MICROSOFT® SOFTCARD™ II

System

Installation and Operations Manual and Programmer's Manual

For Apple® II, II+, and IIe

MICROSOFT LICENSE AGREEMENT

CAREFULLY READ ALL THE TERMS AND CONDITIONS OF THIS AGREEMENT PRIOR TO BREAKING THE DISKETTE SEAL. BREAKING THE DISKETTE SEAL INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS.

If you do not agree to these terms and conditions, return the unopened diskette package and the other components of this product to the place of purchase and your money will be refunded. No refunds will be given for products which have opened diskette packages or missing components.

1. LICENSE: You have the non-exclusive right to use the enclosed program. This program can only be used on a single computer. You may physically transfer the program from one computer to another provided that the program is used on only one computer at a time. You may not electronically transfer the program from one computer to another over a network. You may not distribute copies of the program or documentation to others. You may not modify or translate the program or related documentation without the prior written consent of Microsoft.

YOU MAY NOT USE, COPY, MODIFY, OR TRANSFER THE PROGRAM OR DOCUMENTATION, OR ANY COPY EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT.

2. BACK-UP AND TRANSFER: You may make one (1) copy of the program solely for back-up purposes. You must reproduce and include the copyright notice on the back-up copy. You may transfer and license the product to another party if the other party agrees to the terms and conditions of this Agreement and completes and returns a Registration Card to Microsoft. If you transfer the program you must at the same time transfer the documentation and back-up copy or transfer the documentation and destroy the back-up copy.

3. COPYRIGHT: The program and its related documentation are copyrighted. You may not copy the program or its documentation except as for back-up purposes and to load the program into the computer as part of executing the program. All other copies of the program and its documentation are in violation of this Agreement.

4. TERM: This license is effective until terminated. You may terminate it by destroying the program and documentation and all copies thereof. This license will also terminate if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy all copies of the program and documentation.

5. HARDWARE COMPONENTS: Microsoft product hardware components only include circuit cards and the mechanical mouse.

6. LIMITED WARRANTY: THE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE PROGRAM IS ASSUMED BY YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND NOT MICROSOFT OR ITS DEALERS) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. FURTHER, MICROSOFT DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF, OR THE RESULTS OF THE USE OF, THE PROGRAM IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE; AND YOU RELY ON THE PROGRAM AND RESULTS SOLELY AT YOUR OWN RISK.

Microsoft does warrant to the original licensee that the diskette(s) on which the program is recorded be free from defects in materials and workmanship under normal use and service for a period of ninety (90) days from the date of delivery as evidenced by a copy of your receipt. Microsoft warrants to the original licensee that the hardware components included in this package are free from defects in materials and workmanship for a period of one year from the date of delivery to you as evidenced by a copy of your receipt. Microsoft's entire liability and your exclusive remedy shall be replacement of the diskette or hardware component not meeting Microsoft's limited warranty and which is returned to Microsoft with a copy of your receipt. If failure of the diskette or hardware component has resulted from accident, abuse or misapplication of the product, then Microsoft shall have no responsibility to replace the diskette or hardware component under this Limited Warranty. In the event of replacement of the hardware component the replacement will be warranted for the remainder of the original one (1) year period or 30 days, whichever is longer.

THE ABOVE IS THE ONLY WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE THAT IS MADE BY MICROSOFT ON THIS MICROSOFT PRODUCT. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

NEITHER MICROSOFT NOR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THIS PROGRAM SHALL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES ARISING OUT OF THE USE, THE RESULTS OF USE, OR INABILITY TO USE SUCH PRODUCT EVEN IF MICROSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR CLAIM. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

7. UPDATE POLICY: In order to be able to obtain updates of the program, the licensee and persons to whom the program is transferred in accordance with this Agreement must complete and return the attached Registration Card to Microsoft. IF THIS REGISTRATION CARD HAS NOT BEEN RECEIVED BY MICROSOFT, MICROSOFT IS UNDER NO OBLIGATION TO MAKE AVAILABLE TO YOU ANY UPDATES EVEN THOUGH YOU HAVE MADE PAYMENT OF THE APPLICABLE UPDATE FEE.

8. MISC.: This license agreement shall be governed by the laws of the State of Washington and shall inure to the benefit of Microsoft Corporation, its successors, administrators, heirs and assigns.

9. ACKNOWLEDGEMENT: YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT THIS AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF AGREEMENT BETWEEN THE PARTIES AND SUPERCEDES ALL PROPOSALS OR PRIOR AGREEMENTS, VERBAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN THE PARTIES RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

Should you have any questions concerning this Agreement, please contact in writing Microsoft, Customer Sales and Service, 10700 Northup Way, Bellevue, WA 98004.

Microsoft®
SoftCard™ II

for Apple® II, II Plus, and *//e* Computers

Installation and Operation Manual

Microsoft Corporation

If you have comments about this documentation or the enclosed software, complete the Software Problem Report at the back of this manual and return it to Microsoft.

Part No. 028-023-006
Document No. 8820-22X-00

Preface

Your Microsoft® SoftCard™ II system is a valuable addition to your Apple®][,][Plus, or *//e* computer. It is your key to running the many programs and languages available under the CP/M® operating system. The SoftCard II system is easy to install and use. In addition to the standard CP/M utility programs, it includes several utility programs written by Microsoft and the Microsoft BASIC Interpreter, so you can create your own programs.

Important

Before you break the seal on the disk envelope, you must read the Microsoft License Agreement in the Customer Service Plan booklet. Also before you break the seal on the disk envelope, read the Digital Research License Agreement included with your Customer Service Plan booklet. If you agree with the terms of the agreement, fill out the Microsoft Product Registration Card and mail it to Microsoft immediately.

About the Microsoft SoftCard Manuals

The SoftCard system is documented in four manuals: the *Microsoft SoftCard II Installation and Operation Manual*, the *Microsoft BASIC Interpreter Reference Manual*, the *Osborne® CP/M User Guide*, and the *Microsoft SoftCard II Programmer's Manual*.

The *Microsoft SoftCard II Installation and Operation Manual* introduces the SoftCard II package. It also describes how to install the SoftCard II circuit board, how to load and use the CP/M operating system, and how to use CP/M built-in commands and certain transient programs. It should be read before installing the SoftCard II circuit board and the software.

For programmers who want to connect nonstandard I/O devices or use software requiring modifications to CP/M, the *SoftCard II Programmer's Manual* contains the necessary information. It is also a reference manual for utility programs, commands, and CP/M system calls. This manual can be obtained from Microsoft by sending in your Microsoft Product Registration card.

The *Microsoft BASIC Interpreter Reference Manual* explains how to use Microsoft BASIC and provides a reference for all of the commands, statements, and functions contained in Microsoft BASIC. This manual is intended for users who already know how to program in BASIC. If you are new to BASIC, see the list of recommended reading for more information about programming in BASIC.

If you are new to the CP/M operating system, the *Osborne CP/M User Guide* will teach you how to use the CP/M built-in commands and transient programs. It will guide you step-by-step through the different functions of CP/M.

How to Use This Manual

This is your SoftCard II system owner's manual; it shows you how to install and operate your SoftCard II system. It is organized so you can find the information you want quickly and easily.

The chapters are organized as follows:

Chapter 1, "Introduction," introduces the SoftCard II system and lists the syntax notation used in SoftCard II documentation.

Chapter 2, "Installation," describes what is needed to install the SoftCard II circuit board and how to do it. This chapter also tells you what other accessory boards are compatible with the SoftCard II.

Chapter 3, "Getting Started," tells you how to load CP/M and lists the procedures for making backup copies of your SoftCard II Master disk.

Chapter 4, "An Introduction to CP/M," introduces the CP/M operating system and describes the role of an operating system within the computer.

Chapter 5, "Using CP/M With the Apple Computer," describes how CP/M works with the Apple *//e* computer. Includes descriptions of the special features of the SoftCard II system.

Chapter 6, "CP/M Commands and Utility Programs," explains how to use CP/M commands and describes the transient programs you will use most often.

Digital Research License Information

Our license with Digital Research for the CP/M operating system requires that each purchaser of the SoftCard with CP/M register with Microsoft Corporation so that records can be maintained of all CP/M owners. This requirement is made by Digital Research, not by Microsoft. A post card is enclosed for reply. The serial number on the card is the number stamped on the disk labels. Before signing the card and returning it to Microsoft, read the software license agreement below carefully.

Software License Agreement

Important: All Digital Research programs are sold only on the condition that the purchaser agrees to the following license. READ THIS LICENSE CAREFULLY. If you do not agree to the terms contained in this license, return the packaged disk UNOPENED to your distributor and your purchase price will be refunded. If you agree to the terms contained in this license, fill out the REGISTRATION information and RETURN by mail to MICROSOFT CORPORATION.

DIGITAL RESEARCH agrees to grant and the Customer agrees to accept on the following terms and conditions non-transferable and nonexclusive licenses to use the software program(s) (Licensed Programs) herein delivered with this agreement.

Term

This agreement is effective from the date of receipt of the above-referenced program(s) and shall remain in force until terminated by the Customer upon one month's prior written notice, or by Digital Research, as provided below.

Any license under this Agreement may be discontinued by the Customer at any time upon one month's prior written notice. Digital Research may discontinue any license or terminate this Agreement if the Customer fails to comply with any of the terms and conditions of this Agreement.

License

Each program license granted under this Agreement authorizes the Customer to use the Licensed Program in any machine readable form on any single computer system (referred to as System). A separate license is required for each System on which the Licensed Program will be used.

This Agreement and any of the licenses, programs, or materials to which it applies may not be assigned, sublicensed or otherwise transferred by the Customer without prior written consent from Digital Research. No right to print or copy, in whole or in part, the Licensed Programs is granted except as hereinafter expressly provided.

Permission to Copy or Modify Licensed Programs

The Customer shall not copy, in whole or part, any Licensed Programs which are provided by Digital Research in printed form under this Agreement. Additional copies of printed materials may be acquired from Digital Research.

Any Licensed Programs which are provided by Digital Research in machine readable form may be copied, in whole or in part, in printed or machine readable form in sufficient number for use by the Customer with the designated System, to understand the contents of such machine readable material, to modify the Licensed Program as provided below, for back-up purposes, provided, however, that no more than five (5) printed cpies will be in existence under any license at any one time without prior written consent from Digital Research. The Customer agrees to maintain appropriate records of the number and location of all such copies of Licensed Programs. The original, and any copies of the Licensed Programs, in whole or in part, which are made by the Customer shall be the property of Digital Research.

This does not imply, of course, that Digital Research owns the media on which the Licensed Programs are recorded. The Customer may modify any machine readable form of the Licensed Programs for his own use and merge it into other program material to form an updated work, provided that, upon discontinuance of the license for such Licensed Program, the Licensed Program supplied by Digital Research will be completely removed from the updated work. Any portion of the Licensed Program included in an updated work shall be used only if on the designated System and shall remain subject to other terms of this Agreement.

The Customer agrees to reproduce and include the copyright notice of Digital Research on all copies, in whole or in part, in any form, including partial copies of modifications, of Licensed Programs made hereunder.

Protections and Security

The Customer agrees not to provide or otherwise make available any Licensed Program including but not limited to program listings, object code, and source code, in any form, to any person other than Customer or Digital Research employees, without prior written consent from Digital Research, except with the Customer's permission for purposes specifically related to the Customer's use of the Licensed Program.

Discontinuance

Within one month after the date of discontinuance of any license under this Agreement, the Customer will furnish Digital Research a certificate certifying that through his best effort, and to the best of his knowledge, the original and all copies, in whole or in part, in any form, including partial copies in modifications, of the Licensed Program received from Digital Research or made in connection with such have been destroyed, except upon written authorization from Digital Research, the Customer may retain a copy for archive purposes.

Disclaimer of Warranty

Digital Research makes no warranties with respect to the Licensed Programs. The sole obligation of Digital Research shall be to make available all published modifications or updates made by Digital Research to Licensed Programs which are published within one (1) year from date of purchase, provided Customer has returned the Registration Card delivered with the Licensed Program.

Limitation of Liability

THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL DIGITAL RESEARCH BE LIABLE FOR CONSEQUENTIAL DAMAGES EVEN IF DIGITAL RESEARCH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

General

If any of the provisions, or portions thereof, of this Agreement are invalid under any applicable statute or rule of law, they are to that extent to be deemed omitted.

Contents

Chapter 1
Introduction

The Microsoft SoftCard II system is a hardware and software product that greatly enhances the capabilities of your Apple][or *//e* computer. SoftCard hardware adapts your computer for the CP/M operating system, which comes with the SoftCard. In addition, the SoftCard II package includes an extra 64K bytes of memory, CP/M utility programs, and Microsoft BASIC Interpreter.

Hardware

The circuit board that you receive in your SoftCard II package is actually three circuit boards in one. It combines the functions of a Z80® coprocessor board and a 64K memory expansion board.

The combination of two functions on one circuit board saves the remaining accessory slots in the Apple computer for other purposes.

The coprocessor section of the SoftCard II circuit board contains a Z80 microprocessor with the interface circuitry necessary for communicating with the Apple I/O bus. A coprocessor is an additional microprocessor which shares control of the computer. Thus, when you install the SoftCard II system into your Apple, you are really creating two computers from one: a 6502 computer that will run Apple DOS programs, and a Z80 computer that will run CP/M programs.

The memory section of the SoftCard II circuit board contains 64K bytes of RAM (Random Access Memory) for use by the Z80 microprocessor. The memory section permits large application programs (up to 59K bytes) to run under CP/M.

Once the SoftCard II circuit board has been installed, you can operate your Apple computer in either 6502 mode (using the 6502 microprocessor) or CP/M mode (using the Z80 microprocessor). When you are in 6502 mode, the SoftCard will not affect the operation of your Apple. In CP/M mode, you can run any CP/M-based program or language, including the Microsoft BASIC Interpreter.

Software

The SoftCard II package includes the CP/M operating system (CP/M-80); the Microsoft BASIC Interpreter; and special transient programs to perform utility functions, such as copying disks and modifying CP/M to your particular system environment.

CP/M Operating System and Programs

The CP/M operating system is one of the most widely implemented 8-bit operating systems in use today. Because of its widespread use, an extensive library of high-level languages and application software is available for CP/M-based computers.

In addition to supporting a wide variety of software, CP/M offers many convenient features. These include the capability of implementing machine-language programs; faster disk I/O access; simple file transfer operations; and "wild card" file naming conventions that allow you to refer to multiple files with one name. The SoftCard version of CP/M also includes several programming tools for program development and utility programs for everyday operation.

Microsoft BASIC Interpreter

The Microsoft BASIC Interpreter is the most widely implemented BASIC in use today. In addition to the standard BASIC commands and statements, the SoftCard version of BASIC includes high-resolution graphics commands and statements. The *Microsoft BASIC Interpreter Reference Manual* describes how to use Microsoft BASIC.

SoftCard II Utility Programs

Utility programs perform certain time-consuming tasks, such as disk formatting and file transfer. In addition to the CP/M utility programs PIP and STAT, the SoftCard II system includes programs that allow you to:

Copy Apple DOS files to your CP/M disk with the APDOS program.

Create startup disks with the AUTORUN program.

Display an alphabetical list of disk files and other information with the CAT program.

Configure CP/M for specific I/O devices and programs with the CONFIGIO program.

Format and copy disks with the COPY program.

Copy files in a single-drive system with the MFT program.

Several other utility programs and programming tools are also included. These are listed in Chapter 6, "CP/M Commands and Utility Programs."

Command Line Notation

Before you can use the computer, you must learn how to communicate with it. Communication is more than just typing words on the keyboard. Instructions to the computer must be in a certain format to ensure the computer understands exactly what you want it to do.

Most instructions to the computer are in the form of *commands*. Commands often consist of a keyword and command line. The keyword is usually the name of the command that you want to execute, such as COPY. The command line that follows the keyword specifies what the command must do. For example,

```
COPY B:=A:
```

instructs the COPY program to copy the contents of the disk in drive A: to the disk in drive B:. Without the command line, COPY would not know what to copy.

In the SoftCard II documentation, special notation has been developed to show the differences between what you enter on the keyboard and what you see in the manual. The following notation is used in this manual to help you understand how commands are entered into the computer.

ital — Italics indicate information that you enter. Italicized lowercase text is for an entry that you must supply, such as a *filename*.

[ ] — Square brackets indicate that the enclosed entry is optional. In the "Examples," the /F and /V entries can be included at your discretion. They are not necessary to execute the command.

{ } Braces indicate a choice between two or more entries. At least one of the entries enclosed in braces must be chosen, unless the entries are also enclosed in square brackets. In the COPY example, you must type either the /S or the /D entry in the command line.

| Vertical bars separate choices within braces.

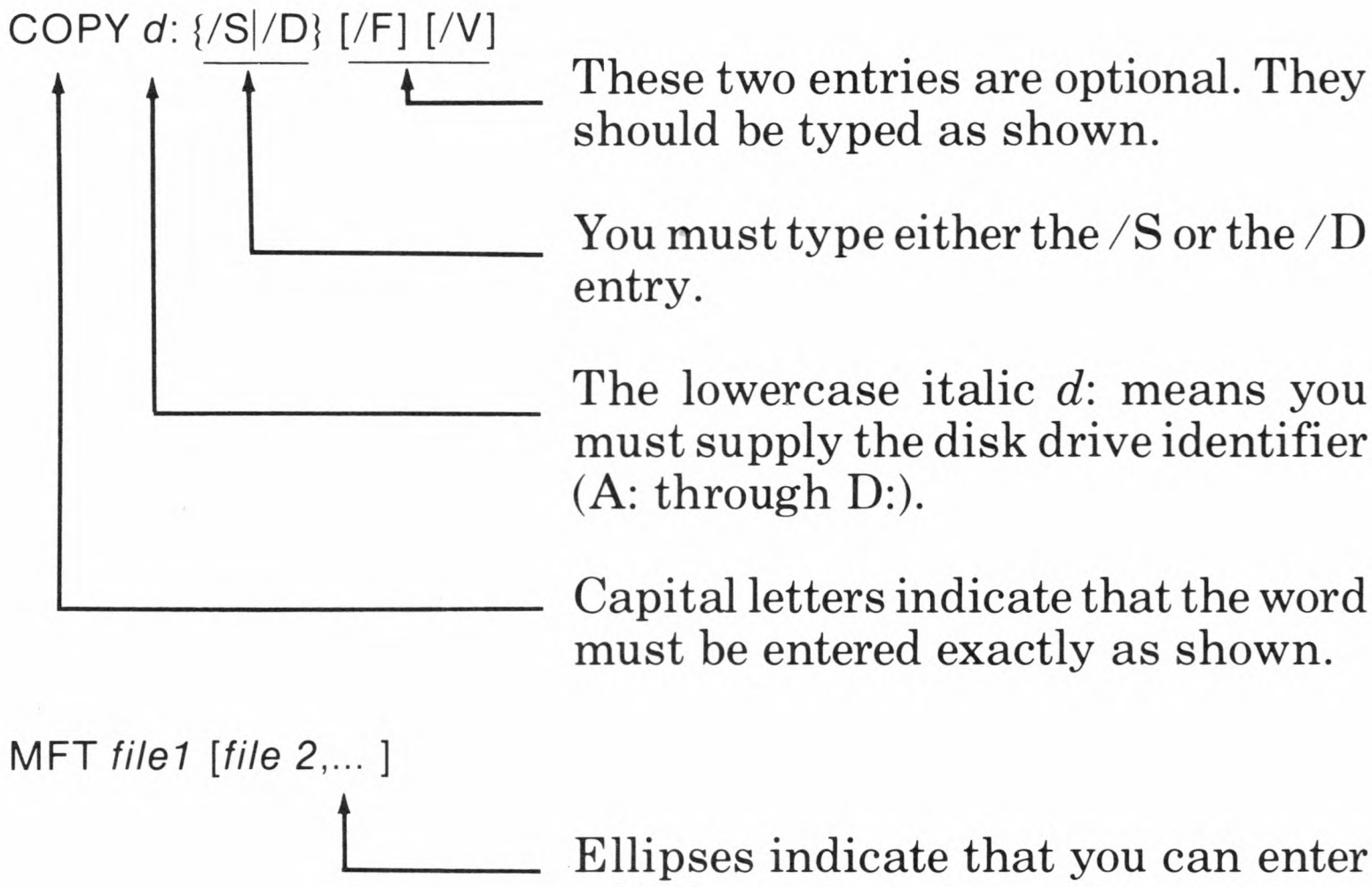
. . . Ellipses indicate that an entry can be repeated as many times as needed or desired. In the following MFT example, the ellipses indicate that you can include additional files in the command line.

CAPS Capital letters not enclosed in the other elements of syntax indicate portions of commands that must be entered exactly as shown, such as command keywords. Small capital letters indicate that you must press a key named by the text; for example, "press the RETURN key."

All other punctuation, such as commas, colons, slash marks, and equal signs, must be entered exactly as shown. In the COPY example, the / character must be included with the entry. Unless specified in the description of the command, spaces are optional. Spaces are usually shown in the documentation for clarity.

Examples

Command Line



Chapter 2
Installation

This chapter gives instructions for installing the SoftCard II circuit board. It also provides other information about setting up your SoftCard II system. We recommend reading this entire chapter before installing the SoftCard II circuit board.

Preliminary Information

Before installing the circuit board, make sure that your system meets all the criteria listed in the following "System Requirements" section and that your Apple computer is set up and operational as described in your Apple owner's guide.

System Requirements

To use your SoftCard II system successfully, you will need the following items:

1. An Apple][or *//e* computer with 64K of memory and 80-column display capability. For Apple][and][Plus computers, you also need an 80-column video display board.

2. A SoftCard II printed circuit board.

3. Disk drives—only one drive is needed, but two drives are recommended.

4. A screen monitor or an external terminal.

5. The SoftCard II CP/M Master disk.

6. At least three blank floppy disks.

Unpacking

Upon receipt of your SoftCard II package, check carefully for missing items or shipping damage. If any of the following items are damaged or missing, contact your computer dealer.

Your SoftCard II system should consist of the following:

The SoftCard II printed circuit board

The SoftCard II Master floppy disk, which includes the CP/M operating system and the following files:

APDOS.COM	DDT.COM	MFT.COM
ASM.COM	DUMP.COM	PATCH.COM
AUTORUN.COM	DUMP.ASM	PIP.COM
BOOT.COM	ED.COM	STAT.COM
CAT.COM	GBASIC.COM	SUBMIT.COM
CONFIGIO.BAS	LOAD.COM	XSUB.COM
COPY.COM		

The *Osborne CP/M User Guide*

A green Microsoft binder containing the following manuals:

Microsoft SoftCard II Installation and Operation Manual

Microsoft SoftCard II Programmer's Manual

Microsoft BASIC Interpreter Reference Manual

A Customer Service Plan

Disk drive labels

If reshipment of the SoftCard II system is necessary, contact Microsoft Corporation prior to returning the circuit board.

Safety and Handling Precautions

The following paragraphs contain some common-sense precautions you should be aware of before attempting installation.

Safety Precautions

Before installing the SoftCard II (or any other circuit board), you should know about the possible shock hazards that are present in any electronic device, including a personal computer. Although the Apple computer is designed to minimize the danger of electrical shock, it is prudent to exercise caution whenever the cover has been removed from the computer.

It is dangerous to open any electrical or electronic device while the power is on. Attempts to insert or remove circuit boards while power is on will usually result in damage to the board and to the computer. Use standard electrical safety precautions whenever the cover is off the computer.

Handling Precautions

The SoftCard II circuit board contains integrated circuits (called "ICs" or "chips") which can be damaged by electrostatic discharge during handling. "Handling" is defined as physically holding or moving the circuit board outside the computer. The only time you should be handling the circuit board is when you are installing it.

To decrease the chance of damaging the circuit board (and possibly voiding the warranty), follow these simple guidelines when handling the SoftCard II or any other circuit board.

1. Discharge any personal static electricity before handling accessory boards by touching the metal power supply chassis.

   Make sure that the power is turned off and the power cord is disconnected before touching the power supply.

2. Hold the circuit board by its edges to avoid contaminating it with oil from your hands. Be especially careful to avoid touching the gold edge-connector on the bottom of the board.

3. When handling circuit boards or ICs, avoid any contact with plastic, vinyl, and styrofoam. These substances can cause electrostatic buildup.

4. If you need to remove or replace ICs, keep handling to a minimum. Take the ICs from their containers only when necessary. Always use anti-static containers for storage. Also, always handle ICs by the casing instead of by the connector pins. This will minimize the electrostatic shock danger and will prevent possible damage to the pins.

Other Accessory Boards

The CP/M operating system requires accessory boards be installed into specific slots, depending on their intended use. For example, if you have a printer interface board, it should be installed in slot 1. This allows you to refer to the printer without specifying a slot number, as is necessary with Applesoft BASIC and Integer BASIC. The accessory slots are numbered from 1 through 7, as shown in Figure 2.1. (Slots are numbered 0 through 7 in the Apple][and][Plus computers.) To determine the correct slot for using other accessory boards with CP/M, read the following paragraphs.

Note

The accessory slot assignments for CP/M are exactly the same as those for Apple Pascal. Therefore, if you have your system configured for use with Apple Pascal, no rearrangement is necessary.

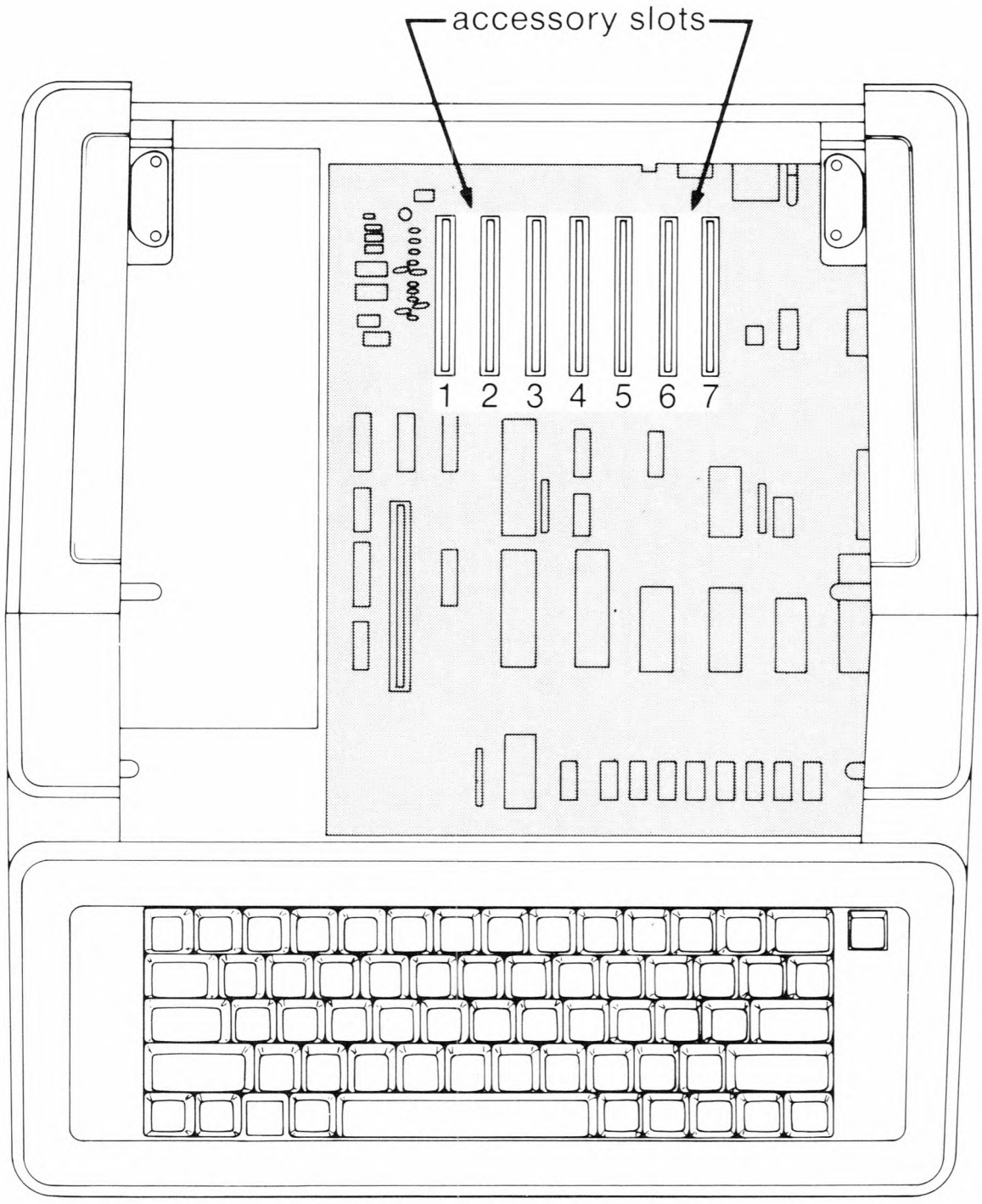


Figure 2.1. Apple *//e* Accessory Slots

Compatible Accessory Boards

Table 2.1 lists the accessory boards that are directly compatible with the SoftCard II system and CP/M. These boards, when installed in the appropriate Apple accessory slots, will work without any software modifications. (An underlined "x" indicates the recommended slot for installation.)

Table 2.1

Compatible Accessory Boards

	Assigned Slot							
Accessory Board	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Softcard II		x	x	x	x			x
Apple Disk][Controller						x	x	
Apple Communications Interface		x	x	x	x			
California Computer Systems® (CCS) 7710A™ Serial Interface*		x	x	x	x			x
Apple High Speed Serial Interface		x	x	x	x			x
Apple Silentype® Printer Interface		x	x	x	x			x
Apple Parallel Printer Card		x			x			x
Apple Firmware Card		x	x	x	x			x
Apple Super Serial Interface		x	x	x	x			x
Microsoft RAMCard (or other memory expansion boards for Apple][or][Plus computers)	x							

* The CCS 7710A Serial Interface is the preferred interface board for serial communications. It supports standard communication protocols and variable baud rates from 110 to 19200 baud.

There are some accessory boards not listed in Table 2.1 that are compatible with CP/M. As a general rule, any accessory board or I/O device that is directly compatible with the Apple Pascal operating system without requiring any software modifications will be compatible with CP/M as well. Other accessory boards or I/O devices can be used if the software supplied by the board manufacturer can be configured to your CP/M system with the CONFIGIO program. Instructions on how to use CONFIGIO are given in Chapter 6 of the *SoftCard II Programmer's Manual.*

Apple Disk Controller Boards

The SoftCard II version of CP/M communicates with a maximum of four disk drives. (Two disk drives can be connected to each Apple Disk][Controller board.) As indicated in Table 2.1, Apple Disk][Controller boards can be installed in slots 5 and 6. The first controller board must be installed in slot 6 and the second in slot 5.

Apple disk drives are designated with numbers. The first drive is drive 1, the second drive is drive 2, and so on. CP/M uses letters followed by a colon to identify the disk drive. The first drive (connected to the disk controller in slot 6) is always assigned as drive A:. Each sequential drive is assigned the next successive letter. For example, drive 2 is drive B:, and drive 3 (connected to the disk controller in slot 5) is drive C:.

General Purpose I/O Accessory Boards

Accessory boards for general purpose I/O devices (such as modems, paper-tape readers, and punches) must be installed in slot 2. Any board that is compatible with Apple Pascal or has accompanying interface software (from the board's manufacturer) is compatible with CP/M.

External Terminals

If you are using an external terminal and want to use CP/M, we recommend you use either a CCS 7710A Serial Interface board or a modified Apple Communications Interface board to connect the terminal to your SoftCard II system. The Apple High Speed Serial Interface board is compatible but is not recommended, since there is no way for CP/M to check the "status" of this device. The preferred accessory slot for the external interface board is slot 3.

Important

Before using an external terminal with CP/M, certain portions of CP/M must be modified with the CONFIGIO program. Do not install any circuit board into slot 3 until the I/O configuration process is completed. Once CP/M is set up for the external terminal, you may then install the interface board.

Circuit Board Installation Procedure

Use the following procedure to install the SoftCard II circuit board. We recommend that you first read all the instructions to acquaint yourself with the overall procedure. Then perform each step with care exactly as described.

The SoftCard can be installed in slots 1 through slot 7. Because CP/M does not use the slot for a specific purpose, we recommend installing the SoftCard in slot 4.

1. Set the Apple POWER switch to OFF (see Figure 2.2).

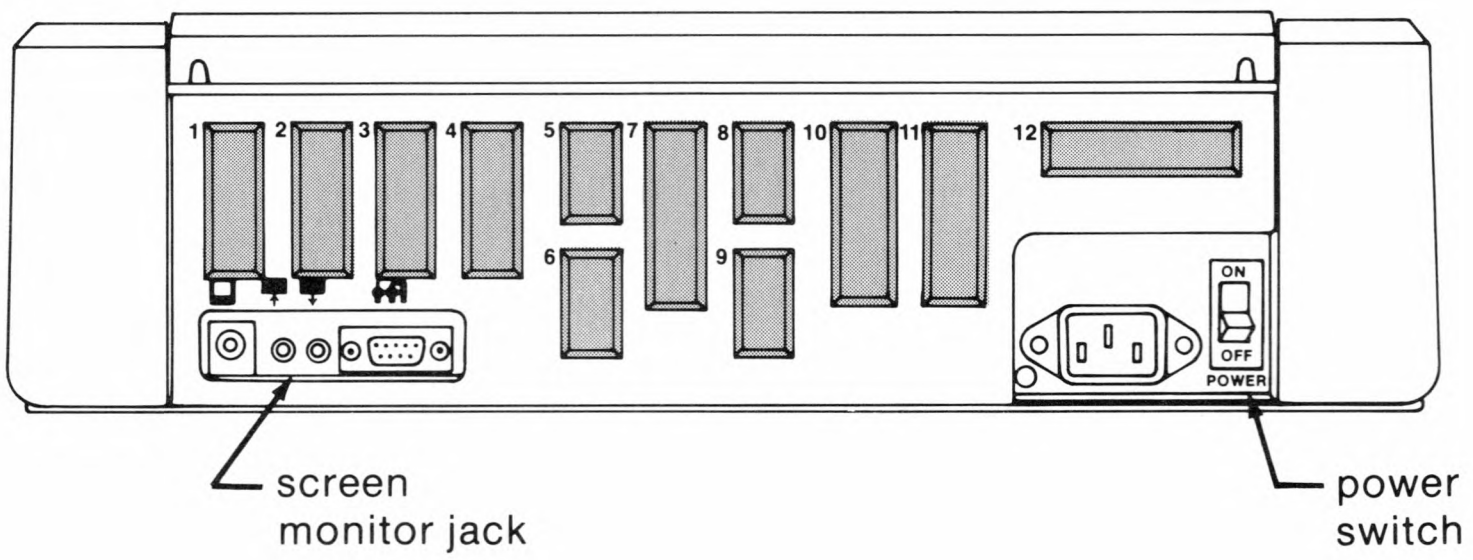


Figure 2.2. Apple Rear Panel

2. Set all connected accessory external power switches to OFF (display monitor, printer, and other external devices).

3. Ensure that there is nothing on the top cover of the computer.

4. Position your Apple computer with the keyboard directly in front of you, then remove the top cover, as shown in Figure 2.3.

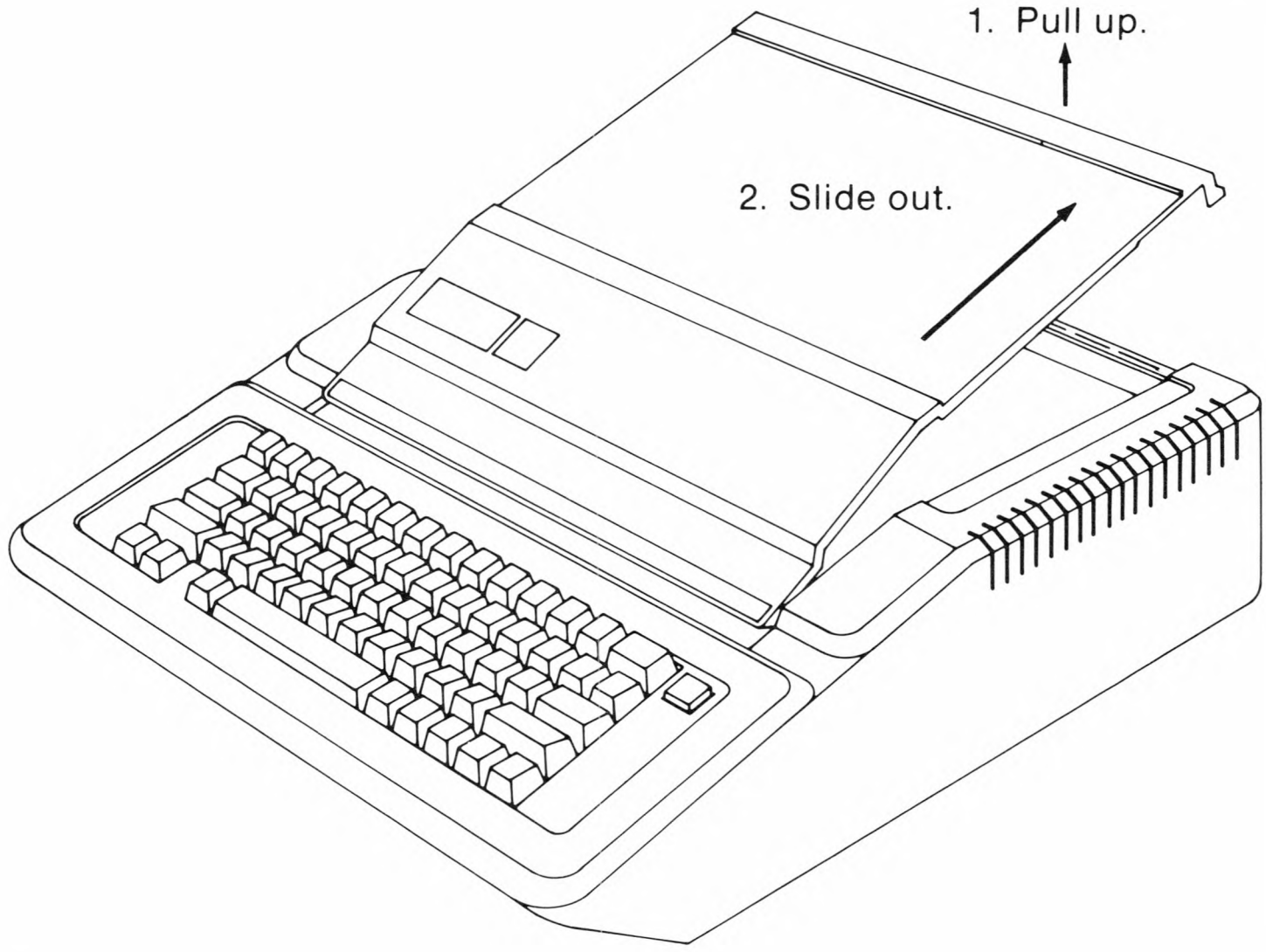


Figure 2.3. Top Cover Removal

5. Slide the cover back and remove it from the computer. Once the cover has been removed, place it somewhere out of your way.

6. Using Figure 2.4 as a guide, locate accessory slot 4.

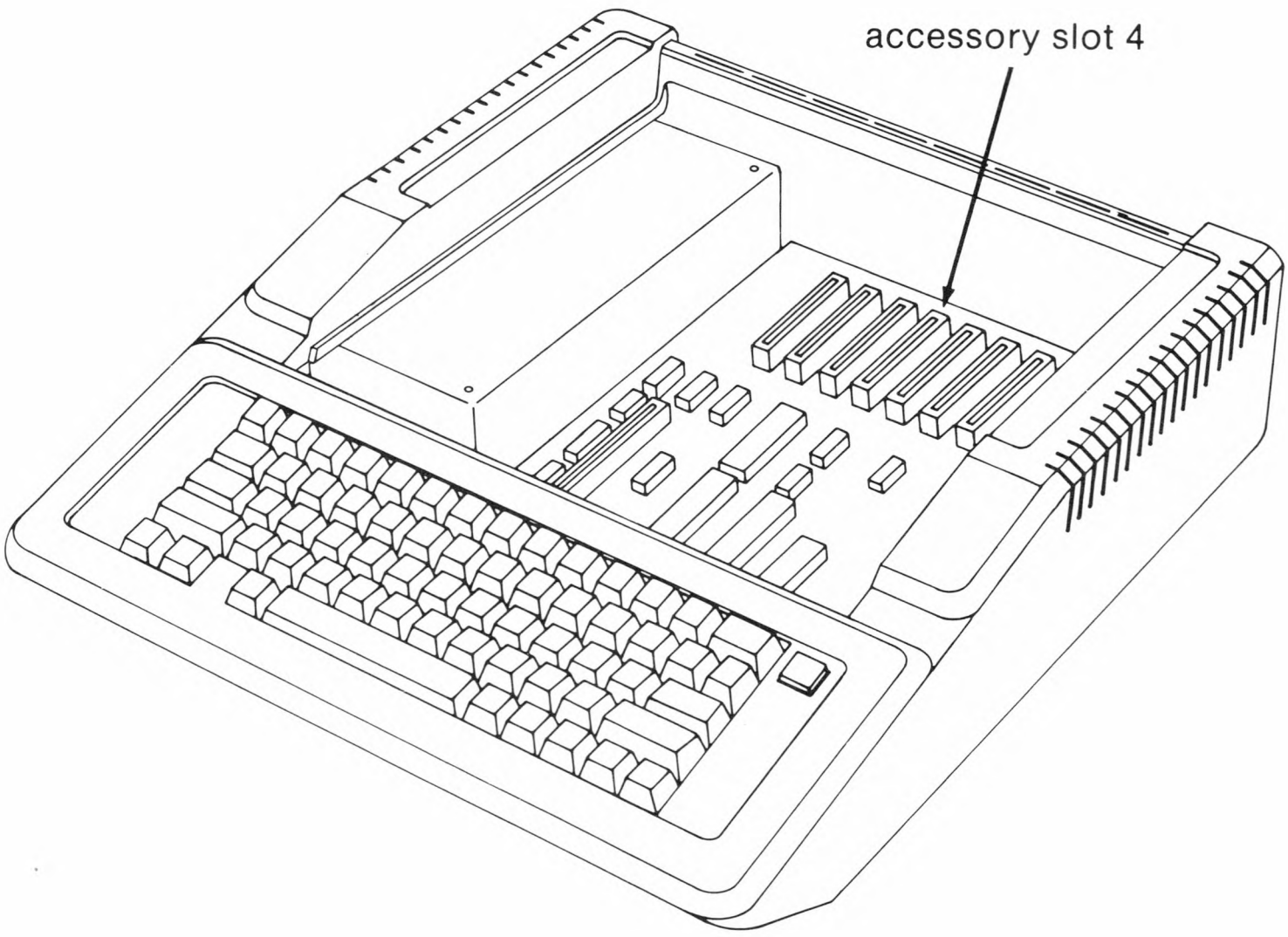


Figure 2.4. Accessory Slot 4

7. Remove the SoftCard II circuit board from its container.

8. Align the SoftCard II circuit board in the accessory slot so that the edge-connector is directly over the connector in the slot. The component side should face the right side of the chassis. Use Figure 2.5 as a guide.

9. Press the circuit board down into the connector using steady but firm pressure. Make certain that the SoftCard circuit board is level and not tilted down toward the front of the Apple.

10. Refer to Table 2.1 for the correct installation configuration of your other accessory boards. See Figure 2.1 for accessory slot locations. If you have any other accessory boards to install, do it now.

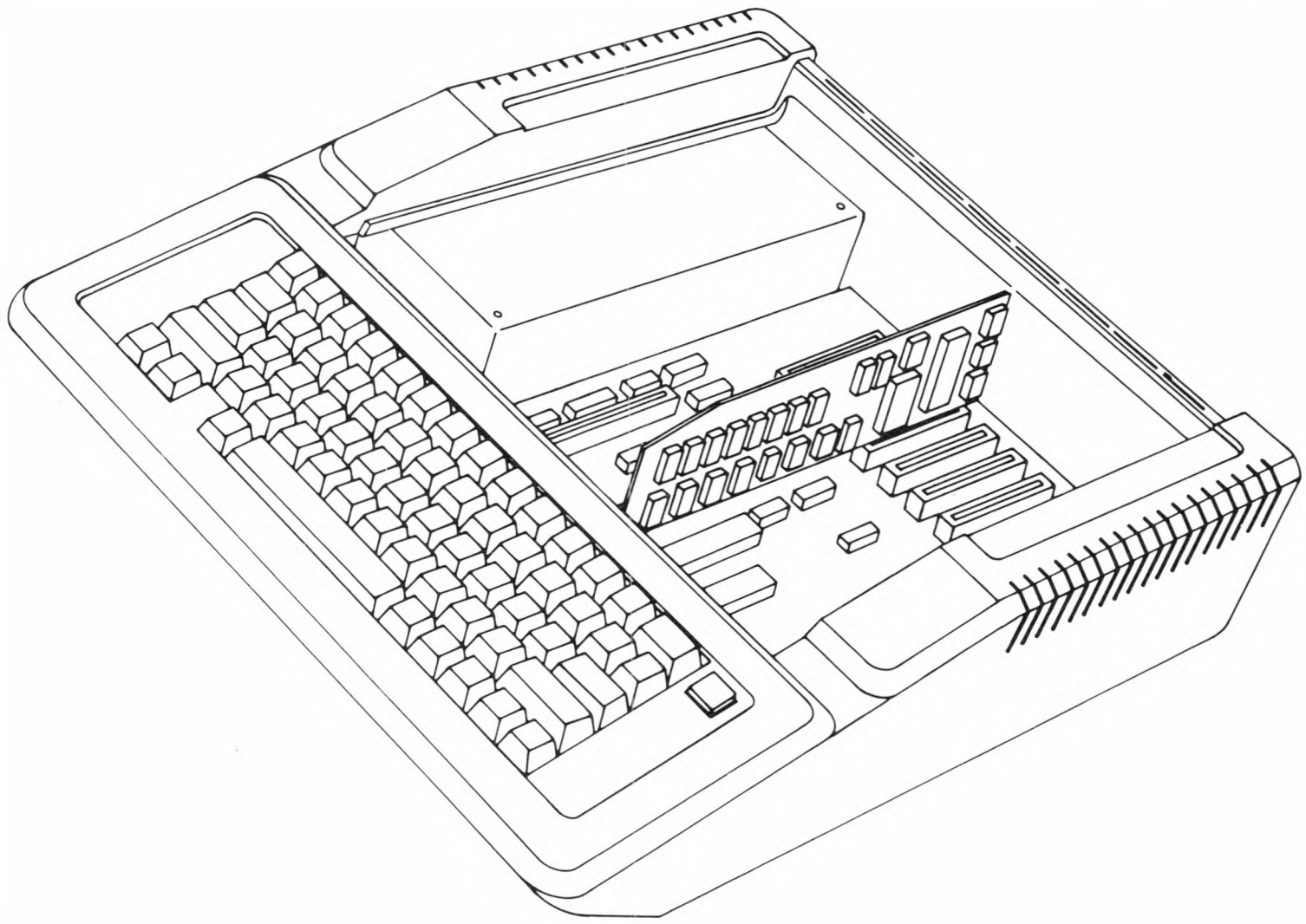


Figure 2.5. Installation of SoftCard II

11. Carefully place the top cover back on the computer.

12. Slide the forward edge of the cover under the forward lip of the chassis, as shown in Figure 2.6.

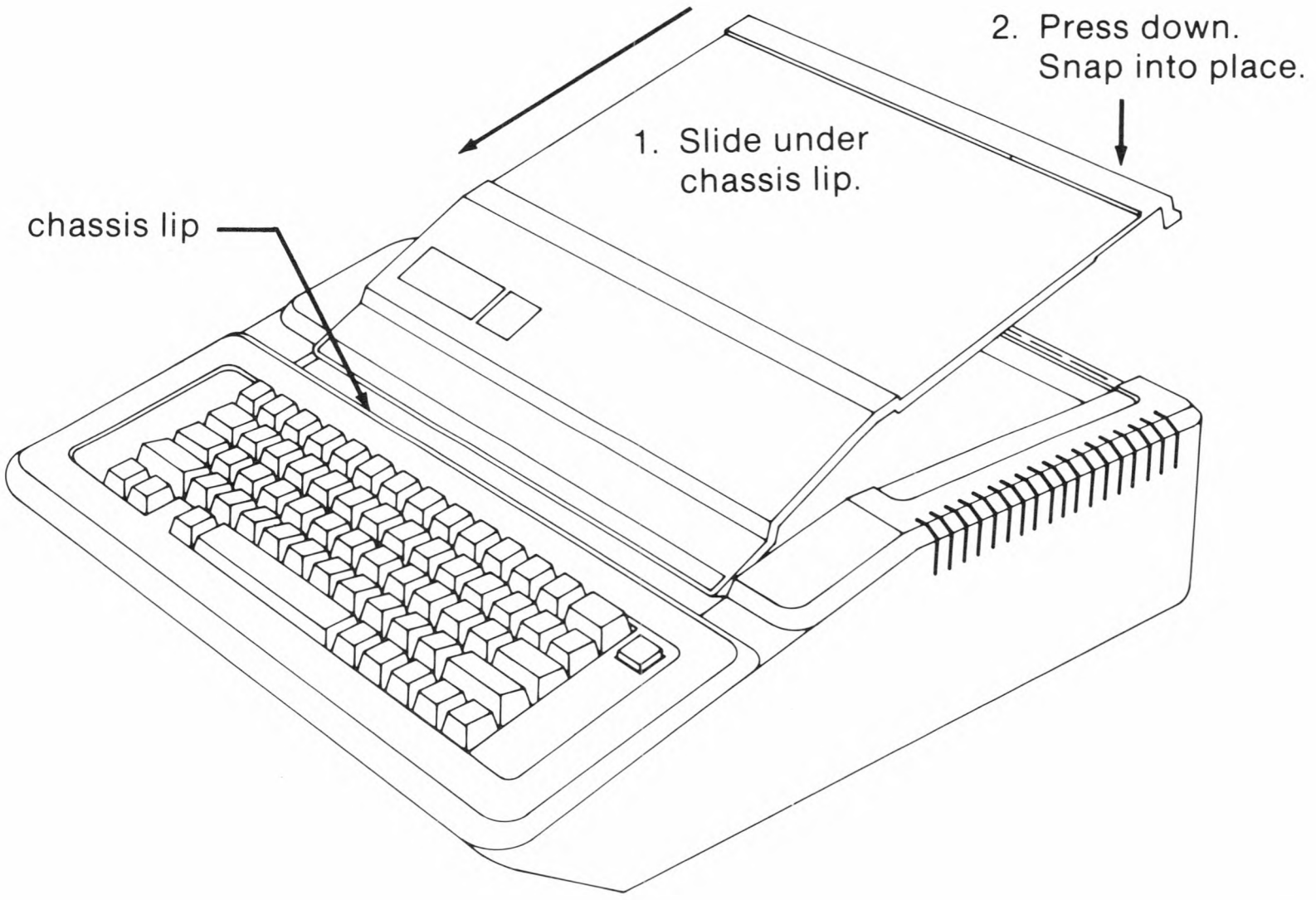


Figure 2.6. Top Cover Replacement

13. Press the rear portion of the cover down until the corners pop back into place.

14. Turn all other external device power switches (printer, monitor, etc.) to ON.

This completes the circuit board installation procedure. Since CP/M uses letters to identify disk drives, we recommend applying the labels from the SoftCard II carton to the front of each disk drive, as indicated:

Drive 1 is labeled drive A:

Drive 2 is labeled drive B:

and so on. Each successive drive is labeled with the next letter.

The hardware of your SoftCard II system is now ready to use. The next step is to "install" the software and start using your SoftCard II system. The next chapter will show you how.

Chapter 3
Getting Started

Before using a CP/M program, three steps must be taken. These are:

Loading the CP/M operating system into memory

Making backup copies of your SoftCard Master disk

Making any adjustments to CP/M required by your program or an I/O device

This chapter will tell you how to do the first two steps and refers you to the appropriate section for the last step. If you are new to CP/M, you should read this chapter completely before using CP/M.

Loading CP/M

CP/M is loaded into memory by inserting the SoftCard II Master disk into disk drive A: and setting the POWER switch to ON. (CP/M disks are inserted into the Apple disk drives the same way Apple DOS disks are inserted. That is, the disk label faces up and the write-protect notch is on the left.) When you turn on the power, CP/M is automatically loaded into memory. (This process is known as "booting.")

After a few seconds, the screen displays:

```
SoftCard II CP/M
64K Version x.xx

(c) 1984 Microsoft Corporation

SoftCard is a trademark of Microsoft Corporation.
CP/M is a registered trademark of Digital Research, Inc.

A>
```

For Apple *//e* computers, there is a second method you can use if the power is already turned on. If you have an Apple disk in disk drive A:, remove it and insert the SoftCard Master disk. Then, while holding down the OPEN-APPLE and CONTROL keys, press the RESET key. As in the first method, this will load and run CP/M. The screen display will be the same.

The A> on your display is a prompt indicating that CP/M is ready to accept a command.

As noted in the previous chapter, CP/M uses letters to identify disk drives. The A> indicates that all disk operations will be performed from drive A:, unless you command otherwise. (See the "d:" command in Chapter 6.)

To see if CP/M is operational, type

```
CAT
```

and press the RETURN key. The CAT command is similar to the Apple DOS CATALOG command, but returns slightly different results. You will see the following display:

```
APDOS    .COM  2K  | CONFIGIO .BAS  7K  | ED      .COM  7K  | PIP     .COM  8K  |
ASM      .COM  8K  | COPY     .COM  2K  | GBASIC  .COM  26K | STAT    .COM  6K  |
AUTORUN  .COM  1K  | DDT      .COM  5K  | LOAD    .COM  2K  | SUBMIT  .COM  2K  |
BOOT     .COM  1K  | DUMP     .COM  5K  | MFT     .COM  2K  | XSUB    .COM  1K  |
CAT      .COM  1K  | DUMP     .ASM  1K  | PATCH   .COM  1K  |

Total of 88K bytes in 19 files, 38K bytes available
```

When you typed *CAT* and pressed RETURN, the two actions instructed the CP/M operating system (which is now in the Apple IIe's memory) to find the program CAT on the disk drive and execute the instructions contained within. By doing this, you effectively tested the different software modules of CP/M and the transient program CAT. CAT also shows you the files you have on disk, how much memory space you have left on your SoftCard II Master disk, and the size of each of the files. Check what you see on the screen with the display above; they should match.

When you have verified that your SoftCard Master disk has the same files as shown in the display, the next step is to make backup copies of your SoftCard Master disk.

Backing Up the SoftCard Master Disk

A "backup copy" is a duplicate copy of a program or set of programs, usually on a floppy disk or magnetic tape. Making backup copies is a good programming practice that ensures that you will always have a copy of your software in case the original copy is damaged or erased, or just wears out.

Making a backup copy also permits you to modify CP/M for different software and I/O configurations. You should always work with the backup copy and never with the SoftCard Master disk.

Making backup copies of your SoftCard II Master disk is a simple, one-step process with the SoftCard COPY program. (The COPY program is fully documented in Chapter 6 of this manual.) When you have made backup copies, store the SoftCard Master disk in a safe, dry place away from magnetic interference.

Use the procedure designed for your system configuration (single-drive or multiple-drive) to make backup copies of the SoftCard Master disk. Both procedures assume that CP/M has been loaded into memory and that the A> prompt is on the screen.

Note

If you make a mistake when typing a command, use the ← (left-arrow) key to backspace and correct it. A description of CP/M line editing commands can be found in Chapter 5 of this manual.

Making Backup Copies With a Single-Drive System

For single-drive systems, use the following procedure to make a backup copy of your SoftCard II Master disk. The COPY program will format the backup disk as it copies it.

1. Insert the SoftCard Master disk in drive A:, type

   ```
   COPY A:=A:
   ```

   and press the RETURN key. After a few seconds, the screen displays:

```
SOFTCARD II CP/M
Disk Copy Program
(c) 1984 Microsoft Corporation

Insert SOURCE   disk and press RETURN
```

2. Leave the SoftCard Master disk in drive A: and press the RETURN key. The program will then copy a portion of the SoftCard Master disk. The COPY program responds by displaying:

   ```
   Insert DESTINATION disk and press RETURN
   ```

3. Remove the SoftCard II Master disk and insert a blank disk into drive A:. Then press the RETURN key.

Note

For single-drive systems, all data copied from the source disk to the destination disk must be held in memory while you change disks. Because the Apple *//e* memory is smaller than the amount of data to be copied, only part of the data can be copied at a time.

When the program has written a portion of the disk into memory, it prompts you with:

```
Insert     SOURCE     disk and press RETURN
```

Remove the destination disk (the disk you just copied data to) and insert the SoftCard Master disk back into the disk drive. Press the RETURN key. This process will be repeated several times until the entire disk has been copied. When the copy process is complete, the screen displays:

```
Operation completed

Do you wish to repeat this operation?
```

4. You now have a backup copy of your SoftCard Master disk. To make additional copies , press the *Y* key. If you do not want to make another copy, press the *N* key. The screen displays:

```
Insert CP/M system disk into drive A:

Press RETURN
```

Since you already have a copy of your SoftCard Master disk in the drive, simply press the RETURN key. This will return you to CP/M command level, and you will see the A> prompt.

Making Backup Copies With a Multiple-Drive System

For multiple-drive systems, use the following procedure to make a backup copy of your SoftCard II Master disk. The COPY program will format the backup disk as it copies.

1. Insert your SoftCard II Master disk into drive A: and type

```
COPY B:=A:
```

and then press the RETURN key. After a few seconds, the screen displays:

```
        SOFTCARD II CP/M
        Disk Copy Program
(c) 1984 Microsoft Corporation

Insert SOURCE      disk into drive A:
Insert DESTINATION disk into drive B:

Press RETURN to begin
```

2. Insert a blank disk into drive B:. Press the RETURN key to start the copy process.

   When the copy process is complete, the screen will display the message:

   ```
   Operation completed

   Do you wish to repeat this operation?
   ```

3. You now have a backup copy of your SoftCard Master disk. If you want to make another copy, press the *Y* key and follow the instructions given on the screen.

   When you have finished copying disks, press the *N* key. The program responds with:

   ```
   Insert CP/M system disk into drive A:

   Press RETURN
   ```

   Remove the SoftCard Master disk and store it in a safe, dry place away from magnetic interference.

4. Remove the backup copy you have just made from drive B: and insert it into drive A:. Press the RETURN key to exit to CP/M command level.

I/O Configuration

If you are using nonstandard I/O devices, the final step in getting started with CP/M is modifying the I/O portion of CP/M. (See "Compatible Accessory Boards" in Chapter 2 for information on nonstandard devices.) If you are using standard devices, you can start using your SoftCard II system now.

The CP/M operating system in your SoftCard II package is configured internally to work with most standard Apple][and *//e* accessories. However, you may have to modify the I/O portion of CP/M to accommodate some of the accessories that

are not directly compatible with CP/M. This is particularly true if you are using:

An external terminal

An 80-column video display board for different character fonts on the screen

Nonstandard I/O "driver" software

A modem for telecommunications

A different disk drive system, such as a "hard disk"

Read the manufacturer's manual for instructions on how to use I/O device software with your CP/M system. If there are no instructions, contact your dealer.

Most of the modifications to CP/M can be made with the CONFIGIO program. For more information, read the following sections:

To use an external terminal or an 80-column video display board with your system, read the "Screen Function Interface" section in Chapter 6 of the *SoftCard II Programmer's Manual*.

If your terminal requires additional software to run with CP/M, read "Nonstandard I/O Devices and User Software" in Chapter 6 of the *SoftCard II Programmer's Manual*.

To add additional I/O software, read "Nonstandard I/O Devices and User Software" in Chapter 6 of the *SoftCard II Programmer's Manual*.

To use a different disk drive system with your Apple (other than the Disk][drives), you will probably need additional software. The manufacturer of the disk drive system will usually provide explicit instructions for modifying CP/M for the disk drive system. Check with your computer dealer before installing non-Apple disk drives into your system.

Chapter 4
An Introduction to CP/M

This chapter explains in general terms how the CP/M operating system works with your Apple][,][Plus, or *//e* computer. Reading this chapter is not necessary for running application programs, but it is recommended for a better understanding of how the components of a computer system work together. If you are new to CP/M, we recommend you read the *Osborne CP/M User Guide* after completing this chapter.

Components of a Computer System

In order to understand how CP/M works, it is necessary to understand how computers work in general. This section describes the different components of a computer system and explains how they work together.

Computers are used for a wide variety of purposes, from scientific and business applications to home entertainment. All applications, regardless of their purpose, perform one common function: processing data (information) for a desired end result. For example, a business program may process the figures that are part of a debit account and return those figures in an accounts receivable format. A game, on the other hand, may take the data supplied by your hand movements (through a game-controller) and process that data to move a figure on the screen. To accomplish either goal, the computer processes the data through the various components that comprise the computer system.

A computer system has many components that are divided into two general categories: hardware and software. Both are necessary to process data.

Hardware

Hardware is the term that is applied to the physical components of the computer system—the keyboard, screen monitor, accessory boards, and I/O devices (printers, disk drives, and so on). The most important hardware components are the CPU, memory, and the input/output interface.

The CPU (central processing unit) is a device called a *microprocessor*. A microprocessor is an integrated circuit (also called an IC or "chip") that performs the actual processing of data ("computing") by executing instructions stored in the computer's memory.

There are many types of microprocessors, and they vary in how much data they can process and how fast they can process it. The Apple computer uses the 6502 microprocessor as its CPU. Other microprocessors include the Zilog Z80 and the Intel® 8080. Each of these microprocessors has its own instruction set, which is simply the total repertoire of commands that the CPU will recognize and execute.

The second major component of a computer is *memory*. Memory is where programs and data are stored. The Apple uses two types of memory: *internal memory* for storing programs and data for immediate execution; and *mass external storage* (usually a disk drive) for storing files and large amounts of data that are not needed by the computer for immediate execution.

Internal memory is either RAM (random access memory) or ROM (read only memory). RAM is used when data or programs must be stored and revised easily. It is usually the largest portion of the internal memory in a typical computer. ROM, on the other hand, is used for short programs which are never revised at all, such as a *bootstrap loader* (also called a *boot program*). A bootstrap loader is an initializing program which loads other programs into memory. It is executed from ROM when power to the computer is turned on.

The third major hardware component of the computer is a set of circuits collectively known as the input/output (I/O) interface. The I/O interface includes the *I/O Bus* (the circuit paths between the CPU and the accessory slots) and any interface printed circuit boards that are installed in the accessory slots. The interface boards connect to external I/O devices, such as, printers, terminals, or disk drives. Although the keyboard on the Apple appears to be an integral part of the computer, it is really an I/O device for operator input.

Software

The *software* components control the actions of the computer. The software components you will use most often are called *programs*. Programs are defined as a set of instructions that tell the computer to perform a certain task in a specified manner. Software generally means a "program" but can also include simple machine instructions. Programs are further divided into routines, subroutines, statements, and instructions.

The fundamental building block of all software components is the *machine instruction*. A machine instruction is a coded number that the CPU recognizes as a command to perform a low-level or "primitive" task, such as sending a character to memory. Because it is difficult to communicate with the CPU in machine instructions, they are organized into blocks of instructions, called assembly language instructions.

Assembly language is a low-level programming language which uses mnemonic symbols to indicate what each CPU instruction does when executed. Note that there is no one language called "assembly language" as there is a language called FORTRAN. Assembly language is a generic term for a microprocessor's instruction set that can be used for programming. Therefore, each microprocessor has its own assembly language. To make writing programs easier, assembly language instructions are organized into larger blocks for use in high-level languages.

High-level languages are programming languages that use English-like statements for instructions. For example, the statement "PRINT" is an instruction to print a character (or characters) on the screen. It corresponds to several assembly language instructions, which in turn correspond to several machine instructions. The term "high-level" refers to the degree of complexity in how the language is structured and the amount of memory required to run it. BASIC, FORTRAN, COBOL, and LISP are just some of the high-level languages available. BASIC is the most commonly used high-level language for microcomputers.

Application programs are programs that perform a certain set of functions associated with a specific task, such as word processing. Application programs can be written either in a high-level language or in assembly language. They form the final level of translation between the user and the CPU. Most application programs use normal English sentence structure and menus.

The Role of an Operating System

An operating system is a "program" that coordinates the different components of a computer system and provides you with a direct way of controlling the computer. You could control your computer system without an operating system by using assembly language instructions, but doing so would be very time-consuming and tedious. An operating system performs three functions:

1. It controls the activities of the disk subsystem (the disk drives, interface circuits, and disk software). An operating system manages the storage and retrieval of data from these files and monitors the location and the amount of memory each file occupies.

2. It provides a convenient means for loading and executing programs from storage devices, such as disk drives.

3. It controls the activities of the I/O subsystem. Through an operating system, you can control the flow of data between the CPU and I/O devices such as the terminal, printer, and disk drives.

For an operating system to perform these functions, it must be able to communicate with the CPU and the other components of the computer system. Since different microprocessors have different instruction sets for communication, an operating system must be written for a specific microprocessor.

CP/M

CP/M (Control Program/Microprocessors) is an operating system written for the 8080 and Z80 microprocessors, but not for the Apple 6502 microprocessor. The reason is the incompatibility of the 6502 instruction set. (There are two operating systems written for the 6502 microprocessor: Apple DOS and Apple][Pascal.)

CP/M was originally written for the Intel 8080 microprocessor. The Z80 microprocessor, which was introduced later, has a very similar instruction set and is compatible with CP/M. Many computers use either the 8080 or the Z80 microprocessor and thus share a common means for programming. This is the primary reason why many computers use CP/M for an operating system.

SoftCard, CP/M, and the Apple Computer

To run CP/M programs on the Apple computer, an 8080 or Z80 microprocessor is needed. The SoftCard circuit board contains a Z80 microprocessor and the interface circuitry for running CP/M programs.

The Z80 microprocessor on the SoftCard allows you to run CP/M programs whenever the CP/M operating system is loaded into memory from disk. The CP/M bootstrap loader has been modified by Microsoft to activate the Z80 microprocessor. Because all CP/M instructions are written for the Z80, all commands from the keyboard are executed by the Z80. The Z80, however, uses the 6502 microprocessor to process I/O data to and from the I/O subsystem.

Note

When you load the Apple DOS or the Apple Pascal operating systems into memory, the 6502 will execute all instructions. The 6502 does not use the Z80 for any purpose. The Z80 therefore remains in a "wait" state until it receives a command that is part of its instruction set.

CP/M Bootstrap Loader

As explained in the previous section, an operating system is the "program" that, among other functions, permits the computer to gain access to data and other programs from a mass storage device, such as a disk drive. Because an operating system cannot be in memory to load itself, however, another program must perform this function. This is done with a program called a bootstrap loader.

The bootstrap loader in the Apple is an assembly language routine in ROM that reads the first sector of the first track of a disk drive into the lowest memory locations every time power is turned off and then on, or when CONTROL-RESET is pressed. When a CP/M disk is in drive A:, the Apple bootstrap loader loads into memory the data that is stored on Track 0, Sector 0 of the disk. ("Tracks" and "sectors" are explained in the "Disk Organization" section of this chapter.) This data contains a second bootstrap loader. When loaded into memory, the second bootstrap loader in turn loads the rest of CP/M. This process is shown in Figures 4.1 and 4.2. Once CP/M has been loaded into memory, it uses other loader routines to load CP/M programs.

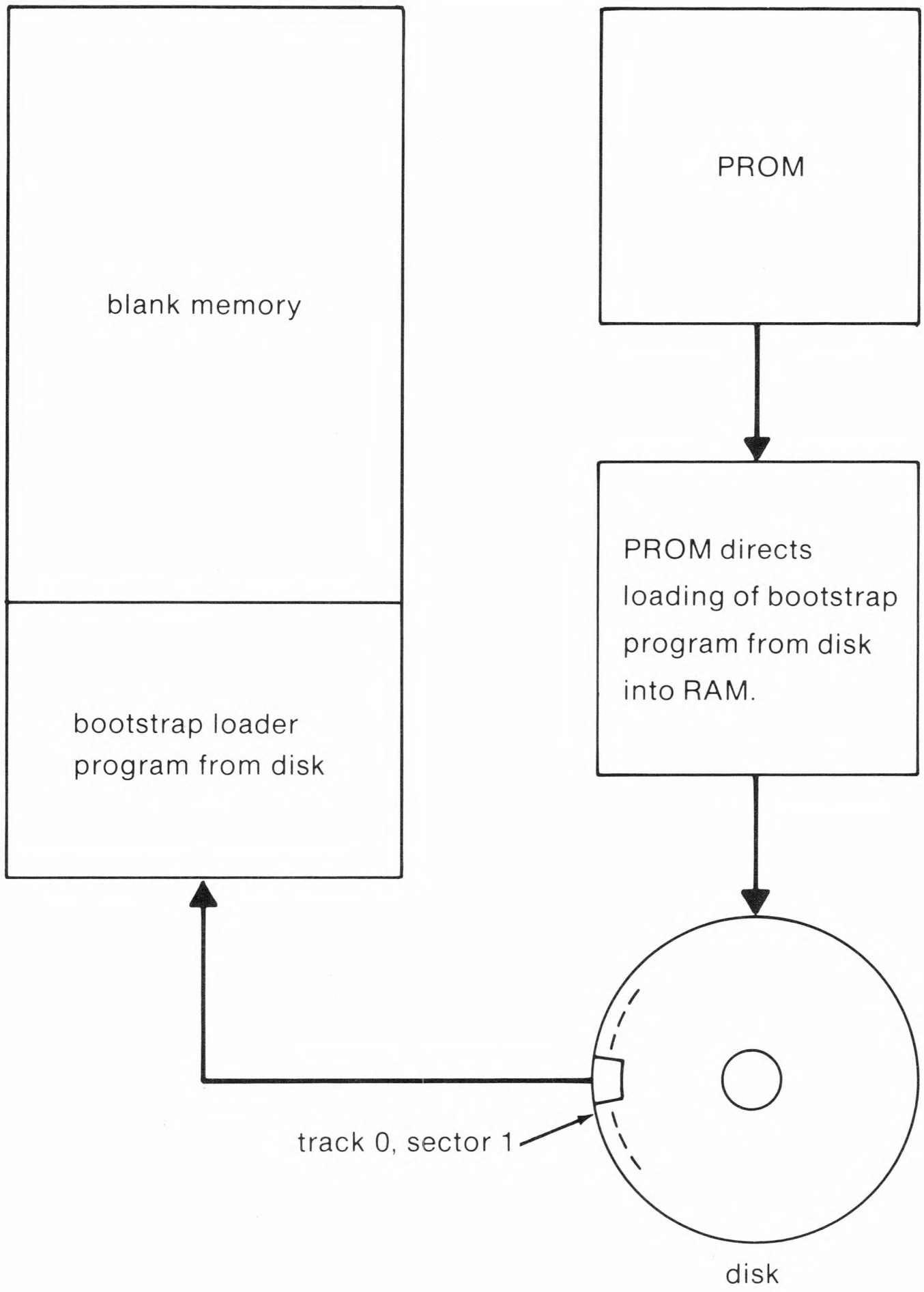


Figure 4.1. Memory Locations

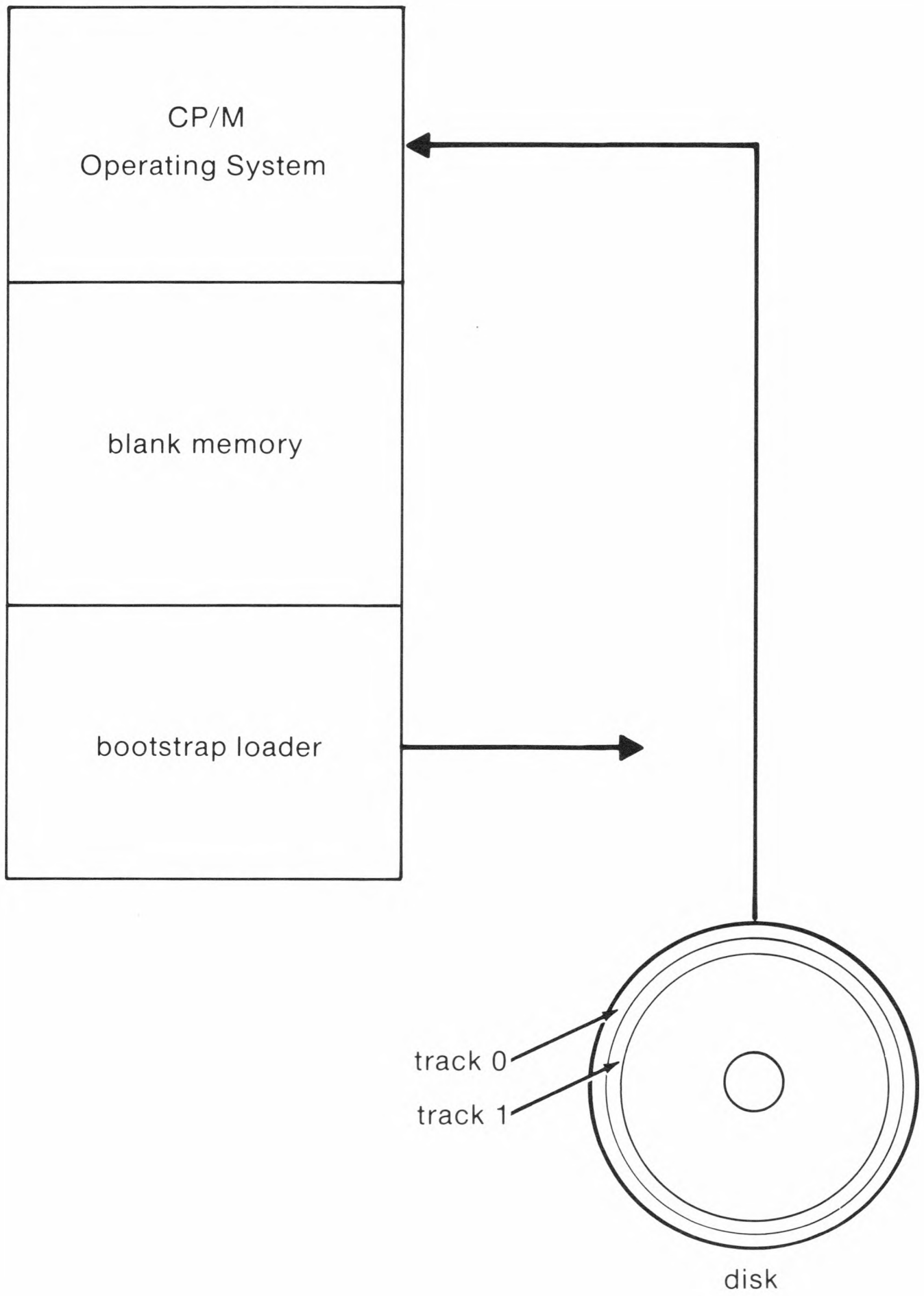


Figure 4.2. The Bootstrap Loader

The process of loading and starting the operating system is called "booting." CP/M can be booted by one of two methods. The first is a *cold start*, which loads the entire operating system into memory after the power has been turned off and then back on. Cold starts are performed whenever you want to reload the entire operating system. For example, if you were using Apple DOS and wanted to use a CP/M program, you would first remove the Apple DOS disk from drive A: and replace it with a CP/M system disk. Then, you would perform a cold start by pressing the RESET key while holding down the CONTROL and OPEN-APPLE keys (Apple *//e* computers only).

Note

On Apple][and][Plus computers, a cold start is performed by turning the power off and then back on.

The other booting method is a *warm start*. A warm start is performed by pressing CONTROL-C. The difference between the two is that a cold start reads the whole CP/M operating system into memory while a warm start reads in only a portion of CP/M. The rest of CP/M is assumed to be intact since there has been no loss of power. The programs, if operating properly, will not alter the memory containing the other part of the CP/M operating system. Warm starts are used whenever you change disks in the active drive; or when you need to clear an error condition.

How CP/M Uses Memory

The CP/M operating system that you receive with the Soft-Card II system consists of two types of software: sets of assembly language routines organized into modules, and executable programs called transient programs. (Transient programs are explained in the "CP/M Transient Programs" section in this chapter.) The software modules that are loaded into memory form the nucleus of CP/M. These modules perform the functions of an operating system. (See "The Role of an Operating System" earlier in this chapter.)

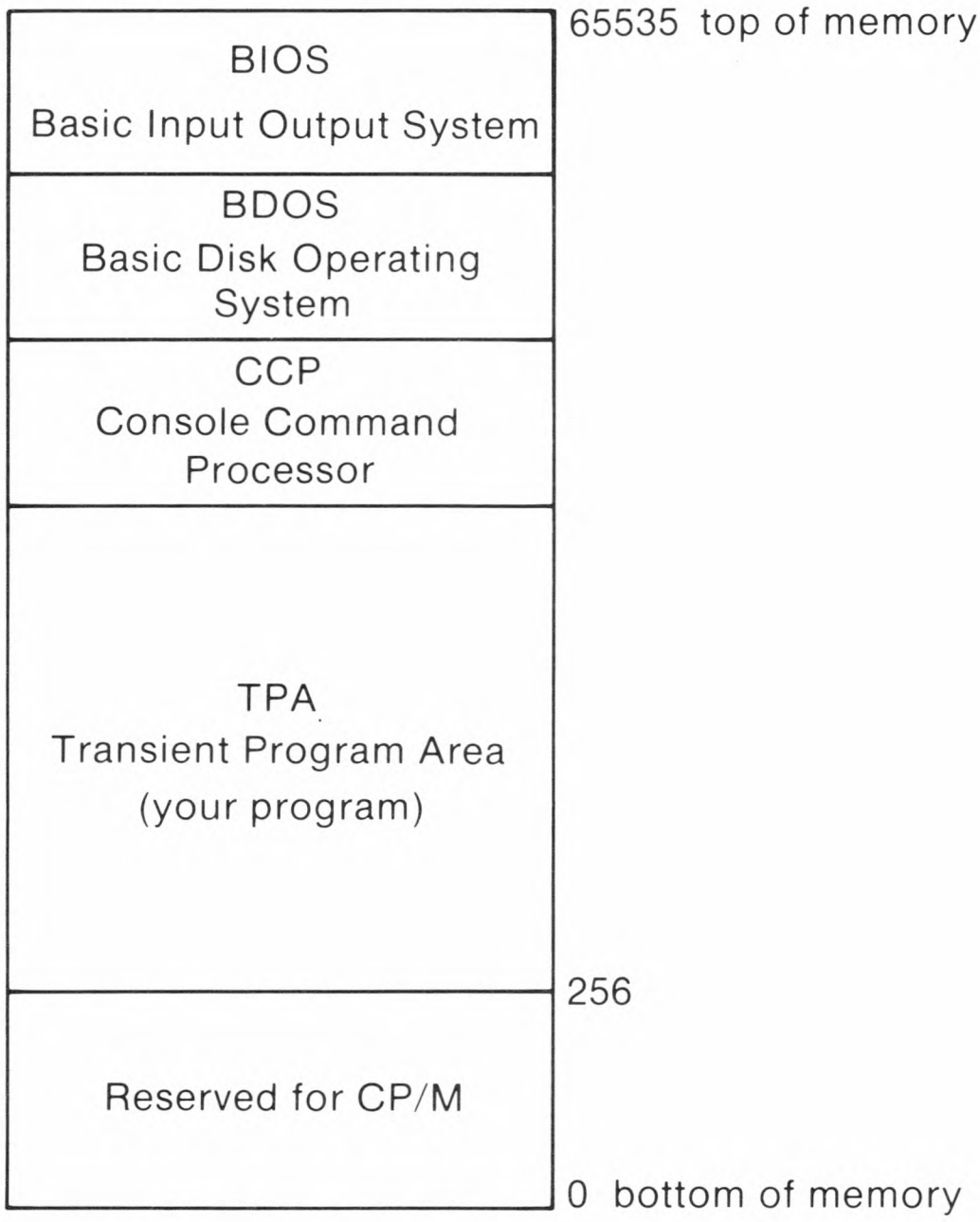


Figure 4.3. Software Module

When you load CP/M with a cold start, you are loading only the software modules into memory. The modules are loaded into specific areas of memory, as shown in Figure 4.3. The lowest section of memory is reserved for the bootstrap loader routine first, and then any other software necessary for performing warm and cold starts. (Note that the TPA is not a software module, but a dedicated area of memory for programs.) The three CP/M software modules are the CCP, BIOS, and BDOS.

The *TPA* (Transient Program Area) is where programs or languages are stored and executed under CP/M. For the SoftCard version of CP/M, programs up to 59K bytes in size can run in the TPA.

The *CCP* (Console Command Processor) is the software module which controls the interaction between you and the computer at CP/M command level. The CCP is the part of CP/M that allows programs to be loaded into the TPA and run. It also permits files to be created and deleted, and performs other "housekeeping" functions. The CCP is discussed in more detail in the section on built-in commands in this chapter.

Note

The "command level" mode of operation is when CP/M controls the computer and all commands are executed through the CCP. The inverse is the "program level" mode of operation, when a program controls the computer and permits only certain commands to be typed.

The *BIOS* (Basic Input and Output System) is the software module which contains the assembly language routines in CP/M that are *machine-dependent*. These are the routines that are written for a specific implementation of CP/M (in this case, the SoftCard II system in the Apple computer). The BIOS module contains all the I/O programs for communicating with the terminal, the disk controller interface, and other I/O devices.

The *BDOS* (Basic Disk Operating System) is the software module which manages the disk subsystem. The BDOS, unlike the BIOS, is *machine-independent*. The assembly language routines contained in the BDOS module are the same for all computers, regardless of the disk drive interface circuitry or the particular combination of I/O devices connected to the computer (the system configuration). The BDOS can be considered the core, or the heart, of CP/M.

Because the BDOS and the CCP modules are generally the same for all computers and the BIOS can be modified for each type of computer, CP/M can run on a variety of computers.

I/O Communication

CP/M communicates with I/O devices through the BIOS module. The BIOS module contains four 2-part interface routines that can be modified to accommodate a wide variety of I/O devices. The four I/O interface routines are each divided into two categories: *logical devices* and *physical devices*.

The logical device is an assembly language subroutine in the software that is a logical representation of the I/O function (as opposed to an actual device). In CP/M there are four logical devices, each corresponding to a general I/O function. They are: CONSOLE (CON:), for input and output to and from a console or terminal; READER (RDR:) for input; PUNCH (PUN:) for output only; and LIST (LST:) for output to a listing device, such as a printer.

The other part of the I/O interface is the physical device. The physical device is a vector that points to an assembly language routine called a *driver*. (A vector is an address containing an instruction that causes the CPU to "jump" to another address that is usually the start of another routine.) A driver routine is the software that is written to communicate with a specific type of physical device.

CP/M Physical Devices

In the SoftCard implementation of CP/M there are 11 physical devices; each corresponds to a specific type of I/O device. The following list shows the possible physical devices for the SoftCard version of CP/M.

Device	Description
TTY:	(Teletype) Normally points to the Apple keyboard and monitor.
CRT:	(Cathode ray tube) Same as TTY:, but used for an external terminal.
UC1:	(User-defined console device) An I/O device that can be used for input or output.

Device	Description
PTR:	(Paper-tape reader) Used for an input-only device from slot 2.
UR1:	(User-defined reader #1) Same as PTR:, but can be modified by the user.
UR2:	(User-defined reader #2) This device is the same as UR1:.
PTP:	(Paper-tape punch) Any standard Apple interface board capable of processing output from accessory slot 2.
UP1:	(User-defined punch #1) Same as PTP:, but can be modified by the user.
UP2:	(User-defined punch #2) This device is the same as UP1:.
LPT:	(Line printer) The LPT: device is any standard Apple interface board installed into slot 1 capable of receiving output.
UL1:	(User-defined list device) Same as LPT:, but can be modified by the user.

The Apple computer communicates with I/O devices using a method called *memory-mapped I/O*, so that each of the physical device routines communicates with a specific accessory slot. Most driver subroutines are located in the ROM of the installed interface board in the accessory slot.

An example of how memory-mapped devices work would be a line printer. The interface board for the line printer must be installed in slot 1. For CP/M to communicate with the line printer, the LPT: physical device points to the slot address of accessory slot 1. The actual communication subroutine for communication between the LST: logical device and the printer is contained in ROM on the interface board.

One advantage of using the logical device/physical device interface is that it permits you to select the I/O device you wish to communicate with, by using an operating system command or a statement in a program. For example, if you are using two printers, printer output can be directed to the printer of your choice by using the STAT program. (STAT changes logical to physical device assignments.)

Logical to Physical Device Assignments

The possible logical to physical device assignments are noted below. (The first physical device for each logical device is the normal assignment.)

Logical Device	Valid Physical Devices
CON:	TTY:, CRT:, UC1:, BAT:
RDR:	TTY:, PTR:, UR1:, UR2:
PUN:	TTY:, PTP:, UP1:, UP2:
LST:	LPT:, TTY:, CRT:, UL1:

Note

BAT: (Batch processing mode) directs input from the currently assigned RDR: device to the CON: device. Output is directed to the currently assigned LST: device.

The other advantage of the logical/physical device system is that it can be used in many different implementations, thus freeing the programmer from adding any additional code for each system's particular I/O configuration. The user doesn't need to monitor which I/O devices are connected to the computer and in what particular configuration.

Communication with the disk drive system is similar to I/O communication, but is confined to the logical disk device (DSK:) being assigned to the available disk drives. Only one disk drive (of the four possible disk drives) can be accessed at a time.

CP/M Disk Drive System

The disk drive system is the permanent storage media for CP/M. To use the disk drive system effectively, you should be familiar with how CP/M stores data on the disk drive system.

Disk Organization

Information is stored on a disk in blocks of 128 bytes. Each of these blocks is referred to as a "sector." Each sector has a unique address or location on the disk and information is stored and retrieved by telling the disk drive to read or write information to a specific sector. The sectors are laid out on the disk in concentric, circular tracks, as shown in Figure 4.4.

In the SoftCard version of CP/M, there are 16 sectors per track and 35 tracks per disk. Fortunately, when you type a command to access information on a disk, you don't have to know the track and the sector. You need to know only the name of the file and the disk that it is located on. CP/M will find the sector and track automatically.

Note

See the section "CP/M Disk Files" in this chapter for an explanation of the CP/M disk file system and instructions on how to access information on the disk.

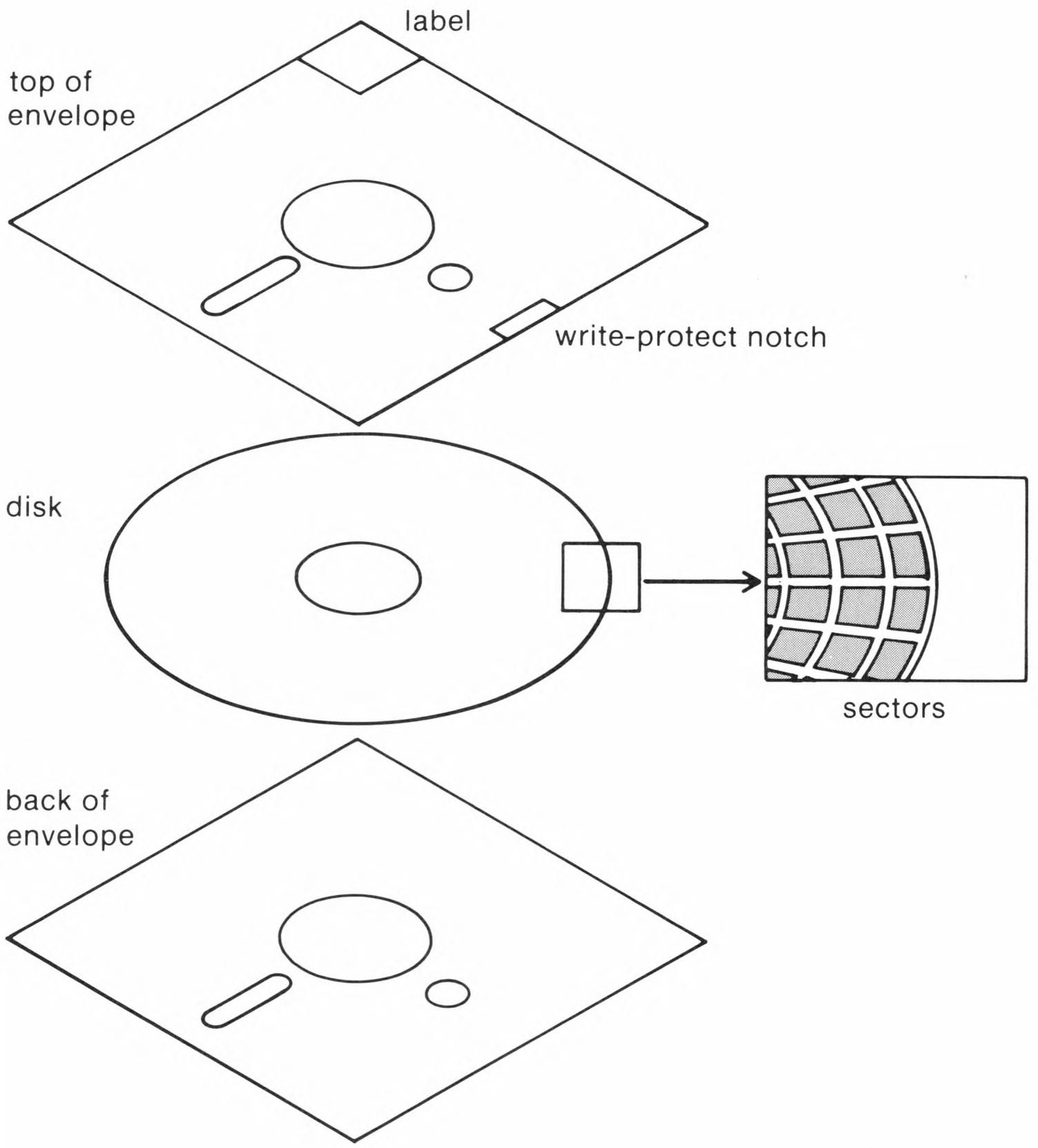


Figure 4.4. Disk Organization

CP/M Disk Types

CP/M recognizes two types of disks: system disks and data disks.

A *CP/M system disk* contains the CP/M operating system and can be loaded into memory with a warm start or a cold start from drive A:. A system disk must contain the CCP, BDOS, and BIOS modules. Any CP/M transient programs contained on the disk are optional. CP/M system disks can be created with the /S switch of the COPY utility program.

Another type of system disk is the startup disk (also called a boot disk). Startup disks are used by application programs; they have either an operating system on them or the parts of an operating system necessary to handle the program's needs. You can create your own CP/M startup disks with the AUTO-RUN program described in Chapter 6.

Data disks have no operating system data on them. They are used for the storage of programs and data files only. Since operating system information is not included, data disks have an additional 12K bytes of disk space available. Data disks are created with the /D switch of the COPY utility program.

Important

You should avoid using data disks in drive A: and in single-drive systems. The lack of an operating system on a data disk makes CP/M unable to perform a warm start and recover from errors.

Changing Disks

Whenever you change CP/M disks, you must perform a warm start because specific disk directory information is stored in memory at all times. This information is used to allocate space on the disk. When you change disks, this information must be replaced with the directory information of the newly inserted disk.

CP/M Disk Files

All data stored on disks is organized into files. A file is any collection of data, text, or program instructions. All files are referenced by file specifications, which keep track of where information is stored on the disk and what type of information it is. A CP/M file specification, or *filespec*, consists of three parts and is shown in the following notation:

[*d*:]*filename*[.*ext*]

The *d*: is the disk drive identifier, the *filename* is the filename, and .*ext* is the filename extension. All three parts are explained in the following paragraphs.

Disk Drive Identifier

The disk drive identifier is a one-letter code (A-D) followed by a colon (:). It tells CP/M which drive the file is located in. Note that the disk drive identifier is optional. If it is not included, CP/M looks for the specified file on the default or *active drive*. The active drive is the disk drive that you are currently working from. For example, when you see the A> prompt, drive A: is the disk drive that will be accessed when you give a command without a disk drive identifier. It also means that you are at the CP/M command level of operation.

Note

In other documentation, the active drive is also referred to as the *currently logged drive*.

Filename

The filename identifies the disk file and is the only required part of a filespec. CP/M filenames must start with a letter. (In some operating systems, filenames can begin with numbers or special characters.) CP/M filenames can be from one to eight characters in length and can consist of uppercase and lowercase characters. Filenames that contain both uppercase and lowercase characters will have all lowercase characters transposed into uppercase characters. For example, the filename "Program" would be transposed by CP/M into "PROGRAM."

If you include a filename extension (a three-character code) with the filename, it must be separated by a period (called a *delimiter*).

The following are examples of valid CP/M filenames:

A:MAILLIST

Refers to the file MAILLIST on drive A:.

R

Refers to the file R on the currently logged drive. Notice that this filename has only one letter.

B:BARBARA

Refers to the file BARBARA on drive B:.

Filename Extension

The filename extension denotes either the internal format of a file (the type of information in the file) or the different versions of a file. The filename extension can be from 1 to 3 characters long. For example, FNAME.1 could be the first version of the program FNAME. If you create a second version (or revise the first), you can save both versions by giving them different filename extensions (FNAME.1 and FNAME.2).

Several file types have meanings that are unique—to the CP/M operating system, to the standard CP/M transient programs, and to the high-level languages. For example, a .COM file is a "Command" file; that is, a directly executable transient program. Since certain program file operations could destroy the contents of a data file, it is a good idea to use the file type as the filename extension when you create the file. This avoids confusion when you want to use the file at a later date. Table 4.1 lists the file types commonly used for CP/M.

Table 4.1

CP/M File Types

File Extension	Type of File
.ASM	Assembly language source code
.BAK	Backup file
.BAS	BASIC source code
.COB	COBOL source code
.COM	Command file
.CRF	Relocatable assembler cross-reference
.DAT	ASCII data (FORTRAN default)
.DOC	Text document file
.FOR	FORTRAN source code
.HEX	Intel HEX format object code file
.LIB	Library file
.MAC	Macro assembler source file
.OBJ	Machine code (object file)
.OVR	COBOL compiler overlay
.PRN	Assembly language list file (PRINT file)
.REL	Relocatable object file
.SUB	SUBMIT command file
.SYM	Assembler symbol table
.TXT	Text file
.XRF	Assembler cross-reference table
.$$$	ED or PIP temporary file

Wild Card Filename Specifications

File specifications can also refer to more than one file at a time. This is done with "wild card" characters. CP/M has two wild card characters for use with filespecs: the asterisk (*) and the question mark (?). The asterisk character will match any string of characters in the filespec. The question mark character will match any character in the position occupied by the question mark during a directory search for the filename match. The following examples show some of the ways you can use wild card characters.

A:*.COM

Refers to all files on drive A: with an extension of .COM.

B:*.*

Refers to all files on drive B:.

B:????????.???

Exactly the same as B:*.* above.

DUMP.*

Refers to all files on the currently logged drive beginning with the filename "DUMP."

C*.*

Refers to any file on the currently logged drive beginning with the letter C and containing any extension.

O.

This is the same as *.*. The asterisk (*) is an abbreviation for a string of question marks (?). If an asterisk is included as part of the string, CP/M ignores all characters to the right of the asterisk and treats the whole string as a wild card character. Note the difference between this example and the next example.

?O??????.*

Refers to all files with O as the second letter of the filename on the active drive with any filename extension.

Built-in Commands

CP/M executes two types of commands, built-in and transient. Built-in commands are programs that reside permanently in the CCP module and can be used at any time. Transient commands are programs stored on a disk. Transient commands are also called transient programs.

Built-in commands are direct commands to the CPU given at the CP/M command level. They are always present whenever CP/M is active and no other programs are running. Built-in commands perform tasks such as displaying the contents of a file or a directory of disk files, renaming and erasing files, and saving the contents of memory on disk.

CP/M Built-in Commands

The SoftCard version of CP/M has seven built-in commands. Each command and its purpose is listed below. For instructions on using each of the commands, see the appropriate section in Chapter 6.

Command	Purpose
d:	Logs onto another disk drive
DIR	Displays a directory of files on disk
ERA	Erases a file or files
REN	Renames a file

Command	Purpose
SAVE	Saves the contents of memory in a file on disk
TYPE	Displays the contents of a file on the monitor screen
USER	Creates another area within the same directory

CP/M Transient Programs

A transient program is a program that can be executed as a command. The main difference between built-in commands and transient programs is that transient programs are stored on disk until they are executed. Built-in commands are stored in the CCP module in memory. Transient programs perform operations associated with programming and utility tasks such as copying files and transferring data between devices.

When not in use, transient programs are stored on disk in .COM (command) files. When you type the name of a .COM file, CP/M will load the contents of the file into memory and execute the instructions it finds in the file.

Most CP/M commands and transient programs (with a few exceptions, such as REN) are *extensible*. That is, they may be extended semantically to include additional operations. For example, the DIR command could include an argument (an entry you type in the command line) for a list of specific file types (such as BASIC files). In this case, you could type **.BAS* in the DIR command line. This instructs CP/M to display only those files with the extension of .BAS (BASIC files).

SoftCard CP/M includes 16 transient programs. Table 4.2 lists the names of the programs, their purpose, and the section of the manual that gives instructions on their use.

Table 4.2
CP/M Transient Programs

Program	Purpose	Refer to:
APDOS	Transfers text files and binary files from Apple DOS to CP/M.	Chapter 6
ASM	* Assembles 8080 assembly language programs.	*Osborne CP/M User Guide*
AUTORUN	Automatically executes a previously specified CP/M command line when the system is booted.	Chapter 6
BOOT	Exits CP/M and reboots your Apple *//e* system.	Chapter 6
CAT	Displays an alphabetical listing of the directory on the specified drive.	Chapter 6
COPY	Makes duplicate copies of disks. Options to COPY let you format disks and create CP/M system disks.	Chapter 6
DDT	* Debugs 8080 assembly language programs.	*Osborne CP/M User Guide*
DUMP	* Displays a file in hexa-decimal form.	*Osborne CP/M User Guide*
ED	* Creates and edits CP/M text files.	*Osborne CP/M User Guide*
LOAD	* Converts a .HEX file into a .COM file.	*Osborne CP/M User Guide*
MFT	Copies files from one disk to another on a single-drive system.	Chapter 6

Program	Purpose	Refer to:
PATCH	Makes program updates and modifications.	Chapter 6
PIP	* Copies and/or appends disk files and devices.	Chapter 6
STAT	* Displays status and makes device assignments.	Chapter 6
SUBMIT	* Batch processes commands from a disk file.	*Osborne CP/M User Guide*
XSUB	* Allows character input to a program from a Submit input file.	*Osborne CP/M User Guide*

* These programs are part of CP/M 2.2 and were written by Digital Research, Inc. All other transient programs were written by Microsoft.

Chapter 5
Using CP/M With the Apple Computer

This chapter explains how CP/M works with the SoftCard II system and the Apple][or *//e* computer. Although most of the material in this chapter applies to both the Apple][and the *//e* computers, the section called "CP/M and the Apple][" is for Apple][users only.

CP/M and the Apple][

CP/M operation differs slightly from the Apple][(and][Plus) computer and the Apple *//e* computer for two reasons. First, the Apple][has a different keyboard, and second, the Apple][wasn't designed for 80-column video display. Otherwise, the SoftCard II system operates the same in both models.

The Apple][Keyboard

Because the Apple][does not have an ASCII standard (typewriter configuration) keyboard, certain characters, such as right and left brackets ([ ]) and the backslash (\) are not included on the Apple][keyboard. For a program that requires these characters, you can, with the CONFIGIO program, redefine other keys to type these characters. See "Keyboard Character Definition," Chapter 6 in the *SoftCard II Programmer's Manual*, to learn how to change key assignments.

Since the Apple][and][Plus computers don't have the special Apple keys, cold starts are performed by turning the power off and then back on.

80-column Displays

Because the Apple][was originally designed for a 40-column display, you must install an 80-column video display board in accessory slot 3. Otherwise, you'll get only half the display. (See Table 2.1 in Chapter 2.) However, before using it, please read Appendix C in the *SoftCard II Programmer's Manual* for notes on 80-column operation.

Using the Apple *//e* Keyboard With CP/M

With CP/M, the typewriter portion of the keyboard (see Figure 5.1) works as it does with the Apple DOS or Apple Pascal operating systems. Several keys, however, work differently and there are certain CONTROL key sequences that are unique to CP/M.

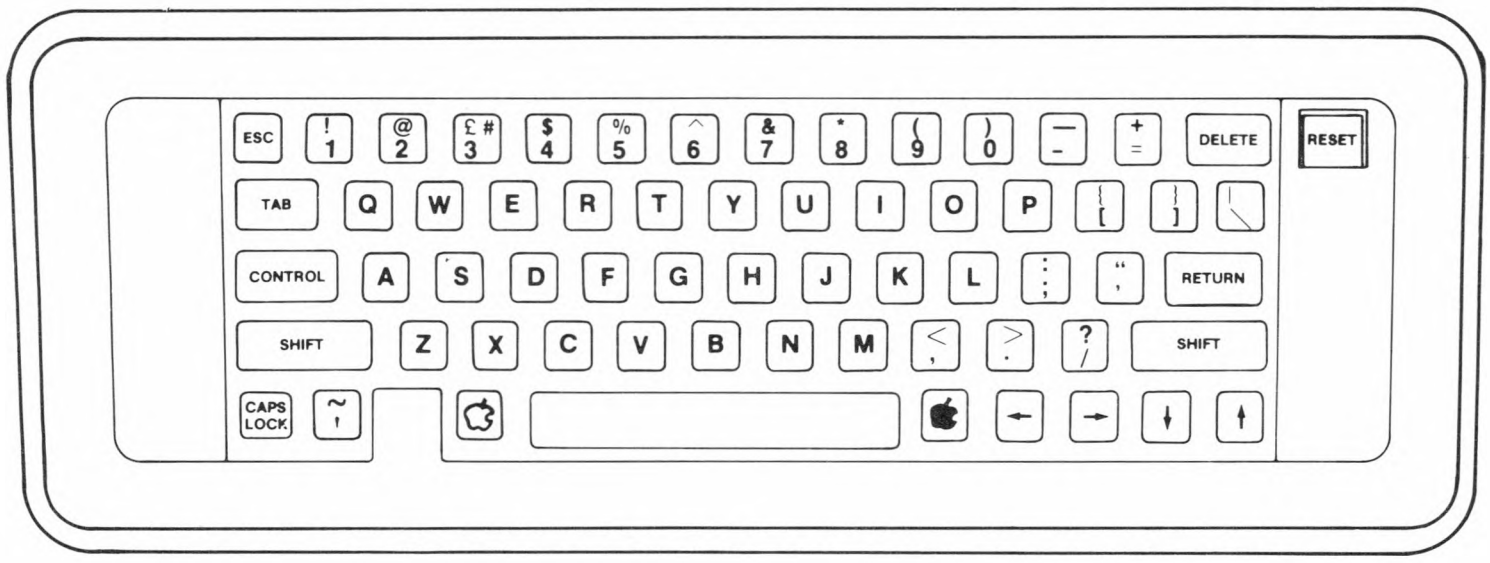


Figure 5.1. Apple *//e* Keyboard

Keys You Must Use Precisely

The keys described in the "Keys You Must Use Precisely" section of the *Apple //e* Owner's Manual must be used exactly as described in that section. The only exception is the DELETE key. The DELETE key has been redefined by the SoftCard II system as the CP/M key RUBOUT.

Cursor Movement Keys

The left cursor key (←) and the DELETE key delete characters as they move over them. The TAB key moves the cursor seven spaces to the right. The TAB and RETURN keys are used in the same manner with CP/M as with Apple DOS and Apple Pascal.

Unless special software is provided by an application program, CP/M does not support cursor movement with the right, up, or down cursor keys.

Apple Escape Key Sequences

CP/M does not support Apple DOS ESC key sequences for cursor movement or editing. The SoftCard II version of CP/M does, however, support two ESC key sequences. These are ESC-(and ESC-) and they are part of the default screen function interface. Pressing the ESC-(keys and then RETURN switches the screen display to inverse video (dark characters on a light background.) Pressing ESC-) and RETURN switches the screen back to the normal display.

Apple *//e* Special Function Keys

The OPEN-APPLE and SOLID-APPLE keys are used as described in the *Apple //e Owner's Manual*. That is, pressing the OPEN-APPLE key has the same effect as pressing the button on hand-control 0; and pressing the CLOSED-APPLE key has the same effect as pressing the button on hand-control 1.

Pressing the RESET key while you are at CP/M command level will not have any effect. Pressing the CONTROL-RESET keys will cause a CP/M cold start.

Line Editing Commands

CP/M supports several line editing commands that allow you to edit a CP/M command line or to edit data input to CP/M transient programs. Most line editing commands are executed by using CONTROL characters. CONTROL characters (denoted by "CONTROL-") are used by first pressing the CONTROL key and then holding it down while you type the indicated character. Do not press RETURN after typing a CONTROL character. Table 5.1 lists the CONTROL characters associated with line editing commands.

Table 5.1.

Line Editing Commands

Key	Function
←	Moves the cursor one character position to the left and deletes characters as the cursor passes over them.
↓	Backspaces and deletes the entire line.
CONTROL-E	Moves the cursor to the beginning of the next line. However, the previous line is not terminated until the RETURN key is pressed. Note that the carriage return/linefeed character sequence generated by CONTROL-E is not entered into a line, but only sent to the console.
CONTROL-H	Same as the left cursor key (←).
CONTROL-J	Terminates input (linefeed).
CONTROL-M	Functions the same as RETURN.
CONTROL-P	Sends all ASCII character output to the printer and to the monitor screen. This "printer echo" mode remains in effect until CONTROL-P is typed, or until a CP/M warm start is performed. CONTROL-P is accepted only when console input is required.
CONTROL-R	Redisplays the current line.
CONTROL-S	Suspends ASCII character output to the terminal. Output resumes when any other key is pressed.
CONTROL-X	Same as the down cursor key.
DELETE	Same as the left cursor key.

Type-ahead Buffer

The SoftCard II version of CP/M has a type-ahead buffer for keyboard input. This permits you to enter commands and text while CP/M is performing other operations. When CP/M finishes an operation, it scans the type-ahead buffer for additional commands and data. This ensures that none of your input from the keyboard is lost. The type-ahead buffer holds up to 256 characters.

Using I/O Devices With CP/M

The default I/O device assignments for the SoftCard II version of CP/M are as follows:

Logical Device	Physical Device
CON:	TTY: (Apple keyboard and monitor)
RDR:	TTY: (Apple keyboard and monitor)
PUN:	TTY: (Apple keyboard and monitor)
LST:	LPT: (interface board in accessory slot 1)

The TTY: physical device communicates with the Apple keyboard and screen monitor if there is no interface board installed in accessory slot 3. If there is an interface board installed in slot 3, the TTY: physical device will communicate with the device connected to the installed board.

The LST: physical device communicates with the interface board installed in accessory slot 1. If there is no interface board installed in slot 1, the TTY: physical device is used.

To use other physical devices, special I/O software must be added to the patch areas of CP/M to define the location of the physical device (i.e., the accessory slot the physical device will communicate with). See "I/O Configuration" in the *SoftCard II Programmer's Manual* for instructions on adding I/O software to the patch areas.

Print Operations

CP/M provides several methods for sending data to a printer. The printer, however, must be connected to an appropriate interface board in the recommended accessory slot. See Table 2.1 in Chapter 2 for a list of recommended slot assignments.

Once all physical connections are made, you must make sure that the logical LST: device is assigned to the right physical device. Check the assignments with the STAT command by typing

```
STAT DEV:
```

and pressing the RETURN key. If the LST: logical device is not assigned to the LPT: or UL1: physical device, then use STAT to change the device assignment. See "I/O Communication" in Chapter 4 for an explanation of logical and physical devices. Instructions for using STAT are given in the "STAT" section in Chapter 6.

Note

The LPT: physical device is the I/O interface board in accessory slot 1. UL1: is an undefined I/O interface board.

When the LST: logical device assignment has been made, you can use either CP/M commands or the commands provided by the application program to send output to the printer.

Many application programs, such as text editors and electronic spreadsheets, have built-in print functions. The ability to send data to the printer is included as part of the program. If your program does have commands or statements for sending data or files to a printer, you should use those commands when running the application program. The major advantage of using a built-in printing program is that it usually prints your file in the format required by the application program.

If you want to print a file at CP/M command level, you can use either CONTROL-P and the TYPE command or the PIP program. Instructions on how to use TYPE and PIP are given in Chapter 6.

Running Application Programs

Most application programs have explicit instructions for loading and running the program. Others do not. If your application program is vague about how to load the software into the Apple/SoftCard system, you can follow these guidelines. If the program still can't be run, contact the computer dealer or the program manufacturer.

Guidelines for Running Application Programs

If the program is written in BASIC, it will often require the GBASIC.COM file to be loaded into memory prior to running the program.

Programs written in other languages, such as FORTRAN or COBOL, will require the appropriate language compiler or interpreter to run, unless the program is compiled into an executable (.COM) file.

Check to see how much memory the application program needs to run. The SoftCard II system has up to 58.5K bytes of memory for application programs to use. If the program requires more than that, contact your computer dealer or the program manufacturer.

Some programs will require you to re-configure the screen function interface. This is an area of CP/M that translates what the program sends to the operating system into what you actually see on the terminal.

Application programs that are in a different disk format will require that the disk be copied to the 5 1/4-inch disk format used by the SoftCard II system. Contact your computer dealer for more information.

Programs created under CP/M 2.0 and earlier versions are compatible with the SoftCard II version of CP/M.

Important

Programs written for the previous SoftCard versions of CP/M which access specific 6502 memory addresses must be changed. This includes programs that were written under Microsoft FORTRAN-80 and Microsoft BASIC Compiler. For more information about using FORTRAN and BASIC Compiler programs with the SoftCard II system, see the *SoftCard II Programmer's Manual*.

Chapter 6
CP/M Commands and Utility Programs

This chapter explains the CP/M built-in commands and utility programs you will use most often. The commands and programs that are discussed in this chapter are listed below.

CP/M Built-in Commands

d:	Logs onto another disk drive.
DIR	Displays a directory of the files on disk.
ERA	Erases a file or files.
REN	Renames a file.
SAVE	Saves the contents of memory in a file on disk.
TYPE	Displays the contents of a file on the monitor screen.
USER	Creates another area within the same directory.

Utility Programs

APDOS	Copies text and binary files from Apple DOS disks to CP/M disks.
AUTORUN	Automatically executes a CP/M command line when the system is booted.
BOOT	Exits CP/M and reboots your Apple *//e* system.
CAT	Displays an alphabetical directory listing of the drive specified.
COPY	Makes duplicate copies of disks. Options to COPY let you format disks and create CP/M system disks.
MFT	Copies files from one disk to another on a single-drive system.

PATCH	Makes program updates and modifications to CP/M.
PIP	Transfers, copies, and/or appends disk files and devices.
STAT	Displays status information and assigns devices.

Command and Program Execution

All commands and programs are executed either from CP/M *command level* or from the *program level*. The following paragraphs define both terms.

"Command level" is the CP/M command level of operation; all commands are executed through the CCP module. (See "How CP/M Uses Memory" in Chapter 4.) Command level is indicated by the CP/M command prompt (the active disk drive letter followed by the > sign).

To execute utility programs at command level, type the command line using the program name and arguments. Do not type the .COM extension. Programs executed at command level will always return control of the computer back to command level. This is useful for a single task, such as copying a single file from one disk to another.

The "program level" of operation is when a program controls the computer and permits only certain commands to be typed from the keyboard. For programs that are part of the SoftCard II system, the asterisk prompt (*) indicates that the computer is at program level. Other application or utility programs may display another character or a *menu* to indicate program level operation.

To execute commands at program level, you must first type the name of the program and then press RETURN. The program is loaded into memory and the asterisk prompt is displayed. You may then enter the command line without typing the program name. Commands executed at the program level will return control back to the program level. This is useful for repetitive tasks, such as copying more than one disk.

Using Utility Programs

SoftCard II utility programs can be executed at CP/M command level or at program level. The AUTORUN, BOOT, CAT, and STAT utility programs can be executed from CP/M command level only. The APDOS, COPY, MFT, PATCH, and PIP utility programs are run from CP/M command level or program level.

To stop or abort a utility program, press CONTROL-C. You can also use the line editing commands described in Chapter 5 to edit utility program commands.

Using CP/M Built-in Commands

CP/M built-in commands are executed from the CP/M command level only. Their use does not require that a system disk be in the active drive. However, if you encounter a "BDOS ERR ON d:" type of error, you must insert a CP/M system disk in the active drive to recover from the error and to continue.

Built-in Commands

The following section explains how to use the CP/M built-in commands. Examples are included with each command.

d:

The d: command allows you to change active drives in multiple-drive systems. The active drive is the disk drive that contains the CP/M system disk you are currently working from. (See "CP/M Disk Files" in Chapter 4.)

To change the active drive, type the letter which represents the drive you wish to designate followed by a colon (:), and press RETURN. For example,

```
B:
```

followed by pressing the RETURN key changes the active drive to drive B:. If you change the active drive, CP/M changes the prompt letter accordingly.

DIR

The DIR (DIRECTORY) command scans a specified disk to determine what files are on that disk. Typing *DIR* (with no arguments) displays only the sequential list of filenames on a disk in the specified drive. DIR can also display specified files when you use arguments in the command line.

Displaying a Disk Directory

DIR, when used in the following format, scans the disk directory of the disk in the drive specified and displays the directory entries (files) it finds.

```
DIR [d:]
```

Typing DIR without the *d*: argument scans the disk directory of the active drive. For example,

```
DIR
```

displays the disk directory of files on the active drive. To scan the disk directory of another drive, type the drive letter (*d*:) in the command line.

```
DIR B:
```

displays the disk directory of files on the disk in drive B:. If no files are found, CP/M displays the message:

```
NO FILE
```

Displaying Single and Multiple Disk Directory Entries

To find and display a specific file(s) on a disk, type DIR in the following format,

```
DIR [d:][filename.ext]
```

and press the RETURN key. The *d*: argument permits you to search other disk drives for the specified file or files. For example,

```
DIR B:GBASIC.COM
```

scans drive B: for the file GBASIC.COM.

To find a particular file, type the *filename.ext* of the desired file. If you want to display a certain type of file, type "wild card" characters (? or *) in the *filename.ext* argument. (Wild card characters are explained in "CP/M Disk Files" in Chapter 4.) Wild card characters also allow you to search for files that begin with a certain letter or share a common name. For example,

```
DIR D???.BAS
```

searches the disk directory of the active drive for all files beginning with D and having between one and four characters

in the filename with a filename extension of .BAS. Another way to use a wild card character in the filespec would be to type:

```
DIR *.COM
```

The *.COM instructs DIR to scan the active drive for all files with a .COM filename extension.

ERA

The ERA (ERASE) command erases specified files from a disk. You may use ERA to erase files from any disk as long as you include the file specification (filespec). Wild card characters can be used in the filespec.

Erasing a File

To erase a single file, type ERA in the following format,

ERA *filespec*

and press the RETURN key. The *filespec* is the name and the location of the file. For example:

```
ERA B:TEMP.OLD
```

erases the file TEMP.OLD from the disk in drive B:.

Erasing Multiple Files

To erase multiple files, include wild card characters (? or *) in the *filespec* argument of the command line. For example,

```
ERA C:*.BAS
```

erases all files with the extension .BAS from the disk in drive C:. Another example would be:

```
ERA *.*
```

This command line erases all files on the disk in the active drive. If you attempt to erase all the files on a disk, CP/M will ask: ALL (Y/N)? If you want to erase all the files on the disk, press the *Y* key. Otherwise, press either the *N* or the RETURN key.

REN

The REN (RENAME) command renames files while leaving the file text intact. Unlike the other built-in commands, wild card characters cannot be used in the filespec. Therefore, you can only rename one file at a time.

Renaming a File

To rename a file, type REN in the following format:

REN [*d*:]*new filename.ext=old filename.ext*

Press the RETURN key to execute the command. The *new filename.ext* argument is the new name of the file and *old filename.ext* is the original name of the file. For example,

```
REN TEMP.NEW=TEMP.OLD
```

renames the file TEMP.OLD as TEMP.NEW on the active drive.

If the file is on a disk drive other than the active drive, include the drive identifier (*d*:) in the command line, as in the following example:

```
REN B:PEAR.COM=APPLE.COM
```

SAVE

The SAVE command saves the contents of memory in a specified disk file. It is used mainly for program development.

Saving Memory Contents in a Disk File

You can save what you entered into memory by typing SAVE in the following format:

```
SAVE nnn filespec
```

The *nnn* argument is the number of memory pages to be saved. (A page of memory is equal to 256 bytes.) The *filespec* is the drive and the file where you will save the memory contents. For example,

```
SAVE 2 B:DATA.BIN
```

saves 2 pages of memory in a file called DATA.BIN on disk drive B:.

To use SAVE, you must know how many memory pages are to be saved. The memory pages to be saved will start at memory location 100H (hexadecimal). The *nnn* argument, however, must be entered as a decimal number. Instructions on how to convert the hexadecimal address to a decimal number are given in the *Osborne CP/M User Guide*.

TYPE

The TYPE command displays the contents of a specified text file on the screen. This provides a quick way of examining a file for errors or to check the contents. It can also be used to print a file in conjunction with the CONTROL-P line editing key.

Note

If you attempt to TYPE a file that is not a text file, meaningless characters will appear with unpredictable results.

Displaying a Text File on the Monitor Screen

To display a file on the monitor screen, enter TYPE in the following format:

TYPE *filespec*

The *filespec* is the location and name of the file. No wild card characters are allowed in the filespec. For example,

```
TYPE DUMP.ASM
```

displays the contents of the file DUMP.ASM in the active drive on the screen.

Printing a Text File With TYPE

The CONTROL-P is an on/off line editing command that controls the output to the printer. When used with the TYPE command, it permits you to print the contents of a text file while it is being displayed on the screen.

Note

CONTROL-P assumes there is a print device assigned to the LST: logical device. Before printing a file with TYPE, check the logical device assignments with the STAT command.

To print a text file, press CONTROL-P and then type the command:

TYPE *filespec*

and press RETURN. For example, press CONTROL-P, then type:

```
TYPE B:DUMP.ASM
```

Press RETURN to execute the command. This example will print the file DUMP.ASM in drive B:, as it is being displayed on the screen.

USER

The USER command separates disk memory into user areas. The user areas are designated by numbers (i.e., 0, 1, 2, and so on). This command is useful for creating multiple file directories (one per user area) on disks. However, USER is of limited value with floppy disks because of the small memory areas they contain. If you have a hard disk, USER allows you to maintain separate memory areas on the same disk.

Copying files from one user area to another is described in the "PIP" section of this chapter.

Creating a User Area

To create a new user area on the disk in the active drive, type USER in the following format,

USER *n*

and press the RETURN key. The *n* argument is the number (any unused number between 0 and 15) of the new user area.

If the specified number hasn't been used on that particular disk, USER creates the user area for that number. For example, if you already have three user areas (0, 1, 2) and you wish to create a fourth one, type

USER 3

and press RETURN. This immediately creates user area 3 and transfers you to that area.

Changing the Active User Area

To change the active user area, type

USER *n*

and press the RETURN key. The *n* argument is the number of the desired user area. For example, type

USER 0

to change the active user area back to area 0.

Note

If you attempt to execute a program in a .COM file which is not in the current user area, CP/M automatically searches user area 0 for that file. This applies to .COM (Command) files only.

Utility Programs

The following section explains the CP/M utility programs included in the SoftCard II package. These are the programs that you will use most often. Examples are given with each program.

APDOS

APDOS.COM is a utility program that copies Apple text and data files from Apple DOS disks to CP/M system disks.

Note

Apple DOS text and data files are usually incompatible with CP/M. You can, however, copy the files to CP/M system disks and modify them with a text editor.

APDOS can copy files from CP/M command level or from the APDOS program level. From CP/M command level, the command line is typed and then executed by pressing the RETURN key. APDOS will exit to CP/M command level after each execution.

To run APDOS at the program level, type *APDOS* and press RETURN. The program level command line can then be executed when the asterisk prompt appears. APDOS will return to the program level prompt after executing the command line.

Copying Single Apple DOS Data and Text Files to CP/M

Use the following procedure to copy a single Apple DOS data or text file to CP/M. This procedure assumes that you are at CP/M command level.

1. Put a CP/M system disk with the file APDOS.COM into drive A:.

2. Type the name of the file to be transferred in the following format,

APDOS [*d*:]*cp/mfilename.ext*=[*s*:]*dosfilename*

where *d*: specifies the destination disk drive and *s*: specifies the source disk drive. For single-drive systems, enter *A*: for both source and destination drives.

The *cp/mfilename.ext* argument is the name of the CP/M destination file and *dosfilename* is the name of the Apple DOS file you wish to copy. When you have typed the command line, press the RETURN key to execute the command.

3. When APDOS has been loaded into memory, it will then display:

INSERT APPLE DOS DISK IN DRIVE *s*:
INSERT CP/M DISK IN DRIVE *d*:
PRESS RETURN TO BEGIN

Follow the instructions shown on the screen and insert the disks into the appropriate drives. For single-drive systems, you will have to change the disks in drive A: after the Apple DOS file has been copied into memory.

When the copy process has been completed, APDOS displays the message

TRANSFER COMPLETE

and exits to CP/M command level.

4. Type

DIR *filespec*

and press the RETURN key to verify that the Apple DOS file has been transferred to your CP/M disk.

Copying Multiple Apple DOS Data and Text Files to CP/M

Use the following procedure to copy multiple Apple DOS data and text files to CP/M. The procedure assumes that you are starting at CP/M command level.

1. Put a CP/M system disk with the file APDOS.COM into drive A:.

2. Type *APDOS* and press RETURN. When you see the asterisk prompt, type the name of the file to be transferred in the following format:

   [*d*:]*cp/mfilename.ext*=[*s*:]*dosfilename*

   cp/mfilename.ext is the name of the CP/M destination file and *dosfilename* is the name of the Apple DOS file you want to copy. When you have typed the command line, press the RETURN key to execute the command.

3. When APDOS is loaded into memory, it will display:

   INSERT APPLE DOS DISK IN DRIVE *s*:
   INSERT CP/M DISK IN DRIVE *d*:
   PRESS RETURN TO BEGIN

   Follow the instructions shown on the screen and insert the disks into the appropriate drives. For single-drive systems, you will have to change the disks in drive A: during the copy process. The screen will display instructions for doing so.

   When the copy process has been completed, APDOS displays the message

   TRANSFER COMPLETE

   and displays the APDOS program prompt. Repeat steps 2 and 3 for each Apple DOS file you plan to copy.

4. To exit APDOS, press CONTROL-C.

5. Type

   DIR *filespec*

   and press the RETURN key to verify that the Apple DOS file has been transferred to your CP/M disk.

Copying Apple BASIC Programs to CP/M

The format of Apple DOS BASIC files (Applesoft BASIC or Integer BASIC) must be modified before they can be copied with the APDOS program. Use the following procedure to modify and copy Apple DOS BASIC files to a CP/M disk.

1. Insert the source Apple DOS disk into drive A:.

2. Load Apple DOS into memory by pressing the CONTROL and RESET keys or by turning on the power. When the Apple DOS prompt (*) appears, type

   LOAD *filename*

   and press the RETURN key to execute the command.

3. Type *LIST* to obtain a listing of the file. After the listing appears, type the following line as the first line of the program:

```
0 PRINT"CHR$(4)+"OPEN APPLEPROG":PRINT"CHR$(4)
WRITE APPLEPROG":POKE 33,33:LIST:PRINT
"CHR$(4) CLOSE":END
```

Note

The program line shown above should be typed without pressing the RETURN key. It may appear to be two lines as you type it, but it is actually only one line in the BASIC program.

Press the RETURN key when finished.

4. Run the program by typing

   ```
   RUN
   ```

   and then pressing the RETURN key. The program makes a text file copy of your program called APPLEPROG.

5. Remove the Apple disk from drive A:. Insert a CP/M system disk containing the APDOS program into drive A:.

6. Load CP/M into memory by typing

   ```
   PR#6
   ```

   and press the RETURN key.

7. At CP/M command level, type

   ```
   APDOS
   ```

   and press the RETURN key.

8. If you have a multiple-drive system, type

   ```
   APPLE.BAS=APPLEPROG
   ```

   at the APDOS program prompt (*), and press the RETURN key. If you have a single-drive system, type

   ```
   APPLE.BAS=A:APPLEPROG
   ```

   and press the RETURN key.

9. Follow the instructions shown on the screen.

10. Exit APDOS by pressing CONTROL-C.

11. At CP/M command level, type

    ```
    GBASIC
    ```

    and press the RETURN key.

12. When you see the GBASIC "Ok" prompt, type

    LOAD "APPLE"

    and press the RETURN key.

13. Delete line 0 (the line entered in step 3).

This completes the APDOS BASIC file copy procedure. A copy of your Apple DOS BASIC program (called APPLE.BAS) now exists on the CP/M disk and in memory. If you want to copy more than Apple BASIC file to the same CP/M disk, rename the APPLE.BAS file and repeat the procedure.

Note

Because of the differences between Apple BASIC and Microsoft BASIC, the copied program must be edited to change the Apple POKE, PEEK, CALL, and disk file statements into their equivalent Microsoft BASIC statements. See Appendix D in the *Microsoft BASIC Interpreter Reference Manual* for more information on converting programs to Microsoft BASIC.

AUTORUN

AUTORUN.COM is a utility program that permits you to create startup disks. Startup disks are system disks that automatically execute a startup program when a cold or warm start is performed.

When you create a startup disk with AUTORUN, any CP/M command line (that is, any CP/M program or command) can be automatically executed each time the system is started with either a warm or cold start. Thus, you can automatically load and run programs without ever seeing the CP/M A> prompt.

Creating Startup Disks

Use the following procedure to create a CP/M startup disk.

1. Load CP/M into memory.

2. Use the PIP utility program to copy AUTORUN.COM onto the system disk that you plan to use.

3. At CP/M command level, type the command line in the following format,

   AUTORUN [*command line*]

   and press the RETURN key. The *command line* argument is any executable CP/M program name or CP/M built-in command. For example,

   AUTORUN CAT

   displays the directory on the default drive when you boot the system.

Repeat the procedure for each startup disk.

To change the command line on a disk, type *AUTORUN* again with a new command line. Typing AUTORUN without a command line deletes the previous AUTORUN command line from the disk.

Loading Startup Disks

To execute a startup disk, it must be loaded in the active drive (usually drive A:). When you start the system, the command line will be executed immediately after the CP/M operating system modules are loaded into memory. For example,

AUTORUN GBASIC PROG

loads and runs GBASIC after CP/M. The BASIC program PROG is loaded and executed after GBASIC.

BOOT

BOOT.COM is a program that reboots your Apple][or *//e* computer from any system disk at CP/M command level. BOOT performs the same function as a CP/M "cold start." It can boot Apple DOS, Apple Pascal, Applesoft BASIC, Integer BASIC, or any Apple][or *//e* application software disk.

Loading CP/M With BOOT

Insert a system disk that contains BOOT.COM into the active drive. Type

```
BOOT
```

and press the RETURN key. BOOT executes the cold start loader in ROM which loads CP/M.

Note

When BOOT is executed, the operating system is reloaded and all programs that were in memory are erased.

Loading Other Operating Systems With BOOT

To load any other operating system, insert a CP/M system disk containing BOOT.COM into the active drive. Type BOOT in the following format,

```
BOOT [{number|M}]
```

and press the RETURN key. The *number* argument is the slot number (4, 5, or 6) of the disk controller board connected to the disk drive you are loading from. If you load the operating system from drive A: or B:, the number can be omitted. (The disk controller board for drives A: and B: is installed in slot 6.)

The *M* argument allows you to boot from the Apple Monitor in ROM. (The Apple Monitor is the Applesoft or Integer BASIC interpreter in ROM.)

After you have pressed RETURN, the screen displays:

```
INSERT DISK AND PRESS RETURN TO REBOOT SYSTEM:
```

Insert the system disk into the appropriate drive. If you are loading an Apple DOS 3.2 disk, press the *3* key. If you are loading an Apple DOS 3.3 disk, press the RETURN key.

After you press the appropriate key, the operating system will be loaded into memory. This will be indicated by a logon message. For example, if you have a CP/M disk in drive A: (the active drive), and want to load Apple DOS from drive B:, type

```
BOOT 6
```

and press RETURN. When the BOOT prompt appears on the screen, press the RETURN key. Apple DOS 3.3 should be loaded into memory and the following logon message is displayed:

```
APPLE ][

DOS 3.3 VERSION 3.3 SYSTEM MASTER

JANUARY 1, 1984

COPYRIGHT APPLE COMPUTER INC., 1980, 1982

BE SURE CAPS LOCK IS DOWN

]
```

CAT

The CAT utility program is similar to the DIR command. It scans the directory of a disk to determine which files are on that disk. The list displayed by CAT, however, is in alphabetical order and shows the size of each file and the amount of remaining unused disk space (in kilobytes).

Displaying a Disk Directory

CAT, when used in the following format, scans the directory of a disk in the specified drive and displays all the directory entries (files) it finds.

```
CAT [d:]
```

Typing CAT without the *d*: argument scans the disk directory of the active drive. For example, to scan the SoftCard Master disk in the active drive, type

```
CAT
```

and press the RETURN key. The CAT program would display the files as follows:

```
APDOS    .COM  2K  | CONFIGIO .BAS  7K  | ED      .COM  7K  | PIP     .COM  8K  |
ASM      .COM  8K  | COPY     .COM  2K  | GBASIC  .COM  26K | STAT    .COM  6K  |
AUTORUN  .COM  1K  | DDT      .COM  5K  | LOAD    .COM  2K  | SUBMIT  .COM  2K  |
BOOT     .COM  1K  | DUMP     .COM  5K  | MFT     .COM  2K  | XSUB    .COM  1K  |
CAT      .COM  1K  | DUMP     .ASM  1K  | PATCH   .COM  1K  |

Total of 88K bytes in 19 files, 38K bytes available
```

Compare the result of the CAT command to that obtained by the DIR command (see the "Built-in Commands" section of this chapter).

To scan the directory of another drive, include the drive identifier (*d*:) with the CAT command. For example,

```
CAT B:
```

scans the disk directory in drive B:. If no files are found, CAT displays the message:

```
NO FILE
```

Displaying Single and Multiple Disk Directory Entries

To find and display a specific file or files on a disk, type CAT in the following format,

```
CAT filespec
```

and press the RETURN key. The *filespec* argument is the name and location of the file or files sought. For example,

```
CAT B:GBASIC.COM
```

scans drive B: for the file GBASIC.COM. If the file is found, CAT displays:

```
GBASIC.COM
Total of 26K bytes in 1 file, 38K bytes available
```

To search for a type of file, use wild card characters (? or *) in the filename and extension part of the *filespec* argument. (Wild card characters are explained in "CP/M Disk Files" in Chapter 4.) Wild card characters also allow you to search for files that begin with a certain letter or share a common name. For example,

```
CAT A???.BAS
```

searches the disk directory of the active drive for all files beginning with the letter A having between one and four characters in the filename, with a filename extension of .BAS. Another way to use a wild card character in a filename would be to type

```
CAT *.COM
```

which scans the active drive for all files with a .COM filename extension.

Note

When CAT is used with drives other than Apple Disk][drives (such as a hard disk), the displayed file size will differ from that shown by the STAT program. CAT displays the actual size of the file, while STAT displays the amount of space on the disk allocated by the file. The latter figure may be larger.

COPY

COPY.COM is a utility program that copies and formats CP/M disks. By using its software "switch" options, you can:

Format a disk with the /F switch.

Copy the contents of one disk onto another (no switch needed).

Verify that the copied disk contents match the contents of the original disk with the /V switch.

Create CP/M system disks with the /S switch.

Create CP/M data disks with the /D switch.

Like other utility programs, COPY can be used from CP/M command level or from the COPY program level.

Formatting a Disk

Disk formatting prepares the disk to store data in a certain format. Whenever CP/M system disks or data disks are created, they are formatted automatically. This is also true if you copy the entire contents of one disk onto another. To perform the formatting function only, for just one disk, type the COPY command line in the following format from CP/M command level:

```
COPY d:/F
```

The *d*: argument specifies the drive of the disk to be formatted. The /F is a software switch that instructs COPY to format the disk only. For example,

```
COPY B:/F
```

formats the disk in drive B:.

To format disks from the program level, type *COPY* and press RETURN. When you see the asterisk prompt, type:

```
d:/F
```

In either command, the formatting process is the same. If you plan to format several disks at a time, use COPY from the program level.

Copying CP/M Disks

The most common use of COPY is to copy the entire contents of one disk to another. To copy disks from CP/M command level, type the COPY command line in the following format:

```
COPY d:=s:[/V]
```

Press the RETURN key to execute the command. The *d*: argument is the *destination drive* (the drive you wish to copy to, A: through D:). The *s*: argument is the *source drive* (the drive you

are copying from, A: through D:). The /V switch is the "verify copy" option. It instructs COPY to verify that no errors occurred during the copying process. For example,

```
COPY B:=A:/V
```

copies the contents of the disk in drive A: to the disk in drive B: and verifies the copy process by comparing the data contents of the two disks.

To use COPY from the program level, type

```
COPY
```

and press the RETURN key to load the COPY program into memory. When the asterisk prompt (*) appears, use the same command line arguments as used at CP/M command level. Pressing the RETURN key at program level executes the command. For example, if you first type *COPY* and press RETURN, the command line

```
B:=A:/V
```

will perform the same copy process as in the previous example, but it will be executed from the program level. When the copy process has been completed, COPY returns to the program level.

While the COPY program is running, it will give you instructions for each step of the copy process and status messages. See the section entitled "Backing Up the SoftCard Master Disk" in Chapter 3 for examples of screen instructions and messages.

Note

You can use COPY with either a single-drive system or a multiple-drive system. With a single-drive system, you must specify the destination drive and the source drive as the same drive.

Creating CP/M System Disks

A *CP/M system disk* contains the CP/M operating system and can be loaded into memory with a warm start or a cold start from drive A:.

CP/M system disks created with the /S switch include only the CP/M operating system software with the CP/M built-in commands; they do not include CP/M utility programs. Utility programs must be copied onto a CP/M system disk with the MFT or PIP programs.

To create CP/M system disks, use COPY in the following format:

```
COPY d:/S[/F][/V]
```

Press the RETURN key to execute the command. The *d*: argument is the destination drive (A: through D:). /S is a software switch that instructs COPY to copy only the CP/M operating system onto the first three tracks of the disk. The /F switch formats the disk and the /V switch verifies the copy process. For example,

```
COPY B:/S
```

copies the operating system software from the disk in drive A: onto the disk in drive B:.

To use COPY from the program level, type

```
COPY
```

and press the RETURN key to load the COPY program into memory. When the asterisk prompt (*) appears, use the same command line arguments as used at CP/M command level. Pressing the RETURN key at program level executes the command.

Note

When you include the /S switch in the COPY command line, COPY will format the disk if it hasn't been formatted previously. If the disk is already formatted, the files on the disk are not deleted unless the /F switch is used.

Creating CP/M Data Disks

Data disks are disks that have no operating system data on them. They are used for the storage of programs and data files only.

Important

You should avoid using data disks in drive A: and in single-drive systems. The lack of an operating system on data disks prevents CP/M from performing a warm start and recovering from errors.

To create CP/M data disks, use COPY in the following format:

```
COPY d:/D[/F][/V]
```

Press RETURN to execute the command. The /D switch instructs COPY to create a data disk. As with the /S switch, if the disk has been formatted, the files on the disk will not be deleted unless the /F switch is used. If the disk is already a CP/M system disk, the CP/M system is removed, and an additional 12K bytes of disk space is made available for programs and data. The /V switch verifies the copy process.

To create CP/M data disks from the program level, type

```
COPY
```

and press the RETURN key to load the COPY program into memory. When the asterisk prompt (*) appears, enter the command line in the following format:

```
d:/D[/F][/V]
```

To execute the command line, press the RETURN key. The arguments are used in the same manner as from CP/M command level.

MFT

MFT.COM is a utility program for copying files from one disk to another. MFT is similar to the PIP program, but is designed specifically for single-drive systems. It can be run from either CP/M command level or from MFT program level.

Copying Files from CP/M Command Level

Before using MFT, make sure the file MFT.COM is on the system disk you plan to use. To copy files at CP/M command level, type MFT in the following format:

```
MFT filename.ext1[,filename.ext2...]
```

The *filename* arguments following MFT are the names of the files you want to copy. Wild card characters (? and *) can be used in the filenames or filename extensions. The copy process is started when you press the RETURN key. For example,

```
MFT *.COM
```

copies all .COM files on the source disk to the destination disk at CP/M command level.

While the MFT program is running, it will display instructions on the screen for changing disks. Whenever you want to cancel the copy process, press CONTROL-C.

Important

You must have a CP/M system disk in disk drive A: before pressing CONTROL-C. CP/M initiates a "warm start" whenever CONTROL-C is typed from the keyboard. If you don't have a CP/M system disk in the disk drive, CP/M cannot restart itself and displays "A BDOS ERROR on d: Bad Sector" on the screen.

Copying Files from MFT Program Level

To use MFT at the program level, type *MFT* and press the RETURN key. When the MFT program is loaded into memory, the screen displays the MFT program banner and the asterisk prompt. At the asterisk prompt, type the MFT arguments in the same manner as you would from the CP/M command level. For example, if you load MFT and press RETURN, then type,

```
GBASIC.COM,CONFIGIO.BAS
```

MFT copies the GBASIC.COM and CONFIGIO.BAS files from the source disk to the destination disk.

PATCH

PATCH.COM is a utility program for installing program updates and modifications to the CP/M system modules.

The only time you should have to use PATCH is when you receive explicit instructions from a software vendor, such as Microsoft. If you wish to install your own modifications or updates without instructions from a vendor, do so at your own risk. General instructions for using PATCH are given in the *SoftCard II Programmer's Manual*.

PIP

PIP (Peripheral Interchange Program) is one of the most frequently used CP/M programs. The primary purpose of PIP is to transfer data between devices. Its most frequent use is in copying files from one disk to another, but PIP can also be used to:

Rename the destination file during the copy process

Copy files from different user areas to the active user area

Append disk files (concatenation)

Merge disk files

Send data to an output device, such as a printer or terminal

Copy data between I/O devices

The most commonly used functions are discussed in this section. PIP can also be used as an aid in program development and is discussed in Chapter 3 of the *Osborne CP/M User Guide*.

You can use PIP at CP/M command level or at the PIP program level. From either level, the command line arguments are the same. When PIP is executed from CP/M command level, it returns to CP/M command level after executing the command line. When executed from program level, it returns to the program level after executing the command line. To exit PIP from the program level, press CONTROL-C.

PIP can be aborted at any time by pressing the space bar or any other key during the copying process. PIP confirms that the process has been aborted by displaying the message "ABORTED."

Copying Files From Disk to Disk

The most common use of the PIP program is to copy files from one disk to another. To copy a file or files to another disk from CP/M command level, type the PIP command in the following format

PIP *d*:[*filespec*]=[*s*:]*filespec*[*p*]

and press the RETURN key.

To copy files from the PIP program level, type *PIP* and press RETURN. Then type:

d:*filespec*=*s*:*filespec*[*p*]

In both formats, the *d*: argument is the *destination drive* (the drive you want to copy to) and the *s*: argument is the *source drive* (the drive you are copying from). The *filespec* argument is the file specification of the file or files you are copying from. There is no need to include the destination filespec unless the file is to be renamed. For example,

```
A:=B:ED.COM
```

copies the file ED.COM from drive B: to drive A: under the same filename. If you want to rename a file, use the same command line format as before, but specify a new filespec for the destination file. For example,

```
PIP DOG.COM=CAT.COM
```

copies the file CAT.COM into a new file called DOG.COM on the disk in the active drive.

You may also use wild card characters (* or ?) in the filespec arguments to copy more than one file. The command line

```
PIP B:=*.BAS
```

copies all files on the disk in the active drive with the extension of .BAS to the disk in drive B:.

Note

If you plan to copy more than one file, use PIP from the program level. This will save time and eliminate unnecessary keystrokes.

The optional [*p*] (parameter) argument modifies the copy process or permits certain conditions to be set. If you include a parameter argument, the square brackets must enclose the parameter. For most disk to disk copy operations, parameters are not needed.

If your disk system is divided into user areas, the [g] parameter can be used to copy files from a different user area into the active user area. For example,

```
PIP A:=B:ARCH.BAS[g4]
```

copies the file ARCH.BAS from user area 4 in drive B: to the active user area.

Important

You cannot have two files with the same filename on the same disk or in the same user area.

Copying Parts of a File

PIP can copy portions of a file when used with one of two parameters. PIP with the [G*string* CONTROL-Z] parameter copies from the beginning of a file to the point denoted by the *string* CONTROL-Z. The [S*string* CONTROL-Z] parameter instructs PIP to copy a file from *string* CONTROL-Z to the end of the file. The brackets *must* be included. For example,

```
PIP B:=A:BIO.TXT[SA minute passed. CONTROL-Z]
```

copies from the point "A minute passed." to the end of the BIO.TXT file in drive A: and places the text in a file with the same name in drive B:.

Appending Files

PIP can be used to append several text files to a destination file (*concatenation*). Before files can be concatenated, there must be sufficient space on the disk to copy the files into a destination file.

To concatenate files, type PIP in the following format:

```
PIP [d:]dest=[d:]source1,source2...
```

Press RETURN to execute the command. The *dest* argument is the destination file of the copy operation. The *source* arguments are the source files. Commas must separate the source file arguments. For example,

```
PIP BOOK.TXT=CHAP1.TXT,CHAP2.TXT,CHAP3.TXT
```

copies the text files CHAP1.TXT, CHAP2.TXT, and CHAP3-.TXT into the file BOOK.TXT on the active drive.

If you are concatenating text files, no parameters are needed. If you are concatenating other types of files (e.g., hex files), an H, I, or O must be included in the command line. For example,

```
PIP PLAN.HEX=P2A.HEX,P3A.HEX,P4A.HEX[H]
```

copies the three hex source files into the file PLAN.HEX. The [H] parameter denotes a hexadecimal data transfer.

Copying Files to a Printer

PIP can copy files to I/O devices. If a printer is connected to the LST: logical device, a file can be sent to the printer.

To print a file, type the following command line:

```
PIP LST:=[d:]filespec[parameter]
```

Press RETURN to execute the command. For text files, add the parameters [T8P60] to put the output data into the proper format for printing. The T8 parameter substitutes tab stops for spaces and the P60 parameter inserts form feed characters every 60 lines. For example,

```
PIP LST:=BOOK.TXT[T8P60]
```

copies the file BOOK.TXT to the LST: device, and substitutes eight spaces for each tab stop in the file and a form feed command every 60 lines as it copies BOOK.TXT to the LST: device.

Other Uses for PIP

PIP can also be used to copy data between devices, and for copying between devices and files. For instructions on using PIP for these tasks, see the *Osborne CP/M User Guide*.

STAT

STAT.COM is the CP/M utility program for displaying status information and changing device assignments. The functions STAT performs are:

Displaying disk drive status

Displaying active disk and user area status

Displaying file status

Displaying device assignments

Changing device assignments

Assigning attributes to files and disks

Each of these functions is discussed in the following paragraphs. STAT is executed from CP/M command level only.

Disk Drive Status

Use the following format to display the status and the amount of free disk space in a specified disk drive:

```
STAT [d:]
```

For example,

```
STAT A:
```

displays the amount of free disk space for the disk in drive A:.

If you type *STAT* with no arguments, STAT will display the amount of disk space remaining on the active drive and the assigned attributes. For example, if you have the SoftCard Master disk in drive A: and type *STAT*, you will see the following display:

```
A: R/W, Space:20K
```

Active Disk and User Area Status

In the previous example, STAT displayed only the disk attributes and the amount of free space remaining. To display the statistical data for each disk drive or user area, use the following command line format:

STAT {*d*:DSK:|USR:}

The DSK: argument permits you to display disk characteristics of the active disk drive, and the USR: argument displays the current and active user areas. For example,

STAT B:DSK:

displays:

```
   B:    Drive Characteristics
1120:    128 Byte Record Capacity
 140:    Kilobyte Drive Capacity
  48:    32 Byte Directory Entries
 128:    Records/Extent
   8:    Records/Block
  32:    Sectors/Track
   3:    Reserved Tracks
```

Status Information About Files

To display status information about files, type

STAT *filespec*

and press the RETURN key. The *filespec* argument is the name and location of the file or files you want status information on. To obtain status information on more than one file at a time, use wild card characters in the filespec. In either case, STAT

will display the size of the file (or files) in both bytes and records; the number of extents the file contains; the file attribute set; and the filename itself. For example,

```
STAT DUMP.BAK
```

displays the size and attributes of the DUMP.BAK file on the active drive in the following format:

```
Recs      Bytes      Ext      Acc
  33        5K         1      R/W A:DUMP.BAK
Bytes Remaining On A: 20K
```

Assigning Attributes to Files and Disks

You can use STAT to set certain conditions for accessing files or disks. For example, you can make a file or disk a read-only file (a file or disk that can be read from, but not changed).

To change the attributes of a file or disk, type

```
STAT {d:|filespec}$attribute
```

and press RETURN. The *attribute* argument assigns one of the attributes from the list below to the file or disk. For example,

```
STAT B:DOG.COM $R/O
```

assigns the Read-Only attribute ($R/O) to the file DOG.COM on drive B:.

Attribute	Explanation
$R/O	(Read Only) Prevents writing to or deleting the file.
$R/W	(Read/Write) Allows writing to and deleting the file. This attribute cancels $R/O.
$SYS	(System) Prevents the display of file when the DIR built-in command is invoked.
$DIR	(Directory) Cancels the $SYS attribute.

Assigning I/O Devices

One of the strong points of CP/M is that you can change the I/O device assignment of your system without having to remember the exact slot assignment of each I/O device. CP/M provides four logical device names that can be assigned to any number of I/O devices by using the STAT program. (For an explanation of physical and logical devices, see the section entitled "I/O Communication" in Chapter 4.) To make device assignments, type:

```
STAT log:=phy:
```

The *log*: argument is the logical device and *phy*: is the physical I/O device. For example,

```
STAT CON:=TTY:
```

assigns the physical device TTY: to the logical device CON:.

To see the possible device assignments for your system, type

```
STAT VAL:
```

and press the RETURN key. STAT displays the possible STAT command arguments and device assignments. To see the current device assignments, type

```
STAT DEV:
```

and press the RETURN key.

Index

Microsoft® SoftCard™ II

for Apple® II, II Plus, and *//e* Computers

Programmer's Manual

Microsoft Corporation

If you have comments about this documentation or the enclosed software, complete the Software Problem Report at the back of this manual and return it to Microsoft.

Document No. 8821-22X-00

Contents

Introduction

This is the *Programmer's Manual* for the Microsoft® SoftCard™ II system. It is designed to give you the information you need to:

Use the CP/M® operating system calls to perform I/O and disk operations

Use 6502 BIOS calls to perform low-level I/O and disk operations

Use the CONFIGIO program to modify your CP/M I/O module for nonstandard I/O devices

Reference SoftCard utility programs, CP/M commands, and utilities such as ASM, DDT, and ED

This manual is for system and application programmers who plan to write or modify programs for the CP/M Apple® *//e* with SoftCard programming environment. No tutorial information is provided. We assume that you already know how to program in either assembly language or another high-level language.

We also assume you have read the *Microsoft SoftCard II Installation and Operation Manual* and are now familiar with the CP/M operating system, its commands, and attendant utility programs. Tutorial information about CP/M and its programming utilities is given in the *Microsoft SoftCard II Installation and Operation Manual* and the *CP/M User Guide*.

Specifically, this manual is for users who want to:

Implement their own software routines in the 6502 BIOS module

Write assembly programs that will run in the TPA area of memory

Use CP/M system calls from within their program

Connect nonstandard devices to their system

Change the I/O configuration

Important

This manual does not show how to change the BIOS module. If your application requires changing any of CP/M system modules (other than the patch areas provided), we recommend purchasing the Digital Research *CP/M Technical Manual*. Vendors needing more information for interfacing their products to the SoftCard II system should contact Microsoft Corporation directly.

Further Reading

If you would like to know more about programming, we suggest you read any of the following:

Barbier, Ken. *CP/M Assembly Language Programming*. Englewood Cliffs, NJ.: Prentice-Hall, 1983.

Spracklen, Kathe. *Z-80 and 8080 Assembly Language Programming*. Rochelle Park, NJ.: Hayden Book Company, 1979.

Leventhal, Lance A. *Z80 Assembly Language Programming*. Berkeley, CA.: Osborne/McGraw-Hill, 1979.

In addition to the books listed, there are several magazines and magazine columns for CP/M programmers. Here are just a few:

Microsystems. The CP/M User's Journal. Ziff-Davis Publishing, New York, NY.

CP/M Review. CP/M Review Company, Mercer Island, WA.

"SoftCard Symposium," *Softalk Magazine*. Softalk Publishing, North Hollywood, CA.

"CP/M Exchange," *Dr. Dobb's Journal*. Peoples Computer Company, Menlo Park, CA.

How to Use This Manual

This manual serves as: a reference manual for using CP/M commands and programs, and a technical manual for programming in the SoftCard II environment. Information in this *Programmer's Manual* is organized into the following chapters and appendices:

Chapter 1, "Elements of CP/M," describes the different elements of CP/M and how it is organized.

Chapter 2, "Programming Considerations," describes how to use the CP/M system calls and provides other pertinent information about programming in the Apple *//e* and SoftCard environment.

Chapter 3, "CP/M System Calls," is a reference section for the 39 CP/M system calls. It includes a listing of the parameters needed for each call.

Chapter 4, "6502 BIOS," is a reference section for the seventeen 6502 BIOS system calls.

Chapter 5, "Command Directory," is a quick reference guide to the CP/M commands and utility programs contained in the SoftCard II system.

Chapter 6, "I/O Configuration," explains the different I/O functions and tells how to add I/O drivers to patch areas.

Appendix A, "Error Messages," lists and explains the error messages that may be encountered in using CP/M and its utility programs.

Appendix B, "SoftCard Version Differences," explains what you should know about the SoftCard implementation of CP/M and explains the differences between the standard or "generic" implementation of CP/M version 2.2 and the SoftCard implementation, version 2.25.

Appendix C, "80-Column Operation and the SoftCard II" describes what you should know about using 80-column display boards with SoftCard equipped Apple][computers.

"Glossary," defines terms used in CP/M and SoftCard documentation.

Notation Used in This Manual

This manual uses the same notation as the *Microsoft SoftCard II Installation and Operation Manual* to demonstrate the differences between what you enter on the keyboard and what you see in the manual. The following elements are used in this manual to help you understand how commands are entered into the computer.

ital	Italics indicate information that you enter. Italicized lowercase text is for an entry that you must supply, such as a *filename*.
[ ]	Square brackets indicate that the enclosed entry is optional.
{ }	Braces indicate a choice between two or more entries. At least one of the entries enclosed in braces must be chosen, unless the entries are also enclosed in square brackets.
|	Vertical bars separate choices within braces.
. . .	Ellipses indicate that an entry can be repeated as many times as needed or desired.
CAPS	Capital letters not enclosed within the other elements of syntax indicate portions of commands that must be entered exactly as shown, such as command keywords. Small capital letters indicate that you must press a key named by the text. For example, "press the RETURN key."

All other punctuation, such as commas, colons, slash marks, and equal signs, must be entered exactly as shown.

European Apple *//e* Differences

On the European version of the Apple *//e* computer, the following keys display symbols on the key face, instead of key names.

United States Version	European Version
TAB	→
RETURN	←
SHIFT	↑

In addition to these keys, the CONTROL key is labeled "CTRL", and the DELETE key is labeled "DEL". The *Installation and Operation Manual* and *Programmer's Manual* refer to the keys by their American key names.

The European Apple *//e* has two character sets: a standard ASCII character set, and a character set indigenous to a particular country. You can switch between the character sets by switching the toggle located under the righthand side of the keyboard.

Note

The *Installation and Operation Manual* and *Programmer's Manual* assume the toggle switch is set for the ASCII character set. If it is not, some character substitutions may appear on the screen.

Chapter 1
Elements of CP/M

This chapter describes the different elements of CP/M as implemented by the SoftCard II system.

CP/M Memory Organization

The SoftCard II version of CP/M (version 2.25) consists of three software modules (the BIOS, BDOS, and CCP) and various system parameters. CP/M software resides on disk in system tracks zero through two.

CP/M is loaded into the 64K of random access memory located on the SoftCard circuit board. In memory, CP/M occupies the locations shown in the following figure.

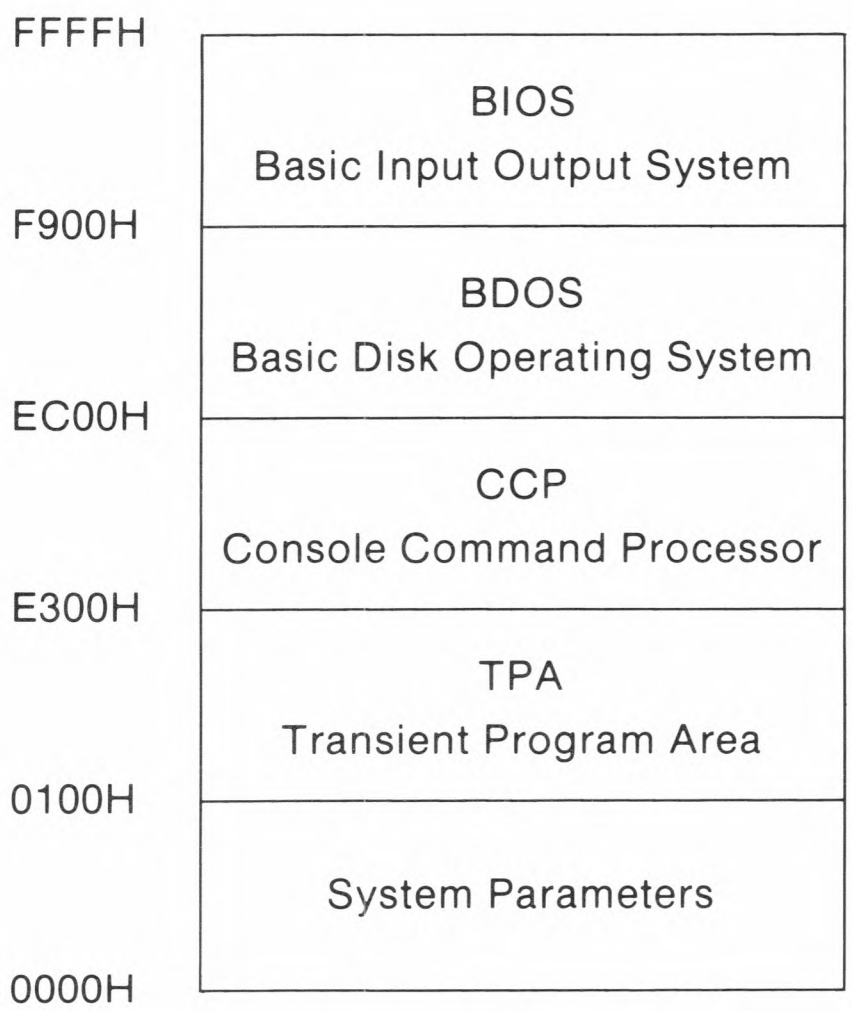


Figure 1.1. CP/M Memory Organization

BIOS (Basic Input and Output System)

The BIOS module in the SoftCard implementation of CP/M has the following features added:

"Patch" areas for implementing additional software or for interfacing nonstandard I/O devices

Entry points for using 6502 subroutines

Tables for modifying screen functions for different hardware and software configurations

A table for redefining the ASCII values of the keys on the keyboard

System calls to the 6502 BIOS

BDOS (Basic Disk Operating System)

The SoftCard implementation of CP/M uses the standard CP/M BDOS module for system calls and other disk I/O routines. The standard 39 system calls of CP/M version 2.2 are implemented through a jump table in the BIOS. (See Chapters 2 and 3 for more information on system calls.)

CCP (Console Command Processor)

The SoftCard implementation of CP/M uses the standard CP/M CCP module as an operator interface to the screen monitor and keyboard.

The CCP can be overwritten by a program to gain an additional 2K bytes of memory if the program requires it. If the CCP is overwritten by a program, it can be reloaded into memory by pressing CONTROL-C.

TPA (Transient Program Area)

The TPA in the SoftCard version of CP/M occupies approximately 59K bytes of memory between the addresses shown in Figure 1.1.

Programs that overwrite the CCP must end with a System Reset, system call 0, or a JMP instruction to the BIOS entry point (address 0000H).

System Parameters

The system parameter area of memory is initially loaded with the cold start loader program and then used as a system work area. Table 1.1 shows the location and contents of routines stored in this area of memory.

Table 1.1.

System Parameter Area Contents

Memory Address	Contents
0000H to 0002H	Z80 jump vector to the BIOS jump table (used during a warm start).
0003H	IOBYTE address, which is a single byte used for logical to physical device assignment. See "I/O Communication and the IOBYTE" in this chapter.
0004H	A single byte which indicates the active drive (drive a=0, b=1, c=2, and d=3). The default value is 0 when loading the system during a cold start.
0005H to 0007H	The Z80 jump vector to the BDOS entry point. It is used by transient programs when making system calls to the BDOS.
0008H to 0037H	Reserved for future use, but not used at this time.
0038H to 003AH	Vector address if a Restart 7 instruction is encountered.
003BH to 003FH	Reserved for future use, but not used at this time.
0040H to 004FH	Reserved for CP/M (See Chapter 4, "6502 BIOS").
0050H to 005BH	Reserved for future use, but not used at this time.
005CH to 007CH	Default File Control Block (FCB) for disk operations. (See "The CP/M File Structure" in this chapter.)
007DH to 007FH	Default random record positions for the file named in the FCB.
0080H to 00FFH	Optional 128-byte disk buffer used during disk file accesses. It is also used to store the command line being entered when the CCP is active.

CP/M Operation

In most implementations of CP/M, operation is controlled by a program running in the TPA section of memory or by commands translated by the CCP from the keyboard. All program instructions or commands from the CCP are executed by function requests to the BDOS module for one or more of 16 low-level system functions called *primitives*.

A primitive function is an assembly language routine in the BDOS module which performs a disk or I/O related task such as reading a character from the keyboard or writing data to a disk file. The 16 primitive functions are divided into two groups called *character I/O* functions and *disk I/O* functions. The following table outlines the functions.

Table 1.2.

CP/M Primitive Functions

Function	Type	Description
CONIN	Character I/O	Console input
CONOUT	Character I/O	Console output
CONST	Character I/O	Console status
HOME	Disk I/O	Seek track 0
LIST	Character I/O	List output
LISTST	Character I/O	List status
PUNCH	Character I/O	Punch output
READ	Disk I/O	Read disk sector
READER	Character I/O	Reader input
SETDMA	Disk I/O	Select memory range
SELDSK	Disk I/O	Select disk drive
SETSEC	Disk I/O	Seek disk sector
SECTRAN	Disk I/O	Convert logical sector to physical sector
SETTRK	Disk I/O	Seek disk track
WBOOT	Disk I/O	System warm start
WRITE	Disk I/O	Write disk sector

As described in Chapter 4 of the *Microsoft SoftCard II Installation and Operation Manual*, all nondisk I/O communication takes place through the four logical devices: CON:, LST:, PUN:, and RDR:. Character I/O functions transfer single-byte ASCII characters between a logical device and a register in the central processing unit. The logical devices are part of the software translation interface between CP/M and the actual I/O devices.

Disk I/O functions are similar to character I/O functions but are for transferring larger amounts of data (usually 128-byte data blocks). These functions are described in "The CP/M File Structure" later in this chapter.

Each of the primitive functions can be used either individually or in combination with each other to perform the 39 function requests known as *system calls*. All system calls are designated by a number and are executed by a Z80® CALL instruction. The Z80 CALL instruction is invoked from the CCP program running in the TPA.

When system calls are executed, control of the computer is passed to CP/M. CP/M executes the function called and then returns control back to the program. For example, a program calls for a character to be sent to the terminal. At the appropriate point in the program, the character to be sent and the system call number are processed by the CPU, transferring control to a specific function routine in the BDOS module of CP/M. The function routine performs the tasks necessary to cause the character to be displayed at the terminal. The last instruction of the assembly language routine tells the CPU to return control to the calling program immediately following the system call.

The use of system calls gives CP/M programs the advantage of *portability*. That is, a program can run on many different computers without program modifications for each particular computer.

System operation of the SoftCard version of CP/M differs slightly from the standard CP/M version 2.2 because the Z80 CPU uses the Apple 6502 as an I/O processor. Thus, any system calls that require I/O operations will first transfer control to the CP/M. The Z80 then "calls" the 6502 to execute the appropriate set of instructions. For CP/M programs that do not call 6502 routines directly, this entire process is "invisible." To use 6502 subroutines in your program, see "Calling 6502 Subroutines" in Chapter 2.

I/O Communication and the IOBYTE

CP/M communicates with nondisk I/O devices through four logical devices. CP/M also communicates with nondisk I/O devices through vector routines (known as physical devices) and a translation routine, if needed.

The logical device (as opposed to an actual physical device) is implemented by an assembly language subroutine that presents a logical representation of the I/O function. The logical devices are named by function in the following list:

Console (CON:)	Input and output to and from a console or terminal
List (LST:)	Output to a listing device, such as a printer
Punch (PUN:)	Output only
Reader (RDR:)	Input only

A physical device is assigned to a logical device. A physical device is addressed by a vector that points to a driver routine. There are 12 physical devices; each corresponds to a specific type of I/O device. Table 1.3, "Physical Device Descriptions," describes each of the physical devices, except for BAT:. See "Logical to Physical Device Assignments" in Chapter 4 of the *Microsoft SoftCard II Installation and Operation Manual* for more information on the BAT: physical device.

Table 1.3.

Physical Device Descriptions

Device	Description
TTY:	The TTY: device communicates with the standard Apple screen monitor and keyboard if slot 3 is empty. It communicates with an external terminal or 80-column video display board if there is an interface board installed in slot 3. The TTY: routes communication through Console Input Vector #1 and Console Output Vector #1. The console status is always input through the Console Status Vector.
CRT:	The CRT: device is defined the same as the TTY: device. A substitution patch routine must be written to redefine the device and its location before it can be used.
UC1:	UC1: is user-defined console device. It routes communication through Console Input #2 and Console Output #2. A substitution patch routine must be written to define the device and its location before it can be used.
PTR:	PTR: points to a standard Apple interface board capable of processing input from accessory slot 2. If slot 2 is empty, the PTR: device always returns a 1AH (end-of-file character) in register A, when called. Input from PTR: is through Reader Input Vector #1. Characters are returned in the A register.

Table 1.3 *(continued)*

Device	Description
UR1:	UR1: is user-defined reader device #1. A character read from this device is returned in the A register. Input is through Reader Input Vector #2. A substitution patch routine must be written to define the device and its location before it can be used.
UR2:	UR2: is user-defined reader device #2. This device has the same definition as UR1:.
PTP:	PTP: is any standard Apple interface board capable of processing output from accessory slot 2. If slot 2 is empty, the PTP: device does nothing when called. Output to the PTP: device is through Punch Output Vector #1. A substitution patch routine must be written to define the device and its location before it can be used.
UP1:	UP1: is user-defined punch device #1. The character in register C is output through Reader Input Vector #2. A substitution patch routine must be written to define the device and its location before it can be used.
UP2:	UP2: is user-defined punch device #2. This device has the same definition as UP1:.
LPT:	LPT: is any standard Apple interface board installed into slot 1 capable of receiving output. The character in register C is output through List Output Vector #1.
UL1:	UL1: is a user-defined list device. The character in register C is output through List Output Vector #2. A substitution patch routine must be written to define the device and its location before it can be used.

Because there are four logical devices, only one physical device can be assigned to a logical device at a time. The IOBYTE is used by CP/M to monitor and change the current logical to physical device assignments.

The IOBYTE is a single byte located at memory address 0003H that is divided into four two-bit fields. The fields represent each of the logical devices as shown in the following figure.

Field	LST:	PUN:	RDR:	CON:
Bits	7—6	5—4	3—2	1—0

Figure 1.2. The IOBYTE at Address 0003H

The value of the bits determines which physical device is assigned to the logical device. Table 1.4 lists the possible IOBYTE assignments.

Table 1.4.
IOBYTE Device Assignments

	Fields			
Bit Value	**LST:**	**PUN:**	**RDR:**	**CON:**
00	TTY:	TTY:	TTY:	TTY:
01	CRT:	PTP:	CRT:	CRT:
10	LPT:	UP1:	PTR:	BAT:
11	UL1:	UP2:	UR2:	UC1:

The SoftCard implementation of the IOBYTE is based on memory mapping of the seven accessory slots. Slots one through three are initially mapped to the LPT:, PTR:, and TTY: devices, respectively. To implement other physical devices, substitution I/O routines must be written into the I/O patch area of the BIOS. See "Adding Nonstandard I/O Devices and User Software" in Chapter 6 for more information.

Usually, the IOBYTE is changed with the STAT transient program. Programs, however, can also change the IOBYTE through two character I/O calls: Get IOBYTE, system call 7, and Set IOBYTE, system call 8. "I/O Device Assignment Calls" in Chapter 2 describes how to use these system calls.

Physical devices are implemented as addresses in memory that point to a vector which in turn, points to an address of an accessory board. Of the 12 physical devices, only three are mapped to an accessory board address. The other nine are either undefined or route communication to one of the implemented devices. See Table 1.3 for descriptions of the physical devices.

To use one of the unimplemented devices, a special driver routine must be written in one of the patch areas in the BIOS. Instructions on how to use the patch areas are given in "Adding Nonstandard I/O Devices and User Software" in Chapter 6.

Disk Communication

Disk communication is performed through a set of nine primitive functions, that, like the I/O primitive functions, can be called either individually or in combination with each other to perform higher-level functions. CP/M provides some of the higher-level functions through the numbered system calls that are standard in CP/M. The disk I/O functions are similar to character I/O functions but are for transferring larger amounts of data.

The File Control Block

Because the data transferred is larger than the capacity of the CPU registers, CP/M sets up two areas of memory to transfer data and parameters between the calling program and the disk. The first area is called the *disk data buffer*, and is used for disk read and write operations. It can be located anywhere in memory and occupies 128 bytes. The second area is called the *File Control Block* (FCB). It is used to pass parameters which control the disk I/O transfer between the disk and CP/M.

The FCB consists of 36 bytes and can be located anywhere in memory. It is usually located at memory address 005CH. The FCB is used for the same purpose as the CP/M registers for passing parameters.

The FCB format is shown in Figure 1.3. Each field in the FCB must contain the appropriate parameter before a disk I/O system call can be executed. The calling program provides the information in the first four fields to identify the file to be accessed. The d0—dn field is used by the BDOS module to keep track of the file contents.

Field	dr	fn	type	ex	s1—s2	rc	d0—dn	cr	r0—r1	r2
Bits	0	1—8	9—11	12	13—14	15	16—31	32	33—34	35

Figure 1.3. File Control Block

dr — Is the drive code. It identifies the drive in which the file is located.

fn — Is the filename. If the filename is less than nine characters, the remaining bytes in the field are padded with blanks.

type — Is the file type (filename extension). If the extension is less than three characters, the remaining bytes are padded with blanks.

ex — Is the current file extent number (the number of the extent that is being accessed). It is normally set to 0, but ranges between 0 and 31 during file I/O operations.

s1—s2 — Is reserved for system use. s2 is set to zero during OPEN, MAKE or SEARCH operations.

rc — Is the record count or current extent size (0 to 128 records).

d0—dn — Is the disk allocation map. This field is filled in and used by CP/M.

cr — Is the current record number (the current record to be read or written in sequential file operations).

r0—r1 — Is the random record number. The random record number (0—65535) is a 16-bit value with byte r0 as the lower 8 bits and byte r1 as the upper 8 bits.

r2 — Is the overflow byte for the random record number.

The CP/M File Structure

Chapter 4 in the *Microsoft SoftCard II Installation and Operation Manual* explains how a disk is organized into tracks and sectors. In CP/M terminology, each 128-byte disk sector is called a *record*. A disk file contains up to 65,536 records and is organized into blocks of records called *extents*.

All CP/M files contain one or more extents. An extent consists of 128 records (16K bytes). Extents allow CP/M to keep track of the physical location of the records for each file in conjunction with another unit of organization called *allocation blocks*.

To keep track of the sector's physical location on the disk, the disk is divided into allocation blocks. An allocation block consists of 8 sectors or 1024 bytes of data.

Note

The SoftCard version of CP/M uses a 5-1/4 inch disk as its primary storage medium. These disks have a total capacity of 140K bytes (or 128 sectors) of storage space. Since the CP/M system modules are stored in the first three tracks (0, 1, and 2) of the disk, the first allocation block starts with track 3, sector 1. (Tracks are numbered 0—35 and sectors are numbered 1—31.) The allocation blocks are consecutively numbered until the last sector on the disk (track 35, sector 31) has been included in an allocation block. Thus, on a 5-1/4 inch disk, there can be a total of 16 allocation blocks.

When a disk file requires additional space, an allocation block is assigned to the file through the extent field of the FCB. This gives the file an additional 1024 bytes of storage space although it may only require 64 bytes at the time. For example, if a file contains 16 records, and a disk write operation adds a seventeenth record, CP/M assigns a new allocation block to the file. The new allocation block will contain file records 17 through 24 even though only record 17 is currently written.

An extent can have up to 16 allocation blocks assigned to it. The number of each allocation block assigned to an extent is stored in the d0—dn field of the FCB (bytes 16—31), where one byte equals one allocation block.

CP/M keeps a table of all allocation blocks in memory. Whenever a file requires an additional allocation block, CP/M assigns the next available allocation block to the FCB of the file and updates the table in memory. CP/M also reclaims allocation units as a file decreases in size or is deleted. By assigning and reclaiming allocation blocks, CP/M dynamically manages the storage space on the disk. This permits the records that make up a file to be placed in random locations on the disk.

CP/M also keeps track of the files on disk through the *disk directory*. The directory is stored at track 3, sector 1, and contains an entry for each extent of each file on the disk. If a file has more than one extent assigned, the disk directory will have multiple entries for that file.

Note

When the DIR built-in command is executed, the CCP reads the disk directory but only displays the first occurence of each file.

Each directory entry is a copy of the first 32 bytes of the FCB for that given extent. As shown in File Control Block format, the first 32 bytes contain the filename, file type, the extent, and allocation block map of the extent. In the SoftCard version of CP/M, there is space allocated for 48 directory entries. Since each directory entry takes up 32 bytes, the directory takes up the first two allocation blocks of the data storage space.

Chapter 2
Programming Considerations

This chapter describes the assembly language programming tools included with the SoftCard II system. It also provides guidelines for using CP/M system calls within your programs.

Assembly Language Programming

With the SoftCard II system you may use either 8080A or Z80 assembly language programs. Although the SoftCard circuit board is designed around a Z80 microprocessor, most 8080A assembly language programs can be run by the SoftCard system without modifications. There are, however, several 8080A/Z80 compatibility characteristics that you should be aware of. These are discussed in the following sections.

Programming Tools Provided

Programming tools are software programs which permit the programmer to write and run an assembly language program or subroutine for a specific programming environment. The SoftCard II system includes the following CP/M programming tools that are standard in most CP/M implementations:

ED	CP/M text editor
ASM	8080 assembler
DDT	8080 Dynamic Debugging Tool
DUMP	Hex dump program
LOAD	8080 load program
SUBMIT/XSUB	Batch command files

Some of these programs are for the 8080A microprocessor only. Because the Z80 microprocessor uses a different set of mnemonics for instructions, the ASM, DDT, and LOAD program cannot be used with Z80 programs. The ED, DUMP, and SUBMIT/XSUB programs can be used with either instruction set.

To use the Z80 instruction set, a Z80 assembler and LOAD program are needed. The Microsoft Assembly Language Development System (A.L.D.S.®) contains the necessary programming tools in addition to several programs designed for the assembly language programmer. A.L.D.S. is available separately from Microsoft.

Instruction and Register Differences

A Z80 microprocessor can use the P flag of the F (Flags) register to indicate two's complement overflow after arithmetic operations. An 8080A microprocessor will always use this flag for parity.

The DAA instruction is executed differently by the Z80 and 8080A. The Z80 DAA instruction corrects decimal subtraction as well as decimal addition. The 8080A DAA instruction only corrects decimal addition.

Z80 "rotate" instructions, when executed, clear the AC flag in the F register. The 8080A "rotate" instructions do not.

Instruction Execution Time

The time it takes to execute an instruction differs for the 8080A and the Z80 microprocessors. In addition, the Z80B microprocessor executes instructions three times faster than its predecessors. 8080A and Z80A programs that depend on precise timing loops should be rewritten for the faster execution speed of the Z80B.

6502 BIOS Calls

The Z80 performs I/O operations through the 6502 microprocessor by accessing a set of 17 function request routines called the "6502 Basic Input Output System," or 6502 BIOS. The 6502 BIOS calls were implemented as a means of accessing the Apple 6502 memory when running CP/M programs.

6502 BIOS calls are accessed by storing information in a seven-byte area located between memory addresses 0045H—004BH, and then performing a Z80 CALL instruction to memory location 0040H. Information from the I/O system is returned in the same seven-byte area.

6502 BIOS calls should be used only when there is a need to access the 6502 memory for Apple specific functions such as game ports, 6502 subroutines, or routines for creating music. Programmers should use CP/M system calls whenever possible.

Guidelines for Use

To use 6502 BIOS calls in programs, the following protocol must be observed. The protocol governs the passing of information between the 6502 BIOS and the calling CP/M program.

1. Enter the 6502 call number in location 49H.

2. Store the needed parameters in the indicated memory location.

3. Perform an assembly language CALL instruction to location 40H.

4. If applicable, read the returned information from the indicated memory location.

6502 BIOS Call Example

The following example shows how a 6502 BIOS call is made.

```
;
;SUBROUTINE TO READ THE VALUE OF PADDLE
;ZERO INTO REGISTER A.
;
;DEMONSTRATES 6502 SUBROUTINE CALLING
;CONVENTIONS AND PARAMETER PASSING.
;
XREG      EQU     46H         ;X register pass area
YREG      EQU     47H         ;Y register pass area
CMD       EQU     49H         ;6502 BIOS command
ADDR      EQU     4AH         ;Place to store 6502 sub address
X6502     EQU     40H         ;6502 transfer address
GOSUB     EQU     0           ;CMD 0—GOSUB 6502
PADDLE    EQU     0FB1EH      ;Location of paddle routine
PDL:      XRA     A           ;Set for paddle zero
          STA     XREG        ;XREG=paddle number
          LXI     H,PADDLE    ;Address of monitor routine
          SHLD    ADDR        ;Set the address
          MVI     A,GOSUB     ;We want to execute a 6502 subroutine
          STA     CMD         ;Set the command
          CALL    X6502       ;Call the routine
          LDA     YREG        ;Get the paddle value
          RET                 ;Home, James
```

Using CP/M System Calls

The following section describes how to use the CP/M system calls from your program.

Calling From an Assembly Language Program

To use CP/M system calls in programs, the following protocol must be observed. The protocol governs the passing of information between CP/M and the calling program in the TPA.

1. The calling program must enter the number of the system call in register C of the CPU.

2. For single-byte output data, the calling program must place the data byte in register E.

3. 16-bit data is either sent or read to a pair of registers (usually registers DE) by the calling program. See Chapter 3 for specific information about each system call.

4. Data longer than 16-bits is placed in an area of memory called a *parameter block*. The address of the parameter block is placed in the DE or HL register pair.

5. The calling program must issue a CALL 0005 instruction or equivalent.

6. The calling program reads register A for single-byte input values.

Assembly Language Program Example

The following assembly language program demonstrates how system calls are used in a typical program. The program reads characters from the Apple *//e* keyboard, and writes them to a specified file until CONTROL-Z is typed. It then closes the file and returns to CP/M command level. The program is written in 8080 assembler code.

Example Notes

To demonstrate different program concepts, the example program performs some unnecessary steps and also lacks several features to make it useful. For example, it only displays the characters that you type (including carriage returns and control characters). To make the program useful, modify the loop section to check for a carriage return entered from the keyboard. If a carriage return is entered, the program would then display and write a linefeed (ASCII 0AH) immediately following the carriage return.

Another problem is the backspace character. Most often, a backspace is used to move the cursor back to a typing error, and the error is corrected. This appears to work properly on the screen, but the program is unnecessarily writing the error character followed by the backspace character to your file.

Running the Example

To use the program once it is assembled and loaded, type:

```
SAMPLE FILENM
```

and press RETURN. The FILENM may be any filename you choose. The Console Command Processor (CCP) will put the filename into the default File Control Block located at memory address 005CH.

Example Listing

```
;
;
;Sample program FILENM
;
;
BDOS        EQU     5           ;Equate BDOS to represent memory location
                                ;0005H. This is the address that the program
                                ;jumps to when it requests a function from
                                ;CP/M. Any reference to BDOS in this
                                ;program now refers to 5.
                                ;
                                ;
            ;CP/M system call numbers used in this program.
                                ;
                                ;
GETCH       EQU     1           ;System call 1 gets character from console
DISTRNG     EQU     9           ;System call 9 prints an ASCII string
CLOSFL      EQU     16          ;System call 16 closes a file
KILLFL      EQU     19          ;System call 19 deletes a file
WRITE       EQU     21          ;System call 21 writes sequential
BLDFIL      EQU     22          ;System call 22 creates a file
                                ;
FCB         EQU     005CH       ;Address of default File Control Block
DMA         EQU     0080H       ;Address of default disk buffer
                                ;
                                ;
;Begin actual code
                                ;
                                ;
            ORG     0100H       ;Tell loader to locate the program at 100H.
                                ;This is the location used for almost all
                                ;CP/M programs.
                                ;
            LXI     SP,STACK    ;Set up stack pointer for this program.
                                ;STACK is actually an address defined in
                                ;the data area that follows.
```

```
                                        ;
                                        ;
;Create file
                                        ;
                                        ;
LXI     D,FCB           ;Load the D and E registers with the FCB
                        ;address. (Since this is a 16-bit operation,
                        ;the higher-order byte of the FCB is in the D
                        ;register.)
                                        ;
MVI     C,KILLFL        ;Before creating the file we must make sure
                        ;that it doesn't already exist. Function 19
                        ;deletes an existing file of the same name.
                                        ;
PUSH    D               ;Save address of the FCB in case the call
                        ;destroys it.
                                        ;
CALL    BDOS            ;Kill file if the file is there.
                        ;Normal procedure would be to check if
                        ;function was successful, but we don't care
                        ;with this function.
                                        ;
POP     D               ;Restore D from stack (previously PUSHed).
MVI     C,BLDFIL        ;Select build file routine.
CALL    BDOS            ;Call CP/M to create file.
                                        ;
CPI     255             ;Compare contents of register A with 255 to
                        ;indicate if the build file failed from a lack of
                        ;directory space or a similar problem.
                                        ;
JNZ     BLDOK           ;Jump if not zero—if previous compare
                        ;operation yielded a zero, then a match was
                        ;found and file not built. If not zero, then file
                        ;was built.
                                        ;
                                        ;
;File build error—display message then quit
                                        ;
                                        ;
LXI     B,BLDERR        ;Load D register with error message
                        ;address.
                                        ;
MVI     C,DISTRNG       ;Select display string CP/M function (9).
CALL    BDOS            ;Call CP/M to perform function.
                                        ;
JMP     QUIT            ;Jump to the quit label that returns to
                        ;CP/M.
                                        ;
                                        ;
;Build OK—Set up for input
                                        ;
                                        ;
BLDOK:  LXI     D,DMA           ;Load D and E registers with address of
                                ;default DMA area.
                                        ;
        MVI     B,O             ;Set character counter to zero.
                                        ;
        LXI     H,DMA           ;Set up memory pointer to DMA area
                                ;(H and L).
```

```
                                                    ;
                                                    ;
                ;Loop to input characters
                                                    ;
                                                    ;
LOOP:   MVI     C,GETCH     ;Load C register with 1 (get character from
                            ;keyboard).
                                                    ;
        PUSH    B           ;Save BE register pair.
        PUSH    H           ;Save H and L register pair.
        CALL    BDOS        ;Request CP/M to get a character.
        POP     H           ;Restore HL.
        POP     B           ;Restore BE.
                                                    ;
        CPI     26          ;Compare A register against 26
                            ;(CONTROL-Z).
                                                    ;
        JZ      CLOSE       ;If equal then zero flag set and jump is
                            ;performed to close routine.
                                                    ;
        MOV     M,A         ;Move character just typed from A register
                            ;to memory address pointed to by M
                            ;(H and L regs).
                                                    ;
        INX     H           ;Increment memory pointer (HL) for next
                            ;character.
                                                    ;
        INR     B           ;Increment character count (INC and INX
                            ;perform same function but INC deals with
                            ;8 bits, INX deals with 16).
                                                    ;
        MOV     A,B         ;Move contents of B to A register.
                                                    ;
        CALL    128         ;Has there been 128-bytes written since last
                            ;write?
                                                    ;
        JNZ     LOOP        Buffer not full—get another character.
                                                    ;
                                                    ;
;Write DMA buffer to disk
                                                    ;
                                                    ;
        LXI     D,FCB       ;Load DE registers with address of FCB.
        MVI     C,WRITE     ;Select write function.
        CALL    BDOS        ;Request CP/M to write 128 bytes to disk.
        CPI     0           ;Check if successful (A=0 means yes).
        JNZ     WRTERR      ;If not zero then error occurred.
        MVI     B,0         ;Reset character counter since last write.
        LXI     H,DMA       ;Reload memory address of buffer area.
        JMP     LOOP        ;Get another character and continue.
                                                    ;
WRTERR: LXI     D,WRTERM    ;Load DE with address of write error
                            ;message.
                                                    ;
        MVI     C,DISTRNG   ;Select display string function.
        CALL    BDOS        ;Call CP/M to display string.
        JMP     QUIT        ;Jump to quit program.
```

```
                                        ;
                                        ;
        ; Write last sector then close file
                                        ;
                                        ;
CLOSE   MOV     M,A             ;A contains CONTROL-Z (end-of-file marker).
                                ;Move to disk transfer area.
        MVI     C,WRITE         ;Select CP/M write function.
        LXI     D,FCB           ;Load DE register with address of FCB.
        CALL    BDOS            ;Write DMA buffer to disk.
        CPI     0               ;Check if A register equals 0.
        JNZ     WRTERR          ;Jump if not zero to write error routine.
        LXI     D,FCB           ;DE must point to FCB.
        MVI     C,CLOSFL        ;Select CP/M close file function.
        CALL    BDOS            ;Request CP/M to perform close.
                                ;
                                ;
        ;All done—Return to CP/M (CCP)
                                ;
                                ;
QUIT    JMP     0               ;Perform warm start.
                                ;
                                ;
        ;Data used
                                ;
                                ;
        ;This section reserves some areas of memory for work space
        ;and initializes some areas with data (error messages, etc).
                                ;
                                ;
BLDERR: DB 'CANNOT BUILD FILE$'
                                ;
                                ;Put that value in memory with the address
                                ;referenced by BLDERR. The $ tells the
                                ;print string function when to quit printing
                                ;data.
                                ;
WRTERM  DB 'DISK WRITE ERROR$'
                                ;
                                ;Same as previous except for write error.
                                ;
        DS      32              ;Reserve 32 bytes for stack data.
                                ;
STACK:                          ;This doesn't actually do anything with the
                                ;stack or stack pointer until the address of
                                ;this data (STACK) is loaded into the stack
                                ;pointer (SP). Notice that the label appears
                                ;after the reserved data. This is because the
                                ;STACK decrements towards 100H.
                                ;
        END                     ;Tell assembler we are through.
                                ;
        ;End of program
```

Calling From a High-Level Language

System calls can be used from any high-level language whose interface modules can be linked with assembly language routines. (The interface module translates the high-level language's assembly language routine protocol to the CP/M protocol.) For specific information on how to implement system calls for a particular language, see the language's user manual or equivalent. For the Microsoft BASIC Interpreter, this information is contained in Appendix E, "Microsoft BASIC Assembly Language Subroutines" of the *Microsoft BASIC Interpreter Reference Manual.*

Calling 6502 Subroutines

6502 subroutines (assembly language subroutines executed by the 6502 microprocessor) can be called from a CP/M program through the 6502 BIOS call 0, CALLSUB. For instructions and more information on this call, see Chapter 4.

Returning Control to the CCP

Programs which run in the TPA, and do not use the memory reserved for the CCP, can return control to the CCP using a RET assembly language instruction. Otherwise, System Reset, system call 0, is used by programs to execute a warm start and return system control to the CCP. This call is identical in operation to executing a JMP 0000 instruction, which is the way most programs execute a warm start.

Interrupt Handling

Z80 interrupts are not supported on SoftCard II. 6502 interrupts can be used in programs by ending the interrupt processing routine with a 6502 RTI instruction.

I/O Device Calls

The five system calls listed in Table 2.1 provide basic communication with I/O devices other than the disk drive system.

Table 2.1.

Basic I/O Communication System Calls

Name	Call	Purpose
Console Input	1	Reads a character from the assigned Console device.
Console Output	2	Sends a character to the assigned Console device.
Reader Input	3	Reads a character from the assigned Reader device.
Punch Output	4	Sends a character to the assigned Punch device.
List Output	5	Sends a character to the assigned Listing device.

Because of the dual microprocessor programming environment, use of the five basic I/O communication system calls are dependent on current logical device assignment. Initially, all four logical devices are assigned to the TTY: physical device. However, each of the logical devices can be assigned to one other implemented physical device. If reassigned, this can affect the operation of the system call.

Note

Although there are 16 possible physical devices available, only the TTY: and one alternate physical device (per logical device) are implemented initially. "I/O Communication and the IOBYTE" in Chapter 1 explains the reasons and the technical details for this. The rest of this discussion addresses the effect the alternate physical device assignment has on the system calls.

The Console system calls (Console Input, system call 1, and Console Output, system call 2) transfer single characters between the Console device and the CPU. Usually, the Console device is assigned to the TTY: physical device, which is normally the Apple monitor and keyboard. If, however, an interface board is installed in slot 3, that board becomes the TTY: physical device and console I/O is routed to slot 3. It is possible to have an interface board installed in slot 3 and still have the Apple keyboard and monitor as the TTY: device. "Adding Nonstandard I/O Devices and User Software" in Chapter 6 gives information on how to change the slot assignment.

The alternate device assignment for the Reader device (PTR:) and for the Punch device (PTP:) both route information to accessory slot 2. If the PTR: device is assigned, Reader Input will return information from that slot. The same is true for Punch Output if the PTP: device is assigned.

List Output always returns information from slot 1 if the LPT: device is assigned.

Other Console Device System Calls

CP/M provides two other system calls for direct access to the console. (Direct access is defined as accessing the Console device without buffering or CP/M line editing commands.) Get Console Status, system call 11, determines if a character has been entered at the physical device assigned to the console. If a character has been entered, CP/M enters 0FFH in register A. If no character has been entered, register A contains 00.

The other call for direct access to the console is Direct Console I/O, system call 6. This permits programs to communicate directly with the Console device in special applications where normal console I/O would cause problems with the program. System call 6 differs from the other system calls by supporting both input and output. If register E contains the value 0FFH, then CP/M assumes input is being requested from the console and returns the next character in register A. If register E contains any other value, then CP/M assumes output is being requested, and sends the value, contained in register E, to the Console device.

Buffered Console System Calls

The SoftCard implementation of CP/M also supports buffered I/O. Buffered I/O is the input and output of character strings through the assigned Console device. Print String, system call 9, and Read Console, system call 10, permit programs to input or output a string of characters with one system call, instead of using a separate system call for each character.

I/O Device Assignment Calls

The IOBYTE is used by CP/M to monitor and change the current logical to physical device assignments. (For more information on IOBYTE, see the section "I/O Communication and the IOBYTE" in Chapter 1.) Two system calls are provided to manipulate the IOBYTE: Get IOBYTE, system call 7, and Set IOBYTE, system call 8. The Get IOBYTE call returns the current value of the IOBYTE in register A, and the Set IOBYTE call changes the IOBYTE value. The IOBYTE values and the corresponding device assignment are listed in "8 Set IOBYTE" in Chapter 3.

Creating Files

Files are created with Make File, system call 22. Make File creates a directory entry for the file. Once a file has been created, it can then be accessed by a program or the CCP. As the file requires additional storage space, CP/M will automatically create new directory entries for each new extent as required. This eliminates the need for subsequent Make File system calls every time the file size requires another extent.

Deleting Files

Files are deleted from the disk with the Delete File system call. Delete File erases all directory entries for the specified file on the disk, and thus reclaims the file's allocation units.

Opening and Closing Files

Before a file can be accessed for either read or write operations, CP/M must know where the file's physical location is on the disk and the number of extents. The Open Call system call provides this information by copying the disk directory information from the disk and into the FCB in memory.

Before an Open File call can be executed, the FCB must contain the filename in the filename field, and zeros in all other fields. After the Open File call is issued, the remaining fields are filled with data corresponding to the allocation block map for that particular file.

CP/M will update the FCB allocation block map in memory, as it reads or writes new data to the file. After a read or write operation, the new allocation block map is written back into the disk directory with Close File, system call 16. This is required to prevent data from being lost.

Note

Read operations do not change the FCB allocation unit map in the disk directory. It is good programming practice, however, to close all files after read operations.

Searching for a File

To find out if a file exists on disk, Search For First, system call 17, is used. System call 17 returns a zero in register A if the file named in the FCB is found on the disk and an FF value if the file is not present. To find ambiguous filenames, wild card characters can be used in the filename field of the FCB. If one or more "?" characters are encountered in the filename, the call will return a 00 value for the first filename that matches. To find other files that match, Search Next File, system call 18, must be used. Search Next File returns a 00 value for each file that matches the filename and FF if no matches are found.

File Read and Write Operations

When a file has been opened, data can be read from or written to the file. CP/M supports two types of read/write operations: *sequential access* and *random access*.

Sequential Access

Sequential read or write operations access successive records of an open file. When a file is opened, each successive read or write operation reads or writes the next record in the file. CP/M automatically updates the record number (byte 32 of the FCB) of the accessed file every time a Read or Write system call is performed. A program can set the initial extent and record to be read by setting bytes FCB 12 and 32 to the desired values. This permits sequential reading anywhere in the file without having to read all of the previous records.

The disadvantage of sequential access is that it is very time consuming and requires that the records following the written record be read and rewritten. Because of this limitation, sequential access is rarely used.

Random Access

Random access read and write operations access records that are in random locations on the disk. The SoftCard version of CP/M supports full random access records, whereas earlier versions of CP/M support only a limited version of random access.

Note

Programs using random access methods (using the sequential read/write commands) written under CP/M version 1.4 are permitted with the SoftCard version of CP/M.

The random access system calls (Read Random, Write Random, and Set Random Record) have two enhancements which make true random access possible. The first is that records do not have to be contiguous, and the second is the ability to convert record numbers from 1 to 65536 into the proper extent/record designations. This frees the program from having to convert records. To maintain compatibility with earlier versions, CP/M version 2.2 places the random access record number in the r0—r2 field of the FCB.

Note

The read/write sequential calls will only update the extent and record bytes in the FCB and not the random access record number bytes.

Miscellaneous System Calls

Several other disk I/O system calls are provided for using the CP/M file structure in certain situations. They are used to initialize or interrogate certain disk functions.

The most commonly used of these is Set DMA, system call 26. Set DMA sets the disk I/O buffer to the 128-byte block of memory beginning with the address contained in the DE registers. (The SoftCard version of CP/M uses memory locations 0080 to 0FF, but any 128-byte block of memory can be used.) Use Set DMA to change the buffer location in memory.

The remaining system calls are used mainly by CP/M to implement the various disk-related functions specified by the CP/M utilities.

Chapter 3
CP/M System Calls

This chapter numerically lists the 39 CP/M system calls supported by the SoftCard II system. A listing of the system calls is shown in the following table. Guidelines for using CP/M system calls are given in "Using CP/M System Calls" in Chapter 2.

Table 3.1.

CP/M System Calls Available

Call Number	Name	Call Number	Name
0	System Reset	20	Read Sequential
1	Console Input	21	Write Sequential
2	Console Output	22	Make File
3	Reader Input	23	Rename File
4	Punch Output	24	Return Login Vector
5	List Output	25	Return Current Disk
6	Direct Console I/O	26	Set DMA Address
7	Get IOBYTE	27	Get Addr Alloc
8	Set IOBYTE	28	Write Protect Disk
9	Print String	29	Get Read/Only Vector
10	Read Console Buffer	30	Set File Attributes
11	Get Console Status	31	Get Addr Disk Parms
12	Return Version Number	32	Set/Get User Code
		33	Read Random
13	Reset Disk System	34	Write Random
14	Select Disk	35	Compute File Size
15	Open File	36	Set Random Record
16	Close File	37	Reset Drive
17	Search for First	40	Write Random With Zero Fill
18	Search for Next		
19	Delete File		

System Call Parameters

In each of the system call descriptions, a table of parameters shows the required parameters, and into which registers they are loaded. The parameters in each table are:

Entry point	The system call number and the register it is loaded into.
Entry value	The data to be sent to the CPU for processing.
Returned value	The data returned by the CPU as a result of the system call.

For example, the following table shows the value returned in register A which contains either an ASCII character or zero, depending on how the call was executed.

Parameter	Register	Contents
Entry point	C	06H
Entry value	E	0FFH (input) or character (output)
Returned value	A	Character or 00H

In addition to the parameter table, a remarks section describes any special conditions or singularities for using the system call.

0 System Reset

Purpose

Performs a warm start.

Parameters

Parameter	Register	Contents
Entry point	C	00H
Entry value	None	None
Returned value	None	None

Remarks

System Reset instructs CP/M to perform a warm start. (This is the same as JMP instruction to location 00H.) Specifically, System Reset performs the following actions:

Reinitializes the disk drive system by selecting drive A: as the active drive

Reads the CCP module into memory from the disk in drive A:

Initializes all I/O devices that have an initialization routine

Clears the contents of the disk file buffer

Transfers control to the CCP module

1 Console Input

Purpose

Reads an ASCII character from the logical Console device.

Parameters

Parameter	Register	Contents
Entry point	C	01H
Entry value	None	None
Returned value	A	Character from the Console device

Remarks

Console Input reads the next character from the physical device assigned to the Console (CON:) device into register A. If a carriage return, linefeed, backspace, or graphic character is read, Console Input "echoes" the character back to the Console device for display. If a tab character (CONTROL-I) is read, the cursor is moved eight spaces to the next tab stop.

Console Input also checks for CONTROL-S (start/stop scroll), and CONTROL-P (start/stop printer echo). If CONTROL-P is present, all subsequent characters are echoed to the logical LST: device. Control is not returned to the calling program until the next character is entered from the Console device.

A subsequent CONTROL-P will disable echoing of characters to the printer.

2 Console Output

Purpose

Sends an ASCII character to the logical Console device.

Parameters

Parameter	Register	Contents
Entry point	C	02H
Entry value	E	A character
Returned value	None	None

Remarks

Console Output sends a character to the logical Console device from register E. If a tab character (CONTROL-I) is sent, up to eight blanks are output to move the cursor to the next tab stop. Console Output also checks for CONTROL-S (start/stop scroll), and CONTROL-P (start/stop printer echo).

3 Reader Input

Purpose

Reads a character from the current logical Read device (RDR:).

Parameters

Parameter	Register	Contents
Entry point	C	03H
Entry value	None	None
Returned value	A	A character

Remarks

Reader Input reads into register A the next character from the physical device assigned to RDR:. As in system call 1, Console Input, control is not returned to the calling program until a character has been read.

4 Punch Output

Purpose

Sends an ASCII character to the logical Punch device (PUN:).

Parameters

Parameter	Register	Contents
Entry point	C	04H
Entry value	E	ASCII character
Returned value	None	None

Remarks

Punch Output sends an ASCII character to the logical Punch device (PUN:) from register E. Control is not returned to the calling program until the character has been sent.

5 List Output

Purpose

Sends an ASCII character to the logical List device (LST:).

Parameters

Parameter	Register	Contents
Entry point	C	05H
Entry value	E	ASCII character
Returned value	None	None

Remarks

List Output sends an ASCII character to the logical List device (LST:) from register A. Control is not returned to the calling program until the character has been sent.

6 Direct Console I/O

Purpose

Initiates direct console I/O.

Parameters

Parameter	Register	Contents
Entry point	C	06H
Entry value	E	FFH (input) or character (output)
Returned value	A	Character or 00H

Remarks

Direct Console I/O initiated in register E either contains a value of FFH for console input request, or an ASCII character for output. Upon return, if the value in register E was FFH, register A will contain 00H. Otherwise, register A will contain the next input character from the console.

Note

We do not recommend using Direct Console I/O, since it bypasses all of CP/M's normal control character functions, such as CONTROL-S and CONTROL-P. Programs which perform direct I/O through the BIOS under previous releases of CP/M, however, should be changed to use direct I/O under BDOS so they can be fully supported under future releases of MP/M™ and CP/M.

7 Get IOBYTE

Purpose

Returns the current value of the IOBYTE.

Parameters

Parameter	Register	Contents
Entry point	C	07H
Entry value	None	None
Returned value	A	I/O byte value

Remarks

The IOBYTE determines the logical to physical device assignment. The IOBYTE value can be displayed at the Console device by using system call 2, Console Output.

8 Set IOBYTE

Purpose

Changes the logical to physical device assignment.

Parameters

Parameter	Register	Contents
Entry point	C	08H
Entry value	E	New I/O byte value
Returned value	None	None

Remarks

Set IOBYTE permits changing the IOBYTE value within programs running in the TPA. The IOBYTE format is shown in the following table. Table 3.2 also shows the possible values of the IOBYTE.

Table 3.2.

IOBYTE Values

Field (Bits)	Decimal Value	Description
Console	xxx0	TTY: assigned (Default)
	xxx1	CRT: assigned
	xxx2	Batch (BAT:) mode
	xxx3	UC1: assigned
Reader	xx0x	TTY: assigned (Default)
	xx1x	CRT: assigned
	xx2x	PTR: assigned
	xx3x	UR2: assigned
Punch	x0xx	TTY: assigned (Default)
	x1xx	PTP: assigned
	x2xx	UP1: assigned
	x3xx	UP2: assigned
List	0xxx	TTY: assigned
	1xxx	CRT: assigned
	2xxx	LPT: assigned (Default)
	3xxx	UL1: assigned

9 Print String

Purpose

Sends a character string to the logical Console device.

Parameters

Parameter	Register	Contents
Entry point	C	09H
Entry value	DE	String address
Returned value	None	None

Remarks

Print String sends a character string from the address contained in register pair DE to the logical Console device. Character strings must end with a "$" character. If the character string contains tab characters, they are expanded in the same manner as in system call 1, Console Input. Print String also checks for CONTROL-S (start/stop scroll) and for CONTROL-P (printer echo).

10 Read Console Buffer

Purpose

Reads the contents of the Console device buffer.

Parameters

Parameter	Register	Contents
Entry point	C	0AH
Entry value	DE	Buffer address
Returned value	Buffer	Console characters

Remarks

Read Console Buffer reads the edited input from the Console logical device into the buffer address specified in register pair DE. (The buffer address is determined by the calling program.) Input is terminated when either the buffer overflows (maximum 255 characters), or a terminating character (carriage return or linefeed) is read into the buffer. The Read Console Buffer is in the following format:

Byte	0	1	2	3	4	5	6	7	8	...	n
Field	mx	nc	c1	c2	c3	c4	c5	c6	c7	...	cn

Figure 3.1. Console Buffer

mx — Equals 255 characters (the buffer's maximum capacity).

nc — Equals the number of characters read (set by FDOS upon return).

c1—cn — Equals the characters read from the Console device.

If nc is less than mx, the uninitialized positions follow the last character (cn).

Input to the buffer can be edited with the following line-editing commands:

CONTROL-C	Performs a warm start (if entered at the beginning of line)
CONTROL-E	Denotes the end of the line
CONTROL-H	Backspaces one character position
CONTROL-J	Terminates the input line (linefeed)
CONTROL-M	Terminates the input line (carriage return)
CONTROL-R	Retypes the current line after a new line
CONTROL-X	Backspaces to the beginning of the current line

Note

Line editing commands which move the cursor to screen column 0 (e.g., CONTROL-X) will only move the cursor to the column position where the screen prompt ended. This allows for a more legible display. (In earlier CP/M versions, the cursor was returned to the column 0.)

---

11 Get Console Status

Purpose

Monitors the logical Console device for input.

Parameters

Parameter	Register	Contents
Entry point	C	0BH
Entry value	None	None
Returned value	A	Console status value

Remarks

If CON: sends a character, register A will contain FFH. Otherwise, register A contains 00H.

12 Return Version Number

Purpose

Returns the CP/M version number.

Parameters

Parameter	Register	Contents
Entry point	C	0CH
Entry value	None	None
Returned value	HL	Version number

Remarks

Return Version Number provides a means of programming that is not version dependent. When called, Return Version Number returns a two-byte value representing the version number in register pair HL. The value in register H indicates CP/M (H=00H) or MP/M (H=01H). The value in register L indicates the version of CP/M as follows:

L=00H	All releases prior to 2.0
L=20H	CP/M 2.0
L=21H	CP/M 2.1
L=22H	CP/M 2.2

Return Version Number is useful for writing application which provide both sequential and random access functions. (Random access is disabled for CP/M releases prior to 2.0.)

13 Reset Disk System

Purpose

Resets the disk system from within a program.

Parameters

Parameter	Register	Contents
Entry point	C	0DH
Entry value	None	None
Returned value	None	None

Remarks

Reset Disk System resets the disk drive system from within a calling program. This is useful for application programs that require a disk change without a warm or cold start.

When called, Reset Disk System assigns all drives with read or write only attributes and makes disk drive A: the active drive. It also sets the default DMA address to BOOT+0080H. (See system calls 28, Write Protect Disk, and 29, Get Read/Only Vector, for more information on read and write only attributes.)

14 Select Disk

Purpose

Changes the active drive.

Parameters

Parameter	Register	Contents
Entry point	C	0EH
Entry value	E	Selected disk
Returned value	None	None

Remarks

Select Disk changes the current active disk drive to the drive represented by the value in register E. The value of register E is as follows:

E=00H Drive A:
E=01H Drive B:
E=02H Drive C:
E=03H Drive D:

When selected, the active drive is placed "on-line," which activates its directory in memory until the next cold start, warm start, or disk system reset operation is performed. If the disk is changed while it is on-line, the drive's status is changed to read/only status (see system call 29, Get Read/Only Vector).

During file operations, an FCB which contains 00 for the drive code will automatically access the active drive. Drive codes one through three ignore the active drive and access the selected drive (A: through D:).

15 Open File

Purpose

Opens an existing file.

Parameters

Parameter	Register	Contents
Entry point	C	0FH
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

Open File searches the disk directory in the current user area for a filename that matches the name in the FCB contained in register pair DE. Wild card characters (? and *) can be used in the fn and type fields of the FCB. If no wild card characters are included, bytes ex and s2 of the FCB are set to zero. See "The File Control Block" in Chapter 1 for a description of the FCB fields.

If the FCBs match, the relevant file directory information from the disk is copied into the d0—dn field of the addressed FCB. This allows access to the file through subsequent read and write system calls.

Note

Existing files should not be accessed until a successful open operation is completed.

When a file has been opened, Open File returns the directory code with the value zero through three in register A. Otherwise, Open File returns 0FF in register A. If there are wild card characters in the addressed FCB, then the first matching directory FCB is selected. If the file is to be accessed sequentially from the first record, the calling program must set the current record (cr) field to zero.

16 Close File

Purpose

Closes an existing open file.

Parameters

Parameter	Register	Contents
Entry point	C	10H
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

Close File closes an existing open disk file in the current user area. If the file was opened using an Open File or Make File system call, Close File records the file's new FCB information in the referenced disk directory. The FCB matching process for the Close File system call is identical to the Open File system call. When a file is closed, the directory code in register A is 0H, 1H, 2H, or 3H. Otherwise, 0FFH is returned if the filename cannot be found in the directory.

If a file has been written to, it must be closed in order to update the FCB of the file. A file need not be closed for read operations.

17 Search for First

Purpose

Searches for the first file match.

Parameters

Parameter	Register	Contents
Entry point	C	11H
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

Search for First searches the disk directory of the current active user area for the first filename matched by the addressed FCB. If found, Search for First returns a value between 0H and 3H in register A. Otherwise, FFH is returned if the file is not found.

When the addressed file is found, Search for First writes the matching directory entry and the relative starting position into the current DMA address. This is not normally required for application programs, but it permits the directory information to be obtained from the DMA buffer by the calling program.

The ? wild card character can be used in FCB fields f1—f8, t1—tn, and ex to match the corresponding field of a directory entry on the active drive. If the dr field contains a question mark (?), however, the active disk select function is disabled and the default drive is searched. Search for First will then return any matched entry, allocated or free, belonging to any user number. This is not normally used by application programs, but permits greater flexibility to search all current directory entries. If the dr field is not a question mark, the s2 byte is automatically set to zero.

18 Search for Next

Purpose

Searches for the next file match.

Parameters

Parameter	Register	Contents
Entry point	C	12H
Entry value	None	None
Returned value	A	Directory code

Remarks

Search for Next is similar to the Search for First system call, except that the directory search continues from the last matched entry. If a match is found, Search for Next returns a value between 0H and 3H in register A. 0FFH is returned when no more directory items match.

19 Delete File

Purpose

Deletes a file or files.

Parameters

Parameter	Register	Contents
Entry point	C	13H
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

The FCB in register pair DE may contain wild card characters (? or *) in the f1—f8 and t1—t3 fields, but not in the dr field (as in the Search for First and Search for Next system calls).

If the file(s) exist and can be deleted, Delete File returns a value between 0H and 3H in register A. If the file(s) cannot be found, a value of 0FFH is returned.

20 Read Sequential

Purpose

Reads a record sequentially.

Parameters

Parameter	Register	Contents
Entry point	C	14H
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

Read Sequential reads the next 128-byte record from the addressed file into memory at the current DMA address. Before Read Sequential can be used, the FCB in register pair DE must be activated through system call 15, Open File, or system call 22, Make File. If the FCB is present, Read Sequential reads the next 128-byte record from the file into memory at the current DMA address. The record's location is read from the FCB cr (current record) field of the extent, and the value of cr is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next read operation.

Read Sequential returns a value of 00H in register A when the operation has been completed, and a non-zero value if no data exists at the next record position until the end of the file is reached.

21 Write Sequential

Purpose

Writes data to a file sequentially.

Parameters

Parameter	Register	Contents
Entry point	C	15H
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

Write Sequential writes the next 128-byte record to the addressed file at the current DMA address. Write Sequential can be used only if the FCB address in register pair DE has been activated through system call 15, Open File, or system call 22, Make File. If the FCB is present, Write Sequential writes the next 128-byte data record to the open file from the current DMA address. The cr field of the addressed FCB is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero to prepare for the next write operation. Records written into an existing file overlay those which already exist in the file.

When the write operation has been completed, Write Sequential returns a value of 00H in register A, or a non-zero value for an unsuccessful write operation due to a full disk.

22 Make File

Purpose

Creates or "makes" a new file.

Parameters

Parameter	Register	Contents
Entry point	C	16H
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

Make File is similar to the Open File system call, except that the FCB must not contain a filename of an existing file in the active disk directory. When executed, Make File also creates the file and initializes both the FCB disk directory and the FCB in memory.

Make File returns a value between 0H and 3H in register A if the file was created, and 0FFH if no more directory space was available to create the file. Make File also activates the FCB, so a subsequent open operation is not necessary for writing to the file.

23 Rename File

Purpose

Renames an existing file.

Parameters

Parameter	Register	Contents
Entry point	C	17H
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

Rename File changes the filename and extension in the first 16 bytes of the addressed FCB to the filename and extension in the second 16 bytes. The FCB drive code (dr) selects the drive, while the drive code for the new filename in d0 (byte 16) is assumed to be 0H.

Rename File returns a value between 0H and 3H in register A when the file is renamed. If the file cannot be renamed, Rename File returns a value of 0FFH.

24 Return Login Vector

Purpose

Writes the CP/M login vector into register pair HL.

Parameters

Parameter	Register	Contents
Entry point	C	18H
Entry value	None	None
Returned value	HL	Login vector

Remarks

CP/M returns a 16-bit login vector value in register pair HL. The least significant bit position in register L denotes the first drive (A:), and the most significant bit position in register H denotes the fourth drive (drive D:). "0" bit indicates that the drive is off-line, while a "1" bit indicates the drive is on-line. Drives can be brought on-line by an explicit disk drive selection, or by an implicit drive selection caused by a file operation which specified a non-zero dr field.

Note

To maintain compatibility with earlier CP/M releases, registers A and L contain the same values upon return of the call.

25 Return Current Disk

Purpose

Indicates the current active drive.

Parameters

Parameter	Register	Contents
Entry point	C	19H
Entry value	None	None
Returned value	A	Current disk

Remarks

Return Current Disk returns a value in register A that corresponds to the current active drive. The possible values in register A are as follows:

0H	Drive A:
1H	Drive B:
2H	Drive C:
3H	Drive D:

26 Set DMA Address

Purpose

Changes the default DMA address.

Parameters

Parameter	Register	Contents
Entry point	C	1AH
Entry value	DE	DMA address
Returned value	None	None

Remarks

Set DMA Address changes the default DMA address. DMA (Direct Memory Address) is a method of transferring data directly between memory and the disk subsystem. In CP/M, the DMA address is the address of the 128-byte data record before a disk write operation, or after a disk read operation occurs.

When a cold start, warm start, or disk system reset operation is performed, the DMA address automatically resets to 0080H. The DMA address can be changed with the Set DMA Address call to access another area of memory where data records reside. The DMA address specified in register pair DE remains unchanged until the next Set DMA Address call, cold start, warm start, or disk system reset operation is performed.

27 Get Addr Alloc

Purpose

Returns the allocation vector base address of the active drive.

Parameters

Parameter	Register	Contents
Entry point	C	1BH
Entry value	None	None
Returned value	HL	Allocation vector address

Remarks

CP/M maintains an allocation vector in memory for each on-line disk drive. Programs such as STAT and PIP use the information provided by the allocation vector to determine the amount of remaining storage.

Note

Allocation vector information can be invalid if the selected drive has a read only attribute. Get Addr Alloc is not normally used by application programs.

---

28 Write Protect Disk

Purpose

Sets write-protect status on the active drive.

Parameters

Parameter	Register	Contents
Entry point	C	1CH
Entry value	None	None
Returned value	None	None

Remarks

Write Protect Disk sets a temporary write-protect attribute on the active drive which disables write operations. The attribute is removed by the next cold or warm start.

29 Get Read/Only Vector

Purpose

Determines which drives have the temporary read/only bit set.

Parameters

Parameter	Register	Contents
Entry point	C	1DH
Entry value	None	None
Returned value	HL	R/O vector value

Remarks

Get Read/Only Vector determines which drives have the temporary read/only bit set through a 16-bit vector in register HL. The least significant bit position in register L denotes the first drive (A:), and the most significant bit position in register H denotes the sixteenth drive (P:). A "0" bit indicates that the drive is R/W, while a "1" bit indicates the drive is R/O. The R/O bit is set either by system call 28, Write Protect Disk, or automatically by CP/M when it detects a changed disk.

30 Set File Attributes

Purpose

Sets file attributes from a program.

Parameters

Parameter	Register	Contents
Entry point	C	1EH
Entry value	DE	FCB address
Returned value	A	Directory code

Remarks

Set File Attributes allows a program to change attributes of the file specified by the addressed FCB. Specifically, this system call either sets or resets the read only and system attributes in the FCB t1—t2 field. When called, Set File Attributes searches for a matching FCB, and changes the matched directory entry to contain the selected attributes.

Note

Although the FCB indicators f1' through f4' are not currently used, they can be useful for application programs. (f1 and f4 are not involved in the matching process during file open and close operations.) Indicators f5' through f8' and t3' are reserved for future system expansion.

31 Get Addr Disk Parms

Purpose

Reads the address of the disk parameters into register A.

Parameters

Parameter	Register	Contents
Entry point	C	1FH
Entry value	None	None
Returned value	A	DPB address

Remarks

Get Addr Disk Parms returns the address of the BIOS disk parameter block in register pair HL. This address can be used for the following purposes:

1. To get the disk parameter values for display

2. To compute the amount of free disk space

3. To change the current disk parameter values

Normally, application programs will not require the use of this system call.

32 Set/Get User Code

Purpose

Reads or changes the current user code.

Parameters

Parameter	Register	Contents
Entry point	C	20H
Entry value	E	0FFH (get) or user code (set)
Returned value	A	Current code or 0FFH (no value)

Remarks

Set/Get User Code allows an application program to read or change the current user number. To read the current user number, register E must contain the value 0FFH. Set/Get User Code will return the value of the current user number (0 to 15) in register A. If the value in register E is not 0FFH, then the current number is changed to the value of E (modulo 32).

33 Read Random

Purpose

Reads a record using random (direct) access.

Parameters

Parameter	Register	Contents
Entry point	C	21H
Entry value	DE	FCB address
Returned value	A	Return code

Remarks

Read Random is similar to system call 20, Read Sequential, except that the read operation takes place at the record number selected by the r0—r2 field of the FCB. Read operations only use bytes r0 and r1. (Byte r2 is used only in computing the size of a file. See system call 35, Compute File Size, for more information on computing the size of a file.)

The r0—r1 byte pair contains the value corresponding to the record to be read. The value range of r0—r1 (0H to 65535H) can access any particular record of an eight-megabyte file. Byte r2 must be set to zero, since a non-zero value indicates overflow past the end of the file.

Before a file can be read with a Random Read call, it must be opened with either an Open File or Make File system call. This ensures that the information in the file's FCB is read into the FCB contained in the DE register pair. When the file is opened, the selected record number is read into the FCB record field (r0—r1), and then Read Random can read the record. When the call is completed, register A contains either the value 00H to indicate a successful read operation, or an error code.

When the read operation has been completed, the current DMA buffer will contain the data of the selected record.

The FCB record number is not incremented by the system call. This differs from a Read Sequential system call where the record number is incremented. By not incrementing the record number, subsequent Read Random system calls continue to read the same record.

After each Read Random call, the logical extent and current record values are automatically set to the appropriate values to allow the file to be sequentially read or written, starting from the current randomly accessed position. The random record position can be advanced optionally by the program following each random read or write operation to obtain the effect of a sequential I/O operation.

Note

The first Read Sequential call after a Read Random call rereads the record in the DMA buffer.

The following error codes are returned in register A, if the read operation was unsuccessful:

01H	Reading unwritten data
03H	Cannot close current extent
04H	Seek to unwritten extent
06H	Seek past physical end-of-disk

Error codes 01H and 04H occur when a random read operation accesses a data block which has not been previously written, or an extent which has not been created. These are equivalent conditions. Error code 03H does not normally occur under proper system operation, but can be cleared by simply rereading, or reopening extent zero as long as the disk is not physically write-protected. Error code 06H occurs whenever byte r2 contains a non-zero value under the current CP/M version 2.0 release. Normally, non-zero return codes can be treated as missing data, with zero return codes indicating that the operation is complete.

34 Write Random

Purpose

Writes a record using random (direct) access.

Parameters

Parameter	Register	Contents
Entry point	C	22H
Entry value	DE	FCB
Returned value	A	Return code

Remarks

The Write Random system call is similar to the Read Random system call, except that data is written to the specified file on disk from the current DMA buffer.

If the addressed file's extent has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random system call, the random record number is not changed as a result of the write operation. The extent number and current record fields of the addressed FCB are set to correspond to the random record which is being written.

After a random write operation has been performed, sequential read or write operations can commence with the notation that the currently addressed record is to be either read or written again as the sequential operation begins. The random record field can also be advanced by the programmer following each write operation to achieve the effect of a sequential write operation.

Note

Reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

The error codes returned by a random write are identical to the random read operation, with the addition of error code 05H, which indicates that a new extent cannot be created due to directory overflow.

35 Compute File Size

Purpose

Determines the size of the file.

Parameters

Parameter	Register	Contents
Entry point	C	23H
Entry value	DE	FCB address
Returned value	FCB	Random record field set

Remarks

Compute File Size determines the size of the file specified in the DE register pair. The FCB in register pair DE cannot contain wild card characters and the r0—r2 field is used for random access.

When the call is completed, the r0—r2 field of the FCB contains the record address of the virtual file size. If the value r2 is 01H, the file contains 65536 records, which is the maximum size of a file. If r2 is 00H, r0 and r1 contain the file size, which is a 16-byte value with r0 as the least significant byte.

Compute File Size can be used to append data to the end of a file by setting the random record position to the end of the file and then performing a sequence of random write operations, starting at the preset record address.

Note

If the file is written to by sequential write operations, the virtual size of a file is the same as the physical size. If the file was written to in random mode, "holes" exist in the allocation map and the file may contain fewer records than the size indicates. For example, if only the last record of an eight-megabyte file is written in random mode (i.e., record number 65535), then the virtual size is 65536 records, although only one block of data is actually allocated.

---

36 Set Random Record

Purpose

Sets the random record position from a sequentially accessed file to a specific value.

Parameters

Parameter	Register	Contents
Entry point	C	24H
Entry value	DE	FCB address
Returned value	None	Random record field set

Remarks

Set Random Record sets the random record field of the specified file to a new value. This system call can be used in two ways.

First, it can eliminate the task of searching a sequentially accessed file to get the contents of various "key" fields. As each field is encountered, Set Random Record is called to compute the random record position for the data corresponding to this key. If the size of the data block is 128 bytes, the record position is placed into a table with the key for later retrieval.

After searching the entire file and tabulating the key fields and their record numbers, you can move instantly to a particular keyed record by performing a random read operation and by using the corresponding random record number which was saved earlier. This method can be used when variable record lengths are involved, since the program need only store the buffer-relative byte position along with the key field and record number to find the exact starting position of the keyed data.

The second use of Set Random Record is for switching from sequential access operations to random access operations. If a file is sequentially accessed to a particular point in the file, Set Random Record is called to set the record number. Subsequent random read and write operations continue from the selected point in the file.

37 Reset Drive

Purpose

Resets specified disk drives to their initial values.

Parameters

Parameter	Register	Contents
Entry point	C	25H
Entry value	DE	Drive vector
Returned value	A	00H

Remarks

Reset Drive allows a calling program to reset a specified drive. The drive vector parameter is a 16-bit vector of the drive to be reset where the least significant bit represents drive A:.

40 Write Random With Zero Fill

Purpose

Writes a zero record using random (direct) access.

Parameters

Parameter	Register	Contents
Entry point	C	28H
Entry value	DE	FCB address
Returned value	A	Return code

Remarks

Write Random With Zero Fill is similar to system call 34, Write Random, but writes zeros into a previously unallocated block before data is written.

Chapter 4
6502 BIOS

This chapter describes the 17 functions requests, the 6502 BIOS calls, that access the 6502 microprocessor. A listing of the system calls is provided in the following table.

Table 4.1.

6502 BIOS Calls

Call Number	Name	Call Number	Name
0	CALLSUB	9	WSTART
1	READMEM	10	FORMAT
2	WRITEMEM	11	UPDATE
3	READSEC	12	BEEP
4	WRITESEC	13	CLEAR
5	READSLOT	14	INVERT
6	WRITESLOT	15	SETPT1
7	STATSLOT	16	SETPT2
8	INITSLOT		

The guidelines for using the 6502 BIOS calls are the same as the guidelines for using CP/M system calls, except that parameters are transferred in and out of a seven-byte block of memory (SoftCard addresses 0045H—004BH) instead of the CPU registers. See "6502 BIOS Calls" in Chapter 2 for details.

Installing User-Written Software in the 6502 BIOS

You can install your own device drivers or other user-written software as part of the 6502 BIOS with the SoftCard II. This section describes the facilities and programming conventions you will need to perform this task. Strict adherence to the programming conventions will ensure compatibility between user-written programs in the 6502 BIOS.

Important

Before attempting to install software in the 6502 BIOS, you should have experience in assembly language programming and be familiar with both Z80 and 6502 instruction sets.

6502 BIOS Operation

When a SoftCard II is installed in the Apple][,][Plus or *//e* computer and CP/M is loaded into memory, both the Z80 and the 6502 microprocessors are run simultaneously. The Z80, however, has executive control over the system. Because the Z80 cannot address the 6502 RAM directly and the Apple uses memory-mapped I/O, the Z80 must use the 6502 to perform all I/O operations. The 6502 software that performs the I/O processing is called the 6502 BIOS.

The SoftCard II BIOS consists of two parts: the Z80 or CP/M BIOS, which interfaces to the CP/M operating system, and the 6502 BIOS which controls the I/O devices and implements the print spooler.

Usually the 6502 BIOS, represented by Figure 4.1, is in a loop waiting for a command from the Z80. When the 6502 receives a command, it sends a command back to halt the Z80. The 6502 BIOS then performs a setup routine and jumps (CMDJMP) to the main command handling routine (DOCMD). When the main command handling routine is finished, it executes a clean-up routine (CMDONE). The CMDONE cleanup routine passes parameters to the Z80 memory and starts the Z80.

```
CMDLP:                  ;Main 6502 BIOS loop (waits
                        ;for a command from the Z80).
CMDINI                  ;Get a command, perform setup
                        ;processing.
CMDJMP:  JMP DOCMD      ;Jump to the main command
                        ;handling routine.
DOCMD:                  ;Execute the command.
CMDONE:                 ;Perform cleanup routine
                        ;and turn on the Z80.
         JMP CMDLP      ;Jump to CMDLP and wait for
                        ;the next Z80 command.
```

Figure 4.1. A Simplified Representation of 6502 BIOS

Changing the 6502 BIOS

The 6502 destination address of CMDJMP and the beginning address of CMDONE are stored in Z80 memory locations CMDVEC and CMDEXT. This allows you to implement your own software in the 6502 BIOS. If you replace the destination address of CMDJMP with the 6502 address of your own program or driver, control is passed to your program instead of the main command routine when a 6502 BIOS call is made. At this point, your program must decide whether to process the call or to let the main 6502 command routine process the call. Use the following code segment to define CMDEXT and CMDVEC in Z80 memory:

```
CMDEXT:  DW CMDONE     ;CMDEXT is at 0F38EH
CMDVEC:  DW CXMJMP+1   ;CMDVEC is at 0F390H
```

If your program processes the call, the program should jump to CMDONE when it is finished. If it lets the 6502 BIOS perform its usual processing for this call, the program will then jump to DOCMD.

6502 Memory Map

Two blocks in Figure 4.2 show the areas in the 6502 memory that are reserved for user-written programs. The rest of the 6502 memory space is reserved for the 6502 BIOS, the print spooler, text and graphics screens, and various other Apple hardware interfaces.

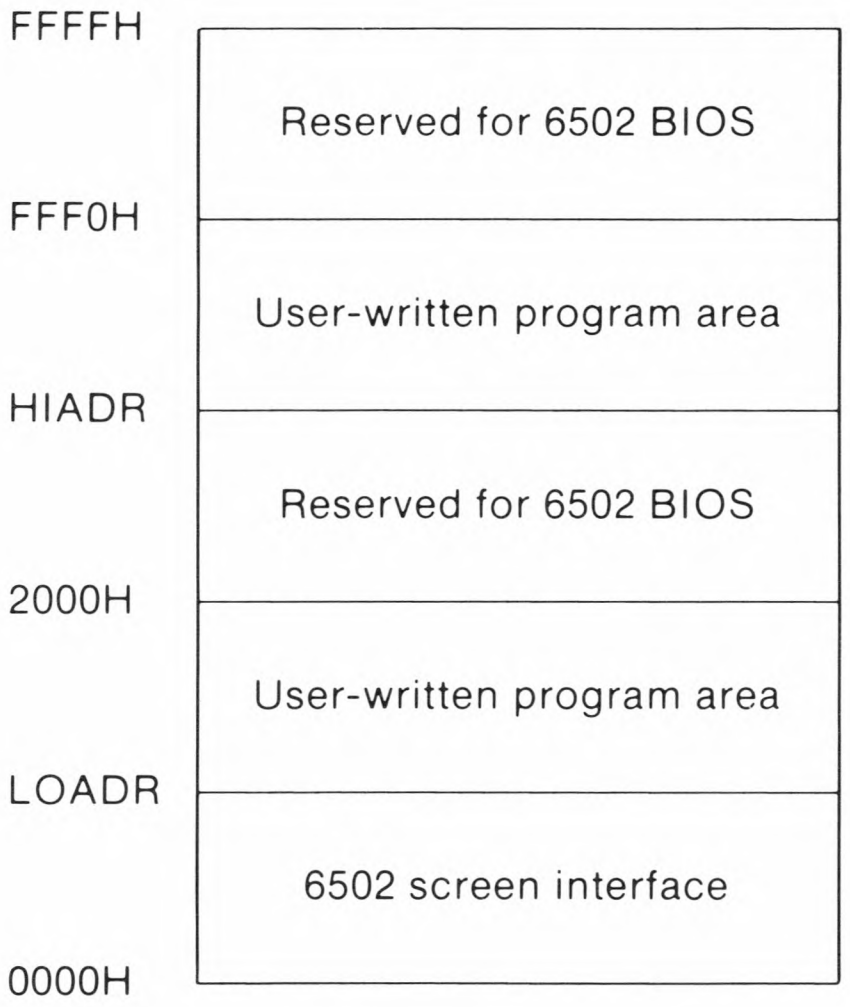


Figure 4.2. 6502 BIOS Memory Map

The highest address available in the first user area is 1FFFH. In the second user area, the highest address is FFF0H. The low addresses for each area are dependent on which routines have already been implemented in this area. Location LOMEM in Z80 memory contains the lowest address currently available in the first area (LOADR), while HIMEM contains the lowest addresses available in the second area (HIADR). The Z80 addresses of both locations are listed in Table 4.2.

Table 4.2.

6502 BIOS Vector Table

Name	Z80 Address	Function
HIMEM	F394H	Contains the lowest address of the high 6502 free memory area.
LOMEM	F392H	Contains the beginning address of the low 6502 free memory area.
CMDVEC	F390H	Contains the destination address of the JMP instruction to the main 6502 BIOS call handling routine (CMDJMP).
CMDEXT	F38EH	Contains the destination address of the JMP instruction to 6502 BIOS cleanup subroutine (CMDONE).

Implementing Your Own Software

To install your own program or driver in 6502 memory, use LOMEM or HIMEM to determine the amount of available memory for your routine. Use 6502 BIOS call 1, READMEM, to write your program into the designated 6502 user area, starting at the location contained in either LOMEM or HIMEM. The last step is to change the value of LOMEM or HIMEM to point to the byte following the last byte of your program or driver. If you don't update LOMEM or HIMEM, the next user routine or program that is installed could overwrite your program. Use the following code segments to define CMDEXT and CMDVEC in Z80 memory:

```
LOMEM:  DW LOADR     ;LOMEM is at F392H
HIMEM:  DW HIADR     ;HIMEM is at F394H
```

6502 BIOS Call Descriptions

In each of the 6502 BIOS call descriptions, a table is provided to show which parameters are needed for each call and the addresses they are stored in. Each system call includes a table of parameters showing the initial values and the returned values (if any). For example, in the following table:

Parameter	Address	Contents
Entry point	49H	1
Entry value	4AH	Low part of 6502 address
	4BH	High part of 6502 address
Returned value	45H	Data byte read from 6502 address

SoftCard memory address 49H contains the BIOS call number. The entry value shows the type of information needed to make the call and the memory locations that are relevant to the call. (All 6 locations can be used for entry values if needed.) The returned value parameter shows the SoftCard address and data returned after each system call is made.

0 CALLSUB

Purpose

Calls a 6502 subroutine.

Parameters

Parameter	Address	Contents
Entry point	49H	0
Entry value	45H	6502 register A
	46H	6502 register X
	47H	6502 register Y
	4AH	Low part of 6502 subroutine address
	4BH	High part of 6502 subroutine address
Returned value	45H	6502 register A
	46H	6502 register X
	47H	6502 register Y
	48H	6502 status register

Remarks

A 6502 subroutine is executed with a 6502 JSR instruction. Before the JSR instruction is executed, the 6502 registers are loaded from the Z80 register pass area. At the same time, the Apple monitor ROM is banked in. When the subroutine has run, the contents of the 6502 registers are stored in the same register pass area as before (addresses 45H—4BH) and control is returned to the Z80.

1 READMEM

Purpose

Reads a byte from 6502 memory.

Parameters

Parameter	Address	Contents
Entry point	49H	1
Entry value	4AH	Low part of 6502 address
	4BH	High part of 6502 address
Returned value	45H	Data byte read from 6502 address

Remarks

READMEM reads the contents of the 6502 address contained in SoftCard memory locations 4AH and 4BH. The value is returned in memory location 45H.

2 WRITEMEM

Purpose

Writes a byte to a 6502 memory location.

Parameters

Parameter	Address	Contents
Entry point	49H	2
Entry value	45H	Data byte to be written
	4AH	Low part of 6502 address
	4BH	High part of 6502 address
Returned value	—	None

Remarks

A byte stored at SoftCard address 45H is written to the indicated 6502 address.

3 READSEC

Purpose

Reads from a disk sector.

Parameters

Parameter	Address	Contents
Entry point	49H	3H
Entry value	45H	Track number (0—34)
	46H	Disk drive number (1 or 2)
	47H	Slot number (4—6)
	48H	Sector number (0—15)
	4AH	Low part of 6502 disk sector address
	4BH	High part of 6502 disk sector address
Returned value	45H	Error return code 0 = no error 16 = write-protect error Other = I/O error

Remarks

READSEC performs a low-level disk sector read operation. The address of the 256-byte sector is stored at SoftCard memory locations 4AH and 4BH.

4 WRITESEC

Purpose

Writes to a disk sector.

Parameters

Parameter	Address	Contents
Entry point	49H	4H
Entry value	45H	Track number (0—34)
	46H	Disk drive number (0—4)
	47H	Slot number (1—7)
	48H	Sector number (0—15)
	4AH	Low part of 6502 disk sector address
	4BH	High part of 6502 disk sector address
Returned value	45H	Error return code 0 = no error 16 = write-protect error Other = I/O error

Remarks

A low-level sector write is performed. The 256-byte sector is read from memory locations 4A and 4B.

5 READSLOT

Purpose

Reads a character from an accessory slot.

Parameters

Parameter	Address	Contents
Entry point	49H	5H
Entry value	47H	Slot number (1—7)
Returned value	45H	Character read

Remarks

A character is read from an accessory board installed in the indicated Apple accessory slot. The board must be of type 3, 4, or 6. See Table 6.6, "Accessory Slot Addresses and Assignments," in Chapter 6, for a description of accessory board types.

6 WRITESLOT

Purpose

Writes a character to an accessory slot.

Parameters

Parameter	Address	Contents
Entry point	49H	6H
Entry value	45H	Character to be written
	47H	Slot number (1—7)
Returned value	—	None

Remarks

A character is written to the accessory board in the indicated slot. The board type must be 3, 4, 5, or 6. If the board type is either 3, 5, or 6, and the slot number is 1, then the data is buffered in the 32K-byte print buffer. See Table 6.6, "Accessory Slot Addresses and Assignments," in Chapter 6, for a description of accessory board types.

7 STATSLOT

Purpose

Gets the input status of an accessory slot.

Parameters

Parameter	Address	Contents
Entry point	49H	7H
Entry value	47H	Slot number
Returned value	45H	Slot status FFH = character ready 00H = character not ready

Remarks

If a character is ready to be read from the specified accessory slot, the value of memory location 45H will be FFH. If no character is ready, the value will be 00H.

8 INITSLOT

Purpose

Initializes a slot.

Parameters

Parameter	Address	Contents
Entry point	49H	8H
Entry value	47H	Slot number
Returned value	45H	Slot status FFH = character ready 00H = character not ready

Remarks

Initializes the accessory board in the indicated slot if the board is of type 3, 4, or 6. Other board types are unaffected. See Table 6.6, "Accessory Slot Addresses and Assignments," in Chapter 6, for a description of accessory board types.

9 WSTART

Purpose

Performs a CP/M warm start.

Parameters

Parameter	Address	Contents
Entry point	49H	9H
Entry value	—	None
Returned value	—	None

Remarks

Performs a warm start by reloading the CCP module and first 256 bytes of the BDOS module into the appropriate addresses of the SoftCard memory.

10 FORMAT

Purpose

Formats a disk.

Parameters

Parameter	Address	Contents
Entry point	49H	AH
Entry value	46H	Drive number (1 or 2)
	47H	Slot number (5 or 6)
Returned value	45H	Error return code 0 = no error 1—15 = I/O error 16 = write-protect error

Remarks

Formats the disk in the indicated drive for CP/M.

11 UPDATE

Purpose

Updates keyboard definition and screen function interface tables.

Parameters

Parameter	Address	Contents
Entry point	49H	BH
Entry value	—	None
Returned value	—	None

Remarks

After changes are made to the Keyboard Definition Table, or the screen function interface tables (Software Screen Function Table or Hardware Screen Function Table) CP/M uses UPDATE to make the new information active.

12 BEEP

Purpose

Creates a tone of specified pitch and duration.

Parameters

Parameter	Address	Contents
Entry point	49H	CH
Entry value	45H	Tone duration
	36H	Tone period
Returned value	—	None

Remarks

Performs the same function as the BEEP statement in GBASIC. BEEP is intended for sound effect purposes.

13 CLEAR

Purpose

Clears the screen.

Parameters

Parameter	Address	Contents
Entry point	49H	DH
Entry value	4AH	Byte written to even screen addresses
	4BH	Byte written to odd screen addresses
Returned value	—	None

Remarks

Performs the same function as the GBASIC GR 1 command. The byte at 4AH is written to all even locations on the high-resolution graphics screen. The byte at 4BH is written to all odd locations.

For more information on the GBASIC GR1 command, see the *Microsoft BASIC Interpreter Reference Manual*.

14 INVERT

Purpose

Inverts the screen in GBASIC high-resolution screen mode.

Parameters

Parameter	Address	Contents
Entry point	49H	EH
Entry value	—	None
Returned value	—	None

Remarks

All bytes on the high-resolution graphics screen are inverted.

15 SETPT1

Purpose

Sets High-Resolution Graphics Point 1.

Parameters

Parameter	Address	Contents
Entry point	49H	FH
Entry value	46H	Exclusive OR mask
	47H	AND mask
	4AH	Low part of 6502 addresses
	4BH	High part of 6502 addresses
Returned value	—	None

Remarks

The indicated screen byte is first XORed with the data at 46, the result is then ANDed with the data at 47, and finally this result is XORed onto the screen.

16 SETPT2

Purpose

Sets High-Resolution Graphics Point 2.

Parameters

Parameter	Address	Contents
Entry point	49H	10H
Entry value	45H	Byte to exclusive OR with screen byte
	4AH	Low part of 6502 addresses
	4BH	High part of 6502 addresses
Returned value	—	None

Remarks

The data at 45 is XORed onto the screen at the 6502 memory location stored at addresses 4A and 4B.

Chapter 5
Command Directory

This chapter is a directory for the CP/M commands and utility programs contained in the SoftCard II system.

Command and Utility Program Guidelines

Commands and utility programs are listed in alphabetical order. In each command and program description, the possible command line formats are shown followed by an explanation of the format. The syntax elements of the format are explained in a "Remarks" section. Where applicable, the different commands that can be used with the utility program are also listed.

This chapter assumes that you know how to use the command or program. If you are unsure of how to use a command or program, see Chapter 6, "CP/M Commands and Utility Programs," in the *Microsoft SoftCard II Installation and Operation Manual*.

APDOS

APDOS [*d*:]*cp/mfilename.ext*=[*s*:]*dosfilename*

Purpose

Copies Apple text and data files from Apple DOS disks to CP/M system disks.

Remarks

d: is the destination disk drive and *s*: is the source disk drive. *cp/mfilename.ext* is the CP/M destination file and *dosfilename* is the Apple DOS source file.

Different procedures are used for copying BASIC files and text files. See "APDOS" in Chapter 6 of the *Microsoft SoftCard II Installation and Operation Manual* for instructions and examples.

ASM

ASM *filename*[*.shp*]

Purpose

Converts a source program written in 8080A assembly language into a HEX file.

Remarks

filename is the name of the source file with an extension of .ASM. The filename extension should not be included in the command line; ASM assumes the file will have an extension of .ASM.

s specifies the disk drive (A: through D:) other than the active drive that contains the source disk.

h specifies the drive that will receive the HEX file. If a HEX file is not needed, *Z* is entered in place of the drive letter.

p specifies which drive should receive the PRN file. A PRN file is the listing of the file with error messages. Enter *Z* to disable the generation of the PRN file. Enter *X* to display the listing on the screen.

If no parameters are specified, ASM assumes that the source file is in the active drive and will create HEX and PRN files as output.

ASM is invoked by typing the ASM command line at CP/M command level. ASM can be stopped or aborted at any time by typing CONTROL-C.

ASM generates two types of error messages. Terminal errors indicate what conditions prevented ASM from assembling the program. Source program errors indicate errors in the source program but don't prevent ASM from assembling the program. All error messages are listed in Appendix A, "CP/M Error Messages."

The following table lists the directives ASM recognizes in addition to the 8080 instruction set.

Table 5.1.
ASM Assembler Directives

Directive	Description
ORG	Define starting address of the program or data section.
END	End program assembly.
EQU	Define a numeric constant.
SET	Set a numeric value.
IF	Begin conditional assembly.
ENDIF	End of conditional assembly.
DB	Define data byte.
DW	Define data word.
DS	Define data storage area.

Examples

ASM FONT

Assembles the source file FONT.ASM from the active drive. Both the HEX file (FONT.HEX) and the PRN file (FONT.PRN) are saved on the same drive.

ASM MACRO.ABX

Assembles the source file MACRO.ASM from drive A:. ASM saves the HEX file on drive B: and displays the listing at the terminal.

AUTORUN

AUTORUN [*command line*]

Purpose

Permits you to create startup disks.

Remarks

command line is any executable CP/M program name or CP/M built-in command.

A startup disk must be loaded in the active drive to be executed. The active drive is usually A:. When you start the system, the command line will be executed immediately after the CP/M operating system modules are loaded into memory.

To change the command line on a disk, type *AUTORUN* again with a new command line. Typing *AUTORUN* without a command line deletes the AUTORUN command line from the disk.

Example

AUTORUN CAT

Displays the directory on the default drive when the CP/M is loaded into memory from a cold start.

BOOT

BOOT [{*number*|*M*}]

Purpose

Reboots your Apple computer from any system disk at CP/M command level.

Remarks

number is the slot number (4, 5, or 6) of the disk controller board connected to the disk from which you are loading. If you load the operating system from drive A: or B:, the number can be omitted. (The disk controller board for drives A: and B: is installed in slot 6.)

M allows you to boot from the Apple Monitor in ROM. (The Apple Monitor is the Applesoft™ or Integer BASIC Interpreter in ROM.)

BOOT performs the same function as a CP/M cold start. It can boot Apple DOS, Apple Pascal, Applesoft BASIC, Integer BASIC, or any Apple IIe application software disk.

Make sure that the appropriate disk is in the drive from which you are loading. To load CP/M, type *BOOT* and press the RETURN key. To load any other operating system, type *BOOT* followed by the appropriate argument, and press the RETURN key.

CAT

CAT [*filespec*]

Purpose

Scans the directory of a disk to determine which files are on that disk.

Remarks

filespec is name of the file or files CAT scans for. Wild card characters (? or *) can be used in the filename and extension. CAT (with no arguments) displays an alphabetical list of filenames on a disk in the specified drive.

The list displayed by CAT is in alphabetical order and shows the size of each file and the amount of remaining unused disk space in kilobytes.

Examples

CAT

Scans the disk in the active drive and displays an alphabetical list of files found.

CAT GBASIC.COM

Scans the disk in the active drive for the file GBASIC.COM. If found, it displays the file, the size of the file in kilobytes, and the amount of free storage space remaining on the disk.

COPY

COPY *d*:=*s*:[/*V*]

Copies the contents of one disk onto another.

COPY *d*:/F

Formats a disk.

COPY *d*:/D[/F][/V]

Creates a CP/M data disk.

COPY *d*:/S[/F][/V]

Creates a CP/M system disk.

Purpose

Copies and formats CP/M disks.

Remarks

The *s*: and *d*: arguments indicate the source drive and destination drive. Each of the different functions of COPY are performed by including the software switch in the COPY command line. The software switches and their function are listed in Table 5.2.

Table 5.2.
Software Switches

Switch	Function
/D	Instructs COPY to create a data disk.
/F	Copies the format to disk.
/S	Instructs COPY to only copy the CP/M operating system onto the first three tracks of the disk.
/V	Verifies the copy process.

COPY can be used from either CP/M command level or from COPY program level. COPY is invoked by typing the appropriate command line format and pressing the RETURN key to execute the command.

If you include the /S switch in the COPY command line, COPY will format the disk if it hasn't been formatted previously. If the disk is already formatted, the files on the disk are not deleted. Use the /F switch to delete the previously formatted files.

If the /D switch is used and the disk is already a CP/M system disk, the CP/M system is deleted and an additional 12K bytes of disk space is made available for programs and data.

Important

Avoid using data disks in drive A: and in single-drive systems. The lack of an operating system on data disks prevents CP/M from performing a warm start and recovering from errors.

Examples

COPY B:=A:/V

Copies the contents of the disk in drive A: onto the disk in drive B: and verifies the copy process by comparing the data contents of the two disks.

COPY A:=C:/V/F

Formats the disk in drive A: and then copies the contents of the disk in drive C: onto the disk in drive A:.

COPY B:/F

Formats the disk in drive B:.

C:=A:

The COPY command line is executed from the program level; it copies the contents of the disk in drive A: onto the disk in drive C:.

B:/S

The COPY command line is executed from the program level; it copies the operating system software from the disk in drive A: onto the disk in drive B:.

d:

d:

Purpose

Changes the active drive in multiple-drive systems.

Remarks

d: is the disk drive identifier.

DDT

DDT [*filename.ext*]

Purpose

Tests and debugs 8080A assembly language programs.

Remarks

filename.ext is the name of the source file to be examined or modified. The source file must have an extension of .COM or .HEX, or DDT will not recognize it. If you do not enter the filename with the DDT command, DDT is loaded into memory and waits for further instructions.

DDT is the CP/M Dynamic Debugging Tool. It is used in conjunction with the ASM assembler to test and debug assembly programs. You can also use DDT for examining and modifying your programs.

To invoke DDT, type the DDT command line and press RETURN. If you include the filename in the command line, DDT will display the DDT version number, the next available memory location (denoted by NEXT), the program counter setting (denoted by PC), and the DDT program prompt (-). If you enter DDT without the filename, only the version number and the prompt appear.

When the DDT program prompt appears, you can use any of the DDT commands listed in Table 5.3.

Table 5.3.
DDT Commands

Command	Purpose
A*nnnn*	Enters assembly language statements starting at address *nnnn*.
D	Displays the contents of the next 192 bytes of memory.
D*ssss,ffff*	Displays memory contents starting at address *ssss* to address *ffff*.
Fssss,ffff,cc	Fills memory with constant *cc* from address *ssss* to address *ffff*.
G	Begins execution at the address contained in the program counter.
G*ssss*	Begins execution at address *ssss*.
Gssss,bbbb	Sets a breakpoint at address *bbbb*; begins execution at address *ssss*.
G,*bbbb*	Sets a breakpoint at address *bbbb*; begins execution at the address contained in the program counter.
G,*bbbb,cccc*	Sets breakpoints at addresses *bbbb* and *cccc*; begins execution at the address contained in the program counter.
I*filename.ext*	Sets up the default File Control Block using the name *filename.ext*.
L	Lists the next eleven lines of the assembly language program.
L*ssss*	Lists eleven lines of the assembly language program starting at address *ssss*.
L*ssss,ffff*	Lists the assembly language program which starts at address *ssss* and finishes at address *ffff*.

Table 5.3. *(continued)*

Command	Purpose
M*ssss,ffff,dddd*	Moves the memory block (address *ssss* to *ffff*) to address *dddd*.
R	Reads a file from disk.
Rnnnn	Reads a file from disk, beginning at address *nnnn*.
S*ssss*	Displays memory contents at address *ssss* and optionally changes the contents.
T*nnnn*	Traces the execution of *nnnn* program instructions.
U*nnnn*	Executes *nnnn* program instructions, then stops and displays the contents of the CPU registers.
X	Displays the contents of the CPU registers.
X*r*	Displays contents of CPU registers or flag *r* and optionally changes it.

DDT can be aborted at any time by typing CONTROL-C.

Examples

The following example shows how DDT would be invoked and the results of using some of the DDT commands.

```
DDT DUMP.COM
```

Loads DDT and the file DUMP.COM into memory.

```
DDT VERS 2.2
NEXT   PC
1E00  0100
-
```

Displays the DDT version number. NEXT identifies the next free memory location (1E00). PC identifies the program counter setting (0100). "-" is the DDT prompt.

L

When RETURN is pressed, DDT displays the next 11 lines of assembly language disassembled from memory.

```
0100  LXI     H,0000
0103  DAD     SP
0104  SHLD    0215
0107  LXI     SP,0257
010A  CALL    01C1
010D  CPI     FF
010F  JNZ     011B
0112  LXI     D,01F3
0115  CALL    019C
0118  JMP     0151
011B  MVI     A,80
```

DIR

DIR [*d*:][*filename.ext*]

Purpose

Scans a specified disk to determine what files are on that disk.

Remarks

d: is the specified drive and *filename.ext* is name of the file or files DIR scans for. Wild card characters (? or *) can be used in the filename and extension. Entering DIR without any arguments displays only the sequential list of filenames on a disk in the specified drive.

Examples

DIR

Displays all files on the disk in the active drive.

DIR GBASIC.COM

Displays GBASIC.COM on the disk in the active disk.

DIR A:*.COM

Displays all the files with an extension of .COM on disk in drive A:.

DIR B:

Displays all files on the disk in drive B:.

DUMP

DUMP *filespec*

Purpose

Displays the contents of a disk file in hexadecimal form.

Remarks

filespec is the location and name of the file.

To invoke DUMP, type *DUMP* in the command line format at CP/M command level and press RETURN. The hexadecimal contents of the file will be displayed on the terminal's screen. DUMP lists 16 bytes at a time with each line's absolute address on the left.

Example

```
DUMP B:CAT.COM
```

This command line will display the contents of the CAT.COM file in the following format:

```
0000 ED 73 DD 03 31 05 04 CD 45 01 CD 4C 02 0E 11 11
0010 5C 00 CD 05 00 3C 28 16 CD DF 01 CD 80 01 0E 12
...
```

ED

ED *filespec*

Purpose

Creates and edits CP/M ASCII text files.

Remarks

filespec is the location and name of the file to be edited. You must include the extension with the filename. Enter the drive letter (d:) if the file is on a drive other than the active drive.

ED is the CP/M editor. It is used to create and edit CP/M ASCII text files. ED provides the basic requirements for inserting and deleting text, moving from line to line, and searching for text.

At CP/M command level, type *ED* and the filename. Press RETURN to load ED into memory. ED then creates a temporary file (the name of the file with an extension of .$$$) for editing.

When you see the asterisk prompt on the screen, the file is ready to edit. The commands available for editing are listed in Table 5.4.

Table 5.4.
Commands for Editing

Command	Action
*n*A	Moves the number of lines specified by *n* from the temporary file to the edit buffer.
B	Moves the character pointer (CP) to the beginning of edit buffer. The CP takes the place of the cursor.
-B	Moves the CP to end of edit buffer.
*n*C	Moves the CP *n* characters forward.
-*n*C	Moves the CP *n* characters backward.
*n*D	Deletes *n* characters after the CP.
-*n*D	Deletes *n* characters before the CP.
E	Ends edit session, closes files, and returns to CP/M.
*n*F*string* CONTROL-Z	Finds the *n*th occurrence of *string*.
H	Ends edit session, closes and reopens files.
I	Enters insert mode.
I*string* CONTROL-Z	Inserts *string* into the edit buffer.
I*string*	Insert a line of text specified by *string*.
*n*J*fstring*CONTROL-Z*istring1*[,*istring2*, *istring3*,...]CONTROL-Z*estring*CONTROL-Z	The *n* argument specifies how many times the following operation is repeated. Beginning after the CP, ED searches for *fstring*. If found, it inserts *istringn* after it. Then, ED deletes all characters following up to, but not including, *estring*.
*n*K	Deletes *n* lines after the CP.
-*n*K	Deletes *n* lines before the CP.
*n*L	Moves the CP forward *n* lines.
-*n*L	Moves the CP backward -*n* lines.
*n*M*cmdstring* CONTROL-Z	Repeats execution of the ED commands specified by the command string *cmdstring n* times.

Table 5.4. *(continued)*

Command	Action
*n*N*string* CONTROL-Z	Searches for the *n*th occurrence of *string* throughout the file.
O	Returns to original file.
*n*P	Moves the CP forward and prints *n* pages.
-*n*P	Moves the CP backward *n* pages and displays the page following the CP.
Q	Quits edit session with no changes saved.
R	Reads temporary file, X$$$$$$$.LIB into the edit buffer.
R*filename*	Reads library file *filename*.LIB into the edit buffer.
*n*S*fstring*CONTROL-Z*rstring*CONTROL-Z	Searches for *fstring* and replaces with *rstring*. Repeats the operation *n* times.
*n*T	Displays *n* lines preceding the CP.
-*n*T	Displays *n* lines following the CP.
0T	Displays all text from the beginning of the line to the CP.
T	Displays all text from the CP to the end of the line.
0TT	Displays the entire line without moving the CP.
U	Converts text to uppercase.
0V	Displays edit buffer free space in bytes.
V	Verifies line numbers.
*n*W	Writes *n* lines to disk.
*n*X	Copies *n* lines (starting at the CP) to temporary library file X$$$$$$$.LIB.
*n*Z	Delays execution of the command which follows by *n* seconds.
n:	Moves the CP to line number *n*.
[-]*n*	Moves the CP forward or backward and displays one line.

Examples

ED DEMO.BAS

Loads ED into memory and creates the temporary file DEMO.???. The temporary file is then loaded into the edit buffer.

:*

ED command prompt; ED is ready for your command.

ERA

ERA *filespec*

Purpose

Erases specified files from a disk.

Remarks

filespec is the location and the name of the file or files to be erased. Wild card characters (? or *) can be used in the filename or extension.

Examples

ERA B:TEMP.OLD

Erases the file TEMP.OLD on the disk in drive B:.

ERA C:*.BAS

Erases all files with the extension .BAS on the disk in drive C:.

ERA * *

Erases all files on the disk in the active drive.

LOAD

LOAD *filespec*

Purpose

Performs the final step in preparing an assembly language program for execution by converting a disk file with the extension .HEX into a machine-executable command file (with an extension of .COM).

Remarks

filespec is the location and the name of the file with a .HEX extension. The extension need not be included with the filename. LOAD assumes it is a HEX file. Enter the drive letter if the file is on a drive other than the active drive.

At CP/M command level, type *LOAD* in the specified format and press RETURN. LOAD creates a COM file in memory which begins with address 0100H. To save the COM file, use the SAVE command.

Example

```
LOAD B:TIME
```

Loads the TIME.HEX file from drive B:.

```
FIRST ADDRESS       0100
LAST ADDRESS        0222
BYTES READ          0130
RECORDS WRITTEN     02
```

When the file is loaded, the screen displays the starting address (0100), the last address (0222), the number of bytes (130), and the number of records (2) written by LOAD into the file TIME.COM.

MFT

MFT *filespec1*[,*filespec2*...]

Purpose

Copies files from one disk to another on single-drive systems.

Remarks

filespec is the specification of the files to be copied. Wild card characters (? and *) can be used in the file specifications.

MFT is invoked by typing *MFT* at CP/M command level. The copy process is started when you press the RETURN key.

Important

You must have a CP/M system disk in disk drive A: before typing CONTROL-C.

Examples

MFT *.COM

Copies all COM files on the source disk to the destination disk at CP/M command level.

MFT MBASIC.COM,CONFIGIO.BAS

Copies the GBASIC.COM and CONFIGIO.BAS files from the source disk to the destination disk at CP/M command level.

PATCH

PATCH {*filespec*|*offset*}=[*p1 p2 p3...*][(*v1 v2 v3*)]

Purpose

Installs program updates and modifications to the CP/M system modules.

Important

The only time you should have to use PATCH is when you receive explicit instructions from Microsoft Corporation. If you wish to install your own modifications or updates without instructions from Microsoft, do so at your own risk.

Remarks

filespec is the name of the COM file to be modified.

offset is a one through six digit hexadecimal byte offset. The offset is from the beginning of the disk if the CP/M system tracks are to be modified.

p1, *p2*, *p3* are two-digit hexadecimal byte "patches."

v1, *v2*, *v3* are optional two-digit hexadecimal verification bytes.

Spaces are required between all byte arguments.

If modifications are made to a COM file, specify the disk and the file by typing the *filespec* argument. If modifications are made to the CP/M system tracks, use the *offset* argument. If the *filespec* is included, the offset is from the beginning of the file starting at byte 0.

The bytes following the equals operator (=) are written to the specified file. If there is no file specified, the bytes are written to the location specified by the offset argument.

Once the patch is made, the asterisk prompt reappears. Repeat the procedure to install another patch, or type CONTROL-C to return to CP/M command level.

PIP

PIP *d*:[*filespec*]=[*s*:]*filespec*[*p*]

Copies a file to another disk.

PIP [*d*:]*newfilespec*=[*s*:]*oldfilespec*[*p*]

Renames the destination file during the copy process.

PIP *d*:[*filespec*]=[*s*:]*filespec*[*gn*]

Copies files from different user areas to the active user area.

PIP [*d*:]*dest*=[*d*:]*source1*,*source 2...*

Appends disk files (concatenation).

PIP LST:=*filespec*[*p*]

Sends data to an output device, such as a printer or terminal.

PIP *ddest*:=*sdest*:[*p*]

Copies data between I/O devices.

Purpose

Copies data between files or devices.

Remarks

d: is the destination drive and *s*: is the source drive.

filespec is the file specification of the file or files from which you are copying. If you are changing the name of the copied file, *newfilespec* is the new filename and *oldfilespec* is the old filename.

[*p*] is the parameter argument. The parameters that can be used with PIP are listed in Table 5.5, "PIP Parameter Summary."

If you are copying files, *dest* is the destination file of the copy operation and *source* is the source file. Commas must separate the source file arguments.

If you are copying data between devices, *ddest*: is the destination device and *sdest:* is the source device of the copy process.

PIP can be used by typing the appropriate command line format. Press RETURN to execute the command. PIP can be aborted at any time by pressing the space bar or any other key during the copying process. PIP confirms that the process has been aborted by displaying the message "ABORTED."

Table 5.5.
PIP Parameter Summary

Parameter	Action
B	Specifies block mode transfer.
D*n*	Deletes all characters after the *n*th column.
E	Echoes the data being copied to the screen during the copy process.
F	Removes formfeed characters from data during the copy process.
G*n*	Copies a file from user area *n* to the active user area.
H	Checks for proper Intel® HEX file format.
I	Ignores any null records in Intel HEX file copy operations.
L	Translates uppercase letters to lowercase.
N	Adds a line number to each line copied.
O	Object file copy operation (ignores end-of-file markers).
P*n*	Inserts page ejects after every *n*th line; the default value is 60 lines.
Q*string*CONTROL-Z	Copies only a portion of the file up to *string*.
R	Directs PIP to copy from a system file.
S*string*CONTROL-Z	Copies only the portion of the file from *string* to the end of the file.
T*n*	Sets tab stops to every *n*th column.
U	Translates lowercase letters to uppercase.
V	Verifies copy by comparison after the copy process has been finished.
W	Directs PIP to copy onto an R/O file.
Z	Zeros the parity bit on input for each ASCII character.

Examples

PIP B:=*.BAS

Copies all files with the extension of .BAS on the active drive to drive B:.

PIP DOG.COM=CAT.COM

Copies the file CAT.COM into a new file called DOG.COM on the active drive.

B:ED.COM=A:

Copies the file ED.COM from drive A: to drive B: under the same name.

B:=S*.COM

Copies all the files on the active drive that start with the letter "S," and have an extension of .COM to drive B:.

REN

REN [*d*:]*new filename.ext=old filename.ext*

Purpose

Renames files while leaving the file text intact.

Remarks

new filename.ext is the new name of the file and *old filename.ext* is the original name of the file. Wild card characters cannot be used in either the old filename or the new filename.

Examples

REN TEMP.NEW=TEMP.OLD

Renames TEMP.OLD as TEMP.NEW.

REN B:PEAR.COM=APPLE.COM

Renames APPLE.COM on drive B: as PEAR.COM.

SAVE

SAVE *nnnfilespec*

Purpose

Saves the contents of memory in a specified disk file.

Remarks

nnn is the number of memory pages to be saved.

filespec is the drive and the name of the file in which to save the memory contents.

Example

SAVE 26 C:MYPROG.COM

Saves 26 pages of memory in a file called MYPROG.COM on disk drive C:.

STAT

STAT [*d*:]

Displays disk drive status.

STAT *d*:{DSK:|USR:}

Displays active disk and user area status.

STAT *filespec*

Displays file status.

STAT {*d*:|*filename.ext*}$*attribute*

Assigns attributes to files and disks.

STAT *log:=phy*:

Makes device assignments.

STAT VAL:

Displays possible STAT commands.

STAT DEV:

Displays the current device assignments.

Purpose

Displays status information and changes device assignments.

Remarks

d: is the disk drive identifier.

filespec is the name of the file or files from which you want to obtain status information. Wild card characters can be used to obtain status information on more than one file at a time.

attribute is one of the attributes from Table 5.6, "File and Disk Attributes," that can be assigned to the file or disk.

log: and *phy*: are the logical and physical I/O devices.

STAT is executed by typing the appropriate command and pressing the RETURN key. STAT is executed from CP/M command level only.

Table 5.6.

File and Disk Attributes

Attribute	Action
$R/O	Prevents writing to or deleting the file.
$R/W	Allows writing to and deleting the file. This attribute cancels $R/O.
$SYS	Prevents the display of the file when the DIR built-in command is invoked.
$DIR	Cancels the $SYS attribute.

Examples

STAT

Displays file attributes and amount of free space (in kilobytes) for all disk drives since the last warm or cold start.

STAT B:

Displays amount of disk free space in drive B:.

STAT DEMO.BAS

Displays size and attributes of DEMO.BAS file on the active drive.

STAT B:DOG.COM $R/O

Assigns the $R/O attribute to DOG.COM on drive B:.

STAT CON:=TTY:

Assigns the physical device TTY: to the logical device CON:.

STAT C:$R/O

Assigns a temporary write-protect status to drive C:.

SUBMIT

SUBMIT *filespec abc*

Purpose

Creates a file which contains commands to be executed from a disk file rather than from the keyboard.

Remarks

filespec is the location and filename of a text file to be submitted. The filename must have a .SUB extension. The extension need not be included with the filename; SUBMIT assumes it is a SUB file. Enter the drive letter if the file is on a drive other than the active drive.

a, *b*, and *c* are arguments for optional variables in the SUBMIT file. The variables can be filenames or other information needed by the commands in the SUBMIT file. The symbols $1, $2, and $3 are substituted for missing parameters in format 2.

Examples

File: TEST.SUB

The name of the SUBMIT file.

CAT $1.BAS

The contents of the SUBMIT file.

```
PIP $2:=$1.BAS
GBASIC $1
SYSTEM
ERA $1.BAS
```

This program looks for a GBASIC file named by variable $1. PIP copies the file to the drive named by variable $2. GBASIC then executes the file. The file is erased after execution.

SUBMIT TEST DEMO B

Loads SUBMIT into memory and creates the SUBMIT file, $$$.SUB, from the file, TEST.SUB. $$$.SUB executes the commands from TEST.SUB: and searches for DEMO.BAS. The SUBMIT command then copies it to drive B:, runs GBASIC and DEMO.BAS, and erases DEMO.BAS after execution.

TYPE

TYPE *filespec*

Purpose

Displays the contents of a specified text file on the screen.

Remarks

filespec is the location and the name of the file. No wild card characters are allowed in the filespec.

Example

TYPE DUMP.ASM

Displays the contents of the file DUMP.ASM on the screen.

USER

USER *n*

Purpose

Separates disk memory into user areas.

Remarks

The user areas are designated by numbers. *n* is the number of the user area.

XSUB

XSUB

Purpose

XSUB is a variation of SUBMIT, which allows constant character input from a disk file during program execution.

Remarks

Introduce XSUB as the first line of a SUBMIT file (*filename.SUB*). Run the SUBMIT file as instructed by the command prompts. When CP/M processes the SUBMIT file, it relocates the XSUB program directly below the CCP in memory in order to process the command lines of the SUBMIT file. The XSUB program remains active until all the commands in the SUBMIT file have been executed or until a cold start has been performed.

Chapter 6
I/O Configuration

The SoftCard version of CP/M can be modified for use with different I/O devices and software. This chapter describes the following areas of CP/M that can be modified:

The screen function interface

The Keyboard Character Definition Table

Patch areas for I/O software

All three areas can be changed or examined with the CONFIGIO utility program.

CONFIGIO

CONFIGIO is a utility program that changes designated areas of the BIOS. CONFIGIO consists of a series of menus that allow you to perform the following functions:

Examine and modify the screen function interface for use with an external terminal

Redefine keyboard characters

Load user I/O driver software into designated user patch areas

Save changes made with CONFIGIO on a system disk

Running the CONFIGIO Program

The CONFIGIO program is on the SoftCard II Master disk. To run it, insert a CP/M system disk that contains CONFIGIO.BAS and GBASIC.COM into drive A:. Load CP/M with a cold start. When you see the CP/M command level prompt A>, type

```
GBASIC CONFIGIO
```

and press the RETURN key.

When CONFIGIO has been loaded into memory, the screen displays a menu, as shown below. Each selection allows you to perform the task named. To select a task, press the number key corresponding to the task you wish to perform.

```
+ + CONFIGIO SELECTION MENU + +

   1.  Configure Screen Function Interface

   2.  Redefine Keyboard Characters

   3.  Load User I/O Driver Software

   4.  Read/Write Changes Made

   Q.  Quit Program

   Select - ▒
```

CONFIGIO Menu Selections

1. Configure Screen Function Interface

   This selection allows you to specify the control sequences required for an external terminal or application program to execute specific screen functions. Instructions for configuring the screen function interface for an external terminal are provided in the "Configuring the Screen Function Interface" section of this chapter.

2. Redefine Keyboard Characters

   This selection allows you to redefine the ASCII value assigned to any particular key on the keyboard, such as a seldom-used control character. Instructions for redefining keyboard characters are given in the "Redefining Keyboard Characters With CONFIGIO" section of this chapter.

3. Load User I/O Driver Software

   This option allows you to load the necessary I/O driver software into the patch areas for use with nonstandard Apple I/O devices or I/O software. If you are adding an I/O device that requires special I/O software, the technical manual for that device should give explicit instructions on how to load the I/O software into memory. If it does not, contact the manufacturer of the I/O device.

   If you are planning to add your own I/O software to the patch areas, read "Adding Nonstandard I/O Devices and User Software" in this chapter.

4. Read/Write Changes Made

   This option allows you to save the changes made with CONFIGIO menu selections 1 through 3. Instructions for using menu selection 4 are listed with instructions on using the other menu selections in this chapter.

Q. Quit Program

   Pressing *Q* exits the CONFIGIO program and returns to the CP/M operating system.

Screen Function Interface

The screen function interface controls how characters are displayed on the Apple screen or on the screen of an external terminal. Screen functions (also called screen attributes) are special control sequences that govern the display characteristics of the screen monitor or terminal. Some application programs are written for more than one computer and must be modified to display characters on the screen correctly.

Most popular terminals, including the standard Apple screen monitor, support special screen functions such as direct cursor addressing, screen clear, and highlighted text. Many CP/M application programs, such as word processing packages and business software, use these functions as part of the application display. The character sequences, however, often differ from terminal to terminal.

The screen function interface is configured for the standard Apple screen monitor. The Soroc IQ™ 120/IQ 140, Hazeltine™ 1500/1510, and Datamedia terminals can be used as external terminals without any modifications to the screen function interface. If you use an external terminal that is not compatible with your application software, special assembly language subroutines must be written to resolve the differences.

Screen Function Tables

The screen function interface solves the compatibility problem by translating the functions (as they are received from the user software) into the corresponding functions expected by the screen display's circuits. This is carried out by two translation tables: the Software Screen Function Table and the Hardware Screen Function Table.

The Software Screen Function Table recognizes an incoming screen function sequence and translates it into the corresponding sequence found in the Hardware Screen Function Table. This sequence is then sent to the terminal device.

Screen Functions Supported

The screen function interface recognizes and translates the following screen functions:

Clear Screen

Clears the entire screen, fills the screen with spaces, and places the cursor in the home position.

Clear to End-of-Page

Clears all information from the cursor (including the cursor position) to the end of the page.

Clear to End-of-Line

Clears all information from the cursor (including the cursor position) to the end of the line.

Set Normal (lowlight) Text Mode

Sets the normal video display mode; characters are displayed as white characters on a black background.

Set Inverse (highlight) Text Mode

Sets the inverse video display mode; characters are displayed as black characters on a white background.

Home Cursor

Moves the cursor to the first character position on the first line.

Address Cursor

Sets the cursor address for a specified printer offset.

Move Cursor Up

Moves the cursor up one line. If the cursor reaches the top line of the screen, it remains there and no scrolling occurs.

Move Cursor Forward

Moves the cursor one cursor position to the right, but does not destroy the character in that position. If the cursor is at the right end of the line, it will remain there.

In addition, there are two other screen functions which are used on all terminals: backspace and linefeed. The backspace character (ASCII 8) function moves the cursor backwards, and the linefeed character (ASCII 10) function moves the cursor down one line.

The control sequences for screen functions are a single control character or an ASCII character preceded by a single lead-in character. Control sequences consisting of three or more characters are not supported.

Configuring the Screen Function Interface

Load and run the CONFIGIO program as instructed in the "CONFIGIO" section at the beginning of this chapter.

Note

Before configuring the screen function interface for an external terminal, ensure that there is no accessory board installed in slot 3. If there is, turn the power off, remove the board, and use the standard Apple video screen monitor (see Figure 2.2 in Chapter 2 of the *Microsoft SoftCard II Installation and Operation Manual*). Once the configuration process is complete, you can reinstall the board and use its screen monitor as before.

When the CONFIGIO selection menu appears (see page 162), press the *1* key. CONFIGIO will display the Hardware and Software Screen Function Tables as shown below:

```
+ SCREEN FUNCTION INTERFACE MENU +

     FUNCTION          SOFTWARE              HARDWARE

Clear Screen     :    ESC *          →          FF
Clr To EOS       :    ESC Y          →          VT
Clr To EOL       :    ESC T          →          GS
Lo-lite Text     :    ESC )          →          SO
Hi-lite Text     :    ESC (          →          SI
Home Cursor      :        RS         →          EM
Address Cursor   :    ESC =          →          RS
XY Coord Offst   :        32         →          32
XY Xmit Order    :        YX         →          XY
Cursor Up        :        VT         →          US
Cursor Forward   :        FF         →          FS

        1. SOROC IQ 120 IQ 140
        2. HAZELTINE 1500/1510
        3. DATAMEDIA
        4. Other
        Q. Quit

     Select -
```

The previous menu shows the default values of the Hardware and Software Screen Function Tables. Items in the Software column are the default control sequences of the Software Screen Function Table. Items in the Hardware column are the ASCII codes needed by the terminal hardware to perform the stated screen function. A NUL (ASCII 00) entry in either table indicates that the function is not available.

Three of the numbered entries in the lower section of the screen are for terminals for which CONFIGIO has data. To configure the screen function interface for any of the terminals listed, type the menu number corresponding to the terminal. For terminals not listed, or for application programs requiring modifications to the screen function interface, press the *4* key. To return to the main CONFIGIO menu, press the *Q* key.

For application programs requiring changes to the screen function interface, the Software Screen Function Table is modified. External terminals will usually require modifications to the Hardware Screen Function Table.

The Software Screen Function Table must match sequences sent by the application program to perform screen functions. The Hardware Screen Function Table must have non-zero entries in all of the nine functions. We recommend setting up the Software Screen Function Table to emulate a Soroc IQ 120/IQ 140 terminal. This is a common configuration that is supported by most CP/M software.

Configuring for an External Terminal

For Soroc IQ 120 or IQ 140 terminals, no changes are needed for either screen function table. However, when you first turn on Soroc terminals, text is shown in the "highlight" mode. CP/M will reset the screen to display in a normal "lowlight" mode whenever a cold start is performed.

For Hazeltine 1500/1510 terminals, use the Hardware Screen Function Table only. (CP/M translates the Hazeltine cursor-addressing function with no XY coordinate offset.) We do not recommend using the Hazeltine screen function sequences in the software table. It is best to set up the hardware table for the Hazeltine, and the software table for another common terminal, such as the Soroc IQ 120/IQ 140.

For Datamedia terminals, set up the Hardware Screen Function Table only. (Datamedia terminal control sequences are not usually supported by CP/M software.) Set the hardware table for use with a 24x80 video board, and the software table for another common terminal type, such as the Soroc IQ 120/ IQ 140.

Note

Highlight text and lowlight text screen functions (GBASIC commands INVERSE and NORMAL) are not supported by Datamedia terminals. Thus, the table entries specified for these functions are set to an arbitrary value to ensure that these two entries will have non-zero values.

To configure the screen interface for a terminal not listed in the menu, press the *4* key when the Screen Function Interface menu appears. CONFIGIO will load and display the list of configurable screen functions shown in the following figure.

```
+ + SCREEN FUNCTION DEFINITION + +

    1 -  Lead-in Character
    2 -  Clear Screen
    3 -  Clr To EOS
    4 -  Clr To EOL
    5 -  Lo-Lite Text
    6 -  Hi-Lite Text
    7 -  Home Cursor
    8 -  Address Cursor
    9 -  Cursor Up
   10 -  Cursor Forward
    Q -  Quit

    Select -
```

You can now change any of the values in the Terminal Screen Function Definition Table.

Note

The appropriate screen function command characters for your terminal are described in the technical manual for that terminal. To find out which codes are transmitted by a particular program (for example, a word-processing program), consult the manual for that program.

Select a number (1 through 10) to define the character sequences for any of the functions listed in Table 6.1.

Table 6.1.

Screen Function Descriptions

Number	Title	Description
1	Lead-in character	Defines the lead-in character: the character (usually an escape sequence) that precedes the screen function command character. A particular screen function may or may not require a lead-in character.
2	Clear screen	Clears the screen and places the cursor at the top left corner of the screen.
3	Clear to EOS	Clears the screen from the cursor to the end of the screen.
4	Clear to EOL	Clears the screen from the cursor to the end of the line.
5	Lowlight text	Sets the normal video mode for displaying text.
6	Highlight text	Sets inverse or double intensity video mode, depending on which mode your terminal supports.
7	Home cursor	Puts the cursor at the top left corner of the screen, but does not clear the screen.
8	Address cursor	Tells the terminal to go to a cursor address defined by the next two characters entered.
	XY coordinate offset	Defined as part of selection 8. The XY coordinate offset is the number that is added to the X and Y coordinates when they are sent to the terminal (usually 32).
	XY transmit order	Also defined as part of selection 8. Establishes the order in which coordinates are transmitted. Must be either XY or YX (usually YX).
9	Cursor up	Moves the cursor up one line on the screen.
10	Cursor forward	Move the cursor forward on a line without deleting the character under the cursor.

To assign an escape sequence to any of these functions, type the corresponding number and press the RETURN key.

For example, press the *1* key if you wish to specify a screen function lead-in character. The program will display:

```
LEAD-IN CHAR:
```

Enter the lead-in character required. Characters can be typed in any one of the following formats:

aaa	where *aaa* is a 2- or 3-character ASCII name.
c	where *c* is any keyboard character.
CONTROL-*c*	where *c* is any character.
LC-*c*	LC- indicates that the following character is lowercase. Type this in place of a lowercase character if your keyboard has no lowercase characters.
&H *hh*	*hh* is the ASCII hexadecimal code (preceded by &H). Use this format if the character cannot be typed. (See "ASCII Character Codes," Appendix H, in the *Microsoft BASIC Interpreter Reference Manual*.)

After you have entered the lead-in character, the program will ask:

```
SOFTWARE OR HARDWARE (S/H)?
```

If the lead-in character is to be used in the Software Screen Function Table, press the *S* key. If the lead-in character is to be used in the Hardware Table, press the *H* key.

To define any of the other screen functions, press the number for that function. The program will prompt you for the command character for that particular function.

The program then returns to the Screen Function Definition menu and waits for you to select another number or *Q*. You can make as many changes to the tables as you wish in this way.

The process for the address cursor function differs somewhat. If you press *8*, address cursor, the process is the same as with the other selections, until you see the prompt:

```
REQUIRE LEAD-IN (Y/N)?
```

After you answer this prompt by pressing *Y* or *N*, the computer displays:

```
XY COORD OFFST :
```

Type a numeral for the number of spaces that are to be added to the X and Y coordinates before they are transmitted. Finally, the program asks:

```
XY XMIT ORDER :
```

If the X and Y coordinates are transmitted in the order Y then X, enter *YX*. If the coordinates have been transmitted X then Y, enter *XY*.

The program then asks

```
SOFTWARE OR HARDWARE (S/H)?
```

and then continues in the same manner as with the other functions.

Configuring for Application Programs

Use the same procedure as that used for external terminals. Most application programs will give explicit instructions on how to configure the screen function interface. If a program requires changes to the screen function interface, but doesn't give instructions, use the following procedure:

1. Load and run the CONFIGIO program as instructed in the "CONFIGIO" section in the beginning of this chapter.

2. When the CONFIGIO selection menu appears (see page 162), press the *1* key. CONFIGIO will display the Screen Function Interface Menu as shown on page 167.

3. Press the *4* key.

4. Select the desired function by pressing the appropriate key or keys.

5. When

   SOFTWARE OR HARDWARE (S/H)?

   appears, press the *S* key.

6. Type the appropriate control sequence listed by the application program documentation.

7. Save the changes that you have made in the screen function interface. (See the following section for more information on saving changes.)

Saving the Changes to the Screen Function Interface

Save the changes made in the screen function interface by first pressing the *Q* key. When the main CONFIGIO menu appears, press the *4* key. The program will display:

```
+ READ/WRITE I/O CHANGES MADE +

Read Or Write (R/W)?
```

Press the *W* key. The program will display:

```
Destination Drive (A:-D:)?
```

Press the *A* key to save the changes made in the screen interface on the system disk in drive A:. The program will then display the main CONFIGIO menu.

Using the Screen Function Interface From Within a Program

The screen functions listed in Table 6.1, "Screen Function Descriptions," make it possible to write programs that perform special screen functions. Table 6.2, "Screen Function Interface Addresses," shows the correspondence between the Software and the Hardware Screen Function Tables in memory. It lists the function number and the hexadecimal address of each entry. The internal format of the two 11-byte tables is identical.

Table 6.2.

Screen Function Interface Addresses

Function Number	Software Table Address	Hardware Table Address	Function Description
	0F396H	0F3A1H	Cursor address coordinate offset. Range: 0 to 127. If the high-order bit is 0, the X and Y coordinates are expected to be transmitted Y first, X last. If the high-order bit is 1, the coordinates are sent X first, Y last.
	0F397H	0F3A2H	Lead-in character. This byte is zero if there is no lead-in character.
1	0F398H	0F3A3H	Clear screen.
2	0F399H	0F3A4H	Clear to end-of-page.
3	0F39AH	0F3A5H	Clear to end-of-line.
4	0F39BH	0F3A6H	Set normal (lowlight) text mode.
5	0F39CH	0F3A7H	Set inverse (highlight) text mode.
6	0F39DH	0F3A8H	Home cursor.
7	0F39EH	0F3A9H	Address cursor.
8	0F39FH	0F3AAH	Cursor up.
9	0F3A0H	0F3ABH	Cursor forward.

A NUL character entry in either Screen Function Interface Table will disable that function on the standard Apple screen monitor.

The standard Apple screen monitor supports all nine screen interface functions, independent of the Hardware Screen Function Table. However, if a Software Screen Function Table entry is zero, the function is disabled.

If the lead-in character of the Hardware Screen Function Table is OFF, the entire table is bypassed.

If a numbered table entry is zero, the function is not implemented.

If the entry has 1 as the high-order bit, the function requires a lead-in character.

An entry with the high-order bit set to zero indicates that the function does not require a lead-in character.

To ensure portability, the Hardware Screen Function Table must be set up correctly for the specific terminal. The following example lists a short segment of 8080 assembly language code which illustrates the use of the Screen Function Tables for terminal independent screen programming.

```
;               Terminal Independent Screen I/O
;
;               This routine will execute the screen function specified by E, where E
;               contains the screen function number from one to nine. If the function is
;               not implemented, the subroutine simply returns, and all registers are
;               destroyed.
;
;               Equates:
;
BDOS            EQU     0005H           ;CP/M function call address
SXYOFF          EQU     0F396H          ;Software cursor address XY coordinate offset
SFLDIN          EQU     0F397H          ;Software function lead-in character
SSFTAB          EQU     0F398H          ;Software screen functions
;
SCRFUN:         MVI     D,0             ;Prepare for index
                LXI     H,SSFTAB-1      ;Point to Software Screen
                                        ;Function Table minus one
                DAD     D               ;Index to desired function character
                MOV     A,M             ;Get the character
                ORA     A               ;See if a lead-in is required
                RZ                      ;If the function isn't there, quit
                JP      CONOUA          ;If positive, no
                PUSH    PSW             ;Save character
                LDA     SFLDIN          ;Get software lead-in character
                CALL    CONOUA          ;Output character in A
                POP     PSW             ;Get character again
CONOUA:         MOV     E,A             ;Put character in its place
CONOUE:         MVI     C,2             ;Console output function
                JMP     BDOS            ;Call CP/M BDOS at 0005H
;
;               This routine will position the cursor at the X,Y coordinates in HL.
;
GOTOXY:         PUSH    H               ;Save coordinates while we do sequential
                                        ;addressing
                MVI     E,7             ;Do an address cursor function
                CALL    SCRFUN
                POP     H               ;Get coordinates back
                LDA     SXYOFF          ;Get software XY coordinate offset
                ORA     A               ;Set CC's on A
                JP      NORVS           ;Reverse coordinates if negative
                MOV     E,L             ;Reverse H and L
                MOV     L,H
                MOV     H,E
NORVS:          MOV     E,A             ;Save offset
                ADD     H               ;Add offset
                MOV     H,A             ;Save for later
                MOV     A,E             ;Get offset again
                ADD     L
                PUSH    H               ;Save all this
                CALL    CONOUA          ;Output first coordinate
                POP     H               ;Restore coordinates
                MOV     E,H             ;Output second coordinate and return
                JMP     CONOUE
```

Keyboard Character Definition

Some CP/M application programs require the use of keys which are not available on the Apple keyboard. For example, the Apple keyboard does not have a RUBOUT key. This can be resolved by redefining specific keys in the Keyboard Character Definition Table located at memory locations F3ACH through F3B7H.

Keyboard Character Definition Table

The Keyboard Character Definition Table supports up to six character redefinitions. Entries in the table consist of two bytes: the first byte is the ASCII value of the keyboard character to be redefined, and the second byte is the desired ASCII value of the character. Both bytes must have their high-order bits cleared.

If there are fewer than six entries in the Keyboard Character Definition Table, a byte with the high-order bit set is put at the end of the table.

Redefining Keyboard Characters With CONFIGIO

Load and run the CONFIGIO program as instructed in the "CONFIGIO" section in this chapter. When the first CONFIGIO selection menu appears, press the *2* key. CONFIGIO will display the Keyboard Character Definition Table.

To redefine a character response for a key, press the *A* key. To delete an entry from the table, press the *D* key. Press the *Q* key to return to the main CONFIGIO menu.

When you press the *A* key, the CONFIGIO program displays:

```
CHAR:
```

Type the character or character sequence to be defined. The table entry can be typed in one of the following formats:

aaa	where *aaa* is a 2- or 3-character ASCII name.
c	where *c* is any character.
CONTROL-*c*	where *c* is any keyboard character.
LC-*c*	LC- indicates that the following character (*c*) is lowercase. Type this in place of a lowercase character if your keyboard has no lowercase characters.
&H *hh*	*hh* is the ASCII hexadecimal code (preceded by &H). Use this format if the character cannot be typed. See "ASCII Character Codes," Appendix H, in the *Microsoft BASIC Interpreter Reference Manual.*

Save the changes made to the Keyboard Character Definition Table by pressing the *Q* key. When the main CONFIGIO menu appears, press the *4* key. The program will display:

```
+ READ/WRITE I/O CHANGES MADE +

READ OR WRITE (R/W)?
```

Press the *W* key. The program will display:

```
DESTINATION DRIVE (A:-D:)?
```

Press the *A* key to save the changes made in the screen function interface on the system disk in drive A:. The program will then display the main CONFIGIO menu.

Example

CONTROL-C can be redefined as a NUL character (ASCII code 00) to prevent the user from exiting a BASIC program. This is accomplished by running the CONFIGIO program and selecting "2. Redefine Keyboard Characters" from the main CONFIGIO menu.

When the Keyboard Character Definition menu appears, press the *A* key. When the CHAR: prompt appears, type:

```
CONTROL-C
```

and press the RETURN key. If the character is acceptable, the program prompts you to enter the new definition of the character with an arrow as shown:

```
CONTROL-C →
```

Now type

```
NUL
```

and press the RETURN key. If your entry is not acceptable, the computer will erase what you have just entered and wait for an acceptable character entry.

If the entry is acceptable, the Keyboard Character Definition menu is displayed again with the new definitions added to the menu.

Note

If you have followed the example, you will find that you cannot exit the CONFIGIO program with CONTROL-C.

To delete the entry just made, type *D*. CONFIGIO will display the CHAR: prompt again. Now type

CONTROL-C

and press the RETURN key. The list is displayed again with the CONTROL-C → NUL entry deleted.

Type *Q* to return to the main menu.

Notes on Keyboard Character Definition

We recommend that you delete entries to the Keyboard Character Definition Table if they do not apply to your keyboard. For example, if your keyboard has a RUBOUT key, you should delete the DEL entry.

Redefining CONTROL-C as a NUL character to prevent exiting BASIC programs with CONTROL-C is useful, but it can cause problems at CP/M command level. CONTROL-C is used by CP/M for a warm start.

Certain terminals and 80-column display boards perform their own character redefinitions. For example, the Videx™ Videoterm™ display board uses CONTROL-A to switch between uppercase and lowercase input mode. Since CONTROL-A is also used in BASIC to enter edit mode, we recommend redefining another character as CONTROL-A (such as CONTROL-W).

Adding Nonstandard I/O Devices and User Software

The user patch areas and the I/O Vector Table provide a means of using nonstandard I/O devices with CP/M or adding special I/O software. I/O devices include printers, communication interface boards, modems, and other physical devices in addition to terminals. I/O software can be either *substitution* routines or *filter* routines.

Note

Most Apple I/O interface boards contain 6502 ROM drivers. The easiest way to interface these board types to SoftCard CP/M is to call the 6502 subroutines in the ROM. This should be sufficient to interface most common I/O devices to SoftCard CP/M. (See "0 CALLSUB" in Chapter 4 for more information on calling subroutines.)

Substitution routines are the assembly language routines which allow CP/M to communicate with nonstandard I/O devices. ("Nonstandard" applies to any device that is not normally configured for CP/M or Apple Pascal). Most accessory boards will have an accompanying substitution routine for interfacing the board to CP/M.

Substitution routines also include routines that change the normal format of I/O data (from the I/O device) with which the BIOS communicates. The SoftCard version of CP/M treats all substitution routines as "type 1" vector patches. Type 1 vector patches are user-written assembly language routines that are not dependent on the standard BIOS routines.

Filter routines are assembly language routines that change the input data before sending it to the standard BIOS I/O routines. They are called filter routines because they filter the incoming data. Filter routines are considered "type 2" vector patches.

Any I/O routines added to CP/M must be written into the designated user patch areas of the BIOS. I/O routines must have code that alters the BIOS vector so that the BIOS vector points to the user-written routine instead of the standard I/O routine. If your I/O routine is a substitution type of routine, no further action is necessary. If, however, it is a filter type, the normal BIOS vector must be saved and placed in your routine.

User Patch Areas

The SoftCard version of CP/M provides four 64-byte areas for user-written I/O assembly language routines. Three of the areas are for a certain slot. The fourth is for general usage. Table 6.3 shows the memory location of each patch area, the slot assignment, and the assigned logical device for the patch area.

Table 6.3.

User Patch Areas

Address Range	Assigned Slot	Assigned Logical Device
0FE00H—0FE3FH	1	LST:
0FE40H—0FE7FH	2	PUN: and RDR:
0FE80H—0FEBFH	3	TTY:
0FEC0H—0FEFFH	None	Use for filter routines or to continue a substitution routine.

If there is no board installed in a particular slot, its allocated 64-byte space in the patch area can be used for other purposes relating to its assigned logical device.

I/O Vector Table

All of the "primitive" character I/O functions used by the Apple I/O system are performed through the I/O Vector Table. These vectors point to the standard I/O subroutine located in the CP/M BIOS, but can be altered by the CONFIGIO program to point to user-installed I/O driver subroutines.

I/O driver subroutines are "patched" to CP/M by adding the appropriate I/O vector which points to the specified subroutine. Table 6.4 lists vector locations and their purposes.

Table 6.4.
I/O Vector Table Description

Number	Address	Name	Description
1	0F3C0H	Console Status	If a character is ready, the console status returns 0FFH in register A. If not, 00H is returned.
2	0F3C2H	Console Input vector 1	Reads a character from the console into the A register with the high-order bit clear.
3	0F3C4H	Console Input vector 2	Same as Console Input vector 1.
4	0F3C6H	Console Output vector 1	Sends the ASCII character in register C to the console device.
5	0F3C8H	Console Output vector 2	Same as Console Output vector 1.
6	0F3CAH	Reader Input vector 1	Reads a character from the paper tape reader device into register A.
7	0F3CCH	Reader Input vector 2	Same as Reader Input vector 1.
8	0F3CEH	Punch Output vector 1	Sends the character in register C to the paper tape punch device.
9	0F3D0H	Punch Output vector 2	Same as Punch Output vector 1.
10	0F3D2H	List Output vector 1	Sends the character in register C to the line printer device.
11	0F3D4H	List Output vector 2	Same as List Output vector 1.

Note

If a Console Output vector is specified, the B register will contain a number corresponding to a screen function output. (The B register contains zero during normal character output.) The B register will also contain a non-zero number during the output of the address cursor function (X and Y coordinates) after executing screen function number 7.

Adding I/O Software to the User Patch Areas

To add I/O software to the user patch areas, you must first create an executable COM file with the ASM and LOAD programs. Then, load the file into the patch area with the CONFIGIO program. CONFIGIO will also save the changes to a CP/M system disk.

When creating the COM file, the first 11 bytes of the actual routine must be in the format shown in Table 6.5. Only one patch routine can be written into a patch area per COM file. You can use as many vectors in the I/O Vector Table as desired. Examples of patch routines are given in "Substitution I/O Routine Example," and "Filter I/O Routine Example," at the end of this section.

Table 6.5.

Format for User-Written Patch Routines

Byte	Contents
1	The number of patches to I/O Vector Table to be made.
2 and 3	The destination address of the patch routine.
4 and 5	The length of the routine.
6*	Vector patch type which is either type 1 or type 2. 1 = substitution patch 2 = filter patch
7*	The vector number (1—11) to be patched.
8 and 9*	If the routine is a type 1 patch, bytes 8 and 9 contain the address to be patched into the vector. The address points to the user's code. If the routine is a type 2 patch, bytes 8 and 9 contain the address where the current contents of the specified vector are placed. (This can be the address field of a JMP instruction, etc.)
10 and 11*	The new address to be placed in the specified vector.

* Bytes 1 through 5 are repeated for each I/O vector patch made. If there is more than one patch made, then bytes 7 through 11 will be offset by the number of times you repeat bytes 1 through 5.

The actual program code follows the patch information described in Table 6.5. Conversion restricts the size of the program code to 64 bytes per slot-dependent patch area. Use the patch area appropriate for your application and slot use. (See Table 6.3 for more information on user patch areas.)

Steps for Adding I/O Software to the User Patch Areas

If the software already exists in a disk file, start the procedure at step 3. If you are entering a program from the keyboard, continue with the next step.

1. Use the DDT "S" command to enter the program into memory at location 100H.

2. Save the program with the SAVE command by typing:

   SAVE 1 *filespec*

   and then pressing the RETURN key to execute the command.

3. Run the CONFIGIO program. When the main menu appears, press the *3* key. The program will display:

   + + LOAD USER I/O DRIVER SOFTWARE + +

   SOURCE FILE NAME?

4. Type the *filespec* of the file containing the I/O software and press the RETURN key. The program will display

   LOADING...

   as it loads the routines from file into the user patch area. After the routines are loaded, the program returns to the main CONFIGIO menu.

5. Press the *4* key. CONFIGIO will display:

```
+ READ/WRITE I/O CHANGES MADE +

READ OR WRITE (R/W)?
```

6. Press the *W* key to write the I/O software into the BIOS in memory. The program will display:

```
DESTINATION DRIVE (A:-D:)?
```

   To save the changes to the BIOS on the system disk that is in active drive, press the *A* key. For any other CP/M system disk, go to step 4.

7. Insert a CP/M system disk into the appropriate drive. Type the corresponding drive letter and press the RETURN key. The BIOS on the disk is replaced with the one currently in memory. When the BIOS is copied into the CP/M system tracks on the destination disk, the program returns to the main CONFIGIO menu.

Note

Pressing the *R* key in step 6 permits the BIOS to be read from the CP/M system disk and loaded into memory. When the operation is complete, the program returns to the I/O Configuration Program menu.

Substitution I/O Routine Example

```
;
; Substitution routine
;
; This is a substitution routine for a second printer installed in the slot defined by "SLOT".
; This routine assumes there is a CCS 7710 interface board installed in SLOT and that all
; protocol is done elsewhere.
;
; SLOT is the value used to build the addresses used for status and data for the CCS 7710.
; Change it to whatever slot the CCS board is in.
;
; Miscellaneous definitions:
;
0002 =          SLOT    EQU   2
C0A0=           STAT    EQU   0C080H+(SLOT SHL 4)
C0A1=           DATA    EQU   STAT+1
FE00 =          DEST    EQU   0FE00H
;
; 6502 Subroutine call definitions
;
0040 =          X6502   EQU   40H          ;6502 transfer address
0045 =          AREG    EQU   45H          ;6502 register A pass area
004A =          ADDR    EQU   4AH          ;Address to PEEK or POKE
0049 =          CMD     EQU   49H          ;Command pass area
0002 =          PEEK    EQU   2            ;The PEEK command
0003 =          POKE    EQU   3            ;The POKE command
;
                OFFSET  SET   DEST-UL1
0100                    ORG   100H
;
; Information block for CONFIGIO
;
0100 01                 DB    1            ;Type 1 patch
0101 00FE               DW    DEST         ;Tells CONFIGIO where to put it
0103 2900               DW    LENGTH       ;How many bytes to store
0105 01                 DB    1            ;Type 1 patch
0106 0B                 DB    11           ;Patch list out vector 2 (UL1 :)
0107 00FE               DW    DEST         ;Address to patch into vector
                                           ;table
;
; The actual driver
;
0109 3E02       UL1:    MVI   A,PEEK       ;Do a PEEK command
010B 324900             STA   CMD          ;Store the command
010E 21A0C0             LXI   H,STAT       ;This is the address we want to
                                           ;read
0111 224A00             SHLD  ADDR         ;Store the address
0114 CD4000             CALL  X6502        ;Go read the 6502 memory
                                           ;location
0117 3A4500             LDA   AREG         ;Get the result
011A E602               ANI   2            ;Mask status bit
;
```

```
011C CA00FE              JZ     UL1+OFFSET      ;If zero, then character not ready
                                                ;to send
011F 79                  MOV    A,C             ;Get the character
0120 324500              STA    AREG            ;Store it for the 6502
0123 3E03                MVI    A,POKE          ;POKE this byte
0125 324900              STA    CMD             ;Store the command
0128 21A1C0              LXI    H,DATA          ;This is the address we want to
                                                ;write
012B 224A00              SHLD   ADDR            ;Store the address
012E CD4000              CALL   X6502           ;Go POKE the output byte
0131 C9          RET                           ;And go home
;
0029 =           LENGTH  EQU    $-UL1
0132                     END
```

Filter I/O Routine Example

```
;
; nolf - eliminate extra linefeeds
;
; This program removes linefeeds from a CR-LF sequence sent to the printer.
;
0100                     ORG    100H
                 ORIGIN  SET    0FE3FH-LENGTH   ;Origin for ASM
                 OFFSET  SET    ORIGIN-NOLF     ;True origin:end
                                                ;
0100 01                  DB     1               ;Number of patches
0101 2BFE                DW     ORIGIN          ;Place to put code
0103 1400                DW     LENGTH          ;Length of code
0105 02                  DB     2               ;Type of patch
0106 0A                  DB     10              ;Vector to change
0107 3DFE                DW     NOTCR+OFFSET+1  ;Place to put old vector contents
0109 2BFE                DW     ORIGIN          ;New vector contents
                                                ;
010B 00          CRFLAG  DB     0               ;Last character to pass
                                                ;through
                                                ;
010C 212AFE      NOLF:   LXI    H,CRFLAG+OFFSET ;Point to crflag
010F 7E                  MOV    A,M             ;Get last character
0110 71                  MOV    M,C             ;Save current character
0111 FE0D                CPI    13              ;Last char a cr?
0113 C23CFE              JNZ    NOTCR+OFFSET    ;No...just pass through
0116 79                  MOV    A,C             ;Get current character...
0117 FE0A                CPI    10              ;Is it a line feed?
0119 C8                  RZ                     ;Yes...don't print it
011A FE0D                CPI    13              ;Is it another cr?
011C C8                  RZ                     ;Yes...don't print it
011D C30000      NOTCR:  JMP    0000            ;This address patched
                                                ;by CONFIGIO
;
0014 =           LENGTH  EQU    $-NOLF
0120                     END
```

I/O Device Protocols for Assembly Language Programs

The I/O device protocol is similar to that supported by Apple Pascal, release 1.0, which requires the installation of accessory board types in specific accessory slots. Table 6.5 shows the required slot assignments. In addition to the standard Apple I/O devices, the SoftCard implementation of CP/M supports many other I/O devices.

Note

Contact your dealer or Microsoft Corporation to determine which I/O devices are compatible with the SoftCard II system.

Table 6.6.

Accessory Slot Addresses and Assignments

Slot	Acceptable Board Type	Slot Address
1	2,3,4,6	C100H—C1FFH
2	2,3,4,6 (input) 1,2,3,4,6 (output)	C200H—C2FFH
3	2,3,4,6	C300H—C3FFH
4	Any type	C400H—C4FFH
5	2	C500H—C5FFH
6	2	C600H—C6FFH
7	Any type	C700H—C7FFH

Type 1 is an unknown board type.

Type 2 is Apple][Disk Controller.

Type 3 is an Apple Communications Interface board or California Computer Systems® 7710A™ Serial Interface board.

Type 4 is an Apple High-Speed Serial Interface board or Apple Silentype® interface board.

Type 5 is an Apple Parallel Printer Interface board.

Type 6 is an Apple Firmware board.

Slots Type Table

The programmer may access the Slots Type Table from within a program. The table is located at addresses F3B9H through F3BFH.

When CP/M is loaded into memory with a cold start, each of the Apple I/O accessory slots is checked to see if a standard Apple peripheral board is installed. This is done by checking to see if there is ROM present in the slot-dependent address allocated for accessory board ROMs and then comparing two signature bytes to those of the standard Apple I/O peripheral boards.

This information is then stored in the Slots Type Table, which is located at 3B8H in the I/O Configuration Block. There are seven bytes in the Slots Type Table, each byte corresponding to the seven slots from one to seven. The value of a table entry may range from zero to six. The meaning of each value is as follows:

Value	Explanation
0	No peripheral board ROM was detected which usually means that no board is installed in the slot.
1	A peripheral board ROM was detected, but it is of an unknown type.
2	An Apple Disk][Controller board is installed in the slot.
3	An Apple Communications Interface board or CCS 7710A Serial Interface board is installed in the slot.
4	An Apple High-Speed Serial Interface, Videx Videoterm, M&R Sup'R Terminal or Apple Silentype printer interface is installed in the slot.
5	An Apple Parallel Printer Interface is installed in the slot.
6	An Apple Firmware Card is installed in the slot.

Disk Drive Byte

Disk drive byte is a single byte for monitoring the number of disk drives in the system. The byte is equal to the number of disk controller boards in the system multiplied by two. This value does not reflect an odd number of disk drives such as only one drive connected to a controller board. The disk drive byte is located at Z80 address F38BH.

Appendix A
Error Messages

This appendix lists in alphabetical order the error messages for the SoftCard implementation of CP/M. Each error message includes the possible causes and what actions you may take in response to them. In each of the error message descriptions, *d*: represents the disk drive identifier (A:-D:).

Aborted

Cause. PIP stopped; a key was pressed by the user.

Action. Retry the command.

Bad Delimiter

Cause. The wrong delimiting character was used in the STAT command line.

Action. Check for typing errors and try the command again.

BDOS ERR ON *d*: Bad Sector

Cause. An attempt was made to execute a command (built-in or transient) when:

There was no disk in the specified drive

The drive door was not closed

The disk was inserted into the specified drive improperly

The drive was not connected to a disk controller board (see SELECT error)

The disk was damaged or worn

Action. Correct the error condition. Then press the CONTROL-C keys to perform a warm start. Retry the command.

BDOS ERR ON *d*: File R/O

Cause. A write operation was attempted to a file that has a Read Only (R/O) attribute.

Action. Type any character to perform a warm start and return to CP/M command level.

BDOS ERR ON *d*: R/O

Cause. The disk in the accessed drive was changed without pressing CONTROL-C; or there is a write-protect tab on the disk.

Action. Press any key to perform a warm start and return to CP/M command level.

BDOS ERR ON *d*: Select

Cause. An attempt was made to access a disk drive when either the drive was not connected to a controller, or the controller has been installed in the wrong accessory slot.

Note that if you have only one drive attached to a disk controller board, an attempt to access a drive that is not installed results in a BAD SECTOR error instead of a SELECT error.

Action. Press any key to perform a warm start and return to CP/M command level.

Cannot Close Destination File-*filespec*

Cause. The output file specified in the PIP command line cannot be closed. This is usually caused by a write-protect tab on the disk.

Action. Remove the write-protect tab from the disk and try the command again.

Cannot Close File

Cause. A write operation has been attempted to a disk that has a write-protect tab on it.

Action. Make sure the disk is not write-protected and try the command again.

Cannot Read

Cause. The PIP program cannot read the source device specified in the command line.

Action. Check the RDR: device assignment and the physical connections to the current reader device.

Cannot Write

Cause. An invalid destination device was specified in the PIP command line.

Action. Check the device assignments and retry the command.

Checksum Error

Cause. PIP encountered a hex checksum record error while copying a hex file.

Action. Recreate the hex file with an assembler program and retry the command.

Checksum Error Load Address ...

Cause. The file specified in the LOAD command line contains errors.

Action. Recreate the hex file with an assembler program and retry the command.

Command Buffer Overflow

Cause. There are too many characters in the SUBMIT command buffer.

Action. Make sure that the submit input file doesn't exceed 2048 characters.

Command Error

Cause. Either there is a syntax error in the command line or the command is not understood (i.e., the arguments in the command line were not recognized by the program). Command Error is generated by utility programs written by Microsoft.

Action. Retype the command line in the correct format and retry the command.

Command Too Long

Cause. A command in the submit input file is longer than 125 characters.

Action. Check the commands in the submit input file and re-submit the input file.

Correct Error, Type Return or CTRL-Z

Cause. A hex record checksum error was encountered during the transfer of a hex file by PIP.

Action. Correct the hex file and retry the command.

Destination is R/O, Delete (y/n)

Cause. The destination file specified in the PIP command line is designated R/O.

Action. Enter *Y* to delete the existing file and PIP will complete the copy process. Enter *N* to abort the copy process.

Directory Full

Cause. An attempt was made to copy files to a destination disk which has no more storage space.

Action. Insert another disk in the destination drive and retry the command.

Disk Full

Cause. An attempt was made to copy files to a destination disk which has no more storage space. This error message is generated by the APDOS or MFT programs.

Action. Insert another disk in the destination drive and retry the command.

Disk I/O Error

Cause. The COPY program cannot format the disk. This is caused by either a bad or a worn-out disk, or the disk drive door is not closed.

Action. Ensure that the disk drive door is closed. If the same error message appears, replace the disk.

Disk Read Error

Cause. The source file specified in the PIP command contains an end-of-file character in the wrong place.

Action. Make sure that the end-of-file character is in the right place.

Disk Write Error

Cause. A disk write operation was attempted to a full disk.

Action. Either erase some files or try the write operation with another disk.

Disk Write-Protected

Cause. An attempt was made to write to a disk that has a write-protect tab on it. This error message is generated by the APDOS and COPY programs.

Action. Remove the write-protect tab from the disk and retry the command.

Error:Bad Parameter

Cause. There is an illegal parameter in the PIP command line.

Action. Check the command line and retry the command.

Error:Cannot Close File, Load Address *xxxx*

Cause. An error exists in the program being loaded with the LOAD program. The disk may be write-protected.

Action. Check the source program for errors. Check the disk for a write-protect tab.

Error:Cannot Open Source, Load Address *xxxx*

Cause. The LOAD program cannot find the file specified in the LOAD command line; or no filename was specified.

Action. Check the filename of the source file to be loaded. Make sure the filename is included in the LOAD command line. Retry the command.

Error:Disk Inverted, Load Address *xxxx*

Cause. The address of a record was too far from the address of the previously processed record.

Action. Use DDT to read the hex file into memory, then use the SAVE command to store the file back to the disk. Retry the LOAD command.

Error:Disk No More Directory Space, Load Address *xxxx*

Cause. The destination disk in the active drive is full.

Action. Change disks and retry the command.

Error:Disk Read, Load Address *xxxx*

Cause. The file specified in the LOAD command line cannot be found on the disk.

Action. Check to see that the file exists on the disk in the active drive.

Error:Disk Write, Load Address *xxxx*

Cause. The destination disk in the active drive is full.

Action. Change disks and retry the command.

Error On Line *nnn*

Cause. There is an error in the submit input file at the specified line number (*nnn*).

Action. Correct the error and retry the command.

File Error

Cause. The disk is full and the ED program cannot write any more data to the accessed file.

Action. Copy the file to another disk or delete other files from the same disk.

File Exists

Cause. An attempt was made to change the name of a file to an existing filename.

Action. Make sure the "new" filespec argument in the REN command line does not match any existing filenames on the same disk. Retry the command.

File Exists, Erase It

Cause. The destination file named in the ED command line already exists.

Action. Place the destination file on another disk or in a different user area.

File Is Read Only

Cause. The file specified in the ED command line has an R/O attribute.

Action. Change the file status with the STAT program.

File Not Found

Cause. The source file specified in the APDOS, AUTORUN, CAT, COPY, MFT, PATCH, or STAT command line does not exist.

Action. Check the spelling of the filename and reenter the command line.

Filename?

Cause. An incorrect use of wild card characters in the REN command line.

Action. Retry the command with no wild card characters in the command line.

Filename Required

Cause. ED was invoked without a filename argument in the command line.

Action. Include a filename in the ED command line.

hhhh?? = *dd*

Cause. The mnemonic (*dd*) at address (*hhhh*??) is not an 8080 or Z80 assembly language instruction.

Action. Correct the mnemonic.

Insufficient Memory

Cause. There is not enough memory available to load the specified file with the DDT R or E command.

Action. Reduce the size of the file and load in segments of the file.

Invalid Assignment

Cause. One of the device names specified in the STAT command line is either misspelled or cannot be assigned to the other specified device.

Action. Check the spelling of the device name and retry the command. If the same error message appears again, check for the valid device assignments by typing *STAT VAL:*.

Invalid Control Character

Cause. An invalid CONTROL character was included in a submit input file.

Action. Use only ^A through ^Z CONTROL characters in a submit input file.

Invalid Digit

Cause. The hex file specified in the PIP command line contains an invalid hex digit.

Action. Correct the hex file and retry the PIP command.

Invalid Disk Assignment

Cause. An attempt was made to assign an attribute other than R/O to a disk drive with the STAT program.

Action. Assign only the R/O attribute to disk drives. Remove the R/O attribute with the STAT program.

Invalid Disk Select

Cause. A command line specified a nonexistent disk drive.

Action. Specify only disk drives A: through D: in the command line. Check for any loose connections to the disk drives and for unformatted disks.

Invalid File Indicator

Cause. STAT did not recognize the attribute in the STAT command line.

Action. Specify only R/O, R/W, DIR, or SYS in the STAT command line.

Invalid Format

Cause. The PIP command line was in the wrong format.

Action. Check the command line format and retry the command.

Invalid Hex Digit ...

Cause. The file specified in the LOAD command line contains an incorrect hex digit.

Action. Correct the file and retry the command.

Invalid Separator

Cause. An invalid separator character was used in the PIP command line.

Action. Check the command line format and retry the command.

Invalid User Number

Cause. An invalid user number was specified in the PIP command line.

Action. Use only user numbers 1 through 15.

n?

Cause. A number greater than 15 was specified in the USER command line.

Action. Use only user numbers 1 through 15.

No Directory Space

Cause. There is no room on the disk for the .PRN and .HEX files generated by ASM.

Action. Either delete files from the active drive or specify another drive.

No File *filespec*

Cause. The file specified in the command line cannot be found.

Action. Recheck the spelling of the filespec and try again.

No File(s) Found, *xxx*k Bytes Available

Cause. The file specified in the CAT command line does not exist.

Action. Check the spelling of the filename and reenter the command line.

No Input File Present On Disk

Cause. The file specified in the DUMP command line does not exist.

Action. Recheck the spelling of the filespec and try again.

No Source File Present

Cause. The ASM assembler could not find the file specified in the command line.

Action. Check spelling of the file and ensure that the disk is listed in the disk directory. Retry the command.

No Space

Cause. An attempt was made to save the contents of memory with the SAVE command, but there is no space left on the disk.

Action. Use a disk with sufficient storage space and retry the command.

No Sub File Present

Cause. The SUBMIT program was run but no submit input file was specified.

Action. Create a submit input file.

Not A Character Source

Cause. An invalid source device was specified in the PIP command line.

Action. Use either RDR: or CON: as source devices.

Not Deleted

Cause. The file specified in the PIP command line has an R/O attribute and cannot be deleted.

Action. Change the status of the file with the STAT program.

Not Found

Cause. PIP cannot find the file specified in the command line.

Action. Check the spelling of the file and try again.

Output File Write Error

Cause. Either a file with write-protect status has been specified as the ASM destination file, or there is no free space left on the disk.

Action. Check the attributes of the destination file and the amount of free disk space with the STAT program.

Parameter Error

Cause. The submit input file contains an invalid parameter.

Action. Use only valid parameters ($0 through $9) in the submit input file.

Quit Not Found

Cause. The *Q* parameter was specified in the PIP command line but there is no string argument in the input file.

Action. Insert the appropriate string argument in the input file specified by PIP.

Read Error

Cause. The file specified in the TYPE command line contains an error.

Action. Use the STAT program to check the disk and the file. Retry the command.

Reader Stopping

Cause. The read operation from the RDR: device has been interrupted. (A key was pressed during the read operation.)

Action. Retry the command.

Record Too Long

Cause. The file or device specified in the PIP command line contains a record longer than 128 bytes.

Action. Use the STAT program to check for any records longer than 128 bytes.

Source File Name Error

Cause. Wild card characters * and ? were specified in the source filename argument of the ASM command line.

Action. Specify only one source filename in the ASM command line.

Source File Read Error

Cause. The file read by the ASM assembler is in the wrong format or has instructions the ASM assembler cannot understand.

Action. Check the file for the proper format and check that the assembly language instructions are 8080 mnemonics.

Start Not Found

Cause. PIP cannot find the string argument in the input file specified by the S parameter.

Action. Check the input file for the appropriate string argument.

Too Many Files

Cause. STAT cannot process the files specified. Either there are too many files (more than 512), or there is not enough free RAM available to process the files.

Action. Delete or transfer files to another disk. Retry the command and specify fewer files.

Unexpected End Of Hex File *filespec*

Cause. The hex file specified in the PIP command line contains an end-of-file character before a termination hex record.

Action. Correct the hex file and retry the command.

Unrecognized Destination

Cause. PIP did not recognize the destination file or device specified in the command line.

Action. Make sure that the destination device is a currently assigned device, or that the destination file exists.

Use: STAT *d*:=R/O

Cause. The drive argument (*d*:) in the STAT command line was used in the wrong format.

Action. Use the proper format (STAT *d*:=R/O) for the drive argument.

Verify Error

Cause. The data copied onto a destination disk does not match the data on the source disk. This is caused by a worn or damaged disk, or by hardware problems. The VERIFY ERROR message is generated by the COPY and PATCH transient programs.

Action. Try using a different disk and repeat the command. If the same error message appears again, check the connections to the disk drives and the disk controllers. If there is a hardware problem, contact your dealer.

? (Syntax error)

Cause. The command was not understood. Either the command was mistyped, or invalid arguments were included in the command line.

Action. Retype the command line in the correct format.

Appendix B
SoftCard Version Differences

This appendix describes the differences between the SoftCard II system and earlier ones.

SoftCard II Enhancements

Because of the Apple I/O interface and dual microprocessor environment, the SoftCard implementation of CP/M has the following enhancements:

Patch areas in the BIOS for adding user-written I/O driver software

A screen function interface for modifying the screen attributes for a specific terminal or program

A character redefinition table for redefining the ASCII characters produced by the keys

A type-ahead buffer for keyboard entry while CP/M is performing other operations

A print buffer that allows the printing of a file while performing other operations

Up to 58.5K bytes of memory for application programs

CP/M Implementation Differences

The SoftCard version does not include the MOVCPM or the SYSGEN utilities. Because the SoftCard implementation of CP/M is a "fixed" size, and SoftCard COPY program allows you to put CP/M onto another disk, there is no need for either utility.

SoftCard Differences

The SoftCard II system has several features that the previous SoftCards do not have. There are also differences in the way the SoftCard II system performs I/O functions.

The following features are unique to the SoftCard II and the Premium SoftCard IIe.

- A type-ahead buffer for keyboard entry while CP/M performs other operations
- A print buffer that allows printing of a file while CP/M performs other operations

Differences in Hardware

The SoftCard II circuit board contains a Z80B microprocessor which operates three times as fast as those for previous SoftCard circuit boards. The Z80B is not synchronized or phase-locked to the Apple *//e* internal clocks.

The SoftCard II circuit board can be installed in any accessory slot.

There are no switches on the SoftCard II circuit board.

Differences in Software

The SoftCard II has a larger TPA (58.5K bytes) for running programs.

Because all memory is contained on the SoftCard II circuit board, there is no need to change the size of CP/M. Therefore, the SoftCard II package does not include the CPM60 utility program.

There is no I/O Configuration Block (IOCB). Some of the SoftCard II I/O configuration tables and routines are located in different areas of the BIOS and not in a contiguous block.

The screen menus in the CONFIGIO utility program have been changed.

The Microsoft BASIC Interpreter has been condensed into one file (GBASIC.COM). High-resolution graphic commands are available whenever BASIC is running.

Because of the Z80B microprocessor, programs running under the SoftCard II version of CP/M execute three times faster than programs running under previous SoftCards.

Differences in I/O Operation

The SoftCard II uses a method of accessing the I/O system that differs from the previous SoftCards. The SoftCard Z80 microprocessor uses the 6502 as an I/O processor and the Apple memory for I/O communications. Therefore, it is not possible with the SoftCard II version of CP/M to directly access Apple I/O memory locations.

Note

The previous SoftCards access I/O functions directly through memory-mapped locations in the Apple's memory and do not use the 6502 except to call 6502 subroutines.

The SoftCard II calls 6502 routines differently than previous SoftCards. The Z80 performs I/O operations through the 6502 microprocessor by accessing a program called the "6502 Basic Input/Output System," or 6502 BIOS. There are 15 separate functions. All are accessed by storing information in a seven-byte area located at 45—4B, and then performing a Z80 CALL instruction to memory location 40. Information from the I/O system is returned to the same seven-byte area.

Appendix C
80-Column Operation and the SoftCard II

This appendix explains what you should know about using the SoftCard II system with the 80-column video display for Apple][(and][Plus) and earlier Apple *//e* computers with no built-in 80-column display boards. If you have an Apple *//e* with built-in 80-column display features, you may disregard this appendix.

Apple *//e* Computers With No 80-Column Video Display Boards

If you have an Apple *//e* with no 80-column video display board, we recommend using the Apple 80-Column Text Card or Extended 80-Column Text Card. The SoftCard may be installed in accessory slot 3. Otherwise, SoftCard operation is the same.

Apple][and][Plus Computers

80-Column Video Output

The Apple][and][Plus computers cannot display 80-columns on a terminal screen unless you install an 80-column display board. Because some display boards use a separate output jack for 80-column output, you may have a problem sending graphic output to the screen.

To solve this problem, you may have to physically switch the output video jacks or use a "soft switch" as the one provided by the Videx Videoterm board. (A soft switch permits you to switch outputs through software commands.)

Keyboard Character Redefinition

The Apple][keyboard cannot generate certain characters without hardware modifications. The SoftCard II system compensates for this condition by allowing you to redefine the output of up to six keys with the CONFIGIO utility program. (See "Redefining Keyboard Characters With CONFIGIO" in Chapter 6.) Before redefining any of the keys, check to see if any character definitions are being made by your application program or by the 80-column video board you installed. The technical manual for the product should tell you if it does.

Special Note for Videx Videoterm Display Board Users

There are two characters the Videoterm display board cannot use as input: CONTROL-A and CONTROL-K.

The Videoterm board uses CONTROL-A to switch between uppercase and lowercase characters. Because the board uses CONTROL-A internally, it cannot directly generate this character for application programs. If your application program requires CONTROL-A for input, use CONFIGIO to redefine another character as CONTROL-A.

Because CONTROL-K is translated by the Videoterm board as a left square bracket ([), CP/M will display a "[" whenever you type CONTROL-K. If your program requires a CONTROL-K as input, use CONFIGIO to translate the "[" back to CONTROL-K. In this way, when you type CONTROL-K, you will receive a CONTROL-K, even though it requires two separate stages of translation.

Appendix D
CP/M ProFile

This appendix will show how to install and use CP/M on the Apple ProFile hard disk drive. To install CP/M on the ProFile, follow these steps:

1. Set up the ProFile and format it.

2. Allocate space on the ProFile by creating a Pascal area and hard disk volume.

3. Copy ProDOS and CP/M to the ProFile. (ProFile will not work properly without ProDOS.)

Appendix D is organized for both new and experienced ProFile users. If you are a new user, we recommend going through all the procedures listed in this section. If you are an experienced ProFile user, follow the procedures that apply to your situation. For example, if you already have formatted the ProFile and have created a Pascal area, skip the first two procedures and continue with the third.

What You Need to Install CP/M on Your ProFile

In addition to the SoftCard circuit board and the items listed in the *ProFile Owner's Manual*, you will need the following disks:

- Apple PPM Startup disk
- Apple PPM Program disk
- Apple ProDOS User's disk
- Microsoft SoftCard ProFile disk
- Microsoft SoftCard Master disk

Before you start the following procedure, make sure that your ProFile is set up and operational as described on pages 1 through 16 in the *ProFile Owner's Manual*. For systems that have only one floppy disk controller card, install the ProFile interface card in slot 5. If you have two floppy disk controller cards in slots 5 and 6, install the ProFile interface card in slot 4. You may have to move the SoftCard to a different slot to accomplish this. Table 2.1 in Chapter 2 of the *Installation and Operation Manual* lists the slots available for the SoftCard.

Note

The following procedures assume that you have only one floppy disk drive. Certain steps ask you to "swap disks" (remove one disk and insert another). If you have more than one drive, you may use the additional drives instead of swapping disks. The software will automatically search all drives until it finds the file you specified.

Formatting the ProFile

Formatting not only prepares the ProFile to receive information but also deletes all data previously stored on it. If you already formatted the ProFile and have data that you don't want to lose, skip steps 4 through 11.

1. Turn on the ProFile power switch, and wait for the light on the front of the drive to stop blinking.

2. Insert the ProDOS User's disk into drive A.

3. Turn on the video monitor and the computer. When ProDOS is loaded into memory, the screen displays the ProDOS Main menu as shown in the figure below:

```
                    PRODOS USER'S DISK

                COPYRIGHT APPLE COMPUTER, INC. 1983

YOUR OPTIONS ARE:
        ? - TUTOR: PRODOS EXPLANATION
        F - PRODOS FILER (UTILITIES)
        C - DOS <-> PRODOS CONVERSION
        S - DISPLAY SLOT ASSIGNMENTS
        T - DISPLAY/SET TIME
        B - APPLESOFT BASIC
PLEASE SELECT ONE OF THE ABOVE
```

Figure D.1 ProDOS Main Menu

4. Enter *F* from the Main menu. The screen will display the ProDOS Filer menu.

5. Enter *V* from the Filer menu. The screen will display the Volume Commands menu.

6. Enter *F* (for Format a Volume), and the screen will display a message asking you to type in a slot number.

7. Enter the slot number the ProFile interface card is in (either slot 4 or 5). Another message on the screen will ask you for a new volume name.

8. To assign your ProFile a volume name, type */PROFILE* and press the RETURN key. You will see the message

   DESTROY "/PROFILE"?(Y/N)

   WARNING: YOU ARE ABOUT TO FORMAT A LARGE DISK.

9. Press the *Y* key to begin the format process. When the ProFile is formatted, you will see

   FORMAT COMPLETE

10. Press the ESC key twice to return to the ProDOS Systems Utilities Filer menu.

11. Quit the Filer menu by pressing the *Q* key and then the RETURN key. The display will show the ProDOS Main menu again.

The ProFile is now formatted and can be partitioned into different storage areas. The next task is to create a Pascal area on the ProFile.

Creating a Pascal Area on the ProFile

The next stage of transferring CP/M to your ProFile is creating a Pascal area on the ProFile and then a hard disk volume. A Pascal area is an area of the disk allocated for the Pascal operating system. Because CP/M uses a disk storage format similar to Pascal, the area allocated for CP/M must be within the Pascal area. A *hard disk volume* is a smaller storage unit within the Pascal area. Each hard disk unit is equivalent to a floppy disk, but of a variable size. Hard disk volumes are explained in the "Creating a Hard-Disk Volume" section of the *Pascal ProFile Manager Manual*.

The following steps will show you how to create a Pascal area on the ProFile, assuming that the ProFile is already formatted.

1. Insert the PPM Startup disk into floppy disk drive A (Apple drive 1).

2. Turn on the video monitor and the computer. When you see the message

```
Insert boot disk
with SYSTEM.PASCAL on it,
then press RETURN.
```

   remove the startup disk from drive A and insert the PPM Program disk in its place. Press the RETURN key.

Note

If you have two or more disk drives, you can leave the startup disk in drive A and insert the program disk into drive B.

---

3. When you see the message

   Assign volumes to their default number? (Y/N)

   type *N*. You will first see the message

   Loading Pascal ProFile Manager ....

   followed by the PPM menu display as shown in Figure D.2.

```
Pascal ProFile Manager     Copyright 1983 Apple Computer, Inc.     Version 1.0

Type the first letter to select an option. Option?

V(olume Manager program

X(tended Filer program

B(ackup program

C(reate a Pascal Area on the ProFile

D(elete Pascal Area from the ProFile

Q(uit
```

Figure D.2 Pascal ProFile Manager Display

4. Press the *C* key to create a Pascal area on the ProFile.

5. When you see the message

   Create a Pascal Area ... Create Pascal area on which drive?
   (Enter number.)

   Enter *0* and press the RETURN key. (Drive 0 is the ProFile.) After the PPM program creates the Pascal area, the screen will show the PPM Main menu display again. Continue with the next procedure "Creating the CP/M Hard Disk Volume."

Creating the CP/M Hard Disk Volume

The following steps create a hard disk volume for CP/M.

Note

This procedure assumes that you have just completed the previous procedure. If not, insert the PPM disk into drive A and turn on the computer. You should see the PPM Main menu.

---

1. Create a hard disk volume by entering a *V* from the PPM Main menu.

   When you press the *V* key, the display shown in Figure D.3 will appear.

```
Volume Manager        Copyright 1983 Apple Computer, Inc.        Version 1.0

A(ssign, R(elease, C(reate, D(elete, W(Prot, M(odify, K(runch, N(ext Q(uit

ProFile drive: 0

    WP Name Description Unit
```

Figure D.3 Pascal ProFile Volume Manager Display

2. Select the Create command by pressing the *C* key. The screen will display the message:

   Create a volume...What is the name of this volume?

3. Type

   CP/M

   and press the RETURN key. The program will respond with the message:

   What is the description field?

4. Type

   CAT

   and press the RETURN key. The description field contains the name of the CP/M file you wish to automatically execute when the system is booted. The CP/M CAT command displays a directory of files and the amount of volume space available. CAT is given as an example, but you may use other command filenames as well.

   When you press the RETURN key, the program displays the message:

   What is the size of this volume in blocks?

5. You may enter a value equal to [(64 * *x*) + 1], where *x* is any number from 1 to 48. The minimum size then is 65 and the maximum is 9473: (64 * 48) + 1 = 9473. To create a volume of the maximum size, type

   9473

   and press the RETURN key. Note that it will take several minutes for the ProFile to create a large volume.

   Now instead of sorting through a pile of floppy disks when you want to use a program, you will go to the Volume Manager to select the volume you want to use.

6. Press the *Q* key to quit the Volume Manager program. The PPM Main menu display will appear on the screen.

Copying ProDOS onto the ProFile

The following steps copy the ProDOS operating system onto the ProFile disk and software that allows the ProFile to communicate with the SoftCard. This procedure assumes that you have just completed the previous procedure. You should have a Pascal area with a hard disk volume named CP/M on the disk.

1. Insert the ProDOS User's disk into drive A.

2. Turn the computer's power switch on. When ProDOS is loaded into memory, you should see the ProDOS Main menu.

3. Enter *F* (for Filer menu) from the Main menu.

4. When you see the Filer menu display, press the *F* key. You should see a list of file commands.

5. Enter *C* from the list of file commands. The screen will display

```
--COPY--
   PATHNAME: (                    )

     TO PATHNAME:
```

6. Type

```
/USERS.DISK/=
```

and press the RETURN key.

7. Type

```
/PROFILE/=
```

and press the RETURN key. The screen will display the following:

```
--COPY--
   PATHNAME: /USERS.DISK/=

     TO PATHNAME: /PROFILE/=

--INSERT DISKS AND PRESS <RET>--
```

8. Remove the ProDOS User's disk and insert the SoftCard ProFile disk into drive A. Press the RETURN key.

9. Type

```
/MICROSOFT/SOFTCARD.SYSTEM
```

and press the RETURN key. The screen will display:

```
--COPY--
   PATHNAME: /MICROSOFT/SOFTCARD.SYSTEM

    TO PATHNAME: (                    )
```

10. Type

```
/PROFILE/SOFTCARD.SYSTEM
```

and press the RETURN key. The screen will display the following:

```
--COPY--
   PATHNAME: /MICROSOFT/SOFTCARD.SYSTEM

    TO PATHNAME: /PROFILE/SOFTCARD.SYSTEM

--INSERT DISKS AND PRESS <RET>--
```

11. Press the RETURN key. The screen will display the Filer menu again.

12. Exit the ProDOS Filer program by pressing the *Q* key.

13. Type

```
-/MICROSOFT/SOFTCARD.INIT
```

and press the RETURN key. This utility initializes the entire CP/M directory and performs other one-time housekeeping chores. The CP/M hard disk volume is ready for you to copy the CP/M operating system and programs onto the ProFile.

14. Type

```
-/MICROSOFT/SOFTCARD.SYSTEM
```

and press the RETURN key.

Copying CP/M Programs into the CP/M Hard Disk Volume

The following steps copy the SoftCard Master disk into the CP/M hard disk volume on the ProFile.

1. Insert the SoftCard Master disk into drive A.

2. Perform a cold start by turning the computer off and then on again.

3. When you see the CP/M A: prompt, type

   ```
   PIP C:=A:
   ```

   if the ProFile disk controller is in slot 5, or type

   ```
   PIP E:=A:
   ```

   if the ProFile disk controller is in slot 4.

4. Press the RETURN key to start the copy process. This should take about a minute to complete.

5. When you see the A> prompt again, change the active drive to the ProFile by typing either

   ```
   C:
   ```

   or

   ```
   E:
   ```

   depending on which slot the ProFile controller card is in. When you see the ProFile driver letter, use the CAT command to verify that all CP/M files from the master disk have been copied over.

This completes the last procedure for preparing the ProFile for CP/M.

How to Start CP/M from the ProFile

When you turn the computer on, it will automatically look for a ProDOS Startup file and execute ProDOS. When you see the ProDOS prompt, type

```
-SOFTCARD.SYSTEM
```

and press the RETURN key. If you used CAT in the description field, you should see a list of CP/M files from your SoftCard Master disk and a CP/M prompt (either C: or E: depending on which slot the disk controller card is in).

You may now use the CP/M operating system and CP/M application programs as described in the rest of the softcard documentation and the application program manuals.

Glossary

Access

An operation to obtain data from or place data into a storage device, register, or memory.

Accessory board

A printed circuit board installed in the accessory slots of the Apple *//e* computer. It usually performs as an interface to I/O devices.

Active drive

The disk drive that all disk file operations are performed from or to if no other drive is specified. Also called the currently logged drive.

Address

A number representing a location where information is stored or where an I/O device is located.

Alphanumeric

Characters which include the letters of the alphabet, numerals, and other symbols used for punctuation and mathematical operations.

Ambiguous filename

A filename containing wild card characters in the filename or in the filename extension. An ambiguous filename is used to refer to more than one file at a time.

ANSI

American National Standards Institute. An organization devoted to establishing industry standards for computing and data processing.

Argument

A user entry in the command line of a command or program statement. Also called option, user entry, or parameter.

Assembler

A program that translates symbolic assembly language into binary machine language for execution by the computer.

Backup disk

A disk that contains information copied from another disk. It is used in case the original disk is unintentionally altered or destroyed.

BDOS

Basic Disk Operating System. The CP/M system module that handles disk operations.

BIOS

Basic Input/Output System. The CP/M system module that handles communication with the computer's I/O system.

Block

A basic unit of disk space allocation used by CP/M. Each disk drive has a fixed block size defined in its disk parameter block in the BIOS. A block can consist of 1K, 2K, 4K, 8K, or 16K consecutive bytes. Blocks are numbered relative to zero.

Boot

The process of loading an operating system into memory. A boot (bootstrap loader) program is a small program that automatically executes when the power is applied to the computer. The boot program loads the rest of the operating system into memory.

Built-in commands

Commands that reside in the CCP module. They can be used at any time from CP/M command level.

Call

See system call.

Calling program

A program or software module (such as the CCP) running in the TPA that executes a system call.

Card

See printed circuit board.

CCP

Console Command Processor. The CP/M software module that handles operator communication.

Character position

A location on the screen where one character can be displayed.

Character set

All of the characters that can be displayed and entered from a terminal.

Command file (.COM file)

An executable program in machine language.

Command line

A command to the computer that consists of the command word and the arguments or parameters that modify the execution of the command.

CON:

Mnemonic for the logical console device.

Console

See terminal.

Control character

A character used with another character to perform a special operation. See "Line Editing Commands" in Chapter 5 for a list of control characters used with CP/M.

CP/M command level

A mode of operation where the CCP module controls the other CP/M system module and hence the computer. The command level mode of operation allows direct commands by the operator to the operating system. Command level is indicated by the CP/M A> prompt.

Data file

A file containing information that will be processed by a program.

Debug

The process of detecting, locating, and removing errors in a computer program. Programs such as DDT help perform this task.

Delimiter

A special character, such as a comma, that separates different items in a command line.

Destination disk/file

The disk or file that information is to be copied to by the COPY, PIP, or MFT program.

DOS

The mnemonic name for disk operating system.

Editor

A utility program that creates and modifies text files. It can also be used to create document files or code for programs.

Extent

A CP/M measurement unit (usually 16K consecutive bytes) for storing data in a file.

External terminal

Refers to a terminal connected to an interface board in accessory slot 3 of the Apple *//e* System. The external terminal replaces the Apple keyboard and screen monitor as the primary I/O device for operator input.

FDOS

An arbitrary area of memory consisting of the BDOS and BIOS software modules.

File specification

Also called filespec. A series of bytes that indicate the name, type, and location of a disk file.

HEX file

A printable form of a command (machine-language) file.

Instruction set

The list of all instructions which a given microprocessor will understand and execute.

I/O

Input/output. The transfer of data into and out from a computer and its peripheral devices.

I/O Bus

The communication circuits between the Apple CPU and the other components of the computer system.

I/O devices

The hardware devices of a computer system used to enter data into the computer, such as a keyboard; or to accept data from the computer, such as a printer.

I/O port

A register or set of registers used by the CPU for input or output of data to and from the I/O system.

I/O system

I/O devices such as printers, terminals, modems, etc., and the necessary interface circuits that permit communication with a computer.

Lead-in character

A character used by the computer to denote the beginning of a special function or routine.

Line editing

In CP/M, the act of editing the current command line.

List device

The I/O device (usually a printer) on which data can be listed or printed. LST: is the name of the logical list device.

Loader

A utility program that loads a machine-executable program into memory.

Logical device

The software representation of the actual physical I/O devices that the computer can communicate with.

LST:

The logical list device name.

Master disk

The disk that comes with the SoftCard package containing the CP/M operating system and all the software that is part of the SoftCard package.

Mode

A certain way of performing tasks. For example, a computer receives data from an I/O unit in either synchronous or asynchronous mode. In asynchronous mode, data is sent serially with no synchronization between the I/O unit and the computer. In synchronous mode, data is sent in synchronization with the computer's clock frequency.

Module

A set of routines and subroutines organized into a logical unit and designed to work with other software modules. In CP/M, there are three software modules: the CCP, BIOS, and BDOS.

MP/M

Multi-Programming Monitor control program. The multi-user version of CP/M.

Object code

Executable binary code (the output code of an assembler).

Object program

A source program that has been translated into object code that can be executed or "run" without any additional preparation.

Option

See argument.

Page

256 consecutive bytes in memory.

Parameter area

The memory area between addresses 000 and 0100. Used to hold important system parameters.

PATCH

A short section of program code that replaces a section of another program to correct errors, make changes, or supply additional data.

Peripheral devices

See I/O devices.

Physical device

A vector location in the BIOS module that points to a specific assembly language routine for I/O communication.

Port

On a terminal or computer, the physical connection facilities (i.e., sockets, connectors and cables) to an I/O device.

Printed circuit board

Interface circuits contained on a board that plugs into the Apple *//e* accessory slots for interfacing to an I/O device, additional memory, or a coprocessor.

Program-dependent

Input and output devices whose functions can be defined by a computer program.

Program level

When the operation of the computer is controlled by a program (such as GBASIC) running in memory. Any commands given by the operator are processed by the program instead of by the operating system's command module (in CP/M, the CCP). For example, when GBASIC is running, all commands are processed by the BASIC Interpreter. The resulting actions are passed to the BIOS and BDOS modules by GBASIC. The program level mode of operation is usually indicated by the program's prompt: in the case of GBASIC, the word "Ok."

Prompt

Instructions or symbols displayed on the screen to indicate what the operator should do next.

PUN:

The logical punch device name.

RDR:

The logical reader device name.

Read only (R/O)

An attribute that can be assigned to a disk or disk file. When assigned to a file or a disk, it does not allow changes to be made on the file or disk.

Read/write (R/W)

An attribute that can be assigned to a disk or disk file. It allows both read and write operations.

Record

A group of 128 bytes in a disk file.

Source file

The original file (usually an ASCII text file) in which a program is prepared prior to processing by the computer.

Spooling

The process of accumulating printer output in a file while the printer is busy. The file is printed when the printer becomes free.

System call

A request from a program or from the CCP to an assembly language routine that performs a low-level function such as displaying a character on the screen. In CP/M, the assembly language routines are stored in the BDOS module and are identified by numbers.

System tracks

The tracks on the disk reserved for the CP/M system.

Terminal

An input/output device; a terminal usually has a keyboard and monitor for entering and displaying data.

TPA

Transient Program Area: the area of memory where user programs are loaded and run.

Track

A separate recording path on a magnetic tape or disk.

Utility program

A program that enables the user to perform certain operations, such as copying disks.

Vector

A location in memory that "points" to a subroutine or another memory address.

Index

MICROSOFT®
10700 Northup Way, Bellevue, WA 98004

Software Problem Report

Name _____

Street _____

City _____ State _____ Zip _____

Phone _____ Date _____

Instructions

Use this form to report software bugs, documentation errors, or suggested enhancements. Mail the form to Microsoft.

Category

_____ Software Problem

_____ Software Enhancement

_____ Documentation Problem
(Document #_____)

_____ Other

Software Description

Microsoft Product _____

Rev. _____ Registration # _____

Operating System _____

Rev. _____ Supplier _____

Other Software Used _____

Rev. _____ Supplier _____

Hardware Description

Manufacturer _____ CPU _____ Memory _____ KB

Disk Size _____" Density: Sides:

Single _____ Single _____

Double _____ Double _____

Peripherals _____

Problem Description

Describe the problem. (Also describe how to reproduce it, and your diagnosis and suggested correction.) Attach a listing if available.

Microsoft Use Only

Tech Support _____ Date Received _____

Routing Code _____ Date Resolved _____

Report Number _____

Action Taken:

DIGITAL RESEARCH LICENSE INFORMATION

CAREFULLY READ ALL THE TERMS AND CONDITIONS OF THIS AGREEMENT PRIOR TO BREAKING THE DISKETTE SEAL. BREAKING THE SEAL INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS.

IMPORTANT: Our license with Digital Research for the CP/M® Operating System requires that each purchaser of the SoftCard™ with CP/M register with Microsoft® Corporation so that records can be maintained of all CP/M owners. This requirement is made by Digital Research, not by Microsoft, and a post-card is enclosed for reply. THE SERIAL NUMBER ON THE CARD IS THE NUMBER STAMPED ON THE DISK LABELS.

SOFTWARE LICENSE AGREEMENT

IMPORTANT: All Digital Research programs are sold only on the condition that the purchaser agrees to the following license. READ THIS LICENSE CAREFULLY. If you do not agree to the terms contained in this license, return the packaged diskette UNOPENED to your distributor and your purchase price will be refunded. If you agree to the terms contained in this license, fill out the REGISTRATION information and RETURN by mail to Microsoft Corporation.

DIGITAL RESEARCH agrees to grant and the Customer agrees to accept on the following terms and conditions nontransferable and nonexclusive license to use the software program(s) (Licensed Programs) herein delivered with this agreement.

1. TERM: This agreement is effective from the date of receipt of the above-referenced program(s) and shall remain in force until terminated by the Customer upon one month's prior written notice, or by Digital Research as provided below.

Any license under this Agreement may be discontinued by the Customer at any time upon one month's prior written notice. Digital Research may discontinue any license or terminate this Agreement if the Customer fails to comply with any of the terms and conditions of this Agreement.

2. LICENSE: Each program license granted under this Agreement authorizes the Customer to use the Licensed Program in any machine readable form on any single computer system (referred to as System). A separate license is required for each System on which the Licensed Program will be used.

This Agreement and any of the licenses, programs or materials to which it applies may not be assigned, sublicensed or otherwise transferred by the Customer without prior written consent from Digital Research. No right to print or copy, in whole or in part, the Licensed Programs is granted except as hereinafter expressly provided.

3. PERMISSION TO COPY OR MODIFY LICENSED PROGRAMS: The Customer shall not copy, in whole or in part, any Licensed Programs which are provided by Digital Research in printed form under this Agreement. Additional copies of printed materials may be acquired from Digital Research.

Any Licensed Programs which are provided by Digital Research in machine readable form may be copied, in whole or in part, in printed or machine readable form in sufficient number for use by the Customer with the designated System, to understand the contents of such machine readable material, to modify the Licensed Program as provided below, for back-up purposes, OR FOR ARCHIVE PURPOSES, provided, however, that no more than five (5) printed copies will be in existence under any license at any one time without prior written consent from Digital Research. The Customer agrees to maintain appropriate records of the number and location of all such copies of Licensed Programs. The original, and any copies of the Licensed Programs, in whole or in part, which are made by the Customer shall be the property of Digital Research. This does not imply, of course, that Digital Research owns the media on which the Licensed Programs are recorded. The Customer may modify any machine readable form of the Licensed Programs for his own use and merge it into other program material to form an updated work, provided that, upon discontinuance of the license for such Licensed Program, THE LICENSED PROGRAM SUPPLIED BY DIGITAL RESEARCH WILL BE COMPLETELY REMOVED FROM THE UPDATED WORK. ANY PORTION OF THE LICENSED PROGRAM INCLUDED IN AN UPDATED WORK SHALL BE USED ONLY IF ON THE DESIGNATED SYSTEM AND SHALL REMAIN SUBJECT TO ALL OTHER TERMS OF THIS AGREEMENT.

The Customer agrees to reproduce and include the copyright notice of Digital Research on all copies, in whole or in part, in any form, including partial copies of modifications, of Licensed Programs made hereunder.

4. PROTECTION AND SECURITY: The Customer agrees not to provide or otherwise make available any Licensed Program including but not limited to program listings, object code and source code, in any form, to any person other than Customer or Digital Research employees, without prior written consent from Digital Research, except with the Customer's permission for purposes specifically related to the Customer's use of the Licensed Program.

5. DISCONTINUANCE: Within one month after the date of discontinuance of any license under this Agreement, the Customer will furnish Digital Research a certificate certifying that through his best effort, and to the best of his knowledge, the original and all copies, in whole or in part, in ANY form, including partial copies in modifications, of the Licensed Program received from Digital Research or made in connection with such license have been destroyed, except that, upon prior written authorization from Digital Research, the Customer may retain a copy for archive purposes.

6. DISCLAIMER OF WARRANTY: Digital Research makes no warranties with respect to the Licensed Programs. The sole obligation of Digital Research shall be to make available all published modifications or updates made by Digital Research to Licensed Programs which are published within one (1) year from date of purchase, provided Customer has returned the Registration Card delivered with the Licensed Program.

7. LIMITATION OF LIABILITY: THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL DIGITAL RESEARCH BE LIABLE FOR CONSEQUENTIAL DAMAGES EVEN IF DIGITAL RESEARCH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

8. GENERAL: If any of the provisions, or portions thereof, of this Agreement are invalid under any applicable statute or rule of law, they are to that extent to be deemed omitted.