (for example, turn machine language back into assembly language instructions)
- Examine and modify the contents of the internal registers of the CPU
- Set breakpoints—locations at which to stop the program and check what has happened so far
- Trace the execution of a program—follow what happens to memory and the internal registers of the CPU.

By typing in the following:

```
DDT<cr>
```
CP/M-80

or

```
DDT86<cr>
```
CP/M-86

you load the Dynamic Debugging Tool program into the memory of your computer where it waits for further instructions.

Again, typing the following:

```
DDT d:filename.typ<cr>
```
CP/M-80

or

```
DDT86 d:filename<cr>
```
CP/M-86

loads DDT into memory and also loads the designated file into memory for examination, modification, or extension. The file type must be "COM" or "HEX" for CP/M-80; DDT-86 assumes a file type of "CMD" for loading.

When DDT or DDT-86 are resident in your computer's memory, they display a prompt that consists of a hyphen. The hyphen indicates that DDT is waiting for a command.

DDT Commands

Once DDT has been loaded into memory (with the optional file name, if specified, being placed into memory as well), DDT is ready to accept commands. A brief summary of the available commands is shown in Table 4-1.

Most DDT commands require additional information in order to be used. Often, this additional information is a memory address or a hexadecimal value. DDT and DDT-86 expect slightly different formats to be used for specifying locations in memory, because CP/M-80 only uses 16-bit addresses, while CP/M-86 uses 20-bit addresses.

CP/M-80

0000 to FFFF are valid addresses

CP/M-86

0000:0000 to F000:FFFF are valid addresses. The number before the semicolon is the 16-bit segment number; the number after the semicolon is the 16-bit address within the segment specified. If you are not sure about

TABLE 4-1. DDT Commands and Functions

Function	Letter Typed	
	CP/M-80 (DDT)	CP/M-86 (DDT-86)
Assemble instructions	A	A
Display memory	D	D
Load program		E
Fill memory	F	F
Execute program	G	G
Hexadecimal arithmetic	H	H
Set up FCB	I	I
List instructions	L	L
Move memory	M	M
Read disk file	R	R
Set memory to value	S	S
Trace program	T	T
Execute partially	U	U
Show disk file read		V
Write contents to disk		W
Examine/modify registers	X	X

what a memory segment is, refer to the section on CP/M-86 in Chapter 7 or consult a technical reference on the design of the 8086 CPU. You do not need to specify the segment number—it is valid only to specify the 16-bit address. In addition, you may use the name of one of the four 8086 segment registers as the number before the semicolon and DDT-86 will fill in the proper offset.

Assembly Instructions

A#<cr>

Enter assembly language instructions beginning at the hexadecimal address specified. There should be no space between the command letter (A) and the address you specify. After pressing CARRIAGE RETURN, DDT displays the memory address and waits for you to type a valid 8080 (or 8086 for DDT-86) mnemonic or operand. Mnemonic and operand should be separated by one space, and each instruction is terminated by issuing a CARRIAGE RETURN. After each instruction is entered, the next available address is displayed, and you may continue entering assembly language instructions in the fashion just described. To end the entry of assembly language instructions, simply type a period followed by a CARRIAGE RETURN instead of a valid instruction. Entry can also be terminated by merely pressing the CARRIAGE RETURN key as the first character on a new line.

```
-A0100<cr>
0100 MOV A,C<cr>
0101 .<cr>
```

If DDT or DDT-86 do not understand your instruction, they present a question mark and repeat the memory address to be filled.

Display Memory

D<cr>

Three forms of the display memory command exist. The first is merely the letter "D" followed by a CARRIAGE RETURN. The single letter tells DDT to display memory at the current memory location. On most systems, a total of 192 bytes (12 rows of 16 bytes each) are displayed. Some systems, notably those with screens less than 80 characters wide, may display fewer characters, but there is no functional difference to the display memory command.

Memory is displayed with the first memory location for a line at the left-hand edge of each line, followed by 16 hexadecimal representations of the bytes in memory beginning at that address. Next, 16 ASCII characters are displayed (with the period being used to substitute for any nondisplayable character) on the far right of each row. These 16 characters are the ASCII representations of the 16 bytes being displayed in hexadecimal.

```
-D<cr>
0120  3F 07 00 FE C8 DA AC 23 21 00 00 39 22 25 07 31  ?.......!..9"%.1
0110  00 C8 3E 11 D3 FA 21 27 07 7D D3 FD 7C D3 FD CD  ..>...!'.}..|...
0120  3B 02 11 13 04 CD 2B 02 CD 3C 02 11 55 04 CD 2B  ;.....+..<..U..+
0130  02 1E 07 0E 07 CD 95 00 FE 21 CD 05 00 FE 1B CA  .........!......
0140  A1 03 FE 0D C2 28 01 11 91 04 3D 28 02 3E 4D 32  .....(....=(.>M2
0150  7A A7 21 10 07 36 0B 7E D6 0D D3 FF 3E 07 32 1E  z.!..6.~....>.2.
0160  07 21 2A 07 35 3E 01 32 2E 07 11 AE F9 3D AE 32  .!*.5>.2+....=.2
0170  3E A4 32 20 07 CD 69 03 21 27 07 35 21 CE 2E 02  >.2 ..i.!'.5!...
0180  FA 36 02 CD 25 02 C3 69 01 21 1E 07 35 C2 E1 01  .6..%..i.!..5...
0190  3E 23 32 24 07 FD 75 02 3A 2A 07 C6 07 3D 3A 07  >#2$..u.:*...=:.
01A0  36 07 21 2A 07 35 3E 02 32 21 07 CD 65 02 21 27  6.!*.5>.2!..e.!'
01B0  07 36 32 CD 7F 02 FA E3 02 3E 01 32 2E 07 11 AE  .62......>.2+...
-
```

Each successive display memory command displays the next 192 bytes of memory. The starting address for the first use of "D" depends upon the file, if any, loaded with DDT. Common starting values are 0100 hex and 0000 hex.

D#<cr>

The second form of the display memory command allows you to specify the starting address to use. If you start a display command with an address that is not a multiple of 16 (does not end with a 0), the first line of the display will not have 16 locations represented. DDT likes nice round numbers, and round numbers to it

mean multiples of 16. If we typed D501, the first line of our displayed memory would show only 15 bytes, for example. Any subsequent letter "D" typed without an address will resume where the "D#" display leaves off.

```
-D500<cr>
0500  52 52 4F 52 20 20 20 44 52 3E 42 24 52 45 41 44  RROR   DR>B$READ
0510  20 45 52 52 4F 52 20 20 20 20 44 52 3D 42 24 56   ERROR    DR=B$V
0520  45 52 49 46 59 20 45 52 52 4F 52 20 20 42 4C 4F  ERIFY ERROR  BLO
0530  43 4B 3D 2D 2D 24 20 20 54 52 4B 3D 2D 2D 20 20  CK=--$  TRK=--
0540  53 43 54 52 3D 2D 2D 24 20 20 53 54 53 57 44 3D  SCTR=--$  STSWD=
0550  2D 2D 2D 2D 2D 20 2D 2D 20 20 24 AE 05 C1 05 05  -----  --  $.....
0560  05 E5 05 F5 05 08 06 1F 06 9B 05 2F 06 3F 06 58  ...........?.?.X
0570  06 9B 05 9B 05 9B 05 9B 05 9B 05 71 06 83 06 95  ...........q....
0580  06 A7 06 BE 06 D4 06 EB 06 FF 06 9B 05 9B 05 9B  ................
0590  05 9B 05 9B 05 9D 05 9B 05 9B 05 4E 4F 20 45 52  ...........NO ER
05A0  52 4F 52 20 4D 53 47 20 46 4F 55 4E 44 24 44 52  ROR MSG FOUND$DR
05B0  49 56 45 20 4E 4F 54 20 4F 50 45 52 41 42 4C 45  IVE NOT OPERABLE
```

D#,#<cr>

A third form of the display memory involves typing two numbers: the starting and ending locations of the memory to display. If you start or end the display with a memory location that does not end in 0, your display will not have exactly 16 bytes in the beginning and ending lines, respectively. Any subsequent letter "D" typed without an address resumes where the "D#,#" display leaves off.

DDT-86 allows you to display information as 16-bit values, normally referred to as words of memory, not the eight-bit values normally shown. To specify 16-bit value displays, insert a "W" between the "D" and the address parameters.

DW<cr>	Displays 96 words of memory
DW#<cr>	Displays 96 words of memory beginning at "#"
DW#,#<cr>	Displays 96 words of memory beginning at the first number and ending at the second.

Fill Memory

F #,#,#<cr>

The fill memory command allows you to fill a block of memory with a single, constant byte value. A common use of this instruction is to put the null character (00 hex) into a section of memory before doing any new work in that area.

The fill memory command requires three numbers to follow the letter "F" that identify it. The first number is the starting address of the memory block to fill, the second is the ending address of the memory block, and the third is the byte to insert into the block of memory.

```
-F0100,01F0,0A0<cr>
-
```

The above command would place the value A0 hex in every memory location from 0100 hex to 01F0 hex, inclusive. As with many of the other commands, you can tell DDT-86 to use 16-bit words instead of bytes by placing the letter "W" immediately following the "F" that starts the fill memory command (remember to change the byte value to a word value or you might get some unexpected results).

You can use the fill memory command as a crude memory test to isolate blatant memory errors. Try filling memory with "00," "55," "AA," and "FF" with successive commands. After each command use the display memory command to make sure that the memory actually received the characters you sent it. If you fill memory with 55 hex and see a byte with 54 in it, you most likely have a memory problem and you should take your machine to your vendor for repairs. Be careful, however, not to fill memory locations used by CP/M or DDT (generally the first 256 memory locations and the last 6K to 10K memory locations).

Execute (Go at Location)

G<cr>

The "G" command tells DDT to begin execution of the instructions of memory at the memory address contained in the CPU's program counter. DDT-86 begins execution at the location derived from the CS and IP registers.

You seldom use the "G" command when debugging a program without specifying a final address because DDT has no control over execution with this form of the command. After the "G" command is used by itself, you have no way of stopping the program and returning to DDT unless you have placed the correct RST instruction for DDT somewhere in the executed code.

G#<cr>

To specify execution to begin at an address other than the current program counter (or CS and IP register for DDT-86), follow the "G" command with a number indicating the starting address to use. This form of the command places the hexadecimal number you specify into the program counter (or CS and IP registers) and then lets the CPU begin execution at that address. As with the "G" command, DDT may never regain control.

G#,#<cr>

To specify that instructions beginning at one address and ending at another be executed, follow the "G" command with the two addresses separated with a comma. This command first sets a breakpoint at the second address by storing an RST 7 instruction there. Then execution is started at the first address. If the CPU executes the breakpoint at the second address, execution of the program stops, and DDT prompts for another command.

You may specify two addresses which, when encountered, cause the CPU to restart DDT. The form of this command is G#,#,#<cr>. If either breakpoint is executed, DDT prompts for another command.

G,#<cr>

You may omit the starting address in any otherwise valid "G" command to tell DDT to use the current program counter (or CS and IP register values). This form of the "G" command is useful for continuing execution after encountering a previous breakpoint.

Hexadecimal Math

H#,#<cr>

The hexadecimal math command computes the sum and difference of the two hexadecimal numbers you specify. The numbers are first converted to 16-bit values (if you type less than four hexadecimal digits) before the calculations are made. The first number shown will be the sum, the second the difference between the two numbers.

Input Command

Ifilename<cr>

The "I" command identifies the file you wish to load into memory using the "R" command. The first file name following the "I" command is placed into the default file control block (normally at 005C hex). If more than one file name is specified (separated with a space), the second one is placed in the second portion of the file control block (normally 006C hex). The length of the command following the "I" is placed at 0080 hex, followed by the command, followed by a terminating binary zero character.

In short, the "I" command prepares the file control block so that you can use the "R" command to read a file from disk into memory. The "I" command always sets up the file control block for accessing the currently logged disk drive. You cannot include a drive specifier—such as B:—with the "I" command; an error message is issued if you do. You could, however, modify the file control block using the "S" command to change the byte at memory location 005C from 00 to the value that corresponds to the desired drive. Choose one of the values from Table 4-2.

In DDT-86, if a file has been loaded using the "E" command, DDT-86 copies the file control block information from the base page of the program loaded.

Because the "I" command also fills in the command buffer at 0080 hex exactly as if the user had typed it, the "I" command can be used to simulate what would happen if a user typed in a command at the CP/M prompt level and traced execution of the program using DDT.

List Memory

L<cr>

To disassemble a portion of memory, use the "L" (list) command. The "L" command lists the contents of memory in assembly language, beginning after the

TABLE 4-2. Values for Disk Drive Specifier in Default File Control Block

Contents of Location 005C (Hexadecimal)	Disk Drive Selected
00	Currently Logged Drive
01	Drive B
02	Drive C
03	Drive D
04	Drive E
06	Drive F
07	Drive G
08	Drive H
09	Drive I
0A	Drive J
0B	Drive K
0C	Drive L
0D	Drive M
0E	Drive N
0F	Drive O
10	Drive P

last listed address (or 0100 hex, if none have been listed).

```
-L<cr>
0151 MOV D,A
0152 MVI E,00
0154 PUSH D
0155 LXI H,0200
0158 MOV A,B
0159 ORA C
015A JZ 0165
015D DCX B
015E MOV A,M
015F STAX D
0160 INX D
-
```

When DDT does not know how to display a hexadecimal value encountered in 8080 assembly language, the message ??=## is displayed, where "##" is the hexadecimal byte found by DDT.

Each successive use of "L" disassembles the next eleven instructions. Operands that are numeric values are always displayed as hexadecimal numbers; labels and symbols are not displayed (Digital Research's ZSID Symbolic Debugger displays labels, if known).

L#,#<cr>

Like many of the other DDT commands, you may specify a starting and ending address for the "L" instruction. You may omit either number, but the comma must be present if you omit the starting address.

Moving Memory

M#,#,#<cr>

The "M" (move memory) command requires three numbers to function. The first number is the first memory location to be moved, the second is the last memory location to be moved, and the third is the first memory location of the new position for the block of memory specified.

There are a number of considerations to keep in mind with the move command. Do not specify a destination within the block of memory to be moved.

The move command does not relocate a program. A program usually contains references to other locations within the program. The move command does a literal move; each byte in the original block is moved exactly as is to the new location. If a program is moved away from its normal location, it most likely will not work at its new location.

If a question mark appears after you type the "M" command, at least one of the numbers you specified is not a valid hexadecimal address. Also, if the last memory location to be moved is a lower address than the starting memory location, nothing is moved by DDT.

Read File

DDT
R#<cr>

To read a file from a diskette, first use the "I" command to identify the file, then issue the "R" command. The "R" command is followed by an optional one-to-four character hexadecimal address offset.

The optional number is an offset, or bias, as Digital Research refers to it. If omitted, an offset of zero is assumed. The offset is added to the normal load location, and the file is loaded at the resulting address.

Any address that would exceed FFFF hex wraps around to 0000 hex. Thus, a file that would load at 8000 hex with an offset of zero would load at 0100 if loaded with an offset of 8100 (R8100<cr>). Also, you may not load a file so that it resides in the portion of memory between 0000 and 0100 hex or so that it overlays DDT.

You should not load a file that would load into any part of memory used by CP/M-80. The forbidden areas are 0000 to 0100 hex and the CP/M-80 area below FFFF hex. The address of the start of this area can be found by entering the command L5,7<cr>; the address shown after the JMP mnemonic is the lowest address used by CP/M and DDT.

If you entered a file type of "COM" with the "I" command, then loading begins at 0100 plus the offset. If you entered any other file type, including "HEX," the "R" command assumes the file will be in Intel hex format, and it will add the offset to the addresses contained in the hex format file to determine load addresses.

DDT-86
R filename<cr>

DDT-86 does not allow an offset to be used. The "R" command reads the specified file into memory. The 8086 CPU allows up to seven files, plus DDT-86, to be resident in memory at one time without overlap. Thus, DDT-86 loads files into memory with the "R" command in the first free memory area.

Set Memory

S#<cr>

The set memory command allows you to display and optionally change (set) the contents of memory, beginning at the location you specify immediately following "S." In DDT-86 you may follow the "S" with a "W" to indicate that words are to be set instead of bytes.

DDT displays a memory location, the current contents of that location, and then waits for you to enter a byte (or word) value, or simply press CARRIAGE RETURN to leave the setting as it is.

```
-S0100<cr>
0100 C3 3D<cr>          C3 become 3D
0101 28 4F<cr>          28 become 4F
0102 38 <cr>            Leave as is
0103 C# <cr>            Terminate input
-
```

A question mark appears if DDT or DDT-86 cannot understand your input or you typed an invalid hexadecimal digit or address.

Trace

DDT, DDT-86
T#<cr>

To selectively trace program execution, use the trace command. Trace displays CPU registers as they exist immediately prior to the next program instruction's execution. The number following the "T" command indicates how many program instructions to execute. Pressing any key terminates tracing and returns control to DDT.

A program being traced runs about 500 times slower than normal because DDT instructions simulate each program instruction. The trace command enables interrupts. This can be a major problem if your program requires that interrupts be disabled.

The trace display looks like the following for CP/M-80:

```
C0Z0M0E0I0 A=00 B=0FB6 D=0000 H=0000 S=0100 P=013D LXI SP,0200
```

where C0 means the carry flag is set at zero
Z0 means the zero flag is set at zero
M0 means the minus flag value is zero
E0 means the even parity flag is zero
I0 means the intermediate carry flag is zero
A=00 means the A register contents are 00
B=0FB6 means the BC register contents are 0FB6
D=0000 means the DE register contents are 0000
H=0000 means the HL register contents are 0000
S=0100 means the stack pointer is at 0100
P=013D means the program counter is at 013D
LXI SP,0200 is the next instruction to execute.

The trace display looks like the following for CP/M-86:

```
          AX   BX   CX   DX   SP   BP   SI   DI   IP
---SZAPC 0003 0100 0000 0000 119E 0000 0000 0000 001C INC   SI
```

where the left column is the flags that are set
the next nine columns are the 8086 registers
the last column is the next instruction to execute.

DDT-86 also allows you to display the 8086 segment registers during the trace operation. To do so, specify an "S" following the "T" (TS#<cr>).

Untrace

U#<cr>

The untrace command is exactly the same as the trace command, except that you see only one line that represents the CPU registers before the completion of any program steps. It is similar to setting a breakpoint in a "G" command and preceding this with an "X" command. The "U" command thus differs from the "T" command in that the "U" command displays register values only once, while the "T" command displays registers at each step. Using "U" is faster than using "T", but slower than using "G." Using "U" is preferable to using "G" without breakpoints because you can interrupt execution by pressing a key on the keyboard. As with the trace command, an "S" following the "U" displays the segment registers using DDT-86.

Examine CPU State

X<cr>

To examine the current state of the CPU, you use the "X" command. The format of the display is the same as with the "T" and "U" commands, unless you specify a single register to display, such as

DDT:

XC	Carry flag
XZ	Zero flag
XM	Minus flag
XE	Even parity flag
XI	Interdigit carry flag
XA	Accumulator
XB	Register pair BC
XD	Register pair DE
XH	Register pair HL
XS	Stack pointer register
XP	Program counter register

DDT-86:

XO	Overflow flag
XD	Direction flag
XI	Interrupt enable flag
XT	Trap flag
XS	Sign flag
XZ	Zero flag
XZ	Auxiliary carry flag
XP	Parity flag
XC	Carry flag

Plus all registers (AX,BX,CX, and so forth).

Load for Execution (DDT-86 Only)

Efilename<cr>

The "E" command allows you to load an 8086 file into memory so that a subsequent "G," "T," or "U" command begins program execution. If no file type is specified, the file type of "CMD" is used. As usual with 8086 program files, the header information in the file is used to alter the segment registers and the IP register before execution. DDT-86 displays the starting and ending addresses of each segment in the program when the file is loaded.

Value (DDT-86 Only)

V<cr>

The value command displays information pertaining to the last file loaded using an "E" or "R" command. If the file was loaded using the "E" command, the "V" command displays the start and end addresses of each of the segments in the file. If the last file was read with an "R" command, "V" displays only the starting and ending addresses of the memory block where the file was loaded.

Write File (DDT-86 Only)

Wfilename,#,#<cr>

The write command is similar to the CP/M-80 SAVE command in that it allows you to write a block of information to a disk file. The two numbers following the filename and separated by commas are the starting and ending addresses of the memory block. If these addresses are omitted, the start and end addresses found in the last "R" command are used.

LOAD—CREATE AN EXECUTABLE PROGRAM

The CP/M-80 LOAD command has only one function: it takes a file of type "HEX" and converts it into an executable file with the "COM" file type.

A "HEX" type file is created by the CP/M-80 assembler. A "HEX" file contains Intel hex format machine code ready to be tested using DDT or converted into an executable file. LOAD creates a "COM" file, which begins at 0100 hex, and which contains executable machine code.

To use LOAD, first assemble your file using "ASM." Next, use LOAD to create a "COM" type file.

```
A>LOAD B:PDX<cr>
FIRST ADDRESS          0100
LAST ADDRESS           0234
BYTES READ             0135
RECORDS WRITTEN          02
A>
```

If your "HEX" file is a long one, the disk drive may be active for an unreasonably long time as LOAD performs its function. The messages printed out are informative only, and tell you some basic information about the size of the "COM" type file created.

Load Error Messages

Several error messages may be presented by LOAD if your file is in the incorrect format or if a problem occurs in making the conversion. They may be

CANNOT OPEN SOURCE, LOAD ADDRESS xxxx

This message is displayed if LOAD cannot find the file name you specify or if no file name is specified.

INVERTED LOAD ADDRESS

The program origin is less than 0100 hex or the program contains at least one statement with an address less than the address of the previous instruction (the hex format records are not in ascending order by address). Check the ORG statements in your assembly language file.

INVALID HEX DIGIT
CHECKSUM ERROR

These messages appear when the information within the "HEX" type file is incorrect. This normally would not occur if you immediately assembled the output of an assembly, but may occur if you use Intel hex format to pass information between computers and then edit the file.

DISK READ
DISK WRITE
NO MORE DIRECTORY SPACE
CANNOT CLOSE FILE

Each of the above errors are printed in conjunction with specific problems encountered by LOAD trying to read from or write to the diskette. Check for a full diskette, a full directory on the diskette, or if the diskette is set to read-only status.

GENCMD—CP/M-86 Create an Executable Program File

CP/M-86 allows programs to reside in almost any portion of memory, unlike CP/M-80, which assumes that all transient commands (programs) start at 0100 hex. Because of this difference, LOAD is not appropriate to the CP/M-86 programmer; instead, you use GENCMD.

GENCMD takes a file of type "H86" (created by ASM-86) and creates a file of type "CMD" from it. In this sense, it functions almost exactly like LOAD.

GENCMD also contains a great number of options. These options allow you to specify such things as using the 8080 model for intermixing code and data, location of the code, data, or stack elements, or the specification of the memory size used for each segment.

The format for the GENCMD command is

```
A>GENCMD filename options
```

where "filename" is the name of the H86 file to use and "options" is the list of optional settings to use during the generation of the CMD file. The valid options are

8080
: Specifies that code and data are intermixed into a single 64K byte segment, regardless of the use of specific segment directives in the program

CODE[#]
: Specifies the start of the code segment

DATA[#]
: Specifies the start of the data segment

EXTRA[#]
: Specifies the start of the extra segment

STACK[#]
: Specifies the start of the stack segment

X1[#]
X2[#]
X3[#]
X4[#]
: Specify the start of auxiliary segments.

CHAPTER

5 Transient Programs And CP/M

You will use many different types of programs in conjunction with CP/M. Digital Research provides a number of support programs to help you develop your own software, but Digital Research markets little or no user application programs. Remember, CP/M is not the entire solution, although application programs contribute significantly to the solution.

The four classes of *solution* programs this chapter addresses are

- Utilities
- High-level languages
- Application programs
- Word processors

In Chapter 2 we used the term *housekeeping* to describe programs that help you keep your diskette and its files clean. We refer to another class of related programs as *utilities*. A utility diskette contains programs that maintain your diskette collection. Such utility programs are

- Editors
- Disk copying programs
- Disk formatting programs
- CP/M system generators
- Disk viewers

Editors differ in degrees of sophistication; a pencil, scissors, tape, typewriter, and copy machine individually add or modify information on paper. An editor or word processing program adds, deletes, or modifies information on a disk.

You may add information on a disk using Digital Research's Context Editor, ED, or by using a powerful word processing program (such as WordStar). In this chapter we briefly describe several popular editors and explain how their use relates to the use of CP/M.

Before we proceed with a description of the utility, language, and application programs you will use with CP/M, it is appropriate to back up a step and make sure that you understand the hierarchy that software follows.

Machine instructions are one of several layers of software existing between the computer and the user. Historically, software was frequently designed to make subsequent development of microcomputer programs easier. In this manner, several layers of software exist; each layer is directly dependent on an earlier development.

Machine instructions are at the core of programming. They are the binary 1's and 0's the CPU interprets. Each different pattern of 1's and 0's causes the CPU to perform a unique task. Each CPU model has a particular set of instructions it understands. This is why CP/M-80 and CP/M-86 differ. To program (instruct) a computer at this level, you need a method of entering the machine instructions. CP/M does not directly provide this facility, but you may enter instructions indirectly with DDT (see Chapter 4). Two conceptual problems are apparent with machine instructions, however.

A machine instruction (such as 11000011) bears little resemblance to the operation that the computer performs in response to it. This is true of any code or language that does not use hieroglyphics. While the 1's and 0's represent various ON and OFF states to the computer, we must translate the 1's and 0's further. In human languages, number strings do not form basic components of meaning. A series of numbers may represent something, but we require an intermediary translation. Computers are similar, and, therefore, the 11000011 translates into an instruction to "jump" to an address for 8080-based computers.

Instructions perform operations that are firmly rooted in computer architecture and terminology. The task you wish to perform may be difficult to describe using the selected computer's machine instructions. Suppose you want to place a character on a piece of paper using a printer; some of the necessary instructions might be

Hexadecimal Representation	Binary Machine Representation
DB	11011011
03	00000011
E6	11100110
01	00000001
C2	11000010
00	00000000

Hexadecimal Representation	Binary Machine Representation
00	00000000
3E	00111110
58	01011000
D3	11010011
02	00000010

Assembly language moves one step away from the computer, providing a mnemonic to describe the action of each machine instruction. Using the above 8080 example (sending a character to the printer to type), we now get

```
IN    03
ANI   01
JZ    0000
MVI   A,58h
OUT   02
```

If you know what MVI, IN, ANI, JZ, and OUT mean, then the instruction is now represented in a shorthand that is translated into human-understandable language. Unfortunately, these five assembly language program lines convey only the meaning of each single instruction. What do the five lines together, as a program, accomplish?

One step further removed from machine instructions, the computer interprets a high-level language. Here assembly language instructions are combined into larger building blocks. Our program now becomes

```
LPRINT "X"
```

which sends the character "X" to the printer. The concept of what the program does is now evident. LPRINT "X" may actually trigger the execution of several (or several hundred) machine instructions. But by LPRINT "X" we immediately understand the function of the program. As we get closer to understanding the task the computer accomplishes, we get further from the machine instructions the CPU actually executes. The first high-level languages were created using the next most sophisticated method, in this case, assembly language. And the first assembly language programs were written in machine language.

The last level we will consider is the application program. An application program causes a computer to execute a specific task, for example, to create and maintain a list of names and addresses. Most computer users run application programs unaware of any lower level of program interface.

Now that we have introduced the levels of program instructions, let us now return to look in more detail at each of the four types of programs you may use with CP/M.

UTILITY PROGRAMS

We will present several useful utility programs in a general fashion. Each supplier of CP/M may also provide some utility programs that are similar to, but not the same as, the programs described here. It is important to be aware of these utilities and to learn how to use them. This is particularly true of the disk formatting and copying programs. As you read the following sections, consult the manual(s) that accompany the version of CP/M you purchased. Compare the material in your manuals with our discussion to determine how to use your CP/M utility programs.

Format—Preparing a Diskette for Use

How does the computer know where to put information on a diskette? We discussed how the diskette is laid out in sectors and tracks, but how does the computer distinguish between sector and tracks? How does it assign information within a sector or track? Part of the job is done by the hardware (equipment) and CP/M, but another part requires you to prepare the diskette.

Almost any CP/M-80 or CP/M-86 package includes a format or initialization type of program. Among others, the following names have been given to initialization programs:

FORMAT	INIT	IN
DSKFMT	FMT	CREATE
INITDSK	INITDISK	FORMT#
FORMTHD	MFORMAT	HDF

The exact name of your formatting program depends on your CP/M source. In some rare instances, there may be no initializing program; instead, the accompanying documentation describes how to initialize a diskette.

Every blank diskette must be formatted before you can put data onto it. This is a good practice even if your diskette vendor claims to have formatted the diskettes for you.

Assume your initialize diskette program is called FORMAT. A FORMAT program session might look like this.

```
A>FORMAT<cr>
DISK TO FORMAT? (A,B,C,D) B<cr>
PRESS RETURN TO BEGIN FORMATTING <cr>
DISK FORMATTED. MORE? (Y/N) N<cr>
A>
```

While the exact syntax of the messages may differ, the following steps normally occur in any format program:

1. Run the program (type its name, then press CARRIAGE RETURN)
2. Tell the program which drive contains the diskette to format

3. If your system runs both single and double density diskettes, then you have to specify a density
4. Tell the program to start formatting
5. Tell the program whether you want to quit or format another diskette when it finishes.

The program may also direct you to remove any diskette you are not formatting to prevent accidental formatting of a diskette that contains valuable, irreplaceable information.

The format program may or may not obey the write-protect notch on the diskette. To determine this, try to format a protected diskette. If the format program ignores the write-protect mechanism, you must be extra careful not to inadvertently format a diskette containing useful information.

Initializing a diskette only prepares it for use. If you ask for a directory of the diskette after initializing it, messages like NO FILE or NOT FOUND appear, since no files are yet present. Also, initializing a diskette does not mean that the diskette can be used to boot (start) the system; the diskette does not contain the CP/M system yet (*see* SYSGEN below).

The format program moves the magnetic head of the disk drive to the first sector of the first track. Then the program writes some known dummy information and moves on. Each sector of a track is written into, then the head moves to the next track. Most programs fill the diskette with the hexadecimal byte E5. You probably do not care what is done, or how it is done, so long as the diskette is properly formatted.

Be extra careful when formatting hard disks. Normally, such an operation is only done once. Formatting a hard disk erases all information from the disk surface and, since so much data is stored there, it can be a time-consuming process to recover your information if you inadvertently format over it.

Copy—Transferring Information from One Disk to Another

Suppose you want to make a copy of an entire diskette. You could use PIP, as described in Chapter 3, but it is slow and does not initialize a blank diskette. Instead, you normally use a COPY program.

Like initialization, the name of the program used to copy entire diskettes varies depending upon your CP/M implementation. Names vary because Digital Research included neither an initialize nor copy program with CP/M-80 (Digital Research does include a COPYDISK utility with CP/M-86). Efficient initialize and copy programs must be written specifically for each hardware system; manufacturers and distributors generally provide custom versions of these two programs. Among the various copy program names you may encounter

COPY	DISKCOPY	COPYDISK
DSKCPY	BACKUP	COPYATOB

COPY1D	SDCOPY	MCOPY
FCOPY	FASTCOPY	DUPE

Using any of these programs is similar to using the formatting program we discussed.

```
A>DISKCOPY<cr>
SOURCE DRIVE? A<cr>
DESTINATION DRIVE? B<cr>
PRESS RETURN TO BEGIN COPYING<cr>
COPY COMPLETE. MORE? (Y/N) N<cr>
A>
```

Again, most copy programs function similarly, but they may not conform exactly to the prompts in our example. The following steps usually occur:

1. Type the name of the copy program to execute it.
2. Indicate which drive contains the original diskette (the one to be copied FROM; known as the *source* diskette).
3. Indicate which drive contains the diskette you want the copy placed onto (the *destination* diskette).
4. Begin the copying process by pressing CARRIAGE RETURN.
5. When the copy is complete, indicate whether you wish to quit or make more copies.

Check the manuals you received with your CP/M to determine whether or not your copy program requires an initialized diskette as the destination diskette. Most do not, but a few do. If you try using the copy program with a blank diskette and get an error message, try initializing the diskette first. Most good copy programs initialize the diskette.

MOVCPM—Adjusting CP/M to Memory Capacity

When you first receive your CP/M-80 diskette, it is usually ready to operate in a 16K or 24K computer. This means you need only 16 or 24K bytes of RAM to use CP/M-80. You will quickly find this is not enough memory to execute most programs, especially when using a higher-level language like BASIC or Pascal. In this case, you need to make CP/M-80 aware of the maximum amount of memory available in your system. If you have 48K bytes of memory and you receive a CP/M-80 expecting only 24K, you waste 24K bytes of memory. CP/M-86 does not require MOVCPM, as it normally uses the lowest portion of available memory, leaving the rest available to you.

The MOVCPM command provides a simple method of changing CP/M-80's memory size expectations. Changing CP/M-80 to expect a different quantity of memory is called "moving." Two ways can be used to move CP/M-80.

- Move CP/M and immediately execute it, but not save it on diskette
- Move CP/M and save the new configuration on diskette.

The first possibility is perfect for the day you borrow an extra 8K bytes of memory from another system to learn what you can do with more memory.

To move CP/M and immediately execute (use) it, type

```
MOVCPM<cr>
```

Makes use of all existing memory

or

```
MOVCPM# <cr>
```

Where # is the decimal number of kilobytes of memory you want CP/M-80 to recognize. Use this command when you want to reserve some room for a special program or utility above CP/M.

For example, if you want to create and execute a 48K CP/M-80 system (one that makes use of 48K bytes of your available memory), type MOVCPM 48<cr> or MOVCPM<cr>, if you have only 48K bytes of RAM memory. After a few moments, you see a message followed by the A> prompt; you are now using the 48K CP/M system you created.

Most of the time, however, you will want a more permanent solution. To move CP/M-80 and then save it to diskette, type

```
MOVCPM * *<cr>
```

Makes use of all existing memory

or

```
MOVCPM # *<cr>
```

Where # is the number of kilobytes of memory you want CP/M-80 to recognize.

The "#" in the above examples must be

- A decimal number between 16 and 64 inclusive for CP/M-80 versions 1.3 and 1.4, or
- A decimal number between 20 and 64 inclusive for CP/M-80 versions 2.0 and newer, and
- Less than or equal to the number of kilobytes of memory in your computer.

Saving your new sized CP/M system on diskette requires another step. MOVCPM prompts you for this step by displaying

```
READY FOR ''SYSGEN'' OR
''SAVE 32 CPM##.COM'''
```

This cryptic message means that in order to save the CP/M you just created you must type SYSGEN<cr> or SAVE 32 CPM##.COM ("##" is the size of the new CP/M-80 system given in the MOVCPM command above). Unless you are an experienced CP/M-80 user or a brave soul, you should immediately execute the SYSGEN program by typing SYSGEN<cr> (see the next section).

MOVCPM is sometimes called another name, such as CPM, NEWCPM, MAKECPM, CPMGEN, or MOVECPM. MOVCPM does not affect any files on a diskette.

If MOVCPM does not work properly your computer may "die" (fail to operate properly). Press the RESET button to recover. If this happens, it means that the portion of CP/M-80 that customizes it for a particular computer (the BIOS) has not been placed in the MOVCPM program (*see* Chapter 7), or you specified a greater amount of memory than your system contains. This last possibility can often be overlooked; the Osborne 1 computer, and most computers that use memory-mapped video, uses the top 4K of memory for video displays, meaning you can only create a 60K CP/M-80 system, not a 64K one.

SYSGEN—Placing the CP/M-80 System On a Diskette

SYSGEN is short for *system generation*. System refers to the CP/M-80 operating system. Placing a copy of the operating system on a diskette is called system generation. This is the purpose of SYSGEN.

Why not copy the CP/M-80 system just as you would copy a file? Because the system is not stored as a file. The details of this oddity are explained at the end of this SYSGEN section. For now, remember that SYSGEN copies a CP/M-80 operating system from one diskette to another.

System generation is accomplished in one of the three following ways:

1. By using SYSGEN alone to copy the system from a diskette, without change, and place it on a new diskette.
2. By using MOVCPM and then using SYSGEN to place the new system on a diskette.
3. By using MOVCPM, saving the result on diskette, then reloading the result for modification by DDT, then saving it on a diskette using SYSGEN.

Modifying the system is described in Chapter 7. Here we will first describe the use of SYSGEN alone, and then describe its use in conjunction with MOVCPM. SYSGEN does not affect any file on either the source or destination diskette.

When copying the system from one diskette to another, your dialog with the computer looks like the following:

```
A>SYSGEN<cr>
SYSGEN VER 2.2
SOURCE DRIVE NAME (OR RETURN TO SKIP) A
SOURCE ON A:, THEN TYPE RETURN <cr>
FUNCTION COMP_ETE

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) B
DESTINATION ON B:, THEN TYPE RETURN <cr>
FUNCTION COMPLETE
```

```
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <cr>
A>
```

Let us examine each step a little more closely. First, you type SYSGEN<cr> in order to load and execute the SYSGEN program. It requests the source location; you type the letter "A." That tells SYSGEN to get the system from the diskette in drive "A."

Next SYSGEN asks where to put the system; you specify drive "B." SYSGEN then tells you to put a diskette in drive "B" (DESTINATION ON B:) and press the CARRIAGE RETURN key when you are ready. After you press the CARRIAGE RETURN, the drives whir and clack for a few moments while the system is written onto the diskette in drive "B," then the process starts over (DESTINATION DRIVE NAME...). A CARRIAGE RETURN ends the process, or another drive specifier tells SYSGEN that you wish to place the same system onto another diskette.

If you want to save a different-sized CP/M-80 system you created using MOVCPM on the diskette in drive "A," you could specify this drive instead of drive "B." But when you press the CARRIAGE RETURN key at the end of the program you may discover something is wrong. Usually you must exit from SYSGEN using the diskette with which you first started the system.

When saving the new size system you created with MOVCPM, your dialog with SYSGEN differs slightly. Assume you type MOVCPM * * <cr> or MOVCPM ## * <cr>, and now MOVCPM reports that you are ready for a SYSGEN or a SAVE. Proceed as follows:

```
READY FOR ''SYSGEN'' OR
''SAVE 32 CPM##.COM''
A>SYSGEN<cr>
SYSGEN VER 2.2
SOURCE DRIVE NAME (OR RETURN TO SKIP) <cr>
DESTINATION ON B:, THEN TYPE RETURN <cr>
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <cr>
A>
```

As the example shows, SYSGEN continues to ask for a destination diskette until you respond with a CARRIAGE RETURN only; you can update many diskettes in one session.

The SYSGEN command can also include the name of a CP/M-80 system that has been saved as a file, for example, SYSGEN CPM48.COM<cr>. This form of the SYSGEN program loads the system file into memory and then asks only for the destination drive.

Why SYSGEN Is Necessary

The SYSGEN procedure requires further clarification.

The first two tracks—tracks 0 and 1—of every CP/M-80 diskette do not contain

files. Instead, this 6656-byte space is reserved for bootstrap (cold start) loaders and the CP/M-80 operating system. These programs are not stored as files; they do not appear in the file directory; nor are they accessible as files. However, it is sometimes necessary to read or write to these tracks.

Diskette tracks 0 and 1 are always set aside, whether or not they contain the loader and system programs. It is not necessary to keep these programs on every diskette. They are required on a diskette only if you do a cold or warm start with it.

Most diskette copying programs copy the system, or lack of it, from one diskette to another. Diskettes that contain application programs you receive from software companies rarely, if ever, contain CP/M-80 on the system tracks.

That is why a special program, SYSGEN, is needed to place a new or modified operating system on diskettes. Only by using SYSGEN can you save your changed CP/M-80.

One last word on the subject of system generation: MOVCPM moves only the raw, bare bones of the CP/M-80 operating system. Any additional special printer drives or other changes you have made to the BIOS section of CP/M (*see* Chapter 7) are not moved by MOVCPM. If you use MOVCPM and then the printer (or any other special device you may be using) refuses to work, you might need to add the special drives for those devices again. SYSGEN, on the other hand, always copies the entire operating system, including all of BIOS. In fact, SYSGEN copies all of tracks 0 and 1.

Generating a New CP/M-86 System

Thus far, we have discussed new systems only in the context of CP/M-80. CP/M-86 operates in a different manner than CP/M-80, thereby differing in the way in which a new system is created.

CP/M-86 loads the operating system from a file saved on diskette named CPM.SYS. In order to properly load CP/M-86, however, a cold start loader program must be resident on the first two tracks of your CP/M-86 diskette. Since the task is complex and should only be performed by someone familiar with 8086 assembly language programming, we will simply outline the steps involved in creating a new loader program.

1. Assemble a special version of your BIOS with the name LDBIOS.A86.
2. Concatenate the resulting file with LDBDOS.H86 and LDCPM.H86 by typing

   ```
   PIP LOADER.H86=LDCPM.H86,LDBDOS.H86,LDBIOS.H86<cr>
   ```

3. Generate a command file (type CMD) for the resulting file

   ```
   GENCMD LOADER 8080 CODE[A400]<cr>
   ```

   (The above example creates an 8080 memory model system beginning at the absolute address of 400 hex.)
4. Use DDT or DDT-86 to place a copy of the LOADER.CMD file in memory, and use SYSGEN or LDCOPY to place a copy of the CP/M-86 loader on the system tracks of your diskette.

When you start your CP/M-86 system, the loader on the system tracks is placed in memory and executed, an act that results in the file CPM.SYS being loaded into memory and executed. The organization of the CPM.SYS file is as follows:

- 128-byte header identifying load parameters
- CCP.CMD
- BDOS.CMD
- BIOS.CMD

Creating a CPM.SYS file is similar to the process just described for the system loader.

1. Assemble your BIOS file (BIOS.A86).
2. Concatenate the resultant file with CPM.H86 into

```
PIP CPMTEMP.H86=CPM.H86,BIOS.H86<cr>
```

3. Convert the resultant file into a command file

```
GENCMD CPMTEMP 8080 CODE[A40]<cr>
```

4. Rename the resultant file for use

```
REN CPM.SYS=CPMTEMP.CMD
```

CPM.SYS should be the first file on a CP/M-86 boot diskette.

HIGH-LEVEL LANGUAGES

As we mentioned earlier in this chapter, high-level languages are one of the building blocks that program developers use. They write computer programs in a high-level language for many reasons. The most frequently mentioned reasons are

- High-level languages are easier to use since one high-level instruction conveys the meaning of many machine language instructions. Programming is faster and programs are easier to understand.
- It is easier to conceptualize the executed process when the command resembles human communication. For example, PRINT is understandable in both human and computer languages.
- There are high-level languages that have been designed for particular computerized tasks (computer control of machines, emphasis on numerical calculations, and so on).

Inclusion in or exclusion from this chapter of a particular language does not reflect an endorsement or condemnation of the software by the author or publisher. The relative merits of any given package are not to be implied from any discussion within this chapter.

History of High-Level Languages For CP/M-80 and CP/M-86

Before describing the details of several popular high-level languages, a brief history of the development of high-level languages using CP/M is necessary.

CP/M became a quasi-standard operating system because it was one of the first operating systems available. An operating system acts as the scheduler and arbitrator of the various tasks a computer must perform. In other words, to use disk drives you must have a disk operating system, but the disk operating system is only a go-between, not an end in itself.

Soon after CP/M-80 became available to the general public, Gordon Eubanks, Jr. released a high-level language called EBASIC that he developed as part of his doctoral dissertation. (EBASIC is sometimes called BASIC-E.) EBASIC was written using another high-level language called PL/M. While other languages could be used with CP/M-80, EBASIC became available relatively early and made good use of CP/M logical and physical device handlers. (See the section on devices in Chapter 3.) EBASIC was written using government facilities, and this automatically placed the software in the public domain. No copyright privileges could accrue to Eubanks, nor could EBASIC be sold for more than a "reasonable copying charge." In fact, the PL/M instructions (called source code) that comprise EBASIC are available from the CP/M Users' Group (*see* Appendix G).

EBASIC is a type of compiler for BASIC. You enter BASIC language instructions using an editor, then EBASIC creates an intermediate file of instructions. But let us back up a step and describe what happens inside the computer when a program or high-level language is executed.

The computer executes only machine language instructions, those 1's and 0's that keep popping up throughout this book. BASIC is a computer language developed at Dartmouth to help beginners use computers; it has instructions like

PRINT
GOTO
LET
IF

How do these instructions translate into 1's and 0's? In EBASIC, when you have entered a set of BASIC instructions in a file with an editor, you have a file of text. We call this file the program *source code*. The source code consists of recognizable letters and numbers. Using other CP/M-80 programs you can manipulate the source code text. For example, you can use PIP to copy the text to a device like a printer and thus create a printed copy of the text.

On your BASIC diskette you have a file named EBASIC.COM (sometimes abbreviated BAS.COM). Use that file to create an intermediate code file from the source code file. If you have a BASIC program in a file named SOURCE.BAS, type

```
EBASIC SOURCE<cr>
```

The conversion from text to a more compact form that the computer uses occurs automatically. When it is complete, your diskette has a file named SOURCE.INT. This new file contains no text; instead, each BASIC instruction has been compacted into a one-byte representation. For instance, a PRINT instruction from the source code file (SOURCE.BAS) is stored as hexadecimal 1A in the intermediate code file (SOURCE.INT). A number of advantages result from this compacting process (often called *compiling,* although a true compiler generates real machine instructions, not representations of high-level language instructions). The most significant advantages are a reduction in the file size and faster execution speed. It takes less time to interpret one byte, such as 1A, than it does to interpret a series of letters, such as the PRINT instruction.

To execute a program written in EBASIC, you use another portion of EBASIC called RUN.COM. This program interprets compacted instructions in the file SOURCE.INT. RUN.COM is loaded into memory and begins execution, then SOURCE.INT is loaded into memory and is interpreted by RUN.COM. While this sounds confusing, EBASIC takes care of the details. However, in order to use RUN.COM you must write (or purchase) an EBASIC program, use EBASIC to create the intermediate file, and use RUN to begin interpretation of the intermediate file.

EBASIC is not well-suited to business software. Nevertheless, because it was one of the first high-level languages available on CP/M-80 and because of its low cost, EBASIC was quickly adopted by those supplying CP/M-80 with systems. While EBASIC is actually an extension of the standard Dartmouth BASIC, it lacks several features of a suitable business processing language.

Fortunately, Eubanks did not stop with EBASIC. Using his experience in writing and developing EBASIC, he formed a small firm and developed CBASIC (and later, CBASIC2). CBASIC is an upward enhancement of EBASIC; many EBASIC programs will run using CBASIC, but not necessarily vice versa. Many program developers have used CBASIC and CBASIC2 to write programs, and you may find application programs that require CBASIC in order to run. (CBASIC, by the way, uses the same three-step process we described for EBASIC. The program used to run the CBASIC intermediate file is named CRUN.)

While EBASIC and CBASIC were developing as one primary high-level language for CP/M-80, two young men from Washington began supplying another version of BASIC out of a company called Microsoft. Microsoft's clients have included MITS—the company that started the personal computer revolution—Radio Shack, Apple, Texas Instruments, Exidy, Ohio Scientific, Osborne, IBM, and other microcomputer manufacturers. What began as a cassette-based BASIC interpreter of modest size has grown into a comprehensive, disk-based language. The primary versions of Microsoft BASIC have been

8K BASIC
: Usually used with cassettes; this version is used in various forms by MITS, Ohio Scientific, Apple, Radio Shack, and Exidy.

Disk BASIC
: Originally developed for MITS but modified for both Radio Shack and CP/M-80 systems.

Extended Disk BASIC
: The current version of the interpreter which is used with CP/M-80 and CP/M-86.

Until 1980, all versions of Microsoft BASIC were interpreter high-level languages, as opposed to compiler languages like CBASIC. Rather than using an editor to enter programs, with an interpreter BASIC you enter programs directly into the computer's memory. The interpreter BASIC has a built-in line editor; type a command to load BASIC into memory (MBASIC <cr>), and the interpreter BASIC accepts valid statements thereafter. These statements are immediately interpreted as you enter them. (Actually, they are interpreted once you press a CARRIAGE RETURN.) Each line is processed when completed.

Again, a compacting scheme saves both space and time in the actual execution of a program. The statement is compacted as soon as it is entered for immediate execution.

Unlike compilers, an interpreter language allows you to develop and test each piece of a program individually. You may at any time stop entering instructions and begin executing the partial program by typing RUN<cr>. If an instruction error is present, the line on which it occurs is identified and a message details the type of error encountered. You can immediately edit the offending line, correct the error, and run the program again.

Compare this to the method of program development you must use with a compiler. You must leave BASIC, return to CP/M, invoke the editor, edit the program file, then recompile it—all before you get another chance to run it. An interpreter performs these same tasks, but the interpreter handles most of the details for you. Microsoft BASIC became popular, even though it largely duplicated features and instructions available from EBASIC and CBASIC.

In 1980 Microsoft released a compiler version of their Extended Disk BASIC. A unique advantage of the Microsoft BASIC allows you to use the interpreter for program development, then compile the final version. Many people believe Microsoft's compiler BASIC provides the fastest execution of any BASIC currently available.

Computers have existed since the 1950's. As computers advanced, so did languages to be used on them. With the introduction of CP/M, many existing languages were adapted for microcomputer systems use.

Microsoft released COBOL and FORTRAN soon after CP/M was available. Since these languages are two of the most popular on large systems, it was logical that they would be two of the first to be developed for operation under CP/M. For nearly two years, CP/M users could choose among various BASIC dialects, COBOL and FORTRAN, but in time, several more languages have become available under CP/M.

While any high-level language can be used to achieve a given task, each approach can be so varied as to make you wonder which to use. To help you deal with this problem, we discuss the reasons to choose one language over another and give you some short profiles to some of the more popular languages.

Choosing a Language

When choosing a language, you should remember the following two rules:

1. Many applications programs require run-time modules to operate. Sometimes vendors supply the necessary software, but most of the time they assume you will supply it yourself. A typical example would be programs written in any of the BASIC dialects, and especially CBASIC. Many excellent accounting programs available today are written in CBASIC, but CBASIC programs need the CRUN module to run. Therefore, before you could run a program written in CBASIC, you would need to purchase the CBASIC language if the vendor did not supply CRUN with the software package you purchased.
2. If you are already familiar with a high-level language, choose a similar language available for operation under CP/M. Why buy Pascal if you are already well-versed in COBOL?

If your expertise with computers is limited, or if you are interested in acquiring another language to satisfy your curiosity or expand your knowledge, read the next section thoroughly. This will help you decide which language or languages will meet your goals.

A Dictionary of Languages

Following are short descriptions of currently available languages for operation under CP/M. Refer to Appendix G for a list of companies currently offering them. This list is by no means complete because companies are continually forming and existing companies keep expanding their lines of available software. Reading trade journals, such as *BYTE* and *Personal Computing*, keeps you current with the latest software available. An important item to remember about Microsoft's software is that, unlike most companies, Microsoft does not sell any of its manuals separately. If you purchase a software package, make sure you backup not only the software but manuals as well.

ADA

ADA is a new language developed by the Department of Defense. It is named after Augusta Ada Byron, who is thought of as the "first programmer." ADA was developed because all other languages evaluated by the Department (Pascal, PL/I, COBOL, FORTRAN, and so forth) failed to meet the needs of military software systems. All programs now used by the Department must be written in ADA. It is a highly structured and sophisticated language usable for all types of programs from simple applications to highly technical systems programming.

Assemblers

In addition to the assembler which Digital Research supplies with CP/M, several advanced assemblers are now available. If you are well-versed in assemblers, a more advanced product could be well worthwhile. If your system is Z80-based rather than an 8080 machine, investigate them because they allow you to program directly in Z80 *and* 8080 codes, letting you take advantage of the Z80's more powerful instruction set.

BASIC

BASIC stands for Beginner's All-purpose Symbolic Instruction Code. This language was developed by Dartmouth College in the 1960's to teach computer use and programming. Although BASIC was partially derived from FORTRAN, it has a simpler syntax; BASIC is slightly more understandable to the casual program reader.

Unfortunately, it is difficult to generalize about BASIC. There is an American National Standards Institute (ANSI) definition of the BASIC instruction set, but this standard was defined after the language had established itself as the most popular microcomputer programming language. By the time the standards were set, almost every extant version of BASIC deviated from the standards. In fact, language developers frequently add instructions from other languages to BASIC (most notably from Pascal) to overcome deficiencies.

Reduced to simplest terms, here are some advantages of BASIC.

- It is widely available; almost every microcomputer uses a version of BASIC.
- It is easy to learn. Hundreds of books provide introductions to programming in BASIC. Many schools use BASIC for introductory programming courses, and most computer retailers offer instruction using BASIC.
- It is simple to understand; BASIC uses words and phrases similar to English words. As Radio Shack, Commodore, and Apple have shown in computer sales to grade schools, with proper instruction children can learn the commands and write programs.
- A great number of the programs published in computer magazines are written in BASIC, thus providing samples of programming style and logic to the BASIC user.
- Most BASICs are interpreters; you can type in instructions and immediately see the results. That makes BASIC an excellent learning tool.
- There are more BASICs available than any other language (at least for CP/M systems). You will probably find one dialect which suits your style and personality.

Some of the disadvantages of BASIC are

- There is no standardization. While the frequently used instructions remain

essentially constant in all BASICs, there are many extensions and added instructions. Extensions may please individual programmers, but they frustrate standardization.

- BASIC programs execute slowly in comparison to other languages. Notwithstanding occasional compiler versions of BASIC, the fact that most BASICs are interpreters immediately adds an extra level of execution (the interpretation). Even BASIC compilers generate programs that execute slowly. The structure of the language and the resulting program do not make good use of memory space.
- You can write a sloppy BASIC program and still have the program work. BASIC does not impart much structure to programs. Unfortunately, the slack BASIC allows in programming style encourages programming "off the top of the head."
- Standard BASIC has not kept pace with other developments in the computer industry. BASIC is over 20 years old, and computers have changed radically in that time. The versions of BASIC which accommodate these changes inevitably suffer from not being standard.

Overall, BASIC is probably the most approachable language available to CP/M users. The number of programs written in BASIC, the number of books written about it, and its wide application all suggest you will find help in understanding the language if you need it.

C

Bell Laboratories developed C; it is an integral part of the Bell operating system, UNIX. Currently there are a few C's available for CP/M, and more will be developed as the language becomes more popular.

Several languages are relatively close: C, ALGOL, PL/M, and Pascal. These four languages all require *structured programming techniques*. In terms of instruction execution order, structured programs are easy to trace and are not easily written off the cuff.

The language C differs from these other languages in a number of ways. C often uses abbreviations or shorthand instructions where Pascal and the others use complete words and phrases. For instance, C uses the instruction

```
INT X;
```

Pascal expands that same concept in an instruction like

```
X: INTEGER;
```

In addition, C demands more understanding and memorization by its users but requires less typing.

Despite being exact, C is not difficult to learn. C clearly excels over other languages in the instruction set extensions which make it capable of many tasks normally reserved for assembly language.

Users can develop modules with C that can be used repeatedly without reentering

instructions. This can be done both internally in a program (using user-defined "procedures") or externally (using the #INCLUDE function built into C). Repetitive tasks do not require the same amount of programming effort each time they occur.

Above all C is transportable. Since it was specified and created by Bell Laboratories, it does not suffer from innumerable versions. Your C programs are likely to run on a larger variety of microcomputers than programs written in any other language.

Among its other advantages, C executes programs extremely fast and creates extremely compact program codes (it is, in fact, a compiler). Its features allow easy access to peripheral devices at a machine language level.

The primary disadvantage of C is its newness. Few resources are available for learning C. It is so new to the CP/M environment that only a few application programs are presently commercially available in C, nor have any of the primary computer magazines presented more than a few examples of C programs.

COBOL

COBOL is an acronym for Common Business-Oriented Language. COBOL program statements are much like spoken English. In fact, programmers refer to COBOL statements as *sentences* and use a number of predefined words.

A short section of COBOL program code looks like the following:

```
PROCEDURE CALCULATION.
DETERMINE-COST.
   COMPUTE OUR-COST=LIST-PRICE - 10.05
   IF OUR-COST > ZERO
      SET PRICE-TO-US TO OUR-COST
      MOVE ''OK'' TO VALID-PRICE-CODE
   ELSE
      MOVE ''NO'' TO VALID-PRICE-CODE.
```

Notice how the sentences end with a period. By reading the sentences aloud, you learn what the program does. COBOL has little unintelligible computer code built into the language.

The origin of COBOL is unique in the computer world. COBOL is a committee-designed computer language. A group named CODASYL (Conference On Data Systems Languages) established the COBOL language and also defined a standardized database structure for business computing. Unlike the other languages described in this book, COBOL is extremely standardized. A program written in ANSI COBOL (the standardized definition) should run on any computer utilizing the language. The United States government recognizes the standard COBOL version and periodically officially recognizes COBOL versions as meeting the accepted minimum standards. At present COBOL is one of the few languages the U.S. government checks to assure a standard definition is met.

Since COBOL was developed with business applications in mind, you would expect it to be used primarily by businesses. This is true for the larger computer

environment, but several factors have minimized COBOL's impact in the microcomputer world.

First, COBOL is a large language. COBOL uses full sentences to indicate the task to be performed, and it is a carefully structured language. Statements must appear in a certain order, and a number of system specific statements must also be included. In consequence, COBOL is not well-suited to microcomputers. This was especially true when memory was a relatively expensive component of microcomputer systems. With the decrease in memory prices, 48K- and 64K-equipped microcomputers are commonplace, and several microcomputer versions of COBOL have appeared.

COBOL is not very efficient; a microcomputer CPU takes a long time to decipher its lengthy program statements alone. This applies primarily to compilation rather than program execution. In addition, processing usually makes COBOL a relatively slow language compared to others. On larger computer systems, there was plenty of execution speed to spare and tasks were often handled in a batch mode rather than as interactive processes. Microcomputers, however, are often taxed by the number of internal manipulations COBOL makes.

COBOL was designed at a time when interactive computing was relatively unknown. It cannot efficiently utilize the console device in a CP/M system. Both Microsoft COBOL and CIS COBOL, the two most popular microcomputer versions of the language, include extra (and thus non-standard) instructions to utilize the high-speed console devices featured on most available microcomputers.

FORTH

FORTH is one of the more misunderstood microcomputer languages. A relatively new language (invented in the early 1970's by Charles Moore), FORTH has been described as everything from "an assembly language like BASIC" to "a religion."

A FORTH program is not easily deciphered; it is a *threaded* language. You use its basic building blocks to make larger ones. In fact, it is extremely difficult to describe just how to program in FORTH. Consider the following program (reprinted from *BYTE,* August 1980, page 158):

```
 0 (BREAKFORTH/MMSFORTH, BY ARNOLD SHAEFFER, PART 5 OF 6)
 1
 2 : CLR
 3  XPOS @ 2- 124 AND 2+ DUP 4 + SWAP DO YPOS @ I DCLR LOOP
 4  YPOS @ 27 - ABS SCORE+! 0 32 PTC SCORE ? BOP
 5  YDIR @ MINUS YDIR !
 6 ;
 7
 8 : BALLCHKY DIR @ YPOS+! XDIR @ XPOS +! XCHK YCHK PCHK
 9                  YPOS @ XPOS @ D? IF CLR THEN
10 ;
11
12 : BALL YPOS @ XPOS @ DCLR
```

```
13                BALLCHK DUP 0= IF YPOS @ XPOS @ DSET THEN ;
14
15 : GAMECHK SCORE @ 1800 MOD 0= IF 191 15616 320 FILL THEN;
```

Pretty intimidating, right? FORTH includes many aspects a beginner prefers not to find in a computer language. It frequently uses abbreviations; numeric manipulations are done in Reverse Polish Notation (the same as some pocket calculators); and FORTH allows the user to invent new commands (BALLCHK is one such invention in our small sample).

If FORTH has so many apparent disadvantages, why does it exist? First, it is fast and well-suited to applications which require quick screen or disk manipulations. Second, the machine language code created by FORTH is generally much smaller than other languages (another factor in its speed advantage). And finally, FORTH is the language you make it. The fact that the programmer can extend the language at will is a strong asset; programs can be tailored to the task.

If you are interested in getting a clearer description of FORTH's virtues, see *Discover FORTH.**

FORTRAN

FORTRAN is another acronym; it stands for Formula Translator language. It was designed for complex numerical calculations where speed is the primary factor. FORTRAN is not good at manipulating characters like letters of the alphabet; it does not handle input/output devices efficiently; neither was it designed for interactive use (although it can be modified).

In many ways FORTRAN is the parent of BASIC; many BASIC statements are FORTRAN descendents. The primary differences lie in BASIC's efficient input and output to the console and its ability to manipulate strings. In FORTRAN, the FORMAT statement, which is difficult to understand, must be used to prepare information for transfer to the console or other devices.

FORTRAN, like C and COBOL, is a true compiler. Programs are input using an editor, then reduced to machine instructions. The FORTRAN compiler most often used with microprocessors is the Microsoft version, which is a subset of FORTRAN IV. New programs are rarely written in FORTRAN; it is primarily used to run existing programs in the microcomputer environment.

Pascal

Like C and COBOL, Pascal is a structured language; programs must follow a particular structural concept. Statements must be executed in order, and blocks of program code are identified to make the block easy to use elsewhere. The same block may be used in several programs. BASIC, FORTRAN, and several other languages allow the programmer to jump program execution from one set of statements to another set in an entirely different area. Logical grouping of program

*Thom Hogan, *Discover Forth* (Berkeley: Osborne/McGraw-Hill, 1982).

statements into *subroutines* (sections of code which perform one function and then return control to the calling statement) is possible in BASIC, but not with the flexibility offered in Pascal.

Developed in Switzerland in 1968 by Nicklaus Wirth, Pascal relates directly to Algol. A typical section of a Pascal program might look like the following:

```
VAR HOUR, MINUTES, SECONDS: INTEGER;
BEGIN
  HOURS:=1;
    MINUTES:=1;
    SECONDS:=15;
    REPEAT
      WRITELN('TIME LEFT =');
      WRITE(HOUR,':')
      WRITE(MINUTES,':')
      WRITE(SECONDS,':');
      WHILE SECONDS>=0 DO
        SECONDS:=SECONDS-1;
      END;
      IF SECONDS=-1 THEN
        SECONDS:=59;
        MINUTES:=MINUTES-1;
        IF MINUTES=-1 THEN
          MINUTES:=59;
          HOURS:=HOURS-1;
    UNTIL (HOURS=0) AND (MINUTES=0) AND (SECONDS=0);
    WRITELN('TIME UP!');
END.
```

Consider a Pascal program's appearance. First, it has a definite structure. The indentations are not required, but they are strongly suggested to reflect the structure of the program. Next, like COBOL, Pascal uses *sentences* whose meanings are immediately apparent. Unlike COBOL, however, Pascal has relatively few predefined statements, and the sentences formed tend to be shorter. This means Pascal programs are not quite as unwieldy as COBOL in terms of length, but they still remain readable.

Pascal also has an attribute known as *recursiveness.* To varying degrees, so do Algol, PL/I-80, FORTH, and C. This means a block within the program may call itself, or pass execution back to itself. This is a useful feature for complex procedures which occur repeatedly within a program. Long division, that terrible mathematical tool we all struggled with at some point in our elementary education, employs a recursive function; you apply the same concept repeatedly until you get a zero or repeating pattern. Pascal is well-suited to tasks which encompass a number of repeated procedures due to its recursive nature.

Programmers like Pascal because their conceptions of a program fit directly into

the structural constraints of Pascal. In fact, many programmers use a pseudo-Pascal language to describe the flow of program execution.

```
WHILE RESULT NOT ZERO DO
   COMPUTE NEW VALUE
   IF NEW VALUE > OLD
      THEN PERFORM FUNCTION X
   ELSE
      PERFORM FUNCTION Y
END WHILE
```

The portion of a Pascal program that accomplishes the above logic might be

```
WHILE RESULT NOT 0 DO
   FIRSTNUM:=SECONDNUM-OFFSET;
   NEWVALUE;<-------------------- Passes execution to a routine
   IF NEWRESULTRESULT THEN          which calculates NEWRESULT
            X
   ELSE
            Y;<-------------------- X and Y represent procedures
                                     which are performed each time
                                     they are encountered
END{WHILE}
```

We have discussed the appearance of a Pascal program for a number of reasons. Pascal is frequently hailed as the language of the future because of its easily understood structure. Also, Pascal is a good beginner language because it is straightforward. If you do not try to write Pascal programs off the top of your head, you will be able to create working programs. Of course, learning to program any microcomputer requires a good deal of dedicated effort and time, so seek expert instruction (many universities offer free or inexpensive courses for beginners in Pascal programming).

PL/I-80

With the introduction of PL/I-80 in mid-1980 by Digital Research, almost every major programming language is now available to microcomputer users. PL/I-80 combines the structure of Pascal, the simplicity of Pascal and BASIC, and the ability to perform complex operations with various peripheral devices.

PL/I-80 and its cousin, Intel PL/M, are frequently described as *system development* languages, or languages used by computer manufacturers and systems integrators to create other software. In fact, EBASIC and a good portion of CP/M were written in PL/M, as were several other high-level languages.

Digital Research's PL/I compiler uses a subset, called Subset G, of the original language. PL/I programs execute fast and are generally relocatable.

Other Languages

RPG (Report Program Generator) was originally developed by IBM for non-programmers. It is less a language than a way of selecting and outputting data from a disk file. The only readily available RPG is marketed by Cromemco; it runs only with the CDOS operating system (*see* Chapter 6). If RPG has been recommended to you, or you wish to find an easy way to create, maintain, and select information from a data base, consider some of the more sophisticated data management systems like Selector (from MicroAp), PEARL (from Relational Systems International) or HDBMS (from Micro Data Base), or Dbase II (from Ashton-Tate).

LISP (List Processing) is an interpretive language developed in conjunction with artificial intelligence research at Stanford University. LISP's primary application is string processing, and it is not particularly well-suited for numeric applications.

APL stands for A Programming Language. Its modest name does not properly reflect the immense power this language possesses. Rather than using words to express a given programming concept, APL uses graphic characters to represent certain commands. This following APL program calculates the average of a list of numbers and demonstrates APL's conciseness.

```
average
''Enter the numbers you want averaged''
NUM ← □
  SUM ← +/NUM
  ANS ← SUM -:- ρNUM
  ''The sum of your numbers is '';ANS
```

Note the use of symbols (∇, □, ←) to represent a concept. One statement, such as NUM ← □, specifies a number of computer operations. The number typed is moved to the variable NUM for each number typed, rather than representing each operation with a statement. BASIC and other languages sometimes eliminate this limitation through statements which repeat the execution of another statement.

Word Processors

One of the most useful application programs you can purchase is a *word processor* or *text editor*. The Digital Research editor (ED.COM, described in Chapter 3) provides basic editing functions. You may need a versatile editor with CP/M-80 and CP/M-86 for the following reasons:

- To create and edit programs
- To create and edit data files
- To perform emergency maintenance on text files
- To create and edit documents (letters, articles, documentation, and so on)
- To format documents and print them.

Some questions immediately come to mind. The first may be "Why do I need to

do all that? I thought the programs I bought would take care of everything I need."

Your computer primarily creates and stores information. A typewriter does the same thing, using paper as a storage media. You would not attempt serious typing without some method of correcting your mistakes, and serious computing also requires a way to correct the information you put on diskette. A good editor performs this function for you.

You could use ED.COM as your editor. Digital Research also sells a companion text formatting program called TEX. ED and TEX do not reflect state-of-the-art editing software, however. Today, a number of extremely sophisticated word processing editors exist that you should investigate.

Magic Wand. Magic Wand (soon to be named Peach Wand and also available in a slightly modified version of SuperWriter) is a no-nonsense, straightforward editor suitable for word processing. Magic Wand does not sacrifice ease of use for a reduced number of features. Magic Wand consists of two modules, EDIT and PRINT.

The EDIT module consists of two modes: *command* and *edit*. To edit a document, you need to learn only a few commands for cursor movement, character insertion and character deletion. The EDIT module is screen-oriented; you see the changes as they are made. A single-key command takes you from the edit mode to the command mode. In the command mode you may specify a number of other instructions to Magic Wand. Most of these additional instructions manipulate information in diskette files or instruct Magic Wand to format and print the information.

The PRINT module prints the document created using Magic Wand. Depending upon the document you wish to print, this session can be automatic or may request information from you before the printing begins.

Magic Wand can incorporate information from separate data files into a document being printed. This is especially useful for generating form letters or for adding a personalized touch to an otherwise standard document. Magic Wand can incorporate names and addresses into a letter and can print envelopes as well as letters.

Electric Pencil. Electric Pencil was one of the first dynamic screen-oriented word processors available for the CP/M-80 operating environment. A number of Electric Pencil versions are available. You need the version designed specifically for the memory-mapped video system in your computer. Electric Pencil will not work with serial terminals.

Once Electric Pencil has been loaded, it presents a copyright message on an otherwise clear screen. You can then choose either the typing, disk, or print mode. Enter the typing mode by beginning to type. Enter the disk mode by typing CONTROL K. Enter the print mode by typing CONTROL P.

Files created by Electric Pencil are not immediately usable as program or data files under CP/M-80 because Electric Pencil inserts CARRIAGE RETURN without line feeds into the text. To use Electric Pencil to create program or data files, you must purchase a program called CONVERT from Michael Shrayer, Inc. A convert program is also available from the CP/M Users' Group.

WordStar. WordStar is a text processing program that emulates dedicated word processing systems. Every now and then a program comes along that is well-suited to computer novices. WordStar is one of those programs.

Once WordStar is loaded into the computer, you may specify the size of the help menu, a constantly displayed list of command choices. Note that just because the program displays excellent on-screen aids is no reason to skip reading the manual.

You must give WordStar information about the operating environment before you use it for the first time. Use the INSTALL program supplied with WordStar. If you have "standard" equipment, the installation process is straightforward. If your equipment is not listed as one of the predefined choices available under INSTALL, then consult your computer dealer or WordStar vendor.

This book was written using WordStar. Three features dictated the choice of WordStar: the ability to see the text format on the screen exactly as it would print out, the ability to perform the number of manipulations necessary for numerous major revisions, and the fact that the publisher can typeset directly from the author's diskettes.

SELECT, Benchmark, Memorite, Electric Blackboard. A number of other good word processing programs have appeared since the first edition of this book. Each has its good points, and you should investigate each before making a choice of which one to purchase.

SELECT's major advantages are its ease of use and its built-in self-teaching abilities. In other respects, SELECT is very much like WordStar. Benchmark also has many features similar to WordStar, and some users prefer its simpler editing commands. Memorite is an extremely flexible and powerful word processor, but it is restricted to use with Vector Graphic systems. Electric Blackboard is unique in that it allows the user to divide the screen into multiple "windows," which can be manipulated individually. In addition, Electric Blackboard is probably the best of the word processors to create text wider than the normal 65-character margins on $8\frac{1}{2} \times 11$ inch paper.

CHAPTER

6 MP/M, CP/NET, and CP/M Derivatives

The immense popularity of CP/M has spawned a number of similar operating systems. Like Cromemco's CDOS, some of these systems are direct descendants of CP/M but are not totally compatible with it. Other "work-alikes," I/OS and TP/M, for example, retain CP/M compatibility and claim to improve CP/M features. Digital Research expanded CP/M capabilities with MP/M and CP/NET.

CP/M sales exceed those of all other microcomputer operating systems, including Apple DOS and Radio Shack TRSDOS. CP/M-80 is available for both the Apple and Radio Shack computers, and CP/M-86 is now available with an add-on 8088 processor board for the Apple II computer.

The popularity of CP/M has generated a growing library of CP/M-compatible software. Compared to application programs for other operating systems, CP/M-compatible software was developed largely for business applications.

You may have consciously chosen CP/M, you may not have had a choice, or you may simply have bought another operating system because you felt it offered additional useful features. In any case, learn the relationship between your operating system and CP/M to determine if your computer will run CP/M-compatible software.

The degree of compatibility between CP/M and other operating systems varies. CDOS cannot be used on an 8080-based system since CDOS includes some Z80 instructions. In addition, CDOS has several extensions, and programs that utilize these extensions are not usable on a CP/M-80 system. Other operating systems, like I/OS and TurboDOS, claim to correct several CP/M "faults" but otherwise retain compatibility with it. Some, like 86-DOS, are not really compatible since they utilize a different file format than CP/M-80 and CP/M-86.

Erroneous assumptions about the compatibility of operating systems and application programs cause problems for everyone. This chapter clarifies compatibility between selected operating systems and discusses the features of MP/M and CP/NET, two Digital Research extensions of the CP/M family.

MULTIUSER AND MULTITASKING SYSTEMS

Up to now, we have described computer systems that perform one task for one user at a time.

Multitasking computer systems perform a number of chores concurrently. Other names that refer to this multitasking computer setup include multiuser, time-sharing, time-slicing, multiprogramming, and multiterminal. Unfortunately, the microcomputer industry changed some definitions that had previously referred to minicomputer and mainframe computer systems. In this text, multitasking refers to a microcomputer system performing two or more jobs simultaneously.

How does a multitasking computer system work? In a normal computer, a sequence of events occurs for each task. For example, when you press a key on the keyboard, the following (simplified) sequence occurs:

1. You press the key
2. The keyboard sends the character to the computer
3. The I/O device within the computer captures the character
4. The CPU gets the character from the I/O device
5. The CPU processes the character according to a set of instructions already in memory.

While steps 1 through 3 take place at one keyboard, the CPU could perform step 5 (processing a character) for a second keyboard. If the CPU is waiting for the character from the first keyboard, it is idle.

Multitasking, in its simplest form, requires the CPU to perform a second job while it waits to continue processing for the first job. The computer appears to perform two things simultaneously, since the tasks outlined above are measured in thousandths or millionths of a second.

Were everything as simple as our description, all computer systems would be multitasking devices. We must address one problem to fully describe the concept of multiple jobs on a single computer. The problem simply stated is: how do you know the two tasks will not interfere with each other? How does the CPU know when to work on which task?

The answer is that computer designers cheat. In a multitasking system, the designers make some assumptions and take one of two conventional approaches.

- Each user is given a *time slice*. If the slice of CPU time is a particular length, the two users of the system are not aware of the other's presence on the

computer. Nor do they notice any appreciable slowing of the system since the computer wastes time waiting for the user to do something. Multitasking has certain advantages. For I/O bound systems where time is spent on input to and output from various devices—for example, a word processing system—the computer waits for keys to be depressed; this is clearly a good candidate for the time slice system.

- The CPU can also switch jobs each time it must wait for another device. A related method processes one job until another job demands CPU attention (this is called an *interrupt-driven* system). The differences between these two methods are more subtle than they first appear.

These approaches may or may not be satisfactory methods for dividing CPU time. If the CPU rarely waits for I/O in one job, the second job may not be activated for quite some time. On the other hand, if one device constantly demands attention, the other devices are ignored. The result is the same: one task gets an advantage over the others.

These methods have been inaccurately described as *polling* systems. Polling means checking to see if a device is ready. A time slice system may include polling. It is possible to implement a multitasking system without polling, however.

Microcomputer multitasking systems often combine these two methods. Each user has a time slice, but if the CPU must wait during that time slice, or if another device demands attention, the CPU alternates to another job before the time slice ends. This solves any problems that result from unequal demands on the CPU by either the devices or the operator.

One way in which jobs demand attention from the CPU involves *interrupts*. When you press a key, a character needs to be processed; your input must reach the CPU before you release the key. A special line monitored by the CPU is triggered, and the CPU immediately switches to the routine that fulfills that process. When this routine is done (the *interrupt task*), the CPU returns to the *interrupted* task.

A method of "pseudo-multitasking" involving multiple connected computers that share devices is known as *networking*. This concept involves two considerations.

- The computers must be linked by a physical means (like a telephone connection or a simple cable) and a software means. The software means is the *protocol*.
- The method in which computers are linked together depends on the interfacing capabilities of each computer in the network. In some instances all computers can use all devices on all other computers. In other systems a *hierarchy* operates: one machine acts as a host to several others, or only certain linkages are allowed.

With this introduction to multitasking, we now turn our discussion to the features of MP/M and CP/NET.

MP/M

In general, this discussion pertains to MP/M II; however, users of the earlier MP/M version will find most of the information useful as well.

The letters MP/M stand for Multi-Processing Monitor Control Program. MP/M is an operating system that can control more than one console terminal and more than one program at each terminal. Thus, several users can each run several programs "simultaneously" on one computer.

Differences Between MP/M and CP/M

You will notice three general differences between the operation of MP/M and CP/M: the prompt is different, there are new control characters, and there are new commands. Let us examine these differences between MP/M and CP/M.

1. *The prompt.* Like CP/M, MP/M displays the drive identifier of the currently logged drive (A, B, C, and so forth) followed by a >, but MP/M also includes the currently logged user number (0 to 15, inclusive) before the drive identifier.

CP/M prompt	A>Drive A is the default
	B>Drive B is the default
	C>Drive C is the default
MP/M prompt	0A>User 0 on drive A
	4F>User 4 on drive F
	3D>User 3 on drive D

Each user number is associated with a group of files on disk. Enter the USER command to switch to another user area on the disk.

2. *Extra control characters.* MP/M recognizes several control characters that CP/M does not. These are

^D Detach console from current job
^Q Restart console after ^S pressed
^Z End input from console

CONTROL-D lets you "detach" the console from a job. This is useful when a job requires little or no input from the console or when you wish to suspend one job while starting another. You may reattach a job by typing ATTACH <cr> in response to the MP/M prompt.

The characters ^S stop the console display, just as they do in CP/M. However, unlike CP/M, only a ^Q will restart the console after using ^S.

CONTROL-Z ends the input from the console device. You will rarely need ^Z. When the console functions like a disk device ^Z sends an *end-of-file* marker.

3. *Additional commands and utilities.* MP/M has the following new commands:

DIR[SYS]
  Displays a directory including system files
ERAQ
  Enables a query mode for erasure of files
CONSOLE
  Displays the console number
DSKRESET
  Enables user to change diskettes
GENHEX
  Creates a hex file from a COM file
PRLCOM
  Creates a COM file from a special PRL file
GENMOD
  Creates a PRL file from a special HEX file
SPOOL
  Sends printer output to a spooling device
STOPSPLR
  Stops spooler output
TOD
  Sets or displays time and date
SCHED
  Schedules a task to be run automatically
ABORT
  Aborts a task, even if detached from console
ATTACH
  Reattaches a detached job
MPMSTAT
  Displays MP/M status
PRINTER
  Selects printer to be used
SDIR
  Displays directory and options
SHOW
  Displays disk status
SET
  Sets disk and file status, passwords, and time stamping

Detailed descriptions of these new commands follow in the next section.

MP/M Commands

When you purchase MP/M you receive a number of programs unique to MP/M. All MP/M commands, except for the control characters, are transient commands or programs. We will briefly summarize each MP/M program.

DIR[SYS]

Type the DIR *.* command followed by a space and an "S" with MP/M 1.1, or type DIR[SYS] with MP/M 2.0, to receive a directory of files that includes *system* files. System files would not otherwise appear in the directory display. Normally, all program files are made system files in MP/M so that all users have access to them.

In addition to the normal DIR commands, MP/M 2.0 recognizes the following:

DIR [G8]
: Displays directory of user 8

DIR file,file
: Displays directory matching the file specifications of the first file and the second file. Each additional specification must be separated by a comma.

ERAQ

The ERAQ command is an extension of the CP/M ERA (erase) command. When you use ERAQ instead of ERA, MP/M asks about each file; the "Q" stands for *query*. For example,

```
0F>ERAQ *.*<cr>
F:IRAN     OIL? y
F:SAUDI    OIL? n
F:ISRAELI  MEN? n
F:AFGHAN   OIL? n
0F>
```

In our example, we erased IRAN.OIL but not the other three files on the diskette. Using the ERAQ command instead of ERA decreases accidental erasures.

CONSOLE

There can be as many as 16 independent console devices in MP/M. These terminals are numbered console device number one, console device number two, and so forth.

The console number is distinct from the user number, and the distinction is important. One fixed console number is assigned to each terminal. The user number, on the other hand, is associated with a group of files on diskette.

To see which console you are using, type

```
CONSOLE<cr>
```

MP/M replies

```
Console = x
```

where "x" is your console number.

DSKRESET

Unfortunately, several users may share one or two disk drives. What if one user must change the disk in the drive to access a different disk, or to load another program? Remember, after changing diskettes in CP/M you must perform a warm start. With MP/M you want to perform that warm start for your job only, not for everyone else's.

DSKRESET selectively changes diskettes. Type the command with no other parameters to reset all disk drives on the system. Follow the command with a list of valid drive identifiers (A:, B:, and so on), each separated by a comma, to reset only the drives you specify. For example, the command

```
DSKRESET E:,N:,D:<cr>
```

resets drives "E," "N," and "D."

MP/M does not allow you to reset a drive if some task is using files on that drive. When that happens, you see

```
Disk reset denied, Drive x: Console y Program z
```

This message informs you which console and program still have open files on that diskette. If your console is specified, complete your use of the program (at least finish using the files it is addressing) and then retry DSKRESET. If your console is not listed, you must wait until the other user finishes before you can reset that disk drive, or check for another available drive.

You *must* use DSKRESET before removing a diskette from the system or you may cause another user to have unrecoverable errors.

GENHEX

This program is the opposite of LOAD; it reads a COM-type file and creates a "HEX" type file to be used either by LOAD or GENMOD.

PRLCOM

Files destined for MP/M with the "PRL" file type may be changed to files usable on CP/M-based systems (for example, "COM" type files) by using PRLCOM. The format of this command is

```
PRLCOM prltype.fil comtype.COM<cr>
```

This command line violates the common NEW=OLD convention of most computer commands.

GENMOD

With CP/M, LOAD changes the HEX file created by the assembler to a COM type file to be loaded and executed by typing its name. MP/M locates an individual user program at the beginning of the memory dedicated to that user, but COM files always load beginning 0100 hex. GENMOD converts a "HEX" type file into a "PRL" type file that can be loaded at some other memory address.

GENMOD reads a file consisting of two assembled versions of a program (each assembly with a different beginning location) and creates a file with the PRL file type. Two assemblies are required at different beginning locations since GENMOD compares the two resulting sets of object code to ascertain where specific addresses appear in the program. Since MP/M does not assume any one beginning location for a program, it identifies portions of the program that use specific locations so they can be changed to reflect the memory area where the program is loaded.

SPOOL

Several users on a system may share a single printer. Instead of sending your information directly to the printer, with MP/M you normally send your output to a disk file, and then use SPOOL to print the file.

SPOOL maintains a *queue* of files to be printed. If user 1 asks for a file to be printed using SPOOL, and user 2 asks for some printed report moments later, SPOOL will first finish printing user 1's report, then print the information for user 2. When you invoke SPOOL, it summarizes the queue and then returns the MP/M prompt to your console. You are free for other jobs; the printer automatically prints your document according to the queue.

To get your files into the spooler queue type

```
SPOOL filename.typ, filename.typ <cr>
```

You may specify any number of files to add to the queue, each separated by a comma, up to the limitation of the command line length.

STOPSPLR

To stop the SPOOL function type STOPSPLR followed by your console number. If you have any files waiting to be printed, they will all be removed from the queue. This is a drastic action. It may result in a partially printed report. Be sure you want to stop the printing of your files.

TOD

MP/M maintains a clock to track both date and time. Do not assume its complete accuracy. First, if the computer shuts down for any time period—even a fraction of a second—MP/M restarts the clock. Second, the clock's accuracy depends on how your MP/M system was implemented. If you are using a Digital Microsystems computer or an Intel MDS-based system, the clock functions exactly as Digital Research intended. Other implementations may or may not function properly.

To set the clock, type

```
TOD mm/dd/yy hh:mm:ss<cr>
```

Type in the month, day, year, hours, minutes, and seconds in exactly that order and format. Here is a valid entry

```
TOD 01/16/83 10:00:00<cr>
```

This tells the computer it is the 16th day of January, 1983, and the time is 10 a.m. When you type the above command, MP/M responds

```
Strike key to set time
THU 01/16/83 10:00:00
```

Press any key to begin the clock and return you to MP/M. Now, any time you type TOD<cr>, you see the current date and time as MP/M maintains it. Typing TOD P<cr> continuously displays the date and time until you press another key.

SCHED

A clock allows automatic scheduling of tasks. The computer could perform a task when no one else uses the system. This is especially useful for long reports that might slow the computer's response time. To schedule tasks, type

```
SCHED mm/dd/yy hh:mm task<cr>
```

where task is a valid MP/M command line and the "mm/dd/yy hh:mm:ss" is the exact date and time you wish the task to begin. You cannot specify execution of tasks in increments shorter than one minute.

ABORT

To cancel any task type ABORT task<cr>. If the task was invoked from a different console, you must add the console number from which you scheduled the task

```
ABORT task <cr>
```

MP/M's Internal Structure

MP/M is very similar in structure to CP/M-80. MP/M equipment requirements are

- An 8080, 8085, or Z80 CPU (MP/M-86 uses an 8088 or 8086 CPU)
- At least 32K bytes of memory
- At least one disk drive
- A keyboard for input of characters
- A printer or CRT for output of characters
- A clock timer interrupt.

Consider how CP/M loads into memory (*see* Chapter 1). Load MP/M with a

cold start loader just like CP/M, or use the special MP/M loader—a CP/M transient program—to load and execute MP/M for later use.

The internal structure of MP/M is similar to CP/M. Where CP/M had a Transient Program Area (TPA), Console Command Processor (CCP), Basic Disk Operating System (BDOS), and Basic Input/Output System (BIOS), MP/M has the following parts:

TPA
  Transient Program Area

MEMSEG
  Between 1 and 8 memory segments, one for each user

XDOS
  Extended Disk Operating System

BDOS
  CP/M's Basic Disk Operating System with a bank switching utility added

XIOS
  Extended Input/Output System

Although MP/M adds to the CP/M structure, it does not significantly alter CP/M. Remember, MP/M has been expanded to include 16 user terminals, 16 printers, and 16 drives, each with up to 512 megabytes. MP/M does not support PUNCH and READER devices, as does CP/M. Where CP/M BIOS describes one user terminal, reader, punch, and printer, MP/M XIOS describes terminals and printers for each user. In other words, the concept has not been changed, but the operating system has been expanded to include information about all users. This is one reason why MP/M requires more memory space than CP/M.

MP/M functions differ slightly from CP/M. CP/M always loads a utility program beginning at 0100 hex and stores it in a "COM" type file. MP/M retains this feature but adds the ability to use "PRL" (page relocatable) type files and place/execute them beginning at any interval of 0100 hex (0100, 0200, 0300, and so on).

Suppose two users each have 48K bytes of memory, and each job requires only 20K bytes of memory. If both jobs must start at location 0100 hex there is a major problem: each user would try to use 20K bytes of memory beginning at 0100 hex, and the top 28K bytes would not be used.

Some programs you buy from vendors other than Digital Research come only as "COM" type files. They must be loaded and executed beginning at 0100 hex. Can two users use that program concurrently?

No. But fortunately there is a solution. Most MP/M computers utilize a feature called *bank selectable memory*. The 8080/8085/Z80 microprocessors can directly address only 64K bytes of RAM. The key word is *directly*. Imagine you can tell a memory section to ignore all processes until it receives a special coded signal from the CPU. Another memory section pays attention to all processes until it receives that special coded signal, then ignores the proceedings. Your imagined system is a bank selectable memory system.

TABLE 6-1. An MP/M Memory Map

Bank 1 64K Bytes of RAM	Memory Address (Hexadecimal)	Bank 2 64K Bytes of RAM
	0000	
Reserved for MP/M		Reserved for MP/M
	0100	
User 1 Program (TPA)		User 2 Program (TPA)
	C000	
Not Used		User 3 Program (Optional)
	E000	
MP/M		
	FFFF	

With MP/M, each user usually gets one bank of memory. If you have two *banks* of 48K bytes of RAM in your system, the first user has 48K from the first bank, the second user has 48K in the second bank, and 16K of common memory area in the first bank is reserved for MP/M. A sample *memory map* is shown in Table 6-1. Notice the *addresses* in the middle column. Both user 1 and user 2 use the same addresses for their programs; they do not interfere with one another because MP/M uses only the currently active memory bank. You do not need to tell the system when to change memory banks. This is done automatically by MP/M.

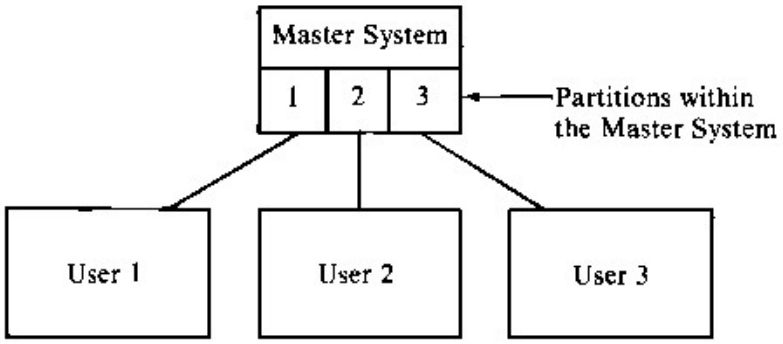
If your memory address begins at an address other than 0100 hex, you must use "PRL" type files rather than "COM" type files to store programs. Digital Research provides its programs as "PRL" type files. In addition, the Digital Research MAC assembler can create "PRL" files from assembly language programs. You can do this with "ASM," but with greater difficulty. Digital Research now includes RMAC (a relocatable assembler), LINK-80, and a library of common routines with MP/M so you can create "PRL" type files.

CP/NET

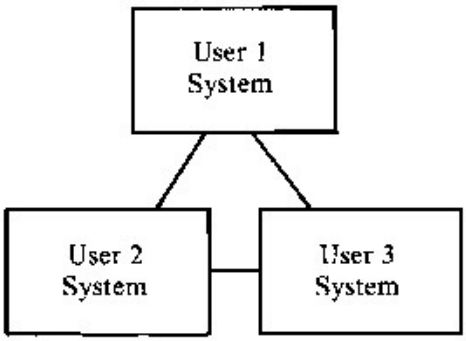
CP/NET is the Digital Research networking addition to the CP/M family of operating systems. CP/M is designed for a single user on a single computer system; MP/M is designed for multiple users on a single computer system; CP/NET is designed for multiple users on multiple computer systems.

The concept of computer networks differs slightly from multitasking. Multitasking tends to be hierarchical.

Each user is a separate entity; multitasking allows little or no interaction between users.

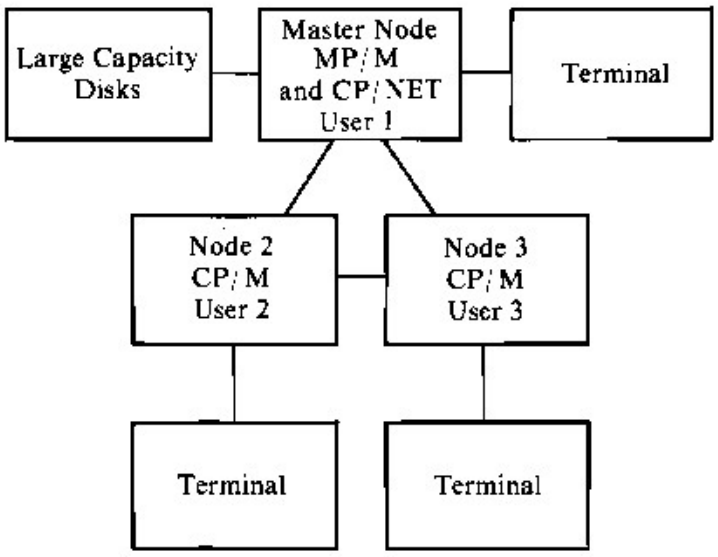


CP/NET operates on a different concept.



Each of the three users are represented as *systems*, not merely users. The logical *linking* between each user is more direct; there is no *master* system.

CP/NET differs slightly from the concept we illustrated. CP/NET requires at least one *node* in the network to act as a master. One user manages the network. The master node must have MP/M and disk drives. Other nodes may consist only of an 8080/8085/Z80 CPU and a minimum of 16K bytes of memory. While these nodes need no other components, a terminal, more memory, and disk drives increase a node's functions. Considering these requirements for a CP/NET system, we will modify our diagram as follows:



There are two primary advantages to a network system of computers: sharing of resources and speed of execution.

The primary shared resource for a microcomputer network is high capacity disk drives. Hard disk drives are expensive. It may be unreasonable to buy three hard disk drives for three computer systems. Large database applications (storage, maintenance, and use of large bodies of data) may require all users to share one common file or group of files.

Another shared resource is usually a printer. Typewriter-quality printers are both expensive and slow. There may be a typewriter-quality printer at one node, while another may have a faster but less readable dot matrix printer. For rough drafts you might use the faster dot matrix printer. But since each user in this network may send output to either printer, you might want to produce final copies with the typewriter-quality printer.

Each node on a CP/NET system can function independently; each node may be a separate computer. Nodes connect to the network only to share a resource in the network. A business might use CP/NET to consolidate several different machines into one system.

CP/NET nodes can use all MP/M or CP/M commands and programs; the exceptions are minimal. In addition, CP/NET includes the following new commands:

LOGIN
  Logs the user into the master node

LOGOFF
  Signs the user off the network

SNDMAIL
  Sends mail (text message) to another user

RCVMAIL
  Receives mail from another user

BROADCST
  Sends mail from master node to all users

MRCVMAIL
  Receives mail from all users to the master node

NETWORK
  Enables one node to use network devices

LOCAL
  Reassigns local devices in place of network ones

ENDLIST
  Sends CONTROL-Z to list device

In the time since its introduction, CP/NET has not attracted a large number of users. This is primarily because many computer manufacturers replace the BIOS in CP/M with a special "networking" BIOS that fools CP/M into thinking that it has the network's resources as its own. A number of microcomputer networks work this

way. The only disadvantage is that the creator of the networking system must provide a few extra programs to resolve contentions over network resources and to provide the mail functions CP/NET offers.

OPERATING SYSTEMS SIMILAR TO CP/M

Many software creators have tried to improve CP/M's control over the computing process. Most of these attempts have been based upon CP/M-80 version 1.4.

A number of other manufacturers provides CP/M-80 compatible operating systems. Some, like ADDS (Applied Digital Data Systems), have written their own operating system to function like CP/M-80. Others have simply licensed CP/M-80 from Digital Research and added a few features that reflect specific abilities of their equipment.

Figure 6-1 summarizes how the major operating systems that are similar to CP/M developed. Several other operating systems might fit into this outline. Subtle

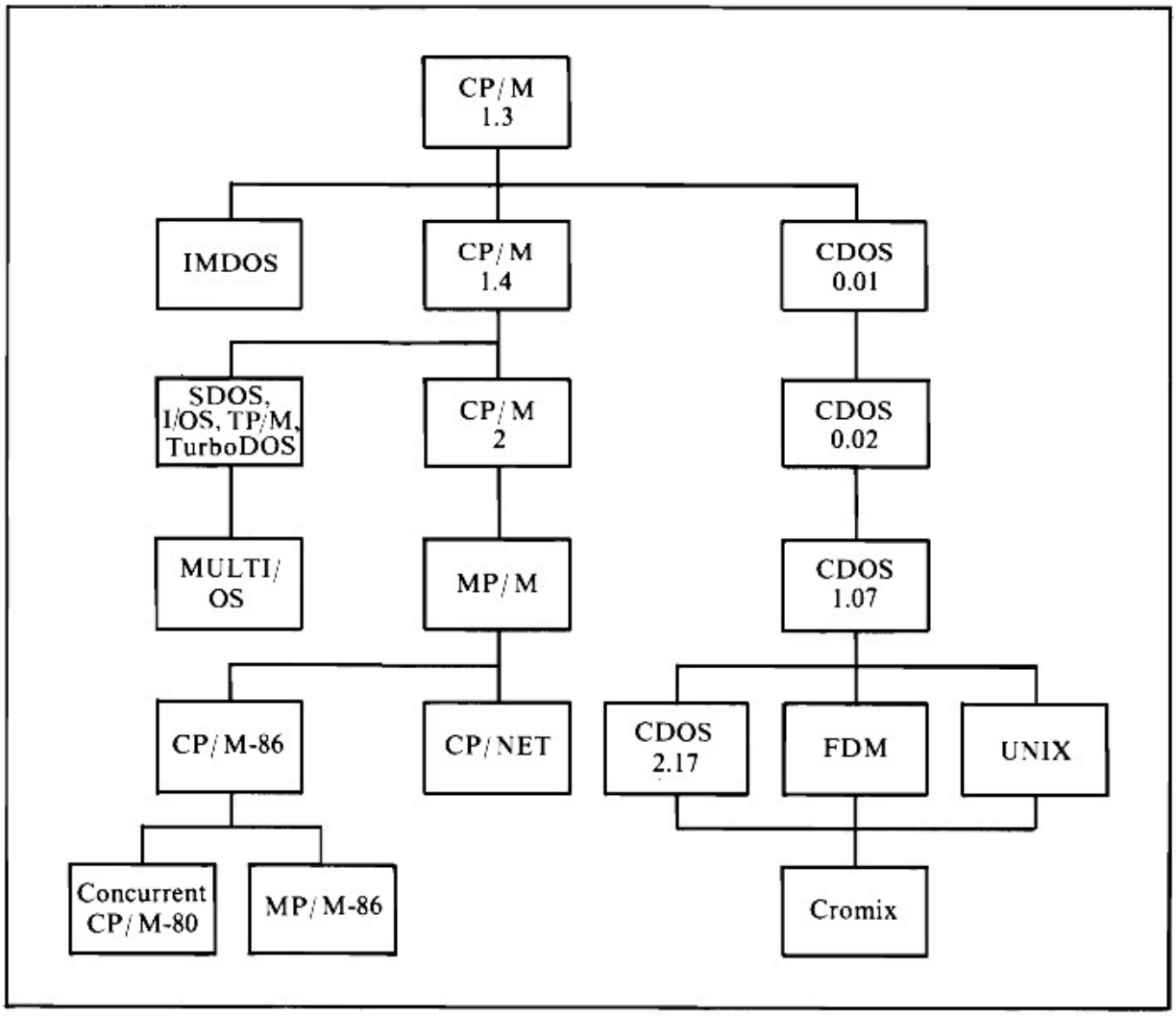


FIGURE 6-1. The evolution of CP/M-like operating systems

changes in an operating system (CP/M 1.41 versus 1.42, for instance) are not reflected in this illustration; only major changes are shown.

Cromemco CDOS

Cromemco was one of the first microcomputer manufacturers. Its first products were not microcomputers but components used with IMSAI and Altair microcomputers. Rapid growth and careful attention to the order of new product development allowed Cromemco to introduce a complete system.

Cromemco's first disk-based system was the Z-2 computer. This system was housed in a rugged industrial-grade cabinet and included two built-in disk drives. Later, Cromemco designed the System 3, a computer for the business environment. Both computer systems were introduced with an operating system developed from CP/M-80 by a firm named InfoSoft (then named TSA); the operating system was named Cromemco Disk Operating System (CDOS).

Originally, CDOS was little more than a rewrite of CP/M-80; it took advantage of the enhanced abilities of the Z80 microprocessor. CDOS, however, continuously evolved, with more subtle refinements than CP/M-80 has encompassed. Cromemco has been able to update CDOS because it has full control over the design of the computer systems that utilize it. CDOS functions only with the Cromemco disk controller and a Z80 CPU. Digital Research has no control over the microcomputers that utilize CP/M-80, thus necessitating fewer changes.

CDOS Compatibility with CP/M-80

Cromemco includes the following notice regarding CDOS's compatibility with CP/M-80 with each copy.

> NOTE: The Cromemco Disk Operating System (CDOS) is an original product designed and written in Z80 machine code by Cromemco, Inc. for its own line of microcomputers. However, due to the large number of programs currently available to run under the CP/M operating system, CDOS was designed to be upwards CP/M-compatible. Cromemco is licensed by Digital Research, the originator of CP/M, for use of the CP/M data structures and user interface. This means that most programs written for CP/M (versions up to and including 1.33) will run without modification under CDOS. This also means that programs written for CDOS will not generally run under CP/M.*

CDOS evolved from the first commercially available version of CP/M, version 1.3; it does not include features Digital Research added in later versions. This is only a minor inconvenience if you use CDOS and try to run programs written for CP/M-80 version 1.4; the differences between CP/M 1.3 and 1.4 are not as extreme as those between 1.4 and 2.2.

A program written using features of CP/M-80 2.2 (actually, any version after 2.0) will not run on a CDOS-equipped computer. However, most programs written for CP/M-80 version 2.2 do not necessarily use the added features. A program developed to run with EBASIC works equally well with CP/M-80 version 1.4 or 2.2 and

*From *Cromemco Users Bulletin*, Issue #1, December 1978.

retains compatibility with CDOS. Versions of Microsoft BASIC and CBASIC2, especially the new compilers for these languages, may not work properly on your CDOS-equipped computer.

A program written for CP/M-80 will not run correctly on your CDOS system just because you can load and execute it. To guarantee everything works correctly, you may

- Test the programs thoroughly, or
- Buy a version of CP/M-80 for your Cromemco computer.

The preferred choice is the second one. Remember Cromemco's advice on CDOS-CP/M-80 compatibility: the CDOS file structure is the same, but some of the system functions are not. Use a CP/M-80 operating system on your Cromemco machine to execute a CP/M-80 program. This is more reliable and economical than modifying CP/M-80 programs to fit CDOS.

CDOS Commands and Utilities

The following CDOS commands operate like CP/M-80 commands:

BYE
  Returns to the Cromemco System Monitor

DIR*
  Displays a diskette file directory

ERA*
  Erases a file or files from a diskette

REN*
  Renames a file

SAVE*
  Saves a portion of memory in a diskette file

TYPE*
  Displays an ASCII file on the console device

Cromemco includes the following utility programs (transient commands):

BATCH*
  Submits a list of commands for execution

DUMP*
  Displays hexadecimal representations of a disk file's contents

EDIT*
  A simple character-oriented editor

INIT
  Initializes (formats and prepares) a diskette

WRTSYS*
  Copies CDOS from one diskette to another

XFER*
Transfers files from disk or device to another disk or device

STAT*
Displays statistics regarding disk and devices

*Command similar to ones recognized by current version of CP/M-80.

These are the minimum commands and utilities supplied. New versions of CDOS (1.07, 2.17, and later) include other utilities and enhance other commands and utilities. Early versions of the STAT program, for example, display the amount of disk space left, the number of directory entries, the names of any null files, and an error message for diskette problems. The latest versions of CDOS include STAT programs that display all of the above plus specific information on disk status, usage, and device assignments. (Detailing the differences between each version of CDOS and CP/M-80 and explaining how all CDOS commands work would require another book.)

Some versions of CDOS include several other utility programs. They are

CDOSGEN
Creates a different-sized operating system

DEBUG*
A debugging tool similar to DDT

LINK
A linker program that takes compiled program code from Cromemco's languages and creates an executable file

SCREEN
A sophisticated editor that works only with Cromemco terminals and may be used for system development or word processing tasks

MEMTEST
Tests Cromemco RAM and reports errors

CDOS Relatives

Cromemco has spawned two derivatives of CDOS. The first is not a unique operating system but rather an extension of CDOS. This relative is a multiuser system based on Cromemco Multi-User BASIC. Cromemco also developed another operating system similar to some popular minicomputer operating systems, called Cromix.

The Multi-User BASIC system allows seven users to run BASIC programs independently and simultaneously. Alternatively, one user can run up to seven independent programs concurrently. Almost complete compatibility exists between Multi-User BASIC and CDOS-developed programs and disk files.

Cromix operates in a completely different user environment, one derived from the minicomputer operating system called UNIX. A single-user Cromix system requires twice as much memory (64K bytes for the operating system plus 64K bytes

for the user) as does CDOS (16K bytes for the operating system and 48K bytes for the user). Cromix uses this extra memory to provide

- File names up to 24 characters long
- Files that are directories of other files. This is known as a *tree-structured directory;* if you were to sketch a complex file system, your structure would look like an upside-down tree
- Twelve built-in commands
- Over 35 utility programs
- *Privilege* levels, that is, protection of files from unauthorized users
- Multiple tasks and multiple users
- Date and time support.

Cromix is not directly compatible with CDOS; Cromix maintains information on the disk in a different fashion (a 24-character file name would not fit into a CP/M-80 directory). Cromemco provides utility programs to convert CDOS files to Cromix files and vice versa. Thus, programs developed with CDOS can still be used. Consider Cromix and CP/M-80 as two totally different operating systems with no compatibility. Running a CP/M-80 program with Cromix requires a multistep conversion process, with no guarantee of success. Converting Cromix files and programs back to CP/M-80-compatible ones is theoretically possible, but so fraught with complications that a non-programmer should not bother to attempt it.

For a Cromemco user, Cromix and Multi-User BASIC are worth considering because each has specific abilities that transcend the abilities of both CDOS and CP/M-80. For a CP/M-80 user, neither offers any improvements that can easily be transferred to the CP/M-80 environment.

CP/M-80 Work-alikes

A number of CP/M-80 work-alike operating systems are now in existence. Generally each has its advantages and disadvantages when compared to the standard CP/M-80 operating system. The most popular of these work-alikes are

I/OS. Formerly known as TSA/OS, I/OS is written by the same people who wrote CDOS. I/OS maintains compatibility with CDOS and CP/M-80, and this is its most obvious advantage. Recently, a multiuser version of I/OS has evolved named MULTI/OS; users of MP/M-80 might be interested in investigating the features of MULTI/OS.

TP/M. TP/M is essentially a Z80 version of CP/M-80; it will not run on 8080- or 8085-based computers. The advantage over CP/M-80 usually cited in conjunction with TP/M is that the use of Z80 instructions in the operating system improves the overall speed of the operating system noticeably. Such claims, of course, are equipment specific, and you may or may not detect any noticeable differences in speed.

SDOS. SD Systems made some minor and cosmetic changes to CP/M-80 to create SDOS. The current version of SDOS, however, is a derivative of CP/M-80 version 1.4 and, therefore, does not feature some of the improvements Digital Research made to CP/M-80 when version 2.2 was introduced.

TurboDOS. A relative newcomer, TurboDOS introduces a number of state-of-the-art software concepts to the CP/M-80 environment, with the result being a substantially faster operating system when disk-oriented tasks are performed. TurboDOS actually attempts to anticipate which sections of the diskette the user will reference and to make sure that they are available when the user calls for them. In addition, some of the more aggravating aspects of CP/M-80, most notably the fact that you cannot exchange diskettes without telling CP/M-80, are eradicated by use of TurboDOS. A multiuser version of TurboDOS is also now available.

CP/M-80 for Zenith/Heath, Polymorphic, and Radio Shack Computers. Owners of early Zenith/Heath, any Polymorphic, or Radio Shack Model I or III computers need to be careful when purchasing CP/M-80 for their systems. While CP/M-80 may operate in the same manner as described in this book, its memory locations have been moved to reflect design differences in these computers. The relocation of CP/M-80 and its reference points in memory make it so that CP/M-80 programs that are normally interchangeable will have to be modified to be used on your machine. Make sure a program version is suitable for your machine if you have any of the following computers and use CP/M-80:

- Poly 88
- Polymorphic 8813
- Radio Shack TRS-80 Model I
- Radio Shack TRS-80 Model III
- Heathkit H8
- Heathkit H89
- Zenith Z89

In each of the above cases, it is possible to modify the equipment to run a standard CP/M-80 operating system, but such modifications should be made by a competent computer technician.

Alternatives for CP/M-86

With the introduction of the IBM Personal Computer in 1981, the CP/M work-alike competition extended itself to CP/M-86.

The initial operating system provided with the IBM Personal Computer was one called, alternatively, IBMDOS or Microsoft DOS. In fact, this operating system is similar to one introduced earlier as 86-DOS, a product of Seattle Computer Products.

86-DOS was written before CP/M-86 was available and, possibly because of this, incorporates an interesting mixture of the features of CP/M-80 and CP/M-86. The built-in commands for 86-DOS are listed below.

86-DOS	CP/M-86
DIR	DIR
RENAME	REN
ERASE	ERA
COPY	(same features of PIP)
TYPE	TYPE
CLEAR	(same function as FORMAT program)

In addition, the original 86-DOS operating system was provided with the following transient commands:

RDCPM
  Copies files from CP/M-80 to 86-DOS format

MKRDCPM
  Creates RDCPM for user's version of CP/M-80

HEX2BIN
  Same as CP/M-80 LOAD.COM

CHKDSK
  Scans diskette for bad sections

SYS
  Similar to CP/M-80 SYSGEN.COM

EDLIN
  Similar to CP/M-80 ED.COM

ASM
  Similar to CP/M-86 ASM-86

TRANS
  Translator of Z80 to 8086 source code

DEBUG
  Similar to CP/M-86 DDT-86

In addition, the command interpreter of 86-DOS (the equivalent to CP/M-80's CCP) regards any file with the type "BAT" as a batch submit file and processes the commands in that file in the order in which they are read.

Microsoft purchased the rights and adapted 86-DOS to the IBM Personal Computer. The documentation provided with the operating system is excellent, and, together with this book, the user should be able to quickly learn to operate 86-DOS, IBMDOS, Microsoft DOS, or anything else one wants to call this operating system. The roots to CP/M-80 are clearly evident in the choice of file name size and file naming conventions, the use of control characters to edit the command line, the prompt displayed on the screen, and many other "visible" aspects of 86-DOS.

Where 86-DOS differs, however, is in the inner reaches of the operating system. The BDOS calls are different for either CP/M-80 or CP/M-86, although they are clearly intended to be similar. What is especially important, however, is that the

layout of information on the diskette is different between 86-DOS and both versions of CP/M. The critical difference is simple: the arrangement of the directory and system tracks are different. For the same disk format, here is the way 86-DOS and CP/M utilize the space.

	86-DOS	CP/M-86
Track 0	DIRECTORY	SYSTEM
Track 1	SYSTEM	SYSTEM
Track 2	SYSTEM	DIRECTORY
Tracks 3-77	DATA	DATA

The very earliest versions of 86-DOS also used a 16-byte directory entry format, although this has now been changed to the 32 bytes used by CP/M. 86-DOS, however, does not require that the user inform the operating system when a diskette is changed.

In short, 86-DOS is an alternative to CP/M-86. Microsoft and Digital Research, once close partners in software development, have taken on an adversarial relationship over which of the two operating systems deserves to be the "standard" for 8086-based systems. Unfortunately, it looks like neither may win, as Microsoft has already announced its intention of creating a superset of 86-DOS that is more like Xenix, their UNIX-derivative operating system, while Digital Research is working on what it calls "concurrent CP/M-86." Only CP/M-86 is directly compatible with the format of CP/M-80 diskettes. However, for users moving from Z80 to 8086-based systems, this may prove to be the deciding factor, because a diskette must either be converted to be used with 86-DOS, or an extra program that translates the diskette format must first be loaded into the computer.

CHAPTER

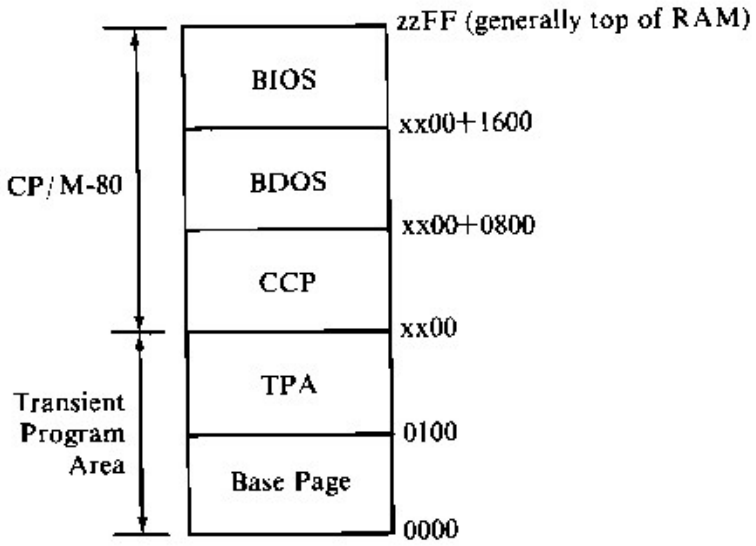
7 Technical Aspects Of CP/M

In the previous chapter we presented information you must know in order to use CP/M. This chapter presents information the assembly language programmer should know in order to program effectively in the CP/M environment. If you are interested in learning about the internal structure of CP/M, then read this chapter. Many readers will find the material presented here too technical for their purposes.

THE STRUCTURE OF CP/M

When you *cold start* CP/M in your computer, CP/M-80 normally loads into the topmost free memory area. CP/M-80 itself takes approximately 6K bytes of memory space, but the machine-dependent portion may take anywhere from another 1K to 8K of memory. Thus, in a normal CP/M-80 system, the top 7K to 14K of memory are occupied by the instructions we have been referring to as CP/M-80 (CP/M-86 will be discussed later in this chapter).

In addition to the chunk of memory CP/M-80 occupies at the top of memory, the first 256 bytes of memory (called the "base page") in your computer system are reserved for use by CP/M-80. Your programs and data may occupy the area in memory between 0100 hex and the bottom of CP/M-80. A 32K CP/M-80 system does not contain 32K of memory for your programs and data, but instead allows about 24K of memory to be used as you please.



NOTE: All addresses are in hexadecimal. The "xx" and "zz" are dummy arguments to represent addresses that change depending upon memory size.

When CP/M-80 is first started, the base page is filled in with some special information. This information is covered in detail later in the chapter, but for now we will summarize the information that is stored in the base page as follows:

Memory Location	Function
0000-0002	Jump to BIOS warm start routine
0003	IOBYTE
0004	Current drive number, current user number
0005-0007	Jump to BDOS entry vector
0008-0037	Reserved for machine interrupts
0038-003A	RST7, used by DDT
003B-003F	Reserved for machine interrupts
0040-004F	Reserved for "scratch" use by BIOS
0050	Drive command was loaded from CP/M 3 or MP/M 2
0051	Address of password for first default FCB (CP/M 3, MP/M 2)
0053	Length of password for first FCB (CP/M 3, MP/M 2)
0054	Address of password for second default FCB (CP/M 3, MP/MP/M 2)
0056	Length of password for second FCB (CP/M 3, MP/M 2)
0057-005B	Reserved
005C-006B	First default file control block (FCB) (CP/M 3, MP/M 2)
006C-007F	Second default file control block (FCB) (CP/M 3, MP/M 2)
0080-00FF	Default disk buffer/command buffer

NOTE: CP/M 1.4, CP/M 2.2, and MP/M 1 all use only one file control block beginning at 005C hex, with the area from 007D-007F hex reserved for the random record position in CP/M 2.2 and MP/M 1. Digital Research is in the process of changing the base page slightly to add special functions like file protection and multiuser capabilities.

Now that you know the basic overall structure of CP/M-80, let's take a look at the pieces one by one.

CCP—The Console Command Processor

The CCP module interprets the CP/M-80 commands you type. This portion of CP/M is generally only relevant when you see an A> (or the prompt for another

drive) on your terminal. The CCP recognizes only a few commands and a handful of control characters (*see* Chapter 2).

The command you type when you see the A> is first stored at location 0080-00FF hex. If the CCP doesn't recognize the command, it checks the disk directory for a file with the type "COM" whose name matches the first eight characters (or up to the first space in the command line). The "matched" file is loaded into memory beginning at 0100 hex, and execution is passed to 0100 hex when the file is completely loaded. Digital Research calls such a file a "transient command."

The command buffer extends from 0080 hex to 0100 hex, meaning that the longest command CP/M-80 recognizes is 128 characters in length.

Since the default disk buffer is not used to load the program, the command that the CCP executed still begins at 0080 hex and may be used by the program for subsequent processing. This is how long command lines, such as MBASIC FILENAME /F:3, are executed. In this example, the CCP would first load the contents of MBASIC.COM into memory, then pass execution to 0100 hex. The MBASIC program then looks at the command line beginning at 0080 hex, detects that additional information was typed, and attempts to continue executing your original command.

There is another trick that the layout of the CCP allows advanced programmers to use: a program may be automatically started. Some special locations within the CCP must first be learned.

CCP start + 07 hex	Length of the command
CCP start + 08 hex	First byte of command
.	
.	
.	
CCP start + 87 hex	Last byte that can be used for command
CCP start + 88 hex	Low-order byte of pointer
CCP start + 89 hex	High-order byte of pointer

Simply place your command in the area beginning at the memory location of the CCP plus 8; the command may be up to 80 hex (128 hex) characters in length. The command string must end with a 00 (null) character for the automatic start to work properly.

The two "pointer" bytes must contain the location of the first byte of the command (low-order byte first, high-order second, as is normal in 8080 programming conventions). The CCP uses this pointer to keep track of how much of the command it has processed. If you do not reset the pointer, the CCP may assume it has already processed part of your command.

Here is a short assembly language program that uses the autostart concept. You can append this program to another assembly language program in order to automatically transfer execution from one program to another.

```
;-----------------------------------------------------------------------------
; AUTOSTRT.ASM  version 1.0  10/12/81                                        ;
; Copyright 1981 by Thom Hogan                                               ;
```

```
;                                                                               ;
; May be used for non-commercial purposes without                               ;
; permission.                                                                   ;
;                                                                               ;
;-------------------------------------------------------------------------------
;
BDOS    EQU     0005H           ;cp/m BDOS entry point
;
        ORG     0100H
;
START:  LHLD    0001H           ;get address of BIOS + 3
        MVI     L,00H           ;in the zero low-order byte of H6
        MOV     A,H             ;get it where we can use it
        SUI     16H             ;subtract offset to CCP
        MOV     H,A             ;put it back in H_
        SHLD    CCP             ;save CCP location for later use
;
; DO ANY EXTRA PROCESSING YOU WISH IN HERE
;
MIDDLE: LXI     D,FILENAME      ;point to command
        LXI     B,10            ;set to length of command + 1
        LHLD    CCP             ;get CCP location
        MVI     L,07            ;location of length is after CCP
        CALL    MOVE            ;move command string
        LHLD    CCP             ;get back CCP location
        MVI     L,88H           ;location of lsb pointer
        MVI     A,08H           ;get default lsb start
        MOV     M,A             ;put it in place
        INX     H               ;location of msb pointer
        MOV     A,H             ;get default msb start
        MOV     M,A             ;put it in place
        LHLD    CCP             ;get back CCP location
        PCHL                    ;and go to it!
;
MOVE:   LDAX    D               ;get byte to move
        MOV     M,A             ;move it
        INX     H               ;increment destination
        INX     D               ;increment source
        DCX     B               ;decrement count
        MOV     A,B             ;get counter into A
        ORA     C               ;check if done
        JNZ     MOVE            ;...not done
        RET                     ;...done
;
```

```
CCP:    DS      2
FILENAME:  DB  00,'XXXXXXXXX',00
;
```

Command length Command Trailing zeros at end

```
        END
```

There are more elegant methods of programming the above action (especially if you have a Z80-based computer), but this process we show here is not masked by programming tricks. You could even follow the above example, but recode it into BASIC. It would be slow and cumbersome, but if done properly would still perform the same function: directly executing any valid CP/M-80 command at the end of any program.

BDOS—The Basic Disk Operating System

All disk drive activity and most console activity passes through this section of CP/M-80. BDOS is not accessible through direct commands at the console however. Instead, the CCP or a transient program places the number of the desired function in the microprocessor's internal C register and then executes a CALL instruction to location 0005 hex.

Some BDOS functions also require that extra information be placed in other internal registers. If you wanted to make a particular character appear on the console display, you would put the character to be displayed in internal register E.

Other BDOS functions return information to the calling program (or portion of CP/M-80). If you ask BDOS to get a character from the console, the character will be returned to your program in the A register.

Let's briefly examine each of the BDOS functions and what they do. Before doing so, however, some information about the notation we will be using is necessary.

The 8080 CPU, for which CP/M was originally written, provides seven 8-bit general purpose internal storage registers and eight status bits. Six of these registers are often paired into three 16-bit registers. The 8080 registers and their normal usage by assembly language programmers in CP/M are as follows:

Register	Usage
A (Accumulator)	Eight-bit character, input/output, or manipulated data register
B C	When used separately, 8-bit registers; when paired, a 16-bit address register
D E	When used separately, 8-bit registers; when paired, a 16-bit address register
H L	When used separately, 8-bit registers; when paired, a 16-bit memory address register

Status	Eight bits of flags to indicate carry, parity, zero, and so on.

Digital Research uses the registers in a particular way for all BDOS functions in CP/M-80. To be specific, the usage breaks down as follows:

A (Accumulator)	Any 8-bit value passed to or from BDOS
B	Not used for BDOS calls
C	BDOS function number to use
DE	16-bit memory location of variable
HL	16-bit value passed to or from BDOS and L; copies A register contents.

To understand the function definitions shown in Table 7-1, you will also need to know about the file control block (FCB) and the IOBYTE. Detailed descriptions of these two concepts appear later in this chapter.

To use any BDOS function, a program loads the function number into register C, loads other registers as required, and calls BDOS via memory location 0005 hex. BDOS performs the function and returns control to the calling program.

We will demonstrate this by displaying a string on the console. As Table 7-1 shows, the print string function number is 09 hex. The DE register pair must contain the address of the string, and the string must terminate with a "$" in order for the function to work properly. A string is displayed as follows:

1. Your program must place the string somewhere in memory, terminated by a "$."
2. Place a 09 hex in the C register.
3. Your program then places the starting address of the string in register pair DE.
4. Your program then CALLs BDOS via the BDOS entry vector at 0005 hex.
5. BDOS performs the "print string" function and returns execution to your program.

One way to perform these steps in assembly language is shown below.

```
;-------------------------------------------------
; PRINT A STRING EXAMPLE;                        ;
; Uses BDOS entry vector at 0005 hex             ;
;-------------------------------------------------
;
BDOS    EQU     0005H       ;the BDOS entry vector
STRING  EQU     09H         ;the PRINT STRING function #
;
        ORG     0100H       ;start at the first TPA address
DOIT:   MVI     C,STRING    ;load the function number
```

TABLE 7-1. BDOS Function Definitions for CP/M-80 Version 2.2

Function No.	Function Name	Entry Parameter(s)	Exit Parameter(s)	Explanation
00	SYSTEM RESET	None	None	Restarts CP/M-80 by returning control to the the CCP after reinitializing the disk subsystem.
01	CONSOLE INPUT	None	A = ASCII character	Returns the next character typed to the character calling program. Any non-printable character is echoed to the screen (like BACKSPACE, TAB, or CARRIAGE RETURN). Execution does not return to the calling program until a character has been typed. Standard CCP control characters are recognized and their actions performed (CONTROL-P begins or ends printer echoing and so on).
02	CONSOLE OUTPUT	E = ASCII character	None	Displays the character in the E register on the console device. Standard CCP control characters are recognized and their actions performed (CONTROL-P begins or ends printer echoing and so on.).
03	READER INPUT	None	A = ASCII character	Returns the next character received from the reader device to the calling program. Execution does not return to the calling program until a character is received.
04	PUNCH OUTPUT	E = ASCII character	None	Transmits the character in the E register to the punch device.
05	LIST OUTPUT	E = ASCII character	None	Transmits the character in the E register to the list device.
06	DIRECT CONSOLE IN	E = FF hex	A = ASCII	If register E contains an FF hex, the console device is interrogated to see if a character is ready. If no character is ready, a 00 is returned to the calling program in register A; otherwise the character detected is returned in register A. If register E contains any character other than an FF hex, that character is passed to the console display. All CCP control characters are ignored. The user must protect the program against nonsensical characters being sent from or received by the console device.
	DIRECT CONSOLE OUT	E = ASCII character	None	
07	GET IOBYTE	None	A = IOBYTE	Places a copy of the byte stored at location 0003 hex in the A register before returning control to the calling program.
08	SET IOBYTE	E = IOBYTE	None	Places a copy of the value in register E into the memory location of 0003 hex before returning control to the calling program.

NOTE: CP/M-80 always copies the contents of the H register in the A register if nothing is to be specifically returned in the A register. Some manufacturers, specifically Microsoft, make use of such information to reduce movement of information between the H and A registers.

TABLE 7-1. BDOS Function Definitions for CP/M-80 Version 2.2 (continued)

Function No.	Function Name	Entry Parameter(s)	Exit Parameter(s)	Explanation
09	PRINT STRING	DE = String address	None	Sends the string of characters stored beginning at the address stored in the DE register pair to the console device. All characters in subsequent addresses are sent until BDOS encounters a memory location which contains a 24 hex (an ASCII "$"). The CCP control characters are checked for and performed if encountered.
0A	READ CONSOLE BUFFER	DE = Buffer address	Data in buffer	This function performs essentially the same as the CCP would in that it takes the characters the user types and stores them into the buffer that begins at the address stored in the DE register pair. The first byte in the buffer pointed to by the DE pair must be the maximum length of the command; BDOS will place the number of characters encountered in the second byte, with the typed command beginning with the third byte pointed to by the DE pair. All standard CCP editing characters are recognized during the command entry.
0B	GET CONSOLE STATUS	None	A = Status	BDOS checks to the status of the console device and returns a 00 hex if no character is ready, FF hex if a character has been typed.
0C	GET VERSION NUMBER	None	HL = Version	If the byte returned in the H register is 00 hex then CP/M is present, if 01, then MP/M is present. The byte returned in the L register is 00 if the version is previous to CP/M 2.0, 20 hex if the version is 2.0, 21 hex if 2.1 and so on.
0D	RESET DISK SYSTEM	None		Used to tell CP/M to reset the disk subsystem. Should be used any time diskettes are changed.
0E	SELECT DISK	E = Disk number	None	Selects the disk to be used for subsequent disk operations. A 00 hex in the E register indicates disk A, a 01 hex indicates disk B, etc.

NOTE: CP/M-80 always copies the contents of the H register in the A register if nothing is to be specifically returned in the A register. Some manufacturers, specifically Microsoft, make use of such information to reduce movement of information between the H and A registers.

TABLE 7-1. BDOS Function Definitions for CP/M-80 Version 2.2 (continued)

Function		Entry Parameter(s)	Exit Parameter(s)	Explanation
No.	Name			
0F	OPEN FILE	DE = FCB address	A = 'Found'/ not found code	Used to activate a file on the current disk drive and current user area. BDOS scans the first 14 bytes of the designated FCB block and attempts to find a match to the filename in the block. A 3F hex (ASCII "?") can be used in any of the filename positions to indicate a "don't care" character. If a match is found, the relevant information about that file is filled into the rest of the FCB by CP/M-80. A value of 00 hex to 03 in register A upon return indicates the open operation was successful, while an FF hex indicates that the file could not be found. If question marks are used to identify a file, the first matching entry is used.
10	CLOSE FILE	DE = FCB address	A = 'Found'/ not found code	Performs the opposite of the open file function. A close file function must be performed upon completion of use of any file which has had information written into it.
11	SEARCH FOR FIRST	DE = FCB address	A = 'Found'/ not found code	Performs the same as the open file function with the difference being that the current disk buffer is filled with the 128-byte record which is the directory entry of the matched file.
12	SEARCH FOR NEXT	None	A = 'Found'/ not found code	Performs the same as search for first function except that the search continues on from the last matched entry.
13	DELETE FILE	DE = FCB address	A = 'Found'/ not found code	Changes a flag on the directory entry for the file pointed to by the FCB so that CP/M-80 no longer recognizes it as a valid file. No information is actually erased when this function is performed, although subsequent writes to diskette may use some of the area previously associated with the "deleted" file.
14	READ SEQUENTIAL	DE = FCB address	A = Error code	If a file has been activated for use by an open file or make file function, the read sequential function reads the next 128-byte block into memory at the current DMA address. The value of 00 hex is returned in the A register if the read was successful, while any non-zero value in the A register indicates failure.
15	WRITE SEQUENTIAL	DE = FCB address	A = Error code	If a file has been activated for use by an open file or make file function, the write sequential function writes the 128-byte block of memory at the current DMA address to the next 128-byte record of the named file.

NOTE: CP/M-80 always copies the contents of the H register in the A register if nothing is to be specifically returned in the A register. Some manufacturers, specifically Microsoft, make use of such information to reduce movement of information between the H and A registers.

TABLE 7-1. BDOS Function Definitions for CP/M-80 Version 2.2 (continued)

Function		Entry Parameter(s)	Exit Parameter(s)	Explanation
No.	Name			
16	MAKE FILE	DE = FCB address	A = DIR code	Creates a new file with the information (name) indicated by the FCB. CP/M-80 does not check to see if the file indicated already exists, so you must first check to see if the file exists (or delete it). A newly created file need not be opened, as the make file function also performs the necessary opening operations.
17	RENAME FILE	DE = FCB address	A = DIR code	Changes the name of the file referenced by the first 16 bytes of the FCB to the name in the second 16 bytes.
18	RETURN LOGIN VECTOR	None	HL = Disk login	The bits in the HL register are used to specify which disk drives are active. The first bit in the L register refers to drive A, the last bit in the H register corresponds to drive P, the highest possible drive. A bit value of 1 indicates active status, a zero denotes an inactive drive.
19	RETURN CURRENT DISK	None	A = Current disk	The numbers 0 through 15 are used to represent the current default disk drive upon return from this function.
1A	SET DMA ADDRESS	DE = DMA	None	Used to select the 128-byte memory block to be used for buffering all disk transfers. Upon system or disk reset, cold or warm start, the buffer is reset to 0080 hex on a normal CP/M-80 system.
1B	GET ALLOC ADDRESS	None	HL = Allocation address	Returns the starting address of the allocation vector, a table which is maintained in memory for each on-line disk drive that indicates the portions of the diskette which are in use.
1C	WRITE PROTECT DISK	None	None	Provides temporary write-protection for the diskette in the current default disk drive.
1D	GET R/O VECTOR	None	HL = Disk R/O	Returns a 16-bit value in the HL registers which indicate which drives on the system are write-protected. The drives are assigned as in the LOGIN VECTOR, with a value of 1 indicating write-protection.
1E	SET FILE ATTRIBUTES	DE = FCB address	A = DIR code	Sets the file attributes that indicate system/directory and R/O or R/W file status for the file pointed to by the FCB address.

NOTE: CP/M-80 always copies the contents of the H register in the A register if nothing is to be specifically returned in the A register. Some manufacturers, specifically Microsoft, make use of such information to reduce movement of information between the H and A registers.

TABLE 7-1. BDOS Function Definitions for CP/M-80 Version 2.2 (continued)

Function No.	Function Name	Entry Parameter(s)	Exit Parameter(s)	Explanation
1F	GET DISK PARMS	None	HL = DPB address	Retrieves the disk parameter block for the current active disk drive. These parameters can be used to determine space available on a diskette or to change the characteristics of the disk drive under user control.
20	GET USER CODE SET USER CODE	E = FF E = User code	A = Current User or None	If the E register contains an FF hex, the current user number is returned in the A register. To reset the user number, the appropriate user code is placed in the E register. While the USER command allows user numbers in the range 0-15, this BDOS function can set user numbers in the range of 0-31.
21	READ RANDOM	DE = FCB address	A = Error code	Reads the random record number contained in the 33rd, 34th, and 35th byte (a 24-bit address) of the BCB pointed to.
22	WRITE RANDOM	DE = FCB address	A = Error code	Writes information from the current DMA address to the random record pointed to by the number contained in the 33rd, 34th, and 35th byte of the indicated FCB.
23	COMPUTE FILE SIZE	DE = FCB address	RRF set	Returns the current size of the random record file in the three bytes that constitute the random record field of the FCB. If the third byte contains a 1, then the file contains the maximum record count of 65536, otherwise the value in the first two bytes is a 16-bit value that represents the file size.
24	SET RANDOM RECORD	DE = FCB address	RRF set	Returns the next random record (fills in the random record field of the FCB) after the last sequentially read record. Digital Research suggests that this function is most appropriate to file indexing.
25	RESET DRIVE	DE = Reset drive bits	A = Error code	Forces the specified drives to be reset to the drive bits initial non-logged status.
26	WRITE RANDOM (ZERO)	DE = FCB address	A = Error code	Writes a record of all zeros to diskette before a record is written; useful for identifying unused random records (an unused record would contain zeros instead of data).

NOTE: CP/M-80 always copies the contents of the H register in the A register if nothing is to be specifically returned in the A register. Some manufacturers, specifically Microsoft, make use of such information to reduce movement of information between the H and A registers.

```
            LXI     D,LABEL         ;load the string pointer in
                                    ;reg. pair DE
            CALL    BDOS            ;do it!
;
GETOUT:     JMP     0000H           ;restart CP/M (not part of
                                    ;the example, but necessary
                                    ;to make this a ''runnable''
                                    ;example).
;
LABEL:      DB      'DOCTOR, MY PROBLEM IS $'
;
            END
```

The example just presented is a complete CP/M-80 program, and should run if you type it in and assemble it as explained in Chapter 4.

Most experienced assembly language programmers use the "equate" pseudo-operation EQU to assign names to the values for each function and for the location of BDOS, as in the previous example. This practice makes the program much easier to understand. A novice programmer might create the same assembly language program as follows:

```
            ORG     100H
            MVI     C,9
            LXI     D,$+10
            CALL    0
            DB      'DOCTOR,'
            DB      'MY PROBLEM IS'
            DB      '$'
            END
```

This does the job, but not very well.

It is wise to begin building a set of standard routines for your assembly language programming using the BDOS functions when possible. An experienced programmer might code the above example as follows:

```
;-----------------------------------------------
; PROGRAM TO PRINT OUT A STRING:                ;
;-----------------------------------------------
;
;
WMSTRT      EQU     0000H           ;CP/M warm start address
BDOS        EQU     0005H           ;BDOS entry point
PSTR        EQU       09H           ;print string function number
;
;
            ORG     0100H           ;begin program here
BEGIN:      LXI     D,LABEL         ;load the string address
```

```
            CALL    PRINT       ;execute the print subroutine
            JMP     WRMSTRT     ;restart CP/M
;
PRINT:      MVI     C,PSTR      ;load print string function
            JMP     BDOS        ;no need to CALL BDOS here
                                ;RET performed by BDOS
;
LABEL:      DB      'DOCTOR, MY PROBLEM IS$'
;
            END
```

If the program is to print strings several times, an experienced programmer might eliminate some repeated steps. It helps to separate your own programming from programming that should be standardized (the BDOS calls). Note that the PRINT routine "jumps" to BDOS instead of calling it. This saves a return instruction (one byte) and some execution time, because the BDOS function executes a RET instruction anyway. By calling the PRINT routine and performing a JMP to BDOS, one instruction is saved each time the routine executes.

BIOS—The Basic Input/Output System

A great deal of confusion about CP/M-80 originates with BIOS. Many people purchase CP/M-80 and discover that it does not work properly. This is due to an incorrect BIOS, or one which was not designed for their particular machine. Configuring a CP/M-80 BIOS is straightfoward for software developers, but it can be frustrating (and impractical) for the computer novice. The BIOS provides a number of low-level functions that CP/M programs (and BDOS) use. The BIOS is provided by the manufacturer, dealer, or software vendor from whom you purchased CP/M-80, not by Digital Research.

Most CP/M-80 suppliers provide a sample BIOS with the original diskette. Digital Research provides both a skeletal BIOS (just the bones and a number of comments; the routines do not include specific instructions) and a completed one for the system they use. Many microcomputer manufacturers remove these two sample BIOSs and include their own. Some supply only the BIOS routines that may need changing (the printer, punch, and reader routines).

If you do not know assembly language, or if you are a newcomer to microcomputers, *do not try to write a BIOS section for CP/M-80!*

The BIOS performs crucial, hardware-dependent functions for CP/M-80. It tells CP/M-80 how to access the various devices that constitute your computer system (for example, how to make the head of the disk drive move from place to place). Any errors in the BIOS section of CP/M-80 can cause your computer to function improperly, or not at all.

BIOS for CP/M-80 version 2.2 begins at the memory location 1600 hex bytes after the location of the CCP and begins with a series of jump instructions called, appropriately enough, the "jump table." These instructions must be arranged in the

proper order, and all must be present for the BIOS to work properly. The first section of a BIOS must therefore be as follows:

```
;--------------------------------------------------------
; BIOS FOR TYPICAL CP/M-80 COMPUTER                     ;
;--------------------------------------------------------
;
; The following JMPs must be left in the order in which
; they are presented.
;
; The following ORG statement applies to CP/M-80 version 2.2
;
          ORG     CCP+1600H   ;start of BIOS
;
          JMP     COLDSTART   ;restart CP/M from scratch
          JMP     WARMSTART   ;restart CP/M
          JMP     CONSTATUS   ;console device status
          JMP     CONINPUT    ;console device input
          JMP     CONOUTPUT   ;console device output
          JMP     LISTOUT     ;list device output
          JMP     PUNCH       ;punch device output
          JMP     READER      ;reader device input
          JMP     HOME        ;reset disk head to home
          JMP     SETDISD     ;select disk to use
          JMP     SETTRACK    ;select track to use
          JMP     SETSECTOR   ;select sector to use
          JMP     SETDMA      ;select DMA address to use
          JMP     READDISK    ;read current sector
          JMP     WRITEDISK   ;write current sector
;
; The following two jumps do not apply to CP/M-80 version
; 1.4, but are necessary for version 2.2.
;
          JMP     LISTSTATUS  ;list device status
          JMP     SECTORTRAN  ;sector translation
; End of standard CP/M-80 jump table, but you may extend
; the table to add other functions, if you desire.
;
```

Each of the "jumps" in the above example points to a particular BIOS routine. An example of a BIOS routine for output to a printer (which the BIOS finds with JMP LISTOUT) follows:

```
;--------------------------------------------------------
; LIST OUTPUT ROUTINE                                   ;
;--------------------------------------------------------
```

```
;
; This routine provides output to the Vector Graphic
; Bitstreamer II I/O board addressed at port 2.
; All CP/M-80 list output will be directed to this
; routine.
;
; Assumptions:
;          ·Bitstreamer is initialized by cold start.
;          ·Character to be printed is in C.
;          ·Bitstreamer is addressed at port 2.
;
LISTST:   EQU     03H           ;status port
LISTDAT:  EQU     02H           ;data port
;
LISTOUT:
          IN      LISTST        ;get current status
          ANI     1             ;check if ready
          JNZ     LISTOUT       ;not ready
;
; Printer is ready if routine gets this far. Print it!
;
LISTON:
          MOV     A,C           ;CP/M-80 has char in C
          OUT     LISTDAT       ;send it
          RET                   ;done, return
;
```

Each routine in your BIOS concludes with a RET (return) instruction. Our sample routine consists of only six instructions; the comments explain the operation of the routine.

Like the BDOS routines, there are certain assumptions made by CP/M-80 as to the entry and exit values of each routine. Refer to Table 7-2 for a full listing of the BIOS routines needed, including entry and exit parameters.

How to Modify BIOS

To create a new or modified BIOS, perform the following steps:

1. On a freshly initialized/formatted diskette, copy the sample BIOS, an editor program, ASM.COM, DDT.COM, SYSGEN.COM, and MOVCPM.COM. Use SYSGEN.COM to copy an unmodified CP/M-80 system onto the diskette. Use a newly formatted diskette for creating your new BIOS so that you do not accidentally destroy a working system.

2. Print a copy of the sample BIOS if you can (you may be changing the BIOS to include routines for a printer). Study the copy carefully. Become familiar with its

TABLE 7-2. CP/M-80 BIOS Routine Definitions

Label in Jump Table	Entry Parameter(s)	Exit Parameter(s)	Explanation
COLDSTART	None	C = 0	Your routine should perform all the necessary start-up operations, including initializing all the values in the base page. Before exiting, the C register must be set to zero.
WARMSTART	None	C = Drive	Your routine should perform all the necessary restart operations but does not need to reinitialize the base page. The C register, on exit, should contain the current drive number.
CONSOLE STATUS (CONSTATUS)	None	A = Status	
CONSOLE* INPUT	None	A = Character	
READER* INPUT	None	A = Character	Your routine should wait for a character to be entered at the appropriate device and then return the character in the A register.
CONSOLE* OUTPUT	C = Character	None	
LIST* OUTPUT	C = Character	None	
PUNCH* OUTPUT	C = Character	None	Your routine should take the character in the C register and display it on the appropriate device.
HOME DISK	None	None	The head of the disk drive should be returned to the home position (track 0, sector 0).
SELECT DISK	C = Drive	HL = DHA	Your routine should select the drive indicated by the number in the C register. The HL register on return should contain the address of the disk parameter header.
SET TRACK	C = Track	None	The track indicated by the C register value should be set as the next track to be accessed by the disk drive.
SET SECTOR	C = Sector	None	The sector indicated by the C register value should be set as the next track to be accessed by the disk drive.
SET DMA ADDRESS	BC = DMA address	None	The DMA address indicated by the BC register pair should be set as the address to use for all information transfers from memory to diskette and vice versa.

*All console and device I/O should be done by first looking at the IOBYTE (0003 hex) to determine which device is selected.

TABLE 7-2. CP/M-80 BIOS Routine Definitions (continued)

Label in Jump Table	Entry Parameter(s)	Exit Parameter(s)	Explanation
READ DISK	None	A = Status	Read the current track and sector and transfer the data to the DMA address already set. A 01 hex should be returned if there was an error during transfer.
WRITE DISK	None	A = Status	Write the current track and sector from the data at the DMA address.
SECTOR	BC = Logical sector	HL = Physical sector	
TRANSLATION	DE = Sector map address		A special routine used for systems which maintain data in other than 128-byte blocks. The logical sector on entry is changed to reflect the appropriate actual sector on the diskette.
LIST STATUS	None	A = Status	Your routine should interrogate the appropriate device to see if a character is ready and return a 00 hex in the A register if not ready, or a FF hex if ready.

*All console and device I/O should be done by first looking at the IOBYTE (0003 hex) to determine which device is selected.

structure, the individual routines required, and the comments included in the sample.

3. Edit the sample BIOS with ED.COM or some other text editor. Begin by inserting comments indicating the date, your name, the reason you are changing the BIOS, and any other information that will help you and others trace problems that may occur later.

4. Minor additions or changes should not prove too difficult, assuming you understand assembly language and the device you wish to manipulate. An example of a minor modification follows. Here is what you might find in the sample BIOS, as a part of the COLDSTART routine.

```
SIGNON:
          DB      14H,0DH,0AH ;clear screen, move down
          DB      'CP/M-80 version 2.2',0Dh,0Ah,'$'
MOVEIT:   LXI     H,SIGNON    ;point to signon message
KEEPON:   MOV     A,M         ;get a byte
          CPI     '$'         ;check for end of message
          RZ                  ;...end of message
          MOV     C,A         ;not end, get ready
```

```
;-----------------------------------------------------
;WARNING! Your CONOUT routine may destroy or undesirably
;alter the contents of the registers used in MOVEIT. To be
;sure, PUSH the contents of any registers not needed by CONOUT
;onto the stack, then POP them off the stack into the original
;register(s).
;-----------------------------------------------------
PUSH    H               ;out of harm's way
CALL    CONOUT          ;send data to console
POP     H               ;all safe and sound
INX     H               ;increment memory location
JMP     KEEPON          ;do it again
```

This routine displays the CP/M-80 sign-on message and initializes the terminal screen with the 14 hex that starts the message. But what if your terminal requires a 04 hex to be initialized (cleared)? Let's change the above BIOS to reflect the new terminal and to personalize the message that is displayed.

```
SIGNON:
        DB      04H,0DH,0AH     ;clear screen, move down
        DB      'Thom Hogan's Vectrola 1.1',0DH,0AH
        DB      '------------',0DH,0AH,0AH
        DB      'CP/M-80 2.2-10/17/80 last update',0DH,0AH
        DB      'no printer installed',0DH,0AH,0AH,0AH,'$'
MOVEIT: LXI     H,SIGNON        ;point to signon message
KEEPON: MOV     A,M             ;get a byte
        CPI     '$'             ;check for end of message
        RZ                      ;...end of message
        MOV     C,A             ;not end, get ready
        PUSH    H               ;out of harm's way
        CALL    CONOUT          ;send data to console
        POP H                   ;all safe and sound
        INX H                   ;increment memory location
        JMP     KEEPON          ;do it again
```

The sample BIOS signs on as follows:

```
<clear screen>
CP/M-80 version 2.2
A>
```

The new version displays

```
<clear screen>
Thom Hogan's Vectrola 1.1
------------------------

CP/M-80 2.2-10/17/80 last update
```

```
no printer installed

A>
```

In this example we have not changed anything important. Notice that the DB '$' line remains so that the routine knows where to end the message.

There are three types of changes to make to BIOS.

- Inserting new material
- Deleting old material
- Changing existing material.

When you insert information into BIOS, previously entered information remains unchanged. You can easily backtrack your steps and restore the BIOS to its original form. We suggest that you identify where you have inserted new material so that it is easier to locate later. A line of hyphens (--------) can separate individual routines. A line of equal signs (=========) can delineate instructions added later. A section of the BIOS might look like the following:

```
;-----------------------------------------------------------------
; MODEM ROUTINES - substitute for PUNCH/READER
;
MODEM           EQU       TRUE        ;yes, we have a modem today
DCH             EQU       FALSE       ;no, it isn't a DC Hayes
;
IF              NOT DCH               ;if standard modem:
MODEMCTL        EQU       03H         ;modem control port
MODEMSBIT       EQU       80H         ;modem send control bit
MODEMRBIT       EQU       40H         ;modem receive control bit
MODEMDATA       EQU       02H         ;modem data port
ENDIF
;
;=================================================================
; 10/17/81 -    We finally got a D.C. Hayes Modem board.
;               Here are the EQUs that we think will
;               operate our new modem. When we've had a
;               chance to check these out, we'll go back
;               and change the DCH EQU to TRUE.
;
IF              DCH
MODEMCTL        EQU       82H         ;DC Hayes control port
MODEMSBIT       EQU       2           ;modem send control bit
MODEMRBIT       EQU       1           ;modem receive control bit
MODEMDATA       EQU       80H         ;modem data port
MODEMCTL2       EQU       81h         ;second control port needed
ENDIF
;=================================================================
```

```
        LXI     H,0
        DAD     SP
        SHLD    STACK
        etc.
```

The routines for the Hayes Microcomputer Products' modem are not yet fully implemented, but you can see how we clearly identified the newly inserted section of program code.

When you delete a section from an existing BIOS, do not erase it; make it a comment instead. Insert semicolons in front of each line you wish to make inoperative. Add a note to explain the deletion.

```
RCVSOH:     MVI     B,1         ;timeout= 1 second
            CALL    RECV        ;get sector
            JC      RCVSTOT     ;got timeout
            MOV     D,A         ;D= block number
            MVI     B,1         ;timeout= 1 second
            CALL    RECV        ;get cma'd sect number
;THE NEXT JUMP CANCELLED 10/1/80 TO DISABLE TIMEOUT
;           JC      RCVSTOT     ;got timeout
;           CMA                 ;calculate complement
;           CMP                 ;good sector number?
;           JZ      RCVDATA     ;yes, got data
; END OF DELETED SECTION OF CODE
;
;
;
;
;
```

NOTE: This routine is from MODEM527.ASM by Ward Christensen—a public domain program available from the CP/M User's Group.

By making the deleted section a comment, you can easily restore the file to its original form; just remove the semicolons.

Changing an area of the BIOS is more difficult; it is not easy to document changes. To trace changes, make the original line a comment, and insert the new line below it. Unfortunately, when you make numerous changes the results are difficult to read. Be sure you always have a copy of the original BIOS; name it BIOSOLD.ASM, BIOS001.ASM, or something similar to indicate that it is not the current version. Each time you edit the BIOS, save a copy of the previous version, and date all versions.

5. Before leaving the editor, return to the beginning of the BIOS file and examine the ORG (origin) directive and any labels named BIOS, BIAS, or OFFSET. Instructions to the assembler on where to start assembling the BIOS differ among programmers. Usually the BIOS is either originated absolutely (by a statement like

ORG 0EF00H) or is calculated using the following method suggested by Digital Research:

```
CCP:        EQU     2900H           ;for 16K CP/M-80 version 1.4
BIOS:       EQU     1500H+CCP       ;location of BIOS
            ORG     BIOS
```

or

```
CCP:        EQU     3400H           ;for 20K CP/M-80 version 2.2
BIOS:       EQU     1600H+CCP       ;location of BICS
            ORG     BIOS
```

Check the CP/M-80 size indicated in such an example. If it matches the CP/M-80 size you are using, go to the next step. If it does not match, or if you cannot tell, do not proceed (size means 48K CP/M-80, 56K CP/M-80 and so on).

Unfortunately, there is no absolute rule about the relationship of the BIOS section with the base section (CCP) of CP/M-80. Knowing the starting location of CP/M-80 does not necessarily mean you can calculate the beginning location of the BIOS section. Digital Research has tried to standardize this, but several distributors and computer manufacturers continue to change the pattern.

Using DDT to locate the start of the BIOS jump table is relatively easy, as shown here.

```
A>  DDT <cr>
    DDT vers. 2.2
    -L0,2 <cr>
    0000 JMP 9603
    003
    -^C
A>
```

Here, the JMP instruction at address 0000 hex, when disassembled, points to the warm start routine. Recall that the warm start is the second entry in the jump table; therefore, subtracting three bytes from 9603 hex will yield the first entry in the table, which shows the start of the BIOS: 9600 hex.

Change the ORG directive in your BIOS to the address found —in this example, 9600 hex.

6. Assemble the new BIOS. If the BIOS assembles without any error messages, proceed to the next step. Always correct your errors before proceeding.

7. To test a new BIOS, perform the following steps:

   a. Create a new CP/M-80 image using MOVCPM * *.

   b. SAVE the new CP/M-80 image using SAVE 34 CPMXX.COM (where "XX" is the size of the CP/M-80 system, for example, 32).

NOTE: With some versions of CP/M-80, MOVCPM will tell you to use a number other than 34 for saving; for example, if MOVCPM displays

```
READY FOR ''SYSGEN''
```

or

```
''SAVE 39 CPM64.COM''
```

then your SAVE command is

```
SAVE 39 CPM64.COM
```

c. Use DDT to load the image of CP/M-80 into memory. If your version of MOVCPM operates correctly, the following addresses apply:

```
BOOT      900H
CCP       980H        (sometimes A00H if long BOOT)
BDOS      1180H       (sometimes 1200H if long BOOT)
BIOS      1F80H       (CP/M 2.2)
          1E80H       (CP/M 1.4)
```

d. Use DDT to load the assembled BIOS into the correct area of memory. The proper method is as follows:

```
-IBIOS.HEX <cr>
-Roffset
```

where *offset* loads your BIOS at the proper point of the CP/M-80 image and not at its eventual location.

The offset is calculated by adding a number to the eventual location of BIOS (its permanent address) so that it loads at 1F80 hex (version 2.2) or 1E80 hex (version 1.4). To calculate this number, first subtract the loading address (1F80 hex or 1E80 hex) from the eventual address.

For a 56K CP/M-80 2.2 system, for example

```
DA00H - 1F80H = BA80H          FFFF - DA00 + IF80
```

it is a good idea to fill memory beginning at 1F80 hex or 1E80 hex with zeros before loading in the BIOS, so you can tell whether or not it was correctly loaded.

e. Exit from DDT with a ^C and save the new CP/M-80 system with a SAVE 34 CPMxxOK.COM, where "xx" is the size of the system you are creating.

f. If you have CP/M-80 version 2.2 you may use SYSGEN to save the new system onto diskette by typing

```
A>SYSGEN CPMxxOK.COM <cr>
```

Older versions of CP/M-80 require that you reload the system image before using SYSGEN with

```
A>DDT CPMxxOK.COM <cr>
-^C
A>SYSGEN <cr>
```

The IOBYTE

The concept of the IOBYTE predates CP/M-80 by several years. Gary Kildall used it in some sample implementations of CP/M-80, and it is documented in most of the Digital Research manuals. Other software creators writing BIOS modules have elaborated upon the IOBYTE concept but have not changed its basic function.

The IOBYTE is a reserved byte of memory that indicates the current assignment of physical devices to logical devices. In CP/M-80, you have the following four logical devices:

```
CON:  Console device
LST:  List device
RDR:  Reader device
PUN:  Punch device
```

When you have two different printers, terminals, or paper tape readers, the IOBYTE indicates which device is to be used.

The IOBYTE is normally located at address 0003 hex. The byte is treated as four separate two-bit indicators.

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

Table 7-3 shows how CP/M interprets the two bits for each device. The device names should look familiar. They are the physical devices that PIP and STAT address. An IOBYTE value of 00100100 (or 24 hex) means the physical devices are currently assigned to logical functions as follows:

- The TTY: device is performing the CONSOLE function
- The PTR: device is performing the READER function
- The UP1: device is performing the PUNCH function
- The TTY: device is performing the LIST function.

A BIOS routine you or your computer vendor wrote examines the IOBYTE to see where to send or receive information. The console status, console input, console output, reader, punch, and list jumps in the BIOS jump table should all point to special routines to see which device to use. These routines proceed as follows:

1. Get the IOBYTE
2. Determine which device to use
3. Go to the routine for that device.

Disk Operations

Some indication of how CP/M-80 stores information on a diskette is appropriate for this chapter. Unfortunately, we cannot be exacting about the information we are

TABLE 7-3. Interpreting the IOBYTE

Logical Device of Function	Physical Device				
	00	01	10	11	
Console	CON:	TTY:	CRT:	BAT:	UC1:
Reader	RDR:	TTY:	PTR:	UR1:	UR2:
Punch	PUN:	TTY:	PTP:	UP1:	UP2:
List	LST:	TTY:	CRT:	LPT:	UL1:

about to present, as each different type of disk drive usually requires a slightly different set of parameters. To avoid confusion, we will talk specifically about 8-inch single-density CP/M-80 version 2.2 and attempt to indicate how other versions may differ.

You already know that information is stored on diskettes in "sectors" which divide "tracks" of information into blocks of information. Each sector stores 128 bytes of data, and with single-density 8-inch diskettes, there are 26 sectors on each of 77 tracks. Here is how some common disk formats differ.

Format	Number of Sectors	Number of Tracks
Eight-inch single density	26	77
Eight-inch double density	26	77
North Star single density	20	35
North Star double density	40	35
Micropolis MOD II	32	77
Intertec Superbrain	40	35
Apple	32	35
Altos double density	48	77
Pickles and Trout TRS-80	64	77
Osborne 1 (256-byte sectors)	10	40

On most implementations of CP/M-80 the first two tracks are reserved as "system tracks." This is where the CP/M-80 is stored by SYSGEN, and sometimes this area also contains special utilities used by your particular computer. The third track on most floppy diskette systems is reserved for the directory. The remaining tracks are used to store data.

CP/M-80 allocates sectors to files in a "group." A group is composed of a fixed number of sectors, usually 8 or 16. Although CP/M-80 stores data in 128-byte blocks, a file will use the entire group it has been assigned ($8 \times 128 = 1024$ bytes) even if it holds only one byte of information. While this wastes space, it does allow the directory to be more efficient, since it need only keep track of groups, rather than individual sectors.

If you have a hard disk or a double-density floppy disk on your system, it is likely

that it has more than eight sectors per group, resulting in 2028-, 4056-, and even 8112-byte groups.

Each directory entry is comprised of 32-byte blocks.

- Byte 1 indicates whether the file is active or erased
- Bytes 2 through 9 are the file name
- Bytes 10 through 12 are the file type
- Byte 13 is the extent number
- Bytes 14 and 15 are reserved for internal use
- Byte 16 is the extent size in sectors
- Bytes 17 through 32 store the groups assigned to the file.

CP/M-80 also uses a concept called "extents." An extent is a group of groups, again usually 8 or 16 in number. Each directory entry is for one extent of a given file. If the file gets big enough so that 16 groups are not enough to store the entire file, a second directory entry is created for the file—the second "extent." The maximum size of a file is limited by the maximum number of extents allowed by your version of CP/M-80, multiplied by 128 times the number of sectors per group.

This is not as complex as it seems. A file consists of some number of extents, each of which consists of some number of groups, each of which consists of some number of sectors. So that you get an idea of this multiextent concept, following are four directory entries as they would appear in both hexadecimal and ASCII format (see the file control block section immediately following for an explanation).

```
00 41 53 4D 20 20 20 20 20 43 4F 4D 00 00 00 40 .ASM     COM...@
1B 1C 1D 1E 00 00 00 00 00 00 00 00 00 00 00 00 ................

00 5A 53 4D 20 20 20 20 20 43 4F 4D 00 00 00 3B .ZSM     COM...;
1F 20 21 22 00 00 00 00 00 00 00 00 00 00 00 00 . !."..........

00 4D 38 30 20 20 20 20 20 43 4F 4D 00 00 00 80 .M80     COM....
23 24 25 26 27 28 29 2A 00 00 00 00 00 00 00 00 #$%&'()*........

00 4D 38 30 20 20 20 20 20 43 4F 4D 01 00 00 09 .M80     COM....
2B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 +...............
```

This directory comes from a diskette the author uses on his Vector Graphic computer system. This particular implementation of CP/M-80 allows eight groups per extent, but the file M80.COM occupies nine groups (23 hex through 2B hex). Thus this file's second extent is marked with a 01 hex immediately following the name of the file, while the first extent has a 00 hex immediately following the name.

File Control Blocks (FCBs)

The file control block is a section of memory reserved by CP/M to hold information about the file currently being used. The FCB is 36 bytes in length and is normally located at 005C hex. Here are the bytes of the standard CP/M (both CP/M-80 and CP/M-86) FCB and what they are used for.

Byte Number	Function
1	Drive code 0 = use default drive 1 = use drive A 2 = use drive B
2-9	File name Eight bytes with unused characters filled with ASCII spaces
10-12	File type Three bytes with unused characters filled with ASCII spaces The high bit of byte 10 is used to indicate that a file is R/O The high bit of byte 11 is used to indicate that a file is SYS
13	The current extent number for the file
14	Reserved for CP/M's use
15	Reserved for CP/M's use; should be set to zero for OPEN, MAKE, or SEARCH FOR FILE functions
16	Record count for the current extent
17-32	Filled in by CP/M with information about location of records in file
33	Current record number to read or write in a sequential file operation
34-35	Random record number
36	Random record number overflow indicator

Disk Parameter Tables

Earlier in this chapter, the disk parameter header was mentioned in passing. This is a special area of BIOS memory reserved for some important information CP/M-80 needs to know in order to use the disk drives you own. Each disk drive has an associated 16-byte parameter header that contains information about the drive and provides some space for BDOS to write temporary information into.

Bytes	Title	Description
1-2	XLT	Translation Vector; the address of the sector translation table which contains the skew factor for the drive. A value of 0000 hex indicates no translation takes place. Drives that use the same skew factors may share the same translate tables.
3-8	None	Bytes of scratchpad RAM for BDOS use.

9-10	DIRBUF	Directory Buffer: the address of a 128-byte RAM area to be used for directory operations for the drive. All drives share the same directory buffer.
11-12	DPB	Disk Parameter Block: the address of the disk parameter block. Drives with identical characteristics may share the same disk parameter block.
13-14	CSV	Check for Changed Diskettes: the address of a scratchpad area used to check for changed diskettes. The address must be different for each drive.
15-16	ALV	Disk Allocation Vector: the address of a scratchpad area used to keep information about the disk drive's storage allocation. The address must be different for each drive.

The disk parameter headers for each drive appear in order, one immediately following the other. The SELDSK routine in BIOS should return the base address of the header for the drive selected. If a drive does not exist, SELDSK returns a 0000 hex.

The translation vectors referred to by the DPH may be located elsewhere in BIOS, and are simply a listing of the sectors to be read in the order in which they are to read. A valid sector translate table might look like the following:

```
DISKXLATE:  dB   1, 7,13,19,25
            dB   5,11,17,23, 3
            dB   9,15,21, 2, 8
            dB  14,20,26, 6,12
            dB  18,24, 4,10,16
            dB  22
```

The translate table above shows a skew factor of six (the disk drive reads a sector, then reads the sector which comes six later, and so on). Changing the translate table is one way in which disk speed may be increased.

The disk parameter block tells CP/M-80 the layout of a drive.

Bytes	Title	Description
1-2	SPT	Number of sectors per track
3-4	BSH/BLM	Block shift factor
5	EXM	Data block allocation size
6-7	DSM	The maximum data block number
8-9	DRM	The number of directory entries
10-11	AL0/1	Directory group allocation
12-13	CKS	Check directory entry
14-15	OFF	Number of tracks skipped at beginning of disk

CP/M-80

For further information on the internal structure of CP/M-80, consult the annotated bibliography in Appendix F. A number of excellent articles about specific portions of CP/M-80 have appeared in trade publications.

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

CP/M-80 and CP/M-86 are designed to work with different CPUs. CP/M-80 works with 8080, 8085, and Z80 chips, while CP/M-86 works only with 8086 and 8088 chips. The former are 8-bit chips, while the latter are 16-bit chips (the 8088, more correctly, is a 16-bit chip which uses an 8-bit data bus).

There are a few differences between CP/M-80 and CP/M-86 that are important to users. Primarily, CP/M-86 can address (use) up to one megabyte of memory, while CP/M-80 is limited to 64K bytes of memory space (version 3.0 of CP/M-80 allows use of an additional 64K of memory). CP/M-80 makes reference to memory in an absolute fashion; in other words, CP/M-80 always references specific memory locations between 0000 hex and FFFF hex. CP/M-86 refers to memory locations by an *offset* method, based on the values in the segment registers of the 8086 or 8088 chips.

Instead of using a BDOS call to location 0005 hex like CP/M-80, CP/M-86 programs use software interrupt number 244 to perform the same function; the registers are first loaded with the appropriate values, followed by the interrupt. CP/M-86 BDOS calls do not use the addresses at 0006 hex and 0007 hex; CP/M-86 maintains these addresses anyway for programs that are translated from the CP/M-80 environment and use that location to determine memory size.

To understand the following descriptions of BDOS functions, a description of the 8086 registers is necessary. Eight 8-bit general purpose data registers are available, and all are grouped together in twos to provide 16-bit register pairs.

8-Bit Registers	become	**16-Bit Register Pairs**
AH + HL		AX
BH + BL		BX
CH + CL		CX
DH + DL		DX

In addition, the 8086 has four 8-bit index (base) registers.

BP
SP
SI
DI

Last, four segment registers (used to calculate the extended addressing needed for more than 64K bytes of memory access) are available.

Code
Data

Stack
Extra

CP/M-86 BDOS calls are detailed in Table 7-4.

For the most part, programmers familiar with CP/M should find CP/M-86 quite similar. At the time this book was revised, Digital Research had just completed a revision of CP/M-86 and also announced a version called Concurrent CP/M-86. Readers interested in more information about the particulars of CP/M-86 should contact Digital Research (the address is given in Appendix H).

TABLE 7-4. BDOS Functions for CP/M-86*

Function No.	Function Name	Entry Parameter(s)	Exit Parameter(s)	Explanation
00	SYSTEM RESET	DL = Abort code	None	Restarts CP/M-86 by returning control to the CCP after reinitializing the disk subsystem. The abort code in DL is 00 hex to terminate the currently active program and control is returned to CCP; if DL is 01 hex, the program remains in memory and the memory allocation state remains unchanged.
01	CONSOLE INPUT	None	AL = ASCII character	Returns the next character typed to the calling program. Any non-printable character is echoed to the screen (like BACKSPACE, TAB, or CARRIAGE RETURN). Execution does not return to the calling program until a character has been typed. Standard CCP control characters are recognized and their action performed (CONTROL-P begins or ends printer echoing and so on).
02	CONSOLE OUTPUT	DL = ASCII character	None	Displays the character in the DL register on the console device. Standard CCP control characters are recognized and their action performed (CONTROL-P begins or ends printer echoing and so on).
03	READER INPUT	None	AL = ASCII character	Returns the next character received from the reader device to the calling program. Execution does not return to the calling program until a character is received.
04	PUNCH OUTPUT	DL = ASCII character	None	Transmits the character in the DL register to the punch device.
05	LIST OUTPUT	DL = ASCII character	None	Transmits the character in the DL register to the list device.
06	DIRECT CONSOLE IN DIRECT CONSOLE STATUS DIRECT CONSOLE OUT	DL = FF DL = FE DL = ASCII character	AL = ASCII character or 00 AL = Status None	If register DL contains an FF hex, the console device is interrogated to see if a character is ready. If no character is ready, a 00 hex is returned to the calling program in register A; otherwise the character detected is returned in register A. If register DL contains any character other than an FF or FE hex, that character is passed to the console display. All CCP control characters are ignored. The user must protect the program against nonsensical characters being sent from or received by the console device.
07	GET IOBYTE	None	AL = IOBYTE	Places a copy of the I/O byte in the AL register before returning control to the calling program.

*NOTE: In CP/M-86, the function number is loaded into the CL Register. This is the same as loading the C register in CP/M-80.

TABLE 7-4. BDOS Functions for CP/M-86* (continued)

Function No.	Function Name	Entry Parameter(s)	Exit Parameter(s)	Explanation
08	SET IOBYTE	DL = IOBYTE	None	Places a copy of the value in register DL into the I/O byte location before returning control to the calling program.
09	PRINT STRING	DX = String address	None	Sends the string of characters stored beginning at the address stored in the DX register pair to the console device. All characters in subsequent addresses are sent until BDOS encounters a memory location which contains a 24 hex (an ASCII "$"). The CCP control characters are checked for and performed if encountered.
0A	READ CONSOLE BUFFER	DX = buffer address	Data in buffer	This function performs essentially the same as the CCP would in that it takes the characters the user types and stores them into the buffer that begins at the address stored in the DX register pair. The first byte in the buffer pointed to by the DX pair must be the maximum length of the command; BDOS will place the number of characters encountered in the second byte, with the command typed beginning with the third byte pointed to by the DX pair. All standard CCP editing characters are recognized during the command entry.
0B	GET CONSOLE STATUS	None	AL = Status	BDOS checks to the status of the console device and returns a 00 hex if no character is ready, 01 hex if a character has been typed.
0C	GET VERSION NUMBER	None	BX = Version	If the byte returned in the BH register is 00 hex, then CP/M is present, if BL = 01, then MP/M is present. The byte returned in the BL register is 00 hex if the version is previous to CP/M 2.0, 20 hex if the version is 2.0, 21 hex if 2.1, and so on.
0D	RESET DISK SYSTEM	None		Used to tell CP/M to reset the disk subsystem. Should be used any time diskettes are changed.
0E	SELECT DISK	DL = Disk number	None	Selects the disk to be used for subsequent disk operations. A 00 hex in the DL register indicates disk A, a 01 hex indicates disk B, and so on.

*NOTE: In CP/M-86, the function number is loaded into the CL Register. This is the same as loading the C register in CP/M-80.

TABLE 7-4. BDOS Functions for CP/M-86* (continued)

Function		Entry Parameter(s)	Exit Parameter(s)	Explanation
No.	Name			
0F	OPEN FILE	DX = FCB address	AL = DIR code	Used to activate a file on the current disk drive and current user area. BDOS scans the first 14 bytes of the designated FCB block and attempts to find a match to the filename in the block. A 3F hex (ASCII "?") can be used in any of the filename positions to indicate a "don't care" character. If a match is found, the relevant information about that file is filled into the rest of the FCB by CP/M-86. A value of 00 hex to 03 hex in register A upon return indicates the open operation was successful, while an FF hex indicates that the file could not be found. If question marks are used to identify a file, the first matching entry is used.
10	CLOSE FILE	DX = FCB address	AL = DIR code	Performs the opposite of the open file function. A close file function must be performed upon completion of use of any file which has had information written into it.
11	SEARCH FOR FIRST	DX = FCB address	AL = DIR code	Performs the same as the open file function with the difference being that the current disk buffer is filled with the 128-byte record which is the directory entry of the matched file.
12	SEARCH FOR NEXT	None	AL = DIR code	Performs the same as search for first function except that the search continues on from the last matched entry.
13	DELETE FILE	DX=FCB address	AL = DIR code	Changes a flag on the directory entry for the file pointed to by the FCB so that CP/M-86 no longer recognizes it as a valid file. No information is actually erased when this function is performed, although subsequent writes to diskette may use some of the area previously associated with the "deleted" file.
14	READ SEQUENTIAL	DX = FCB address	AL = Error code	If a file has been activated for use by an open file our make file function, the read sequential function reads the next 128-byte block into memory at the current DMA address. The value of 00 hex is returned in the AL register if the read was successful, while any non-zero value in the AL register indicates failure.
15	WRITE SEQUENTIAL	DX = FCB address	AL = Error code	If a file has been activated for use by an open file or make file function, the write sequential function writes the 128-byte block of memory at the current DMA address to the next 128-byte record of the named file.

*NOTE: In CP/M-86, the function number is loaded into the CL Register. This is the same as loading the C register in CP/M-80.

TABLE 7-4. BDOS Functions for CP/M-86* (continued)

Function		Entry Parameter(s)	Exit Parameter(s)	Explanation
No.	Name			
16	MAKE FILE	DX = FCB address	AL = DIR code	Creates a new file with the information (name) indicated by the FCB. CP/M-86 does not check to see if the file indicated already exists, so you must first check to see if the file exists (or delete it). A newly created file need not be opened, as the make file function also performs the necessary opening operations.
17	RENAME FILE	DX = FCB address	AL = DIR code	Changes the name of the file referenced by the first 16 bytes of the FCB to the name in the second 16 bytes.
18	RETURN LOGIN VECTOR	None	BX = Disk login	The bits in the BX register are used to specify which disk drives are active. The first bit in the BX register refers to drive A, the last bit in the BX register corresponds to drive P, the highest possible drive. A bit value of 1 indicates active status, a zero denotes an inactive drive.
19	RETURN CURRENT DISK	None	AL=Current disk	The numbers 0 through 15 are used to represent the current default disk drive upon return from this function.
1A	SET DMA ADDRESS	DX = DMA address	None	Used to select the 128-byte memory block to be used for buffering all disk transfers. Upon system or disk reset, cold or warm start, the buffer is reset to 0080 hex on a normal CP/M-86 system.
1B	GET ALLOC ADDRESS	None	BX=Allocation address ES=Segment base	Returns the starting address of the allocation vector, a table which is maintained in memory for each on-line disk drive that indicates the portions of the diskette which are in use.
1C	WRITE PROTECT DISK	None	None	Provides temporary write-protection for the diskette in the current default disk drive.
1D	GET R/O VECTOR	None	BX = Disk R/O	Returns a 16-bit value in the BX registers which indicate which drives on the system are write-protected. The drives are assigned as in the LOGIN VECTOR, with a value of 1 indicating write-protection.
1E	SET FILE ATTRIBUTES	DX = FCB address	AL = DIR code	Sets file attributes to those indicated in the appropriate portion of the FCB pointed to in DX.
1F	GET DISK PARMS	None	BX = DPB address ES=Segment base	Returns the offset and segment base of the disk parameter block of the currently selected drive.

*NOTE: In CP/M-86, the function number is loaded into the CL Register. This is the same as loading the C register in CP/M-80.

TABLE 7-4. BDOS Functions for CP/M-86* (continued)

Function No.	Function Name	Entry Parameter(s)	Exit Parameter(s)	Explanation
20	GET USER CODE SET USER CODE	DL = FF DL = User code	AL=Current user None	If the DL register contains an OFF hex the current user number is returned, otherwise the user number is set to the value in the DL register.
21	READ RANDOM	DX = FCB address	AL = Error code	Reads the random record pointed to by the random record portion of the FCB indicated.
22	WRITE RANDOM	DX = FCB address	AL = Error code	Writes the random record pointed to by the random record portion of the FCB indicated from the current DMA buffer.
23	COMPUTE FILE SIZE	X = FCB address	RRF set	Fills in the random record portion of the FCB indicated to the last record in the file (the size of the file).
24	SET RANDOM RECORD	DX = FCB address	RRF set	Produces the next random record from the position of the last sequential disk read and places it into the correct position in the FCB.
25	RESET DRIVE	DX = Reset drive bits	AL = Error code	Resets the drive(s) indicated to the not-logged-in status.
28	WRITE RANDOM (ZERO)	DX = FCB address	AL = Error code	Writes zeros into the random record before writing the information in the DMA buffer.
32	DIRECT BIOS CALL	DX = BIOS descriptor	None	Allows BDOS to directly access BIOS routines. The first 8-bit value at the location pointed to by DX is the BIOS function number, the next two 16-bit locations are loaded into the 8086 CX and DX registers before the BIOS call is initiated.
33	SET DMA BASE	DX = Base address	None	Sets the base address for subsequent DMA transfers.
34	GET DMA BASE	None	BX = DMA offset ES = DMA segment	Returns the base address for the last DMA transfer made.
35	GET MAX MEM	DX = Offset of MCB	AL=Return code	Finds the largest available memory region which has at least the number of bytes indicated by the current memory control block.
36	GET ABS MAX	DX = Offset of MCB	AL=Return code	Finds the largest possible region of memory at the absolute paragraph boundary indicated by the current memory control block.
37	ALLOC MEM	DX = Offset of MCB	AL=Return code	Allocates memory according to the MCB at the address indicated by DX.

*NOTE: In CP/M-86, the function number is loaded into the CL Register. This is the same as loading the C register in CP/M-80.

TABLE 7-4. BDOS Functions for CP/M-86* (continued)

Function No.	Function Name	Entry Parameter(s)	Exit Parameter(s)	Explanation
38	ALLOC ABS MEM	DX = Offset of MCB	AL=Return code	Similar to function 37, except that the absolute base address is used in the calculation.
39	FREE MEM	DX = Offset of MCB	None	Releases memory areas allocated to a program with the MCB indicated by DX.
3A	FREE ALL MEM	None	None	Releases all memory for use in the system.
3B	PROGRAM LOAD	DX = Offset of FCD	AX=Return code BX = Base page address	Loads a .CMD file.

*NOTE: In CP/M-86, the function number is loaded into the CL Register. This is the same as loading the C register in CP/M-80.

CHAPTER

8 The Systems Approach

Although CP/M is a relatively simple operating system, it has not always been well understood. During its first years of existence, the public criticized the documentation and often mistook hardware problems to be errors in CP/M's design.

Selling computer systems to the general public includes a responsibility for thorough training and product reliability. The microcomputer industry has often been deficient in this regard. Digital Research has left the task of educating the general public in the use of CP/M-80 and CP/M-86 to computer manufacturers, distributors, and dealers. Some do a good job of educating the computer consumer; most have yet to come to grips with the problem.

To simply complain about the industry's neglect of the computer user avoids the main problem being discussed here: how does the end user learn to use CP/M? While this book provides one answer, it is only one of many needed aids.

The following pages present a series of recommendations to CP/M users. They offer a *system* solution to problems you may encounter. For our purposes, *system* will be assumed to include the entire universe of people, equipment, paper, information, and resources involved in the use of a computer. Buying a computer, some software, and buying this book is not enough. You must integrate each component of the system into your current needs and procedures.

SYSTEM RECOMMENDATIONS

Misleading advertising and consumer naivete cause many users to buy a computer system according to price. If two computer sellers promote the XY system, the customer buys from the firm with the best price. This is dangerous. While it is possible to use this approach with items purchased for purely personal reasons, if you do

so while choosing a computer for your business, you may find that the promises made to you are never kept.

Support should be considered just as important as price in your decision of where to purchase computer equipment. Support is a nebulous term that includes such factors as

- Training of personnel in use of the equipment
- Installation and checking of equipment
- Local and on-site service when necessary
- An attempt by the seller to make sure you are getting the type and quality equipment you need, not just a brand name
- The willingness of the seller to answer questions and help you use your system after the sale.

If you are using your computer for a business activity, the diskettes you use to store information become your most valuable computer asset. If your computer breaks down and is unusable for any period of time, it would be like having all the paper records in your office locked in the safe when you cannot remember the combination; in a word, the information stored on diskette is *useless* unless it is available to you.

Thus, you must weigh the initial price of a system against the ongoing support cost. Most reputable computer dealers not only go out of their way to help computer users with problems but also make it a point to help the customer make sure the problem does not occur again. Mail-order firms may offer good prices on equipment, but you must weigh these prices against the amount and type of support you will need after the sale.

An element of support that is often neglected is making sure that the equipment and software chosen meet the requirements of the job they are to perform. CP/M-80 and CP/M-86 dictate many aspects of the computer system they are used on. To operate CP/M effectively, you should have available

- Two disk drives
- 500K or more of disk storage space
- 48K of main computer memory
- A terminal that displays 24 lines, each 80 characters
- A printer
- A modem.

We will examine each of these components individually.

Two Disk Drives

You need two disk drives for several reasons. As we discussed in Chapter 1, diskettes are fragile. They are easily bent, useless when dirty, and can be tampered with, lost, or stolen. In short, they are not a permanent storage medium. Thus, you

need to copy a diskette for *backup* purposes. Copying is far easier and faster with two disk drives.

While you can operate a computer system with only one drive– indeed, some systems only come with one drive—if that drive malfunctions, you can no longer use your system. With two drives, you can often continue to operate, although in a limited fashion, if one drive breaks down.

It is inconvenient to operate with only one drive. If your computer system is difficult to use, as it would be if you had to continually transfer diskettes in and out of a single drive, you might stop using it. Many serious business application programs compatible with CP/M require two disk drives simply because the programs themselves take up all of the space on one diskette. In addition, most computer professionals suggest that it is a good idea to get into the habit of using separate data diskettes and program diskettes, something you probably wouldn't be able to do with only one drive.

500K Bytes of Disk Storage Space

Five hundred thousand characters of information seem like a lot of information. For example, the text of this book is about 450,000 characters. In general, however, most computer users tend to underestimate the volume of information they use and generate.

Serious accounting programs (like the Structured Systems Group Integrating Accounting package or the Peachtree Integrated Accounting package) use as much as 700,000 characters of disk space just to store the programs, although this is normally split among several diskettes. The volume of information a business keeps for tax and auditing purposes is staggering. For example, tracing the meter readings and payments of 1000 utility district customers for one year takes up about 300K bytes.

Consider a typical retail business which averages 100 transactions a day. With only 100 characters of information per transaction, one year's worth of transactions will occupy almost four million characters of information.

It is unrealistic to try to store all the information you need on a single diskette. Instead, apportion the information into *logical* blocks and save a block on an individual diskette. The reason for this suggestion is simple. Suppose your annual business transactions require four million characters of information and you want to see the transactions for the second week of August. On many systems, the program must first wade through January through July before it finds the August data, taking a considerable amount of time just to identify the data. A more practical and convenient method would be to assign each month's information to a single diskette. Only rarely do accounting systems address all data at the same time.

There is another advantage to floppy disk systems with capacities ranging from 500K bytes to 1 megabyte when you use accounting software. Many accounting systems *close* all transactions every month and carry forward only the balances. Usually you cannot look at the details of each transaction in a *closed* month. If you

make an extra copy of your data (by making a duplicate diskette), you can recover the individual transactions whenever necessary. While exact figures are impossible to project because of the variables involved, a month's worth of information could fit comfortably on a single diskette with a capacity of between 241 to 500K bytes. With any less storage space, the entire month's information may or may not fit.

For general accounting purposes, 500K to 1 megabyte of disk storage space is usually convenient. Maintaining the inventory of an electronic parts distributor, though, would probably require more.

Between the publication of the original revision of this book and this second revision, a pattern of manufacturers releasing low-cost CP/M-based computers with minimal disk storage capacities has evolved. Heath/Zenith, Osborne, IBM, and Xerox are all such examples. The 160 to 320K of disk storage these machines provide limits their usefulness for many applications. While you might use such machines for word processing, summary statistics or accounting, budget calculations, or personal tasks, they would not be suitable for general accounting, inventory management, or any other task where manipulation of large amounts of data is necessary.

48K Bytes of Main Memory

WordStar, the program used in the writing of this book, occupies 44K bytes of computer memory. This leaves about 4000 characters of information you can scan at one time (without accessing the disk drives). Many programs now require between 32 and 48K bytes of memory to operate, while some may require as much as 64K bytes. Almost all commercially available programs compatible with CP/M-80 will operate in 56K of memory; most will also operate in 48K of memory. Because of the internal structure of CP/M-86, you should probably have between 96 and 128K bytes of main memory to operate efficiently.

Program size is not the only consideration in determining the proper amount of memory, however. Most accounting or database programs must sort the information stored on the diskette in order to process it. A good sort routine uses all of the computer's memory; the more memory available for the sort routine at one time, the faster it will operate. To give you an idea of the amount of memory versus the efficiency of the sort, consider the following figures:

Task	Time Required
An inefficient sort running in 16K bytes of memory sorting 706 items by two fields.	3 hours, 10 minutes
An efficient sort running in 48K bytes of memory sorting 525 items by five fields.	1 minute, 25 seconds

Note that there are other variables involved here in addition to memory. Giving the inefficient sort more memory speeds up the process but only by an hour. This example typifies real business packages sold by computer stores.

Buy a minimum of 48K bytes of memory for your system. Buy 64K bytes if you can afford it. If you are operating CP/M-86, you might want to consider expanding memory to 128K bytes so that you can more easily utilize some of its advanced features.

Character Terminals

The terminal should have a video screen that can display at least 24 lines of 80 characters each. This recommendation entails several considerations. One of the most practical uses of a computer system is *word processing*: writing, storing, and retrieving textual information. Terminal screen size determines how much of the document or record you can create or edit at a time. A terminal which shows only 16 lines of 64 characters can have a maximum display of 1024 characters at a time. A terminal which shows 24 lines of 80 characters can show you 1920 viewed at a time, almost twice as many characters.

The number of characters on the screen at one time can enhance or impair readability. The program may leave extra space between blocks of information, making it more readable, or it may pack characters close together, making the display confusing. Some of the more sophisticated word processing programs use two or more lines of the screen display to show you the status of a document being edited. WordStar, for example, provides the following information on your current location:

```
    A:CPM8   PAGE 5 LINE 26 COL 35                    INSERT ON
L----!----!----!----!----!----!----!----!----!----!----------R
```

The example shows you are editing the file CPM8 on drive "A," page 5, line 26, and the cursor is in column 35. INSERT ON means you are inserting information into some previously prepared text. The following line shows the left (L) and right (R) margins and tab stops (!). This is valuable information, but if it occupies two lines of a 16-line screen, there is little room left for the document.

Unfortunately, many software creators have assumed the least common denominator (the 16 × 64 terminal) for their programs; users of 24 × 80 character terminals waste some terminal capabilities with such programs.

Remember, a printed page of elite type is 66 lines long by 102 characters wide, 85 lines long in pica. If the prime purpose of your computer system is to automate the paperwork process, you will want to choose a display which most closely approximates your finished paperwork. While such terminals generally cost more, their convenience more than pays for itself in time saved.

Another consideration in choosing a terminal is the type and layout of the keyboard. Almost every terminal being sold has a slightly different arrangement of the keys. Some mimic the IBM Selectric keyboard, while others have their own layout. Some things to watch for in a keyboard are

- Are the shift and control keys conveniently located and not a hindrance when typing?
- Are there cursor arrow keys to move the display cursor?

- Is the RETURN key (sometimes labeled ENTER) conveniently located?
- How is the "touch" of the keyboard? Does the keyboard require force to assure keystrokes are entered?

One important thing to look for is the location of the keyboard in relation to the display screen. If the center of the keyboard is slightly set off from the center of the screen, you might find that you consistently put your hands on the wrong keys. Therefore, a detachable keyboard is preferable to one that is fixed and off-center.

A Printer

Recall that we described diskettes as fragile and temporary records that you must copy to insure against accidental loss of the information. The information stored on diskettes may be displayed on a CRT, but it is often convenient to have a printed record. We recommend using a printer for a number of reasons.

The Internal Revenue Service (IRS) does not always consider computer documents (files on diskettes) as suitable material for an audit. In many cases, the use of a computer for accounting or information storage has not kept the IRS from demanding *printed* copies of the information. If the agency allowed computer audits in all cases, it would have to acquire knowledge of a variety of computers and computer software to insure the integrity of the information. While standards exist for auditing a company's books, the accounting profession has been negligent in adopting and enforcing standards for computerized record keeping. It is unlikely you will find standards adopted outside of large corporations.

Users are often hesitant to commit valuable information to a medium they cannot read. The first copyright case concerning computer software made specific note of the inability of anyone to *read* the information contained with the computer, on diskette, or in Read-Only Memory (ROM). This implies that the same information printed on paper has a different status. Perhaps the current generation of children, growing up with both television and computers, will solve the *official acceptance* problems with information stored as magnetic impulses. Until then, we are limited by a society which places special emphasis on the printed word. The smart computer user creates printed documents to supplement magnetic information storage.

Another reason to keep printed records is the lack of standards among computers. At last count, there were over 50 different formats for diskette storage (all using CP/M), making reliance on magnetic media for all *readers* unrealistic. In fact, even where information can be displayed on a screen, it makes sense to save hard copy versions of the information. Until your eye and mind are trained in using a video display to review information, you will find that you miss errors when reviewing documents on the screen, while you would not miss these errors if you were looking at a printed copy. In short, since you were trained to read looking at paper, it will take a while before you can read as accurately on a video display.

In summation, use a printer to

- Save information for audit purposes

- Make the information portable
- Create human-readable information.

Making paper copies of your data also provides an additional back-up measure. This can be especially effective if the paper copies are not stored in the same place as the diskettes.

Modems

Many people today are finding a modem an indispensable piece of equipment for their computer system. It allows one system to communicate with another over the telephone lines. Modems are available in all price ranges and features, from the simple acoustic coupler to the advanced microprocessor controlled units. Some widely used units are the D.C. Hayes Smartmodem, the PMMI MM-103, and the Novation CAT. Their features include varying data transmission rates, automatic dial/answer, and dialing in either touch-tone or pulse (rotary dial) mode. When buying a modem, make sure you purchase one with everything you need. But do not buy one with more features than you need if a less expensive unit will do. For instance, if you intend to work at only one transmission rate, why buy a higher-priced unit which goes up to a much higher rate when a lesser-priced unit would adequately suit your needs? Likewise, if you need to work with touch-tone dialing, *do not* get a unit that only does pulse dialing. You would be wasting your money, since not all the features you needed would be available.

If you intend to communicate with another remote computer system via telephone with the added ability to send or receive programs, data, and reports and save what you get, then a modem control program is called for. This program is designed to help you make the most efficient use of the modem with your system. Typically, a program will save you the bother of remembering commands to send to the modem and ask you plain questions about what you want to do. For example, you have a Smartmodem and you want to dial a number, using touch-tone dialing. Without a control program, you would have to type AT DT, then type the number. With a control program you would be asked NUMBER? You would then type the number, and the program would automatically send the letters and number for you. Without a control program, you may have to resort to remembering or looking up many different commands. With a control program, this aspect is eliminated. Many programs which use the automatic dial feature allow you to store commonly used phone numbers on your system.

This means that with a few keystrokes, the program will automatically send not only the modem command but also the phone number as well. If you use MCI or Sprint (low-cost long-distance phone services), this feature can be a real timesaver. You type a few keys and the program automatically dials your special number and code.

The real power of a control program is the saving of transmissions onto disk or tape. Suppose you have a friend who wrote a great new game and wanted to give you a copy of it. Without a control program, you must manually enter the game into

your system. But with a control program, your friend sends you the game over the modem, your program puts it in memory as it is received, and then the program is automatically saved to disk or tape. Businesses are finding modems and control programs increasingly valuable in everyday operations such as sending and receiving sales data, reports, and other items. And businesses operating under MP/M with multiple phone lines and modems have found telecommunications capabilities once found only in large systems such as the IBM 370 and similar machines. Common control programs include MODEM7, SMODEM 36, Crosstalk, and ZTERM. MODEM7 and SMODEM 36 are public domain software, with SMODEM 36 having been developed for use with the Smartmodem. Crosstalk is a popular program but with one drawback—it can send and receive data for storage on one system only if the other system is using Crosstalk. ZTERM is considered one of the best programs available today. Unfortunately, it can only be installed on an Apple computer. As is the case with modems, shop around for a control program which meets your own individual needs.

Another common use for modems is accessing two popular timeshare networks, the Source and CompuServe. Users can access the latest news, events, and stock market quotes, learn foreign languages, play games, and take advantage of many other features. To use them, you must have a usable credit card, and pay a cost of between $4 and $10 per hour. If you do not mind the cost, these networks are well worth looking into.

Many free computer bulletin boards are available across the country. A common practice among their subscribers is sending mail to other computer users, listing sales items, items wanted, meetings, and other related news. Much free software is available today on these bulletin boards. Often the only cost is the cost of the telephone call.

If none of the uses described above is planned by you, then a modem would be a complete waste of money. However, if the idea of communicating with other systems, often for little or no cost, appeals to you, then a modem would a wise investment.

PROCEDURE RECOMMENDATIONS

Now here are some recommendations to help you get started using CP/M. These recommendations should help you use CP/M both wisely and efficiently.

Understand what you want the computer to do (see the section on system design at the end of this chapter). Read all manuals, even if you do not understand them all at this point. If you are confused or unsure of your skills at using the computer, request help from an expert on microcomputers, such as your computer dealer.

Figure out how many diskettes you need, both originals and backup diskettes, and format them for use with your version of CP/M.

Create usable copies of all program diskettes. Never use the original diskette. The original copy of the programs (this includes the original CP/M diskette) should be

put in a secure place such as a safe, safe-deposit box, or locked file cabinet. Label all diskettes as they are created or modified. Date the label, indicating the dates you created or modified the diskette.

Use the SYSGEN program to copy CP/M-80 onto every diskette you will use with your system (if you have CP/M-86, you use LDCOPY to copy CP/M-86 loader information onto every diskette's system tracks, then use PIP to copy CPM.SYS onto each diskette). Copy the same CP/M on all diskettes; do not put a 48K CP/M-80 on one diskette and a 56K CP/M-80 on another. You never know when you may have to reboot (warm start) CP/M because of a disk error or an accidental interruption of the computer. If all your diskettes have the same CP/M you will always know how to recover from the error.

Establish a rational backup procedure and stick to it. The following is recommended for business applications: do not trust the computer to remember everything; do not eliminate the current system you use to keep information. Keep any information the computer prints out; file it as you would any other information. Regard the computer as you would any other employee. If you think it is important enough to pay for health insurance for employees, get some for the computer, too; a service contract should guarantee the maximum length of time you will be without a computer.

If you are using the computer to replace a current accounting or information processing system, do not sell or throw away the old system. If the computer provides word processing, do not sell all your typewriters. Likewise, do not lay off every employee who knew the old system. Occasionally your computer will have problems, and you need to be prepared for them.

Computers are productivity boosters, not surrogate employees. Use the computer to augment your operation and increase its efficiency, not to replace people. Making your business or personal data handling more efficient may decrease your need for an individual's participation, but that should not be your intention, nor should you overlook the political ramifications of such an action.

Cut out the "computerese." Why force someone to use a command like PIP when it really means COPY? Use the REN command in CP/M to create program names that are meaningful to you. Here are some starting suggestions.

Change	PIP	to	COPY
	CRUN37	to	RUN
	ED	to	EDIT
	CBAS2	to	COMPILE
	ASM	to	ASSEMBLE
	MOVCPM	to	MOVE-CPM
	WS	to	WORDSTAR

A number of other programs can be changed so you and others using the system can immediately recognize each disk file and not be intimidated by acronyms or

abbreviations. In addition, ask your computer dealer to change (or show you how to change) the following:

Change BDOS ERR ON X: to DISK ERR ON X:

Other small changes can eliminate some of the computer jargon that confuses users. For instance, the following might be useful changes to CP/M:

- For diskettes that include automatic execution instructions whenever CP/M is started (or restarted), CP/M could issue the following message:

  ```
  AUTOMATIC EXECUTION OF program name OCCURRING
  ```

  Since some programs take a long time to load but provide no indication of what is happening, panic sometimes sets in. "What's happening?" is a frequent exclamation at this point and it is usually accompanied by a strong desire to press the RESET button to start over. A message might relieve some of the frustration.

- Programs without adequate techniques to handle errors can be improved by telling the programmer to make sure that the program notifies the user of the error and by providing an option to retry rather than to abort the program and return to CP/M. You should have the error messages in your program rewritten so that you cannot ignore them. Error messages that require you to consult a manual for further interpretation are poor error messages.
- Train everyone who will be using the computer. This book will fulfill a portion of this need. Also, choose a computer dealer who will not desert you but will provide training and education. Learn the computer procedures yourself. It's not wise to have only one employee who can "run the system."

If you implement these suggestions, make sure that you change the appropriate places in the manuals that came with your system. Otherwise, you may end up being more confused than ever.

The microcomputer industry is a young and competitive one. Share your problems with others. There are a number of publications that pride themselves on being advocates of the computer user. These include *InfoWorld, Kilobaud Microcomputing, Creative Computing,* and *Interface Age.* Let others know of your problems and frustrations. Do not continue to buy products from a firm that you are not satisfied with.

THE SYSTEMS APPROACH

This issue is not directly related to CP/M but may affect your ability to use it. Many computer users should not be using computers. Often our society believes it can solve any problem with new and better technology. This just is not true. To

make sure you do not fall into the technology trap, *stop* and analyze your current use of the computer. See if you can answer the following questions:

- Did you buy the computer because someone promised it would solve specific problems? Did you believe that promise?
- Did you buy the computer because you were fascinated by technology?
- Can you do the work the computer does just as efficiently by hand?
- Is the computer used less than an hour a day?
- Does the computer take up too much of your time for the results achieved?
- If you were offered the same deal again, would you decline it?

If you answered yes to any of these questions, chances are you have not developed a systems approach to using your computer. The key word is systems. Systems implies that the computer is used in a systematic, logical fashion.

Consider this example to clarify what we mean by systems approach. You work for a company that sells tennis equipment by mail. You wish to computerize order processing, inventory, and periodic reports of mail and phone orders for your chief executive officer. You can buy a computer and software to perform the order processing, inventory, and report tasks, but is that all you need? Definitely not.

Your company receives orders through the mail and, occasionally, by phone. Your computer system must detail how the computer receives mail and phone information (remember, mail is received only once a day in most companies, while the phone can ring at any time). In addition, you receive the invoices, merchandise, packing slips, and pricing updates from wholesale companies. The computer must receive this information, too. In fact, the computer must have inventory before it can begin selling goods. And how can the computer keep track of order cancellations, queries about order status, lost merchandise, and damaged goods?

Suppose your company employs ten people in addition to its chief executive officer. Do all ten use the computer? Does everyone in the firm have access to all the information stored in the computer? Does everyone in the firm *need* access to all the information stored in the computer?

In practice, a computer is generally a small part in an overall system. Paperwork still exists and gets passed from person to person. In the case of a retail business you can't computerize the merchandise; it still needs to be stored somewhere, identified, and processed for shipping.

To assure that the computer ties in with the other parts of the system, you must talk with the people who sell you your computer and software. If more than one firm is involved, bring them together for the discussion. Make sure that everyone knows you are looking for a complete integration of the computer into your business environment. Deal only with people who accept a systems approach to problems. Remember, a computer dealer who knows everything about wait state, NAND gates, and vectored interrupts but nothing about cash flow, inventory valuation, and charge card procedures, will be of no use to a business like yours.

These discussions will inform you which computer equipment and software you need as well as how the computer will fit into your operating environment. You

may find it helpful to create a systems flowchart to learn how your business really works (track the progress of each physical item your business deals with through every process it goes through).

Once you are positive the computer and software will work for your system instead of against it, proceed with the acquisition. See demonstrations of the equipment and software functioning as it will in your environment. A computer store will use a computer for some portion of their own operations. If not, they are not interested in users, they are hardware freaks, and they are more interested in computers as toys or gimmicks.

Train personnel and prepare the business for the computer before the computer arrives, not afterward. Where will you place it? Does it need a special environment? Does everyone have to learn typing? Many more questions need to be considered *before* the equipment arrives and you try to use it.

What about your old system? If you currently prepare all the order and inventory records by hand, do you stop writing? There are several approaches to introducing a new computer system. You may start using the new system

Cold turkey
: One day you stop using the old system and start using the new one.

Phased in
: You separate each task into its component parts and convert them one at a time. When one part works fine on the new system, you convert the next part. It is a series of small "cold turkeys."

Parallel
: Use your old system and the new system at the same time. You can compare the two side by side to ensure the new system performs the job correctly and completely. You can easily show employees the differences between the systems.

Now look at your projected use of the computer. Is it part of a system? Do the rest of the pieces work with the computer, or is the computer isolated from other procedures?

To reiterate the lessons of this section, remember

- A computer is a tool
- A computer is not a complete solution to any problem
- You control the computer, not vice versa.

Buying a computer is not a necessity. Evaluate how a computer relates to all your business needs.

What Next?

It is somewhat amazing that an operating system that only has a handful of built-in commands takes more than 200 pages to describe in full. In a way, this is an indication of the complexity that computer software users must still deal with.

At the same time, however, it is hoped that this book presents more information than you will ever need about CP/M-80 and CP/M-86. If you use CP/M on your computer and just want to learn more about it, reading through this book one or more times will probably give you enough background to deal with the day-to-day details that the use of CP/M implies. In addition, the layout of this book should be clear enough so that when a particular question does come up, you can turn right to the answer.

Therefore, for most CP/M users, the answer to the question "what next?" is to use the computer and keep this book nearby.

If you are curious, a computer hobbyist, or perhaps even a computer professional, you should not stop after you have finished reading this book. Use the annotated bibliography in the appendix of this book to find more information of interest to you. Use the chapter on the technical aspects of CP/M-80 and CP/M-86 to begin experimenting. Ask your local computer dealer if there are any user groups in your area, and if so, join one. If there are none, consider starting one. In short, do not consider this book the last word on CP/M—it isn't.

APPENDIX

A CP/M COMMAND SUMMARY

This appendix summarizes the command line format and the function of each CP/M-80 and CP/M-86 built-in and transient command. The commands are listed in alphabetical order.

In the space below, fill in the command line format for your disk copy and disk format (initialize) programs. *See* Chapter 5 for more information.

ASM Command Lines

ASM filename<cr>

Assemble the file filename.ASM; use the currently logged disk for all files.

ASM filename.opt<cr>

Assemble the file filename.ASM on drive o: (A:,B:,...,P:). Write HEX file on drive p: (A:,B:,...,P:), or skip if p: is Z:.

Write PRN file on drive t: (A:,B:,...,P:), send to console if p: is X:, or skip if p: is Z:.

ASM-86 Command Lines

ASM-86 filename.typ $options

Assemble the file filename.A86 (or filename.typ if type is specified) using the options specified.

Ad	Source drive (filename.A86)
Hd	Hex code drive (filename.H86)
Pd	Print drive (filename.LST)
Sd	Symbol table drive (filename.SYM)
Fx	Format of hex file (D = Digital Research, I = label)

DDT or DDT-86 Command Lines

DDT<cr> or DDT-86<cr>

Loads DDT and waits for DDT commands.

DDT :filename.typ<cr> or DDT-86 :filename.typ<cr>

Loads DDT into memory and also loads filename.typ from drive x: into memory for examination, modification, or execution.

DDT Command Summary

Assss

Enter assembly language statements beginning at hexadecimal address ssss.

Bssss,ffff,cccc

Compare blocks of memory (DDT-86).

D

Display the contents of the next 192 bytes of memory.

Dssss,ffff

Display the contents of memory starting at hexadecimal address ssss and finishing at hexadecimal address ffff.

Efilename.type

Load program for execution (DDT-86).

Fssss,ffff,cc

Fill memory with the 8-bit hexadecimal constant cc starting at hexadecimal address ssss and finishing with hexadecimal address ffff.

Fwssss,ffff,cccc

Fill memory with 16-bit hex constant cccc (DDT-86).

G

Begin execution at the address contained in the program counter.

G,bbbb

Set a breakpoint at hexadecimal address bbbb, then begin execution at the address contained in the program counter.

G,bbbb,cccc

Set breakpoints at hexadecimal addresses bbbb and cccc, then begin execution at the address contained in the program counter.

Gssss

Begin execution at hexadecimal address ssss.

Gssss,bbbb

Set a breakpoint at hexadecimal address bbbb, then begin execution at hexadecimal address ssss.

Hx,y

Hexadecimal sum and difference of x and y.

Ifilename.typ

Set up the default file control block using the name filename.typ.

L

List the next eleven lines of assembly language program disassembled from memory.

Lssss

List eleven lines of assembly language program disassembled from memory starting at hexadecimal address ssss.

Lssss,ffff

List the assembly language program disassembled from memory starting at hexadecimal address ssss and finishing at hexadecimal address ffff.

Mssss,ffff,dddd

Move the contents of the memory block starting at hexadecimal address ssss and ending at hexadecimal address ffff to the block of memory starting at hexadecimal address dddd.

R

Read a file from disk into memory (use "I" command first).

Rnnnn

Read a file from disk into memory beginning at the hexadecimal address nnnn higher than normal (use "I" command first).

Rfilename (DDT-86)

Read file into memory for debugging.

Sssss

Display the contents of memory at hexadecimal address ssss and optionally change the contents.

Tnnnn

Trace the execution of (hexadecimal) nnnn program instructions.

Tsnnn

Trace execution and show all registers (DDT-86).

Unnnn

Execute (hexadecimal) nnnn program instructions, then stop and display the CPU register's contents.

Usnnn

Execute nnnn program instructions, then display all registers (DDT-86).

V

Show memory layout after disk read (DDT-86).

Wfilename.typ,ssss,ffff

Write memory from ssss to ffff into filename.typ (DDT-86).

X

Display the CPU register's contents.

Xr

Display the contents of CPU register or Flag r and optionally change it.

DIR Command Lines

DIR x:<cr>

Displays directory of all files on drive x:. Drive x: is optional; if omitted, the currently logged drive is used.

DIR x:filename.typ<cr>

Displays directory of all files on drive x: whose names match the ambiguous or unambiguous filename.typ. Drive x: is optional; if omitted, the currently logged drive is used.

DIRS (CP/M-86)

Displays names of system files. Otherwise it functions like DIR.

DUMP Command Line (CP/M-80)

DUMP x:filename.typ<cr>

Displays the hexadecimal representations of each byte stored in the file filename.typ on drive x:. If filename.typ is ambiguous, displays the first file which matches the ambiguous file name.

ED Command Line

ED:filename.typ<cr>

Invokes the editor, which then searches for filename.typ on drive x: and creates a temporary file x:filename.$$$ to store the edited text. The filename.typ is unambiguous. Drive x: is optional; if omitted, the currently logged drive is assumed.

ED Command Summary

NOTE: Non-alphabetic commands follow the "Z" command.

nA

Append lines. Moves "n" lines from original file to edit buffer.

0A moves lines until edit buffer is at least half full.

+/−B

Begin/Bottom. Moves CP.

+B moves CP to beginning of edit buffer
−B moves CP to end of edit buffer.

+/−nC

Move by characters. Move CP by "n" character positions.

+ moves forward
−moves backward.

+/−nD

Delete characters. Deletes "n" characters before or after the CP in the edit buffer.

+ deletes before the CP
−deletes after the CP.

E

End. Ends edit, closes files, and returns to CP/M; normal end.

nFstring^Z

Find string. Find the "n"th occurrence of string, beginning the search after the CP.

H

Move to head of edited file. Ends edit, renames files, and then edits former temporary file.

I<cr>

Enter insert mode. Text from keyboard goes into edit buffer after the CP; exit with CONTROL-Z.

Istring^Z

Insert string. Inserts string in edit buffer after the CP.

Istring<cr>

Insert line. Inserts string and CRLF in the edit buffer after the CP.

nJfindstring^Zinsertstring^Zendstring^Z

Juxtaposition. Beginning after the CP, finds findstring, inserts insertstring after it, then deletes all following characters up to but not including endstring; repeats until performed "n" times.

+/−nK

Kill lines. Deletes "n" lines.
+ deletes after the CP
− deletes before the CP.

+/−nL

Move by lines. Moves the CP to the beginning of the line it is in, then moves the CP "n" lines forward or backward.
+ moves forward
− moves backward.

nMcommandstring^Z

Macro command. Repeats execution of the ED commands in commandstring "n" times. "n" = 0,"n" = 1, or "n" absent repeats execution until error occurs.

nNstring^Z

Find string with autoscan. Finds the "n"th occurrence of string, automatically appending from original file and writing to temporary file as necessary.

O

Return to original file. Empties edit buffer, empties temporary file, returns to beginning of original file, ignores previous ED commands.

+/−nP

Move CP and print pages. Moves the CP forward or backward one page, then displays the page following the CP. "nP" displays "n" pages, pausing after each.

Q

Quit edit. Erases temporary file and block move file, if any, and returns to CP/M; original file is not changed.

R<cr>

Read block move file. Copies the entire block move file X$$$$$$$.LIB from disk and inserts it in the edit buffer after the CP.

Rfilename<cr>

Read library file. Copies the entire file filename with extension LIB from the disk and inserts it in the edit buffer after the CP.

nSfindstring^Zreplacestring^Z

Substitute string. Starting at the CP, repeats "n" times: finds findstring and replaces it with replacestring.

+/−nT

Type lines. Displays "n" lines.

+ displays the "n" lines after the CP

− displays the "n" lines before the CP.

If the CP is not at the beginning of a line

0T displays from the beginning of the line to the CP

T displays from the CP to the end of the line

0TT displays the entire line without moving the CP.

+/−U

Upper case translation. After +U command, alphabetic input to the edit buffer is translated from lower-case to upper-case; after −U, no translation occurs.

0V

Edit buffer free space/size. Displays the decimal number of free (empty) bytes in the edit buffer and the total size of the edit buffer.

+/−V

Verify line numbers. After +V, a line number is displayed with each line displayed; ED's prompt is then preceded by the number of the line containing the CP. After −V, line numbers are not displayed, and ED's prompt is "*."

nW

Write lines. Writes first "n" lines from the edit buffer to the temporary file; deletes these lines from the edit buffer.

nX

Block transfer (Xfer). Copies the "n" lines following the CP from the edit buffer to the temporary block move file X$$$$$$$.LIB; adds to previous contents of that file.

nZ

Sleep. Delays execution of the command which follows it. Larger "n" gives longer delay, smaller "n" gives shorter delay.

n:

Move CP to line number "n." Moves the CP to the beginning of line number "n" (*see* "+/−V").

:m

Continue through line number "m." A command prefix which gives the ending point for the command which follows it. The beginning point is the location of the CP (*see* "+/−V").

+/−n

Move and display one line. Abbreviated form of +/−nLT.

ERA Command Lines

ERA x:filename.typ<cr>

Erase the file filename.typ on the disk in drive x:. The filename and/or typ can be ambiguous. Drive x: is optional; if omitted, the currently logged drive is used.

ERA x:*.*<cr>

Erase all files on the disk in drive x:. Drive x: is optional; if omitted, the currently logged drive is used.

GENCMD Command Line (CP/M-86)

GENCMD filename options

Convert hexadecimal object file type (H86) into executable CMD type file. Options include 8080 model and setting of CODE, DATA, STACK, and EXTRA segment registers.

Line Editing Commands

CONTROL-C

Restarts CP/M if it is the first character in command line. Called *warm start*.

CONTROL-E

Moves to beginning of next line. Used for typing long commands.

CONTROL-H or BACKSPACE

Deletes one character and erases it from the screen (CP/M version 2.0 and newer).

CONTROL-J or LINE FEED

Same as CARRIAGE RETURN (CP/M version 2.0 and newer).

CONTROL-M

Same as CARRIAGE RETURN (<cr>).

CONTROL-P

Turns on the list device (usually your printer). Type it again to turn off the list device.

CONTROL-R

Repeats current command line (useful with version 1.4); it verifies the line is

corrected after you delete several characters (CP/M version 1.4 and newer).

CONTROL-S

Temporarily stops display of data on the console. Press any key to continue.

CONTROL-U or CONTROL-X

Cancels current command line (CP/M version 1.4 and newer).

RUBOUT (RUB) or DELETE (DEL)

Deletes one character and echoes (repeats) it.

LOAD Command Line (CP/M-80)

LOAD x:filename<cr>

Reads the file filename.HEX on drive x: and creates the executable program file filename.COM on drive x:.

MOVCPM Command Line (CP/M-80)

MOVCPM<cr>

Prepare a new copy of CP/M which uses all of memory; give control to the new CP/M, but do not save it on disk.

MOVCPM nn<cr>

Prepare a new copy of CP/M which uses "nn" K bytes of memory; give control to the new CP/M, but do not save it on disk.

MOVCPM * *<cr>

Prepare a new copy of CP/M, which uses all of memory, to be saved with SYSGEN or SAVE.

MOVCPM nn *<cr>

Prepare a new copy of CP/M, which uses "nn" K bytes of memory, to be saved with SYSGEN or SAVE.

The "nn" is an integer decimal number. It can be 16 through 64 for CP/M 1.3 or 1.4. For CP/M 2.0 and newer "nn" can be 20 through 64.

PIP Command Lines

PIP<cr>

Loads PIP into memory. PIP prompts for commands, executes them, then prompts again.

PIP pipcommandline<cr>

Loads PIP into memory. PIP executes the command pipcommandline, then exits to CP/M.

PIP Command Summary

x:new.typ=y:old.typ[p]<cr>

Copies the file old.typ on drive y: to the file new.typ on drive x:, using parameters p.

x:new.typ=y:old1.typ[p],z:old2.typ[q]<cr>

Creates a file new.typ on drive x: which consists of the contents of file old1.typ on drive y: using parameters p followed by the contents of file old2.typ on drive z: using parameters q.

x:filename.typ=dev:[p]<cr>

Copies data from device dev: to the file filename.typ on drive x:.

dev:=x:filename.typ[p]<cr>

Copies data from filename.typ on drive x: to device dev:.

dst:=src:[p]<cr>

Copies data to device dst: from device src:.

PIP Parameter Summary

B	Specifies block mode transfer.
Dn	Deletes all characters after the "n"th column.
E	Echoes the copying to the console as it is being performed.
F	Removes form feed characters during transfer.
Gn	Directs PIP to copy a file from user area "n."
H	Checks for proper Intel Hex File format.
I	Ignores any *:00* records in Intel Hex File transfers.
L	Translates upper-case letters to lower-case.
N	Adds a line number to each line transferred.
O	Object file transfer (ignores end-of-file markers).
Pn	Issues page feed after every "n"th line.
Qs^Z	Specifies quit of copying after the string "s" is encountered.
R	Directs PIP to copy from a system file.
Ss^Z	Specifies start of copying after the string "s" is encountered.
Tn	Sets tab stops to every "n"th column.

U	Translates lower-case letters to upper-case.
V	Verifies copy by comparison after copy finished.
W	Directs PIP to copy onto an R/O file.
Z	Zeroes the "parity" bit on ASCII characters.

PIP Destination Devices

CON:	PUN:	LST:	Logical devices
TTY:	PTP:	LPT:	
CRT:	UP1:	UL1:	
UC1:	UP2:		Physical devices
OUT:	PRN:		Special PIP devices

PIP Source Devices

CON:	RDR:		Logical devices
TTY:	PTR:		
CRT:	UR1:		
UC1:	UR2:		Physical devices
NUL:	EOF:	INP:	Special PIP devices

REN Command Line

REN newname.typ=oldname.typ<cr>

Finds the file oldname.typ and renames it newname.typ.

SAVE Command Line (CP/M-80)

SAVE nnn x:filename.typ<cr>

Saves a portion of the Transient Program Area of memory in the file filename.typ on drive x: where nnn is a decimal number representing the number of pages of memory. Drive x: is the option drive specifier.

STAT Command Lines

STAT<cr>

Displays attributes and amount of free space for all diskette drives accessed since last warm or cold start.

STAT x:<cr>

Displays amount of free space diskette in drive x:.

STAT x:filename.typ<cr> (CP/M 2.0 and newer)

Displays size and attributes of file(s) filename.typ on drive x:. filename.typ may be ambiguous. x: is optional; if omitted, currently logged drive is assumed.

STAT x:filename.typ $atr<cr>

Assigns the attribute atr to the file(s) filename.typ on drive x:. File filename.typ may be ambiguous. Drive x: is optional; if omitted, currently logged drive is assumed.

STAT DEV:<cr>

Reports which physical devices are currently assigned to the four logical devices.

STAT VAL:<cr>

Reports the possible device assignments and partial STAT command line summary.

STAT log:=phy:<cr>

Assigns the physical device phy: to the logical device log: (may be more than one assignment on the line; each should be set off by a comma).

STAT USR:<cr> (CP/M 2.0 and newer)

Reports the current user number as well as all user numbers for which there are files on currently logged disks.

STAT x:DSK:<cr> (CP/M 2.0 and newer)

Reports the characteristics of disk drive x:.

STAT x:=R/O<cr> (CP/M 1.4 and newer)

Assigns a temporary write-protect status to drive x:.

SUBMIT Command Lines

SUBMIT filename<cr>

Creates a file $$$.SUB which contains the commands listed in filename.SUB; CP/M then executes commands from this file rather than the keyboard.

SUBMIT filename parameters<cr>

Creates a file $$$.SUB which contains commands from the file filename.SUB; certain parts of the command lines in filename.SUB are replaced by parameters during creation of $$$.SUB. CP/M then gets commands from this file rather than the keyboard.

SYSGEN Command Line

SYSGEN<cr>

Loads the SYSGEN program to transfer CP/M from one diskette to another.

TOD (CP/M-86)

TOD Options

Sets or displays time of day clock.

TYPE Command Line

TYPE x:filename.typ<cr>

Displays the contents of file filename.typ from drive x: on the console.

USER Command Line

USER n<cr>

Sets the User Number to "n," where "n" is an integer decimal number from 0 to 15, inclusive.

x: Command Line

x:<cr>

Changes the currently logged disk drive to drive x:. Drive x: can be "A" through "P."

APPENDIX

B ASCII Character Codes

The American Standard Code for Information Interchange (ASCII) consists of a set of 96 displayable characters and 32 non-displayed characters. Most CP/M systems use at least a subset of the ASCII character set. When CP/M stores characters on a diskette as text, the ASCII definitions are used.

Several of the CP/M utility programs use the ASCII Character Code. Text created using ED is stored as ASCII characters on diskette. DDT and DDT-86, when displaying a "dump" of the contents of memory, display both the hexadecimal and ASCII representation of memory's contents.

ASCII does not use an entire byte of information to represent a character. ASCII is a seven-bit code, and the eighth bit is often used for *parity*. Parity is an error-checking method which assures that the character received is the one transmitted. Many microcomputers and microcomputer devices ignore the *parity bit*, while others require one of the following two forms of parity:

Even Parity

The number of binary 1's in a byte is always an even number. If there is an odd number of 1's in the character, the parity bit will be a 1; if there is an even number of 1's in the character, the parity bit is made a 0.

Odd Parity

The number of binary 1's in a byte is always an odd number. If there is an even number of 1's in the character, the parity bit will be a 1; if there is an odd number of 1's in the character, the parity bit is made a 0.

Alternative ways of *coding* the information stored by the computer include the eight-bit EBCDIC (Extended Binary Coded Decimal Interchange Code), used by IBM, and a number of *packed binary* schemes, primarily used to represent numerical information.

Table B-1. ASCII Character Codes

				b7 →	0	0	0	0	1	1	1	1
				b6 →	0	0	1	1	0	0	1	1
				b5 →	0	1	0	1	0	1	0	1
b4	b3	b2	b1	Row \ Col.	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	'	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

NUL	Null	DC1	Device control 1
SOH	Start of heading	DC2	Device control 2
STX	Start of text	DC3	Device control 3
ETX	End of text	DC4	Device control 4
EOT	End of transmission	NAK	Negative acknowledge
ENQ	Enquiry	SYN	Synchronous idle
ACK	Acknowledge	ETB	End of transmission block
BEL	Bell, or alarm	CAN	Cancel
BS	Backspace	EM	End of medium
HT	Horizontal tabulation	SUB	Substitute
LF	Line feed	ESC	Escape
VT	Vertical tabulation	FS	File separator
FF	Form feed	GS	Group separator
CR	Carriage return	RS	Record separator
SO	Shift out	US	Unit separator
SI	Shift in	SP	Space
DLE	Data link escape	DEL	Delete

TABLE B-2. ASCII Character Codes in Ascending Order

Hexadecimal	Binary	ASCII	Hexadecimal	Binary	ASCII
00	000 0000	NUL	30	011 0000	0
01	000 0001	SOH	31	011 0001	1
02	000 0010	STX	32	011 0010	2
03	000 0011	ETX	33	011 0011	3
04	000 0100	EOT	34	011 0100	4
05	000 0101	ENQ	35	011 0101	5
06	000 0110	ACK	36	011 0110	6
07	000 0111	BEL	37	011 0111	7
08	000 1000	BS	38	011 1000	8
09	000 1001	HT	39	011 1001	9
0A	000 1010	LF	3A	011 1010	:
0B	000 1011	VT	3B	011 1011	;
0C	000 1100	FF	3C	011 1100	<
0D	000 1101	CR	3D	011 1101	=
0E	000 1110	SO	3E	011 1110	>
0F	000 1111	SI	3F	011 1111	?
10	001 0000	DLE	40	100 0000	
11	001 0001	DC1	41	100 0001	A
12	001 0010	DC2	42	100 0010	B
13	001 0011	DC3	43	100 0011	C
14	001 0100	DC4	44	100 0100	D
15	001 0101	NAK	45	100 0101	E
16	001 0110	SYN	46	100 0110	F
17	001 0111	ETB	47	100 0111	G
18	001 1000	CAN	48	100 1000	H
19	001 1001	EM	49	100 1001	I
1A	001 1010	SUB	4A	100 1010	J
1B	001 1011	ESC	4B	100 1011	K
1C	001 1100	FS	4C	100 1100	L
1D	001 1101	GS	4D	100 1101	M
1E	001 1110	RS	4E	100 1110	N
1F	001 1111	US	4F	100 1111	O
20	010 0000	SP	50	101 0000	P
21	010 0001	!	51	101 0001	Q
22	010 0010	"	52	101 0010	R
23	010 0011	#	53	101 0011	S
24	010 0100	$	54	101 0100	T
25	010 0101	%	55	101 0101	U
26	010 0110	&	56	101 0110	V
27	010 0111	'	57	101 0111	W
28	010 1000	(	58	101 1000	X
29	010 1001	)	59	101 1001	Y
2A	010 1010	*	5A	101 1010	Z
2B	010 1011	+	5B	101 1011	[
2C	010 1100	,	5C	101 1100	\
2D	010 1101	-	5D	101 1101	]
2E	010 1110	.	5E	101 1110	^
2F	010 1111	/	5F	101 1111	_

TABLE B-2. ASCII Character Codes in Ascending Order (continued)

Hexadecimal	Binary	ASCII	Hexadecimal	Binary	ASCII
60	110 0000		70	111 0000	p
61	110 0001	a	71	111 0001	q
62	110 0010	b	72	111 0010	r
63	110 0011	c	73	111 0011	s
64	110 0100	d	74	111 0100	t
65	110 0101	e	75	111 0101	u
66	110 0110	f	76	111 0110	v
67	110 0111	g	77	111 0111	w
68	110 1000	h	78	111 1000	x
69	110 1001	i	79	111 1001	y
6A	110 1010	j	7A	111 1010	z
6B	110 1011	k	7B	111 1011	{
6C	110 1100	l	7C	111 1100	|
6D	110 1101	m	7D	111 1101	}
6E	110 1110	n	7E	111 1110	~
6F	110 1111	o	7F	111 1111	DEL

APPENDIX

C Comparison of CP/M-80 Versions 1.3, 1.4, and 2.2, And CP/M-86

TABLE C-1. Line Editing Commands

Command	Version 1.3	Version 1.4	Version 2.2	CP/M-86
CONTROL-C	Yes	Yes	Yes	Yes
CONTROL-E	No	Yes	Yes	Yes
CONTROL-H or BACKSPACE	No	No	Yes	Yes
CONTROL-J or LINE FEED	No	No	Yes	Yes
CONTROL-M or CARRIAGE RETURN	Yes	Yes	Yes	Yes
CONTROL-P	Yes	Yes	Yes	Yes
CONTROL-R	No	Yes	Improved	Improved
CONTROL-S	Yes	Yes	Yes	Yes
CONTROL-U	Yes	Yes	Improved	Improved
CONTROL-X	No	Same as CONTROL-U	BACKSPACE and ERASE	Yes
DELETE or RUBOUT	Yes	Yes	Yes	Yes

TABLE C-2. New or Changed Commands

Command	Version 1.3	Version 1.4	Version 2.2	CP/M-86
DIR	Yes	Yes	New: Displays 4 file names per line. Displays only DIR file names. Displays file names for current user only.	Yes
DIRS	No	No	No	Displays all files, including system files.
ED	Yes	New: +/−V 0X 0V nnnn: nX :mmmm Singly only: E,H,Q,OTabs always echo as spaces.	New: +V is default. Cannot alter R/O file. Cannot access SYS file.	Yes
ERA	Yes	New: Asks ALL? after ERA *.*	New: Erases files of current user only.	Yes
PIP	Yes	New: Option parameters. Physical device names. Ambiguous file names.	New parameters: Gn R W	Yes
SAVE	Yes	Yes	New: Does not alter user memory (TPA).	No (*see* DDT-86)
STAT	Yes Displays free disk space only.	New: Displays size of each file. log:=phy: x:=R/O Detects switched disks. Displays free space on all active disks.	New: VAL: menu DSK: x:DSK: USR: $S File attributes: R/W, R/O DIR, SYS	Yes
SUBMIT	Yes	Yes	New: XSUB	Yes (No XSUB)
USER	No	No	Yes	Yes

TABLE C-3. Disk Differences

Item	Version 1.3	Version 1.4	Version 2.2	CP/M-86
Maximum number of drives	2	4	16	16
Maximum storage per drive	1 Mbyte	1 Mbyte	16 Mbyte	16 Mbyte
Maximum number of files	64	64	Expandable	Expandable
Access method	Sequential	Sequential	Sequential or random	Sequential or random
Location of disk characteristics	BDOS	Disk parameter block	BIOS	BIOS

TABLE C-4. New or Changed BDOS Functions

Function	Version 1.3	Version 1.4	Version 2.2	CP/M-86
6-Direct Console I/O	No	No	Yes	Yes
10-Read Console buffer	Yes	Yes	Improved line editing	Improved line editing
12-(see right)	Lift disk head	Lift disk head	Return version number	Return version number
15-Open File	Yes	Yes	Improved	Improved
17-Search for First	Yes	Yes	Improved	Improved
18-Search for Next	Yes	Yes	Improved	Improved
19-Delete File	Yes	Yes	Improved	Improved
22-Make File	Yes	Yes	Improved	Improved
23-Rename File	Yes	Yes	Improved	Improved
24-Return Login vector	Yes	Yes	Improved	Improved
28 through 36	No	No	Yes	Yes

APPENDIX

D CP/M Prompts

X>	CP/M waiting for command; drive x: is currently logged drive.
nx>	MP/M waiting for command; drive x: is currrently logged drive; current user number is "n."
*	PIP waiting for command.
*	ED waiting for command.
nnnn:*	ED waiting for command; character pointer is at line number "nnnn."
*	Also used by Microsoft BASIC, EDIT, FORTRAN, COBOL, and Pascal when waiting for a command.
-	DDT waiting for command.

In the space below, write in the prompts of other programs you use.

APPENDIX

E Diskette Selections

Within each category below, diskettes may be single-density or double-density. Be sure to choose the proper density for your system. Double-density systems can usually be operated in single-density mode, so you may be using both varieties if you have such a system.

Systems Which Use 8-Inch Soft-Sectored Diskettes

Altos
Cromemco FDC Controller
Cromemco System 3
Delta
Digital Microsystems
Discus
Dynabyte DB8/4
Godbout
iCOM 3712, 3812
IMS 8000
IMSAI FDC2
IMSAI VDP-80
Intecolor (ISC) 8063, 8360, 8963
Intel MDS
Micromation
Morrow Discus
Mostek
Ohio Scientific C3
Pertec PCC 2000

Processor Technology Helios
Radio Shack TRS-80 Model II
Radio Shack TRS-80 Model I/Micromation
Radio Shack TRS-80 Model I/Omikron
Radio Shack TRS-80 Model I/Shuffleboard
Research Machines
SD Systems
Spacebyte
Tarbell
TEI
Thinker Toys
TRS-80. *See* Radio Shack
Vector Graphic System 2800

Systems Which Use 8-Inch Hard-Sectored Diskettes

MITS 3200, 3202
Processor Technology Helios II

Systems Which Use 5¼-Inch Soft-Sectored Diskettes

Apple/SoftCard
AVL Eagle
BASF System 7100
Cromemco Z2D
DEC VT180
Digi-Log Microterm II
Durango F-85
Gnat
IBM Personal Computer
Hewlett-Packard HP-80 Series
Heath/Zenith double-density
iCOM 2411 Micro Floppy
IMS 5000
IMSAI VDP-40, -42, -44
Intertec SuperBrain
Kontron PSI-80
MSD
Osborne 1
Polymorphic 8813

Quay 500, 520
Radio Shack TRS-80 Model I
Radio Shack TRS-80 Model
I/FEC Freedom
Radio Shack TRS-80 Model
I/Omikron
RAIR
Research Machines
Sanco 7000
SD Systems
SuperBrain. *See* Intertec
TEI
Quay
Vector Graphic (current systems)
Xerox 820

Systems Which Use 5¼-Inch Hard-Sectored 10-Sector Diskettes

Heath H8 and H17, H27
Heath H89
Horizon
Meca
North Star Horizon
Vista V80
Vista V200
Zenith Z89

Systems Which Use 5¼-Inch Hard-Sectored 16-Sector Diskettes

Blackhawk (40 TPI)
CDS Versatile 3B (40 TPI)
CDS Versatile 4 (100 TPI)
COMPAL-80 (100 TPI)
Dynabyte (Some Models)
Exidy Sorcerer (100 TPI)
Micropolis Mod I (40 TPI)
Micropolis Mod II (100 TPI)
Nylac (40 TPI, 100 TPI)
REX (40 TPI)
Sorcerer. *See* Exidy
Vector Graphic (100 TPI)
Vector MZ (100 TPI)
Versatile. *See* CDS

APPENDIX

F Annotated Bibliography

CP/M General

Ballinger, Charles. "HDOS or CP/M?" *Interface Age*, September 1980, pp. 88-91.

Compares two disk operating systems available for Heath computers.

Brigham, Bruce, ed. *CP/M Summary Guide*. Glastonbury, Ct.: Rainbow Associates, 1980.

The book reprints and summarizes the commands available with CP/M, DESPOOL, MAC, TEX, CBASIC, and BASIC-80 from Microsoft. A useful computer-side reference.

Epstein, Jake, and Terry, Chris. "Introduction to CP/M; The CP/M Connection." *S100 Microsystems*, September/October 1980, pp. 10-32.

Continuation of a series of CP/M articles.

Fernandez, Judi, and Ashley, Ruth. *Using CP/M*. New York: John Wiley Sons, 1980.

A self-teaching book that uses a question and answer format while conveying technical information. A quick primer of CP/M commands and syntax.

Fritzon, Richard. "The New CP/M: Is It Worth It?" *Kilobaud Microcomputing*, July 1980, p. 66.

Quickly summarizes the differences between versions 1.4 and 2.0 of CP/M.

Miller, Alan. "Diagnostics Package for CP/M." *Interface Age*, October 1980, p. 104.

Reviews the Supersoft CP/M Diagnostics package; these programs test memory, CPU, disk, printer, and console.

North, Steve. "The CP/M Disk Operating System." *Creative Computing*, November/December 1978, pp. 52-53.

A short summary of the CP/M operating system and the programs that accompany it.

Stewart, John. "CP/M Primer; A Most Sophisticated Operating System." *Kilobaud Microcomputing*, April 1978, pp. 30-34.

An early detailed description of the CP/M operating system.

"Upgraded CP/M Floppy Disc Operating System." *Dr. Dobbs*, November 1976, p. 51.

A detailed summary of the features of CP/M revision 1.4. Includes descriptions of the accompanying 1.4 manuals.

Warren, Jim. "First Word on a Floppy-Disc Operating System." *Dr. Dobbs*, April, 1976, p. 5.

Provides first available information on CP/M.

_____. "The Time for Floppy's Is Just About Now!" *Dr. Dobbs*, August, 1976, p.5.

Reviews the available floppy disk systems of the time and summarizes CP/M's features.

Zaks, Rodnay. *The CP/M Handbook with MP/M*. Berkeley: Sybex, 1980.

Discusses CP/M and MP/M commands, programs, and facilities.

CP/M Compatible Software

Collins, Rosann; Hines, Theodore; and Rowan, George. "Manipulating Pencil Files; Convert Them to BASIC." *Creative Computing*, August 1979, pp. 98-99.

Illustrates how to use files created with Electric Pencil using Processor Technology's BASIC; applicable to many different CP/M compatible BASICs.

Craig, John. "A New Kind of Pencil!" *Creative Computing*, February 1979, pp. 30-33.

Thoroughly reviews the Electric Pencil word processor.

Didday, Rich. "Universal Data Entry System; In a Car Pooling Application." *Creative Computing*, May 1980, pp. 102-110.

Reviews UDE from the Software Store and demonstrates the program.

Eubanks, Gordon. "Notes on CP/M's BASIC-E." *Dr. Dobbs* 19:35.

Letter to the editor describing the versions of BASIC-E, its public domain status, and how to obtain revised versions.

(NOTE: Gordon Eubanks developed BASIC-E for a Master's thesis project; it is the ancestor of CBASIC and CBASIC2.)

Fitzgerald, Jim. "Off-the-Shelf Word-Processing System." *Kilobaud Microcomputing*, September 1980, pp. 92-94.

Describes a CP/M system using Electric Pencil.

Foster, Charlie. "Pascal with a Z80." *Interface Age*, November 1980, pp. 60-62.

Reviews Pascal/Z from Ithaca Intersystems, emphasizing its differences with UCSD Pascal.

Hallen, Rod. "Super Word Processors." *Kilobaud Microcomputing*, June 1980, pp. 214-217.

Summarizes the features of ED, EDIT, Electric Pencil, WordStar, and The Magic Wand; makes no conclusions or recommendations. Provides a brief bibliography of articles on word processing.

_____. "Tarbell Disk BASIC." *Kilobaud*, May 1980, pp. 168-170.

Summarizes the features of Tarbell Disk BASIC; emphasizes the features of Tarbell BASIC in comparison with other interpreters.

_____. "The Battle of the Word Processors." *Creative Computing*, November 1979, pp. 48-53.

Reviews ED, TEX, EDIT, and Electric Pencil.

Hamilton, R.W. "WPDaisy Word Processing System." *Creative Computing*, May 1979, pp. 36-41.

Reviews a little-known, but powerful, word processing system.

Hart, Glenn. "New BASIC from Tarbell." *Creative Computing*, January 1980, pp. 20-23.

Reviews a slightly different BASIC available for CP/M systems.

_____. "Magic Wand Word Processor." *Creative Computing*, August 1980, pp. 38-45.

Reviews the features of the Magic Wand word processing system.

Heintz, Carl. "Analyst: Another Data Base Manager." *Interface Age*, December 1980, pp. 42-44.

Reviews the Analyst data base management system from Structured Systems.

_____. "Maxiledger." *Interface Age*, September 1980, pp. 42-44.

Reviews the Maxiledger general ledger accounting system available from Compumax Associates.

_____. "A Peach of a General Ledger Program." *Interface Age*, October 1980, pp. 46-48.

Reviews the Peachtree General Ledger system.

_____. "Pearl—A Novel Programming Gem." *Interface Age*, November 1980, p. 47.

A program which "writes" programs; Pearl writes custom application packages from user input. Explains the power inherent in such a concept.

Heyman, Victor. "IDSWORD The Comprehensive Processing System for Home and Business." *Creative Computing*, pp. 43-44.

Reviews another little-known word processing system compatible with CP/M systems.

Hogg, Douglas. "How Good Is Microsoft's FORTRAN-80." *Creative Computing*, January 1979, pp. 62-67.

Reviews Microsoft's FORTRAN-80 and examines the FORTRAN implementation on a microcomputer.

Johnson, Bob. "Business Software Review." *Interface Age*, August 1979, pp. 38-39.

Reviews the Graham Dorian Apartment Management and Payroll software packages.

Kendall, Wallace. "Prettyprinting with Microsoft BASIC." *Kilobaud Microcomputing*, May 1979, p. 80.

Presents a simple way to format program listings using Microsoft BASIC.

Knecht, Ken. "CBASIC Review." *80 Microcomputing*, April 1980, pp. 130-132.

Compares CBASIC features with Microsoft BASIC; introduces FMG's CP/M operating system for the TRS-80.

Lindsay, John. "New Version of BASIC." *Kilobaud*, May 1980, pp. 72-74.

Summarizes the features of Microsoft BASIC version 5. Discusses the differences between versions.

Lutz, Dick. "Sharpening Your Pencil." *Creative Computing*, March 1980, pp. 30-35.

Reviews a program designed to add to the features of Electric Pencil.

Magruder, Bob. "The New WPDaisy: Word-Processing Software." *onComputing*, Fall 1980, pp. 68-74.

Extensively reviews WPDaisy.

McClure, James. "CBASIC—A Review." *Creative Computing*, September 1979, pp. 48-51.

Reviews a frequently used high-level language. Provides timing comparisons

for a series of standardized benchmarks, indicating CBASIC's relatively slow speed on many tests.

_____. "Microsoft vs. MicroFocus COBOL." *Creative Computing*, March 1980, pp. 20-29.

Compares the features of the two primary implementations of COBOL available to CP/M users.

_____. "A Personal Finance System." *Kilobaud Microcomputing*, June 1979, pp. 74-78; July 1979, pp. 50-56; August 1979, pp. 66-75.

An extensive CBASIC2 program for managing personal finances.

Miller, Alan. "BASCOM: Microsoft's BASIC Compiler for the 8080/Z80." *Interface Age*, July 1980, pp. 124-126.

Discusses the difference between compilers and interpreters, emphasizing the features of Microsoft's compiler BASIC. Presents sample programs.

_____. "CBASIC: A Business-Oriented Language for CP/M." *Interface Age*, August 1979, pp. 116-119.

Summarizes and reviews the CBASIC high-level language for CP/M.

_____. "CP/M for the TRS-80 Model II: Lifeboat and FMG Corp. Versions." *Interface Age*, November 1980, pp. 94-98.

A detailed comparison of two different implementations of CP/M for the Model II.

_____. "The Electric Pencil for CP/M." *Interface Age*, August 1978, pp. 148-149.

A review of a popular word processing package.

_____. "Pascal for CP/M: Digital Marketing's Pascal/M." *Interface Age*, September 1980, pp. 96-103.

Reviews Pascal/M and conveys the difference between Pascal and other high-level languages.

North, Steve. "Creative Computing Reviews Five Software Packages—Tiny C, Microsoft BASIC 5.0, Research Machines Z80 Algol, Structural Analysis SP80 Macros, Digital Research CP/M 2.0 and MP/M." *Creative Computing*, March 1980, pp. 40-44.

Summarizes the features of the titled packages.

Pournelle, Jerry. "Omikron TRS-80 Boards, NEWDOS+, and Sundry Other Matters." *Byte*, July 1980, pp. 198-208.

A rambling look at Pournelle's experiences with microcomputers in writing; a popular science fiction writer (*Mote in God's Eye, Lucifer's Hammer*, and others), Pournelle offers a personal and fascinating account.

Press, Larry. "A Review of Four Text-Formatting Programs." *onComputing*, Fall 1980, pp. 48-54.

Reviews S-80, TEX, TPS, and Textwriter.

_____. "Word Processors: A Look at Four Popular Programs." *onComputing*, Summer 1980, pp. 38-52.

A detailed review of Auto Scribe, Electric Pencil, Magic Wand, and WordStar.

Sanger, Joseph. "The Electronic Librarian." *Kilobaud Microcomputing*, November 1979, pp. 44-62.

A BASIC-E program for maintaining an information database.

Sjowall, Tor. "CP/M to UCSD Pascal File Conversion." *Dr. Dobbs*, October 1980, pp. 16-19.

Pascal source code for converting CP/M files to the UCSD Pascal format.

VanHorn, Eric. "Super-Sort by Micro-Pro International." *Creative Computing*, July 1979, pp. 34-37.

Reviews the features of Super-Sort.

CP/M and Assembly Language Programming

Barbier, Ken. "CP/M for Single-Drive Systems." *Kilobaud Microcomputing*, September 1980, pp. 94-98.

Explains the problems with using a single-drive CP/M system; includes an assembly language single-drive file copy program source listing.

Barker, Lee. "Help with OSI's CP/M." *Dr. Dobbs*, May 1980, pp. 36-37.

A modification of the BIOS for OSI computers using CP/M to eradicate problems encountered.

Biese, Leo, and Iannuccillo, Emilio. "MASTHEAD: Why Not Title Your Printouts?" *Interface Age*, August 1980, pp. 122-127.

A method of printing large titles on program listings, with source code in both assembly language and Microsoft BASIC.

Cecil, Alex. "ACT: An 8080 Macroprocessor." *Dr Dobbs* 23:20-45.

A "TRAC"-like text interpreter, ACT is written in PL/M. Computer programming students will be interested in the source code.

Christensen, Ward. "An 8080 Disassembler." *Dr. Dobbs*, February 1977, pp. 30-43.

A complete assembly language source listing for an 8080-based disassembler. Documentation explains disassembly to assembly language programmers.

Cotton, Gene. "How to Solve Your Damaged Disk Dilemma." *Interface Age*, September 1980, pp. 80-86, 130-131.

An assembly language program to save bad spots on a diskette as a file named

[unused].bad. Contains a fairly detailed explanation of how CP/M maintains information on the diskette.

Epstein, Jake. "An Introduction to CP/M." *S-100 Microsystems*, January/February 1980, pp. 6-10; March/April 1980, pp. 28-33; May/June 1980, pp. 12-17.

An excellent introduction to CP/M for computer programmers; details specific technical information.

Foster, Charlie, and Meador, Richard. "8080 Dynatrace." *S-100 Microsystems*, July/August 1980, pp. 22-31.

A dynamic screen-oriented assembly language debugging system: source code is in 8080 assembly language. Article assumes expertise required to modify the output routines.

Frantz, James. "Turn-Key CP/M Systems." *Creative Computing*, December 1979, pp. 104-107.

Step-by-step directions to make CP/M automatically execute a program on cold starts.

Friedman, David. "Las Vegas Super Slot: A CP/M Game Machine Program Using Flashwriter I Graphics." *Dr. Dobbs*, November/December 1980, pp. 10-22.

A game program in assembly language for CP/M systems using Vector Graphic's Flashwriter I board.

Gagne, Jim. "Vice Versa—Pencil to CP/M and Reverse." *Dr. Dobbs*, March 1979, pp. 26-29.

How to reformat Electric Pencil files to CP/M format and vice versa; assembly language source code provided.

Haanstra, Bruce. "Optional Printing with CP/M and Microsoft BASIC." *Interface Age*, November 1980, pp. 84-86.

Illustrates how to circumvent Microsoft BASIC's interception of the ^P character used to turn the printer on and off in CP/M.

Hallen, Rod. "Battle of the Assemblers." *Creative Computing*, December 1979, pp. 42-45.

Compares ASM, supplied with CP/M, with ASMB, an assembler available from Technical Systems Consultants. Briefly describes an assembler called "SASSY."

Hoffer, W.C. "Data and Time for the CP/M Operating System." *Interface Age*, August 1978, pp. 152-156.

Assembly language routines enable a CompuTime board to perform automatic time and dating of program listings.

Kildall, Gary. "Simple Technique for Static Relocation." *Dr. Dobbs* 22:10-13.

The author of CP/M describes CP/M's relocation technique.

Miller, Alan. "CP/M Part 2 — A Macro Assembler and Other Goodies." *Interface Age*, December 1978, pp. 130-135.

Reviews the MAC assembler available from Digital Research; the SID symbolic instruction debugger also from Digital Research, and a sample CBIOS for a North Star CP/M system.

_____. "An Interrupt-Driven Keyboard Buffer." *Interface Age*, October 1980, pp. 106-107, 137-141.

A different approach for implementing a keypress detection routine for North Star BIOS in CP/M.

_____. "Structured Assembly-Language Programming for the 8080." *Interface Age*, November 1979, pp. 153-155.

Reviews SP80, a structured assembly language set of macro routines available for CP/M systems. Compares Digital Research's MAC assembler with SP80.

_____. "ZSID Z80 Debugger for CP/M." *Interface Age*, August 1980, pp. 88-90.

Reviews the ZSID symbolic debugger for assembly language programs available from Digital Research; also mentions the DESPOOL program from Digital Research.

Parsons, Ronald. "UCSD Pascal to CP/M File Transfer Program." *Dr. Dobbs*, August 1979, pp. 12-16.

Assembly language program transfers from UCSD Pascal format to CP/M file structure.

Pugh, Tim. "Intelligent Terminal Implementation on S100 Bus." *Dr. Dobbs* 26:4-16.

A well-commented assembly language program that allows communication between any CP/M system using an IDS-88 modem board and any other computer system.

Terry, Chris. "The CP/M Connection." *S-100 Microsystems*, July/August 1980, pp. 32-35.

Discusses CP/M's relocation method and how to interface CBIOS and make other modifications to a relocated system.

Van Buer, Darrel. "A Table-Driven Assembler on CP/M." *Dr. Dobbs*, February 1980, pp. 18-25.

A PL/M source code for a macro assembler.

Willoughby, Steve. "Hardcopy Device Driver Programs for CP/M." *Dr. Dobbs* 48:34-37.

Assembly language routines to be included in CP/M CBIOS section for Diablo 1650 or Teletype printers.

APPENDIX

G CP/M Sources

Sources of CP/M Compatible Languages, Programs, and Computers

Word Processing Programs

Magic Wand	Small Business Applications, Inc.
Microsoft Edit	Microsoft, Inc.
Electric Pencil	Michael Shrayer Software, Inc.
SCOPE	Vector Graphic, Inc.
WordStar	MicroPro International

Languages

Assemblers	Digital Research, Inc. Microsoft, Inc. Sorcim Corp.
ADA	Supersoft, Inc.
BASIC	Microsoft, Inc. Supersoft, Inc.
C	Supersoft, Inc. Whitesmiths, Ltd.
CBASIC	Digital Research, Inc.
COBOL	Microfocus, Inc. Microsoft, Inc.

FORTRAN	Microsoft, Inc. Supersoft, Inc.
FORTH	Forth, Inc. Supersoft, Inc.
LISP	Microsoft, Inc. Supersoft, Inc.
Pascal	Digital Research, Inc. Ithaca Intersystems, Inc. Sorcim Corp.
PL/I	Digital Research, Inc.

Data Management Systems

SELECTOR-IV	Micro.AP, Inc.
Pearl	Computer Pathways Unlimited
HDMS	Micro Data Base Systems

CP/M Derivative Operating Systems

CDOS	Cromemco, Inc.
SDOS	SD Systems
TPM	Computer Design Labs
TSA/OS	TSA Software

APPENDIX

H Manufacturers' Index

ADDS
Applied Digital Data Systems
100 Marcus Boulevard
Hauppauge, NY 11787
(516) 231-5400

Altos Computer Systems
2360 Bering Drive
San Jose, CA 95131
(408) 946-6700

ANSI
American National Standards Institute
1430 Broadway
New York, NY 10018
(212) 354-3300

Apple Computer
10260 Bandley Drive
Cupertino, CA 95014
(800) 538-9696 (except California)
(800) 662-9238 (from California)

BDS
BD Software
See Lifeboat Associates

Bell Laboratories—C Language
Patent License Organization
Western Electric Company
P.O. Box 25000
Greensboro, NC 27420
(919) 697-6530

Byte Publications, Inc.
70 Main Street
Peterborough, NH 03458
(603) 924-9281

Compiler Systems, Inc.
P.O. Box 145
Sierra Madre, CA 91024
(213) 355-4211

Computer Design Labs
342 Columbus Avenue
Trenton, NJ 08629
(609) 599-2146

Computer Pathways Unlimited
2151 Davcor Street S.E.
Salem, OR 97302
(503) 363-8929

CP/M Users Group
1651 Third Avenue
New York, NY 10028

Cromemco, Inc.
280 Bernard Avenue
Mountain View, CA 94043
(415) 964-7400

Digital Microsystems
1840 Embarcadero
Oakland, CA 94606
(415) 582-3686

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

DynaByte, Inc.
115 Independence Drive
Menlo Park, CA 94025
(415) 329-8021

Exidy
1234 Elko Drive
Sunnyvale, CA 94086
(408) 734-9410

FORTH, Inc.
2309 Pacific Coast Highway
Hermosa Beach, CA 90254
(213) 372-8493

Heath Data Systems
Hilltop Road
St. Joseph, MI 49085
(616) 982-3200

Industrial Micro Systems
628 N. Eckhoff Street
Orange, CA 92668
(714) 978-6966

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051
(408) 987-8080

Ithaca Intersystems, Inc.
1650 Hanshaw Road
P.O. Box 91
Ithaca, NY 14850
(607) 257-0190

Lifeboat Associates
1651 Third Avenue
New York, NY 10028
(212) 860-0300

Lifelines Publishing Corp.
1651 Third Avenue
New York, NY 10028
(212) 722-1700

Michael Shrayer Software, Inc.
1198 Los Robles Drive
Palm Springs, CA 92262
(714) 323-1400

Micro Data Base Systems
P.O. Box 248
Lafayette, IN 47902
(317) 448-1616

MICRO.AP, Inc.
9807 Davona Drive
San Ramon, CA 94583
(415) 828-6697

Micro Focus Inc.
1620 Civic Center Drive
Santa Clara, CA 95050
(408) 984-6961

Micromation, Inc.
1620 Montgomery Street
San Francisco, CA 94111
(415) 398-0289

MicroPro International Corp.
1299 4th Street
San Rafael, CA 94901
(415) 457-8990

Microsoft, Inc.
10800 N.E. Eighth, Suite 819
Bellevue, WA 98004
(206) 455-8080

MITS
See Pertec Computer Corp.

MMS
Miller Microcomputer Services
61 Lake Shore Road
Natick, MA 01760
(617) 653-6136

Morrow Designs
5221 Central Avenue
Richmond, CA 94804
(415) 524-2101

Ohio Scientific
1333 South Chillicothe Road
Aurora, OH 44202
(216) 831-5600
(800) 321-6850

Onyx Systems, Inc.
10375 Bandley Drive
Cupertino, CA 95014
(408) 257-8022

Pertec Computer Corp.
20630 Nordhoff Street
Chatsworth, CA 91311
(213) 998-1800

Radio Shack
1300 One Tandy Center
Fort Worth, TX 76102
(817) 390-3700

SD Systems
P.O. Box 28810
Dallas, TX 75228
(214) 271-4667

Small Business Applications, Inc.
3220 Louisiana, Suite 205
Houston, TX 77006
(713) 528-5158

Sorcim Corp.
405 Aldo Avenue
Santa Clara, CA 95050
(408) 727-7634

Supersoft, Inc.
P.O. Box 1628
Champaign, IL 61820
(217) 359-2112

Tarbell Electronics
950 Dovlen Place, Suite B
Carson, CA 90746
(213) 538-4251

Texas Instruments, Inc.
P.O. Box 1444
Houston, TX 77001
(800) 257-7850
(except NJ)
(800) 322-8650
(from NJ)

Thinkertoys
See Morrow Designs

TSA Software
5 North Salem Road
Ridgefield, CT 06877
(203) 438-3954

Vector Graphic, Inc.
31364 Via Colinas
Westlake Village, CA 91362
(213) 991-2302

Whitesmiths, Ltd.
P.O. Box 1132
Ansonia Station, NY 10023
(212) 799-1200

Zenith
See Heath Data Systems

Glossary

ASCII. American Standard Code for Information Interchange refers to the computer's internal recognition of the letters, numbers, and symbols of the English language. Appendix B provides a complete listing of these representations.

Assembly Language. Assembly language consists of a series of names (called mnemonics) that represent the various combinations of 1's and 0's which instruct the computer. Instead of typing "11001001" (machine instruction), the mnemonic "RET" instructs the computer to RETURN to a location stored by the computer.

BASIC. Beginners All-purpose Symbolic Instruction Code is a "high-level" language. It is more than a symbolic representation of the instructions a computer can interpret. For example, the BASIC statement "PRINT" contains many computer instructions.

Batch Processing. Before video terminals and other sophisticated devices were invented, most computers received instructions from punched cards. The holes in the cards represented computer instructions and data. The cards were collected in groups (batches) for effective processing. Today, the term batch processing refers to any process which collects instructions and data (by terminals, keypunches, optical card readers, and so on) and delays processing until a large "batch" accumulates.

Booting/Bootstrap. Booting initializes a computer system by copying an operating system into the internal memory of a computer from a diskette or tape.

Buffer. A buffer is an area within the computer's memory which temporarily stores input from or output to any peripheral computer device (like a disk drive). The central processing unit can more quickly process information to the computer's memory than a slower mechanical device, printer, disk drive, punch, or keyboard.

Byte. The storage capacity of a microcomputer's memory, or its disks, is always described as some number of bytes. A byte is a memory unit capable of storing a single character. For example, the letter "A" or the digit "1" could be stored in one byte of memory. Numbers without decimal points are usually stored in two bytes of memory, while numbers with decimal points may require five or more bytes of memory per number. Memory size is usually expressed not as thousands of bytes, but rather as some number of K bytes. 1K equals 1024. All computers are binary machines that count in twos. You will get the number 1024 if you multiply 2 to the 10th power.

Call. A call instructs the computer to transfer control to another section of memory where the required routine is located. Calls are used often in computing since many processes divide into subroutines. Instead of duplicating the same routine in several memory locations, the routine is stored at one location and called as necessary. When the call is completed, it returns control to the instruction immediately following the one which called it.

Close. To close a file means to finish (at least temporarily) any processing associated with it. Closing makes information storage and retrieval on files quicker and more efficient. CP/M does not completely update a file until it is closed.

COBOL. Common Business Oriented Language instructs the computer in English-like sentences. COBOL programs are easy to read, but require a lot of space. COBOL is a standardized language; it is the same no matter what computer uses it. COBOL is used primarily by large businesses and the government.

Compiler. Compiler applies to a high-level language which does not translate the instructions as you type. If you had a BASIC compiler, for instance, you would enter a program using an editor and then compile the program. Compiling consists of the following steps: first the computer checks your program for blatant errors, then it translates your high-level language instructions into machine instructions. (*See also* Interpreter.)

Configure. To configure a program means to change some program parameters to reflect the equipment used. One program can be used on several combinations of terminals, printers, and other devices. In CP/M, the term configure is usually associated with the BIOS (Basic Input Output System). In order to run CP/M, BIOS must be configured to your equipment. This operation is best left to a computer professional; the process is summarized in Chapter 7.

Console Device. The console device sends and receives input to and output from the computer. For most CP/M users the console device is a standard

terminal with both a keyboard and CRT display. On some systems the keyboard and display may be separate units. Some older systems use a typewriter-like computer printer (one which has a keyboard for input) as the primary console device, but this practice has diminished with the advent of low-cost video displays.

Central Processing Unit. A CPU is the heart of a computer; all information passes through it.

Data. Data refers to any information the computer processes or stores. An important distinction exists between an instruction to the computer and data the computer processes. A computer distinguishes between data and instructions entirely by context. One missing piece of information can cause strange results.

Directory. A directory provides a table of contents of the disk. The directory stores the name of each file and some other information CP/M uses to trace the file's physical status located on the disk and its current use.

DMA. Direct Memory Access bypasses the CPU and information goes directly to and from memory. While the disk drives perform a DMA transfer, the CPU simply waits for the process to be completed. Normally, information must pass through the CPU when traveling from diskette to computer memory or vice versa. Despite bypassing the CPU, DMA transfers are generally faster than if the CPU were involved.

Documentation. Documentation includes all written information about a piece of computer equipment or software. Your computer system manuals are documentation.

Drivers. A driver is a complete set of instructions which controls a piece of equipment. In other words, to run your disk system, you must have drivers which specifically apply to your equipment. CP/M's drivers are in the BIOS (Basic Input Output System).

Execute. To execute a program means that the computer performs all the instructions which comprise that program. When the CPU interprets a piece of information as an instruction, it executes it.

File. File refers to related blocks of data or instructions stored on a disk. One complete program might be stored in a file named "PROGRAM.BAS," for instance. One complete set of related data (check register, name and address file, and so on) may be saved in a datafile.

File Names/File Types. The file name in CP/M has three distinct sections: a name of up to eight characters, followed by a period, and a unique type identifier called a file type. CP/M locates the file by the file name and file type.

Formatting. Formatting puts dummy (nonsense) information onto a diskette prior to use. It also verifies whether the diskette can be read.

Hard Disk. Hard disk refers to media mounted in polished metal surfaces or

platters. The platters are coated with magnetic oxide. In a fixed disk the platters are permanently fixed (and sealed) within the drive; you cannot remove the medium without ruining the drive. In hard disk drives which are not fixed, the media is encased in removable plastic cartridges.

Hex. Hex, short for hexadecimal, refers to a numbering system which counts "1 2 3 4 5 6 7 8 9 A B C D E F," or in math base 16. The smallest computer unit of meaning (a byte—eight 1's or 0's) consists of two hex digits. Thus "11110000" becomes "F0 hex." The table in Appendix B illustrates the equivalencies between binary, hexadecimal, and decimal digits.

Interrupts. An interrupt is a signal to the CPU to stop processing one instruction and to execute a different set of instructions.

Library File. A library file contains a set of standard routines you can add to a new program. You could write a complete program from various library files.

Machine Code/Machine Language/Machine Instructions. Machine code, machine instructions, and machine language all refer to the lowest level of instructions directly understood by the computer.

Machine Dependent. If a program or device is machine dependent, it only operates with a particular set of equipment.

Macro. Macro refers to library files in assembly language; thus there are macro assemblers.

Media. Media is the material on which information is stored. Floppy diskettes and cassette tapes are media.

Memory Location. Computer memory locations are defined by addresses. One specific memory location can store one computer word (byte) of information.

Minifloppies. Minifloppy, a trademark of Shugart Associates, refers to their 5¼-inch floppy diskettes. Other manufacturers also produce 5¼-inch floppy diskettes.

Modem. Modulator/Demodulator, converts data and/or computer instructions into tones (frequencies) and decodes them. Telephone lines carry the modulated signals. Compared to computers, modems are generally very slow devices.

Operating System. The operating system is the brains of the computer. The operating system is configured to "know" which peripheral devices are available and how to communicate with them. The operating system controls all of the equipment but usually (as in the case of CP/M) does not perform complicated procedures.

Punch. A punch device punches holes in a strip of paper or paper tape. Before the advent of floppy disks, data or programs were stored by creating punch cards or paper tape representations.

RAM. Random Access Memory is the primary storage area of a computer. The information stored in RAM is lost when power is disconnected. This is one reason why CP/M must be loaded each time you turn on your computer system.

Random. Disk systems use two storage methods: random and sequential access. Random access implies you may read pieces of information at random. Random access files are accessed by a record number (location within a group of records). In order to find a random access record quickly and accurately, random files consist of same length records.

Read. Read means getting a piece of information from a file. Reading a file means getting the information in that file and transferring it to memory (or another device).

Reader. A paper tape reader reads the holes punched by the paper punch device and interprets them according to a predetermined code.

Register. A register is an internal temporary storage area within a CPU. The 8080 family of CPUs (on which CP/M runs) has a minimum of eight registers. 8080 CPU registers are the A and Flag, B and C, D and E, and H and L registers. The pairing of these registers reflects the 8080 computer instructions which reference a pair of registers. Only assembly and machine language programmers are interested in register contents.

Reset. Reset instructs the computer to restart. It turns the computer off, then on again.

ROM. Read Only Memory can only be read from; you cannot change the information stored in it. Non-changing routines like the system bootstrap or a system monitor (which controls the computer at a higher level than CP/M) are often put into ROM.

Routines. Routines are a set of modules (related instructions) which perform specific tasks. Routines comprise programs but are not programs in themselves. Examples of routines might include a routine to input one character, to see if a key was depressed, to display one character, and to turn on the disk drive motor.

Run-Time. Sometimes modules are referred to as run-time routines, or run-time code. Another name for this is executable code.

Sectors and Tracks. When information is stored on a cassette tape, it is stored as a single track of data down the length of the tape. When information is stored on a diskette, the surface of the diskette is divided into a number of concentric tracks. Each track is divided into a number of sectors, so that CP/M can access any point on the diskette surface, given a sector and track number. Figure 1-3 illustrates the sectors and tracks.

Sequential. Elements of a sequential access file are accessed in order. If you want to read element number ten, you must first read the preceding nine elements. Sequential storage methods are used when the amount of data to be stored cannot be anticipated. Sequential files have an end-of-record marker, and

you may have elements of varying length (called variable length records).

Skewing. Skewing relates to how information is physically stored on a diskette. Information is assigned to non-adjacent locations (the first piece of information is put in sector one, the second in sector 18, the third in sector 25, and so on). This weaving of information is handled completely by CP/M.

Source. Source code refers to the original program in its original state. Since programs are sometimes compiled (translated into machine language, or a more efficient language), the distinction between source and run-time is important. Additionally, the term source describes any original data or program, and backup refers to a copy of the source.

Statement. A statement is one complete instruction (or combination of instructions) to a computer. The complexity of a statement (think of it as a computer sentence) depends entirely on the language used and the programmer's style.

Status. Computer devices are either ready or not ready. When the computer wants to use a device (say a printer), it checks its status; it checks to see if the device is ready. Unfortunately, status schemata are not standardized. A positive voltage emitted from a device can either mean that it is ready or not ready, depending on the manufacturer's design. CP/M performs status checks automatically, assuming a proper BIOS has been created.

Tape. Computer tape is like audio tape. It usually consists of ½-inch or 1-inch tape wound on reels. Computers store a higher density of information onto a tape than do home tape recorders. The information is stored differently; digital pulses versus analog representations. Tape is an easy to handle and inexpensive medium to store data or programs. Unfortunately, tape is not suited for random access. The longer the tape, the longer the wait for record location. Tape remains a mainstay of minicomputer and large computer installations and is used as an inexpensive storage medium for home computers. However, it is not often used with CP/M systems.

Utilities. Utility programs are usually used for more than one purpose by many kinds of users. This is distinct from an application program, which performs only one job.

Wildcard. Global parameters remain constant throughout a process and accept any specification. For example, in the CP/M command, "STAT B:*.ASM", "*" is a global parameter since anything can fill that position in the command line.

Write. Write puts information onto a diskette or other media, including RAM.

Write-Protect. You cannot add information to a write-protected disk; it can only be read. An 8-inch diskette's write-protect notch appears on the bottom of the diskette; a minifloppy diskette is write-protected by covering the notch on the right side of the diskette. The STAT command also write-protects diskettes and files.

8080. The 8080 microprocessor was designed by Intel. The 8080 is a second generation CPU chip and the first to have widespread appeal. The 8080 has eight internal registers: A, B and C, D and E, H and L, and Flag. Its instruction set (the instructions that it can directly interpret) is somewhat limited compared to later CPU designs.

Z80. The Z80 chip is a redesign of the 8080; while it retains full compatibility (it can execute all of the 8080 instructions exactly as the 8080 does), it introduces a number of other features. In addition to the 8080 register set, the Z80 contains a duplicate set, referred to as the A', B' and C', D' and E', H' and L', and Flag'. Also, indexing, a special type of computer instruction, is allowed.

Index

V

W

X

Other Osborne/McGraw-Hill Publications

An Introduction to Microcomputers: Volume 0—The Beginner's Book, 3rd Edition
An Introduction to Microcomputers: Volume 1—Basic Concepts, 2nd Edition
An Introduction to Microcomputers: Volume 3—Some Real Support Devices
Osborne 4 & 8-Bit Microprocessor Handbook
Osborne 16-Bit Microprocessor Handbook
8089 I/O Processor Handbook
CRT Controller Handbook
68000 Microprocessor Handbook
8080A/8085 Assembly Language Programming
6800 Assembly Language Programming
Z80 Assembly Language Programming
6502 Assembly Language Programming
Z8000 Assembly Language Programming
6809 Assembly Language Programming
Running Wild—The Next Industrial Revolution
The 8086 Book
PET®/CBM™ and the IEEE 488 Bus (GPIB)
PET® Personal Computer Guide
CBM™ Professional Computer Guide
Business System Buyer's Guide
Osborne CP/M® User Guide, 2nd Edition
Apple II® User's Guide
Microprocessors for Measurement and Control
Some Common BASIC Programs
Some Common BASIC Programs—PET™/CBM™ Edition
Some Common BASIC Programs—Atari® Edition
Some Common BASIC Programs—TRS-80™ Level II® Edition
Some Common BASIC Programs—Apple II Edition
Some Common BASIC Programs—IBM® Personal Computer Edition
Some Common Pascal Programs
Practical BASIC Programs
Practical BASIC Programs—TRS-80™ Level II Edition
Practical BASIC Programs—Apple II® Edition
Practical BASIC Programs—IBM® Personal Computer Edition
Practical Pascal Programs
Payroll with Cost Accounting
Accounts Payable and Accounts Receivable
General Ledger
CBASIC™ User Guide
Science and Engineering Programs—Apple II® Edition
Interfacing to S-100/IEEE 696 Microcomputers
A User Guide to the UNIX™ System
PET™ Fun and Games
Trade Secrets: How to Protect Your Ideas and Assets
Assembly Language Programming for the Apple II®
VisiCalc®: Home and Office Companion
Discover FORTH
6502 Assembly Language Subroutines
Your ATARI® Computer
The HP-IL System

"...one of the very best CP/M® manuals available."
Creative Computing, May 1982

OSBORNE CP/M® USER GUIDE

Second Edition

In this revised edition, the latest CP/M® developments are discussed, including CP/M™-86, the operating system for 8086- and 8088-based microcomputers such as the IBM® Personal Computer. For computer users who want to know the basics of CP/M,® this guide bridges the gap between technical manuals and your working knowledge of microcomputers.

For beginners, the OSBORNE CP/M® USER GUIDE
- Covers all the CP/M® commands in detail
- Describes standard CP/M® utility programs
- Discusses high-level languages and utility programs that run on CP/M®
- Provides reference lists and tables
- Examines the systems approach to implementing CP/M.®

For more advanced users and programmers, the OSBORNE CP/M® USER GUIDE
- Details CP/M™-80 and CP/M™-86 system calls
- Discusses CP/M® modification for different computer systems.

This edition includes a discussion of CP/M® derivatives, MP/M II™ and CP/NET.®

ISBN 0-931988-82-9