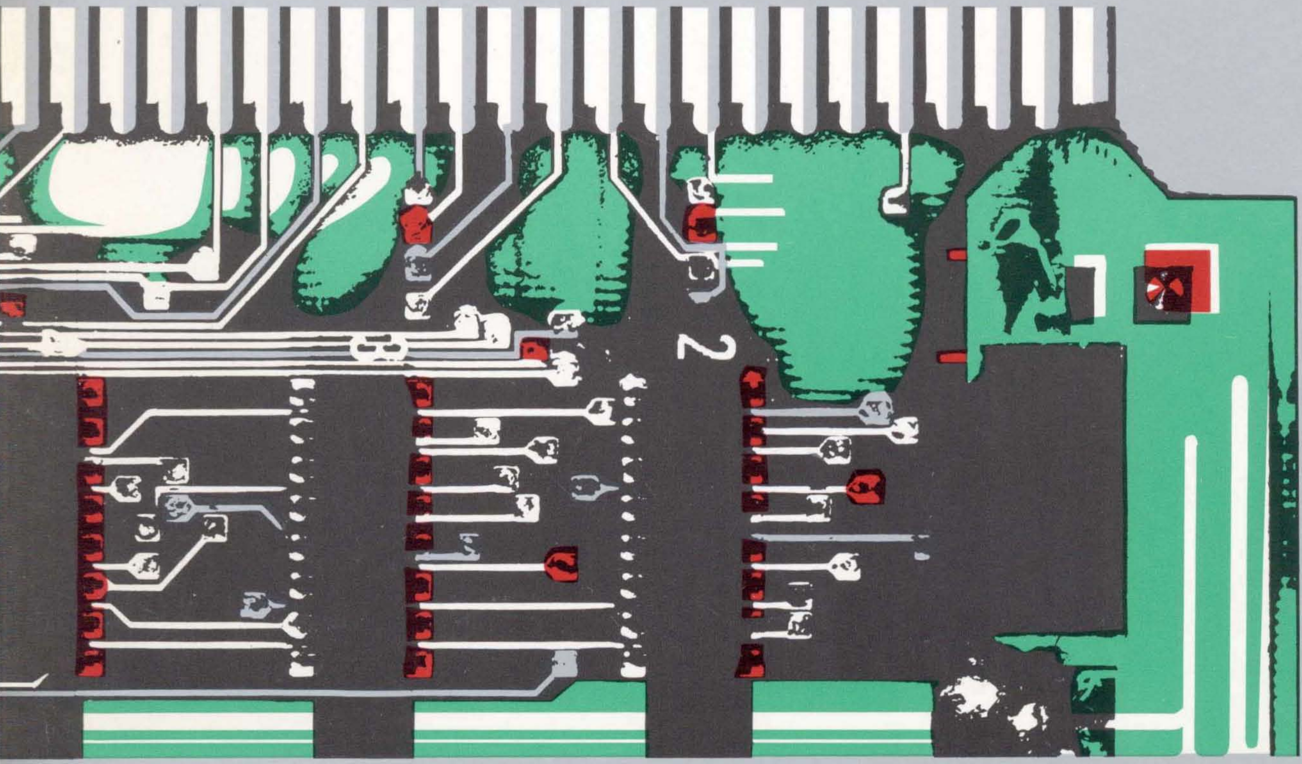
The Apple CP/M Book

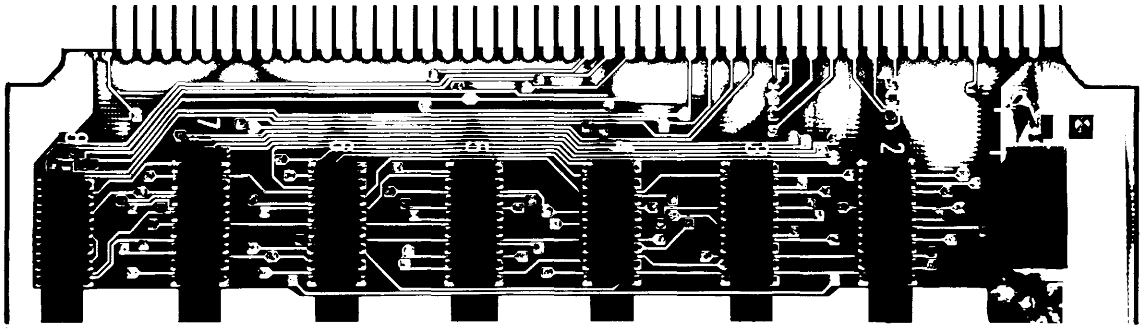
Murray Arnow



The Apple CP/M Book

The Apple CP/M Book

Murray Arnow, Ph.D.



Scott, Foresman and Company
Glenview, Illinois London

ISBN 0-673-18068-9

Library of Congress Cataloging in Publication Data

Arnow, Murray.
The Apple CP/M book.

Includes index.
1. CP/M (Computer operating system) 2. Apple II (Computer)—Programming. 3. Apple II Plus (Computer)—Programming. 4. Apple IIe (Computer)—Programming.
I. Title.
QA76.76.O63A76 1985 001.64'2 85-2393
ISBN 0-673-18068-9

1 2 3 4 5 6 7—RRC—90 89 88 87 86 85

Preface

The intent of this book is to introduce the Apple II, Apple II Plus, or Apple IIe computer user to the world of CP/M. The book is meant primarily to be a tutorial on CP/M and in this sense does not deviate greatly from the large library of books on that subject. This book differs in detail from the others in that the examples used and explanations given are aimed toward Apple users. It is assumed that the Apple user has a working knowledge of Apple DOS; when possible, comparisons are made to Apple DOS. There are many examples and detailed descriptions to aid those with less experience. Some of the explanations require a rudimentary knowledge of binary numbers. Appendix A gives an elementary review of binary numbers.

There is a distinction between *disk* and *diskette* used throughout this book. An 8-inch floppy disk is called a *disk*. A 5¼-inch floppy disk is called a *diskette*. This complies with standard usage. This distinction may seem artificial to an Apple user, but it is necessary. CP/M was first developed for 8-inch disk drives. The understanding of how CP/M works is directly related to this fact.

The needs of advanced users are not ignored. A description of the Microsoft BIOS is included, with suggestions on modifications. It is assumed, however, that the advanced user has some programming skills at the assembly-language level and that he or she is familiar with the physical layout of the Apple.

I have tried to make each chapter self-contained so that the reader need not search through the book whenever a new term or concept is introduced. The problem with such an approach is that it makes some repetition inevitable. My teaching experience has shown me, however, that, although repetition can be boring to some, it is often welcomed by others when unfamiliar material is introduced.

Contents

1 Introduction

What Is CP/M?

CP/M is a copyrighted name belonging to Digital Research, Incorporated; it stands for Control Program for Microprocessors. CP/M is an *operating system* (OS). The term *operating system* is computer jargon for the collection of programs used by a computer to communicate with the world. This world as seen by a computer is its hardware, such as the keyboard, the display device, the printer, the disk drives, and any other devices that may be plugged in. A good operating system should permit the computer user to easily access the hardware.

The disk drive is a special piece of hardware and in general requires a great deal of attention. The disk drive is used to read information from or write information to a magnetic disk placed in the drive (refer to Appendix C for a detailed description of disks and disk drives.) The disk may be thought of as a chalkboard where information can be written and saved or erased, as required. The problem with a disk is that, although information can be saved on it at arbitrary physical locations, these arbitrary locations cannot be random. This means that there must be a plan to putting information on a disk. Without a plan, the computer could never retrieve anything saved on a disk. Information is usually stored on a disk in physical segments called *files*. The file locations are stored in a special disk area called the *directory*. Whenever new information is written to a disk, the OS must create a file and put the address of its location in the directory. If the information is to be retrieved, then the OS uses the directory data to find the file and read its contents back to the computer. An OS must also be able to delete files, add information to a previously created file, and read a specified segment of information from a file.

CP/M is an operating system that has become the de facto standard for computers using the Intel 8080 or 8080-compatible microprocessors. It is virtually impossible to find a computer using the Intel 8080, Intel 8085, or Zilog Z-80 microprocessor that doesn't feature CP/M as its OS. Consequently, there is currently a tremendous inventory of programs, both commercial and in the public domain, available to the user. The remarkable thing about these programs is that they can run, with a few noted exceptions, on any computer operating under CP/M. The reason for this is the machine-independent structure of CP/M; details of the CP/M's internal workings are covered later in this book.

■ Why Use CP/M?

As an Apple owner you are probably wondering what justification there is for installing a CP/M operating system in your computer. There are three reasons serious Apple users such as business people and programmers should consider.

The first reason is that you can tap into a source of program material not available under Apple DOS. For instance, there are compilers available under CP/M for FORTRAN, COBOL, Pascal, FORTH, BASIC, C, ADA, and so on; there are some extremely powerful business packages, such as data-base programs, whose quality is frequently better than that of similar programs found in Apple DOS.

The second reason is related to the lack of standards for Apple DOS. Many vendors, in an attempt to protect their programs written for the Apple, have created their own nonstandard operating systems that are incompatible with Apple DOS. This means that making backup copies of a disk is often difficult, time-consuming, and expensive. The disk files made by these proprietary operating systems are frequently unusable by programs written by different vendors. Conversely, there is no copy protection under CP/M. Technically speaking, a copy-protected disk cannot have a CP/M format. However, there is another reason for the lack of copy protection: the program vendors are sensitive to the fact that many of their customers are businesses that need the ability to back up their disks. After a recent attempt by the vendor of a successful commercial program to copy-protect its disk, there was such a strong public reaction that the vendor relented and distributed its wares on standard CP/M-format disks.

The final reason for acquiring CP/M is that under CP/M, files created by programs written by one vendor can usually be read and modified by programs written by a second vendor. For instance, it is possible to create a document on any of the currently available word-processing programs and have the spelling checked by any of a number of spelling-checker programs. This compatibility is due in part to the CP/M's structure and in part to the fact that vendors realize that the more compatible their programs are, the more easily they are sold.

Let us assume that I have persuaded you to give CP/M a try. You can install CP/M in your Apple by getting one of the currently available CP/M packages. These packages will contain at the very least a Z-80 card (the

Microsoft Z-80 cards are called the SoftCard and the Premium SoftCard IIe). This is required because CP/M is 8080 (or Z-80) microprocessor-based, and the Apple's native microprocessor is the 6502. The Z-80 card will plug into one of the slots on the Apple's motherboard. This card is so designed as to take over the 6502's control of the Apple when a CP/M diskette is booted. The Z-80 may be in control of the Apple, but it still must call on the 6502 to perform certain hardware functions, so in effect the Apple CP/M has two microprocessors working in concert.

The package will also contain the CP/M master diskette, which when booted loads the operating system into the Apple's memory. The diskette will have a collection of programs supplied by Digital Research and the Z-80 card's manufacturer. There will be at least one manual included that will contain the Digital Research description of CP/M and the accompanying *software* (*software* is jargon for computer programs). The card's manufacturer will usually include instructions on installing the card and using the software.

You should be able to run CP/M on an Apple with one disk drive, a Z-80 card, and at least 32K of memory. This is the minimum hardware configuration. I strongly recommend that if you do not possess a second disk drive, you get one. CP/M can create such enormous files that sometimes it is impossible to fit both a program and its data file on one disk. Also, CP/M is not extremely tolerant of disk swapping. Disk swapping is almost inevitable with a one-drive system. Finally, the cardinal rule for all computer users is to make backups; this procedure is greatly simplified with two disk drives.

Many CP/M software packages are designed to run on terminals that have screen widths greater than 40 columns. You are therefore well advised to also install an 80-column video card in your Apple. The 80-column card will give the added convenience of displaying lowercase letters (assuming that your Apple is a version prior to the Apple IIe, which includes lowercase capability).

If you own an Apple IIe, you may wish to install the Microsoft Premium SoftCard IIe. This card goes into the auxiliary slot and provides the capabilities of an extended-memory 80-column card and a Z-80B microprocessor, which operates nearly three times faster than the Z-80A microprocessor used on the cards installed in the other Apple slots. The disadvantages of this card are that it is more expensive and that making modifications to its operating system is extremely difficult, even for the advanced user. This means that highly specialized hardware may have difficulties with the Premium SoftCard IIe.

2 Getting Started

Installing the CP/M Card

The very first thing you should do after acquiring a CP/M package is to read the installation instructions. The instructions will have one thing in common. They will all say to turn off the power to your Apple before inserting or removing any cards in any of the slots on the Apple motherboard. If you do not heed this warning, than you are almost certain to damage the cards, the Apple, or both.

After the CP/M card is installed, it should be *booted. Booting* is jargon for starting up, in this case turning on the power with the CP/M master diskette in the Apple's drive 1. This seemingly simple operation sometimes causes problems for the new user because of the conflicting instructions given in the Apple manuals.

The recommended procedure is to power up your Apple with the door to drive 1 left open. After the noise of the drive recalibration has stopped, place the diskette into drive 1, and gently close the door. This may contradict Apple's instruction not to touch the drive when the in-use light is on, but it is safe. Diskette damage will occur only if the drive is in the write mode. When the Apple is powered up, the drive is in the read mode, and closing the drive door at this time will cause no damage. I recommend this procedure because if there is a malfunction, then minimal diskette damage will result. The drive head will be on track 0 of the diskette, and that will probably be the only damaged track (see Appendix C). Track 0 contains the CP/M operating system only. In this way the data tracks are left undamaged and may be recovered. The description of the Apple diskette tracks can be found in your Apple DOS manual. The description of the diskette's physical layout applies to both Apple DOS and CP/M. Another suggestion is to open the drive door before turning the Apple off. This again will minimize the chances of your damaging a diskette if there is a malfunction.

The power-up boot in CP/M is called the *cold boot.* The first three tracks of the master diskette are put into the Apple's memory. These three tracks are called the *system tracks* because the CP/M operating system is contained there. When the cold boot is completed, a message will be displayed showing the CP/M version number and copyright notices. You will also notice that if you have an 80-column card placed in slot 3 or the auxiliary slot of the Apple IIe, then the display mode is in the 80-column format. You will also get an 80-column display if a Microsoft Premium SoftCard IIe was installed in the Apple IIe auxiliary slot. Below is the cold-boot screen display for one of Microsoft's versions of CP/M.

```
     Softcard CP/M
     44K Ver. 2.23
(c) 1980,1982  Microsoft
```

The A> Prompt

The symbol A> is called a *prompt* and will appear after the cold booting of CP/M in all computers. The meaning of A> is that the *active drive* is A:. The *active drive* is the drive that CP/M will access when a file is requested. The drive names require some explanation. CP/M uses letters to designate drives; Apple DOS uses slot and drive numbers to designate drives. Currently, all Apple CP/M versions designate drive A: to be drive 1 in slot 6, drive B: to be drive 2 in slot 6, drive C: to be drive 1 in slot 5, and so on. The colon (:) after the drive designation is standard CP/M nomenclature to distinguish a device from a file. This will be clarified later, but for the time being assume that whenever you refer to a drive you must always follow the drive name with a colon.

The Directory

The first thing you should do after completing the cold boot is to look at the directory by entering without any spaces

```
DIR<CR>
```

The <CR> stands for *carriage return*, which you get on the Apple by pressing the RETURN key. The diskette directory will then be printed to the screen. The directory is analogous to the Apple DOS CATALOG and is a listing of all the files stored on the diskette. Below is the directory listing of a Microsoft master diskette.

Microsoft Master Diskette Directory

```
A: CAT      COM : CONFIGIO BAS : DDT      COM : BOOT     COM
A: MFT      COM : PATCH    COM : CPM60    COM : PIP      COM
A: STAT     COM : ASM      COM : AUTORUN  COM : LOAD     COM
A: COPY     COM : ADPOS    COM : SUBMIT   COM : XSUB     COM
A: DUMP     ASM : DUMP     COM : DOWNLOAD COM : MBASIC   COM
A: GBASIC   COM : ED       COM
```

■ Making Copies

The second thing you should do is to make a copy of the CP/M master diskette. The master diskette should contain a program for formatting and copying diskettes. The Microsoft copy program is suitably named COPY.COM. Microsoft CP/M version 2.20B requires the running of FORMAT to format the diskettes. To run FORMAT simply type

```
FORMAT<CR>
```

and respond to the screen prompts. Incidentally, FORMAT notices if the diskette has been previously formatted. I suggest that you reformat the diskette you will copy the master diskette to. After the diskette has been formatted, the master diskette can be copied. To invoke COPY, enter

```
COPY<CR>
```

The COPY program will be loaded from the diskette and run. The program will then prompt you in the making of the copy and return you to the console prompt A>. A formatting progam doesn't come with Microsoft's CP/M versions 2.23 and higher. The copy program for these CP/M versions will format diskettes automatically. At this point you may notice the differences between CP/M and Apple DOS. To run a program in CP/M you need enter only the progam's name. To run a program in Apple DOS you must enter RUN or BRUN followed by the program's name.

Place your original CP/M master in a safe place, and make a second copy of the CP/M master using the duplicate master. Always keep at least two copies of the master diskette, and never use the original unless absolutely necessary. Making backups is the most important duty in maintaining a computer. If you are negligent in this task, you may someday find yourself spilling coffee on your only copy of a diskette containing irreplaceable files.

■ A Quick Look at the System

Now that you have made backup copies of the CP/M master diskette, you can safely expore the system. The first thing you probably noticed was that it was when the A> prompt appeared that commands such as DIR were entered into the computer. This is because CP/M loaded a special program into the computer called the Console Command Processor (CCP). Assuming that you already know Apple DOS, you are familiar with how to enter commands such as CATALOG from the keyboard. The keyboard input routine is built into Apple DOS. What we are getting at is that the operating system must supply a means of entering commands from the keyboard, also called the *console*. In CP/M the CCP is the part of the operating system that handles the keyboard commands.

Before proceeding, be sure to write-protect the CP/M master backups by placing tabs over the diskette notches. Put one CP/M master diskette in drive A: and the other in drive B: (drives 1 and 2, respectively), and enter CONTROL-C (hold down the CONTROL key while you press the C key). This is called a *warm boot*, and it tells the operating system to forget about any previous operations and start up fresh. In this particular instance, drive A: will become active; then the A> prompt will appear on the next line, followed by the screen cursor.

A word of caution at this point: Apple DOS allows you to be almost careless about the placement of spaces in a command line; CP/M does not permit this luxury. Commands by the user should start immediately following the prompt; for instance, entering

```
A>DIR<CR>
```

will print the directory of the diskette in drive A:. To print the directory of the diskette in drive B:, enter

```
A>DIR B:<CR>
```

Notice the space between DIR and B:.

The *active drive* in the previous examples has been A:. Any commands that involve disk access automatically assume that the active drive is the drive to be accessed. For instance, the command

```
DIR<CR>
```

printed the directory to drive A: without our explicitly specifying drive A:. The active drive is also called the *default drive*. The word *default* is frequently used in computer jargon to refer to the data the computer assumes if no data is given by the user. Because *default* is so commonly used, we will use it in preference to the word *active*.

You change the default drive to B: by entering B: after the A> prompt, as shown below.

```
A>B:<CR>
```

produces

```
B>
```

on the next line. You will notice that the prompt has changed and now indicates that the default drive is B:. Before changing the default drive, be sure that you have placed a CP/M-formatted diskette in drive B:.

Otherwise, you will get an error message, and the situation can get a little messy. Entering

```
DIR<CR>
```

now will print the drive B: directory. The drive A: directory is printed by entering

```
DIR A:<CR>
```

Changing the default drive back to A: is done analogously by entering

```
A:<CR>
```

This completes the preliminary discussion on starting up CP/M. The remaining chapters will enable you to better understand the role of the software in the operation of CP/M.

3 The Structure of CP/M

The Memory Organization

Knowing how CP/M is laid out in the computer's memory will help you understand how CP/M works. The following description assumes that you are familiar with hexadecimal notation. If you are not familiar with hexadecimal numbers, you may wish to refer to Appendix A.

The CP/M memory is arranged in the following way. The memory locations from 0000H to 00FFH are reserved by CP/M for internal use. The suffix H used here means that the number is a hexadecimal representation. The memory locations from 0100H to the beginning of the BDOS (Basic Disk Operating System) are called the TPA (Transient Program Area). The TPA is where all programs are loaded. This implies that all standard CP/M programs start at 0100H. You may now notice a significant difference between Apple DOS and CP/M. Apple DOS places almost no restrictions on where programs can be loaded or run.

Above the TPA is the BDOS. The BDOS is supplied by Digital Research and contains the routines used by programs in the TPA to read from and write to disk files, operate the keyboard and display device (called the *console*), and I/O (input or output) to other physical devices such as printers and modems. The BDOS starts at CC00H for the 56K version of CP/M, at DC00H for the 60K version of CP/M, and at EC00H for the 64K version of CP/M. The BDOS locations depend only on the memory configuration of the computer. This means that, for example, all computers with the BDOS placed at CC00H will be running the 56K version of CP/M.

Immediately above the TPA is the BIOS (Basic Input Output System). The BIOS is the only part of CP/M that differs from one type of computer to another. The BIOS is used by the BDOS to access the computer's hardware, which includes the console, disk drives, and printers. Since CP/M programs interact with the computer through the BDOS, they are insensitive to the differing BIOSs. There is no restriction on where the BIOS may reside in memory as long as it does not interfere with the BDOS or the TPA. Generally, the BIOS resides directly above the BDOS. In the Apple using the standard Microsoft SoftCard, the 56K BIOS starts at DA00H, which is immediately above the BDOS, while the 60K BIOS starts at F800H, which is not adjacent to the BDOS. An Apple IIe using the Microsoft Premium SoftCard IIe (PS IIe) has part of the BIOS inside the BDOS area. The PS IIe is a 64K CP/M version, and Microsoft calls it version 2.26.

■ The CCP

When CP/M is first booted, it is placed in the command mode. This is evidenced by the A> prompt. The command mode is actually a program called the CCP (Console Command Processor). The CCP is loaded into memory just below the BDOS. Because the CCP is not loaded into the TPA at 0100H, you can run many CP/M programs without overwriting the CCP. This is very useful and convenient for programmers and is particularly helpful in debugging programs. The CCP, however, may be overwritten by a program to make more memory available. For this reason, the warm boot reloads the CCP into memory.

The warm boot is a CP/M routine that reinitializes the computer. It is commonly used by programs as a way of exiting to the command mode. The warm boot must reset certain parameters, reload the CCP, and put the computer into the command mode. Since the CCP is loaded from the disk in drive A:, there must always be a CP/M-bootable disk in drive A: prior to a warm boot. You cause a warm boot, for example, by entering a CONTROL-C while CP/M is in the command mode.

The CCP is used only when CP/M is in the command mode, that is, when a prompt such as A> is displayed. The CCP prompt may take the form B> or C> or D>, and so on, up through P>. The letter in the prompt indicates which disk drive is the default drive. CP/M allows up to sixteen disk drives. The Apple user, however, is permitted only six drives in Microsoft's CP/M version 2.20B and only four drives in Microsoft's CP/M versions 2.23, 2.25, and 2.26.

The CCP is the link between your Apple's keyboard and CP/M. The CCP has built-in commands just as does Apple DOS, but the CCP and Apple DOS commands have only a few similarities, which will become apparent as they are discussed. The CCP *command line* (that is, the line on which the CP/M prompt, such as A>, appears) can be as long as 255 characters. The command is accepted only after a carriage return is entered (you enter a carriage return on the Apple by pressing the RETURN key). There is one exception to the carriage return requirement. If you press CONTROL-C at the beginning of a line, you need no carriage return. CONTROL-C performs a warm boot.

CCP Line-Edit Commands

The CCP has the following line-editing features:

1 ← The backspace (same as CONTROL-U) backspaces and overwrites the character to the cursor's left. The CCP backspace is similar to the Apple DOS backspace, but you cannot go further left than the CP/M prompt, A>.

2 **CONTROL-X** This command deletes the entire command line by backspacing to the CP/M prompt.

3 **DELETE** This is CONTROL-@ on the Apple II. This command removes the character typed and echoes that character back to the terminal. This command is a holdover from teletypewriter (TTY) days, and you do not need it when you are using the Apple display monitor. Its use may cause you some confusion and therefore should be avoided.

4 **CONTROL-E** This command performs a jump to the beginning of the next line without issuing a carriage return. CONTROL-E is of value to TTY users.

5 **CONTROL-R** This command writes a # at the cursor and jumps to the beginning of the next line. The current command line is then retyped. Again, this command is of value to TTY users.

6 **CONTROL-J** This is the line feed command. Entering it has the same effect as pressing the RETURN key.

7 **CONTROL-M** Entering this command is another way of entering a carriage return and therefore has the same effect as pressing the RETURN key.

The CCP console input routine has one more very useful command. CONTROL-P is a switch for echoing the screen output to the printer. First of all, a printer must be connected to an interface card in slot 1 of your Apple in order for this command to have any meaning. You can enter CONTROL-P at any time to cause the screen output to appear at the printer. You turn off the printer output by again pressing CONTROL-P. You turn the printer output on and off by alternate CONTROL-P entries. Be sure that the printer is turned on before using CONTROL-P. If the printer is off, the computer will probably hang up. You can restore the computer to normal operation by turning the printer on.

Technically speaking, the CONTROL-P command is not a CCP command; it is a BDOS command. The BDOS is used by the CCP to print to the console. When the BDOS sees a CONTROL-P coming from the CCP, the console output is then echoed to the printer.

CONTROL-S is another BDOS command that is used by the CCP. Pressing CONTROL-S while there is output to the console, such as with the DIR command, stops the output. The console output is resumed after any other console key is pressed.

Again, the CCP commands are entered immediately after the prompt and are always terminated by the pressing of the RETURN key. The notation <CR> after each command is to be understood and will no longer be explicitly written in this text. A description of the built-in CCP commands follows.

CCP Built-in Commands

DIR

DIR prints the directory of files found on a diskette. DIR is similar to the Apple DOS CATALOG command. If the default drive is A:, then the directory of the diskette in A: is obtained by entering

```
DIR
```

Please note that you do not enter the prompt A>, but that you enter DIR immediately after A>. To get the directory of the diskette in drive B:, enter

```
DIR B:
```

We can generalize this last command to

```
DIR x:
```

where *x*: is the drive on which the directory is to be displayed. The allowable values for *x* are A, B, C, . . . , P. An Apple with two drives will allow only the values A and B for *x*. The directory example below illustrates the way files may be named in CP/M.

Sample Directory

```
A: DDT      COM : PIP      COM : DUMP     ASM : LETTER   TXT
A: INVADERS BAS : REGRESS  FOR : SYSGEN   REL : LETTER
A: RESUME   DOC : DUMP     HEX : INVENT   1   : INVEN    2
A: DUMP     PRN : REPLY
```

You will notice that some file names (PIP COM, LETTER TXT, and DUMP ASM, for instance) consist of two parts. The second part of the file name is called the *extension* or *file type*. File names need not have extensions, but extensions help the user differentiate among various kinds of CP/M files. For instance, here the TXT extension is used to indicate that the file LETTER.TXT is simply a text file, while the ASM extension is used to indicate that file DUMP.ASM is an assembly-language source-code file. The period between the file name and file extension is standard nomenclature in CP/M. The rules to follow for naming files are that

1 The file name must not exceed eight characters in length.

2 The file name must not contain the following characters:

<
>
.
,
;
:
=
?
*
[
]

3 Blank spaces are not permitted.

4 The file extension must not exceed three characters in length.

The DIR command permits the use of ambiguous file names and file types. An ambiguous name is one that contains either question marks (?) or asterisks (*) in place of a character or characters. The question marks and asterisks so used are called *wild-card characters*. For example, let's assume that your diskette contains the files shown below.

```
A: MYTEXT   TXT : OUR      TXT : MYTEXT1  TXT : MYTEXT1A TXT
A: YOUR     TXT : LETTER   TXT : DOCUMENT     : LAST     BAS
```

You wish to get a listing of only those files with names that begin with MYTEXT and have the extenstion TXT. The command

```
DIR MYTEXT?.TXT
```

will print the following display to the console screen:

```
A: MYTEXT   TXT : MYTEXT1  TXT
```

The ? is used as an ambiguous character (a wild card), which means that any character found in the ambiguous character's position is to be ignored. The command

```
DIR ????????.TXT
```

will display all files with the extension TXT. Remember that a file name may contain up to eight characters; therefore, eight ?s are required to list all possible files with the TXT file extension.

The use of ambiguous characters is handy, but having to fill the name with ?s can be tedious. There is a shortcut available. It is the *. The * may be used to fill the remaining available positions with ?s. For example:

```
DIR *.TXT
```

is equivalent to DIR ????????.TXT. Or if you want to list all file names beginning with Q and having the extension DOC, then

```
DIR Q*.DOC
```

will produce the desired results. The ambiguous character may also be used in file-name extensions. For example:

```
DIR MYTEXT?.?X?
```

and

```
DIR MYTEXT?.*
```

are valid commands. Finally, each of the following three commands will produce the complete directory listing of the diskette in drive B:.

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

Please note the use of spaces in the above examples. The DIR is a CCP command. All CCP commands must be followed by one space to separate the command from the remainder of the command instruction. There are no other spaces permitted in the command line. Unlike Apple DOS, CP/M is quite finicky about spaces. If you find that you are getting error messages after entering commands, the first thing you should check is whether there are any extraneous spaces typed into the command line.

REN
REN is used to rename a file. REN is similar to the Apple DOS RENAME command, but there is a major difference in the order in which the file names must appear. To rename the file LETTER.TXT to TEXT.LTR requires entering

```
A>REN TEXT.LTR=LETTER.TXT
```

A DIR command will now show that the file LETTER.TXT has been replaced by TEXT.LTR.

The logic of the REN command line structure is mathematical. You may think of the command as an algebraic equation: the left side of the equation is set equal to the right side. The mathematical analogy is used in all CP/M commands containing an equals sign. (Strictly speaking, this is not algebraic logic but memory replacement logic, the difference being that under memory replacement the memory location remains the same, but the contents of the location get altered, whereas algebraic logic would assign a new location to a new value. Fortunately, the distinction need not be made by the user for nearly all the applications found in this text.)

REN can be used with or without file-name extensions. Consider the following three REN command lines:

```
REN NEWFILE=OLDFILE
REN NEWFILE.DOC=OLDFILE
REN NEWFILE=OLDFILE.TXT
```

The renamed files will appear in the directory with or without extensions as demanded by their new names.

You can rename a file on any drive by specifying the file drive in the command line. For example:

```
REN B:NEWNAME=B:OLDNAME
```

says that the file OLDNAME on drive B: is to be renamed NEWNAME. It is important that the drives specified on both sides of the equals sign be the same; otherwise an error condition will result. If you think of the equals sign as signaling an equation, you can avoid most errors. The use of ambiguous characters is not permitted in the REN command.

Sometimes the error message

```
BDOS ERROR: R/O
```

will appear if an attempt is made to rename a file. There are two sources of this error. The first is that there is a write-protect tab on the diskette. The second is that the diskette was not properly logged in (usually because there has been a diskette change in the drive but there has been no warm boot). The solution to the first problem is to remove the write-protect tab. The solution to the second problem is to do a warm boot by pressing CONTROL-C.

ERA

The ERA command erases files. ERA is similar to the Apple DOS DELETE command. Examples of ERA in use are

```
ERA BADFILE
ERA NONEED.TXT
ERA OLD???.TXT
ERA *.DOC
```

If ambiguous characters are used, all files with names fitting the ambiguous description will be erased. For instance, the command ERA *.DOC will erase all files with the DOC file-name extension. The use of ambiguous characters in the ERA command should not be haphazard. Once a file is erased, there is no CP/M command to restore it. For this reason, when the command

```
ERA *.*
```

is given, the CCP responds with

```
ALL (Y/N)?
```

If a Y is entered, the disk is completely erased.

An attempt to erase a file may produce the error message

```
BDOS ERROR ON x:R/O
```

where *x* is the drive name (any letter A through P). The solution is the same as that given in the REN discussion.

The erasure of a file does not mean that the file is destroyed. It means that the directory space where the file name and file allocation information are stored is internally labeled as available to be overwritten by a new file.

TYPE

The TYPE command has no equivalent in Apple DOS. TYPE reads a file and prints it to the console. This useful command lets you determine what a file contains without having to load it into a text editor. Consider two examples of usage:

```
TYPE LETTER.DOC
TYPE B:DIARY
```

The file prints to the screen quite rapidly. If you want to stop the printing temporarily, press CONTROL-S. To restart the printing, press any other key. To abort the TYPE command, press any key other than CONTROL-S.

Ambiguous file names cannot be used with TYPE. Using the TYPE command on nontext files can produce some strange results and should be avoided.

USER

CP/M allows you to assign files to specific groups in the disk directory. These groups are called *user areas*. Up to sixteen user areas may be specified. To enter a user area, issue the command

```
USER n
```

where *n* is any number 0 through 15. User areas may be used to segregate the diskette files. For instance, the correspondence files may be kept in user area 0, the accounts receivable files in user area 1, the accounts payable files in user area 2, and so forth. Only one user number can be in effect at a time; therefore, the current user number is active for all logged-in drives. For example, if drive A: is active and you change the user area to 1, then the user area for drive B: is automatically 1. You cannot use drives A: and B: simultaneously with different user area numbers.

DIR commands will list the files of the currently active user area only. For that matter, all CCP commands act only on the currently active user area. You cannot erase a file in user area 1 if you are logged into user area 3. Unfortunately, the CCP doesn't provide a means for displaying the user area number. The user area number can be made known through the STAT utility discussed in chapter 5.

Lest you wax ecstatic over the virtues of user areas, you should know that there is a major inconvenience involved in using them. You cannot easily get files to cross user areas. This means, for example, that a word-processing program such as WordStar, if it is stored in user area 0, cannot load files from user area 1. If you need WordStar for user area 1, you must transfer it either by using the CCP SAVE command or by using the PIP utility covered in chapter 6. In either instance, you would have the WordStar program saved twice on the same diskette, which is a wasteful use of disk space. For this reason, the use of user areas is discouraged for the Apple. User areas are better suited to systems with large-capacity disk drives. The Apple 5¼-inch drive has a capacity of a little over 100 kilobytes, which is not enough to make user areas viable.

The default user area number is 0. The cold boot always brings you up in user area 0.

SAVE

The SAVE command is similar to the Apple DOS SAVE, but its use is much more restricted. The SAVE command will write to the diskette the memory area starting at 0100H and ending at a specified location. It is used primarily in conjunction with the DDT utility. The SAVE command is included here for completeness, but it is discussed more fully in chapter 8.

You may have noticed that all the CCP commands have appeared in uppercase letters. Unlike Apple DOS, CCP does not in fact restrict you to using uppercase only. If you have an Apple IIe or a lowercase adapter for your Apple II, you can enter all CCP input in lowercase if you want.

The CCP converts all input to uppercase. The directory appears in all uppercase letters, and the command

```
dir *.txt
```

generates the same results as

```
DIR *.TXT
```

■ The Transient Command File

Comparing the CCP to Apple DOS at this point will give you the impression that Apple DOS has nearly five times as many commands as does the CCP. This is not an accurate impression. The CCP can extend its command options to be almost limitless by using transient command files. To understand how the CCP uses a transient command file, consider the following example.

Assume that you wish to transfer the file DUMP.ASM from the diskette in drive A: to the diskette in drive B:. The diskette in drive A: must have two files on it. It must obviously have DUMP.ASM, and it must also have the file called PIP.COM. The command that will transfer the file is

```
PIP B:=DUMP.ASM
```

The disk drives will go into action in response to this command. The file will be transferred, and the CP/M prompt will reappear. A directory listing of drive B: will now include the file DUMP.ASM.

The CCP handled the PIP command by first looking for PIP in its internal command list. PIP is not an internal CCP command, so the CCP then looked for a transient command file in drive A: with the name PIP. A transient command file is identified by the extension COM. The CCP then loaded PIP into the TPA, where PIP was then run. PIP then processed the rest of the command line and returned to the CCP by doing a warm boot. A complete description of PIP may be found in chapter 6.

Anything entered on a command line and followed by a carriage return is considered by the CCP to be a command. If the command isn't internal or in the transient command file found on the diskette, the CCP echoes the command line and follows it with a ?.

The transient command has a very useful extension. If, for example, the currently logged-in drive is B: and the file PIP.COM is on drive A:, the last example becomes

```
A:PIP B:=DUMP.ASM
```

The generalized transient command is

```
x:tcfile <command line>
```

where *x*: is the drive where the transient command file (*tcfile*) is to be found. The drive can be omitted in the command; then the currently active drive is assumed. An optional command line is permitted. The command line in the example just cited is B:=DUMP.ASM.

The use of transient command files is similar to the use of RUN and BRUN in Apple DOS, but the entering of the file name is sufficient to run the program. Note that running a transient command file requires typing the name of the file without the extension; typing PIP.COM would produce an error message. The transient command file allows the inclusion of a variety of parameters in the command line. Apple DOS allows only a very restricted use of command parameters.

There is a way to fool the CCP into ignoring a command line. Typing one of the following characters as the first entry in a command line will prevent the CCP from acting:

```
:
;
>
<
```

This feature is useful for making comments. The CONTROL-P switch may be activated to get a hard copy. Digital Research doesn't document this feature. It therefore may not be present in CP/M versions other than 2.2x.

The Apple has a RESET key that is not defined by CP/M. Microsoft in its CP/M versions 2.20B and 2.23 has made hitting the RESET key cause a warm boot. Other Apple CP/M versions make pressing the RESET key cause the system to perform a cold boot.

■ A Summary of CCP Built-in and Miscellaneous Commands

The CCP built-in and miscellaneous commands are as follows:

CONTROL-C Causes a warm boot. You need no carriage return (Apple RETURN key) to enter the command.

CONTROL-P Acts as a switch to toggle on and off the listing device, that is, the printer. The Apple's listing device is the device connected to slot 1.

CONTROL-S Stops terminal output. The output resumes if any key is pressed.

DIR x:filename.ext Searches the diskette directory of drive *x*: and displays all matching files. Ambiguous file names are allowed in the command line. If no drive is specified, the active drive is assumed. If no files are specified, then all files are displayed.

ERA x:filename.ext Erases the file on drive *x*: with the specified file name. If no drive is specified, the active drive is assumed. Ambiguous file names are permitted. Using ambiguous file names erases all matching files.

REN x:filename1.ext1=x:filename2.ext2 Renames the file named *filename2.ext2* on drive *x*: as *filename1.ext1*. Ambiguous file names are not permitted. If the drive is not specified, the active drive is assumed.

SAVE n x:filename.ext Saves the memory in the TPA starting at 256 (100H) and ending at $[(n + 1) \times 256]$ to the diskette with the specified name. If the drive is not specified, the active drive is assumed.

TYPE x:filename.ext Prints the specified file to the screen. The CONTROL-P and CONTROL-S switches are active. If no drive is specified, the active drive is assumed.

USER n Changes the user area; *n* may be any value 0 through 15. The cold boot always brings up user area 0.

■ A Summary of CCP Line-Edit Commands

The CCP line-edit commands are as follows:

← The backspace character. Deletes the character to the left of the cursor.

CONTROL-X Deletes all characters on the current command line.

CONTROL-R Retypes the current line.

CONTROL-J Generates a carriage return–line feed combination. This is equivalent to pressing the Apple RETURN key.

CONTROL-E Terminates the current command line without causing the CCP to take action:

DELETE Deletes the character to the cursor's left while echoing that character to the console. The Apple II uses the CONTROL-@ for this function.

4 Fundamentals

The CP/M Concept

CP/M is meant to be an operating system that is not computer-specific, so that a program written on computer *x* can be run without modification on computer *y*. The only condition CP/M imposes is that the computer use an 8080 or Z-80 microprocessor. The method used by Digital Research, Incorporated (DRI) to create this computer-independent operating system will now be discussed.

DRI divided the CP/M operating system into two parts. The first part is called the Basic Disk Operating System (BDOS). The second part is called the Basic Input Output System (BIOS). The BDOS is the heart of the CP/M operating system and is copyrighted by DRI. The purpose of the BDOS is to act as an intermediary between the program and the computer's hardware. A program written for a CP/M-based computer accesses the hardware by calling a routine in the BDOS. As an example, let's write the letter *A* to the video monitor. Incidentally, CP/M assumes that all BDOS calls are in machine language, which means that the example must be in machine language. The prescribed method for writing the *A* to the monitor is to place 41H in the Z-80's E register, place 2H in the Z-80's C register, and perform a call to location 5H (hexadecimal notation is used; see Appendix A). These instructions will write the letter *A* to the screen of any computer operating under CP/M.

CP/M versions 2.20 and higher have thirty-nine BDOS routines for hardware and disk file manipulation. The BDOS calls and procedures are included in the Microsoft CP/M documentation. Some suppliers of Apple CP/M do not include this documentation in their packages. Be sure to inquire if the DRI *CP/M 2.0 Interface Guide* is part of the documentation. It is possible to acquire all the generic CP/M documentation directly from DRI.

The CP/M programmer benefits greatly from the BDOS since writing software is simplified by not having to consider the hardware variations among computers. Computers, however, do vary greatly among manufacturers. CP/M handles these variations by making the BDOS interact with hardware through the BIOS.

The second segment of the CP/M operating system, the BIOS, is not supplied by DRI, but DRI does specify what the BIOS must do. It is left to the computer user or manufacturer to provide the BIOS. The BIOS must contain routines that perform seventeen hardware input/output functions. These I/O functions must be placed in a specified memory location, and they must perform the hardware functions defined by DRI. A detailed discussion of the BIOS is given in chapter 10.

The BIOS hardware routines perform simple functions such as reading a character from the keyboard or reading a disk sector into memory. The BDOS integrates the simpler BIOS functions into routines that can do considerably more complex tasks. For instance, the BDOS uses the BIOS disk-sector read and write routines to create disk files, delete disk files, append disk files, form disk directories, and so on. The BDOS can do this because it knows that in order to read a given disk sector, all it has to do is initialize the registers of the Z-80 in a certain way and call a routine at a known memory location. The BDOS can write to a disk sector and perform similar hardware I/O by analogous methods.

■ CP/M Devices

CP/M brings uniformity to the myriad configurations by creating logical devices. The BDOS performs all its I/O in terms of logical devices. The reason for this is that the BDOS doesn't know what hardware (physical devices) is attached to the computer. A command, such as one that outputs data to a printer, results in the BDOS's writing the data to the logical listing device called LST:. The BDOS output to the LST: device goes to the LST: routine in the BIOS. The BIOS LST: routine then routes the output to the proper physical device, that is, the printer. You can understand the usefulness of such a scheme if you consider the instance of two printers attached to the computer. One is a fast dot-matrix printer, and the other a slower daisy-wheel printer. We wish to quickly print out the rough draft of a letter that is stored in a disk file. The BIOS is instructed (through IOBYTE, discussed below) to write the LST: output to the dot-matrix printer. If we use the CONTROL-P switch and list the letter with the CCP TYPE command, the letter will be printed on the dot-matrix printer. In a similar way, we can instruct the BIOS to print the letter on the daisy-wheel printer.

CP/M Logical Devices

The logical devices used by CP/M are the following:

CON: The console device, which includes the keyboard and display device. This is the default device used by the CCP (Console Command Processor).

RDR: The paper-tape reader. This name is a throwback to an earlier time. Paper punches are not frequently seen on today's microcomputers. RDR: has become a generic name for a wide class of input devices.

PUN: The paper-tape punch, another name that is a throwback. PUN: is the generic name for any of a variety of output devices.

LST: The output listing device. This is most commonly a printer.

You have probably noticed that all devices have names that end with a colon. The colon is used by CP/M as a *delimiter* that identifies devices. *Delimiter* is computerese for a symbol that is used by the computer to indicate the beginning or end of a data string. This is why you use a colon in certain commands when requesting a disk operation, such as

```
A>DIR B:*.COM
```

The disk drive is a device, and the directory command will know to look to drive B: for the directory.

The CON: and LST: devices are directly accessible from the CCP; the RDR: and PUN: devices are not. The LST: device is activated by the CONTROL-P switch.

Associated with each logical device is an assortment of physical devices. The assignment of a physical device to a particular logical device takes place in the BIOS. Because the logical devices are generic, physical devices can be assigned in ways that may seem slightly odd. For example, it is possible to assign the CRT: physical device to the logical LST: device. This assignment makes the output of the LST: device appear on the video monitor instead of going to a printer.

CP/M Physical Devices

The physical devices that may be assigned to the logical devices are the following:

TTY: The teletypewriter device. This is the slow console device, which is usually a printer-keyboard combination.

CRT: The cathode-ray-tube device. This is the high-speed console device.

BAT: The batch-processing device. The input is the current RDR: device. The output is the current PUN: device.

UC1: The user-defined console device. This device can be anything the CP/M user desires, provided it can perform the functions of a console.

PTR: The high-speed paper-punch reader.

UR1: The user-defined reader device. This device is an alternate input device defined by the user.

UR2: A second-user defined reader device.

PTP: The high-speed paper-tape-punch device.

UP1: A user-defined punch or input device.

UP2: A second user-defined punch or input device.

LPT: The line printer.

UL1: A user-defined list device, usually a second printer.

As mentioned above, the physical devices are selected by the changing of the IOBYTE. For further information on the reassignment of physical devices, see chapter 5.

The IOBYTE is found at memory location 0003H and defines the active device by the bit arrangement. Table 4.1 shows the bit configuration for each physical device. Note that this table also gives the allowable physical-device assigments for each logical device.

To see how these concepts are used, consider the case in which the BDOS is requested to send a character to the printer. The BDOS outputs to the printer by placing the character in the C register of the Z-80 and calling the sixth BIOS function, named LIST. The LIST function looks at

Table 4.1 ■ IOBYTE Bit Configuration

Logical Device	*Physical Device*	*IOBYTE*
CON:	TTY:	*xxxx xx*00
	CRT:	*xxxx xx*01
	BAT:	*xxxx xx*10
	UC1:	*xxxx xx*11
RDR:	TTY:	*xxxx* 00*xx*
	PTR:	*xxxx* 01*xx*
	UR1:	*xxxx* 10*xx*
	UR2:	*xxxx* 11*xx*
PUN:	TTY:	*xx*00 *xxxx*
	PTP:	*xx*01 *xxxx*
	UP1:	*xx*10 *xxxx*
	UP2:	*xx*11 *xxxx*
LST:	TTY:	00*xx xxxx*
	CRT:	01*xx xxxx*
	LPT:	10*xx xxxx*
	UL1:	11*xx xxxx*

the 2 high bits of the IOBYTE and decides which output routine to use. If the 2 bits are cleared, the character will be sent to the TTY: device. If the leftmost bit is cleared and the adjacent bit is set, the character is sent to the CRT: device. If the leftmost bit is set and the adjacent bit is cleared, then the character is sent to the LPT: device. If both bits are set, then the character is sent to the UL1: device.

You can alter the IOBYTE by using DDT (see chapter 8) or by using STAT (see chapter 5). Microsoft CP/M has the following default values for the IOBYTE:

```
CON:=CRT:
RDR:=PTR:
PUN:=PTP:
LST:=LPT:
```

The Microsoft Use of Physical Devices

Microsoft CP/M implements the physical devices through the use of vectors in the I/O Configuration Block (IOCB). A vector is a 2-byte location in memory containing the address of a routine. For the case of the IOCB, the vectors point to the I/O routines in the BIOS. The IOCB vector locations depend on whether the BIOS is a standard SoftCard (SS) BIOS or a Premium SoftCard IIe (PS IIe) BIOS. The 2-byte vector locations are defined as shown in table 4.2.

Consider again the BDOS's sending of a character to the logical LST: device when you are using the Microsoft SS BIOS. The BDOS will enter the BIOS through the LIST function. The LIST function checks the IOBYTE. If the IOBYTE is of the form 00*xx xxxx*, then the list function will read locations F386H and F387H and jump to the address found there. If the IOBYTE is of the form 01*xx xxxx*, then the LIST function will again jump to the address found in F386H and F387H. If the IOBYTE is of the form 10*xx xxxx*, then the LIST function will jump to the address found in F392H and F393H. Finally, if the IOBYTE is of the form 11*xx xxxx*, then the LIST function will jump to the address found in F394H and F395H.

Disk I/O

The most complex hardware handling that CP/M has to undertake is the disk I/O. All disk systems partition the media (floppy disks, floppy diskettes, hard disks, and so on) into tracks and sectors (refer to Appendix C). The tracks are concentric rings about the center of rotation. Each track is divided further into sectors. The Apple divides the diskette

Table 4.2 ■ Microsoft IOCB Vector Locations

Logical Device	*Function*	*Microsoft BIOS Version*	
		SS	*PS IIe*
CON:	Console Status (This is not a physical device and is not altered by the IOBYTE. The Console Status is a supplementary function that may be used by any of the physical devices assigned to CON:.)	F380H	F3C0H
	Console input vector for the TTY: and CRT: devices*	F382H	F3C2H
	Console input vector for the UC1: device	F384H	F3C4H
	Console output vector for the TTY: and CRT: devices*	F386H	F3C6H
	Console output vector for the UC1: device	F388H	F3C8H
RDR:	Reader input vector for the PTR: device	F38AH	F3CAH
	Reader input vector for the UR1: and UR2: devices**	F38CH	F3CCH
PUN:	Punch output vector for the PTP: device	F38EH	F3CEH
	Punch output vector for the UP1: and UP2: devices***	F390H	F3D0H
LST:	List output vector for the LPT: device	F392H	F3D2H
	List output vector for the UL1: device	F394H	F3D4H

*Microsoft makes TTY: and CRT: identical.
**Microsoft makes UR1: and UR2: identical.
***Microsoft makes UP1: and UP2: identical.

into thirty-five tracks with each track divided into sixteen sectors. Some geriatric Apples may have thirteen-sector tracks, but these are no longer common and will be neglected in this discussion.

Data is written to and read from the diskette one sector at a time. An operating system such as CP/M will set up a scheme to efficiently store data on the diskette. The method CP/M uses is to arrange the diskette into parts: the *system tracks*, the *directory sectors*, and the *data sectors*. The first two or three tracks are used by CP/M to store the CP/M operating system, which includes the cold boot program, the BDOS, the BIOS, and the CCP. The Apple diskette uses the first three tracks as the system tracks. The track immediately following the system tracks contains the directory sectors. Since the first track is numbered 0, the directory is found on track 3 for the Apple. The directory sectors contain the file names and the locations where the files are stored on the diskette.

The way the BDOS stores files is a little involved. First of all, files are stored as *blocks*, also called *groups*. A block can have many different values depending on the computer and the disk drives. The Apple block is the smallest permissible size of 1,024 bytes. The Apple diskette sector is 256 bytes; therefore, the block is a unit using four sectors. The BDOS must write to the diskette in sector units. The blocking algorithm optimizes the disk-access time by physically locating the block sectors on the diskette to ensure that a minimum length of time is needed to access sequentially written sectors. If this seems complicated, there is more. The BDOS handles the diskette data in logical sectors of 128 bytes. This makes an Apple block eight logical sectors in length. It is up to the BIOS to translate the logical sectors into physical sectors.

The reason for logical sectors is to ensure that CP/M programs can be run on any disk format. The reason for a logical sector's being 128 bytes is that CP/M was originally written for systems using the IBM 3740 disk format, which involves an 8-inch disk having seventy-seven tracks, twenty-six sectors per track, and 128 bytes per sector. The first logical sector is numbered 0. Physical sectors on many disk formats, the Apple included, also start at 0. There is at least one exception to this physical numbering scheme, and it is the 3740 format, which gives the number 1 to the first physical sector. It is up to the BIOS to relate the proper physical sector number to the logical sector number. This may seem to be unnecessary hairsplitting if you have an Apple, since the logical and physical sectors both start at 0. Some disk utilities running under CP/M assume that your disk has a 3740-type format and start physical sector numbering at 1. If you don't know what's going on, you can get into trouble.

The BIOS must provide tables to tell the BDOS how many tracks there are per disk, how many logical sectors per track, how many system tracks, the maximum number of directory entries, and some data for blocking. The Apple diskette has thirty-five tracks, thirty-two logical sectors per track, forty-eight directory entries, and three system tracks. Forty-eight is an unusual number of directory entries for most CP/M systems; sixty-four is more common. Microsoft chose forty-eight entries because the Apple diskette is small in capacity compared to most other systems, and the diskette would most probably be filled up before sixty-four files could be stored. The smaller number of directory entries also increases the speed of disk operations because directory checks are frequently made by the BDOS during those operations. The fewer entries to be checked, the faster the disk operation.

5 The STAT Utility

The transient command program STAT.COM is a CP/M hardware and disk statistics program. STAT.COM is a powerful and extremely useful utility provided by Digital Research. The STAT command enables the user to check on or change the status of the system hardware or disk files.

Disk Space

The STAT command will give the space usage for currently logged-in diskettes. For example, entering STAT followed by a carriage return will print

```
x: R/W, Space: nnnK
```

for each drive that has been logged in. A more specific example is

```
A: R/W, Space:  22K
B: R/O, Space: 100K
```

Here we have two logged-in drives. The R/W means that drive A: can be either read from or written to. The R/O means that drive B: is in read-only condition and cannot be written to. The R/O status can come from swapping diskettes in drive B: and not relogging in B: with a warm boot (by issuing a CONTROL-C, for instance). You also make drive B: R/O by explicitly making it so with the STAT command. The *Space* in each of the above examples is the number of kilobytes remaining on the diskette.

The space remaining on a specific diskette may be found by entering

```
STAT x:
```

where *x*: is the drive requested. For example:

```
STAT A:
```

will print

```
Bytes remaining on A: 22K
```

If you change the diskettes without doing a warm boot, the STAT command gives the incorrect remaining space for the swapped diskette. It is a good idea to do a CONTROL-C before using the STAT command to ensure that you don't get misleading results.

■ File Size and Parameters

The space used by a given file and the status of that file may be found by entering either

```
STAT file.ext
```

or if the file is on drive *x*:

```
STAT x:file.ext
```

For example, we wish to learn the status of the file PIP.COM. The command

```
STAT PIP.COM
```

will print

```
Recs  Bytes  Ext  Acc
  58     8k    1  R/W   A:PIP.COM
```

The *Recs* is the number of 128-byte logical sectors used by the file. *Bytes* is the file length in 1,024-byte (kilobyte) multiples. The bytes used are in block multiples. Remember, for the Apple a block is eight logical sectors, or 1K. Since one block is the smallest unit allocated to a file, the number of records (*Recs*) is often less than the number of blocks times the block size. For example, the *58* records used by PIP.COM indicates that fifty-eight logical sectors are actually required to store that file; that is, PIP.COM is 7,424 bytes long. CP/M, however, requires that eight blocks be reserved for the PIP.COM file. This means that PIP.COM actually uses 8,192 bytes of disk space. CP/M is slightly wasteful in using disk space. What is lost in file space economy is more than made up for in increased speed of disk operations.

The extent (*Ext*) number is the number of *logical extents* occupied by a file. A logical extent is used by the BDOS (Basic Disk Operating System) for file maintenance and is generally transparent to the user. One logical extent equals sixteen blocks. The CP/M disk directory stores files in terms of logical extents. Depending on the file size, the directory will have the number of entries for that file equal to the number of logical extents. Consider a 22K file that is two logical extents in length. This file would be stored in two directory locations and would reduce the number of new files capable of being stored by two. A file using one extent reduces the number of new files capable of being stored by only one.

The file-access type (*Acc*) indicates the file status, which may be read-only (R/O) or read/write (R/W). A file that is read-only cannot be written to or erased. An R/O file is similar to a LOCKED file in Apple DOS. There is a distinction between an R/O file and an R/O disk. An R/O disk is equivalent to a disk with a write-protect tab, and it can be changed to an R/W disk by a warm boot (provided that there really isn't a write-protect tab on the diskette). An R/O file can be changed to an R/W file only by the appropriate STAT command (to be discussed shortly).

The STAT command permits the use of ambiguous file names. For example:

```
STAT *.COM
```

will give the status of all files with the COM extension. In addition, the ambiguous character ? can be used in the same way as described in chapter 3. You can learn the status of all files on a diskette by entering

```
STAT *.*
```

Until now we have considered the status of only *directory files*. A *directory file* is simply a file that is displayed in response to the DIR command. CP/M permits a second type of file called a *system file*. A *system file* is not displayed in the directory. A system file, however, is displayed by a STAT command. If the file SAMPLE.TXT is a system file, then

```
STAT SAMPLE.TXT
```

will display

```
Recs  Bytes  Ext  Acc
  10     2k    1  R/W  (A:SAMPLE.TXT)
```

if SAMPLE.TXT is set to R/W. Notice that a system file is identified by being enclosed in parentheses.

■ Altering File or Disk Status

The STAT command can be used to change the status of a file or disk. A drive may be set to R/O by a command such as

```
STAT B:=R/O
```

or generally

```
STAT x:=R/O
```

where *x*: is the drive name.

A file or a group of files can be made R/O by commands of the following types:

```
STAT LOCKUP.DOC $R/O
STAT *.COM $R/O
STAT MANY???.TXT $R/O
```

These may be generalized to

```
STAT x:filename.ext $R/O
```

where *x*: is the drive name.

Much as files can be made R/O, they can be made R/W. The general instruction is

```
STAT x:filename.ext $R/W
```

A file can be made a system file with the command

```
STAT x:filename.ext $SYS
```

and a file can be made a directory file with the command

```
STAT x:filename.ext $DIR
```

The ambiguous file names can be used in all of the above STAT commands.

■ Physical Device Assignments

STAT will display the current logical device assignments with the command

```
STAT DEV:
```

Typically, STAT DEV: will print

```
CON: is CRT:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:
```

An explanation of the logical and physical devices is given in chapter 4. The logical devices may be reassigned with commands such as the following:

```
STAT LST:=UL1:
STAT PUN:=CRT:
STAT PTR:=UR1:
```

More than one device assignment is permitted in the command line. The last three assignments could have been written as

```
STAT LST:=UL1:,PUN:=CRT:,PTR:=UR1
```

Each assignment must be separated by a comma without spaces. The available physical assignments have been shown above. The Microsoft CP/M BIOS (Basic Input Output System) for the Apple makes the assignments of physical devices unique. Again, please refer to chapter 4 for the discussion of the logical and physical device assignments for the Apple.

Disk Parameters

The command

```
STAT DSK:
```

will display the disk parameters of each logged-in drive. This command is intended primarily for systems with drives that can read more than one disk format. For instance, 8-inch drives can have single-density or double-density formats and can be single-sided or double-sided. *Density* here means the density of bytes that can be stored on a disk; a double-density disk stores approximately twice as many bytes as a single-density disk. The variety of disk formats is almost limitless, especially if high-density hard disks are considered. Due to the varying capacities of disks available, the STAT DSK: can be quite useful. The standard Apple computer, however, can accommodate only one disk format, so that, typically, when you request STAT DSK:, the following is displayed:

```
    A:Drive Characteristics
1024: 128 Byte Record Capacity
 128: Kilobyte Drive  Capacity
  48: 32  Byte Directory Entries
  48: Checked  Directory Entries
 128: Records/ Extent
   8: Records/ Block
  32: Sectors/ Track
   3: Reserved Tracks
```

Let's examine the drive characteristics, entry by entry:

1 The *128-byte record capacity* is the number of 128-byte logical sectors available for data and directory space. The Apple diskette has 1,024 logical sectors, which translates into 512 physical sectors.

2 The *kilobyte drive capacity* is a restatement of entry 1. A kilobyte is 1,024 bytes. It takes eight logical sectors to make 1 kilobyte. Consequently, entry 1 is always eight times entry 2.

3 The *32-byte directory entry* is called a File Control Block (FCB). The Apple can have a maximum of forty-eight FCBs. The FCB contains the information as to where a file is stored on the diskette. At least one FCB is required for each directory entry.

4 The *checked directory entries* gives the number of directory entries identified when the drive is logged in. The purpose of checking the entries is to recognize if the diskette has been changed in the drive without that drive's having been relogged in. If this check isn't made, it is possible that the BDOS could overwrite files and make a shambles of the diskette. The number of directory entries checked is usually the same as the value given in entry 3.

5 An *extent* is 128 logical sectors. A file *record* is one logical sector. Extents are used by the BDOS for file maintenance. In CP/M 2.2 there are 128 records per extent. The extent is 16 kilobytes.

6 The *records per block* is the number of logical sectors per block. The block is the minimum disk space allocated to a file. The Apple block is 1 kilobyte, or eight logical sectors.

7 The *sectors per track* is actually the number of logical sectors per track, which for the Apple is thirty-two.

8 The *reserved tracks* are the tracks forbidden to the BDOS for data storage. These are usually the system tracks where the CP/M image is stored. The cold and warm boots load from the system tracks. The Apple requires three system tracks.

■ User Areas

CP/M gives no ready indication which user area is active for the currently logged-in drives. The STAT command

```
STAT USR:
```

will list the currently active user number and the user areas whose directories have files. A typical display is

```
Active User : 1
Active Files: 0 1 2
```

This display indicates that the current user area is 1 and that user areas 0, 1, and 2 have files stored in their directories.

■ The STAT Help Command

A help command is available for the identification of the possible device assignments. The command

```
STAT VAL:
```

results in the following listing.

```
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:
```

The *d*: in the above is the optional drive name. Because it is possible under CP/M to name a drive d:, the optional drive name in this book is usually referred to as *x*:.

■ A Summary of STAT Commands

The STAT commands are as follows:

STAT Displays the R/W or R/O attributes of each logged-in drive and the space remaining on each drive.

STAT x: Displays the space remaining on the specified drive.

STAT x:filename.ext Displays the statistics of the specified file. Ambiguous file names will list the statistics of all matching files. If the drive is omitted, the active drive is assumed.

STAT x:filename.ext $attribute Places the file attribute in the disk directory. Ambiguous file names are allowed. If the drive is not specified, then the active drive is assumed. The attributes are as follows:

R/O	read-only
R/W	read/write
SYS	system file, not displayed in directory
DIR	directory file, displayed in directory

STAT x:DSK: Displays the drive characteristics of the specified drive. If the drive is not specified, then the characteristics of all logged-in drives are displayed.

STAT USR: Displays user areas with active files for each logged-in drive.

STAT DEV: Displays the current physical device assignments belonging to each logical device.

STAT logical device=physical device or
STAT logdev1=phydev1,logdev2=phydev2,...,logdevn=phydevn
Assigns the specified physical device to the specified logical device by altering the IOBYTE.

STAT VAL: Displays all available STAT commands, file assignments, and device assignments.

STAT USR: Displays user areas with active files for each logged-in drive.

6 The PIP Utility

Copying Files

The Peripheral Interchange Program (commonly called PIP) is another extremely useful utility provided by Digital Research. PIP provides the means to transfer disk files between drives and all other logical and physical devices. PIP can be used even to merge files.

Typing PIP at the prompt will put you in PIP's command mode. The PIP command mode is identified by the prompt's becoming an asterisk (*). You may exit from PIP to the CCP (Console Command Processor) by either pressing RETURN or entering CONTROL-C. PIP uses the equals sign (=) in the logical sense described in chapter 4. PIP is most often used to copy disk files. The files may be copied to another drive, or they may be duplicated on the same drive. The simplest form of the file-copy command is

```
x:file1.ext1=y:file2.ext2
```

where *x*: is the destination drive and *y*: is the source drive. This command will copy file2.ext2 from drive *y*: to drive *x*: and store it under the name file1.ext1. An example is

```
B:MYTEXT=A:YOUR.TXT
```

where the file YOUR.TXT is copied from drive A: to drive B: and renamed MYTEXT.

When drive names are omitted, the currently active drive is assumed. For the case in which A: is the active drive, the above example can be changed to

```
B:MYTEXT=YOUR.TXT
```

If B: is the active drive, then the example becomes

```
MYTEXT=A:YOUR.TXT
```

If the destination file name is omitted, then the copied file has the same name as the source file:

```
B:=A:YOUR.TXT
```

is the same as

```
B:YOUR.TXT=A:YOUR.TXT
```

An interesting aspect of PIP is that the command

```
B:YOUR.TXT=A:
```

will also perform the copying, although this form is not recommended. When the destination file's name is given and the source file's name is omitted, PIP assumes that both names are identical and proceeds with the copying.

PIP may be used to duplicate a file on the same diskette; for example, the command

```
A:OUR.TXT=A:YOUR.TXT
```

will duplicate the file YOUR.TXT and name the duplicate OUR.TXT.

The use of ambiguous file names is permitted when you are copying files between two drives and the source and destination files have the same name. Consider the following examples.

```
B:=A:*.COM
```

copies all files with the COM extension.

```
B:=A:???.TXT
```

copies all files that have three-letter names and the extension TXT.

```
B:=A:X??Y.*
```

copies all files with four-letter names beginning with X, ending with Y, and having any extension. And

```
B:=A:*.*
```

copies the entire diskette in drive A: to the diskette in drive B:.

■ File Concatenation

PIP will also *merge* files, that is, *concatenate* files in a given sequence. The concatenation command is

```
x:newfile.ext=y:oldfile1.ext,z:oldfile2.ext,...,p:oldfileN.ext
```

where *x*:, *y*:, *z*:, and *p*: are drive names. If the drive names are omitted, then the currently active drive is assumed. An example is

```
B:NEW.TXT=MYTEXT,YOUR.TXT,OUR.TXT
```

This command creates the file NEW.TXT with YOUR.TXT added to the end of MYTEXT and OUR.TXT added to the end of that file.

■ Error Conditions

An error condition may occur if the diskette fills up before the copy is completed. PIP will then print an error message and either terminate the copying or ask for a response. If a termination occurs, the copied file appears in the directory with a $$$ extension. PIP uses the $$$ file extension as the destination file name during the copy process. At the completion of the copying, the file is then renamed.

Another error condition is indicated by the message

```
BDOS ERR ON x: R/O
```

where *x*: is the destination drive. This condition occurs if the diskette in the destination drive is swapped but not logged in or if the diskette is write-protected by either a write-protect tab or the STAT command. Because the destination drive must be logged in, PIP cannot be used to copy files between diskettes using the same drive. Remember that the diskette must be logged in with a CONTROL-C, which will also abort PIP.

■ Swapping Disks

There is a condition under which disk swapping is condoned. Assume that we wish to copy some or all of the contents from the diskette in drive A: to the diskette in drive B:. The diskette in drive A: must be logged in. Drive B: must be either logged in or waiting to be logged in. If neither of the diskettes in A: and B: has the PIP.COM file on it, the diskette from a logged-in drive may be temporarily replaced by a diskette with PIP.COM on it. PIP may then be run and the original diskette put back in its drive. The file copying may now proceed. So, for the example cited, we will replace the diskette in drive A: with a diskette with PIP.COM and give the transient command

```
PIP
```

We will now replace the diskette in drive A: with the diskette containing the files we wish to duplicate. Entering

```
B:=A:*.*
```

copies the entire diskette. Entering

```
B:=A:file.ext
```

copies selected files, as previously discussed. This is possible because it is not necessary to log in a diskette if only transient command programs are to be run. A diskette needs to be logged in if files are to be stored on it. A log-in is done when the drive is accessed for the first time after a warm boot.

■ The CCP Command Line

PIP allows the use of the CCP command line. For example, entering

```
PIP B:=A:*.*
```

is equivalent to entering PIP, then giving the command B:=A:*.*, and then returning to the CCP through a warm boot. The only difference is that the CCP command line usage returns you to the CCP if an error is encountered, whereas the PIP command line usage returns you to the PIP prompt after an error.

■ Copying Files to a Device

PIP can also be used to transfer files between logical or physical devices. For instance, a file may be printed on LST:, the logical list device. The command from the PIP command mode is

```
LST:=x:file.ext
```

where *x*: is the drive that contains the file. Again, if *x*: is omitted, then the currently active drive is assumed. The CCP command line usage would be

```
PIP LST:=x:file.ext
```

Output to a physical list device is similarly performed with one of the physical device designations instead of LST:. For example, the command

```
PIP LPT:=x:file.ext
```

will print the file to the printer device assigned as LPT:. Further, the command

```
PIP UL1:=x:file.ext
```

will print the file to the physical UL1: device. The difference between using the logical device and using one of the physical devices in PIP is that using the logical device involves using the physical device indicated by the IOBYTE (see chapter 4), whereas using the physical device directly bypasses the IOBYTE. Recall that the unmodified Microsoft BIOS (Basic Input Output System) has LPT: and UL1: as the same physical device.

That physical device is connected to the card in slot 1. In chapter 13 we will examine how to alter the Microsoft BIOS to take advantage of having more than one physical device.

Transferring a file to the logical or physical punch device is done in a way analogous to the method for the list device. You can output a file to the logical punch device with

```
PUN:=x:file.ext
```

while in the PIP command mode, or by

```
PIP PUN:=x:file.ext
```

when using the CCP command line. The physical devices are directly accessed in a like manner, as in the following examples:

```
PIP PTP:=x:file.ext
PIP UP1:=x:file.ext
PIP UP2:=x:file.ext
```

The Microsoft BIOS makes the physical devices UP1: and UP2: identical, with each using the card in slot 2 as its output port. I show in chapter 13 how to modify the Microsoft BIOS to take advantage of the different physical definitions.

The punch device may be used as a printer if a printer is attached to a card in the Apple's slot 2. Output to the printer is accomplished by a PIP command to the PUN: device. Remember to turn on your printer before using PIP, or you will cause the Apple to hang up. More commonly, however, the PUN: device is used in conjunction with the RDR: device to transfer files between two computers. One usually does this by identifying the PUN: and RDR: logical devices with physical devices that transfer data serially. The Apple requires a serial-type card in slot 2 in order to transfer files between itself and another computer. To perform the transfer, the two computers must be linked by a cable between their serial *ports*. *Port* is jargon for the site at which a computer can accept input and send output. There are often slight differences in physical makeup between the serial ports, and the cables must be designed to accommodate these differences.

Let's assume you have the Apple's serial card connected to the serial port of a second computer, and you wish to copy the file HIS.TXT from the second computer to the Apple and rename that file HER.TXT. First enter the PIP command mode on the Apple, and type

```
x:HER.TXT=RDR:
```

where *x*: is the optional destination drive. Then on the second computer enter the PIP command mode, and type

```
PUN:=y:HIS.TXT
```

where *y*: is the optional source drive. The disk drives on both machines will become active. When the transfer is completed, the prompt will reappear. The sequence of commands just given is arbitrary and is valid for any two computers using CP/M. If you want to send a file from an Apple to another computer, you need only reverse the command sequence.

The RDR: device is the logical reader device. Had we decided to use a physical device instead, one of the following would have been necessary:

```
x:HER.TXT=PTR:
x:HER.TXT=UR1:
x:HER.TXT=UR2:
```

The unmodified Microsoft BIOS makes the physical devices PTR:, UR1:, and UR2: identical and has them all receiving the input from the card in slot 2. See chapter 13 for information on modifying the Microsoft BIOS.

Files may be copied to the remaining logical device: the console, or CON: device. The CON: device is active in two ways: it outputs to the console, and it accepts input from the keyboard. A command such as

```
PIP CON:=x:file.ext
```

(in which the standard definitions apply) will list the named file to the console screen. This is almost equivalent to the TYPE command—*almost* because the TYPE command automatically expands tabs, but the PIP CON: command expands tabs only if requested to do so. As with the previously discussed devices, the console has its share of physical devices, namely the CRT:, the TTY:, and the UC1:. These devices are identical in the Microsoft BIOS.

■ PIP Parameters

The copying of files can be enhanced by the use of parameters. Consider, for example, the PIPping of the file DUMP.ASM to the console. DUMP is a file supplied by Digital Research to display a file in hexadecimal form. DUMP.ASM is a source file that must be assembled before it can be used. If you are not familiar with 8080 assembly language, don't be concerned. We are going to use this file for demonstration purposes only. To continue with the example, the command

```
PIP CON:=DUMP.ASM
```

prints the file contents to the screen, but you should notice that the display lacks ordered columns. This is because the tab markers in the file have not been expanded. (Incidentally, a tab is the character CONTROL-I.) Now enter the command

```
PIP CON:=DUMP.ASM[T8]
```

The characters inside the brackets are PIP parameters. In the example cited, we have requested that the tabs be expanded to eight spaces. The file will now be printed to the screen with the columns aligned. The CCP TYPE command expands tabs automatically, so that the command

```
TYPE DUMP.ASM
```

will produce results that look the same as those produced by the PIP command with the tabs expanded. If we wish to send the output to the printer and send nothing to the console, then

```
PIP LST:=DUMP.ASM[T8]
```

will do the job. If we would like to see the file appear on the console as well as have it printed, then we need

```
PIP LST:=DUMP.ASM[T8E]
```

The E within the brackets tells PIP to reflect the output to the console device. The order of the parameters is not important.

```
PIP LST:=DUMP.ASM[ET8]
```

will work as well.

The Apple IIe has left and right brackets, but the Apple II does not. See the manual for your 80-column card to obtain the keys for the left and right brackets. The 40-column mode of Microsoft's CP/M uses the CONTROL-K for the left bracket; there is no right bracket. The right bracket is not needed in CP/M commands if it is the last character on the command line.

All the PIP parameters are described below.

B

B invokes the block transfer mode. This mode is useful if data is to be copied into a disk file from a reader device that doesn't allow hardware *handshaking*. *Handshaking* is a computer term for the interaction between computer devices while they are transferring data. Typically, handshaking is the sending device's listening to the receiving device for a message to stop sending data because the listening device's input buffer is full. When the listening device has room in its input buffer, it will then tell the sending device to resume transmission. In some instances, the sending device cannot handshake—when the Apple is receiving data by means of a tape recorder, for example. The block transmission mode tells

PIP to accept data into a buffer until an XOFF character (XOFF is the same as a CONTROL-S) is received. The XOFF must be in the source file. PIP then writes the buffer to the disk file, clears the buffer, and is ready to accept more data. Practically speaking, the data transmission must be at a rate that is slow enough to prevent data from being lost while PIP is writing the data to a file, or the buffer must be large enough to accept the entire input. These conditions make the B option difficult to use on the Apple.

Dn

This option deletes all characters extending past the *n*th column on a line. A line is defined as all characters between the ASCII characters for a carriage return (0DH) or a line feed (0AH); refer to Appendix B. The D option is usually used for outputs to the console or a printer. The value of *n* can be any number from 1 through 255.

E

This parameter tells PIP to echo the output to the console. The E parameter should be used only when text files are being copied. A text file is a file generated by a word-processing or editing program and consists of ASCII characters. A non-ASCII file may print unintelligible garbage to the console and cause the system to behave very strangely.

F

The F parameter is a form-feed (ASCII value 0CH) filter. Form feeds imbedded in a file are ignored and do not appear in the destination file. This option is useful for creating files for printers that cannot respond to form feeds from source files that have imbedded form feeds.

Gn

Files may be copied from different user areas to the currently active user area with the G command. For example, the command

```
PIP SAMPLE=SAMPLE[G2]
```

copies the file named SAMPLE from user area 2 to the current user area. The parameter *n* can range from 0 through 15. Ambiguous file names are not allowed for this option.

A minor predicament arises with the G option. How do you get a copy of PIP to the user area where you would like to have the file copied? The answer is, by using the DDT utility, which is discussed more fully in chapter 8. For the time being, simply do the following:

1 Make sure that there are a copy of DDT.COM and a copy of PIP.COM on the diskette to be used. DDT.COM and PIP.COM can

be copied from the CP/M master diskette to user area 0 on the present diskette.

2 Enter the command

```
DDT PIP.COM
```

3 After the DDT prompt (#) appears, press CONTROL-C to reenter the CCP.

4 Go to the desired user area by entering

```
USER n
```

5 Enter

```
SAVE 29 PIP.COM
```

A copy of PIP.COM is now in the current user area.

H

There is a file format called *Intel hex*. This format is generated by the Digital Research assembler for the construction of an intermediate file used in the process of generating a machine-code file from a program source. The copying of hex files is enhanced by the H option. Nonessential characters are removed, and the file is checked for proper format.

I

The I parameter is used for copying hex files. The null character sequence :00 is ignored in the copying. The I parameter automatically invokes the H parameter. The value of this function is primarily for transferring files from paper-tape readers that use null sequences as record separators.

L

The L parameter instructs PIP to convert all alphabetic characters to lowercase when copying a file. This parameter is intended for transferring files to devices that have no provision for uppercase letters.

N

The N parameter puts a number (followed by a colon and a space) at the beginning of each line being copied. This option is intended for only text files. The line number can contain up to six digits. Leading zeroes are suppressed with the N parameter. The N2 parameter prints each line number using six digits, which means that leading zeroes are included. The N2 parameter replaces the colon and space inserted after the line number with a tab, which you may expand using the T parameter option.

O

CP/M uses the character CONTROL-Z (ASCII character 1AH) to indicate the end of a file. The CONTROL-Z is the last character found in all files produced by programs that create text files. PIP normally continues a file copy until it sees a CONTROL-Z, at which time PIP considers the file copying complete and terminates the copying. Now, all CP/M files are not text files. The most common nontext file is the transient command file. CONTROL-Zs may be interspersed throughout a nontext file. PIP may prematurely terminate copying such a file. The O parameter tells PIP to use another method to determine the end of a file. PIP will then copy the file in terms of directory blocks. The file transmission is completed after the last block is transferred. Incidentally, the O parameter is assumed by PIP when it copies files with the COM extension.

Pn

The Pn option is a directive to help paginate output to a printer. A *form feed* is inserted after *n* lines are printed. A *form feed* instructs a printer to advance to the top of the next page. The parameter *n* may range from 1 through 255. If *n* is 1 or is omitted, PIP will place the form feed after sixty lines. You may use the F option with the P option to remove form feeds imbedded in the text and replace them where desired in the printer output.

Qstring^Z

The Q option requests that PIP examine the file or device being copied for a specified string. When that string is found, the copying is terminated. The symbol ^Z stands for CONTROL-Z and is used to terminate the character string for which PIP is searching. An example of usage is

```
PIP LST:=BOOK.TXT[QCHAPTER 2^Z]
```

The file BOOK.TXT is sent to the printer. When the character string CHAPTER 2 is encountered, the printing is terminated.

R

The R parameter is used to copy system files (see chapter 5). System files will be ignored by PIP unless this option is used.

Sstring^Z

The S option is the obverse of the Q option. Transfer from the source file or input device doesn't start until the specified string is encountered. For example, the command

```
PIP LST:=BOOK.TXT[SCHAPTER 1^Z]
```

starts printing the file BOOK.TXT as soon as the character string CHAPTER 1 is found.

Using the S and Q options together will copy a given file segment. For example, the command

```
PIP SHORTER.TXT=BOOK.TXT[SCHAPTER 1^ZQCHAPTER 2^Z]
```

will create the file SHORTER.TXT, which contains only the material between CHAPTER 1 and CHAPTER 2 of the file BOOK.TXT. Note that the start and stop strings are always copied.

If you wish to use lowercase letters in the Q or S options, then enter the PIP command mode. Using the CCP command line will change all lowercase letters to uppercase. The command

```
PIP LST:=BOOK.TXT[SChapter 1^Z]
```

is identical to

```
PIP LST:=BOOK.TXT[SCHAPTER 1^Z]
```

Tn

Use Tn to set tabs at every *n*th column. Usually, program source files contain the tab character, and ordinary document files produced by text processors replace the tabs with spaces. The document files will show no change if the T parameter is used with PIP.

U

The U option converts all lowercase letters being copied to uppercase. This option is useful for transferring files to devices that do not support lowercase letters.

V

This is the file-verify parameter, which is recognized for disk-file destinations only. PIP will verify the copied file against the original file. PIP copies a file by copying the original file into memory and then writing the memory to the new file. When PIP is asked to verify, it reads the segment just written to the new file and compares it to the segment in memory. This comparison is done for each file segment transferred until the entire file is copied. If a discrepancy is found, the error message

```
VERIFY ERROR
```

is printed, and the transfer is aborted. Copying files with the V parameter can increase the time of copying noticeably.

W

You use the W parameter when you wish to copy a file into an already existing file that is designated as R/O. Using the W parameter avoids responding to the message

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

which will otherwise appear. Entering a Y will change the R/O attribute to R/W. Entering an N will produce the

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

message, and the copying will be aborted.

Z

The Z parameter is used to zero the eighth bit on all ASCII characters being copied. This option is intended for receiving input from a device such as a reader. Most Apple users will have little need for this option.

Concatenation of files may be performed with individual parameters for each file. For instance

```
PIP HUGEFILE=MYFILE[VE],YOURFILE[O],OURFILE[V]
```

will copy MYFILE using the verification option and echo the file to the console, copy YOURFILE under the block transfer option, and copy OURFILE with only the verification option.

■ Special PIP Devices

PIP includes some special logical devices. The most useful of these devices is PRN:. The PRN: is a listing device. An example of usage is

```
PIP PRN:=DOCUMENT.TXT
```

which is identical to the command

```
PIP LST:=DOCUMENT.TXT[NPT8]
```

The special NUL: device is intended primarily for paper punches. Paper tapes separate data segments with null characters. Null characters are innocuous instructions that tell the computer to do nothing. A string of nulls is used to allow equipment enough time to initialize. This time was needed for the older and slower equipment. Modern equipment rarely needs the setup time given by this NUL: device. The NUL: device will literally send forty null characters when evoked. If we wished to place a string of nulls at the beginning and the end of a data file transferred by the PUN: device, then we would enter

```
PIP PUN:=NUL:,DATA.TXT,NUL:
```

Apple users should find no occasion to use the NUL: device.

The special EOF: device performs the task of issuing an end-of-file marker, a ^Z. There is practically no need for this device since PIP automatically issues end-of-file markers when the copying of text files is completed.

There are two other special devices, INP: and OUT:. To take advantage of them, you have to be an experienced programmer. The INP: and OUT: devices permit the user to include his or her own input or output routines in PIP. The command

```
PIP OUT:=filename.ext
```

will output a file via a user-supplied routine. The command

```
PIP filename.ext=INP:
```

will copy a file via a user-supplied routine. The programmer must obey some simple rules in writing the routines. First, the routine for OUT: must be accessed by calling location 106H, and the routine for INP: must be accessed by calling location 103H. Second, the output routine should transmit the character in the C register, and the input routine should place in location 109H the input character with the eighth bit cleared. PIP provides the memory area from 109H through 1FFH for use by these special functions. The OUT: and INP: routines must be entered into the PIP.COM program with DDT (see chapter 8). OUT: and INP: are, again, rarely used devices, and most Apple users will have no need for them.

Finally, there is one more device found in PIP; it is the mysterious IRD: device. IRD: is used for input. It has been included in PIP probably through an oversight. IRD: uses the 8080 ports at 01H and 03H for getting input. The input routine is intended for a specific paper-tape hardware configuration and is of no value to the majority of CP/M computers. Needless to say, it cannot be used on the Apple, since the Apple CP/M cards make no provision for the port functions of the Z-80 microprocessor.

■ A Summary of PIP Assignments

File Commands

The PIP file commands are as follows:

x:filename1.ext1=y:filename2.ext2 Copies files on drive *y*: on the right side of the equals sign (=) to drive *x*: and gives the file the name on the left side of the =. If the drives are not specified, the currently active drive is assumed. If a file name is omitted on either side of the =, the stated file name is assumed for the omitted file name. When no file name is given on the left side of the =, a drive must be specified on the left side. Ambiguous file names are permitted on the right side of the = only, in which case the left side of the = must contain no file names, ambiguous or otherwise.

x:filename.ext1=y:filename2.ext2,z:filename3.ext3,... Concatenates files on the right side of the = in the order given and stores them in the file on the left side of the =. Ambiguous file names are not allowed, and a file name must be specified on the left side of the =.

Output Device Commands

All output device commands are of the form

```
dev:=x:filename.ext
```

Ambiguous file names are not allowed. If the specified drive is omitted, the currently active drive is assumed. The logical devices are

CON:	console (Apple video monitor)
LST:	listing device (Apple slot 1)
PUN:	punch device (Apple slot 2)

The physical devices are

TTY:	teletypewriter device
CRT:	fast terminal device
UC1:	user-supplied console device
LPT:	line printer
UL1:	user-supplied listing device
PTP:	paper-tape punch
UP1:	user-supplied punch device
UP2:	second user-supplied punch device

Input Device Commands

All input device commands are of the form

```
x:filename=dev:
```

Ambiguous file names are not allowed. If the specified drive is omitted, the currently active drive is assumed. The logical devices are

CON:	console (Apple keyboard)
RDR:	reader device (Apple slot 2)

The physical devices are

TTY:	teletypewriter device
CRT:	fast terminal device
UC1:	user-supplied console device
PTR:	paper-tape reader
UR1:	user-supplied reader device
UR2:	second user-supplied reader device

Special Devices

These devices are defined only by PIP.

EOF:	command that sends a CONTROL-Z to the device
INP:	optional user-supplied input device
NUL:	command that sends forty nulls to the device
OUT:	optional user-supplied output device
PRN:	same as LST: with parameters [T8NP]

Transfer Parameters

All transfer parameters are written immediately after the file affected as

```
x:filename1.ext1=y:filename2.ext2[parm1parm2...]
```

The file transfer parameters are

B	transfers blocks
Dn	deletes characters after column *n*
E	echoes the transfer to the console
F	filters form feeds from the file
H	transfers hex data
I	ignores :00 in the transfer (for hex-format files)
L	translates uppercase to lowercase
N	adds line numbers to the transferred file
N2	same as N but includes leading zeroes in numbers
O	transfers object files
Pn	places page ejects after every *n* lines
Qstring^Z	stops transferring from the source after *string*
Sstring^Z	starts transferring from the source with *string*
Tn	expands tabs every *n*th column
U	translates lowercase to uppercase
V	verifies the copy
Z	clears the eighth bit from the input device

7 The SUBMIT Utility

The SUBMIT transient command is the CP/M analog to the Apple DOS EXEC command. Like the EXEC command, the SUBMIT command requires a text file filled with the instructions to be performed. The text file can be constructed with a word-processing program such as WordStar or the CP/M text editor, ED.

Assume that we wish to perform the following sequence of commands:

```
DIR B:
PIP B:=A:*.COM
TYPE A:LOOKSEE.DOC
```

One way is to enter the commands from the keyboard. Another way is to create a submit file that contains the three lines

```
DIR B:
PIP B:=A:*.COM
TYPE A:LOOKSEE.DOC
```

Each line is written as if it were a CP/M command line. This means that there are no unnecessary spaces and all lines end with a carriage return. This file is then saved with a .SUB extension. Let's call this file SILLY.SUB. If the transient file SUBMIT.COM is on the same diskette, entering the command

```
SUBMIT SILLY
```

will create the same results as the instructions that were entered individually from the keyboard.

The SUBMIT command can be extended to include parameters in the command line. If we rewrite the submit file to become

```
DIR $2:
PIP $2:=$1:*.$3
TYPE $1:LOOKSEE.DOC
```

and enter

```
SUBMIT SILLY A B COM
```

we produce the same result as the last submit file produced. SUBMIT identifies the first parameter following the submit file name with the variable $1, the second parameter with the variable $2, and so on. The parameters may be any suitable character strings, and adjacent parameters must be separated by a single space. The general SUBMIT command is of the form

```
SUBMIT submitfile parm1 parm2 parm3 ...
```

The SUBMIT command must be given entirely in the command line. Entering SUBMIT without a secondary command sequence will produce the message

```
Error On Line 001 No 'SUB' File Present
```

and return you to the CCP (Console Command Processor).

The submit file may contain a program that, while executing, requires that data be entered from the console. To enter data into that program, use the XSUB.COM file. As an example, let's create a submit file called EXAMPLE.SUB that will copy PIP.COM. Using a text editor, we create a file with the following six lines:

```
XSUB
DDT
I$1
R
GO
SAVE 29 $1
```

The command

```
SUBMIT EXAMPLE PIP.COM
```

copies the PIP.COM file as follows: first, XSUB is invoked because DDT expects data from the console; second, DDT is run; third, while the system is in DDT the file name is entered; fourth, the file is read into memory; fifth, the command GO exits from DDT with a warm boot; and, sixth, the file is saved to the disk. This example is mainly a demonstration and seems to have little practical value.

When XSUB is used, it must be the first entry in the submit file. XSUB cannot be used anyplace but in a submit file. What happens when the XSUB is executed is that it gets relocated to the memory area just below the CCP. It then becomes part of the CCP until there is a cold boot or until a program overwrites it. The message

```
(XSUB Active)
```

will be displayed over the CP/M prompt after the XSUB has been executed. As long as the message is showing, you do not need to include the XSUB instruction in a submit file.

The submit file may contain a SUBMIT command as the last instruction. For example, consider the following submit file:

```
STAT
SUBMIT TRYAGAIN LETTER
```

where the submit file TRYAGAIN.SUB consists of

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

Any number of submit files may be chained together.

The SUBMIT command functions by creating a file named $$$.SUB to contain the commands. The $$$.SUB file lists the commands with the variables $1, $2, and so on replaced by their respective parameters. After the $$$.SUB file is created, the program warm-boots. On a warm boot the CCP looks to see if there is a file named $$$.SUB. IF $$$.SUB is found, the file is read, and the commands are executed just as if they had been entered from the console. After reading the file, the CCP erases it and warm-boots the system.

Finally, control characters are represented in a submit file as preceded by a caret (^), so that a CONTROL-Z is represented as ^Z. Not all control characters can be used in a submit file, particularly those control characters recognized by the CCP. Thus, CONTROL-C, CONTROL-X, and so forth cannot be used in a submit file. This fact is not stated in the Digital Research manuals or in most CP/M tutorials and can be a source of frustration to the unwary. The best thing to do is to avoid using control characters in submit files. If you need a control character, then run an experiment to see if it is recognized. If it is rejected by the submit file, then the program requiring that control character may have to be altered to use a standard character in the place of the control character.

A dollar sign ($) in a submit file represents a variable. When a $ is needed in a submit file as a standard character, it should be written as $$. For example, assume that the file name MONEY$ is required in the submit file. The file name should be written as MONEY$$ inside that submit file.

You can abort a SUBMIT execution by pressing the DELETE key. On the Apple II, press the CONTROL-@.

8 The DDT Utility

The Dynamic Debugging Tool (DDT) is the CP/M version of the Apple monitor and miniassembler. Unlike the monitor and miniassembler, DDT is not resident in the computer. It must be loaded each time it is used. DDT is intended primarily for people familiar with machine-code programming. There is one useful DDT feature that even the least sophisticated user should learn, however: the file-loading facility.

Loading a Disk File into Memory

A file may be loaded into memory with DDT, and it then may be saved back onto a diskette. This is useful for one-drive file copying and for copying files to other user areas. DDT is invoked with the command

```
DDT
```

The next thing to appear on the console is

```
DDT VERS 2.2
-
```

where the – is the DDT prompt indicating the DDT command mode. To exit DDT, enter a CONTROL-C. A program is loaded into memory by entering at the prompt

```
-Ifilename.ext
-R
```

The I identifies the next file to be read. The R tells DDT to read the file into the TPA (Transient Program Area) starting at location 100H. DDT will read the file and respond with the message

```
NEXT  PC
xxxx xxxx
```

where *xxxx* stands for a hexadecimal number. The hexadecimal number under NEXT is the memory location immediately after the last location used by the file that was just loaded into the memory. The hexadecimal number beneath PC is the value of the DDT program counter. This PC value is of only minor interest to machine-language programmers. Nonprogrammers may safely ignore the PC number.

■ Saving Data to a Disk File

Suppose you want to copy the program PIP.COM to another diskette using only one drive. Enter DDT and type

```
IPIP.COM
R
```

DDT will respond with

```
NEXT  PC
1E00 0100
```

This message tells us that PIP.COM has been loaded into the memory region 100H to 1DFFH. We now exit DDT by entering a CONTROL-C. We have just warm-booted into the CCP (Console Command Processor). The diskette on which we wish to save PIP.COM is loaded into the drive. The diskette is logged in with another CONTROL-C. PIP.COM is still in memory. The previous activities did not overwrite or erase the file image. We then type after the CP/M prompt

```
SAVE 29 PIP.COM
```

PIP.COM is now copied onto the second diskette.

The general command

```
SAVE n filename.ext
```

creates a disk file and copies to that file *n* pages of memory starting at location 100H; *n* must be a decimal number. A memory page is a length of memory 256 decimal bytes, or 100H bytes, long. The value of *n* used for saving PIP.COM was calculated this way: the DDT loading message said that the file ended before the location 1E00H. Now, 1E00H is 1DH pages from 100H, the start of the PIP.COM storage location. The decimal value of 1DH is 29. You can avoid most hexadecimal arithmetic by first noting if the hexadecimal number beneath the NEXT in the DDT loading message ends with 00. If it does, then take the two most significant digits, 1EH in the previous example, and convert the number they form to the decimal equivalent; 1EH becomes 30. This number minus 1 is the number of pages occupied by the file in memory. For the case of our example, 30 minus 1 is 29, the result already obtained. When the number under NEXT does not end in 00, then the number of pages exactly equals the two most significant digits. For example, consider saving the file ABITLONG.TXT. First we type

```
DDT ABITLONG
```

The following message is then printed:

```
DDT VERS 2.2
NEXT  PC
1E80 0100
```

The number under NEXT ends with 80H. This means that the number of pages to be saved is 1EH, or 30 decimal. The file is therefore saved with the command

```
SAVE 30 TOOLONG
```

which creates a copy of the file ABITLONG.

You may use DDT in conjunction with the SAVE command to copy progams, as long as you take care not to run transient command files between the time you exit DDT and the time you use the SAVE command. A transient command file loads in the memory area starting at 100H and will overwrite anything that is there. Resident CCP commands such as USER and CONTROL-C do not disturb the TPA, so you may issue them with impunity before using the SAVE command.

■ More on Loading a File

A file may be loaded from the command line with the command

```
DDT x:filename.ext
```

where *x*: is the optional drive. The standard loading message will appear after the file has been loaded. Incidentally, this brings up a peculiarity of DDT. If the file you wish to load is not in the currently active drive, the command

```
Ix:filename
R
```

will not work. The drive designation in the I command is ignored. When I is used, the file name is placed in a file-control block starting at location 005CH. The locations starting at 005DH contain the file name. DDT sets the byte at 005CH to 00H, which tells the BDOS (Basic Disk Operating

System) to use the currently active drive. This location may be changed with the S command, discussed below, to indicate to the BDOS to look for the file on a specified drive. The permissible values for location 005CH are

01H	drive A:
02H	drive B:
03H	drive C:
⋮	⋮
10H	drive P:

■ Displaying Memory

DDT's most useful feature is its ability to display and alter memory. The memory display can take two forms, a memory dump or a disassembly of the memory into 8080 instructions. You obtain the memory dump by typing

```
Dn
```

where *n* is the hexadecimal value of the location to be viewed. Had you entered DDT with the command

```
DDT PIP.COM
```

and then typed

```
D400
```

the console would display

```
0400 24 44 45 53 54 49 4E 41 54 49 4F 4E 20 49 53 20 $DESTINATION IS 
0410 52 2F 4F 2C 20 44 45 4C 45 54 45 20 28 59 2F 4E R/O, DELETE (Y/N
0420 29 3F 24 2A 2A 4E 4F 54 20 44 45 4C 45 54 45 44 )?$**NOT DELETED
0430 2A 2A 24 24 24 24 24 24 24 4E 4F 54 20 46 4F 55 **$$$$$$$NOT FOU
0440 4E 44 24 43 4F 50 59 49 4E 47 20 2D 24 52 45 51 ND$COPYING -$REQ
0450 55 49 52 45 53 20 43 50 2F 4D 20 32 2E 30 20 4F UIRES CP/M 2.0 O
0460 52 20 4E 45 57 45 52 20 46 4F 52 20 4F 50 45 52 R NEWER FOR OPER
0470 41 54 49 4F 4E 2E 24 55 4E 52 45 43 4F 47 4E 49 ATION.$UNRECOGNI
0480 5A 45 44 20 44 45 53 54 49 4E 41 54 49 4F 4E 24 ZED DESTINATION$
0490 43 41 4E 4E 4F 54 20 57 52 49 54 45 24 49 4E 56 CANNOT WRITE$INV
04A0 41 4C 49 44 20 50 49 50 20 46 4F 52 4D 41 54 24 ALID PIP FORMAT$
04B0 43 41 4E 4E 4F 54 20 52 45 41 44 24 49 4E 56 41 CANNOT READ$INVA
```

The D stands for display; that is to say, the memory contents are sent to the screen. The CONTROL-P and CONTROL-S switches are active (see chapter 3), so the display can be echoed to the listing device. The memory contents are displayed in a line, first in hexadecimal notation and then in the ASCII equivalent. The leading number in each line is the memory location of the first data element. The first line in our example shows the data in location 0400H to be 24H ($ in ASCII). The last data element in the first line is found in location 04FFH and is 20H (a blank space in ASCII). To display the next 192 bytes of memory, we simply enter a D followed by a RETURN. The next 192 bytes of the PIP.COM file are

```
04C0 4C 49 44 20 53 45 50 41 52 41 54 4F 52 24 31 F2 LID SEPARATOR$1.
04D0 1D 01 80 00 C5 1E 80 01 CC 1E C0 18 0A 3A CC 1E .............:..
04E0 D6 00 D6 01 9F 32 A5 1E CD 4C 08 EB 3E 20 CD 84 .....2...L..> ..
04F0 1D D2 FD 04 01 4D 04 CD 39 08 CD 00 00 CD 16 09 .....M..9.......
0500 32 C0 1E 11 00 00 0E 19 CD 05 00 32 FC 1D 31 F2 2..........2..1.
0510 1D CD 40 1A 3A C0 1E 32 C1 1E 21 6F 1F 36 00 2B ..@.:..2..!o.6.+
0520 36 00 2B 36 00 21 A6 1E 36 01 23 36 00 21 F3 1D 6.+6.!..6.#6.!..
0530 36 00 23 36 FE 3A A5 1E 1F D2 47 05 0E 2A CD 1C 6.#6.:....G..*..
0540 08 CD 6F 09 CD 2E 08 21 4E 1F 36 FF 3A CC 1E FE ..o....!N.6.:...
0550 00 C2 5E 05 2A FC 1D 4D CD 5E 08 CD 00 00 21 4B ..^.*..M.^....!K
0560 1E 36 00 21 03 1E 36 00 21 A4 1E 36 00 2B 36 00 .6.!..6.!..6.+6.
0570 01 27 1E CD 20 12 3A A9 1E FE 03 C2 81 05 C3 24 .'.. .:........$
```

When the hexadecimal data has no ASCII equivalent, DDT simply represents that data with a period (.) in the ASCII portion of the display.

■ DDT Commands for the Nonexpert

D—The Memory Display Command: Additional Comments

See the discussion above. The D command defaults to displaying twelve lines of data at a time. A range of data may be displayed with the command

```
Dbegin,end
```

where *begin* is the first memory location to be displayed and *end* is the last memory location to be displayed. All data in the specified range will be sent to the screen. The example just discussed can be displayed with the command

```
D400,570
```

Please notice that leading zeroes need not be entered.

S—The Memory Modification Command

DDT provides the means to alter the contents of memory locations through the set (S) command. The command

```
Sn
```

(where *n* is the memory location in hexadecimal notation) places you in the set mode. As an example, let's alter the memory locations at 100H, 101H, 104H, and 200H. First enter

```
S100
```

Remember that a data entry is accepted by the computer when it is followed by a carriage return, which you get on the Apple by pressing the RETURN key. The S100 command will cause DDT to respond with

```
0100 C3
```

The 0100 is the location of the memory data that may be altered; the C3 is the current contents of that location in hexadecimal notation. The cursor waits on the line for another user command. If we enter a hexadecimal number, then the data at that memory location will change to the new value. Continuing with the example, let's change the location's contents to 00H. The command line will become

```
0100 C3 00
```

After the carriage return, DDT will respond by showing the contents of the next location, that is:

```
0101 CE
```

We are going to change these contents to C9H. The command line then becomes

```
0101 CE C9
```

Entering a carriage return advances us to the next memory location, showing

```
0102 04
```

We do not want to change this location, so we enter a carriage return without entering any new values. The memory location is left unchanged, and the next location is displayed. Our example should look like

```
0102 04
0103 C9
```

This location is to be left unchanged, so we press the RETURN key again. We will change the 104H location to the new value C9H, which will make the entire display become

```
0100 C3 00
0101 CE C9
0102 04
0103 C9
0104 00 C9
0105 00
```

Now the next location to be altered is 200H. We can get there by repeatedly pressing the RETURN key, but this is extremely tedious. Another method is to exit the set-command mode and then reenter it at the new location. To exit the set-command mode, enter a . for the new location data, and press the RETURN key. This will place you in the DDT command mode, with the – prompt showing. We can make the last alteration, placing 00H at location 200H and then exiting the set-command mode again.

```
S200
0200 20 00
0201 20
```

Incidentally, the previous example modified the image of the PIP.COM file in memory but did not affect the PIP.COM file on the disk. The modification was strictly a demonstration and produced no useful changes.

The DDT display and set modes are very useful and should be learned by all serious CP/M users. Later in this book we will use the previous material for making corrections and alterations to the Microsoft BIOS (Basic Input Output System).

■ DDT Commands for the Advanced User

The following material is of interest to those who have some 8080 programming experience. If you lack that experience, you may forgo reading the rest of the chapter.

L—The 8080 Disassembler

DDT will disassemble a range of memory into 8080 mnemonics. The disassembler is invoked through the L, or list, command. The command has the form

```
Ln
```

where *n* is the hexadecimal value of the location at which the disassembly is to begin. If we use PIP.COM as an example, the command

```
L4CE
```

produces the following display:

```
04CE LXI  SP,1DF2
04D1 LXI  B,0800
04D4 PUSH B
04D5 MVI  E,80
04D7 LXI  B,1ECC
04DA CALL 0A18
04DD LDA  1ECC
04E0 SUI  00
04E2 SUI  01
04E4 SBB  A
04E5 STA  1EA5
```

The disassembler lists eleven lines unless requested to do otherwise. Entering L alone will cause the disassembly to begin wherever the program counter is pointing. For the example cited, entering L will start the listing at 4E8H and will show the following:

```
04E8 CALL 084C
04EB XCHG
04EC MVI  A,20
04EE CALL 1D84
04F1 JNC  04FD
04F4 LXI  B,044D
04F7 CALL 0839
04FA CALL 0000
04FD CALL 0916
0500 STA  1EC0
0503 LXI  D,0000
```

The disassembler will list any memory range with the command

```
Lbegin,end
```

where *begin* is where the disassembly is to start and *end* is where the disassembly is to stop. You can get the results just cited with the command

```
L4CE,503
```

The disassembler will display any unrecognizable command as ??=. Take, for example, the listing of PIP.COM starting at 10DH:

```
010D NOP
010E NOP
0110 ??=  28
0111 MOV  C,C
0112 MOV  C,M
0113 MOV  D,B
0114 LDA  4F2F
0117 MOV  D,L
0118 MOV  D,H
0119 LDA  5053
```

The ??= indicates an unknown machine code that is 28H. If you try to list some of your Apple CP/M files or certain areas in the BIOS, you may find that DDT reports some unrecognizable code in areas where there shouldn't be any. The reason for this is that the Apple BIOS and some of the Apple CP/M utilities are written in Z-80 code, which supports the same machine codes as the 8080 but also has some additional instructions. It is the additional instruction set that produces the ??= in the disassembly. You will need a disassembler such as that found in Digital Research's ZSID utility to create the proper listing for Z-80 code.

A—The 8080 Assembler

DDT also includes an 8080 assembler that accepts 8080 mnemonic instructions as arguments. The assembler requires the use of absolute locations; that is, no labels are allowed. This assembler is similar to the Apple miniassembler found in the Integer Basic ROM. You enter the assembler by typing

```
An
```

where *n* is the hexadecimal value of the location at which the code is to be placed. The following is an example of usage.

```
A100
```

starts the assembly at 100H. The code is then entered:

```
0100 LDA 3
0102 STA 0AFFF
0105 RET
0106 .
```

The memory locations at the left are automatically printed by DDT. The code is entered by the programmer. You stop the assembler by entering a . for the last instruction.

M—The Data Move Command

There is a memory move command in DDT. The command is

```
Mbegin,end,destination
```

where *begin* is the first address of the memory block, *end* is the last address of the memory block, and *destination* is the address of the new location to which the block is to be sent. The move command actually copies the memory block to the new location, leaving the original block intact (provided that the original block does not overlap the destination block). Let's copy the memory lying between 4FFFH and 60A3H to 1000H. The command

```
M4FFF,60A3,1000
```

will accomplish the task.

F—The Memory Fill Command

A memory fill command is available. Typing

```
Fbegin,end,data
```

places the *data* byte into each memory location starting at *begin* and stopping at *end*. For example, the command

```
F1000,1500,00
```

will place the byte 00H into each location of the memory range beginning at 1000H and ending at 1500H.

G—The Run Program Command

A machine language program can be run from within DDT. The go, or G, command is used for that purpose. The go command has six forms:

```
G
Gstart
Gstart,break
Gstart,break1,break2
G,break
G,break1,break2
```

The G command alone will begin program execution starting at the current value of the program counter. Gstart starts program execution with the code located at *start*. For example, the command

```
G100
```

starts execution for the program beginning at memory location 100H. The Gstart,break begins execution at the location *start* but will stop execution and return to the DDT command mode if the program execution takes the program counter to location *break*. For example, the command

```
G100,230
```

starts execution at location 100H and stops execution when program execution reaches location 230H. The form Gstart,break1,break2 will stop the program if the program counter points to either *break1* or *break2*. The two forms G,break and G,break1,break2 start execution at the current program counter location and stop execution at either *break*, *break1*, or *break2*. When a breakpoint is encountered, DDT prints

```
*n
```

where *n* is the address at which execution has halted. Then DDT returns to the DDT command mode. Programs run with the G command without any breakpoints will usually be unable to return control to DDT unless there is an RST 7 to terminate the program.

T—The Trace Program Execution Command

There are two forms of a command to trace program execution. One form is

```
Tn
```

where *n* is the number (in hexadecimal notation) of program steps to be traced. If *n* is omitted, then it is assumed to be 1 by DDT. The trace starts at the current program counter location. As an example, let's trace PIP.COM starting at 100H. We will look at five lines of execution with the command

```
T5
```

The following is displayed:

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 JMP  04CE
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=04CE LXI  SP,1DF2
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=1DF2 P=04D1 LXI  B,0080
C0Z0M0E0I0 A=00 B=0080 D=0000 H=0000 S=1DF2 P=04D4 PUSH B
C0Z0M0E0I0 A=00 B=0080 D=0000 H=0000 S=1DF2 P=04D5 MVI  E,80*04D7
```

The explanation of the display is as follows:

Cn	the bit in the 8080 carry flag with a value of *n*
Zn	the bit in the 8080 zero flag with a value of *n*
Mn	the bit in the 8080 minus flag with a value of *n*
En	the bit in the 8080 parity flag with a value of *n*
In	the bit in the 8080 interdigit flag with a value of *n*
A	the value in the 8080 accumulator
B	the value in the 8080 BC registers
D	the value in the 8080 DE registers
H	the value in the 8080 HL registers
S	the location of the stack pointer
P	the location of the program counter
instruction	the 8080 code at the program counter location
*n	the next address to be executed

The trace can be terminated from the keyboard by the pressing of the DELETE key, or for the Apple II the CONTROL-@.

U—Another Trace Program Execution Command

The untrace command is similar to the trace command, but it will display the flags and registers for the first step only. The command is identical in format to the trace command but is written

```
Un
```

If the previous example were done with a

```
U5
```

command, then the display would be

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 JMP  04CE*04D7
```

X—The 8080 Register Examination Command

The 8080 registers and flags can be examined and altered with the X command. To examine the current condition of the 8080, enter

```
X
```

The display will be identical to that produced by the command

```
T
```

The registers or flags may be changed with the command

```
Xr
```

where *r* is one of the following:

C	the carry flag
Z	the zero flag
M	the minus flag
I	the interdigit flag
A	the accumulator
B	the BC register
D	the DE register
H	the HL register
S	the stack pointer
P	the program counter

As an example, we will change the program counter from its current value to 100H. First enter

```
XP
```

DDT responds with

```
P=04E1
```

The current value of the program counter is displayed. If we didn't want to change it, we would need only to press RETURN, and we would be put into the DDT command mode with no changes made. To place 100H in the program counter, we type 100, and the display looks like

```
P=04E1 100
```

Entering RETURN places us back into the DDT command mode, with the program counter now pointing to 100H. The identical procedure is used to change any of the flags or registers.

R—The File Read Command: Additional Comments

A note on the R command: a file previously identified with the I instruction can be read into the TPA by the command

```
R
```

There is a second form of this command. It is

```
Rn
```

where *n* is a hexadecimal number indicating an offset value for the read command. For example, the command

```
R100
```

will read the identified file into the TPA offset by 100H; that is, the file will be read into the memory area beginning at 200H. This procedure is useful for modifying files with overlays.

Finally, let's discuss what happens when DDT is invoked. The transient command

```
DDT
```

loads DDT into the TPA and starts execution. DDT moves itself to just below the BDOS, overwriting the CCP, and then goes into the command mode. The transient command

```
DDT filename.ext
```

does all of this, too, except that before going into the command mode, it loads the file in the command line into the TPA.

9 The ED Program

Your CP/M master diskette comes with a text editor called ED.COM. Digital Research prefers to call ED a *context editor*. ED is technically a *line editor* with character features. A *line editor* is one in which the text is divided into *lines*. A *line* is defined as a string of characters ending with a carriage return, the ASCII character 0DH.* ED assigns a line number to each line for internal use. The line number is not included in the text file. All text editing is done with reference to the internal line numbers. A file may have a maximum of 65,535 lines.

When Digital Research introduced CP/M, it included ED to give new users the ability to create assembly-language source files. As CP/M became more popular, other text editors evolved that were more powerful and much easier to use. ED may be considered an artifact from the early days of CP/M whose use should be avoided by any person with extensive need of a text editor. ED's salient feature is that it can quickly create a short text file, say for use as a submit file.

Loading or Creating an ED File

ED is entered by typing after the CP/M prompt

```
ED filename.ext
```

*CP/M and Apple DOS treat carriage returns differently. CP/M performs a carriage return in the strict sense; that is, it places the cursor or printer head at the leftmost column and does not advance to the next line. Apple DOS, when it sees a carriage return, always adds a line feed, the ASCII character 0AH, which advances the cursor or printer head to the next line. As a result, there is a major difference between text files created under Apple DOS and those created under CP/M: all CP/M text files contain carriage return–line feed combinations, but Apple DOS text files have no line feeds following carriage returns. This difference must be corrected when files are transferred between CP/M and Apple DOS. If it is left uncorrected, Apple DOS files will print entirely on one line in CP/M, and CP/M files will have double-spaced lines in Apple DOS.

There is a second major difference between CP/M and Apple DOS. Apple DOS has the most significant bit set in each byte representing an ASCII character; CP/M generally has the most significant ASCII bit cleared. If you don't know what this means, don't worry; the utilities that transfer files between CP/M and Apple DOS will correct for this difference automatically.

A file must be specified in the command line. ED looks in the directory to see if the file already exists. If no file is found, a new file is created, and ED prints

```
NEW FILE
```

ED then goes into the command mode, which is identified by the * prompt.

ED can create or modify text only if that text is in the computer's memory. ED sets aside a portion of memory called the *file buffer* for editing purposes. A text file that is to be edited must be read into the file buffer first; then the changes are made. After the changes are made, the modified text can then be transferred from the memory to a disk file. Of course, if a new file is being created, the reading of a file into the file buffer is omitted. In this instance, the new text is placed in the file buffer by the user, and the text can then be transferred to a disk file. The manipulation of text in the file buffer will be covered later.

The size of the file buffer is a function of available memory. There is a problem if the text file being modified or created is larger than the file buffer. The way ED take cares of this is to create a temporary file whose name is the same as the one in the ED command line but has a .$$$ extension. For instance, if you enter ED with the command

```
ED LETTER.TXT
```

ED will create the temporary disk file LETTER.$$$. When the file buffer is at or near capacity, ED will inform the user of the condition. The user can then write the file buffer to the temporary file with the W command (discussed below). The transferred text is deleted from the file buffer, which leaves room for new text to be placed in the file buffer.

■ Editing Text

ED is a line editor. The maximum number of lines permitted in any text is 65,535. Consider this example of editing a file with two thousand lines. The file's name is EXAMPLE.TXT. We enter ED by typing after the CP/M prompt

```
ED EXAMPLE.TXT
```

Since EXAMPLE.TXT is an existing file, the next thing to appear is the ED prompt, *. At this time the file buffer is empty. We will fill the file buffer with the append command, A. The file is too large to fit in the file buffer, so we will read only a portion of the file into the buffer. Let's edit one hundred lines at a time. The command

```
100A
```

typed after the ED prompt reads the first hundred lines from the file EXAMPLE.TXT into the file buffer. After the text has been modified, we wish to edit the following hundred lines in the file EXAMPLE.TXT. We get into the ED command mode and type

```
100W
```

after the prompt. The file buffer is written to the file EXAMPLE.$$$. There is now room for us to enter more data into the file buffer.
The command

```
100A
```

will read the second hundred lines from the file EXAMPLE.TXT into the file buffer. After the text is altered, it can be added to the file EXAMPLE.$$$ with the command

```
100W
```

The second hundred lines of text written to EXAMPLE.$$$ is sequential to the first hundred lines of text, so the order of the file data remains unchanged. The sequence of reading into and clearing the file buffer is continued until the file EXAMPLE.TXT is completely transferred to the temporary file EXAMPLE.$$$.

When ED is exited with the E command (described below), the file EXAMPLE.TXT is renamed EXAMPLE.BAK and the file EXAMPLE.$$$ is renamed EXAMPLE.TXT. In this example, had we transferred only the first two hundred lines to the temporary file and then exited ED, we would have the file EXAMPLE.TXT containing only the two hundred transferred lines. In order that a file be properly edited, a complete transfer must be made.

Had we chosen for an example a newly created file, we would not need to bother with the append command. We could flush the file buffer as required by using the W command. Incidentally, ED informs the user when the file buffer nears capacity. The temporary file is expanded as previously described, and exiting with the E command renames the temporary file.

There are more commands available for flushing the buffer and exiting ED than those used in the above examples. These commands are discussed later in the chapter.

Before considering how to enter text into the buffer, you will need to know about ED's imaginary character pointer (CP). The CP is a device that ED uses to indicate the location in the file buffer where data is to be inserted. The CP is what makes ED so difficult to use. There is no visual indication of where the CP is in the text, and all relocation of the CP requires the ability to see the text in your imagination in order to know where the CP has gone.

■ An Example

Let's create a text file using ED. First enter

```
ED TEMP.DOC
```

ED responds with

```
NEW FILE
*
```

The NEW FILE means that there is no file on the diskette named TEMP.DOC, so ED just created it. The * is the prompt indicating the ED command mode. Type

```
I
```

(for *insert*) to enter the edit mode. The edit mode will place line numbers before each line to be entered. The line numbers are not stored in the diskette file. Terminating each line with a carriage return (RETURN key), we enter

```
1:Peter Piper picked
2: a peck of pickled
3: peppers.
4:
```

Press CONTROL-Z to exit the edit mode. Pressing the RETURN key without entering data will simply place empty lines in the text. To exit ED, type

```
E
```

If you look at the directory, you will find two files, one named TEMP.DOC and another named TEMP.BAK. ED always saves the original file and names it the same but with the BAK extension (BAK stands for *backup*). The newly edited file is saved under the original name. When a file with the BAK extension already exists, it is erased, and the cycle is repeated. For the case of a new file, the backup file is an empty file.

Let's add some more text to TEMP.DOC. Type

```
ED TEMP.DOC
```

This time NEW FILE doesn't appear because TEMP.DOC was found on the diskette. We must now read TEMP.DOC into the file buffer with the command

```
#A
```

which reads all the lines in the file into the file buffer. The CP is now at the end of the file. By typing I, we can append the text. Then the following additional text is typed:

```
4:A peck of pickled
5: peppers Peter
6: Piper picked.
7:
```

We get out of the edit mode at line 7 by entering a CONTROL-Z. We are now in the command mode.

To see what the file contains, type T (which stands for *type*). The T command will print to the screen the file starting from the CP position. In our example, the CP is at the end of the file, so we will see nothing printed. The CP can be moved to the beginning of the file by typing

```
B
```

Typing T will print the first line and move the CP to the beginning of line 2. Had we typed

```
#T
```

all the lines in the buffer would have been printed. Let's change the *peppers* in line 5 to *pigs' feet*. First we must move the CP to the beginning of line 5 by typing

```
5:
```

Then we put the CP at the beginning of *peppers* by typing

```
1C
```

which moves the CP one position to the right from the line's beginning. Then we delete *peppers* by typing

```
7D
```

which deletes the next seven characters to the right, starting at the CP. The CP is now in the position where the first *p* in *peppers* used to be. We insert the new character string by typing

```
I
```

which will start the insertion mode at the CP, and then typing *pigs' feet*. It is possible to combine steps. We could have gotten to the point of the I instruction by typing

```
5:1C7D
```

We exit the insert mode by typing CONTROL-Z. We can then save the altered file by typing

```
E
```

just as before.

There is an easier way to replace the *peppers* in the above file. The string *peppers* could have been replaced with the string *pigs' feet* on line 5 with the command

```
5:1Speppers^Zpigs' feet^Z
```

where ˆZ is a CONTROL-Z. This search and replace command is just one of the edit commands discussed below.

■ ED Command-Mode Instructions

The ED command-mode instructions are as follows:

nA Reads the host file into the file buffer. The A stands for *append*. The *n* tells how many lines to read into the buffer. A # may be used instead of a number to read as much from the host file as the file buffer can hold. Using the # usually results in reading the entire host file into the file buffer. The command

```
0A
```

will read the file into the file buffer until the buffer is at least half full.

B Moves the CP to the beginning of the file buffer.

–B Moves the CP to the end of the file buffer.

nC Moves the CP *n* character positions forward in the file buffer. (*Forward* means toward the last character in the buffer.)

–nC Moves the CP *n* character positions backward in the file buffer.

nD Deletes *n* characters, starting after the CP and going forward.

–nD Deletes *n* characters, starting at the CP and going backward.

E Ends the edit mode. This command saves the file buffer to the diskette using the source file name, and returns control to the CCP.

nFstring^Z Finds the *n*th occurrence of the designated character string. The character string immediately follows F and must be terminated by a CONTROL-Z, which is denoted by the symbol ^Z. The command instructs ED to find the *n*th occurrence of *string* and then place the CP at the next character in the file.

H Has the same effect as an E command followed by reentry to ED with the updated source file. The H command should be used periodically to save the file buffer to the disk as insurance against losing work in the case of equipment failure or operator blunder. The H stands for the *head* (that is, the beginning) of a new source file. The CP is placed at the beginning of the file buffer. It is advisable to issue a

```
#A
```

after the H command to fill the file buffer with the saved text since the H command flushes the buffer. It is also advisable to type

```
-B
```

so that the editing will begin at the end of the file. If these steps are not taken, the saved file will be appended to the bottom of the new file when the H command is reused or the edit mode is terminated, and a strangely jumbled file will be created.

I Inserts; that is, enters the edit mode at the position of the CP. You terminate the insert mode by typing CONTROL-Z.

Istring Inserts the character string following I into the file buffer after the CP. ED will place a carriage return–line feed combination at the end of the string, which will make the insertion a line. The command returns to the edit command mode.

Istring^Z Inserts the character string after the I into the file buffer after the CP. The string has no carriage return–line feed termination, so the insertion does not create a new line. The command returns to the edit command mode.

nJstring1^Zstring2^Zstring3^Z Juxtaposes. The command searches for *string1* and places the CP after it. *String2* is then inserted at the CP. The CP is advanced beyond *string2*, and all characters are deleted from the CP to the position just before the first occurrence of *string3*. If *string3* is not found, no deletions are made. The process may be repeated *n* times. As always, the ^Z stands for the CONTROL-Z.

nK Deletes *n* lines forward from the current position of the CP. If *n* is omitted, then one line is assumed. Be certain that the CP is at the beginning of the first line to be deleted; otherwise the characters before the CP will not be erased.

–nK Acts much as does nK but deletes *n* lines *backward* from the CP.

nL Moves the CP to the beginning of the line *n* lines forward. The command 0L places the CP at the beginning of the current line.

–nL Moves the CP to the beginning of the line *n* lines backward.

nMcommand sequence^Z Repeats a given sequence of commands *n* times. This is the *macro* command. If *n* is omitted or equal to 0 or 1, the command is repeated until an error condition arises. An example of use is

```
MFHELLO^ZI OUT THERE^Z0LT^Z
```

which will find all occurrences of HELLO, insert the character string OUT THERE (preceded by a space) after HELLO, move the CP to the beginning of the line containing the insertion, and print that line.

nNstring^Z Mimics the actions of the F command, with one exception. This command looks at the entire file, not just the portion in the file buffer. If only a portion of the file is in the buffer, the file remaining on the diskette is also searched.

O Returns to the original file. The file buffer is flushed, and the original file (the file with the BAK extension) is used. This is equivalent to reentering ED. The append command will be needed to fill the file buffer. You will be asked to verify if you want to flush the buffer.

nP Prints *n* pages of text to the screen. (P stands for *page*.) A page is 23 lines of text. The CP is first advanced to the start of the next page, which is 23 lines away. The *n* pages are then printed from the CP. The command

```
0P
```

leaves the CP on the current page and prints that page.

–nP Resembles the nP command, but moves the CP to the previous page and prints the previous *n* pages in ascending order.

Q Quits: terminates ED, does not save the file buffer, erases the temporary file, removes the BAK extension from the original source file, and gives back its original extension. ED will verify the Q command before acting.

R Acts as a block move command that inserts into the file buffer, after the CP, the file X$$$$$$$.LIB from the diskette. This command is used in conjunction with the X command. It is unlikely that you will need an R command unless you are a serious machine-level programmer. If you are a serious programmer, it is also unlikely that you will be using ED.

Rfilename Resembles the R command, but allows any file with the LIB extension to be read into the file buffer. The LIB extension is understood and is not required in the file name.

nSstring1^Zstring2^Z Searches and replaces. This command is probably the most useful of the character-string commands. *String1* is searched for in the file buffer and is replaced by *string2*. The replacement will be made for the first *n* occurrences after the CP.

nT Displays *n* lines after the CP. (T stands for *type*.) If the CP is not at the beginning of a line, the command

```
0T
```

displays the beginning of the line to the CP.

–nT Resembles the nT command, but displays the *n* lines *before* the CP.

U Translates to uppercase all characters entering the file buffer from either the console or a diskette. This command is useful to programmers who have assemblers that can recognize only uppercase letters.

–U Translates all characters to lowercase.

V Verifies line numbers by displaying them next to each line. ED defaults to this mode.

–V Removes the line-number display.

OV Prints the space remaining in the file buffer and the total available space, in the format

```
Free Space/Available Space
```

The numbers printed are in decimal notation. The available space depends on which memory configuration for CP/M is being used. A 56K version will have an available space of 33,719 characters.

nW Writes the first *n* lines of the file buffer to the temporary file. ED flushes those lines from the buffer, but the line-numbering sequence is left unchanged. The command

```
0W
```

writes the buffer to the temporary file until the buffer is at least half empty.

nX Acts as a block transfer command that writes *n* lines after the CP to the file X$$$$$$.LIB. If X$$$$$$.LIB exists, the *n* lines are appended to the file. This command is intended for machine-language programmers who need to create library files.

0X Deletes the contents of the file X$$$$$$.LIB.

nZ Creates a delay before executing the next command; the larger *n*, the longer the delay. This *sleep* command is a holdover from the early days of slow machine responses and shouldn't be necessary for the Apple user.

n Offers a convenient way of commanding

```
nLT
```

–n Acts the same as

```
-nLT
```

n: Places the CP at the beginning of line *n*.

:n When used in conjunction with another command, tells ED to execute the command from the current line until reaching line *n*. For example:

```
:25T
```

prints the lines from the current CP position to line 25. Commands can be concatenated; for example:

```
10::25T
```

will print lines 10 through 25.

When *n* is omitted in those commands using it, a value of 1 is assumed. The # may be used as a value for *n* when all occurrences are to be used.

The command mode of ED has console input editing commands similar to those of the CCP (Console Command Processor). The control characters that perform special functions are listed below.

CONTROL-C Does a warm boot back to the CCP, abandoning the work without updating. This command should be avoided.

CONTROL-E Prints a carriage return–line feed (ASCII 0DH and 0AH) pair, but does not enter the command.

CONTROL-H Acts as does the backspace or left arrow (←) key on the Apple. It performs the delete character left, as does its counterpart in the CCP.

CONTROL-I Acts as a tab, which is of little use to most users.

CONTROL-L Provides another way of placing a carriage return–line feed into the command line. Its primary value is that it can be used to place the 0DH-0AH pair in a character string in the commands using character strings (the I, F, J, and S commands, for instance).

CONTROL-X Deletes the current command line.

CONTROL-Z Is used to terminate strings in the commands that use character strings.

DELETE Performs the same function as in the CCP and produces the same confusing display.

■ ED Insert-Mode Commands

There are some commands active in the insert mode. They are

CONTROL-H Acts as does the backspace or left arrow (←) key on the Apple. This command deletes the character to the immediate left of the cursor and moves the CP left one character position.

CONTROL-I Inserts a tab character (ASCII 09H) into the file buffer. A CONTROL-U or right arrow (→) key will also insert the 09H. On the Apple IIe the TAB key will also insert the 09H. The CP is moved to after the tab.

CONTROL-J Places a simple line feed (ASCII 0AH) in the file buffer. No carriage return (ASCII 0DH) is included.

CONTROL-M Performs a carriage return, just as does pressing the Apple RETURN key. A carriage return–line feed combination is inserted into the file, and the CP is advanced to the beginning of the next line. CONTROL-L is equivalent to CONTROL-M.

CONTROL-R Retypes the line. This command is not very useful.

CONTROL-X Deletes the entire line, which may be rewritten. This is identical to the CCP CONTROL-X. The CP is placed at the beginning of the line.

DELETE Produces the same result as with the CCP. This command should be avoided because of the confusing display it produces.

■ Error Messages

The ED error messages are printed with the last character read before the error followed by one of the four symbols whose meaning is given below.

? The command given was not understood.

> The file buffer is full. To continue, flush the buffer with a W command, or delete some lines. Also, the > may indicate that a string is too long in the ED command.

The command cannot be performed the number of times requested.

O The LIB-extension file cannot be read in the R command.

An error detected in reading a diskette produces the message

```
PERM ERR DISK x
```

where *x* is the currently active drive. You can get rid of the message by pressing any key on the console. The file buffer should then be checked for errors. Otherwise, ED can be terminated with the Q command and the backup file renamed and used as the host file.

10 The BIOS

The Basic Input Output System, or BIOS, is the interface between a computer and CP/M. Any discussion of a BIOS will have to contain references to machine-level programming. For the Apple, the BIOS is a mixture of 6502 and Z-80 language instructions. This chapter will assume that the reader has at least a rudimentary knowledge of programming the 6502 and Z-80 microprocessors and that he or she is also familiar with the Apple hardware.

The BIOS is a strict function of the variety of Z-80 card installed in the Apple. The BIOSs to be discussed in this chapter will be those used in the Microsoft standard SoftCard and Premium SoftCard IIe (PS IIe). The description of the standard SoftCard's hardware is given here, while the PS IIe's hardware description is deferred to chapter 14.

CP/M is structured so that it can be made to run on almost any microcomputer that is 8080 or Z-80 microprocessor–based. The BDOS (Basic Disk Operating System) will do all hardware interfacing if the user provides seventeen routines that perform functions prescribed by Digital Research.

BIOS Functions

BOOT

This is the cold boot routine, which loads into memory the CCP (Console Command Processor), BDOS, and BIOS from the diskette. The IOBYTE is initialized, the default drive is set to A:, and other system parameters are initialized. The sign-on messages, which will include the copyright notices, must be shown on cold boot. The cold boot will then fall into the warm boot.

WBOOT

The warm boot loads the CCP into memory, does some more initializing, and jumps to the CCP. The Z-80 C register is set to the drive currently selected.

CONST

CONST is the console status routine. The Z-80 A register must contain 0FFH if a character is waiting to be read or 00H if no character is waiting.

CONIN

CONIN is the console input routine. The console character is read into the A register. The input is an ASCII character with the parity bit (the eighth bit) set to 0. If CONIN is called and there is no character ready, it must wait until a character is received. The Apple's console is the keyboard and the video monitor. The Apple's keyboard input sets the parity bit to 1; therefore, CONIN must mask this bit to 0.

CONOUT

With this routine, the character in the Z-80 C register is sent to the console. The output is an ASCII character with its parity bit set to 0. The Apple monitor requires that noninverse characters have the parity bit set to 1. CONOUT for the Apple must set the parity bit before printing the character in C.

LIST

This routine sends the ASCII character in the C register to the current LST: device. The character has its parity bit set to 0.

PUNCH

This routine sends the ASCII character in the C register to the current PUN: device. The character has its parity bit set to 0.

READER

This routine reads the ASCII character from the current RDR: device into the A register. The input character must have its parity bit set to 0.

HOME

This routine homes the disk read/write head to track 0 of the currently active drive; that is, it sets the current track to track 0.

SELDSK

This routine selects the disk drive. The C register contains the drive to be selected. The drive values permitted in register C are

00	drive A:
01	drive B:
02	drive C:
⋮	⋮
10	drive P:

Upon returning from a call to SELDSK, the Z-80 HL registers must contain the address of the Disk Parameter Header, which is discussed below in this chapter. If a nonexistent drive is selected, the HL registers should contain 0000H to indicate an error.

SETTRK

With this routine, the track on the current diskette is selected. The Z-80 BC registers contain the track number. The Microsoft BIOS ignores the B register since there are only thirty-five tracks on a standard Apple diskette. The C register will range from 00 through 22H.

SETSEC

With this routine, the physical sector on the current diskette is selected. The BC registers contain the sector number. The Apple has sixteen sectors per track. Microsoft uses only the C register, which will range in value from 00H through 0FH. SETSEC in the Microsoft BIOS actually accepts the logical sector. The translation to physical sectors is done in the BIOS RWTS (Read Write Track Sector) routine. The fact that logical sectors are used in SETSEC causes no difficulty, as we shall see when we discuss the SECTRAN routine.

SETDMA

With this routine, the Direct Memory Access (DMA) location is set. The DMA is the memory region where the data read from the selected diskette sector is stored. The DMA area is used also to store the data to be written to a diskette sector. The DMA address is placed in the BC registers before SETDMA is called; for example, if BC is 0080H, then the read/write buffer will start at 0080H and end at 00FFH. Please recall that CP/M logical sectors are 128 (80H) bytes in length.

READ

This is the routine to read the sector selected by SETTRK and SETSEC on the current drive into the DMA. Upon returning from a read, the A register should contain 00H if no error was encountered or 01H if an error was encountered. Incidentally, the disk need not actually be read each time READ is called. The CP/M logical sector is 128 bytes, but a physical sector on an Apple diskette contains 256 bytes. A call to READ on an Apple will actually read two CP/M logical sectors. If the next CP/M logical sector to be read is from the same physical sector as the previously read CP/M logical sector, no disk access is required. The Apple BIOS uses a 256-byte buffer to read the Apple physical sectors. The BIOS moves either the first half or the second half of this buffer to the DMA area, depending on which CP/M logical sector was requested. If the buffer already contains the CP/M logical sector requested to be read, the corresponding half of the buffer is moved to the DMA area, and the drive is not activated. This procedure increases the speed of disk-read commands.

WRITE

This is the routine to write the contents held in the DMA to the currently selected sector on the currently selected drive. The error codes pertaining to the READ command apply. Writing a sector is slightly complicated if the diskette's physical sectors are more than 128 bytes, as with the Apple. Since the Apple's physical sectors are 256 bytes long, a 256-byte buffer is used by the BIOS for writing physical sectors to the diskette. Since CP/M logical sectors are only 128 bytes, writing a CP/M logical sector to the diskette requires that the write buffer be filled with the remaining 128 bytes to go into the corresponding physical sector. This leads to the BIOS's acting on a series of conditions before actually writing data to a diskette. The BIOS must act in one of three ways:

1 Writing to a physical sector that has information on it (this sector is in an allocated block) requires the BIOS to first read the sector's entire 256 bytes into the buffer. The DMA area is then copied to the appropriate half of the buffer. If the next disk command is a read from this physical sector, don't access the drive. If the next disk command is a write to one of the two CP/M logical sectors contained in the buffer, don't access the drive. If the next disk command is either to read or to write a CP/M logical sector not contained in the buffer, then first write (flush) the buffer to the physical sector, and then proceed with the next command.

2 Writing to a physical sector that has not been used (this sector is in an unallocated block) requires the same activity as with writing to an allocated sector, except that a preread of the physical sector is not required.

3 Writing to a directory sector is an operation that assumes that the write buffer is properly constructed and that the physical sector is written immediately to the diskette.

The C register is used in the write command to indicate which mode is to be applied. The BDOS will set the C register to one of the following:

OOH, if writing to a sector in an allocated block
O1H, if writing to a directory sector
O2H, if writing to an unallocated block

LISTST

This is the listing-device status routine. The A register contains 00H if the device is not ready or 0FFH if the device is ready. The Microsoft BIOS assumes that the driver routines for the LST: device check the status. The LISTST is written to always return 00H in the A register.

SECTRAN

This is the routine to translate logiçal sector numbers to physical sector numbers. The routine provides a means to include a sector-skewing factor to speed up disk access time. Placing sectors in physically sequential order on a track is not efficient. Consider the case in which we would wish to read from sector 1 and sector 2 on a given track. Reading a sector requires first getting the data from the diskette, then decoding the data, and finally placing it in the DMA area. By the time all the procedures are completed, the following adjacent sector has already passed by the drive read/write head. If sector 2 is located, say six sectors from sector 1, then the data from sector 1 can be processed before the read/write head reaches sector 2. Such a scheme permits reading more than one sector in a single diskette revolution. Such an interleaving of sectors is often called *sector skewing*. The BC registers receive the logical sector number, and the DE registers receive the address of the sector-translate table. The sector-translate table is held in the BIOS memory area and contains information about the skewing relationship of the CP/M logical sectors to the CP/M physical sectors. Upon returning from SECTRAN, the HL registers contain the physical sector number to be used by SETSEC. Since the Apple BIOS has the sector skewing built into the RWTS, SECTRAN simply copies the BC registers to the HL registers.

■ The Microsoft Vector Jump Table

The seventeen routines described above are accessed by the BDOS through a *vector jump table*. The vector jump table is usually located at the start of the BIOS. The table of vector jump addresses for the standard SoftCard's 56K CP/M version 2.20B, the standard SoftCard's 60K CP/M version 2.23, and the PS IIe's 64K CP/M version 2.26 are given in table 10.1.

Table 10.1 is required by Digital Research. Z-80 mnemonics are used because Microsoft uses Z-80 code in much of its BIOS. Microsoft makes

Table 10.1 ■ Microsoft BIOS Vector Jumps

Version	*Address*	*Instruction*	*Description*
56K 60K, 64K	DA00H FA00H	JP BOOT	Cold boot routine
56K 60K, 64K	DA03H FA03H	JP WBOOT	Warm boot routine
56K 60K, 64K	DA06H FA06H	JP CONST	Console status routine
56K 60K, 64K	DA09H FA09H	JP CONIN	Console input routine
56K 60K, 64K	DA0CH FA0CH	JP CONOUT	Console output routine
56K 60K, 64K	DA0FH FA0FH	JP LIST	Listing device output routine
56K 60K, 64K	DA12H FA12H	JP PUNCH	Punch device output routine
56K 60K, 64K	DA15H FA15H	JP READER	Reader device input routine
56K 60K, 64K	DA18H FA18H	JP HOME	Home head on selected drive routine
56K 60K, 64K	DA1BH FA1BH	JP SELDSK	Select drive routine
56K 60K, 64K	DA1EH FA1EH	JP SETTRK	Select track routine
56K 60K, 64K	DA21H FA21H	JP SETSEC	Select sector routine
56K 60K, 64K	DA24H FA24H	JP SETDMA	Routine to set DMA address
56K 60K, 64K	DA27H FA27H	JP READ	Routine to read selected sector
56K 60K, 64K	DA2AH FA2AH	JP WRITE	Routine to write to selected sector
56K 60K, 64K	DA2DH FA2DH	JP LISTST	Routine to get listing device status
56K 60K, 64K	DA30H FA30H	JP SECTRAN	Sector translation routine

two changes to the vector jump table. The jump to the LISTST routine is replaced with the instructions

```
XOR A    ;Zeroes the A register and makes the LST: device
         ;never ready.
RET
```

The jump to the SECTRAN routine is replaced with

```
LD H,B   ;Places the logical sector in the HL registers and lets
LD L,C   ;the SETSEC routine perform the sector skewing.
RET
```

■ Other CP/M Requirements

CP/M requires that the user provide, in addition to the vector jump table, disk parameter tables within the BIOS area. The tables are used by the BDOS to identify the disk formats used in the system. It is therefore possible to run CP/M with a variety of drives attached to a single computer. The disk parameter tables are of six types: Disk Parameter Headers (DPH), Sector Skewing Tables (XLT), Directory Buffer (DIRBUR), Disk Parameter Blocks (DPB), Check Sum Vectors (CSV), and Allocation Vectors (ALV).

DPH

The DPH tables are usually located just after the BIOS jump vectors. Each drive requires a DPH. The Microsoft version 2.20B permits up to six drives, while the versions 2.23 and 2.26 permit a maximum of four drives. Each DPH consists of 16 bytes and is arranged in the following order.

DPH = XLT, 0000H, 0000H, 0000H, DIRBUF, DPB, CSV, ALV

The DPH for drive A:, for Microsoft's 60K version 2.23, is

DPHA: = 0000H, 0000H, 0000H, 0000H, FDFEH, 73FAH,
C5FFH, 7DFFH

Please notice that the addresses are all given in the standard format of low byte, high byte, so that, for example, the address of the DPB is actually FA73H. The DPHs for the remaining three drives are listed in the BIOS sequentially as follows:

```
DPHB: = 0000H, 0000H, 0000H, 0000H, FDFEH, 73FAH,
        D1FFH, 8FFFH
DPHC: = 0000H, 0000H, 0000H, 0000H, FDFEH, 73FAH,
        DDFFH, A1FFH
DPHD: = 0000H, 0000H, 0000H, 0000H, FDFEH, 73FAH,
        E9FFH, B3FFH
```

XLT

XLT is the address of the Sector Skewing Table used by the drive assigned to that DPH. If no skew is required, as with the Apple, then

XLT = 0000H

The next 6 bytes are used by the BDOS as a scratch pad. The user doesn't need to be concerned with what their values become.

DIRBUF

DIRBUF is the address of a 128-byte memory location reserved in the BIOS for BDOS directory operations. The DPH for each drive uses the same DIRBUF address.

DPB

DPB is the address of the Disk Parameter Block. The Disk Parameter Block contains the drive characteristics. The Apple has one Disk Parameter Block since all the drives are formatted identically. If the system contained drives with different formats, there would be a Disk Parameter Block for each format.

The DPB is a 15-byte table of values identifying a drive's characteristics. The table contains, in the following order:

SPT A 2-byte area giving the number of sectors per track. The Apple has thirty-two CP/M sectors per track and SPT = 2000H. Remember that all 2-byte values used by CP/M are listed in the order low byte, high byte.

BSH A 1-byte value called the *block shift factor*. This number is used by the BDOS to determine the block size. The BSH is the logarithm of the number of sectors per block. The Apple block is 1,024 bytes, or eight CP/M logical sectors. Then, BSH = 3.

BLM A 1-byte value called the *block mask*. The block mask is the number of sectors per block, minus 1. The BLM for the Apple is 8 - 1, or BLM = 7.

EXM A 1-byte value called the *extent mask*. The extent mask is used by the BDOS to determine which disk format is being used. An extent is 1,024 bytes, or eight CP/M logical sectors. EXM is the maximum number of extents less one that may be used in a File Control Block (FCB) in the disk directory. The Apple directory FCB uses one extent per directory entry to allocate file blocks. The EXM is then 1 - 1, or EXM = 0.

DSM A 2-byte value giving 1 less then the maximum number of blocks available on a disk. This number does not include the system tracks. The Apple has thirty-two tracks available; the first three tracks are system tracks. This computes to 128 blocks; therefore, DSM = 7F00H. DSM is used to compute the size of the allocation table pointed to by ALV in the DPB. The table size is (DSM + 1)/8. The Apple allocation table size is 16 bytes.

DRM A 2-byte value for the maximum number of directory entries, less 1. The Apple allows forty-eight entries; therefore, DRM = 2F00H.

AL0 and AL1 The allocation table assigning the directory blocks. The 1-byte values AL0 and AL1 are actually used together. Each bit in AL0 and AL1 corresponds to a reserved block. The Apple reserves the first two blocks for the directory; therefore, AL0, AL1 = C0H, 00H. More information on AL0, AL1 is found below in the ALV description.

CKS A 2-byte value for the size of the area pointed to by CSV in the DPB. The CSV area size is (DRM + 1)/4, which for the Apple is 12 bytes. See also CSV, discussed below.

OFF A 2-byte number indicating the number of system tracks. The Apple reserves the first three tracks for the operating system; therefore, OFF = 0300H.

CSV

CSV is the address of the scratch-pad area used by the BDOS to check if the disk has been changed. The size of this area is a function of the maximum number of directory entries for that drive and is found in the drive's DPB. Each DPH must have its own scratch-pad area.

ALV

ALV is the address of the disk storage allocation table. Each DPH has its own allocation table. The allocation table is filled during a warm boot and is used to keep track of which disk blocks are occupied. The BDOS checks the allocation table before each write to prevent the overwriting of data. The table is updated after each write to reflect which blocks have become occupied. File deletion will also be reflected in the allocation table. The allocation table can be thought of as a series of bits rather than bytes. Each bit in the table indicates the status of a disk block. If the bit is set, the block is being used. If the bit is cleared, the block is available. Consider the allocation table for disk A: for the 60K version of CP/M. The table starts at FF7DH and ends at FF8EH. The ascending byte memory locations represent ascending block numbers. For example, the following locations represent the respective blocks:

Address	*Block Numbers*
FF7DH	00H–0FH
FF7EH	10H–1FH
FF7FH	20H–2FH

and so on. The blocks are sequenced in each byte such that the high-order bit represents the lowest block number, while the low-order bit represents the highest block number. For instance, if block number 00H is the only occupied block, then the byte at address FF7DH is 80H. If blocks 00H and 01H are occupied, then the byte becomes C0H.

■ Zero-Page BIOS Requirements

CP/M requires that some zero-page memory locations be set aside. The BIOS must take this into account. The memory area 0000H through 00FFH is used as follows:

0000H–0002H The address containing the instruction JP WBOOT. The address of WBOOT is placed in addresses 0001H and 0002H as part of the warm boot routine as designated by Digital Research. Most programs use this jump instruction to locate the vector jump table, and they frequently exit to the CCP by a JP 0000H instruction.

0003H The IOBYTE location. Refer to chapter 4 for a discussion of IOBYTE usage.

0004H The location where the currently active drive is stored. See SELDSK for the number code used to designate a drive.

0005H–0007H The address containing the instruction JP BDOS. The BDOS routines are accessed through a call to this location. See the Digital Research *CP/M Interface Guide* supplied with your Microsoft documentation for details on using the BDOS. Locations 0006H and 0007H contain the lowest address used by the operating system (the top of the TPA). The area from 0100H to the latter address is the space available for program usage.

0008H–0027H Z-80 interrupt locations RST 0 through RST 5, not used by CP/M.

0030H–0037H The RST 7 location, not currently used by CP/M but reserved for future use.

0038H–003AH The RST 8 location, used by DDT but not otherwise used by CP/M.

003BH–003FH An area unused by CP/M but reserved for future use.

0040H–004FH An area that may be used by the BIOS for scratch-pad purposes. CP/M does not use this area. The Microsoft PS IIe BIOS uses this area.

0050H–005BH Another reserved area unused by current CP/M versions.

005CH–007CH The default File Control Block area used by the BDOS and transient command programs. See the *CP/M Interface Guide* for details on usage.

007DH–007FH An optional area for random records. See the *CP/M Interface Guide* for details.

0080H–00FFH The default DMA area. The CP/M warm boot always resets the DMA address to this location.

■ The Standard Microsoft SoftCard

The Mapping of the Standard SoftCard

The Apple BIOS uses both 6502 and Z-80 code. The Apple hardware design places the slot locations and hardware ROM areas in the memory areas bounded by $C000 and $CFFF. The dollar sign ($) is a prefix used to designate hexadecimal locations with respect to the 6502 microprocessor, while the H is a suffix designating hexadecimal locations used by the Z-80 microprocessor. The sixteen hardware pages are in a very inconvenient location for CP/M; they fall inside the Transient Program Area. When the standard SoftCard is activated, it remaps the

Apple Memory as shown in table 10.2. Mapping the memory this way places the Apple hardware pages in the BIOS memory area. The Z-80 mapping is valid only while the Z-80 is running the Apple. When the Z-80 relinquishes control to the 6502, the standard Apple addresses are in effect. See chapter 14 for the PS IIe's memory mapping.

The Operation of the Standard SoftCard

The standard SoftCard's operation is simple and rather clever (see chapter 14 for the PS IIe's operation). You turn the standard SoftCard on by writing to the slot containing the card, and you turn it off by writing again to the slot; that is, the standard SoftCard is toggled on and off by alternate write commands. When the standard SoftCard is turned on, it takes control of the Apple address and data buses and puts the 6502 microprocessor in a standby, or waiting, state. When the standard SoftCard is turned off, the Z-80 is put into a waiting state, and control is returned to the 6502. The switching of control between the Z-80 and the 6502 is easily done because the slots on the Apple's motherboard are

Table 10.2 ■ Standard SoftCard Memory Mapping

6502 Address	*Z-80 Address*
$0000–$0FFF	F000H–FFFFH
$1000–$1FFF	0000H–0FFFH
$2000–$2FFF	1000H–1FFFH
$3000–$3FFF	2000H–2FFFH
$4000–$4FFF	3000H–3FFFH
$5000–$5FFF	4000H–4FFFH
$6000–$6FFF	5000H–5FFFH
$7000–$7FFF	6000H–6FFFH
$8000–$8FFF	7000H–7FFFH
$9000–$9FFF	8000H–8FFFH
$A000–$AFFF	9000H–9FFFH
$B000–$BFFF	A000H–AFFFH
$C000–$CFFF	E000H–EFFFH
$D000–$DFFF	B000H–BFFFH
$E000–$EFFF	C000H–CFFFH
$F000–$FFFF	D000H–DFFFH

connected to the address bus, the data bus, and every important control line. In the Microsoft CP/M versions 2.23 and 2.20B, the routine that controls the toggling between the Z-80 and 6502 is located at $3C0 referencing to the 6502 or F3C0H referencing to the Z-80.

It is instructive to see how the standard SoftCard switches between the 6502 and the Z-80 microprocessors. The routine at location $3C0 for the Microsoft version 2.23 is

```
$3C0: LDA $C083     ;Put the Apple 16K language card into the
      LDA $C083     ;read/write mode.
      STA SOFTCARD  ;Enable the SoftCard and disable the 6502.
                    ;The SOFTCARD address is loaded during the
                    ;cold boot. For the SoftCard in slot 4,
                    ;SOFTCARD is $C400.
START:LDA $C081     ;Enable the Apple Monitor ROM.
      JSR SET6502   ;Load the 6502 registers.
                    ;Accumulator loaded from location $45.
                    ;X-register loaded from location $46.
                    ;Y-register loaded from location $47.
                    ;Status register loaded from location $48.
                    ;Enable the 6502 interrupts.
                    ;SET6502 is located at $E3F in the 60K BIOS.
      JSR ROUTINE   ;Run the 6502 subroutine.
      STA $C081     ;Make sure the ROM is enabled.
      SEI           ;Disable the 6502 interrupts.
      JSR SAVE      ;Place accumulator contents in $45,
                    ;X-register contents in $46, Y-register
                    ;contents in $47, and status register in $48.
      JMP $3C0      ;Loop back to beginning.
```

When the standard SoftCard has been booted by the Microsoft BIOS, the 6502 is put into a waiting state with its program counter (PC) pointing to START. In order for a 6502 routine to run, the routine address must be inserted into the ROUTINE location, which is at $3D0 for the 6502 or at F3D0H for the Z-80. The 6502 registers may be initialized before the call by the loading of memory locations $45, $46, $47, and $48. A write to the standard SoftCard location causes the 6502 to begin executing the program at the address START. The supplied program is run. The 6502 registers are stored back to their respective memory locations. A jump is taken, and a write instruction to the standard SoftCard location puts the Z-80 in control of the Apple. Recall that before a microprocessor executes an instruction, the PC is advanced to the next instruction. This means that the 6502 is put back into a waiting state with its PC pointing to START, and the Z-80 is now executing the instruction following the write to the standard SoftCard location. The cold boot finds the standard SoftCard slot and saves its Z-80 address in locations F3DEH and F3DFH.

We have just seen that the standard SoftCard BIOS is set up so that the 6502 microprocessor can be used in the execution of a routine running under CP/M. It is important to note that the 6502 and Z-80 cannot run simultaneously. If one microprocessor is working, the other must be waiting. A generalized program for running a 6502 program under CP/M versions 2.20B and 2.23 is shown below.

```
LD HL,6502ROUTINE   ;Load the HL registers with the address of
                    ;the 6502 routine. The address must use the
                    ;6502 memory address of that routine.
LD (0F3D0H),HL      ;Place the address in the ROUTINE location.
                    ;Pass values to the 6502 registers
                    ;if required.
LD (0F045H),A       ;For example, set up the accumulator. After
                    ;initializing registers, continue.
LD HL,(0F3DEH)      ;Load SoftCard address in the HL registers.
                    ;For the SoftCard in slot 4, E400H is loaded
                    ;into HL.
LD (HL),A           ;A write instruction to the SoftCard address
                    ;runs the 6502 program, and control is passed
                    ;to the Z-80 when the program executes its
                    ;RTS. Read the 6502 registers by reading
                    ;their respective memory locations.
LD A,(0F045H)       ;For example, read the accumulator. After the
                    ;registers are read, the remainder of the
                    ;Z-80 program follows.
```

■ Peripheral Cards

Peripheral card locations and identities are established when the standard SoftCard goes through a cold boot. A card slot is recognized as being filled or empty when two *checksums* are peformed on the slot page. A *checksum* is simply the result of adding a given number of bytes. If the checksums agree, the slot has a card. The card is identified by a *signature byte*. The *signature byte* is protocol defined by Apple Computer. The signature bytes are located at $Cn05, $Cn07, $Cn0B, and $Cn0C, where *n* is the slot number. Microsoft's CP/M version 2.20B looks at locations $Cn05 and $Cn07 only, while versions 2.23 and higher also inspect the $CnOB byte. Apple peripheral cards are identified by the following values:

	Signature Byte	
Card Type	*$Cn05*	*$Cn07*
Parallel	$48	$48
Communications	$18	$38
Serial	$38	$18
Disk controller	$03	$3C

Firmware cards can have the above values in locations $Cn05 and $Cn07, but they are identified by having a $01 in location $CnOB. It is therefore possible for version 2.20B to wrongly identify a firmware card. This is important since the standard SoftCard cold boot routine assigns input/output routine vectors according to card identities (see chapter 4). The cold boot stores the Disk Count Byte at $3B8; the Disk Count Byte is twice the number of disk controller cards plugged into the Apple. The cold boot also stores the card types in the Slot Type Table starting at $3B9 and ending at $3BF, where slot 1 is $3B9. The BIOS uses the following code to identify the card types:

0	No card identified; the slot may be empty.
1	Card detected, but it cannot be identified.
2	Disk controller card.
3	Communications card.
4	Serial card.
5	Parallel card.
6	Firmware card (versions 2.23 and higher only).

The cold boot routine is needed only when the system is brought up. In order to save space, the BIOS uses the cold boot routine area for the DIRBUF, the CSV scratch-pad areas, and the ALV allocation scratch-pad areas. This means that the cold boot routine must change the BOOT address in the vector jump table. CP/M version 2.20B replaces the JP instruction with a RET instruction. Versions 2.23 and 2.26 replace the bytes at BOOT with three NOP instructions, which means that a cold boot request will fall through to the warm boot.

Cold-booting the system requires more than just doing a jump to BOOT. The sequence of events during a cold boot is as follows for the standard SoftCard and the PS IIe.

1 The boot 0 routine in the disk controller card ROMs (read-only memory) loads track 0, sector 0 from the diskette into the memory starting at $800 and then does a jump to location $801.

2. The boot 1 routine at $801 then loads the CP/M RWTS and the boot 2 routine. The RWTS on the disk controller card is used by boot 1.
3. The boot 2 routine then makes sure that the booting is from the card in slot 6. Microsoft requires that drive A: be in slot 6, so such a precaution is necessary. In the standard SoftCard version, memory pages 2 and 3 are initialized, and the standard SoftCard is located by a routine that treats each peripheral card as if it were a standard SoftCard; that is, it tries to turn it on and perform a simple Z-80 code program. The PS IIe boot 2 doesn't use a card-finder routine. The remaining card slots are identified by a double checksum over the card addresses. The warm loader routine is then used to load the remaining systems. The warm loader is then modified to load only the CCP and BDOS. The standard SoftCard is turned on, and a jump is made to the BIOS cold boot routine. The PS IIe boot 2 differs slightly after the warm boot loader has been used and modified. The BIOS is relocated (see chapter 14) before jumping to the cold boot routine.
4. The cold boot completes the initializations, prints the sign-on message, and performs the warm boot.

11 The Microsoft Version 2.20B BIOS

The BIOS for the Microsoft standard SoftCard 56K CP/M version 2.20B will be described in this chapter. The Z-80 coded sections will use the H suffix for memory locations. The 6502 coded sections will use the $ prefix for the memory locations; the 6502 memory offset is assumed. It will be assumed here that the reader has sufficient programming experience that the descriptions can be kept terse.

The 2.20B BIOS extends into the Apple language card area, but it uses only bank 2 of the language card. The language card bank 1 is left unused.

All the logical device routines use the I/O Configuration Block (IOCB); see chapter 4. The IOBYTE is used to determine which physical device is to be used. The address for that device is taken from the IOCB, and a jump is made to that address.

The BIOS Map

The BIOS map is as follows:

DA00H–DA32H The BIOS vector jump tables.

DA33H–DA92H The Disk Parameter Headers for six drives.

DA93H–DAA1H The Disk Parameter Block.

DAA2H–DAC4H The slot initialization routine, which initializes communications and serial cards. The cards are identified from the Slot Type Table. The initialization starts at slot 7 and goes down to slot 1. The Asynchronous Communication Interface Adapter (ACIA) of the communications cards is set to 7 data bits, even parity, and 2 stop bits, and transmit interrupts are enabled.

DAC5H–DACBH A routine to place En00H in the HL registers; *n* is the slot number passed to the routine in the E register.

DACCH–DB07H WBOOT The warm boot routine. The stack pointer is initialized. The warm loader routine is called at $E00. The slot initialization routine is called. The CP/M BDOS zero-page addresses are initialized. The CCP is patched for either a 2- or 4-column directory. DAFDH = 1 for 2 columns; DAFDH = 3 for 4 columns. The warm boot ends with a jump to the CCP at address C400H.

DB08H–DB0BH CONST The routine to get the address of the console status routine from the IOCB at F380H and jump to that routine.

DB0CH–DB11H The CONST routine for the Apple keyboard.

DB12H–DB28H CONIN The console input routine. A call to the input character routine at DB50H is made, and the character is checked against the redefinition table at F3ACH. The routine returns with a character or translated character in register A.

DB29H–DB3AH The default address in the IOCB for console input. The DE registers are set to 3, for slot 3. If an 80-column card is in slot 3, the cold boot changes the next jump to go to the appropriate input routine. If no 80-column is detected, the routine proceeds to the Apple keyboard input routine starting at DB2FH.

DB3BH–DB41H The routine to set up and make the call to the 6502. The HL registers on entry contain the 6502 program address.

DB42H The routine that places the contents of the A register in C and falls into the CONOUT routine.

DB43H–DB4FH CONOUT The routine that checks the IOBYTE for the output device and then jumps to the selected routine. If the output goes to the logical LST: device, then the physical LPT: device is used.

DB50H–DB61H The character input routine, which checks the IOBYTE for the physical input devices and goes to the selected routine.

DB62H–DB65H A jump to the physical PTR: device. This routine may be used by the console input or logical RDR: device.

DB66H–DB74H LIST The logical LST: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

DB75H–DB86H PUNCH The logical PUN: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

DB87H–DB95H READER The logical RDR: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

DB96H–DBB7H A routine for 80-column cards. It conditions the memory locations and looks to see if an escape sequence is coming. Control is passed to routines to perform specific functions depending on how the output is to be performed.

DBB8H–DBDFH A routine that positions the cursor in the GOTOXY sequence.

DBE0H–DBF4H A routine that checks to see if there was a terminal lead-in character sent and calls routines as required.

DBF5H–DC3DH A routine that considers all the possible combinations and finally prints the character to the console via physical device TTY: or UC1: as required.

DC3EH–DC43H The physical TTY: device. This is the general console output routine. The jump address to the specific output routine is patched in during the cold boot. Since the output routines are slot-dependent, the slot number of the console is supplied in location DC3FH. The slot number here is 3.

DC44H–DCDEH The screen output routine for the standard 40-column Apple screen. This is the routine patched into the former routine if no serial or 80-column card is found in slot 3.

DCDFH–DCE9H The communications card output routine. A status loop is run. When the ACIA is ready, the character in the C register is transmitted.

DCEEH–DD03H A preparatory routine for setting up a serial card for either input or output.

DD04H–DD11H The serial card output routine. The output is performed by calling the 6502.

DD12H–DD1BH The communications card input routine. This routine resembles the output routine in structure.

DD1CH–DD2AH The serial card input routine.

DD2BH–DD30H The physical LPT: device output function. A jump is made to the card driver routine. The jump address is loaded during the cold boot and depends on the card type found in slot 1. Since the card routines are slot-dependent, this routine supplies the slot number in location DD2CH.

DD31H–DD3EH The parallel card output routine.

DD3FH–DD44H The physical PTP: device output function. A jump is made to the card driver routine. The jump address is loaded during the cold boot and depends on the card type found in slot 2.

Since the card routines are slot-dependent, this routine supplies the slot number in location DD40H.

DD45H–DD4CH The physical PTR: device output function. A jump is made to the card driver routine. The jump address is loaded during the cold boot and depends on the card type found in slot 2. Since the card routines are slot-dependent, this routine supplies the slot number in location DD46H.

DD4BH–DD55H HOME A disk routine to select track 0.

DD56H–DD5AH SETTRK A disk routine to select the track in register C.

DD5BH–DD6CH A computational routine used by the peripheral card drivers and disk I/O routines to get needed slot and memory addresses and the numbers passed to them from the physical device routines.

DD6DH–DD88H SELDSK A routine to select the disk drive and set flags to notify the disk I/O routines if the drive has been changed or a nonexistent drive was called.

DD89H–DD8DH SETSEC A routine to select the 128-byte CP/M sector.

DD8EH–DD92H SETDMA A routine to select the disk I/O buffer address.

DD93H–DDA2H READ A routine that sets up the disk read operation according to all the CP/M protocols; see chapter 10.

DDA3H–DDF1H WRITE A routine that performs the disk write operation using the CP/M protocols.

DDF2H–DE72H A routine used by both READ and WRITE to make sure that the CP/M protocols are met. A sector skew is done with the CP/M sector skew table. The data is moved to or from the CP/M RWTS buffer at $800. The read or write operation is then called.

DE73H–DE91H A routine that performs the actual read and write routines by calling the 6502 CP/M RWTS function.

DE92H–DEA1H The CP/M logical sector skew table. The table relates the 256-byte sector number to the logical 128-byte sector number used by CP/M.

F200H–F37FH The I/O Patch area. This is a space for user-provided routines required for special I/O situations. The IOCB must be patched to vector the device I/O to the routines in this area.

F380H–F395H The IOCB containing the vectors to the CP/M physical devices.

F396H–F3AAH A table used by the console routines to perform console functions. The table can be set up so that a variety of terminals may be run off the Apple.

F3C0H–F3DAH The routine that calls the 6502 microprocessor.

F3F0H–F3FFH The space used by the Apple to vector the interrupts and resets. The vectors under CP/M all point to $3C0, so the Z-80 never loses control of'the Apple.

F800H–F900H The buffer used by the CP/M RWTS.

FA00H–FFFCH The CP/M RWTS routines. These are all written in 6502 code.

The CP/M 2.20B includes a patch in the I/O Patch area to allow a serial card to be used in slot 3. The IOCB contains the patch addresses for the primary console routines.

■ The CPM56.COM Map

The program CPM56.COM contains the entire operating system image. You can modify the BIOS most easily by making modifications to CPM56.COM and then running the latter program. Below is a mapping of the CPM56.COM program when loaded into memory by DDT.

100H–2FFH The command portion of CPM56.COM.

300H–3FFH The boot 1 portion, which loads from track 0, sector 0 and is responsible for loading the CP/M RWTS sectors into the memory range $A00–$FFF and the boot 2 portion into the range $1000–$13FF.

400H–9FFH The CP/M RWTS.

A00H–BFFH The boot 2.

C00H–D7FH The I/O Patch area, which gets moved by boot 2 to F200H–F37FH.

D80H The IOCB console status vector.

D82H The IOCB console input vector 1 or TTY: device.

D84H The IOCB console input vector 2 or UC1: device.

D86H The IOCB console output vector 1 or TTY; device.

D88H The IOCB console output vector 2 or UC1: device.

D8AH The IOCB reader vector 1 or PTR: device.

D8CH The IOCB reader vector 2 or UR1: device.

D8EH The IOCB punch vector 1 or PTP: device.

D90H The IOCB punch vector 2 or UP1: device.

D92H The IOCB list vector 1 or LST: device.

D94H The IOCB list vector 2 or UL1: device.

D96H–DFFH The console hardware and software definition tables and the remainder of page 3 routines and vectors. The data in the range D80H–DFFH gets moved by boot 2 to F380H–F3FFH.

E00H–15FFH The CCP.

1600H–23FFH The BDOS.

2400H–29A7H The BIOS.

29A8H–29E7H The cold boot routine.

29E8H–29FFH Patches required for 2.20B to run a turnkey and correct a disk read/write problem.

■ The CPM56 Diskette Map

The Apple CP/M diskette system tracks are mapped as follows:

Track 00H, sector 00H The boot 1 sector.

Track 00H, sector 01H through track 00H, sector 06H
The CP/M RWTS.

Track 00H, sector 07H through track 00H, sector 08H
The boot 2 routine.

Track 00H, sector 09H through track 00H, sector 0AH
The I/O Patch area plus page F300H routines and tables.

Track 00H, sector 0BH through track 01H, sector 02H
The CCP.

Track 01H, sector 03H through track 02H, sector 00H
The BDOS.

Track 02H, sector 01H through track 02H, sector 06H
The BIOS.

The routines listed above use the CP/M RWTS sectors; see Appendix C.

■ CPM56 Card Driver Entry Points

A list of the entry points to the peripheral card drivers will be useful for BIOS patching. The list is given below.

DCDFH	The entry point to the communications card output routine.
DD04H	The entry point to the serial card output routine.
DD12H	The entry point to the communications card input routine.
DD1CH	The entry point to the serial card input routine.
DD31H	The entry point to the parallel card output routine.

All the driver routines require that the DE registers contain the card slot number before entry. The A and C registers are used as required by CP/M protocols.

12 The Microsoft Version 2.23 BIOS

The Microsoft 2.20B BIOS uses some ungainly fixes to correct a few problems, but still a few problems remain in the area of hardware interfacing. Microsoft has corrected most of the 2.20B problems with its standard SoftCard 2.23 BIOS. The few existing problems will be corrected in chapter 13.

The hardware interfacing is greatly improved because version 2.23 uses Apple Computer's protocols for operating what Apple calls *firmware cards*. Most of the cards that can operate a host of peripheral devices and have them do all sorts of neat tricks are firmware cards. Version 2.20B could not identify firmware cards and would often use the wrong I/O drivers. This caused a gnashing of teeth by those unfortunates who invested in expensive equipment and could not get it to operate under CP/M. Version 2.23 will operate the firmware cards, provided that the card manufacturers followed the Apple protocols.

There is another improvement in version 2.23. It is that the BIOS communications card driver uses the 6502 instead of the Z-80 to access the ACIA (Asynchronous Communication Interface Adapter). The Z-80 has a memory-refresh provision, which causes the address to be accessed to be preread before the actual reading or writing occurs. Reading the data port on an ACIA clears the ACIA status flags, which means that the data can disappear before a second read is made. You can lose data when the ACIA is read by the Z-80; using the 6502 instead eliminates this problem.

The 60K 2.23 version has more memory available to programs than the 56K 2.20B version because both 4K banked memories on the language card are used. The 56K 2.20B version uses only bank 2 on the language card. Version 2.23 uses bank 1 to store the BIOS disk-handling routines, which include the 6502 CP/M RWTS, the Z-80 BIOS routines, and two-thirds of the BDOS, which leaves bank 1 available for program memory.

■ The BIOS Map

The BIOS map given below uses the same conventions as are used in chapter 11. The routines written in Z-80 code use the H suffix in their addresses. The routines written in 6502 code use the $ prefix in their addresses.

F200H–F37FH The I/O Patch area. This is a space for user-provided routines required for special I/O situations. The IOCB must be patched to vector the device I/O to the routines in this area.

F380H–F395H The IOCB, which contains the vectors to the CP/M physical devices.

F396H–F3AAH A table used by the console routines to perform console functions. The table can be set up so that a variety of terminals may be run with the Apple.

$3C0–$3DA The routine that calls the 6502 microprocessor.

$3F0–$3FF The space used by the Apple to vector the interrupts and resets. The vectors under CP/M all point to $3C0, so the Z-80 never loses control of the Apple.

$800–$8FF The default I/O buffer area used by the CP/M RWTS.

$900–$9FF A nibble buffer used by the CP/M RWTS.

FA00H–FA32H The BIOS vector jump tables.

FA33H–FA92H Disk Parameter Headers for four drives.

FA93H–FAA1H The Disk Parameter Block.

FA82H–FAB0H The slot initialization routine, which initializes communications, serial, and firmware cards. The cards are identified from the Slot Type Table. The initialization starts at slot 7 and goes down to slot 1. The ACIA of the communications cards is set to 7 data bits, even parity, and 2 stop bits, and transmit interrupts are enabled.

FAB1H–FAB7H A routine to place En00H in the HL registers; *n* is the slot number passed to the routine in the E register.

FAB8H–FB0FH WBOOT The warm boot routine. The stack pointer is initialized. The warm loader routine is called at $E00. The slot initialization routine is called. The CP/M BDOS zero-page addresses are initialized. The CCP is patched for either a 2- or 4-column directory. FB05H = 1 for 2 columns; FB05H = 3 for 4 columns. The warm boot ends with a jump to the CCP at address C400H.

FB10H–FB13H CONST The routine that gets the address of the console status routine from the IOCB at F380H and jumps to that routine.

FB14H–FB19H The CONST routine for the Apple keyboard.

FB1AH–FB32H CONIN The console input routine. A call to the input character routine at FB5AH is made, and the character is checked against the redefinition table at F3ACH. The routine returns with a character or translated character in register 4.

FB33H–FB38H The default address in the IOCB for console input. The DE registers are set to 3, for slot 3. If an 80-column card is in slot 3, the cold boot changes the next jump to go to the appropriate input routine. If no 80-column card is detected, the routine proceeds to the Apple keyboard input routine starting at FB39H.

FB45H–FB4BH The routine that sets up and makes the call to the 6502. The HL registers on entry contain the 6502 program address.

FB4CH The routine that places the contents of the A register in C and falls into the CONOUT routine.

FB4DH–FB59H CONOUT The routine that checks the IOBYTE for the output device and then jumps to the selected routine. If the output goes to the logical LST: device, then the physical LPT: device is used.

FB5AH–FB6BH The character input routine, which checks the IOBYTE for the physical input device and goes to the selected routine.

FB6CH–FB6FH A jump to the physical PTR: device. This routine may be used by the console input or logical RDR: device.

FB70H–FB7EH LIST The logical LST: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

FB7FH–FB90H PUNCH The logical PUN: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

FB91H–FB9FH READER The logical RDR: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

FBA0H–FBCAH A routine for 80-column cards. It conditions the memory locations and looks to see if an escape sequence is coming. Control is passed to routines to perform specific functions depending on how the output is to be performed.

FBCBH–FBF1H The location where the cursor positioning by the GOTOXY sequence routine starts; then the routine jumps to FCA4H. Routines required for the functioning of the routine at FBA0H are also placed out of sequence compared to version 2.20B and start at address FC56H. This displacement is required so that room for a buffer used by the RWTS can be located at $C00.

FBF4H–FBF8H SETSEC A routine that selects the 128-byte CP/M sector.

FBF9H–FBFDH SETDMA A·routine that selects the disk I/O buffer address.

$C00–$C55 The location of one of the CP/M RWTS nibble buffers.

FC56H–FC6AH A routine that checks to see if there was a teriminal lead-in character sent and calls routines as required.

FC6BH–FCB4H A routine that considers all the possible combinations and finally prints the character to the console via physical device TTY: or UC1: as required.

FCB5H–FCBAH The physical TTY: device. This is the general console output routine. The jump address to the specific output routine is patched in during the cold boot. Since the output routines are slot-dependent, the slot number of the console is supplied in location FCB6H. The slot number here is 3.

FCBBH–FD0DH The screen output routine for the standard 40-column Apple screen. This is the routine patched into the preceding routine if no serial or 80-column card is found in slot 3.

FD0EH–FD27H The communications card output routine using 6502 code. A status loop is run. When the ACIA is ready, the character in the C register is transmitted.

FD28H–FD70H Screen function routines, located in the BIOS out of sequence compared to version 2.20B.

FD71H–FD82H The serial card output routine. It performs the output by calling the 6502.

FD83H–FD98H A preparatory routine for setting up a serial card for either input or output.

FD99H–FDA8H The console status routine for a firmware card, which calls a 6502 routine for operation.

FDA9H–FDB6H The firmware card output routine, which calls a 6502 routine for operation.

FDB7H–FDC0H The firmware card input routine, which calls the 6502 routine at $E0F for operation.

FDC1H–FDCFH The serial card input routine.

$DD0–$DE0 The firmware card initialization routine, which is followed by a routine that uses the Apple protocol for firmware I/O.

$DE1–$DEE The firmware card output routine.

$DEF–$DFA The firmware card routine that waits for the card to accept I/O.

$E00–$E02 The CP/M entry to the warm loader routine.

$E03–$E08 The entry to the CP/M RWTS routine on the language card bank 1.

$E09–$E0E The second entry to the warm loader routine on bank 1 of the language card.

$E0F–$E1C The firmware card input routine.

$E1D–$E25 The firmware card routine to obtain the card's I/O status.

$E26–$E3E The routine that sets up all the parameters used by the firmware card protocol and sets up the coresident ROM area at $C800 to be ready for the firmware card's requirements.

$E3F–$E4A The routine that is used by the routine at $3C0 to set all the 6502 registers and flags from their respective memory areas. The 6502 interrupt is also enabled.

FE48H–FE54H The communications card input routine. This routine resembles the output routine in structure.

FE55H–FE5AH The physical LPT: device output function. A jump is made to the card driver routine. The jump address is loaded during the cold boot and depends on the card type found in slot 1. Since the card routines are slot-dependent, this routine supplies the slot number in location FE56H.

FE5BH–FE68H The parallel card output routine.

FE69H–FE6EH The physical PTP: device output function. A jump is made to the card driver routine. The jump address is loaded during the cold boot and depends on the card type found in slot 2. Since the card routines are slot-dependent, this routine supplies the slot number in location FE6AH.

FE6FH–FE74H The physical PTR: device output function. A jump is made to the card driver routine. The jump address is loaded during the cold boot and depends on the card type found in slot 2. Since the card routines are slot-dependent, this routine supplies the slot number in location FE70H.

FE75H–FE7FH HOME A disk routine to select track 0.

FE80H–FE84H SETTRK A disk routine to select the track in register C.

FE85H–FE96H A computational routine used by the peripheral card drivers and disk I/O routines to get needed slot and memory addresses and from the numbers passed to them from the physical device routines.

FE97H–FEC5H SELDSK A routine that selects the disk drive and sets flags to notify the disk I/O routines if the drive has been changed or a nonexistent drive was called.

FEC6H–FECBH READ The entry point to the disk read routine found on bank 1 of the language card.

FECCH–FED1H WRITE The entry point to the disk write routine found on bank 1 of the language card.

FED2H–FED8H The routine used when the 6502 must be called by a routine on bank 1.

FED9H–FEDFH The routine used when a disk I/O error is encountered by the disk-handling routines on bank 1. Bank 2 is switched back on, and the BDOS error routine is called.

FEE0H–FEE3H The function through which the bank 1 routines return. Their return turns on bank 2.

$FFAC–$FFE8 The memory used by the CP/M RWTS for what is called *prenibblizing* routines. It is located above the BDOS in memory and doesn't fit neatly into this memory map. Microsoft had to put it here to fit the second segment of BDOS on the first language card bank. Version 2.23 gets choppy from here on.

The following are located on bank 1 of the language card.

$D000–$D246 The first segment of the CP/M RWTS. The RWTS is split into two segments for reasons known only to Microsoft.

B247H–B256H The disk read operation, set up according to all the CP/M protocols. See chapter 10.

B257H–B270H The disk write operation, performed according to the CP/M protocols.

B271H–B333H A routine used by both READ and WRITE to make sure that the CP/M protocols are met. A sector skew is done with the CP/M sector skew table. The data is moved to or from the CP/M RWTS buffer at $800. The read or write operation is then called.

B334H–B358H The location where the actual read and write routines are performed by the calling of the 6502 CP/M RWTS function.

B359H–B368H The CP/M logical sector skew table. The table relates the 256-byte sector number to the logical 128-byte sector number used by CP/M.

$D369–$D5BC The second segment of the CP/M RWTS.

B5C0H–BFFFH The second BDOS segment. This is not the BIOS, but it is included for completeness.

■ The CPM60.COM Map

The program CPM60.COM contains the entire operating system image. You can modify the BIOS most easily by making modifications to CPM60.COM and then running the latter program. Below is a mapping of the CPM60.COM program when it is loaded into memory with DDT.

100H–3FFH The command portion of CPM60.COM.

400H–500H The boot 1 portion, which loads from track 0, sector 0 and is responsible for loading the CP/M RWTS sectors into the memory range $A00–$FFF, loading boot 2 into $1000–$12FF, and loading the $300-page area into $1300–$13FF.

500H–746H The first segment of the CP/M RWTS.

747H–858H The BIOS read/write portions of the disk-handling routines.

859H–AFFH The second segment of the CP/M RWTS.

B00H–CFFH The boot 2.

D00H–E7FH The I/O Patch area, which gets moved by boot 2 to F200H–F37FH.

E80H The IOCB console status vector.

E82H The IOCB console input vector 1 or TTY: device.

E84H The IOCB console input vector 2 or UC1: device.

E86H The IOCB console output vector 1 or TTY: device.

E88H The IOCB console output vector 2 or UC1: device.

E8AH The IOCB reader vector 1 or PTR: device.

E8CH The IOCB reader vector 2 or UR1: device.

E8EH The IOCB punch vector 1 or PTP: device.

E90H The IOCB punch vector 2 or UP1: device.

E92H The IOCB list vector 1 or LST: device.

E94H The IOCB list vector 2 or UL1: device.

E96H–EFFH The console hardware and software definition tables and the remainder of page 3 routines and vectors. The data in the range E80H–EFFH gets moved by boot 2 to F380H–F3FFH.

F00H–17FFH The CCP.

1800H–1BFFH The non-language card BDOS segment plus the prenibblizing CP/M RWTS routines.

1C00H–26FFH The language card segment of BDOS.

2700H–2BE9H The BIOS.

2BEAH–2BFFH The cold boot routine.

■ The CPM60 Diskette Map

The Apple CP/M diskette system tracks are mapped as follows:

Track 00H, sector 00H The boot 1 sector.

Track 00H, sector 01H through track 00H, sector 06H
The CP/M RWTS and Z-80 BIOS disk routines.

Track 00H, sector 07H through track 00H, sector 08H
The boot 2 routine.

Track 00H, sector 09H through track 00H, sector 0AH
The I/O Patch area plus page F300H routines and tables.

Track 00H, sector 0BH through track 01H, sector 03H
The CCP.

Track 01H, sector 04H through track 01H, sector 07H
The first segment of BDOS.

Track 01H, sector 08H through track 02H, sector 02H
The second segment of BDOS.

Track 02H, sector 03H through track 02H, sector 03H
The BIOS.

The routines listed above use the CP/M RWTS sectors; see Appendix C.

■ CPM60 Card Driver Entry Points

The entry points to the BIOS card driver routines are listed below as an aid to BIOS patching.

FD0EH	The entry point to the communications card output routine.
FD71H	The entry point to the serial card output routine.
FDA9H	The entry point to the firmware card output routine.
FDB7H	The entry point to the firmware card input routine.
FDC1H	The entry point to the serial card input routine.
FE4BH	The entry point to the communications card input routine.
FE5BH	The entry point to the parallel card output routine.

All the driver routines require that the DE registers contain the card slot number before entry. The A and C registers are used as required by CP/M protocols.

13 Patching the Microsoft Standard SoftCard BIOS

In this chapter a variety of patches to the Microsoft standard SoftCard BIOS (Basic Input Output System) will be presented. The standard SoftCard manual discusses making BIOS patches using the CONFIGIO program. The use of this program can be confusing to both the novice and the sophisticated user. For this reason, an alternative approach to BIOS patching will be shown.

The complete CP/M operating system is included in the CPM56.COM and CPM60.COM files. It is an easy matter to place the patches in these files with the aid of DDT (see chapter 8). When CPM60 or CPM56 is run, the altered operating system will be permanently installed on the diskette. The mapping of these files was shown in chapters 11 and 12.

Squashing Microsoft Version 2.20B Bugs

For the initial example, we will correct some bugs that exist in the 60K and 56K CP/M versions. The first thing to be done is to copy DDT.COM and CPM60 or CPM56 to another diskette; don't make any alterations to your original copy. Make sure that the duplicate diskette has the CP/M operating system installed so that it can be placed into drive A:. Place the duplicate diskette in drive A:, and enter a CONTROL-C to warm-boot the system. Use DDT to load the file into memory. The first example is the correction of the bug in the 56K version that exchanges the PTP: and UP1: devices. This bug generally goes unnoticed because the default vectors in IOCB (see chapter 4) both point to the same output routine for the card in slot 2. If you were to have different routines for PTP: and UP1:, then problems would arise.

On with the correction: first type

```
DDT CPM56.COM
```

After the DDT prompt (#) appears, type

```
S2581
```

and remember to terminate all input by pressing the RETURN key. DDT will respond with

```
2581 20
```

After the 20, enter 28, and press RETURN. You have just changed the byte in memory location 2581H from 20H to 28H. On the next line, enter a period (.), and then press RETURN. The DDT prompt will reappear. Type CONTROL-C, and you will exit DDT to the CCP (Console Command Processor). Type

```
SAVE 42 CPM56.COM
```

The altered file CPM56.COM will now be stored on the diskette. Now type

```
CPM56 A:
```

After you respond to the program prompts, the new operating system will be placed permanently on the drive A: diskette's system tracks. The patched operating system may be transferred to other diskettes by the COPY program on the master diskette. The command

```
COPY B:=A:/S
```

will copy only the operating system from the diskette in drive A: to that in drive B:.

The 56K version has an annoying feature when an Apple IIe has an 80-column card in the auxiliary slot. Every time there is a warm boot, the screen is cleared. The reason for this is that the BIOS initializes all the peripheral cards on each warm boot. You may correct this by using DDT to change the following bytes. Type

```
S24D8
```

DDT will respond with

```
24D8 CD
```

Enter 0 and RETURN. DDT will then respond with

```
24D9 A2
```

Enter 0 and RETURN. DDT will then respond with

```
24DA DA
```

Enter 0 and RETURN. DDT will then respond with

```
24DB AF
```

Then enter a period (.) and RETURN to exit the set mode. You have changed the bytes at locations 24D8H, 24D9H, and 24DAH to 00H. This removes the call to the initialization program in the warm boot routine. One more byte needs to be changed. Pressing the RESET key will cause the Apple to perform some reinitializations of its own before CP/M does a warm boot. This will cause a weird screen display. The RESET has to be changed to cause the system to cold-boot. You accomplish this by using the set command in DDT to change the byte in location DF4H from A6H to 00H. Exiting DDT as before and typing

```
SAVE 42 CPM56.COM
```

will place the modified BIOS on the system tracks.

Multiple changes can be made during one session with DDT. The change to the PTP: and UP1: vectors and the warm boot fix can be handled in one pass through DDT.

■ Squashing Microsoft Version 2.23 Bugs

CPM60 is not without problems. The PUN: vector problems were corrected, but an error in the RDR: vectoring was introduced. This error is always noticeable and must be corrected. The Apple IIe warm boot problem is present as well. Using DDT to change the following locations in the CPM60.COM file will correct the problems. To correct the warm boot problem:

Change 0EF4H from A6H to 00H.
Change 27C4H from CDH to 00H.
Change 27C5H from 82H to 00H.
Change 27C6H from DAH to 00H.

To correct the RDR: vector problem:

Change 2897H from 08H to 04H.

Type

```
SAVE 44 CPM60.COM
```

to save the patched version of the file.

■ Using Other Slots for the Printer

Often a user has more than one printer attached to the Apple. The Microsoft BIOS defaults to slot 1 for the LST: device. This means that the card for a second printer, which is in some other slot, is not available. There is a very simple patch to make a peripheral card in any slot become the logical device I/O card.

As an example, let's assume there is a printer attached to a card in slot 1, and we wish to access a printer attached to a card in slot 5. We will assign the physical device type LPT: to the slot 1 printer and the physical device type UL1: to the slot 5 printer. This means that the IOCB (I/O Configuration Block) vector in locations F392H and F393H must point to the BIOS routine that drives the card in slot 1, and locations F394H and F395H must point to the routine that drives the card in slot 5. The routine for the slot 1 card already exists in the BIOS, and we don't have to change the first vector. Microsoft has provided space for BIOS patches in the memory region F200H through F37FH. We will put the routine to drive the slot 5 card in the memory locations starting with the address F200H. Then the second IOCB vector must point to F200H. Therefore, we must place 00H in location F394H and F2H in location F395H; remember that a vector contains the address with the high byte and low byte in an inverted order. In most cases, the driver routine is already available in the BIOS. We can use the existing routines to drive our slot 5 card.

The BIOS card driver routines require some of the Z-80 registers to be initialized before they are entered. First the DE registers must contain the slot number of the peripheral cards. This means that the D register contains 00H, and the E register contains the slot number. Our example of a card in slot 5 requires that the E register contain a 05H. An output routine requires that the ASCII character to be sent to the card be contained in the C register. An input routine will place the input character in the A register. The last two statements reflect the CP/M protocols for input and output devices.

Assume that the slot 5 card is a parallel card and that we are patching the 60K BIOS version. By the time the BIOS jumps to the routine pointed to by the IOCB, the output character is in the C register, so we needn't consider setting up the C register. The routine at F200H need only place the slot number in the DE registers and then jump to the parallel card output routine. The parallel card output is located at F35BH. The routine at F200H should then be

```
F200 LD DE,0005
F203 JP F35BH
```

This is represented in machine code by the following:

Location	*Content*
F200H	11H
F201H	05H
F202H	00H
F203H	C3H
F204H	5BH
F205H	F3H

The above patch can be done to the BIOS with DDT, but it will not be permanent. A cold boot will remove it. To make a permanent patch, use DDT to change the following locations in the CPM60.COM file.

Location	*Insert*
D00H	11H
D01H	05H
D02H	00H
D03H	C3H
D04H	5BH
D05H	F3H

This will place the latter code into the patch area. The next change will alter the IOCB to point to the patch.

Location	*Insert*
E94H	00H
E95H	F2H

Saving the patched CPM60.COM and then running it will place the patch permanently on the system tracks.

The above procedure may be generalized to patch CPM60.COM for a card in any slot. The card's driver routine must be present in the BIOS.

Location	*Insert*
D00H	11H
D01H	Card's slot number
D02H	00H
D03H	C3H
D04H	Low byte of the card driver routine's address
D05H	High byte of the card driver routine's address

If the UP1: device is to be changed, then change the following:

Location	*Insert*
E94H	00H
E95H	F2H

If the LPT: device is to be changed, then change the following:

Location	*Insert*
E92H	00H
E93H	F2H

The addresses of the card driver routines are in chapter 12.

Patching CPM56.COM is analogous to patching CPM60.COM; however, the addresses are different. To patch in the peripheral card's drivers, enter the following:

Location	*Insert*
C00H	11H
C01H	Card's slot number
C02H	00H
C03H	C3H
C04H	Low byte of the card driver routine's address
C05H	High byte of the card driver routine's address

If the UP1: device is to be changed, then change the following:

Location	*Insert*
D94H	00H
D95H	F2H

If the LPT: device is to be changed, then change the following:

Location	*Insert*
D92H	00H
D93H	F2H

The addresses of the card's driver routines are in chapter 11.

It has been assumed that you know your peripheral card type. If you don't know what type your card is, use DDT to examine the SLOTYPS table starting at F3B9H. Enter DDT and type

```
DF3B9
```

Then press the RETURN key. DDT will print a memory dump with the first line possibly looking like this:

```
F3B9 05 03 06 00 05 02 00
```

The bytes following F3B9H are the identification codes for the card types. The first byte is for slot 1, the second is for slot 2, and so on. The above example shows a parallel card in slot 1, a communications card in slot 2, and so on. The identification codes are given in chapter 10. Note the card type from this table, and select the appropriate driver routine for the BIOS patch.

■ Stopping the Printer from Double-spacing

The card driver routines bypass all or most of the software on the peripheral card ROMs. This is why the special printer control sequences do not work in CP/M. An often-encountered problem is the printer's double-spacing of lines. The reason for this double-spacing is that CP/M issues a line-feed instruction after each carriage return. A printer that does an automatic line feed after a carriage return will issue two line feeds instead of one when operating under CP/M. The cure is to turn off the automatic line feed on the printer. This is sometimes impossible; for example, the printer may be a typewriter. Many cards will strip the line feeds from the output if they receive an instruction such as CONTROL-I K from the computer. These card features are accessible under Apple DOS because the card ROMs are used. The same instruction will do nothing under CP/M because the ROMs are bypassed. The following patch is the double-spacing cure.

The patch program looks for a carriage return–line feed sequence. A line feed will not be sent to the printer if the previously sent character was a carriage return. The program should be placed at F200H or the appropriate locations in the CPM60.COM or CPM56.COM files. Remember, if you have already entered one patch at F200H, then you must start this patch at a higher address in the patch area. The program is

```
       LD A,(LAST)   ;Load the last character printed into A.
       CP 0DH        ;Is it a carriage return?
       JR NZ,NOTCR   ;Branch if it isn't.
       LD A,C        ;Place the current output character in A.
       CP 0AH        ;Is it a line feed?
       JR NZ,NOTLF   ;Branch if it isn't.
       LD (LAST),A   ;Save the current character.
       RET           ;Return without printing a line feed.
NOTCR: LD A,C        ;Place the current character in A.
NOTLF: LD (LAST),A   ;Save the current character.
       LD DE,slot    ;Place the card's slot number in DE.
       JP driver     ;Jump to the suitable card driver.
LAST:  DB 00H        ;The last character printed is stored here.
```

Placing the machine-coded version of this program into the CPM60.COM requires using DDT to load the following addresses:

Location	*Insert*
D00H	3AH
D01H	19H
D02H	F2H
D03H	FEH
D04H	0DH
D05H	20H
D06H	08H
D07H	79H
D08H	FEH
D09H	0AH
D0AH	20H
D0BH	04H
D0CH	32H
D0DH	19H
D0EH	F2H
D0FH	79H
D10H	32H
D11H	19H
D12H	F2H
D13H	11H
D14H	Card's slot number
D15H	00H
D16H	C3H
D17H	Low byte of the card driver routine's address
D18H	High byte of the card driver routine's address
D19H	00H

The requisite IOCB vector must be patched. Again, save CPM60.COM, and run the program.

■ Adding XON/XOFF Handshaking

Sometimes a driver routine doesn't exist for the card, or it is impossible to use the existing routines. The patch in these cases must contain the complete driver. We end this chapter with such a patch. Some printers, such as the Diablo brand, do software handshaking. This is done by the printer's sending a message to the computer to stop issuing data when the printer's input buffer is full. When the printer is ready to accept more data, a message is sent to the computer to resume sending. This scheme is required for remote terminals that operate over telephone lines. The programs given are for printers that use the XON/XOFF protocol. This protocol dictates that the sending unit stop sending when it receives an XOFF character (a CONTROL-S) from the receiving unit. The sending unit may continue sending when it receives an XON character (a CONTROL-Q) from the receiving unit. The programs are listed in machine code only since they contain both Z-80 and 6502 codes. The first program is intended for a communications card. The listing is sequential and should be placed in the CPM60.COM file or the CPM56.COM file by DDT. The code should be inserted starting at location D00H for CPM60.COM or at location C00H for CPM56.COM. The code that follows is in hexadecimal notation.

```
3E xx 32 46 F0 79 32 45 F0 21 14 02 22 D0 F3 2A DE F3 77 C9 A5 46
0A 0A 0A 0A AA BD 8E C0 29 03 F0 F9 29 01 D0 06 A5 45 9D 8F C0 60
BD 8F C0 29 7F C9 13 D0 E6 BD 8E C0 29 01 F0 F9 BD 8F C0 4C 1B 02
```

The *xx* is the slot number of the communications card. The IOCB locations for the chosen device must be made to point to F200H in the CPM60.COM or CPM56.COM files. The second program is for the Apple Super Serial card and is to be implemented exactly as for the communications card. The code for the program is as follows:

```
3E xx 32 46 F0 79 32 45 F0 21 14 02 22 D0 F3 2A DE F3 77 C9 A5 46
0A 0A 0A 0A AA BD 89 C0 29 18 F0 F9 29 08 D0 06 A5 45 9D 88 C0 60
BD 88 C0 29 7F C9 13 D0 E6 BD 89 C0 29 08 F0 F9 BD 88 C0 4C 1B 02
```

14 The Microsoft Premium SoftCard IIe (2.26 BIOS)

The Microsoft Premium SoftCard IIe (PS IIe) is constructed very differently from the standard SoftCard. The PS IIe (Microsoft version 2.26 BIOS) will run only on the Apple IIe. It is designed to be put into the Apple IIe auxiliary slot and produces an 80-column display for CP/M without the need of additional hardware. The PS IIe contains 64K of memory and a Z-80B microprocessor. The Z-80B is run three times faster than the Z-80A used on the standard SoftCard. CP/M programs executing on the PS IIe will run up to three times faster than on the standard SoftCard. The 64K of memory on the PS IIe means not only that more memory is available for the CP/M programs but also that some interesting hardware functions can be performed. For instance, when the Apple IIe is not running CP/M, the PS IIe behaves exactly like an extended-memory 80-column card. The other hardware functions will be described later.

The Comparison to Earlier BIOS Versions

The user will not see any difference in the operations of programs with the 2.26 BIOS except as regards speed. The 2.26 BIOS is remarkably bugfree, and no patches will be given for the BIOS alteration. An earlier BIOS (version 2.25) does have some bugs. If you have the earlier version, it is recommended that you get it upgraded. Unfortunately, the PS IIe's manuals contain errors. The most troublesome errors are in the 6502 BIOS section of the manual. The corrections to those errors will be covered later.

The 2.26 BIOS displays some similarity to the Microsoft BIOS versions 2.20B and 2.23 in that patch areas and I/O tables are assigned. The location of these assignments has been changed arbitrarily from those used for the standard SoftCard. The user patch area is the page of memory starting at FE00H (we are using the standard notation of the H suffix for Z-80 hexadecimal addresses and the $ prefix for 6502 hexadecimal addresses). The Slot Type Table is still at F3B8H, with the card types identified as discussed in chapter 10. The IOCB is at F3C0H. The reason for Microsoft's changing the locations of the patch and IOCB areas from the standard SoftCard BIOS versions is known only to Microsoft. No matter which location is chosen, the tables or patch areas will end up in the middle of the BDOS. The 2.26 BIOS has a hole in its middle to make room for the tables.

■ The PS IIe Hardware and Software Configuration

The PS IIe BIOS consists of two parts, the 6502 BIOS and the Z-80 BIOS. The 6502 BIOS resides in the main memory, that is, the memory on the Apple IIe motherboard. The Z-80 BIOS and the BDOS reside in the auxiliary memory, that is, the 64K of memory on the PS IIe. The Premium SoftCard IIe's hardware is designed to isolate the Z-80 from the main memory while running CP/M. This isolation allows the Z-80 and 6502 microprocessors to run programs simultaneously. The standard SoftCard requires that only one microprocessor be in operation at a time.

The Z-80 calls on the 6502 to perform all hardware functions. Because the hardware-handling part of the BIOS is so extensive, it takes up considerable memory space, but this space is occupied in the main memory. This makes the Z-80 BIOS much smaller than its counterpart on the standard SoftCard. This adds to the available room to run CP/M. The standard SoftCard could not use the memory addresses assigned to the hardware slots ($C000 to $CFFF), and in order to make the memory contiguous, it had to remap the memory addresses for the Z-80. This memory problem with the hardware slots doesn't exist for the PS IIe. Since the auxiliary 64K memory is isolated while CP/M is running, the memory addresses are kept unaltered, and direct access to the hardware slots by the Z-80 is impossible.

The PS IIe does not use any of the Apple IIe built-in firmware to operate the 80-column display. The reason is that Apple has so much software overhead to run the video display that the 80-column mode runs more slowly than the 40-column mode. The loss of speed is noticeable when the screen scrolls upward. The PS IIe uses its own screen drivers, which arrangement noticeably increases the response of the 80-column display.

Under standard operation of the Apple IIe auxiliary-slot 80-column card, 1K of memory in the auxiliary memory is used by the 80-column display. Since the PS IIe isolates the entire 64K auxiliary memory, it provides the needed 1K of memory for 80-column operation with an additional 1K video memory bank. The special 1K video memory is used only while CP/M is running.

The method used by the Z-80 to tell the 6502 to perform a task is somewhat involved. The communication is done by the hardware on the PS IIe. The following Z-80 instruction sequence is first executed.

```
LD A,INTVEC     ;INTVEC has a value depending on what the 6502 is
                ;to do.
LD I,A          ;This is the remainder of the 6502
                ;instruction sequence.
```

The hardware then effectively makes the INTVEC byte appear at location $2000 in the auxiliary memory when that memory location is looked at by the 6502. The 6502 polls this location and tests to see if a byte with bit B7 set (see Appendix A for definitions) has been written there. If such a byte appears, then it halts the Z-80. This is to prevent conflicts from occurring if the Z-80 requests the 6502 again before the 6502 has completed its tasks. The parameters needed to run the 6502 routines are then passed from the auxiliary memory to the main memory. The 6502 then executes the routine and passes any data, if required, back to the main memory. The 6502 tells the Z-80 to resume. The 6502 then continues to poll the memory location $2000 for a message from the Z-80.

The polling procedure used by the 6502 is a loop that looks at location $2000 and tests the byte found there. If no Z-80 request is pending, the loop will perform a print-spooling procedure. After it has cycled through the spooling procedure, it goes back to the top of the loop and tests location $2000.

The print-spooling procedure is another feature of the PS IIe. Any request by CP/M to write to the LST: device is handled in the following way by the 6502 BIOS. A write instruction to slot 1 when the slot contains a parallel card or a communications card will not send the data directly to the card. The data is first placed in a buffer. The data is read from the buffer and written to the card only during the Z-80 polling operation. Since a physical device is much slower than the computer, the buffer will absorb the data very quickly, which in turn makes the CP/M program run faster. The memory available for a print spool buffer lies between the locations $4000 and $BFFF. This is 32K of memory, which will hold approximately eight pages of standard text. If the buffer should fill before all the text has been printed, the CP/M system slows down and runs as if no buffer existed. While the buffer is full, each write instruction transfers data to the buffer only after 1 byte has been removed from the buffer and sent to the printer. The CP/M system will run slowly until all the text has been placed into the buffer. When no more text is forthcoming, the CP/M system resumes normal speed while the remaining contents of the buffer are dumped to the printer. Note that the print spooler works only for slot 1, and the card must be either a parallel or communications type.

The print-spooling operation also polls the keyboard. Each time the print spooler is called, the keyboard is checked for input. Any input is stored in the 256-byte keyboard buffer starting at $1400 in the main memory. The keyboard buffer is read when CP/M asks the console for input.

The 6502 sees the PS IIe simply as an 80-column extended-memory card. The standard extended-memory 80-column card permits memory management through the use of soft-switches. The soft-switches used by the 6502 BIOS for memory management not requiring video output are shown in table 14.1.

Table 14.1 ■ The PS IIe's Memory Management Soft-Switches

Function	*Soft-Switch*
Read from auxiliary memory range $200 through $BFFF	Write to $C003
Read from main memory range $200 through $BFFF	Write to $C002
Write to auxiliary memory range $200 through $BFFF	Write to $C005
Write to main memory range $200 through $BFFF	Write to $C004
Make the auxiliary memory ranges $000 through $1FF and $D000 through $FFFF active, and make the auxiliary memory bank-switched memory active	Write to $C009
Make the main memory ranges $000 through $1FF and $D000 through $FFFF active, and make the main memory bank-switched memory active	Write to $C008

The soft-switches permit data to be read from the main memory and stored in the auxiliary memory, and vice versa. The 6502 BIOS uses these switches when data transfers are required between the two memories, such as with disk operations. But note that data transfer between the main and auxiliary memories for memory regions consisting of the zero page, the $100 page, and the banked memory areas is not possible through a soft-switch. This data must be transferred by use of software that moves the data around to a transfer-permitted area. This procedure takes time and has been avoided with the addition of a switchable hardware memory offset in the PS IIe.

When the 6502 BIOS either writes to or reads from the location $C07E, any of four hardware functions on the PS IIe are activated or deactivated. The hardware functions are determined by what is on the data bus at the time of the access to the location $C07E. If data bit B2 is set, the hardware on the PS IIe translates all addresses sent to it from the Apple IIe motherboard. The translation is to EXCLUSIVE-OR the incoming address on the address bus with the value $8000. For example, if the address offset switch is set, then the 6502 BIOS reading from the location $7350 in the auxiliary memory will actually be reading the location $F350 on the PS IIe. The same offset applies to writing. The previous example would correspondingly have the data written to location $F350 when the 6502 BIOS was thinking it was writing to $7350 in the auxiliary memory. By using this address offset switch, you can transfer data directly between memory regions inaccessible before. The 6502 BIOS uses a routine such as the one shown below to transfer data from the auxiliary memory to the main memory.

```
      STA $C003    ;Enable the auxiliary memory read.
      LDA #$05     ;Set bit B2 on the data bus.
      STA $C07E    ;Offset the auxiliary memory bus.
      LDY #$0B     ;Set loop index.
MOVE: LDA $7395,Y  ;Load accumulator with the data from the table
                   ;in the auxiliary memory starting at the
                   ;address $F395.
      STA $131B,Y  ;Store the data in a table in the main memory
                   ;starting at $131B.
      DEY          ;Decrement the loop counter.
      BNE MOVE     ;Loop until the table is filled.
      LDA #01      ;Clear bit B2 on the data bus.
      STA $C07E    ;Disable auxiliary memory offset.
      STA $C002    ;Enable the main memory read.
```

Table 14.2 gives the hardware function associated with the values on the data bus at the time of an access to the $C07x location. The B0 bit is arbitrary. The Apple IIe will accept any value for B0 and give identical results. Microsoft arbitrarily uses $C07E.

The BIOS transfers information between the auxiliary and main memories through the use of the INTVEC (discussed above), the use of the main memory locations $44 through $4B (ZPM memory), and the use of the auxiliary memory locations 44H through 4BH (ZPA memory). Whenever the Z-80 calls the 6502, the ZPA memory is transferred to the ZPM memory. When the 6502 returns control to the Z-80, it always transfers the ZPM memory to the ZPA memory. There are seventeen standard functions the Z-80 BIOS can request the 6502 BIOS to perform. The standard functions require that the memory locations 45H through 4BH be initialized in a prescribed way. After the 6502 call, these latter locations will contain prescribed data. Please notice that the 44H location is not used in the standard 6502 BIOS calls. This location is reserved for transmitting the console status byte. A description of the standard 6502 BIOS calls follows. Any discrepancies between the following and the PS IIe manual should be resolved in favor of this text. To use the standard functions, first do the memory initialization, and then perform

```
CALL 40H
```

which executes the standard 6502 BIOS function.

Table 14.2 ■ $C07x Hardware Functions: Data Bus Value versus Hardware Function

Bit	*Set*	*Cleared*
B6	Do not halt Z-80	Halt Z-80
B2	Offset memory to PS IIe by XOR $8000	Clear PS IIe memory offset
B1	Isolate auxiliary memory buses from main memory	Remove auxiliary memory bus isolation
B0	Do not reset Z-80	Reset Z-80

■ PS IIe Standard 6502 BIOS Calls

PS IIe standard 6502 BIOS calls are as follows:

Function	*Location*	*Initial Value*	*Returned Value*
CALLSUB	49H	00H	N/A
	45H	[A] assignment	[A] last
	46H	[X] assignment	[X] last
	47H	[Y] assignment	[Y] last
	48H	N/A	[S] last
	4AH	6502 routine address, low byte	N/A
	4BH	6502 routine address, high byte	N/A

Note that the CALLSUB function does a JSR to the routine at the address found in 4AH and 4BH. The Apple monitor ROM is enabled. The 6502 registers are defined as follows:

[A]	accumulator
[X]	X register
[Y]	Y register
[S]	status register

Function	*Location*	*Initial Value*	*Returned Value*
READMEM Read 6502 memory	49H	01H	N/A
	45H	N/A	Data byte read
	46H	N/A	N/A
	47H	N/A	N/A
	48H	N/A	N/A
	4AH	6502 memory address, low byte	N/A
	4BH	6502 memory address, high byte	N/A

Function	*Location*	*Initial Value*	*Returned Value*
WRITEMEM Write data to 6502 memory	49H	02H	N/A
	45H	Data to be written	N/A
	46H	N/A	N/A
	47H	N/A	N/A
	48H	N/A	N/A
	4AH	6502 memory address, low byte	N/A
	4BH	6502 memory address, high byte	N/A
READSEC Read a diskette sector to a buffer	49H	03H	N/A
	45H	Track number 00H to 22H	Error code: 00H = no error, 10H = write-protect, other = I/O error
	46H	Drive 01H or 02H	N/A
	47H	Drive slot	N/A
	48H	Sector number 00H to 0FH	N/A
	4AH	Z-80 I/O buffer address, low byte	N/A
	4BH	Z-80 I/O buffer address, high byte	N/A

Note that READSEC is the PS IIe version of a call to the CPM RWTS.

Function	*Location*	*Initial Value*	*Returned Value*
WRITESEC	49H	04H	N/A
Write a buffer to a diskette sector	45H	Track number 00H to 22H	Error code: 00H = no error, 10H = write-protect, other = I/O error
	46H	Drive 01H or 02H	N/A
	47H	Drive slot	N/A
	48H	Sector number 00H to 0FH	N/A
	4AH	Z-80 I/O buffer address, low byte	N/A
	4BH	Z-80 I/O buffer address, high byte	N/A

Note that WRITESEC is the PS IIe version of a call to the CPM RWTS.

Function	*Location*	*Initial Value*	*Returned Value*
READSLOT	49H	05H	N/A
Read data from a recognized card in a slot	45H	N/A	Data byte read
	46H	N/A	N/A
	47H	Slot number 1 to 7	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A

Note that if the card is not recognized in the slot addressed, no read is performed. The routine returns without doing anything, and the value in 45H is arbitrary.

Function	*Location*	*Initial Value*	*Returned Value*
WRITESLOT Write data to a recognized card in a slot	49H	06H	N/A
	45H	Data byte to be written	N/A
	46H	N/A	N/A
	47H	Slot number 1 to 7	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A

Note that if the card is not recognized in the slot addressed, no write is performed. The routine returns without doing anything.

Function	*Location*	*Initial Value*	*Returned Value*
STATSLOT Get status of a recognized card in a slot	49H	07H	N/A
	45H	N/A	Status: 00H = not ready to read, FFH = ready to read
	46H	N/A	N/A
	47H	Slot number 1 to 7	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A

Note that if the card is not recognized in the slot addressed, the routine returns without doing anything.

Function	*Location*	*Initial Value*	*Returned Value*
INITSLOT	49H	08H	N/A
Initialize a recognized card in a slot	45H	N/A	N/A
	46H	N/A	N/A
	47H	Slot number 1 to 7	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A

Note that if the card is not recognized in the slot addressed, the routine returns without doing anything.

Function	*Location*	*Initial Value*	*Returned Value*
WSTART	49H	09H	N/A
Load CCP from diskette in drive A:	45H	N/A	N/A
	46H	N/A	N/A
	47H	N/A	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A
FORMAT	49H	0AH	N/A
Format a diskette	45H	N/A	Error code: 00H = no error, 10H = write-protect, other = I/O error
	46H	Drive 01H or 02H	N/A
	47H	Drive slot	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A

Function	*Location*	*Initial Value*	*Returned Value*
UPDATE Move the keyboard redefinition and software and hardware screen function tables to 6502 BIOS	49H	0BH	N/A
	45H	N/A	N/A
	46H	N/A	N/A
	47H	N/A	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A
BEEP Perform the GBASIC BEEP	49H	0CH	N/A
	45H	Tone duration	N/A
	46H	Tone period	N/A
	47H	N/A	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A
CLEAR Perform the GBASIC G1 command	49H	0DH	N/A
	45H	N/A	N/A
	46H	N/A	N/A
	47H	N/A	N/A
	48H	N/A	N/A
	4AH	Byte written to even screen address	N/A
	4BH	Byte written to odd screen address	N/A

Function	*Location*	*Initial Value*	*Returned Value*
INVERT Invert the GBASIC high-resolution screen	49H	0EH	N/A
	45H	N/A	N/A
	46H	N/A	N/A
	47H	N/A	N/A
	48H	N/A	N/A
	4AH	N/A	N/A
	4BH	N/A	N/A
SETPT1 Set GBASIC high-resolution graphics point 1	49H	0FH	N/A
	45H	N/A	N/A
	46H	XOR mask	N/A
	47H	AND mask	N/A
	48H	N/A	N/A
	4AH	Screen address, low byte	N/A
	4BH	Screen address, high byte	N/A
SETPT2 Set GBASIC high-resolution screen	49H	10H	N/A
	45H	Byte to XOR with screen byte	N/A
	46H	N/A	N/A
	47H	N/A	N/A
	48H	N/A	N/A
	4AH	Screen address, low byte	N/A
	4BH	Screen address, high byte	N/A

■ The 6502 BIOS

The 6502 BIOS will now be described. This BIOS resides in the main memory.

The Low Memory Segment of the 6502 BIOS

The low memory segment of the 6502 BIOS is as follows:

$C40–$C76 The memory and screen initialization routine used during the cold boot.

$C77–$C80 The routine that highlights the cursor position.

$C81–$C8C The loop that waits for the Z-80 to call the 6502. The print spooler is called during the looping.

$C8D–$CB5 The 6502 command processor. The Z-80 is halted. The ZPA memory is moved to the ZPM memory. The command is decoded and executed.

$CB6–$CB8 The entry point to the WSTART command.

$CB9–$CE4 The command exit processor. The ZPM memory is transferred to the ZPA memory. The Z-80 is restarted. The processor waits for the Z-80 to zero its A register and perform the LD I,A instruction.

$CE5–$D06 The command vector table.

$D07–$D1F The CALLSUB routine.

$D20–$D28 The WRITEMEM routine.

$D29–$D1F The READMEM routine.

$D30–$D32 The entry to the READSEC routine.

$D33–$D43 The WRITESEC routine.

$D44–$D4C The FORMAT routine.

$D4D–$D65 The location where the parameters for the RWTS are set.

$D66–$D87 The READSLOT routine. If the slot is slot 3, then use the PS IIe keyboard routine and keyboard redefinition table.

$D88–$DD6 The WRITESLOT routine. If the slot is slot 3, then use the PS IIe screen write routine if there is no terminal card in slot 3.

$DD7–$E06 The write routine if there is a terminal card in slot 3.

$E07–$E6B The screen function processor. Tests are made for hardware and software screen functions and processed as required.

$E6C–$E72 The entry to the INITSLOT and STATSLOT routines.

$E73–$E9F A routine used by the READSLOT, WRITESLOT, STATSLOT, and INITSLOT routines. The card type is checked, and the routine address is taken from a table. After the proper routine is chosen, a jump is made to that routine.

$EA0–$EDE The vector table used by the preceding routine.

$EDF–$EF2 The firmware card initialization routine.

$EF3–$F00 The firmware card write routine.

$F01–$F0E The firmware card read routine.

$F0F–$F17 The firmware card status routine.

$F18–$F23 The routine that waits for the firmware card to be ready for I/O.

$F24–$F2E The serial card write routine.

$F2F–$F3A The serial card initialization routine.

$F3B–$F4B The communications card status routine.

$F4C–$F53 The console status routine.

$F54–$F58 The serial card status routine.

$F59–$F5B The routine called to return the status-unready byte.

$F5C–$F62 The firmware status routine.

$F63–$F65 The routine called to return the status byte.

$F66–$F6F The parallel card status routine.

$F70–$F88 The routine called to set the card parameters for I/O.

$F89–$F95 The communications card read routine.

$F96–$FA9 The communications card write routine.

$FAA–$FBF The parallel card write routine.

$FC0–$FE2 The routine to store LST: device output in the print spooler buffer.

$FE3–$FF0 The routine to initialize the communications card ACIA to 8 data bits, no parity bit, 1 stop bit, and sixteen times the baud rate.

$FF1–$FFF The console input routine, which reads the keyboard input buffer.

$1000–$1014 The routine to get a character from the print spooler buffer.

$1015–$1064 The console general input function. The keyboard is polled for input, which is stored in the keyboard buffer. The print spooler buffer is read, and the data is written to the LST: device. This routine is used before the routine at $FF1.

$1065–$1067 The routine to load $1A or the end-of-file marker, used when a read is made to a slot with an unrecognized card.

$1068–$1074 The routine to sound the Apple monitor bell.

$1075–$1078 The table of screen control characters.

$1079–$10F2 The console output routine. The print spooler is active. This routine when active controls the Z-80 directly without calling the 6502 command processor. The console input status is passed to the Z-80 BIOS through the $44 memory location. There is a simple screen output routine incorporated that does not process control characters. All of this increases the speed of the console routine.

$10F3–$10F6 The routine that places the cursor at the upper left screen corner.

$10F7–$10FB The routine that places the cursor at column 0 of the current line.

$10FC–$1103 The routine that pushes the cursor down to the next line. The column remains unchanged.

$1104–$114E The screen scroll-up routine. The keyboard is polled for input.

$110F–$115F The routine that enables the correct 80-column screen memory page as a function of the cursor's position.

$1160–$116C The location where the pointer to the screen row memory location is determined with the aid of a screen memory map table.

$116D–$119C The 80-column screen memory map table.

$119D–$11B6 A table of screen function addresses used by the routine at $E07.

$11B7–$11BD The routine to move the cursor one position to the left.

$11BE–$11C4 The routine to home the cursor and clear the screen.

$11C5–$11CE The routine to clear the screen from the current line to the screen bottom.

$11CF–$11EB The routine to clear the current line and poll the keyboard for input.

$11EC–$1202 The routine to clear the screen from the current cursor position to the bottom of the screen.

$1203–$121A The routine to clear the line to the cursor's right.

$121B–$121D The routine to set the screen output inverse mask.

$121E–$1222 The routine to clear the screen output inverse mask.

$1223–$1229 The routine to move the cursor up to the next line.

$122A–$1233 The routine to move the cursor right one position.

$1234–$124C The BEEP routine.

$124D–$1269 The INVERT routine.

$126A–$1292 The SETPT1 routine.

$1293–$1298 The SETPT2 routine.

$1300–$130C The character redefinition used by the 6502 BIOS. This is the image of the table in the Z-80 BIOS.

$130D–$131B The image of the Z-80 hardware screen function table.

$131C–$132B The image of the Z-80 software screen function table.

$132C–$1333 The image of the Z-80 Slot Type Table.

$1400–$14FF The keyboard input buffer.

$2000–$BFFF The print spooler buffer.

Before I describe the remainder of the 6502 BIOS, it is useful to insert some detail into the PS IIe cold boot procedure that was not included in chapter 10. The first PS IIe cold boot stage is to load the track 0, sector 0 data into the $800 memory page. The second stage is to jump to the routine on page $800, which loads the 6502 BIOS into the banked memory starting at $D100 and ending at $E1FF. A jump is then made to the third boot stage, which moves the data from the memory range $DA00–$E1FF to the memory range $C00–$13FF. The third stage then identifies the cards in the slots and temporarily stores the results in a table starting at $E12B. The Z-80 CCP, BDOS, and BIOS are loaded into the auxiliary memory with the aid of the warm boot loader routine. The warm boot loader is then modified to load the CCP only. The zero-page Z-80 BIOS vectors and data are then transferred to the auxiliary memory, and the Z-80 is told to reset. The remaining vectors and data tables are then moved to their lower memory locations, and control is given to the Z-80.

The High Memory Segment of the 6502 BIOS

The high memory segment of the 6502 BIOS is as follows:

$D000–$D1FF A nibble buffer used by the RWTS routine.

$D200–$D1BD Routines used by the RWTS routine.

$D1BE–$D1CB The checksum routine used for slot identification.

$D1CC–$D1DC The routine used to move memory segments.

$D1DD–$D1E2 The values used by the third-stage boot to initialize the 6502 reset and interrupt vector addresses.

$D1E3–$D1EF The table of signature bytes used to identify the cards found in the slots.

$D200–$D37B Routines used by the RWTS routine.

$D37C–$D3E2 The routine used by the third-stage boot to load the Z-80 CCP, BDOS, and BIOS. The tables and vectors are moved to their respective locations.

$D400–$D4FE Nibble buffers used by the RWTS routine.

$D500–$D68A More routines used by the RWTS routine.

$D68B–$D699 The sector skew table used by the RWTS.

$D69A–$D6A4 The location where the RWTS parameters are stored, that is, track, sector, drive, and so on.

$D6A5–$D6EF The entry into the RWTS routine. The sector buffer is located in the auxiliary memory.

$D6F0–$D6FF A table used by the disk-formatting routine.

$D700–$D912 The disk-formatting routine.

$D913–$D97C The WSTART routine. This is the warm loader routine.

$D97D–$D986 The routine that tests location $2000 and is used by the routines that see if the Z-80 is calling.

$D987–$D9FD The entry to the third-stage boot. The slots are checked, and the cards are identified. The routine exits with a jump to $D37C.

$D9FE–$DA39 The UPDATE routine.

■ The Z-80 BIOS

The Z-80 BIOS is as follows:

FA00H–FA32H The BIOS jump vectors.

FA33H–FA92H The disk parameter headers for four drives. There is space for six drives, but the remaining two drives are not used.

FA93H–FAA1H The disk parameter block.

FAA2H–FAAEH The routine that initializes all the cards using the 6502 BIOS function.

FAAFH–FAEEH The warm boot routine. The stack pointer is initialized. The cards are initialized. The zero-page CP/M vectors are reset. The WSTART 6502 BIOS routine is called. The DMA is reset to 80H. The routine exits to the CCP.

FAEFH–FAF2H CONST The routine to get the address of the console status routine from the IOCB at F3C0H and jump to that routine.

FAF3H–FAF8H CONIN The console input routine. A call to the routine at FB10H is made to check the input device, and a call to the correct input routine is made.

FAF9H The routine that places the contents of the A register in C and falls into the CONOUT routine.

FAFAH–FB0BH CONOUT The routine that checks the IOBYTE for the output device and then jumps to the selected routine. If the output goes to the logical LST: device, then the physical LPT: device is used.

FB0CH–FB0FH The routine to output a character to the LPT: device.

FB10H–FB1DH The character input routine, which checks the IOBYTE for the physical input device and goes to the selected routine.

FB1EH–FB21H The routine to get input from the TTY: device.

FB22H–FB25H A jump to the physical PTR: device. This routine may be used by the console input or logical RDR: device.

FB26H–FB34H LIST The logical LST: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

FB35H–FB46H PUNCH The logical PUN: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

FB47H–FB55H READER The logical RDR: device routine, which checks the IOBYTE for the physical device. A jump is then made to the selected physical device.

FB56H–FB66H The lead-in routine for output to the console. The character is checked to see if it is a control character. If it is a control character, the routine outputs the character through the WRITESLOT 6502 BIOS routine. Noncontrol characters cause the INTVEC to be loaded with FFH, which will cause the 6502 BIOS to output the character through its special routine. The routine exits with a jump to FB72H.

FB67H–FB71H The routine used to get the console status. A check is made if 44H contains a value other than AAH. An AAH indicates that the last 6502 BIOS call didn't check the slot status. If there is another value, it will be the console input status, and it will return to the caller. The AAH value will cause a call to the 6502 STATSLOT routine.

FB72H–FB75H The entry to the console output routine. The E register is loaded with 3 for slot 3, and the 6502 BIOS is called.

FB76H–FB79H The entry to the console input routine. The E register is loaded with 3 for slot 3, and the 6502 BIOS is called.

FB7AH–FB7DH The entry to the punch output routine. The E register is loaded with 2 for slot 2, and the 6502 BIOS is called.

FB7EH–FB83H The entry to the reader input routine. The E register is loaded with 2 for slot 2, and the 6502 BIOS is called.

FB84H–FB88H The entry to the list output routine. The E register is loaded with 1 for slot 1, and the 6502 BIOS is called.

FB89H–FBB4H The routine that calls the 6502 BIOS and is used by the Z-80 BIOS routines. The Z-80 registers are assigned the zero-page addresses used by the 6502 BIOS. The data is sent to the 6502 BIOS from these registers, and the returning 6502 BIOS data is placed in these registers. The register assignments are:

A	45H
B	46H
E	47H
D	48H
C	49H
L	4AH
H	4BH

The value found in INTVEC is used for the call to the 6502 BIOS. After the return from the 6502 BIOS call, the INTVEC is loaded with 80H.

FBB5H–FBBCH The standard routine to call the 6502 BIOS. This routine is the one jumped by a call to 40H.

FBBDH–FBCCH The CP/M logical sector skew table. The table relates the 256-byte sector number to the logical 128-byte sector number used by CP/M.

FBCDH–FBD5H A computational routine used by the disk I/O routines to get needed memory addresses. The HL register is incremented by sixteen times the A register.

FBD6H–FBE0H HOME A disk routine to select track 0.

FBE1H–FBE5H SETTRK A disk routine to select the track in register C.

FBE6H–FBEAH SETSEC A routine to select the 128-byte CP/M sector.

FBEBH–FBEFH SETDMA A routine to select the disk I/O buffer address.

FBF0H–FC19H SELDSK A routine to select the disk drive and set flags to notify the disk I/O routines if the drive has been changed or a nonexistent drive was called.

FC1AH–FC1EH The routine called when an error was found in SELDSK.

FC1FH–FC2EH READ The disk read operation, set up according to all the CP/M protocols. See chapter 10.

FC2FH–FC76H WRITE The disk write operation, performed according to the CP/M protocols.

FC77H–FD0BH A routine used by both READ and WRITE to make sure that the CP/M protocols are met. A sector skew is done according to the CP/M sector skew table. The data is moved to or from the CP/M RWTS buffer at FF00H. The read or write operation is then called.

FD0CH–FD2FH The location where the actual read and write routines are performed by the calling of the 6502 CP/M RWTS functions.

FD30H–FDFFH The area used for allocation tables, the directory buffer, and data storage required by the BIOS.

FE00H–FEFFH The I/O patch area.

The value found in INTVEC is used for the call to the 6502 BIOS. After the return from the 6502 BIOS call, the INTVEC is loaded with 80H.

FBB5H–FBBCH The standard routine to call the 6502 BIOS. This routine is the one jumped by a call to 40H.

FBBDH–FBCCH The CP/M logical sector skew table. The table relates the 256-byte sector number to the logical 128-byte sector number used by CP/M.

FBCDH–FBD5H A computational routine used by the disk I/O routines to get needed memory addresses. The HL register is incremented by sixteen times the A register.

FBD6H–FBE0H HOME A disk routine to select track 0.

FBE1H–FBE5H SETTRK A disk routine to select the track in register C.

FBE6H–FBEAH SETSEC A routine to select the 128-byte CP/M sector.

FBEBH–FBEFH SETDMA A routine to select the disk I/O buffer address.

FBF0H–FC19H SELDSK A routine to select the disk drive and set flags to notify the disk I/O routines if the drive has been changed or a nonexistent drive was called.

FC1AH–FC1EH The routine called when an error was found in SELDSK.

FC1FH–FC2EH READ The disk read operation, set up according to all the CP/M protocols. See chapter 10.

FC2FH–FC76H WRITE The disk write operation, performed according to the CP/M protocols.

FC77H–FD0BH A routine used by both READ and WRITE to make sure that the CP/M protocols are met. A sector skew is done according to the CP/M sector skew table. The data is moved to or from the CP/M RWTS buffer at FF00H. The read or write operation is then called.

FD0CH–FD2FH The location where the actual read and write routines are performed by the calling of the 6502 CP/M RWTS functions.

FD30H–FDFFH The area used for allocation tables, the directory buffer, and data storage required by the BIOS.

FE00H–FEFFH The I/O patch area.

FF00H–FFFFH The BOOT routine. This is used by the cold boot to initialize the stack pointer and zero-page vectors and data areas to their cold boot values. The routine is used only once. For this reason, this area is used for the CP/M RWTS I/O buffer.

■ The PS IIe Diskette Map

The PS IIe diskette map is as follows:

Track 00H, sector 00H The boot 1 sector.

Track 00H, sector 01H through track 01H, sector 01H
The 6502 BIOS.

Track 01H, sector 01H through track 01H, sector 09H
The CCP.

Track 01H, sector 0AH through track 02H, sector 07H
The BDOS.

Track 02H, sector 08H through track 02H, sector 0DH
The Z-80 BIOS.

15 Uploading and Downloading

The term *uploading* is used to describe the sending of data from your computer to another computer. The term *downloading* is the term for receiving data into your computer from another computer. It is sometimes desirable to transfer a file to or from another computer that doesn't use Apple-compatible diskettes.

For instance, you may wish to use a software package that is available only on a noncompatible 8-inch disk. There are two easily applied methods of transferring the program to an Apple diskette.

The first method is to use PIP to do the file transferring. To use PIP, the Apple must be connected to a CP/M-compatible computer (the host computer) that can read the non-Apple disk. The connection must be made between the serial port in the host computer and the serial port in the Apple. The Apple's serial port is the serial card placed in slot 2. I recommend that the slot 2 card be a communications-type card. After the connection is made and both computers are booted, the procedure to transmit the file from the host computer to the Apple is to enter the following into the Apple at the CP/M prompt:

```
A>PIP A:YOURFILE.DOC=RDR:
```

Then enter the following after the CP/M prompt in the host computer:

```
A>PIP PUN:=A:MYFILE.TXT
```

The file called MYFILE found on drive A: will be sent to the punch device in the host computer. The Apple will create a file on drive A: called YOURFILE.DOC, and the data received from the reader device will be written into YOURFILE.DOC. Other drives may be specified, and all the appropriate PIP options apply (see chapter 6).

The second method requires the use of a telephone *modem*. *Modem* stands for *modulator/demodulator*. Simply put, a modem is a device that transforms a serial stream of voltage data pulses into electrical frequencies (modulation). These frequencies can then be transmitted by telephone to another modem that translates the electrical frequencies back into a serial stream of voltage data pulses (demodulation). Computers use the voltage data pulses to transmit serial data.

To download a file through a modem requires a modem for your Apple, a modem for the host computer, and a telephone line. The advantage over the first method is that the computers do not have to be in the same room or even in the same city. The disadvantage is that a modem is generally more expensive than a serial port, and there is now an additional telephone bill. You will also require the use of a modem program for the Apple. Such programs are available commercially, or they may be obtained at little or no cost from CP/M users' groups. All the modem programs will be able to download a file from the host computer. The use of modems for transmitting files is by far the most popular method.

There is yet one more method for reading files from a noncompatible disk. This method is the most expensive, but it is also the most direct. Interface cards are available for the Apple that will control drives other than the Apple 5¼-inch drives. The interface card I recommend is the one that will control an 8-inch drive. The addition of 8-inch drives to the Apple will increase the storage capacity of the Apple by as much as twenty times. The CP/M interfaces for the 8-inch drives can often run up to three 8-inch disk formats. The IBM 3740 format is the standard CP/M disk format. With the interface card and an 8-inch drive, the Apple can then read a standard 8-inch CPM disk. It is then possible to transfer a file from that disk to one of the Apple's 5¼-inch drives. The latter method requires the acquisition of an interface card and at least one 8-inch drive. Before making any purchases, shop carefully and compare what is available. There is a wide variation in the quality of the 8-inch drive interfaces available for the Apple. Some are much more flexible and easy to use than others.

Appendix A: Binary Numbers

The Definition of a Binary Number

This appendix is intended to be a very brief explanation of the binary number system. Please see any book on computer methods for a more detailed discussion.

Binary numbers are most easily understood in terms of the standard decimal numbers. *Decimal* is based on a Latin word meaning *tenth*. A decimal number is a symbolic way of dividing a quantity into multiples of 10. Consider the number 5,423; it can be written as

$$5423 = (5 \times 1000) + (4 \times 100) + (2 \times 10) + (3 \times 1)$$

which may be expressed in terms of powers of 10 as

$$5423 = (5 \times 10^3) + (4 \times 10^2) + (2 \times 10^1) + (3 \times 10^0)$$

Recall that any number raised to the 0 power is 1.

Instead of dividing 5,423 into multiples of 10, let's divide it into multiples of 2. We will then get

$$\begin{aligned}5423 = {} & (1 \times 4096) + (0 \times 2048) + (1 \times 1024) + (0 \times 512) \\ & + (1 \times 256) + (0 \times 128) + (0 \times 64) + (1 \times 32) + (0 \times 16) \\ & + (1 \times 8) + (1 \times 4) + (1 \times 2) + (1 \times 1)\end{aligned}$$

which may be expressed in terms of powers of 2 as

$$\begin{aligned}5423 = {} & (1 \times 2^{12}) + (0 \times 2^{11}) + (1 \times 2^{10}) + (0 \times 2^9) + (1 \times 2^8) \\ & + (0 \times 2^7) + (0 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) \\ & + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)\end{aligned}$$

The interesting thing about expressing a number in powers of 2 is that the coefficient of each power is either 1 or 0. Numbers expressed as powers of 2 are called *binary* numbers, and there are only two values for the coefficients. Numbers expressed as powers of 10 have ten values for coefficients: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The shorthand method of writing a decimal number is to use the place of the digit in the number sequence to identify which power of 10 is to be multiplied by that digit. Thus, for the number 5,423 we understand that the 3 is multiplied by 1, the 2 is multiplied by 10, and so on. We can use the same scheme to create a shorthand notation for a binary number. Translating decimal 5,423 into binary gives

$$5423_{10} = 1010100101111_2$$

The subscripts indicate which representation is being used. The 10 indicates that the base of the number system is 10, while the 2 indicates that the base of the number system is 2. The base-2 representation is read to mean that, going from right to left, the first digit is to be multiplied by 1, the second digit is to be multiplied by 2, the third digit by 4, and so on.

The binary system is used by computers because of their internal construction. Each binary digit is called a *bit* (the shortened form of *binary digit*). A binary representation can be rather lengthy; therefore, another shorthand has evolved. Binary numbers are frequently formed into groups of 4 bits called *nibbles* (*nibble* is an acronym for nothing; somebody's humor simply took charge). Regrouped, our example becomes

$$5423_{10} = 0001{,}0101{,}0010{,}1111_2$$

Leading zeroes have been included here to form a neater package. Each nibble can take on sixteen values ranging from 0 to 15. Let's change the representation yet again.

■ The Hexadecimal Number

Consider going to a number system whose base is 16. Our example becomes

$$5423_{10} = (1 \times 16^3) + (5 \times 16^2) + (2 \times 16^1) + (15 \times 16^0)$$

It is desirable to use a single digit for a coefficient. The base-16 representation can have coefficients with values as large as decimal 15. We will need to create a new set of symbols for the coefficients in a base-16 representation. The symbols used are

$0_{16} = 0_{10}$
$1_{16} = 1_{10}$
$2_{16} = 2_{10}$
$3_{16} = 3_{10}$
$4_{16} = 4_{10}$
$5_{16} = 5_{10}$
$6_{16} = 6_{10}$
$7_{16} = 7_{10}$

$8_{16} = 8_{10}$
$9_{16} = 9_{10}$
$A_{16} = 10_{10}$
$B_{16} = 11_{10}$
$C_{16} = 12_{10}$
$D_{16} = 13_{10}$
$E_{16} = 14_{10}$
$F_{16} = 15_{10}$

Following the procedure used before, we can rewrite the example as

$5423_{10} = 152F_{16}$

The base-16 number system is called the *hexadecimal* number system. The notation in this book follows the mnemonic convention used for the 8080 and Z-80 microprocessors; hexadecimal numbers are witten with an H suffix, while decimal numbers are written with no suffix. Our oveworked example becomes

5423 = 152FH

The *byte*, by the way, is made up of two *nibbles* (pun intended). The number 152FH consists of the 2 bytes 15H and 2FH. The byte is useful to computer programmers because memory locations in most computers, Apple included, hold 1 byte (8 bits) of data.

It is useful to know how to convert hexadecimal numbers to decimal numbers. The conversion is quite simple. All we need do is use the definitions. Consider converting 2AH into decimal. Now

$$\begin{aligned} 2AH &= [(2 \times 16) + (10 \times 1)]_{10} \\ &= 42_{10} \end{aligned}$$

Let's convert C24BH into decimal. As before:

$$\begin{aligned} C24BH &= [(12 \times 4096) + (2 \times 256) + (4 \times 16) + (11 \times 1)]_{10} \\ &= 49739_{10} \end{aligned}$$

■ Converting Decimal to Hexadecimal Notation

Converting a decimal number to hexadecimal notation is a bit more tedious, but again the conversion comes from the definitions. Let's convert decimal 58,892 into hexadecimal notation. First determine how many times 58,892 is divisible by 4,096.

$58892/4096 = 14 + (1548/4096)$

The first hexadecimal digit is E. The numerator of the fractional part remaining after a division is called the *modulus*. The modulus in this example is 1,548. Now divide the modulus by 256.

1548/256 = 6 + (12/256)

The second hexadecimal digit is 6. Take the modulus of this result, which is 12, and divide it by 16.

12/16 = 0 + (12/16)

The third hexadecimal digit is 0. The modulus of this result is the fourth hexadecimal digit, which is C. The final result is

58892_{10} = E60CH

More Definitions

Computer jargon includes the term *kilobyte*. A *kilobyte* is 1,024 bytes, that is, 2^{10} bytes. This can become a little confusing since memory and disk capacity are often referred to in terms of multiples of kilobytes, or Ks. For instance, a 64K computer such as the Apple is said to have 64 kilobytes, but 64K is actually 65,536 bytes.

There is a standard method for labeling the bits in a byte. The most significant bit (MSB)—that is, the leftmost bit—is numbered B7. The bits are then numbered sequentially in descending order. For example, consider the byte A3H, which has the bit representation

1	0	1	0	0	0	1	1
B7	B6	B5	B4	B3	B2	B1	B0

The number of each bit is shown below the respective bit. The least significant bit (LSB) is called B0. If the bit value is 1, the bit is said to be *set*. If the bit value is 0, the bit is said to be *cleared*. Our example has B7 set, B6 cleared, B5 set, and so on.

Finally, the last bit of binary jargon: the term *page* refers to 256 bytes of memory. These 256 bytes always start at a *page boundary*. A *page boundary* is an address starting at an integral multiple of 100H. For example, the zero page includes the addresses beginning with 0000H and ending with 00FFH, page 1 includes the memory beginning with 0100H and ending with 01FFH, and so on. Four pages equal 1 kilobyte.

Appendix B: ASCII

ASCII stands for American Standard Code for Information Interchange. The code is used to represent certain printable and nonprintable characters. The code assigns a number to each character. There are 128 characters for the number range 0 through 127. The nonprintable characters are often called *control characters* because computer peripheral equipment uses them to control certain functions. Although the control characters are technically nonprintable, some terminals and printers may actually display some of them. The table below gives the ASCII assignments. The table lists control functions as well. No attempt is made here to define the functions. You will have to refer to your equipment manual for that information.

Keyboard	*Control Function*	*Decimal*	*Hexadecimal*
CONTROL-@	NULL	0	00
CONTROL-A	SOH	1	01
CONTROL-B	STX	2	02
CONTROL-C	ETX	3	03
CONTROL-D	EOT	4	04
CONTROL-E	ENQ	5	05
CONTROL-F	ACK	6	06
CONTROL-G	BELL	7	07
CONTROL-H	Backspace	8	08
CONTROL-I	Horizontal tab	9	09
CONTROL-J	Line feed	10	0A
CONTROL-K	Vertical tab	11	0B
CONTROL-L	Form feed	12	0C
CONTROL-M	Carriage return	13	0D
CONTROL-N	SO	14	0E
CONTROL-O	SI	15	0F
CONTROL-P	DLE	16	10
CONTROL-Q	DC1	17	11
CONTROL-R	DC2	18	12
CONTROL-S	DC3	19	13
CONTROL-T	DC4	20	14

Keyboard	*Control Function*	*Decimal*	*Hexadecimal*
CONTROL-U	NAK	21	15
CONTROL-V	SYN	22	16
CONTROL-W	ETB	23	17
CONTROL-X	CAN	24	18
CONTROL-Y	EM	25	19
CONTROL-Z	SUB	26	1A
ESC	Escape	27	1B
CONTROL-[	Same as ESC		
CONTROL-\	FS	28	1C
CONTROL-]	GS	29	1D
CONTROL-^	RS	30	1E
CONTROL-_	US	31	1F
space	N/A	32	20
!	N/A	33	21
"	N/A	34	22
#	N/A	35	23
$	N/A	36	24
%	N/A	37	25
&	N/A	38	26
'	N/A	39	27
(	N/A	40	28
)	N/A	41	29
*	N/A	42	2A
+	N/A	43	2B
,	N/A	44	2C
–	N/A	45	2D
.	N/A	46	2E
/	N/A	47	2F
0	N/A	48	30
1	N/A	49	31
2	N/A	50	32
3	N/A	51	33
4	N/A	52	34
5	N/A	53	35
6	N/A	54	36
7	N/A	55	37
8	N/A	56	38
9	N/A	57	39
:	N/A	58	3A
;	N/A	59	3B
<	N/A	60	3C
=	N/A	61	3D

Keyboard	*Control Function*	*Decimal*	*Hexadecimal*
>	N/A	62	3E
?	N/A	63	3F
@	N/A	64	40
A	N/A	65	41
B	N/A	66	42
C	N/A	67	43
D	N/A	68	44
E	N/A	69	45
F	N/A	70	46
G	N/A	71	47
H	N/A	72	48
I	N/A	73	49
J	N/A	74	4A
K	N/A	75	4B
L	N/A	76	4C
M	N/A	77	4D
N	N/A	78	4E
O	N/A	79	4F
P	N/A	80	50
Q	N/A	81	51
R	N/A	82	52
S	N/A	83	53
T	N/A	84	54
U	N/A	85	55
V	N/A	86	56
W	N/A	87	57
X	N/A	88	58
Y	N/A	89	59
Z	N/A	90	5A
[	N/A	91	5B
\	N/A	92	5C
]	N/A	93	5D
^	N/A	94	5E
_	N/A	95	5F
`	N/A	96	60
a	N/A	97	61
b	N/A	98	62
c	N/A	99	63
d	N/A	100	64
e	N/A	101	65
f	N/A	102	66
g	N/A	103	67

Keyboard	*Control Function*	*Decimal*	*Hexadecimal*
h	N/A	104	68
i	N/A	105	69
j	N/A	106	6A
k	N/A	107	6B
l	N/A	108	6C
m	N/A	109	6D
n	N/A	110	6E
o	N/A	111	6F
p	N/A	112	70
q	N/A	113	71
r	N/A	114	72
s	N/A	115	73
t	N/A	116	74
u	N/A	117	75
v	N/A	118	76
w	N/A	119	77
x	N/A	120	78
y	N/A	121	79
z	N/A	122	7A
{	N/A	123	7B
|	N/A	124	7C
}	N/A	125	7D
~	N/A	126	7E
DELETE	DEL	127	7F

Note that all the characters given are available on the Apple IIe keyboard. Some of the characters are not on the Apple II keyboard but are available with certain keystroke combinations if there is an 80-column card installed. Refer to your manual for the appropriate information.

Appendix C: A Primer on Diskettes

The Physical Diskette

The disk drive's operation is in many ways similar to the operation of a tape recorder. There is a read/write head, which is used to take information from or place information on a magnetic medium. The tape recorder generally keeps its read/write head stationary while the tape passes over it. The disk drive uses a rotating magnetic diskette that passes over the read/write head. Information is stored on or retrieved from the diskette that passes beneath the head. Since the disk drive's read/write head has a small area, only a very narrow circular track is used when the head is in a given position. Most diskettes are considerably larger than the read/write head, which means that a diskette has room for much more than one track. In order to read from or write to other disk tracks, the read/write head can move radially.

Since a read/write head can move radially in a random fashion, there must be means by which the computer knows whether the head is on a given track or in between tracks. In fact, *formatting* a diskette amounts to writing onto the diskette magnetic signatures that can be read by the computer to tell it where the read/write head is located. Each track has its own individual number, which is imbedded in the track's magnetic signature. Thus, if the computer wishes to find track 3, it now knows how to recognize it.

A diskette track has room to store a very large amount of data. Dividing the tracks into equal segments called *sectors* has been found to increase the efficiency of data storage. Sectors are defined on the tracks by magnetic signatures, with each sector having its own number. The sectors are created when the disk is formatted. The computer can now retrieve data from or store data to a particular sector on a given track. This ability to read from or write to a given sector on a given track is called *random access*.

The Apple diskette has thirty-five tracks, which are numbered 00H through 22H. The track nearest the outside edge of the diskette is numbered 00H; the track nearest the inside edge is numbered 22H. The Apple diskette has sixteen sectors per track. The sectors are numbered

00H through 0FH. Each sector area is preceded by a magnetic code that identifies the sector and track number. The sector area is followed by more code designating the sector's end.

■ Sector Skewing

The read/write head travels radially to access a given track. When the chosen sector passes beneath the head, the magnetic code is either read from or written to the rotating disk. Data must be encoded before it can be written to the diskette. By the same token, the diskette data must be decoded before it can be used by the computer. Since encoding and decoding take time, a scheme is needed to optimize diskette I/O. The scheme is called *sector skewing*.

Sector skewing, also called *sector interleaving*, is a method to reduce the time of diskette I/O. Significant time is required after a sector is read to translate the encoded diskette nibbles and store the results in memory. If the sectors are sequential, the diskette will pass the next sector to be read so that there is one diskette revolution for each sector reading. If the sector sequence is changed so that, when the computer has completed storing the data from the previous read, the read/write head is over the next sector to be read, more than one sector can be read per revolution. Sector skewing also reduces the time it takes to write data to a diskette. Skewing permits the read/write head to be over the next sector to be written before the diskette makes one revolution.

Apple diskettes are formatted with each 256-byte physical sector located sequentially on the diskette. The sector skewing is done in software. This means that if you ask DOS 3.3 for sector 01H, you will actually be reading physical sector 0DH. The CP/M RWTS (Read Write Track Sector) subroutine will give you physical sector 02H if you ask for sector 01H.

Apple CP/M track numbers are identical to DOS 3.3 track numbers, but the sector numbering needs some elaboration. There are thirty-two logical sectors on each track. A CP/M logical sector is 128 bytes. This means that there are two logical sectors in each physical sector. The BIOS sorts this mess out so that every time CP/M requests a logical sector the proper half of the proper physical sector is read into the DMA (Direct Memory Access).

The CP/M sector skewing in the Apple is actually a double skewing. When a logical sector is requested, the BIOS translates the logical sector to half of a CP/M physical sector. The CP/M RWTS routine then reads the CP/M physical sector into a buffer and moves the proper half to the DMA. The even-numbered logical sector corresponds to the first 128 bytes of the physical sector, while the odd-numbered logical sector corresponds to the last 128 bytes. To complicate things more, the CP/M physical sector is not the Apple physical sector. The CP/M physical sector is skewed so that when the CP/M RWTS requests reading sector 01H, the Apple physical sector 02H is read. Table C.1 lists the sector relations.

The reason the Apple has a double sector skew is that the Apple system tracks are skewed according to the CP/M physical sector skew, while the data tracks use the logical sector skew. The CP/M physical sector skew provides the fastest relation for reading a sector sequence. This means that a cold or warm boot can be performed in the least time possible. The logical sector is a compromise for getting the fastest sector read skew in conjunction with the fastest sector write skew.

■ The CP/M Diskette

The Apple CP/M diskette reserves the first three tracks to hold the boot routine, which loads the CCP (Console Command Processor), BDOS (Basic Disk Operating System), and BIOS (Basic Input Output System), which are also contained there. The first three tracks are called the *system tracks*. Track 03H (recall that track numbering starts at 00H) contains the CP/M directory. Only six physical sectors on track 03H hold the

Table C.1 ■ Apple Diskette Sector Skew Relations

CP/M Logical Sectors	*CP/M Physical Sectors*	*DOS 3.3 Sectors*	*Apple Physical Sectors*
00, 01	0	0	0
02, 03	9	6	3
04, 05	3	C	6
06, 07	C	3	9
08, 09	6	9	C
0A, 0B	F	F	F
0C, 0D	1	E	2
0E, 0F	A	5	5
10, 11	4	B	8
12, 13	D	2	B
14, 15	7	8	E
16, 17	8	7	1
18, 19	2	D	4
1A, 1B	B	4	7
1C, 1D	5	A	A
1E, 1F	E	1	D

directory. The directory sectors have the CP/M logical values 00H through 0BH. The remaining tracks and sectors are available for file storage.

The Microsoft versions 2.23 and higher use a trick to allow the system tracks to be used for data storage. A file called cp/m.sys is created in user area 31. This user area is inaccessible from the CCP, so this directory entry is normally unseen by the user. The file cp/m.sys is a dummy file whose space occupies the system tracks. The BIOS is written to recognize the system tracks as accessible data areas. Since the directory entry cp/m.sys says that the system tracks are already occupied, the system tracks will not be overwritten by any new directory entries. The version 2.23 COPY.COM program has an option to create a data diskette. The data diskette is one in which the cp/m.sys directory entry has been erased. This creates three more tracks for file storage. Of course, it is impossible to boot a data diskette, but it is safe to use it in any drive other than drive A:.

Appendix D: The CP/M RWTS (Versions 2.20B and 2.23)

The Microsoft BIOS (Basic Input Output System) does the actual disk access through the 6502 microprocessor. The CP/M RWTS (Read Write Track Sector) routine is located at $E03 for CP/M versions 2.20B and 2.23. The CP/M RWTS routine is part of the BIOS Disk I/O Handlers routine. The Disk I/O Handlers routine is mainly in Z-80 code and is responsible for reading and writing the 128-byte sectors used by CP/M. Since the Apple diskette has 256-byte sectors, the CP/M RWTS routine is used for diskette I/O. The CP/M RWTS routine uses an I/O buffer at memory location $800. The diskette sector is read into that buffer, and the contents of that buffer are written to the diskette. The BIOS Disk I/O Handlers routine reads from or writes to the appropriate half of that buffer to simulate the 128-byte CP/M sector disk I/O.

The CP/M RWTS may be directly accessed by the suitable calling procedure to the 6502 as described in chapter 10. The following areas must be initialized before the CP/M RWTS can be used:

$3E0 Place the track to be accessed in here. The Apple track numbers range from $00 through $22.

$3E1 Place the CP/M physical sector to be accessed in here. The Apple sector numbers range from $0 through $F. The sector skew for CP/M physical sectors is used; see Appendix C.

$3E2 and $3E3 These locations are holdovers from the DOS 3.3 RWTS and were used for volume numbers. The CP/M RWTS doesn't use volume numbers, so $00 may be placed in these locations.

$3E4 The drive is placed here. The DOS 3.3 numbers are used, so that this location should contain either a $1 or $2.

$3E5 This is another holdover from DOS 3.3. The last drive used is put into this location.

$3E6 The drive's slot number times sixteen is placed here. If the slot is slot 6, then $60 goes into this location.

$3E7 This is the last slot accessed. The number returns as a multiple of sixteen. If the last slot was slot 6, then $60 is placed in this location.

$3E8 and $3E9 The I/O buffer address is stored in the order low byte, high byte. If the buffer is at $800, the $3E8 contains $00, and $3E9 contains $08.

$3EA The error code is returned in this location:

$00	no error
$10	write-protect error
$40	drive error; that is, a read or write was impossible

$3EB The command code goes into this location:

$01	read sector to I/O buffer
$02	write I/O buffer to sector

CP/M version 2.23 always reinitializes the I/O buffer address to $800 before using the CP/M RWTS. CP/M version 2.20B doesn't reinitialize the I/O buffer address. It is up to the programmer to see that the I/O buffer address is correct for the CP/M RWTS.

The warm boot routine calls a program that reads the diskette by using the CP/M RWTS directly. The sectors containing the CCP (Console Command Processor) and BDOS (Basic Disk Operating System) are quickly read from the diskette to their place in memory below the BIOS. The warm loader routine is located at $E00 for CP/M versions 2.20B and 2.23. The CP/M 2.23 60K version reads the sectors starting at track $0, sector $B and ending with track $2, sector $8 to the memory starting at D300H. The CP/M 2.20B 56K version reads the sectors starting at track $0, sector $B and ending with track $2, sector $0 to the memory starting at C400H.

Index

"This book is for *Apple* CP/M users, an incredibly welcome relief for those of us who have the more general reference books on the shelf. . . . I would recommend it. Definitely."
—Paul Mithra, GTS, Laser Publishing Analyst

>$12.95

Gain an in-depth understanding of the CP/M operating system with **The Apple CP/M Book**. Designed for Apple II, IIe, and II Plus users, this book guides you through CP/M's structure, commands, and utilities.

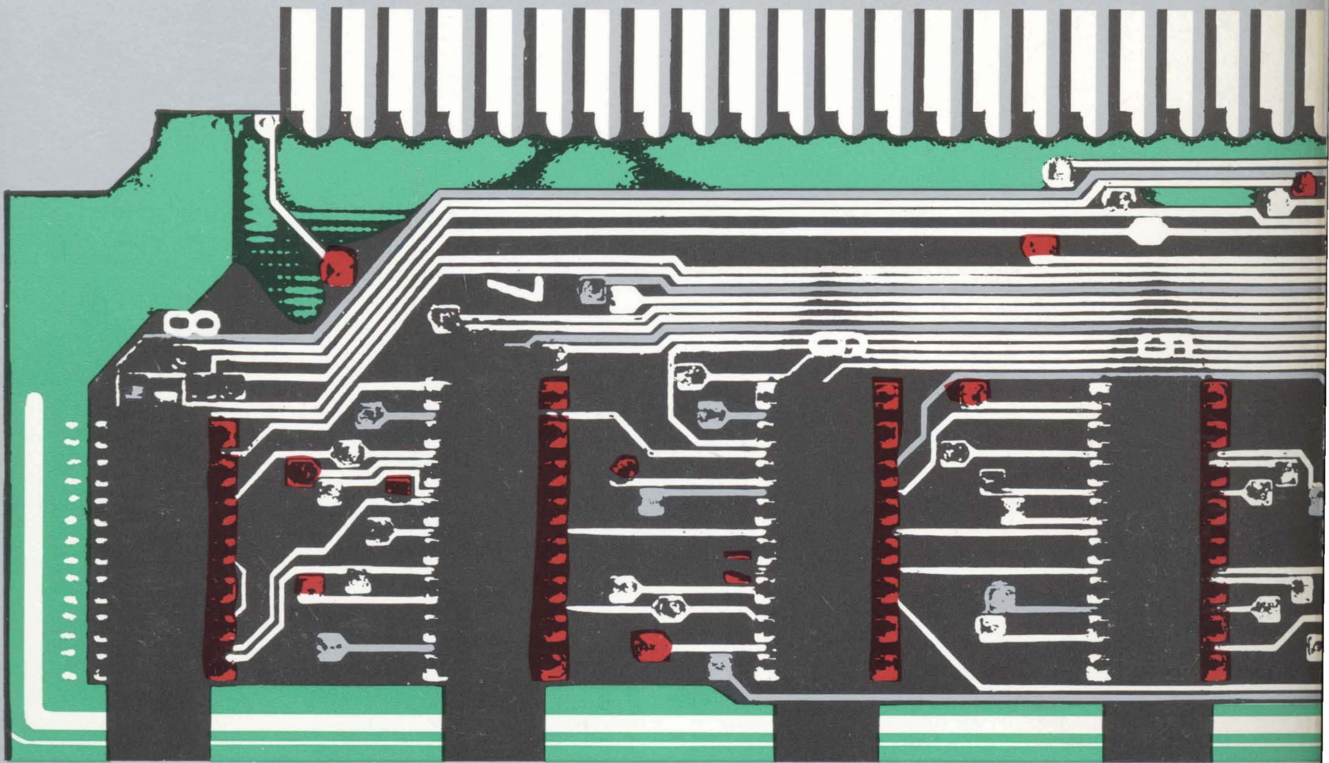
This useful reference manual covers the fundamentals *and* the fine points of CP/M operations, including

- the major CP/M utilities—STAT, PIP, ED, SUBMIT, and DDT
- the Basic Disk Operating System (BDOS) and its routines
- all four versions of the Basic Input Output System (BIOS) from Microsoft
- suggestions for debugging the Microsoft BIOS
- a helpful summary of CP/M commands

and more.

The Apple CP/M Book also explains CP/M's advanced features, including solid examples and a wealth of technical detail. If you want to use the full capabilities of CP/M on your Apple, you need this book.

Murray Arnow serves as a CP/M consultant to A.P.P.L.E., the largest Apple users' group in the United States. He is also a contributing author to *Call-A.P.P.L.E.* magazine and a software consultant. A resident of Chicago, Illinois, Dr. Arnow holds a Ph.D. in Physics.



Scott, Foresman and Company

ISBN 0-673-18068-9