Microsoft®
Premium SoftCard™ IIe

Package

for Apple® IIe Computer

Programmer's Manual

Microsoft Corporation

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

Part No. 23F47BP

Document No. 8816-226-00

Contents

Introduction

This is the *Programmer's Manual* for the Microsoft® Premium SoftCard™ IIe 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® IIe 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 Premium SoftCard IIe System 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 Premium SoftCard IIe System Installation and Operation Manual* and the *Osborne® 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 Premium SoftCard IIe System should contact Microsoft Corporation directly.

How to Use This Manual

This manual serves as:

1. A reference manual for using CP/M commands and programs

2. A technical manual for programming in the SoftCard IIe environment

Some of the material found in the *Microsoft Premium SoftCard IIe Installation and Operation Manual* has been repeated in this manual for the convenience of the reader.

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 IIe 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 Premium SoftCard IIe System.

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

Chapter 7, "Using the SoftCard With Apple Programs," describes the commands for using the SoftCard IIe as extended memory and an 80-column text interfacer board when running Apple programs.

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

Appendix B, "Downloading to the SoftCard" explains how to use the UPLOAD and DOWNLOAD utility programs to load CP/M software from other computers to your Apple IIe.

Appendix C, "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.

Notation Used in This Manual

This manual uses the same notation as the *Microsoft Premium SoftCard IIe System 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 IIe Differences

On the European version of the Apple IIe 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 IIe 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 Premium SoftCard IIe System.

CP/M Memory Organization

The Premium SoftCard IIe 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.

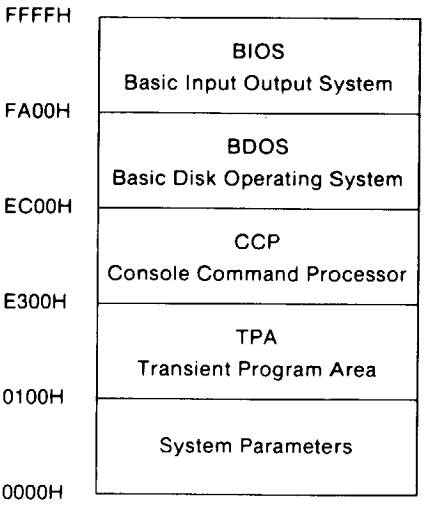


Figure 1.1. CP/M Memory Organization

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 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 Premium SoftCard IIe System 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 Premium SoftCard IIe System 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:
11	UL1:	UP2:	UR2:	UC1:
10	LPT:	UP1:	PTR:	BAT:

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 Premium SoftCard IIe System 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 Premium SoftCard IIe System. It also provides guidelines for using CP/M system calls within your programs.

Assembly Language Programming

With the Premium SoftCard IIe 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 Premium SoftCard IIe 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 IIe 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,0         ;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

Because there is no interrupt line to the AUXILIARY slot of the Apple IIe motherboard, Z80 interrupts are not implemented. 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 Premium 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
13	Reset Disk System	33	Read Random
14	Select Disk	34	Write Random
15	Open File	35	Compute File Size
16	Close File	36	Set Random Record
17	Search for First	37	Reset Drive
18	Search for Next	40	Write Random With Zero Fill
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	WRITEMEM	10	FORMAT
2	READMEM	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 Premium SoftCard IIe. 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 IIe is installed in the Apple IIe 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 IIe 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 IIe 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.

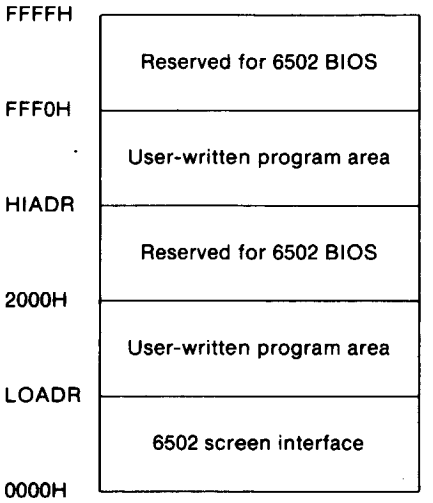


Figure 4.2. 6502 BIOS Memory Map

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, while HIMEM contains the lowest addresses available in the second area. 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, WRITEMEM, 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	2
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 starting point of the function request routine.) 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 WRITEMEM

Purpose

Writes a byte to a 6502 memory location.

Parameters

Parameter	Address	Contents
Entry point	49H	1
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.

2 READMEM

Purpose

Reads a byte from 6502 memory.

Parameters

Parameter	Address	Contents
Entry point	49H	2
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.

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 or 5, 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	47H	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	47H	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 Premium SoftCard IIe 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 Premium SoftCard IIe System 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 Premium SoftCard IIe System 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 IIe 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 Premium SoftCard IIe 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 Premium SoftCard IIe System 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 IIe 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.

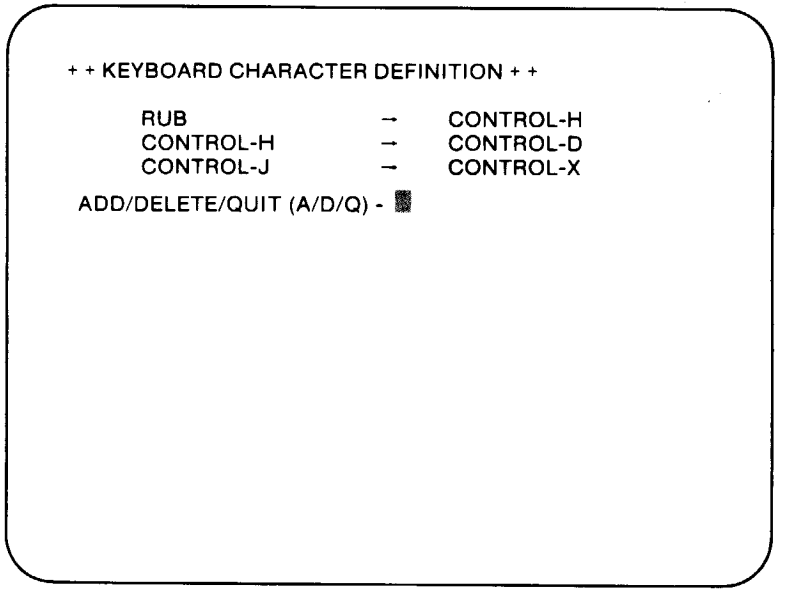
Keyboard Character Definition Table

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 the beginning of this chapter. When the first CONFIGIO selection menu appears, press the *2* key. CONFIGIO will display the Keyboard Character Definition menu as shown below:



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 the initial release of Apple Pascal, 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 Premium SoftCard IIe 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 II 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 II 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.

Chapter 7
Using the SoftCard With Apple Programs

The SoftCard display and memory features can be used with Apple DOS, Apple Pascal and Apple BASIC programs as well as CP/M. This chapter describes how to use the Microsoft Premium Softcard IIe System display and memory features with your Apple programs.

SoftCard Features Under Apple DOS

When Apple DOS, Apple Pascal, or Apple BASIC (Applesoft® or Integer BASIC) programs are running, the SoftCard circuit board can be used exactly like the Apple 80-Column Text Card or Extended 80-Column Text Card. The SoftCard recognizes all commands for either board.

The display circuitry of the SoftCard supports all special features and commands associated with Apple IIe 80-column firmware contained on the Apple IIe motherboard. Display firmware is described in Chapter 3 of the *Apple IIe Reference Manual*. Instructions for using the SoftCard display features are described in the next section, "Using the SoftCard Display Features."

The SoftCard contains 64K bytes of random access memory that can be accessed by the 6502 microprocessor. In the 40-column display mode, the entire SoftCard memory is available for data storage. In the 80-column display mode, 63K bytes are available for storage and the remaining 1K byte is reserved for the 80-column display.

Using the SoftCard Display Features

When running Apple DOS or Apple BASIC programs, the SoftCard display features can be switched between 40-column and 80-column display modes. The Apple Pascal operating system, like CP/M, uses the 80-column display mode only.

Switching Between Displays

There are two methods for switching display modes when running either Apple DOS or Apple BASIC programs. The first method is activating and deactivating the display circuitry of the SoftCard. The second method is a temporary switch between display modes using escape key sequences. Most often, the display features will be activated when Apple DOS or Apple BASIC is loaded into memory. Then, the escape key sequences are used to switch between display modes. Deactivating the SoftCard display features is performed only when required. For example, the display features are turned off when sending output to certain I/O devices such as an Apple Silentype printer. Table 7.1 shows the commands for switching display modes.

Table 7.1.

SoftCard Display Mode Commands

Command	Purpose
PR#3	Activates the 80-column display features of the SoftCard circuit board.
ESC-4	Changes to the 40-column display mode.
ESC-8	Changes to the 80-column display mode.
ESC-CONTROL-Q	Deactivates the 80-column display features of the SoftCard circuit board.
CONTROL-RESET	Deactivates the 80-column display features and clears certain portions of memory.

Changing the Default Display Mode

The 40-column display is the default display mode for Apple DOS or Apple BASIC programs. By modifying the HELLO program, you can automatically switch to the 80-column display mode whenever Apple DOS is loaded into memory. To modify the HELLO program, perform the following steps:

1. Insert a copy of the DOS master disk into drive 1 and load the HELLO program by typing

```
LOAD HELLO
```

2. When you see the Apple DOS "]" prompt, type

```
LIST
```

3. When you see the "]" prompt again, type the following program lines:

```
1 D$=CHR$(4)
2 PRINT D$;"PR#3"
```

4. Now type

```
UNLOCK HELLO
SAVE HELLO
LOCK HELLO
```

This procedure can be repeated to change the default display mode for other Apple DOS system disks.

Deactivating the 80-Column Display Mode

There are two commands for deactivating the 80-column display mode: ESC-CONTROL-Q and CONTROL-RESET. The preferred method of deactivating the 80-column display mode is by typing ESC-CONTROL-Q. Typing CONTROL-RESET, under certain conditions, could erase your program from memory. Using the CONTROL-RESET method will also fill the screen with random characters when the system changes display modes. The random characters are in the video memory only and will scroll off the screen when entering new data.

The only time you should deactivate the 80-column display mode is when certain application programs require you to do so, or when sending the output of the program to another device such as a printer.

Cursor Symbols

Whenever you change display modes, the cursor will change shape. Table 7.2 shows the different shapes of the cursor and the corresponding display mode.

Table 7.2.

Cursor Symbols

Cursor	Mode	Remarks
	Active	The cursor is always a rectangle whenever CP/M or Pascal operating systems are running. Under Apple DOS or Apple BASIC, the cursor will be a rectangle after you activate the 80-column display.
	Inactive	When Apple DOS or BASIC is loaded into memory at system startup time, the cursor is shown as a flashing checkerboard.
	Active	The cursor is shown as in inverse cross in a rectangle when ESC-8 is typed.*
	Inactive	The cursor is shown as an inverse cross in a square when ESC-4 is typed.*

* Applicable in Apple DOS or BASIC only.

80-Column Operation

80-column operation is defined when the 80-column display circuits of the SoftCard are active. When active, the operation of certain display commands are modified. Table 7.3 lists the Applesoft BASIC display commands that are affected when 80-column display mode is active.

Table 7.3.

Restrictions for Using Applesoft BASIC Commands in 80-Column Display

Command	Action
FLASH	*80-column active mode*: Not available in this mode.
	80-column inactive mode: FLASH performs as described in Chapter 4 of the *Applesoft BASIC Programming Reference Manual*; characters flash between normal and inverse video.
INVERSE	*80-column active mode*: INVERSE performs as described in Chapter 4 of the *Applesoft BASIC Programming Reference Manual*; black characters on a white screen. Both uppercase and lowercase characters are allowed. The HOME command clears the screen to a white background.
	80-column inactive mode: Performs the same way as in the 80-column active mode but restricted to uppercase characters only. The HOME command clears the screen to a black background.

Table 7.3. *(continued)*

Command	Action
NORMAL	*80-column active mode*: NORMAL performs as described in Chapter 4 of the *Applesoft BASIC Programming Reference Manual*; it clears the screen to a black background.
	80-column inactive mode: Performs the same way as in the 80-column active mode.
HTAB	*80-column active mode*: HTAB performs as described in Chapter 4 of the *Applesoft BASIC Programming Reference Manual* when using the ESC-4 sequence to display 40 columns. When displaying 80 columns, HTAB is unavailable. (Use the POKE 36,*xx* command instead.)
	80-column inactive mode: HTAB performs as described in Chapter 4 of the *Applesoft BASIC Programming Reference Manual.*
Comma Tabbing	*80-column active mode*: Comma tabbing performs as described in "Print Format" in Chapter 1 of the *Applesoft BASIC Programming Reference Manual* when using the ESC-4 sequence to display 40 columns. Unavailable in the 80-column display mode.
	80-column inactive mode: Comma tabbing performs as described in Chapter 1 of the *Applesoft BASIC Programming Reference Manual.*

Escape Key Sequences

Table 7.4 lists the escape key sequences that can be used when the SoftCard display features are active.

Table 7.4.

Escape Key Sequences for Display Modes

Sequence	Action
ESC-@	Clears the window and moves the cursor to its HOME position.
ESC-A	Moves the cursor up one line.
ESC-B	Moves the cursor right one position.
ESC-C	Moves the cursor left one position.
ESC-D	Moves the cursor down one line.
ESC-E	Clears to the end of the line.
ESC-F	Clears to the bottom of the window.
ESC-I	Moves the cursor up one line and turns on escape mode.
ESC-↑	Same as ESC-I.
ESC-J	Moves the cursor left one position and turns on escape mode.
ESC-←	Same as ESC-J.
ESC-K	Moves the cursor right one position and turns on escape mode.
ESC-→	Same as ESC-J.
ESC-M	Moves the cursor down one line and turns on escape mode.
ESC-↓	Same as ESC-J.
ESC-R	Turns on uppercase restrict mode.
ESC-T	Turns off uppercase restrict mode.
ESC-4	Turns on the 40-column display mode.
ESC-8	Turns on the 80-column display mode.
ESC-CONTROL-Q	Deactivates the 80-column circuits of the SoftCard.

Control Key Sequences

Some of the control key sequences that work with Apple BASIC programs work differently in the active 80-column display mode. Table 7.5 lists the control key sequences and their actions for both 80-column active and inactive modes.

Table 7.5.

Control Key Sequences

Sequence	Apple IIe Name	ASCII Code	Action
CONTROL-G	Bell	7	Produces a tone (1000 Hz) for 0.1 seconds.
CONTROL-H	Backspace	8	Moves the cursor one position to the left.
CONTROL-J	Linefeed	10	Moves the cursor down to the next line.
CONTROL-K	Clear EOS	11	Clears from the cursor position to the end of the screen.
CONTROL-L	Clear	12	Moves the cursor to the left corner of the screen and clears the entire screen.
CONTROL-M	Return	13	Moves the cursor to the leftmost column of the next line.
CONTROL-N	Normal	14	Sets the display format to normal mode.
CONTROL-O	Inverse	15	Sets the display format to inverse mode.
CONTROL-Q	40-column	17	Sets the display to the 40-column mode of operation.
CONTROL-R	80-column	18	Sets the display to 80 columns.

Table 7.5. *(continued)*

Sequence	Apple IIe Name	ASCII Code	Action
CONTROL-S	Stop list	19	Stops the output to the display until another key is pressed. This command will only work at Apple DOS command level and not from within a program.
CONTROL-U	Quit	21	Deactivates the 80-column circuits, places the cursor in the HOME position and clears the screen.
CONTROL-V	Scroll	22	Scrolls the display down one line with the cursor in the same column as before.
CONTROL-W	Scroll-up	23	Scrolls the display up one line with the cursor in the same column as before.
CONTROL-Z	Home	25	Moves the cursor to the upper left corner of the screen.
CONTROL-	Forward space	28	Moves the cursor right one position. If the cursor is on the left side of the screen, it will be moved to the left end of the line below.
CONTROL-]	Clear EOL	29	Clears to the end of the line from the cursor.
CONTROL-^	GOTO xy	30	Moves the cursor to the coordinate given. This key sequence is not supported by Apple BASIC programs.

With the exception of CONTROL-Q, CONTROL-H, CONTROL-J, and CONTROL-M, control key sequences are only available when the SoftCard display features are active.

CONTROL-N, CONTROL-O, CONTROL-Q, CONTROL-R, and CONTROL-U are available only from BASIC programs and will not work at Apple DOS command level.

Using the SoftCard 64K-Byte Memory

The Premium SoftCard IIe System has 64K bytes of additional memory available for data storage. In Apple nomenclature, any memory not contained on the Apple motherboard is referred to as *auxiliary memory*. Therefore, all of the SoftCard memory is considered to be auxiliary memory.

Of the 64K bytes of SoftCard memory, one kilobyte (addresses 0400H—7FFH) is reserved for the 80-column video display. The other 63K bytes can be used for data storage. If the SoftCard display circuits are inactive, all 64K of the SoftCard's memory can be used for data storage.

SoftCard Video Memory Operation

When the SoftCard 80-column features are active, half of the data for the 80-column display is stored in main memory on text page 1 and the remaining data is stored in SoftCard memory. The 80-column display circuits receive data from both memory areas simultaneously and display them as two adjacent characters.

Both memory areas are connected to the address bus in parallel. This allows access to both memory areas during a display cycle. When in the 40-column display mode, the SoftCard display circuits use every other clock cycle to get data from memory. In 80-column display mode, the remaining clock cycles are for processing the additional display data from SoftCard memory.

In 80-column mode operation, a byte of display data from main memory is sent to a buffer on the main logic board, and the display data from SoftCard memory goes into a buffer on the SoftCard. The data bytes from these buffers are then switched onto the video data bus on alternate clock cycles. (The first byte is from the SoftCard and the next byte is from main memory and so on.) The main memory displays characters in odd columns and the auxiliary SoftCard memory displays characters in even columns.

Because the 80-column display contains twice as many characters as the 40-column, it has to output twice as many pixels across the screen horizontally. Pixels are the dots that compose screen characters or graphics. This doubled output increases the data transmission rate to the screen from 7 MHz to 14 MHz, effectively making the characters more narrow and therefore more dim on a normal video monitor. Therefore, to produce a satisfactory 80-column display, a monitor with a bandwidth of at least 14 MHz is required.

This video data transmission scheme applies only to the video display. Access of data in the SoftCard memory is accomplished by switching the data bus to Read/Only from the SoftCard as described in the previous paragraphs. For more information about how the Apple IIe display memory operates, see Chapter 2 and Chapter 7 of the *Apple IIe Reference Manual.*

Addressing SoftCard Memory

Because the 6502 microprocessor can only address 64K bytes of memory at a time, special circuitry has been built into the Apple motherboard to access auxiliary memory. Thus, the locations in the 64K address space can be either in the main memory (Apple motherboard) or in auxiliary memory (SoftCard circuit board). Figure 7.1, "Memory Mapping Under Apple DOS," shows the address mapping of both memory areas.

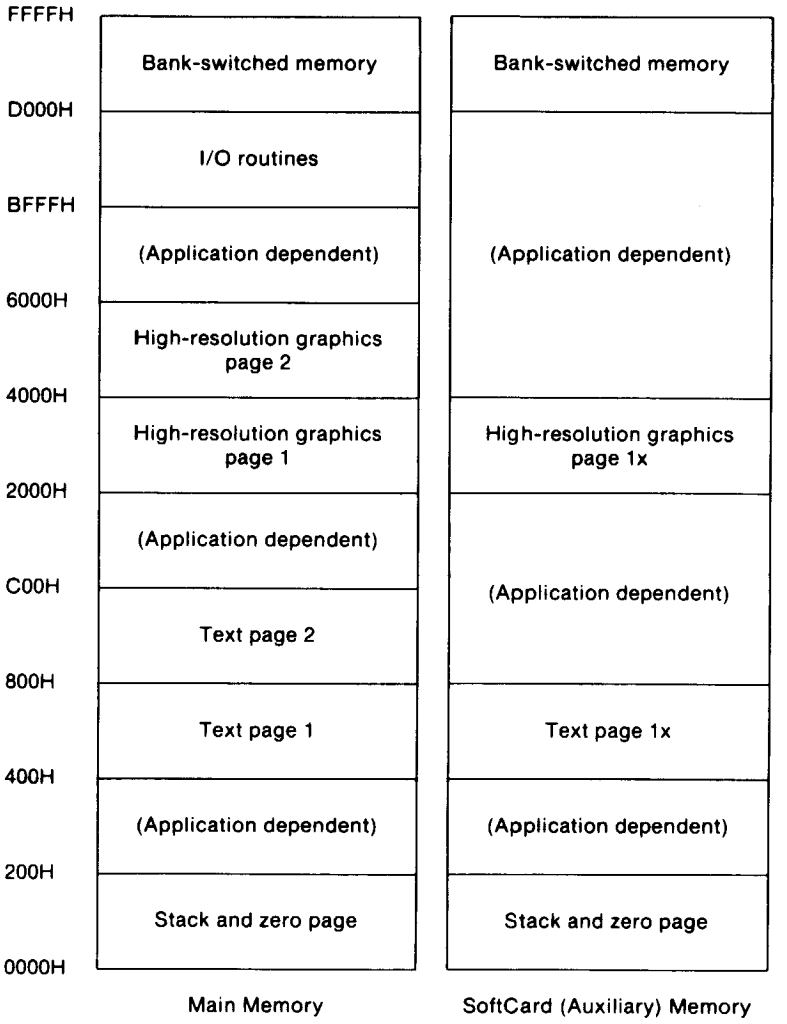


Figure 7.1. Memory Mapping Under Apple DOS

The 6502 addresses the SoftCard memory (or any other auxiliary memory) through the Apple address bus. To use the SoftCard memory for data storage, the 6502 switches its data bus so that it accesses SoftCard memory instead of the main memory. To expand the display for 80-column operation, the 6502 gets data from both memory areas as explained in "SoftCard Video Memory Operation," earlier in this chapter.

The components that control the bus switching are called the Memory Management Unit (MMU). The MMU works in conjunction with the 6502. It also contains firmware (soft switches) that can be set by programs to monitor the address bus and switch the address bus to the appropriate memory area.

The SoftCard memory is divided into three segments. The largest area is referred to as the 48K segment and is for data storage. The bank-switched segment is a 16K-byte section of memory that replaces the main memory in the upper address range (D000H to FFFFH). If you plan to use this part of the SoftCard memory, read the section entitled "Bank-Switched Memory" in Chapter 4 of the *Apple IIe Reference Manual.*

Note

The switching of the ROM and D000H bank is independent of the the auxiliary RAM switching, so the bank switches have the same effect on the SoftCard RAM as they do on the main memory.

The lowest addresses of the SoftCard memory are reserved for the 6502 stack and zero page. (Zero page is used for system parameters.) Any time the addresses are switched to the bank-switched SoftCard memory, the addresses in zero page and page 1 (the 6502 stack) are also switched.

Doubling the Resolution of Graphic Displays

As described in "SoftCard Video Memory Operation" in this chapter, high-resolution graphics in the 80-column display mode will have the same resolution as in the 40-column display mode. A mixed mode text display in 80-column display mode can be used. *Mixed mode text display* is defined as a graphic screen display with the bottom four lines reserved for text.

To increase the horizontal resolution of graphic or mixed mode displays, the Annunciator 3 soft switch is turned on. The SoftCard can be modified to double normal graphics resolution with 80-column text.

Modify the SoftCard by changing the jumper on the SoftCard circuit board to the position shown in Figure 7.2. Use the installation procedure in the *Microsoft Premium SoftCard IIe Installation and Operation Manual* to remove and reinstall the circuit board.

The AN3 (Annunciator 3) switch is turned on by writing to location C05FH and turned off by writing to location C05EH. When you change the jumper 560 x 192 position, turn on the AN3 switch, and select the high-resolution graphics, mixed text display mode.

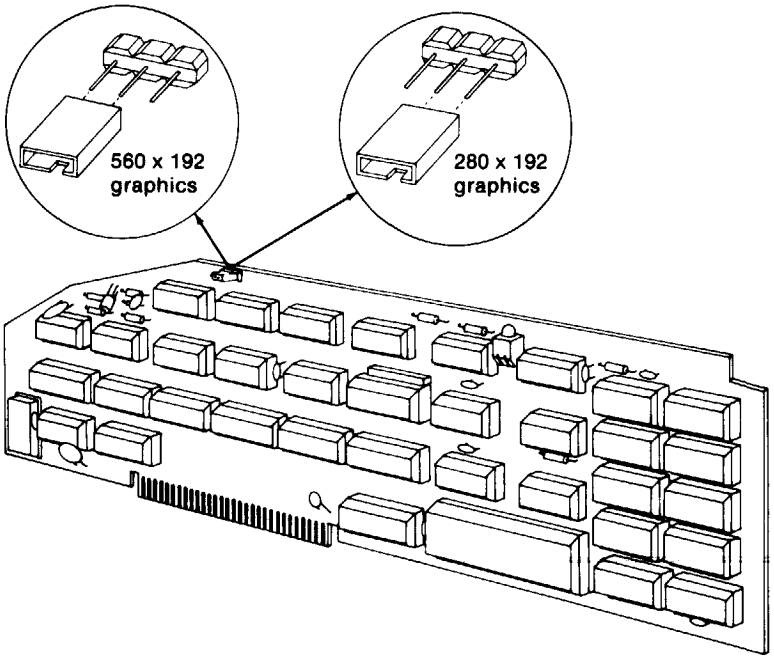


Figure 7.2. Changing the High-Resolution Graphics Jumper

Note

The European version of the SoftCard circuit board differs slightly from the one illustrated in Figure 7.2, but the jumper configurations remain the same.

When the jumpers have been changed, the soft switches described in "Display Mode Switching" in this chapter must be turned on to select the 80-column display mode.

Warning

Doubling the resolution of the 80-column display by the method just described is only applicable to Rev B and later Apple IIe computers. If you have a Rev A Apple IIe, changing the jumpers on the SoftCard circuit board will cause unpredictable results.

The revision letter for the Apple IIe is located on the Apple IIe motherboard near the back panel.

When the SoftCard jumper is changed and the high-resolution graphics display mode is selected, the Apple IIe generates a display using high-resolution page 1 addresses in both main memory and SoftCard memory.

Memory mapping for the high-resolution graphics display doubles the columns as does 80-column text display mode, but uses high-resolution page 1 instead of text page 1. In 80-column text mode, a pair of data bytes is displayed as pairs of characters. In double high-resolution mode, however, the data byte pair is displayed as adjacent screen pixels, seven for each byte. As in 80-column text mode, there are twice as many pixels across the display screen. Therefore, the pixels are only half as wide.

Existing Apple II graphics programs do not support the doubled high-resolution graphics display. Until new programs become available, you will have to write your own plotting routines.

The Extended Display

To take advantage of the additional memory, you must set up your programs to operate in one part of memory while they switch the other part between main and auxiliary RAM. Your program can perform the memory switching by means of the soft switches described in the section "Display Mode Switching" later in this chapter, or by using the AUXMOVE and XFER subroutines described later in this section. Except for these subroutines, most existing Apple II system software (DOS version 3.3, Pascal version 1.1) doesn't support the additional memory.

Warning

Do not use SoftCard memory directly from a program running under interpreter languages such as Apple BASIC or Pascal. Interpreters map the main memory areas differently and will abort if switched to an auxiliary memory area. When restarted, programs and data can be lost.

Display Pages

The Apple IIe video display is generated from data stored in specific memory areas called display pages. The 40-column text and low-resolution display modes use text page 1 (400H—7FFH) and text page 2 (800H—BFFH) in main memory.

The 80-column text display uses a combination of text page 1 in main memory and page 1x in the SoftCard memory. Text page 1x uses the same address range as text page 1, but resides in the SoftCard memory rather than main memory. To store data in page 1x, you must use the appropriate soft switches to enable the 80-column display routines in ROM. The display modes and the corresponding display page addresses are listed in Table 7.6.

Table 7.6.

Display Page Addresses

Display Mode	Display Page	Address Range
40-column text	Page 1	400H—7FFH
Low-resolution graphics (40x48)	Page 2	800H—BFFH
80-column text	Page 1	400H—7FFH
Standard high-resolution graphics (280x192)	Page 1 Page 2	2000H—3FFFH 4000H—5FFFH
Double high-resolution graphics (560x192)	Page 1	2000H—3FFFH

Display Mode Switching

You can select the display mode that is appropriate for your application by accessing the Apple soft switches. The switches have three addresses for enabling, disabling, and checking the status of the switch.

Table 7.7 shows the addresses of the soft switches that control the display modes. The table gives the switch locations in three forms: hexadecimal, decimal, and negative decimal. You can use the hexadecimal (hex.) values in your assembly language programs and the decimal (dec.) values in PEEK and POKE commands in Applesoft BASIC programs. Negative decimal (neg. dec.) values are used in Integer BASIC programs.

Important

Make sure that only the indicated operations are used for the soft switches. If you read a switch marked WRITE, erroneous data is returned.

Table 7.7.
Display Mode Soft Switches

Name and State	Purpose	Memory Addresses		
		Hex.	Dec.	Neg. Dec.
TEXT				
ON	Display text	C051	49233	-16303
OFF	Display graphics	C050	49232	-16304
READ	TEXT status	C01A	49178	-16358
MIXED				
ON	Text with graphics	C053	49235	-16301
OFF	Graphics only	C052	49234	-16302
READ	MIXED status	C01B	49179	-16357
PAGE2				
ON	Display page 2	C055	49237	-16299
OFF	Display page 1	C054	49236	-16300
READ	PAGE2 status	C01C	49180	-16356
HIRES				
ON	High-resolution graphics	C057	49297	-16297
OFF	Low-resolution graphics	C056	49298	-16298
READ	HIRES status	C01D	49181	-16355
80COL				
ON	Display 80-column	C00D	49165	-16371
OFF	Display 40-column	C00C	49164	-16372
READ	80COL status	C01F	49183	-16353
80STORE				
ON	Store in auxiliary page	C001	49153	-16383
OFF	Store in main page	C002	49152	-16384
READ	80STORE status	C018	49176	-16360

80-Column Display Memory

Data for 80-column display mode is stored in the same memory locations in text page 1 (main memory) and in text page 1x (SoftCard memory). Figure 7.3 shows the memory mapping for 80-column text display. Odd bytes are stored in SoftCard memory and even bytes are stored in main memory. When the 80-column display mode is active, the SoftCard reads data from both memory areas simultaneously, but displays data sequentially. The byte from SoftCard memory is displayed first, followed by the byte from the main memory.

You can store data directly into SoftCard memory by setting the 80STORE switch to ON. Setting the 80STORE switch to ON enables the PAGE2 soft switch to select between display stored in page 1 and page 1x.

Important

When you change the 80STORE and PAGE2 switches from the Apple monitor program, it will change the switch setting back to the original setting when you display the commands you type.

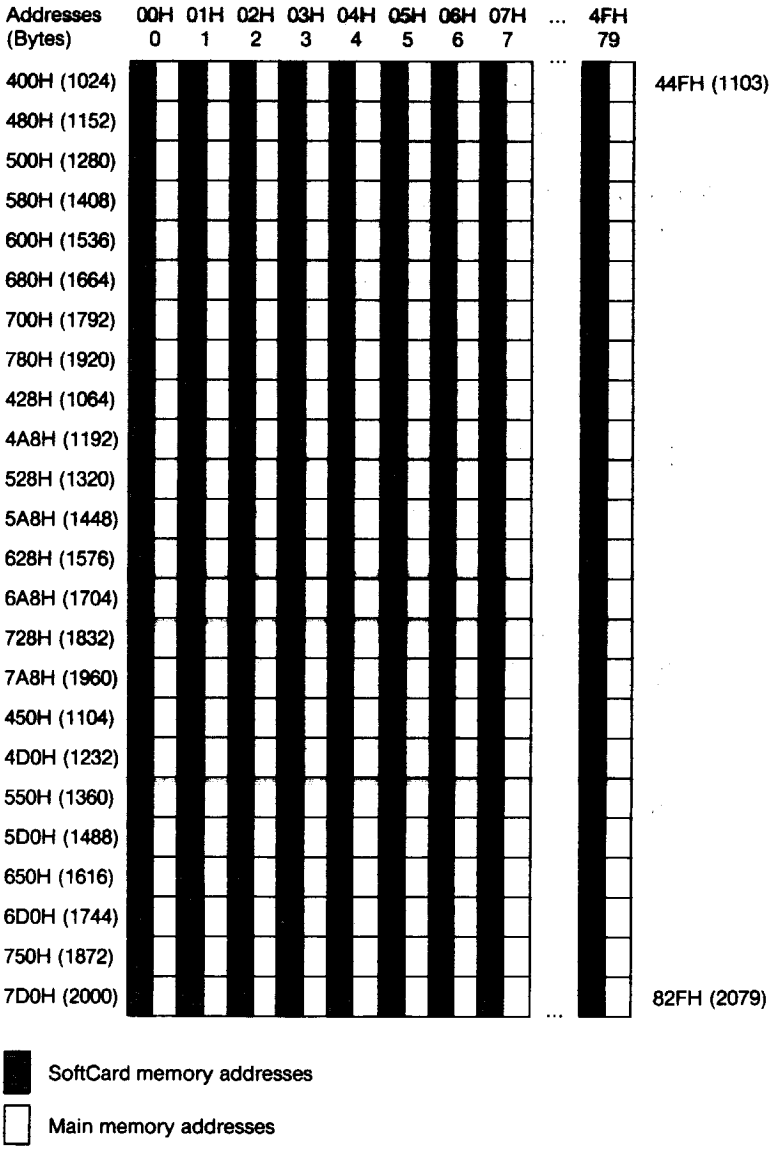


Figure 7.3. 80-Column Text Display Map

To use the 560x192 high-resolution graphics, store data directly on high-resolution page 1x in the SoftCard memory as described earlier in this section. Set both 80STORE and HIRES to ON and use PAGE2 to switch from page 1 in main memory to page 1x in the SoftCard memory.

Memory mapping for 560x192 high-resolution graphics is similar to the standard high-resolution graphics mapping which is described in Chapter 2 of the *Apple IIe Reference Manual*, with the addition of the column doubling produced by the 80-column display. Like the 80-column text mode, the double high-resolution graphics displays two bytes in the time normally required for one, but uses high-resolution graphics page 1 and page 1x instead of text page 1 and page 1x.

In 560x192 high-resolution graphics mode, each pair of data bytes is displayed as 14 adjacent pixels, seven for each byte. The high-order bit (color-select bit) of each byte is ignored. In this mode, the SoftCard memory byte is displayed first to allow data from the SoftCard memory to appear in every other column segment (seven columns each) starting with column segment 0—6 and ending with column segment 547—552. Data from the main memory appears in remaining column segments (7—13, 21—27, etc.) up to column segment 553—559.

As in 80-column text mode, there are twice as many pixels across the display screen making the pixels only half as wide. On a low-bandwidth monitor or a TV set, single pixels are dimmer than normal.

SoftCard Memory Switching

The SoftCard memory can be switched with the soft switches listed in Table 7.8. The sections following the table describe how to use the switches.

Table 7.8.
Memory Select Switches

Name and State	Purpose	Memory Addresses Hex.	Dec.	Neg. Dec.
RAMRD				
ON	Read auxiliary 48K	C003	16155	-16381
OFF	Read main 48K	C002	16154	-16382
READ	RAMRD status	C013	16171	-16365
RAMWRT				
ON	Write auxiliary 48K	C005	16157	-16379
OFF	Write main 48K	C004	16156	-16380
READ	RAMWRT status	C014	16172	-16354
ALTZP				
ON	Access auxiliary memory areas*	C009	16373	-16373
OFF	Access main memory areas*	C008	16374	-16374
READ	ALTZP status	C016	16352	-16352
80STORE				
ON	Access page 1x	C001	16383	-16383
OFF	Use RAMRD, RAMWRT	C000	16384	-16384
READ	80STORE status	C018	16360	-16360
PAGE2				
ON	Access auxiliary memory	C055	16299	-16299
OFF	Access main memory	C054	16300	-16300
READ	PAGE2 status	C01C	16356	-16356
HIRES				
ON	Access high-resolution page 1x	C057	16297	-16297
OFF	Use RAMRD, RAMWRT	C056	16298	-16298
READ	HIRES status	C01D	16355	-16355

* Includes zero page, stack and bank-switched memory.

Note

To retain compatibility with Apple II software, the soft switches in Table 7.8 list their memory locations with the Apple keyboard functions listed in Table 2.2 of the *Apple IIe Reference Manual.* The read and write operations for keyboard functions are different from the read and write functions listed in Table 7.8. For more information about the keyboard functions, see Chapter 2 of the *Apple IIe Reference Manual.*

Switching the 48K Bank

Switching the 48K-byte section of memory is performed by the RAMRD and the RAMWRT switches. RAMRD selects memory for reading and RAMWRT selects memory for writing. Setting the switches independently makes it possible for a program which has instructions that are being fetched from one 48K-byte memory bank to store data in the 48K-byte bank.

SoftCard memory locations corresponding to text page 1 and high-resolution graphics page 1 can be used as part of the 48K bank by using RAMRD and RAMWRT. These SoftCard memory areas can also be controlled separately by using the display-page switches 80STORE, PAGE2, and HIRES described in "80-Column Display Memory" in this chapter. Figures 7.4 and 7.5 show which areas of memory are active or switched for the different settings of RAMRD and RAMWRT.

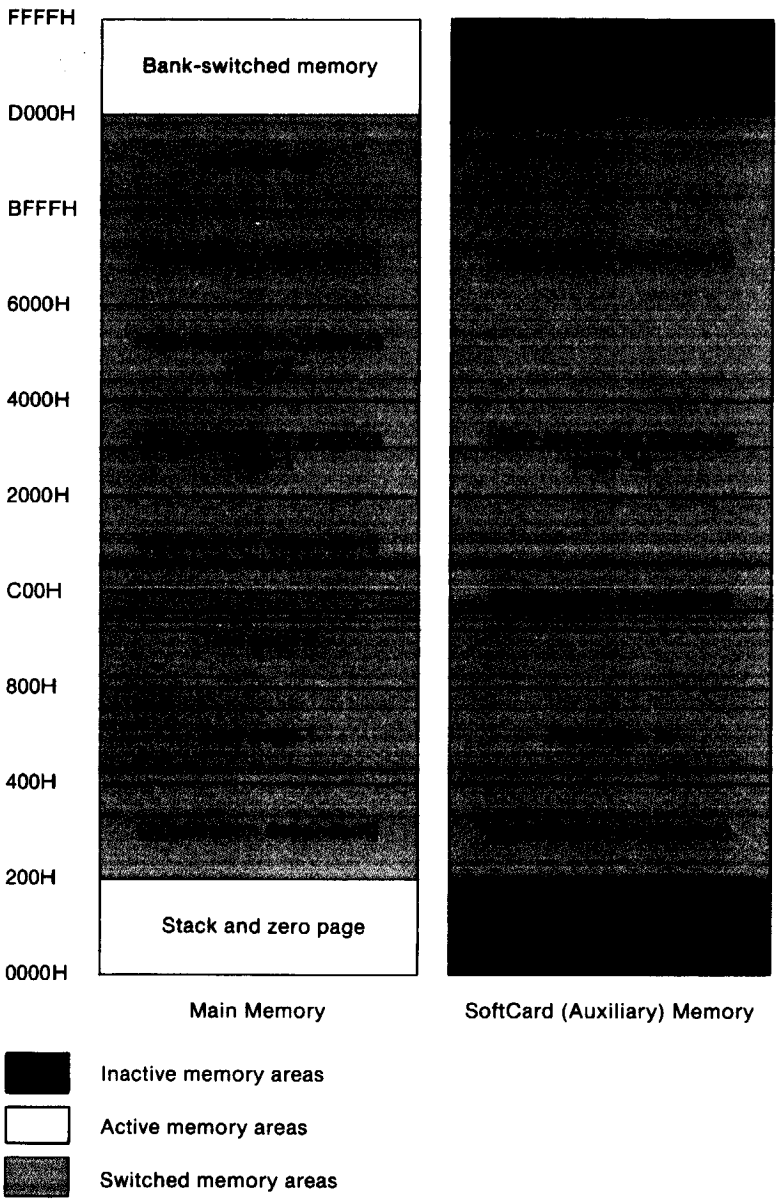


Figure 7.4. Memory Mapping With RAMRD and RAMWRT Switches On, 80STORE Off

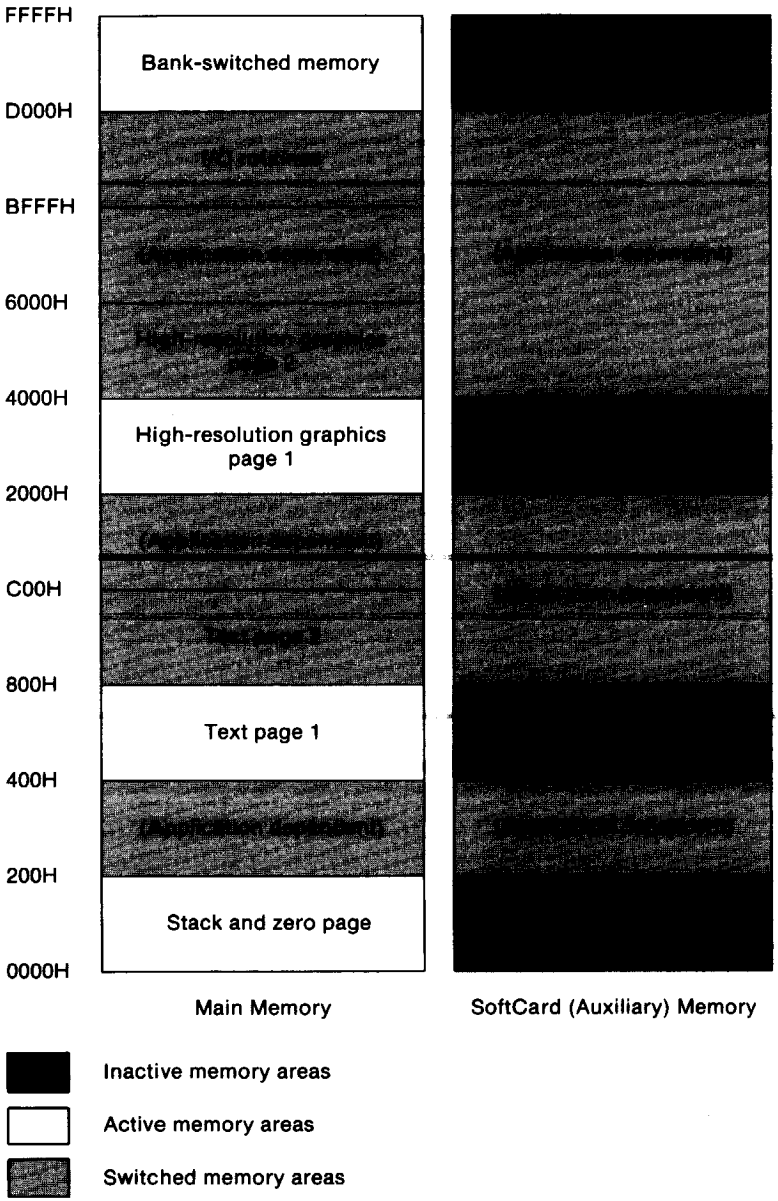


Figure 7.5. Memory Mapping With RAMRD, RAMWRT, 80STORE, and HIRES Switches On

As shown in Table 7.8, the 80STORE switch enables memory switching. When 80STORE is on, PAGE2 can select any memory bank. If the HIRES switch is off, the PAGE2 switch selects main or SoftCard memory in text display page 1 (0400H—07FFH). If HIRES is on, the PAGE2 switch selects main or auxiliary memory in text page 1 and high-resolution graphics page 1 (2000H—3FFFH).

If you use both the bank control switches and the display-page switches, the display-page switches take priority. That is, if 80STORE is off, RAMWRT and RAMRD control the entire memory space from 0200H to BFFFH. If 80STORE is on, RAMWRT and RAMRD have no effect on the display page and PAGE2 controls the text page. If both 80STORE and HIRES are on, then PAGE2 controls both text page 1 and high-resolution graphics page 1 regardless of the settings of RAMRD and RAMWRT. Figure 7.6 shows which segments of memory are used when PA6E2 is on.

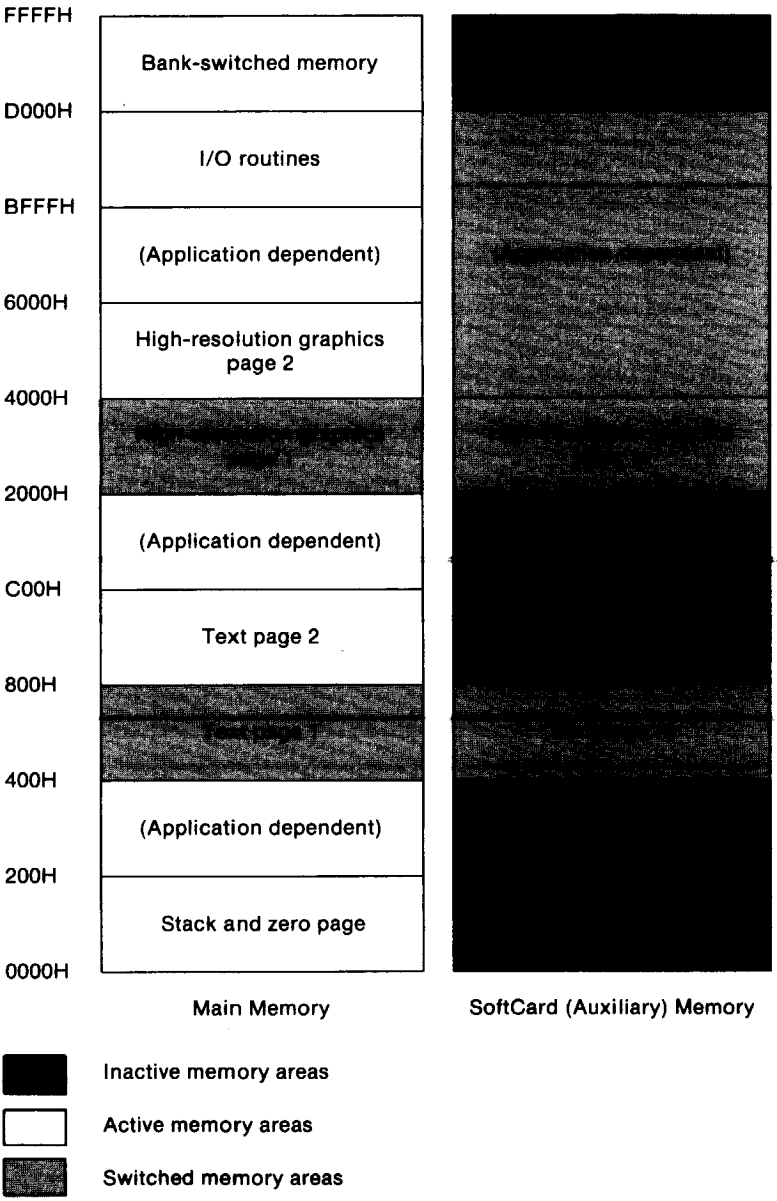


Figure 7.6. Memory Mapping With RAMRD, and RAMWRT Switches Off, and 80STORE, HIRES, and PAGE2 Switches On

To check the status of any of the switches, read the indicated status byte from Table 7.8. If the byte has the high bit set to a one, the switch is turned on. If the high bit is zero, then the switch is off.

Switching Other Segments of Memory

Soft switch ALTZP controls the access to the bank-switched memory, associated stack, and zero page areas. The following section, "Auxiliary Memory Subroutines," describes the firmware that can be called for switching between the main and SoftCard memory areas.

The status of the ALTZP switch is read from location C016H. If the sign bit (bit 7) is set to one, ALTZP is on. If zero, ALTZP is off. Figure 7.7 shows the affects of the ALTZP switch.

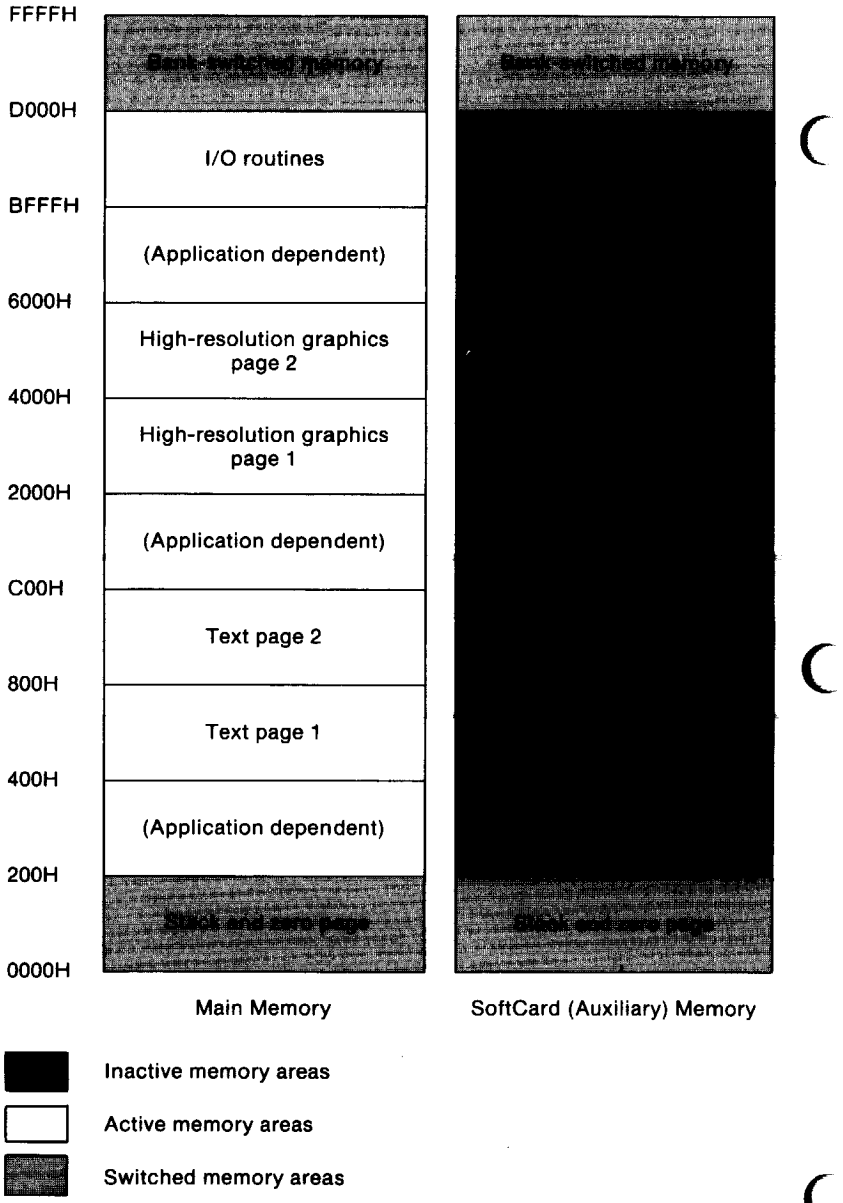


Figure 7.7. Memory Mapping With ALTZP Switch On

Auxiliary Memory Subroutines

You can use the SoftCard memory for assembly language programs or routines without the soft switches described by accessing the AUXMOVE and XFER subroutines stored in ROM. AUXMOVE and XFER make it easier to use the SoftCard memory but don't provide error protection.

The AUXMOVE subroutine copies data between memory areas and XFER transfers control between memory areas. Instructions for using AUXMOVE and XFER are given in Chapter 3 of the *Apple IIe Reference Manual*. AUXMOVE starts at address C311H and XFER starts at address C314H.

Copying Data Between Main Memory and the SoftCard

For 6502 assembly language programs and subroutines, AUXMOVE can be used to copy data between memory areas. Before AUXMOVE can be used, the data addresses must be stored in zero page by byte pairs and the carry bit of the 6502 Processor Status Word (PSW) must be set to one in order to select the direction of the data transfer.

The address byte pairs are called A1, A2, and A4. Their purpose and locations are given in Table 7.9.

Table 7.9.
AUXMOVE Parameters and Locations

Name	Address	Parameters
A1L	3CH	Low-order byte of the source start address.
A1H	3DH	High-order byte of the source start address.
A2L	3EH	Low-order byte of the source ending address.
A2H	3FH	High-order byte of the source ending address.
A4L	42H	Low-order byte of the destination address.
A4H	42H	High-order byte of the destination address.

Put the addresses of the first and last bytes of the block of data you want to copy into address byte pairs A1 and A2. The starting address of the destination memory area is put into address byte pair A4.

The AUXMOVE routine uses the carry bit to select the direction to copy the data. To copy from main memory to SoftCard memory, set the carry bit to one. Set the carry bit to zero to copy data from SoftCard memory to main memory.

When calling AUXMOVE, the subroutine copies the block of data as specified by the 6502 register A and the carry bit. The contents of the accumulator and the index registers (IX and IY) remain the same after the AUXMOVE routine has run.

Transferring Control to SoftCard Memory

Control is transferred between main memory and SoftCard memory through the XFER routine. To use XFER, set the XFER parameters to the following values: the carry and the overflow bits are set to one, and the program starting address is put in locations 3EDH—3EEH. (Setting carry and overflow to zero transfers control back to main memory.) When the parameters are set, use a a jump instruction to pass control to the XFER subroutine. XFER will preserve the contents of the accumulator and transfer address in the program stack. It then sets up the soft switches for the selected parameters and jumps to the new program.

Note

You must save the current stack pointer in the current program memory space before using the XFER routine. XFER writes over the current contents when running. When XFER transfers control back to the main program, it will restore the original stack pointer contents.

Appendix A
CP/M 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, MFT, or UPLOAD 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, STAT, or UPLOAD 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
Downloading to the SoftCard

The Premium SoftCard IIe System allows you to copy CP/M files from CP/M-based computer systems to Apple systems through an RS-232 Serial I/O port. UPLOAD and DOWNLOAD programs permit you to copy programs that are in a different disk format (for example, on 8-inch disk drives) to the SoftCard CP/M format (5-1/4-inch disk drives).

The UPLOAD program is a source listing that must be entered into the source computer (a system that does not support SoftCard) as part of the communication link. UPLOAD must be configured for the source computer's specific I/O environment with the DDT transient program.

DOWNLOAD is the counterpart to the UPLOAD program that must be entered into your Apple IIe computer system.

Warning

Use of UPLOAD and DOWNLOAD assumes familiarity with 8080 assembly language programming. These programs are intended for experienced programmers only!

Program Requirements

To use the UPLOAD and DOWNLOAD programs, you will need the following:

1. A working knowledge of the DDT transient program and 8080 assembly language programming.

2. A CP/M-based computer system (in addition to your Apple IIe) with an RS-232 Serial I/O port other than the port used for console I/O.

3. An Apple Communications Interface board or a CCS 7710A Serial Interface board installed in slot 2 of the Apple IIe computer.

4. An RS-232 Serial I/O port cable.

Downloading Procedure

The following procedure shows how to copy files from another CP/M system to your Apple IIe system.

1. Insert a copy of the SoftCard Master disk into drive A:.

2. Load CP/M with a cold start and then load the DDT transient program.

3. Using DDT, enter the following machine-language program (UPLOAD) into the source computer starting at address 0163H:

```
0163 3A 80 00 B7 11 D7 01 CA CC 01 CD 03 01
0170 0E 0F 11 5C 00 CD 05 00 3C 11 E5 01 CA CC 01 3E
0180 52 CD 43 01 CD 23 01 FE 53 C2 7F 01 3E 47 CD 43
0190 01 11 06 02 CD D2 01 0E 14 11 5C 00 CD 05 00 B7
01A0 C2 C9 01 21 80 00 0E 00 16 80 7E CD 43 01 A9 4F
01B0 23 15 C2 AA 01 79 CD 43 01 CD 23 01 FE 42 CA A3
01C0 01 FE 47 CA 97 01 C3 B9 01 11 F4 01 CD D2 01 C3
01D0 00 00 0E 09 C3 05 00 43 6F 6D 6D 61 6E 64 20 45
01E0 72 72 6F 72 24 46 69 6C 65 20 6E 6F 74 20 66 6F
01F0 75 6E 64 24 0D 0A 55 50 4C 4F 41 44 20 43 6F 6D
0200 70 6C 65 74 65 24 55 70 6C 6F 61 64 69 6E 67 2E
0210 2E 2E 24 FE
```

4. Enter the following three bytes (a JMP 0163H instruction) at address 0100H:

   ```
   C3 63 01
   ```

   When you have finished, check the data entered. Use the DDT "L" command to list the program and compare it with the displayed UPLOAD listing in this section. Before making any of the following program modifications, exit DDT and save the program by typing in

   ```
   SAVE 2 UPLOAD.COM.
   ```

   and pressing the RETURN key.

5. Modify the UPLOAD program to recognize the RS-232 Serial I/O port on the source computer. Use DDT to write subroutines at the three memory addresses outlined in the following list. Each subroutine must begin at the address specified next to the subroutine. 32 bytes are allocated for each subroutine.

Subroutine	*Address*
INITSER	0103H

This subroutine initializes the serial port on the source machine (baud rate, data format, etc.). The data format should be set up for eight data bits, one stop bit, no parity, and for compatibility with the Apple Communications Interface board or CCS 7710A Serial Interface board.

Subroutine	*Address*
RETSER	0123H

If no character is available at the serial port, this subroutine must return A=00. If a byte is available, the routine should read the byte and return it in the A register.

Subroutine	*Address*
WRSER	0143H

This subroutine outputs a byte to the serial port. You must save all registers including register A.

Once the routines are written into the UPLOAD program, save the program in a disk file with the SAVE command.

6. Connect an RS-232 Serial I/O port cable from the Apple IIe to the source computer. One port must be configured as a send (DTE) device, and the other as a receiver (DCE). The XMIT and RCVE lines (pins 2 and 3) may have to be reversed, or certain "handshaking" lines connected. If you are using a CCS 7710A Serial Interface board, connect pins 4, 6, and 20 together on the Apple port. Ensure that the data formats used by the two serial ports are the same.

7. Once UPLOAD has been loaded into the source computer and the connecting cable is configured properly, you are ready to start transferring data.

   On the Apple IIe, type

   ```
   DOWNLOAD filespec
   ```

   filespec is the name of the file to be transferred.

   On the source computer, type

   ```
   UPLOAD filespec
   ```

   As soon as communication is established, the Apple displays

   ```
   DOWNLOADING
   ```

   and the source computer displays

   ```
   UPLOADING...
   ```

   As each 128-byte record is transferred successfully, a period (.) is displayed on the Apple screen. If an error is detected during the transfer of a 128-byte record, a "B" is printed, and the record is retransmitted.

When the transfer process has been completed, the source computer displays

```
UPLOAD COMPLETE
```

and returns to CP/M command level. When this message appears on the source computer, press CONTROL-C. The Apple then displays

```
DOWNLOAD COMPLETE
```

and returns to CP/M command level.

8. This step completes the data transfer procedure. Step 7 must be repeated for each file to be transferred. To transfer more than one file at a time, use the SUBMIT transient program to create a batch file. The DOWNLOAD and UPLOAD programs can also be modified for use with serial communication accessory boards other than those listed in "Program Requirements" at the beginning of this appendix. The source listing for UPLOAD and DOWNLOAD is provided for this purpose.

UPLOAD Source Listing

```
                        ;
                        ;                UPLOAD
                        ;          (C) 1980 MICROSOFT
                        ;
0000    =       BOOT    EQU     0000H   ;Boot system
0005    =       BDOS    EQU     0005H   ;BDOS entry point
005C    =       FCB     EQU     005CH   ;Default FCB
0080    =       BUFFER  EQU     0080H   ;Default buffer address
0100            ;       ORG     0100H   ;Start at TPA
                        ;
0100    C36301  UPLOAD  JMP     ENTRY   ;Jump around these
                        ;
                        ;
                INIT:                   ;Initialize serial port
                        ;
                ; This subroutine should perform any required initialization of the serial
                ; port. If none is required, a 'RET' will do.
                        ;
0103                    DS      32      ;Space for routine
                INPTS:                  ;Input status/read
                        ;
                ; The input status/read routine returns zero in register A if no byte is
                ; available. If a byte is available, the byte should be read and returned in
                ; the A register.
                        ;
0123                    DS      32      ;Space for routine
                OUTPUT:                 ;Send a byte to Apple
                        ;
                ; The output routine should transmit the byte found in the A register to
                ; the Apple through the serial port. All registers including A should be
                ; saved.
                        ;
0143                    DS      32      ;Space for routine
                        ;
                        ;
0163    3A8000  ENTRY   LDA     Buffer  ;Make sure filename is typed
                        ;
0166    B7              ORA     A       ;Is there a non-zero number of
                                        ;characters in the command line?
                        ;
0167    11D701          LXI     D,CMDMSG ;Default to command error
                                        ;message
016A    CACC01          JZ      EXIT    ;Quit
016D    CD0301          CALL    INIT    ;Initialize serial port
0170    0E0F            MVI     C,15    ;Open file command
0172    115C00          LXI     D,FCB   ;Point to FCB
0175    CD0500          CALL    BDOS    ;Open it up
0178    3C              INR     A       ;FF becomes zero
0179    11E501          LXI     D,FNFMSG ;Default to file not found message
                        ;
017C    CACC01          JZ      EXIT    ;No file
                        ;
                ; Send 'R' characters until DOWNLOAD answers
                        ;
```

```
017F  3E52      RDYLP   MVI   A,'R'          ;Send 'R' for 'READY'
0181  CD4301            CALL  OUTPUT         ;Send via serial line
0184  CD2301            CALL  INPSTS         ;Response received?
0187  FE53              CPI   'S'            ;'S' for 'SET'
0189  C27F01            JNZ   RDYLP          ;Then try again
018C  3E47              MVI   A,'G'
018E  CD4301            CALL  OUTPUT         ;Send to DOWNLOAD
0191  110602            LXI   D,WRKMSG       ;Indicate downloading is in
                                             ;progress
                                             ;
0194  CDD201            CALL  PRMSG
0197  0E14      READ    MVI   C,20           ;Read sequential function
0199  115C00            LXI   D,FCB
019C  CD0500            CALL  BDOS           ;Go read 128 bytes
                                             ;
019F  B7                ORA   A              ;Error?
01A0  C2C901    JNZ     EOF                  ;End of file
01A3  218000    TRYAGN: LXI   H,BUFFER       ;Point to the 128 bytes
                                             ;
01A6  0300              MVI   C,0            ;Checksum = 0
01A8  1680              MVI   D,80H          ;Byte count = 128
                                             ;
01AA  7E        LOOP1   MOV   A,M            ;Get character
01AB  CD4301            CALL  OUTPUT         ;Send it (must save A)
01AE  A9                XRA   C              ;Calculate checksum
01AF  4F                MOV   C,A            ;Update it
01B0  23                INX   H              ;Point to the next byte
01B1  15                DCR   D              ;Decrement byte count
01B2  C2AA01            JNZ   LOOP1          ;Keep going until all 128 are sent
01B5  79                MOV   A,C            ;Now send checksum byte
01B6  CD4301            CALL  OUTPUT         ;Send it
                                             ;
                ;Wait for verification: 'G'=GOOD, 'B'=BAD
                                             ;
01B9  CD2301    VFYLP:  CALL  INPSTS         ;Get a character
01BC  FE42              CPI   'B'            ;A bad read?
01BE  CAA301            JZ    TRYAGN         ;Start again
01C1  FE47              CPI   'G'            ;A good read?
01C3  CA9701            JZ    READ           ;Go get next record then
01C6  C3B901            JMP   VFYLP          ;Character must not have been
                                             ;ready
                                             ;
01C9  11F401    EOF:    LXI   D,DONMSG
01CC  CDD201    EXIT:   CALL  PRMSG          ;Output message
01CF  C30000            JMP   BOOT           ;All finished
                                             ;
01D2  0E09      PRMSG:  MVI   C,9            ;Print message function
01D4  C30500            JMP   BDOS
                                             ;
                                             ;
01D7  436F6D6D61CMDMSG:   DB  'COMMAND ERROR$'
01E5  46696C6520FNFMSG:   DB  'FILE NOT FOUND$'
01F4  0D0A55504CDONMSG:   DB  13,10,'UNLOAD COMPLETE$'
0206  55706C6F61WRKMSG:   DB  'UNLOADING....$'
                                             ;
0213            END
```

DOWNLOAD Source Listing

```
                        ;
                        ;               DOWNLOAD
                        ;
                        ;This program works with an Apple Communications Interface or a CCS
                        ;7720A Serial Interface in slot two.
                        ;
                        ;
                        ;
                        ;               (C) 1980 MICROSOFT
                        ;
0000    =       BOOT    EQU     0000H   ;Boot system
0005    =       BDOS    EQU     0005H   ;BDOS entry point
005C    =       FCB     EQU     005CH   ;Default FCB
0080    =       BUFFER  EQU     0080H   ;Default buffer address
                        ;
E0AE    =       COMSTS  EQU     0E0AEH  ;Communication card status
                                        ;location
E0AF    =       COMDAT  EQU     0E0AFH  ;Communication card data—slot 2
E000    =       APPKBD  EQU     0E000H  ;Apple keyboard
                        ;
                        ;
0100                    ORG     0100H   ;Start at TPA
                        ;
                        ;
                        ;
0100    3A8000  DWNLOD: LDA     BUFFER  ;Make sure there's a filename
0103    B7              ORA     A       ;Any characters in command line?
0104    11C001          LXI     D,CMDMSG ;Point to CMD error message
0107    CA8D01          JZ      EXIT    ;Quit
010A    0E13            MVI     C,19    ;Delete file
010C    115C00          LXI     D,FCB
010F    D5              PUSH    D       ;Save PTR: to FCB
0110    CD0500          CALL    BDOS
0113    D1              POP     D       ;Reget PTR: to FCB
0114    0E16            MVI     C,22    ;Make file
0116    CD0500          CALL    BDOS
                        ;
0119    3C              INR     A       ;Check for error
011A    11CE01          LXI     D,NDSMSG ;Get ready to print
                                        ;'NO DIR. SPACE'
011D    CA8D01          JZ      EXIT
                        ;
                ;Wait till UPLOAD sends an 'R'
                        ;
0120    CDA001  RDYLP:  CALL    RDCOM   ;Get a character from
                                        ;communication card
0123    FE52            CPI     'R'     ;'R' for 'READY'?
0125    C22001          JNZ     RDYLP
                        ;
0128    1E53            MVI     E,'S'   ;Get 'S' for 'SET'
012A    CD9301          CALL    WRCOM
                        ;
```

```
012D CDA001   GETGEE:  CALL   RDCOM      ;Wait for 'G' for 'GO'
0130 FE47              CPI    'G'
0132 C22D01            JNZ    GETGEE
                                         ;
0135 21F501            LXI    H,WRKMSG   ;Point to 'DOWNLDNG' message
0138 7E        PRLP:   MOV    A,M        ;Get character
0139 B7                ORA    A          ;Set CC's
013A CA4701            JZ     TRYAGN     ;Go do DOWNLOAD
013D E5                PUSH   H
013E 5F                MOV    E,A        ;Character to register E for
                                         ;CONOUT
013F CDBB01            CALL   CONOUT
0142 E1                POP    H
0143 23                INX    H
0144 C33801            JMP    PRLP
                                         ;
                                         ;
0147 218000    TRYAGN: LXI    H,BUFFER   ;Point to 128-byte buffer
014A 0E00              MVI    C,0        ;Clear checksum
014C 0E81              MVI    C,81H      ;Read 128 bytes + 1 checksum
                                         ;
014E CDA001    LOOP1   CALL   RDCOM      ;Read a byte
0151 77                MOV    M,A        ;Store it
0152 A9                XRA    C          ;Calculate checksum
0153 4F                MOV    C,A        ;Update it
0154 23                INX    H          ;Next byte
0155 15                DCR    D          ;Decrement byte count
0156 C24E01            JNZ    LOOP1      ;Not done--continue
0159 B7                ORA    A          ;Was checksum zero?
015A CA6A01            JZ     GOODRD     ;Things are OK
                                         ;
015D 1E42      BADRD:  MVI    E,'B'      ;'B' for 'BAD'
015F CDBB01            CALL   CONOUT
0162 1E42              MVI    E,'B'      ;Send 'B' to upload
0164 CD9301            CALL   WRCOM
0167 C34701            JMP    TRYAGN
                                         ;
016A 1E2E      GOODRD: MVI    E,'.'      ;Print a period
016C CDBB01            CALL   CONOUT
016F 115C00            LXI    D,FCB      ;Point to FCB
0172 0E15              MVI    C,21       ;Write sequentially
0174 CD0500            CALL   BDOS
0177 1E47              MVI    E,'G'      ;Send UPLOAD 'G' for 'GOOD'
0179 CD9301            CALL   WRCOM
017C C34701            JMP    TRYAGN
                                         ;
                                         ;
017F 3210E0    DONE:   STA    APPKBD+10H ;Clear keyboard strobe
0182 115C00            LXI    D,FCB
0185 0E10              MVI    C,16       ;Close the file
0187 CD0500            CALL   BDOS
018A 11E101            LXI    D,DONMSG   ;All done message
```

```
018D CDB601   EXIT:    CALL   PRMSG     ;Print the message
0190 C30000            JMP    BOOT
                                        ;
0193 3AA330   WRCOM:   LDA    COMSTS    ;Communication card status
0196 E602              ANI    2         ;Check bit 2
0198 CA9301            JZ     WRCOM
0198 7B                MOV    A,E       ;Get character to send
019C 32AFE0            STA    COMDAT    ;Store here
019F C9                RET
                                        ;
01A0 3AAEE0   RDCOM:   LDA    COMSTS    ;Communication card status
01A3 1F                RAR              ;STS bit to carry
01A4 DAB201            JC     READIT
01A7 3A00E0            LDA    APPKBD    ;See if CONTROL-C was typed
01AA FE83              CPI    083H      ;
01AC CA7F01            JZ     DONE
                                        ;
01AF C3A001            JMP    RDCOM     ;No, wait for character
                                        ;
01B2 3AAFE0   READIT:  LDA    COMDAT    ;Get incoming character
01B5 C9                RET
                                        ;
01B6 0E09     PRMSG:   MVI    C,9       ;Print message
01B8 C30500            JMP    BDOS
                                        ;
01BB 0E02     CONOUT:  MVI    C,2       ;Console output
01BD C30500            JMP    BDOS
                                        ;
01C0 436F6D6D61CMDMSG: DB     'COMMAND ERROR$'
01CE 4E6F206469NDSMSG: DB     'NO DIRECTORY SPACE$'
01E1 0D0A446F77DONMSG: DB     '13,10,'DOWNLOAD COMPLETE$'
01F5 446F776E6CWRKMSG: DB     'DOWNLOADING',0
                                        ;
0201                   END
```

Appendix C
SoftCard Version Differences

SoftCard 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

80-column display operation is standard

Up to 59.5K 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, there is no need for either utility.

SoftCard Differences

The Premium SoftCard IIe System has several features that the previous SoftCards do not have. There are also differences in the way the Premium SoftCard IIe System performs I/O functions.

The following features are unique to the Premium SoftCard IIe System.

64K bytes of RAM which can be accessed by either the Z80 or the 6502 microprocessors

Circuitry for a full 80 columns of display

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 IIe 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 IIe internal clocks.

The Premium SoftCard IIe circuit board is physically larger than the previous SoftCards. It is also designed to install into the AUXILIARY connector slot only.

There are no switches on the Premium SoftCard IIe circuit board.

Z80 interrupts to the 6502 are not allowed; there is no circuit path provided by the Apple IIe.

Differences in Software

The Premium SoftCard IIe has a larger TPA (59.5K bytes) for running programs.

Because all memory is contained on the Premium SoftCard IIe circuit board, there is no need to change the size of CP/M. Therefore, the SoftCard IIe package does not include the CPM60 utility program.

There is no I/O Configuration Block (IOCB). Some of the SoftCard IIe 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 DOWNLOAD program is provided on the SoftCard IIe Supplement disk instead of Premium SoftCard IIe Master disk. 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 IIe version of CP/M execute three times faster than programs running under previous SoftCards.

Differences in I/O Operation

The SoftCard IIe 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 IIe 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 IIe 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.

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.