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.

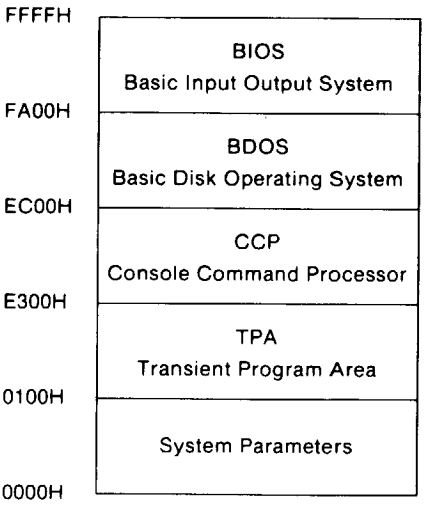

Figure 1.1. CP/M Memory Organization

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.