Software and Hardware Details

PART 2
SOFTWARE AND HARDWARE DETAILS

Chapter 1
Apple II CP/M Software Details

Chapter 2
Apple II CP/M I/O Configuration Block

Chapter 3
Hardware Description

CHAPTER 1
APPLE II CP/M SOFTWARE DETAILS

- **Introduction**
- **I/O Hardware Conventions**
- **6502/Z-80 Address Translation**
- **Apple II CP/M Memory Usage**
- **Assembly Language Programming with the SoftCard**
- **ASCII Character Codes**
- **Interrupt Handling**

Introduction

This chapter deals with the software features that are peculiar to Apple II CP/M, and how these features relate to the I/O hardware installed in the different slots of the Apple. First we will discuss the hardware I/O protocol supported by Apple CP/M. Then we will examine the software support of this hardware protocol: the I/O Configuration Block. For more information on the use of the CP/M operating system, see the "CP/M Reference Manual."

I/O Hardware Conventions

The I/O hardware protocol is identical to that supported by the initial release of Apple PASCAL, with a few exceptions. All standard Apple I/O peripherals are supported, as well as a few others, such as California Computer Systems' 7710A Asynchronous Serial Interface, the Videx Videoterm, and M&R Enterprises Sup-R-Term. Apple CP/M does not support horizontal scrolling on the Apple 24 × 40 video screen.

Apple Peripheral Cards: What Goes Where
Unlike Applesoft and Integer BASIC (but similar to Apple PASCAL), Apple CP/M requires that peripheral I/O cards be plugged into specific slots depending on their functions. For instance, a printer interface card must be plugged into slot one in order to use a printer. When the system is booted, CP/M is able to recognize the presence or absence of certain standard Apple peripheral interface cards. Once the system is booted, I/O is performed by using either the hardware directly or by calling the 6502 software on the card.

Below is a table of the assigned functions for each of the Apple slots, along with the card types that are recognized when plugged into each. (See the list of recognized card types following the table.) Note that unless otherwise noted below, unrecognized cards or empty slots are ignored.

SLOT	VALID CARD TYPES	PURPOSE
0	Not used for I/O	This slot may contain a Language Card or an Applesoft or Integer BASIC ROM card. (the latter are not used by CP/M)
1	types 2,3,4	Line printer interface (CP/M LST: device)
2	input: 2,3,4 output: 1,2,3,4	General purpose I/O (CP/M PUN: and RDR: devices)

3	types 2,3,4	Console output device (CRT: or TTY:) The normal Apple 24 × 40 screen is used as the TTY: device if no card is present.
4	type 1	Disk controller for drives E: and F:
5	type 1	Disk controller for drives C: and D:
6	type 1	Disk controller for drives A: and B: (must be present)
7	any type	No assigned purpose. The SoftCard may be installed in slot 7.

NOTE: The SoftCard may be installed in any empty slot except slot zero.

Below is a list of the I/O peripheral card types that are currently recognized by Apple CP/M.

TYPE	CARD NAME
1	Apple Disk II Controller
2	Apple Communications Interface
	*California Computer Systems 7710A Serial Interface
3	Apple High Speed Serial Interface
	Videx Videoterm 24 × 80 Video Terminal Card
	M&R Enterprises Sup-R-Term 24 × 80 Video Terminal Card
4	Apple Parallel Printer Card

*The CCS 7710A serial interface card is the preferred type 2 card as it supports hardware handshaking and variable baud rates from 110-19200 baud.

6502/Z-80 Address Translation

Because of the memory address translation performed by the hardware on the SoftCard, a particular data byte is not accessed at the same address for both processors. The correspondence of memory addresses between the Z-80 and 6502 is shown below (All addresses are hexadecimal). Use of this table is necessary when translating 6502 BASIC or assembly language software for use with the SoftCard.

Z-80 ADDRESS	6502 ADDRESS	
0000H-0FFFH	$1000-$1FFF	Z-80 location zero
1000H-1FFFH	$2000-$2FFF	
2000H-2FFFH	$3000-$3FFF	
3000H-3FFFH	$4000-$4FFF	
4000H-4FFFH	$5000-$5FFF	
5000H-5FFFH	$6000-$6FFF	
6000H-6FFFH	$7000-$7FFF	
7000H-7FFFH	$8000-$8FFF	
8000H-8FFFH	$9000-$9FFF	
9000H-9FFFH	$A000-$AFFF	
0A000H-0AFFFH	$B000-$BFFF	
0B000H-0BFFFH	$D000-$DFFF	
0C000H-0CFFFH	$E000-$EFFF	
0D000H-0DFFFH	$F000-$FFFF	6502 RESET, NMI, BREAK vectors
0E000H-0EFFFH	$C000-$CFFF	6502 memory mapped I/O
0F000H-0FFFFH	$0000-0FFF	6502 zero page, stack, Apple screen

Apple II CP/M Memory Usage

Here is how the Apple memory is used by Apple CP/M:

6502 ADDRESS	Z-80 ADDRESS	PURPOSE
$800-$FFF	0F800-0FFFF	Apple disk drivers and disk buffers
$400-$7FF	0F400-0F7FF	Apple screen memory
$200-$3FF	0F200H-0F3FFH	I/O Configuration Block.
$000-$1FF	0F000H-0F1FFH	Reserved 6502 memory area – 6502 stack and zero page.
$C000-$CFFF	0E000H-0EFFFH	Apple memory mapped I/O
$FFFA-$FFFF	0DFFAH-0DFFFH	6502 RESET, NMI, and BREAK vectors.
$D400-$FFF9	0C400H-0DFF9H	56K Language Card CP/M (if Language Card installed)
$D000-$D3FF	0C000H-0C3FFH	Top 1K of free RAM space with 56K Language Card CP/M
$A400-$BFFF	9400H-0AFFFH	44K CP/M. (Free memory with 56K CP/M)
$1000-$A3FF	0000H-093FFH	Free RAM (CP/M uses lowest 256 bytes)

Assembly Language Programming with the SoftCard

The Z-80 processor executes all of the 8080 instruction set plus its own set of instructions. You can run software written for either the 8080 or Z-80 processor on the SoftCard. There is, however, a different set of instruction mnemonics for each of the processors.

Included with the standard CP/M utilities are ED, a line oriented text editor; ASM, an 8080 assembler; and DDT, an 8080 machine language debugger. These programs can be used to write and debug 8080 programs.

It is also possible to write 6502 subroutines for use with the SoftCard. The Microsoft Assembly Language Development System is available separately for the development of both Z-80 and 6502 software.

ASCII Character Codes

DEC = ASCII decimal code
HEX = ASCII hexadecimal code
CHAR = ASCII character name

DEC	HEX	CHAR	WHAT TO TYPE
0	00	NULL	ctrl @
1	01	SOH	ctrl A
2	02	STX	ctrl B
3	03	ETX	ctrl C
4	04	ET	ctrl D
5	05	ENQ	ctrl E
6	06	ACK	ctrl F
7	07	BEL	ctrl G
8	08	BS	ctrl H *or* ←
9	09	HT	ctrl I
10	0A	LF	ctrl J
11	0B	VT	ctrl K
12	0C	FF	ctrl L
13	0D	CR	ctrl M *or* RETURN
14	0E	SO	ctrl N
15	0F	SI	ctrl O
16	10	DLE	ctrl P
17	11	DC1	ctrl Q
18	12	DC2	ctrl R

19	13	DC3	ctrl S
20	14	DC4	ctrl T
21	15	NAK	ctrl U *or* →
22	16	SYN	ctrl V
23	17	ETB	ctrl W
24	18	CAN	ctrl X
25	19	EM	ctrl Y
26	1A	SUB	ctrl Z
27	1B	ESCAPE	ESC
28	1C	FS	ctrl [
29	1D	GS	ctrl shift-M
30	1E	RS	ctrl ^
31	1F	US	ctrl _
32	20	SPACE	space
33	21	!	!
34	22	"	"
35	23	#	#
36	24	$	$
37	25	%	%
38	26	&	&
39	27	'	'
40	28	(	(
41	29	)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C

68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[	[
92	5C	\	\
93	5D	]	] (shift-M)
94	5E	^	^
95	5F	_	_
96	60	'	'
97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t

117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B	{	{
124	7C	|	|
125	7D	}	}
126	7E	~	~
127	7F	RUB	

Interrupt Handling

Because of the way the 6502 is "put to sleep" by the SoftCard using the DMA line on the Apple bus, ALL interrupt processing must be handled by the 6502. An interrupt can occur at two times: while in Z-80 mode and while in 6502 mode.

Handling an interrupt in 6502 mode:
Handle the interrupt in the usual way – simply end the interrupt processing routine with an RTI instruction.

Handling an interrupt in Z-80 mode:
Both processors are interrupted when an interrupt occurs in Z-80 mode. Here is the step-by-step process for handling an interrupt while in Z-80 mode:

1. Save any registers that are destroyed on the stack.

2. Save the contents of the 6502 subroutine call address (See Calling of 6502 Subroutines above) in case an interrupt has occurred during a 6502 subroutine call.

3. Set up the 6502 subroutine call address to $FF58, which is the address of a 6502 RTS instruction in the Apple Monitor ROM.

4. Return control to the 6502 by performing a write to the address of the SoftCard (again see Calling of 6502 Subroutines).

5. When control is returned to the Z-80, restore the previous 6502 subroutine call address.

6. Restore all used Z-80 registers from the stack.

7. Enable interrupts with an EI instruction.

8. Return with a RET instruction.

CHAPTER 2
APPLE II CP/M
I/O CONFIGURATION BLOCK

- Introduction
- Console Cursor Addressing/Screen Control
  The Hardware/Software Screen Function Table
  Terminal Independent Screen Functions/Cursor Addressing
- Redefinition of Keyboard Characters
- Support of Non-Standard Peripherals and I/O Software
  Assigning Logical to Physical I/O Devices: the IOBYTE
  Patching User Software Via the I/O Vector Table
- Calling of 6502 Subroutines
- Indication of Presence and Location of Peripheral Cards

Introduction

The I/O Configuration Block contains the information necessary to interface Apple CP/M to the various hardware and software configurations available to the Apple CP/M user. Every Apple CP/M system disk has its own I/O Configuration Block, which is loaded and initialized when the system is booted.

There are five primary functions of the I/O Configuration Block:

1. Console cursor addressing/screen function interface
2. Redefinition of keyboard characters
3. Support of non-standard peripheral devices and I/O software
4. Calling of 6502 subroutines
5. Indication of the presence and location of peripheral cards

Each is detailed in its own section in the following pages.

Note: The CONFIGIO program is used to examine and modify the I/O Configuration Block – See Part 5, "Software Utilities Manual" for more information.

Console Cursor Addressing/Screen Control

Most popular video terminals, including the normal 24×40 Apple screen, can support special features such as direct cursor addressing, screen clear, highlighted text, etc. Apple CP/M applications software such as word processors and business software can easily take advantage of these features.

These advanced screen functions are usually initiated by sending a certain sequence of characters to the terminal. The sequences required to perform a specific screen function are often different for different terminals. Most applications software designed to take advantage of these screen functions can be configured for a number of popular terminals. However, if your terminal is NOT compatible with your software, you must usually write some specialized machine language subroutines to take care of the problem. Since the Datamedia terminal screen function sequences supported by Apple PASCAL and the popular 24×80 plug-in video boards are not considered "popular" by many CP/M applications programmers, they are rarely supported.

Under Apple CP/M, these problems are solved in most cases by translating the functions as they are received, into the corresponding function expected by the terminal hardware. This is achieved by two translation tables: the Software Screen Function Table and the Hardware Screen Function Table, both part of the I/O Configuration Block. Apple CP/M uses the Software Screen Function Table to recognize an incoming screen function sequence, which is then translated to the corresponding sequence found in the Hardware Screen Function Table. This sequence is then sent to the terminal device.

For example: Suppose that you want to use a CP/M screen-oriented word processor (designed to work with a SOROC IQ 120 terminal) with a Videx Videoterm 24×80 video board. The problem: Since the Videoterm board recognizes only the Datamedia type terminal character sequences, it does not recognize the screen function character sequences (meant for the SOROC) that the word processor sends.

To solve this problem, you would use the CONFIGIO utility (see the Software Utilities Manual) to encode the SOROC screen function sequences into the Software Screen Function Table and encode the Datamedia sequences into the Hardware Table. Now when your word processor sends characters to the terminal, they are compared to the SOROC function sequences that have been placed in the Software Screen Function Table. A match means that your word processor is attempting to perform a screen function. Next, the corresponding Datamedia character sequence is taken from the Hardware Screen Function Table and sent to the terminal, where the function is actually performed.

The Hardware/Software Screen Function Table

There are nine screen functions supported by Apple CP/M:

1. Clear Screen
2. Clear to End of Page
3. Clear to End of Line
4. Set Normal (lowlight) Text Mode
5. Set Inverse (highlight) Text Mode
6. Home Cursor
7. Address Cursor
8. Move Cursor Up
9. Non-destructively Move Cursor Forward

The Backspace character (ASCII 8) is assumed to move the cursor backwards, and the Line Feed character (ASCII 10) is assumed to move the cursor down one line.

Screen function character sequences supported by Apple CP/M may be of two forms:

1. A single control character, or
2. Any ASCII character preceded by a single character lead-in.

Screen function sequences longer than two characters are not supported.

The internal format of each of the two 11-byte tables is identical. Below are listed the function number, the hexadecimal address and a description of each table entry.

FUNC. #	SOFTWARE	HARDWARE	DESCRIPTION
	0F396H	0F3A1H	Cursor address coordinate offset. Range: 0-127. If the high order is Ø, 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.
	NOTE: The following rules apply to the screen function table entries below: If the table entry is zero, the function is not implemented. If the entry has the high order bit set, the function requires a lead-in. An entry with the high order bit clear means the function does not require a lead-in.		
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 (low-light) Text Mode
5	0F39CH	0F3A7H	Set Inverse (high-light) Text Mode
6	0F39DH	0F3A8H	Home Cursor

7	0F39EH	0F3A9H	Address Cursor (See above)
8	0F39FH	0F3AAH	Move Cursor Up One Line
9	0F39FH	0F3AAH	Non-destructively Move Cursor Forward

The standard 24 × 40 Apple screen supports all nine functions independent of the Hardware Screen Function Table. However, if a Software Screen Function Table entry is zero, that function will be disabled.

The Hardware and Software Screen Function Tables can be examined and modified with the CONFIGIO program. Use of this program and more information concerning terminal configuration can be found in the Apple CP/M Utilities Reference Manual.

Terminal Independent Screen Functions/Cursor Addressing

Because of the general-purpose nature of the Hardware and Software Screen Function Tables, it is possible to write programs that use the information contained in these tables to perform screen functions. These programs would work with *any* terminal, as long as the Hardware Screen Function Table was set up correctly for the particular terminal. Below is a short segment of 8080 assembly language code that 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. All registers are destroyed.
;               (NK 5/80)
;
;               Equates:
;
BDOS            EQU  0005H          ;CP/M function call address
SXYOFF          EQU  0F396H         ;Software cursor address XY coord.
                                    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 char.
```

```
            MOV   A,M           ;Get the char.
            ORA   A             ;See if a Lead-in is required
            RZ                  ;If the function isn't there, quit
            JP    CONOUA        ;If pos., no
            PUSH  PSW           ;Save char.
            LDA   SFLDIN        ;Get software lead-in char.
            CALL  CONOUA        ;Output char. in A
            POP   PSW           ;Re-get char.
CONOUA:     MOV   E,A           ;Put char. 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 coords
;           in [HL].
;
GOTOXY:     PUSH  H             ;Save coords while we do seq.
            MVI   E,7           ;Do a Cursor Address 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 coords if neg.
            MOV   E,L           :Reverse H&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           ;Re-get offset
            ADD   L
            PUSH  H             ;Save all this
            CALL  CONOUA        ;Output first coord.
            POP   H             ;Restore coords.
            MOV   E,H           ;Output second coordinate
            JMP   CONOUE        ;And return.
```

Notice that the screen function character sequences are determined by the *Software* Screen Function Table in the subroutines above. This is *necessary* for these subroutines to work with the normal Apple screen. Also note that a NUL entry in either Screen Function Table will disable that function on the Apple's 24 × 40 screen.

Redefinition of Keyboard Characters

Some CP/M software requires specific keys for proper operation that are normally unavailable on some keyboards. The Apple keyboard is particularly deficient in this respect. Common characters such as the left square bracket ([), and RUBOUT simply cannot be typed. This problem is solved by the Keyboard Character Redefinition Table found in the I/O Configuration Block.

The function of the Keyboard Character Redefinition table is simple: it redefines any key on the keyboard as any of the ASCII character codes. For example, Ctrl-K could be redefined as the left square bracket. Then when Ctrl-K is typed, the [character appears.

Another somewhat tricky use of Keyboard Character Redefinition is to disable BASIC program termination with Ctrl-C by redefining Ctrl-C as some other character such as NUL. Thus it would be impossible to break out of a BASIC program because it is impossible to type Ctrl-C. (It is also clear from this example that messing around with this table can cause some annoying problems.)

Keyboard redefinition takes place only during input from the TTY: and CRT: devices. (See Assigning Logical to Physical I/O devices below.)

The Keyboard Character Redefinition Table

The Keyboard Character Redefinition Table will support up to six character redefinitions. The table is located at 0F3ACH from the Z-80. Entries in the table are two bytes: the first is the ASCII value of the keyboard character to be redefined, and the second is the desired ASCII value of the character. Both bytes must have their high order bits cleared.

If there are less than six entries in the Keyboard Character Redefinition Table, the end of the table is denoted by a byte with the high order bit set.

Modifications to the Keyboard Character Redefinition Table may be made using the CONFIGIO program. See the "Software Utilities Manual."

Support of Non-Standard Peripherals and I/O Software

The I/O Information Block also provides for the support of non-standard Apple peripherals and I/O software. All of the primitive character I/O functions are vectored through the I/O Vector Table which is contained in the I/O Configuration Block. These vectors normally point to the standard I/O routine located in the CP/M BIOS, but they can be altered by the user to point to his own drivers. Three blocks of 128 bytes each are provided within

the I/O Configuration Block for user I/O driver software. Each of the three 128-byte blocks is allocated to a specific device, and thus to a specific slot, in order to prevent memory conflicts.

ADDR	ASSIGNED SLOT	ASSIGNED LOGICAL DEVICE
0F200H-0F27FH	Slot 1	LST: – line printer device
0F280H-0F300H	Slot 2	PUN: and RDR: – general purpose I/O
0F300H-0F37FH	Slot 3	TTY: – the console device

Most Apple I/O interface cards have 6502 ROM drivers on the card. The easiest way to interface these types of cards to Apple CP/M is to write Z-80 code to call the 6502 subroutines on the ROM. This should be sufficient to interface most common I/O devices to Apple CP/M. (See Calling of 6502 Subroutines below.)

If no card is installed in a particular slot, its allocated 128-byte space can be used for other purposes relating to its assigned logical device. These include lower-case-input drivers for the Apple keyboard, cassette tape interface, etc.

I/O driver subroutines are patched to CP/M by patching the appropriate I/O vector to point to the subroutine. A table of vector locations and their purposes is shown below:

VEC #	ADDR	VECTOR NAME	DESCRIPTION
1	0F380H	Console Status	Returns 0FFH in register A if a character is ready to read, 00H in register A otherwise.
2	0F382H	Console Input vector #1	Reads a character from the console into the A register with the high order bit clear.
3	0F384H	Console Input vector #2	
4	0F386H	Console Output vector #1	Sends the ASCII character in register C to the console device.
5	0F388H	Console Output vector #2	

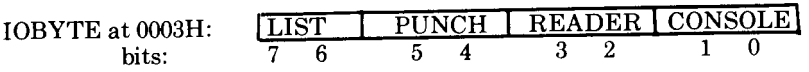
6	0F38AH	Reader Input vector #1	Reads a character from the "paper tape reader" device into register A.
7	0F38CH	Reader Input vector #2	
8	0F38EH	Punch Output vector #1	Sends the character in register C to the "paper tape punch" device.
9	0F390H	Punch Output vector #2	
10	0F392H	List Output vector #1	Sends the character in register C to the line printer device.
11	0F394H	List Output vector #2	

NOTE: During Console Output, the B register contains a number corresponding to one of the nine supported screen functions during output of a screen function. B contains zero during normal character output. B is also non-zero during the output of the Cursor Address X Y coords after executing screen function #7.

Assigning Logical to Physical I/O Devices: the IOBYTE

As explained in the CP/M reference documentation, the IOBYTE can be used to assign logical I/O devices to physical devices. The IOBYTE is changed with the STAT program. See the "CP/M Reference Manual" for more information on changing and using the IOBYTE.

The IOBYTE function creates a mapping of logical and physical devices which can be altered by CP/M programs or with the STAT utility. The mapping is performed by splitting the IOBYTE into four bit fields, as shown below:



The value in each field can be in the range 0-3. The meaning of the values that can be assigned to each field is outlined below:

CONSOLE field (bits 0,1)
0 - CONSOLE is the TTY: device
1 - CONSOLE is the CRT: device
2 - Batch mode – Uses the RDR: device as the CONSOLE input, and the LST: device as the CONSOLE output (BAT:)
3 - User defined CONSOLE device (UC1:)

READER field (bits 2,3)
0 - READER is the TTY: device
1 - READER is the CRT: device
2 - READER is the "paper tape reader" device (PTR:)
3 - User defined READER device #2 (UR2:)

PUNCH field (bits 4,5)
0 - PUNCH is the TTY: device
1 - PUNCH is the "paper tape punch" device (PTP:)
2 - User defined PUNCH #1 (UP1:)
3 - User defined PUNCH #1 (UP2:)

LIST field (bits 6,7)
0 - LIST is the TTY: device
1 - LIST is the CRT: device
2 - LIST is the line printer device (LPT:)
3 - User defined LIST device (UL1:)

Below is a description of the Apple CP/M implementation of the physical devices mentioned above:

TTY: Either the standard Apple screen and keyboard or an external terminal installed in slot 3. This routine vectors through Console Input Vector #1 and Console Output #1. The Console status is always vectored through the Console Status vector.

CRT: Same as TTY:

UC1: User defined console device. This device is vectored through Console Input #2 and Console Output #2.

PTR: A standard Apple interface capable of doing *input* installed into slot 2. If no card is plugged into slot 2, the PTR: device always returns a 1AH end-of-file character. Input from the PTR: device is vectored through Reader Input vector #1. Characters are returned in the A register.

UR1: User defined reader #1. A character read from this device is returned in the A register. This input device is vectored through Reader Input vector #2.

UR2: User defined reader #2. This device is physically the same as UR2:.

PTP: Any standard Apple interface capable of doing *output* installed into slot 2. If no card is plugged into slot 2, the PTP: device does nothing. Output to the PTP: device is vectored through Punch Output vector #1.

UP1: User defined punch #1. The character in register C is output through Reader Input vector #2.

UP2: User defined punch #2. This device is physically the same as UP1:

LPT: The LPT: device is any standard Apple interface card installed into slot 1 capable of doing output. The character in register C is output through List Output vector #1.

ULl: User defined list device. The character in register C is output via List Output vector #2.

The IOBYTE can be changed with the STAT program, or it may be modified from an assembly language program using the CP/M Get IOBYTE and Set IOBYTE (#7 & #8) functions. See "An Introduction to CP/M Features and Facilities" and the "CP/M Interface Guide" in the "CP/M Reference Manual" for more information.

Patching User Software Via the I/O Vector Table

User subroutines can be patched into the I/O Configuration Block with the CONFIGIO program. Any patches made can also be permanently saved onto a CP/M system disk as well as with CONFIGIO.

To create a code file, use ASM to write the driver software, and then use LOAD to create a COM file from the HEX file produced by ASM.

The code file loaded by CONFIGIO must be of a certain internal format. Only one code segment may be patched into the I/O Configuration Block per code file. However, as many vectors in the I/O Vector Table may be patched as desired.

Below is outlined the format of a disk code file to be loaded with CONFIGIO and patched to the I/O Configuration Block:

First byte:	No. of patches to I/O Vector Table to be made.
Next 2 bytes:	Destination address of program code.
Next 2 bytes:	Length of program code.

Repeat for each I/O vector patch to be made:

Next byte:	Vector Patch type – either 1 or 2.
If Vector Patch type = 1 :	
Next byte:	Vector number to be patched. May be from 1-11. (See vector location definitions above)
Next 2 bytes:	Address to be patched into the vector referred by the previous byte. Points into the user's code.
If Vector Patch type = 2 :	
Next byte:	Vector number to be patched. May range from 0-11. (See vector location definitions above)

Next 2 bytes:	Address in which to place the current contents of the specified vector. (May be the address field of a JMP, etc.)
Next 2 bytes:	New address to be placed in the specified vector.
Next:	The actual program code is located after the patch information above. Convention restricts the size of the program code to 128 bytes per slot-dependent block. Use the block appropriate for your application and slot use. (See above)

Below is an example of a program that could be patched into the I/O Configuration Block using CONFIGIO. While it is listed here primarily as a model for writing your own programs, it is useful in its own right with a 24x80 video card or standard Apple video and keyboard, so you may want to enter it for your own use.

Notice how OFFSET is used to allow the program to be ORGed at 0100H.

To patch this program to the I/O Configuration Block, you would:

1. Use the DDT "S" command to enter the program into memory at 100 hex.

2. Use the CP/M SAVE command to save it to disk.

3. Use CONFIGIO option #3 to load the lower case driver into the I/O Configuration Block.

4. Use CONFIGIO option #4 to save the patched I/O Configuration Block to the disk.

If you patch this lower case input routine for your own use, note the following:

This driver defaults in upper case shift lock. The forward-arrow key is used as the shift key. Hit the arrow key once to enter lower-case input mode. Now, all characters typed will be entered in lower case. To shift a letter, hit the arrow key once—don't hold it down. The next character typed will be shifted. To enter shift-lock mode, hit the arrow key twice in a row.

```
;     APPLE CP/M LOWER CASE INPUT ROUTINE
;
; This routine can be assembled using ASM and
; LOAD to produce a file that can be loaded and
; patched into the I/O Confisuration Block with
; CONFIGIO.  It is also intended to be used as
; a model for your own programs.
;
```

```
0015 =          SHFCHR  EQU     21              ;Shift key is the forward-arrow
F3B9 =          SLTTYP  EQU     0F3B9H          ;Slot types table
E000 =          KEYBD   EQU     0E000H          ;Address of Apple keyboard
                ;
0100                    ORG     0100H           ;This is so LOAD will load at 100h
F300 =          ORIGIN  EQU     0F300H          ;Real origin of program
                OFFSET  SET     ORIGIN-LWRCASE  ;Must be added to 16-bit addresses
                ;
0100 01                 DB      1               ;Make one patch
0101 00F3               DW      ORIGIN          ;Destination address of program
0103 3E00               DW      PRGEND-LWRCASE  ;Length of program
                ;
0105 02                 DB      2               ;Patch type 2
                ;
0106 02                 DB      2               ;Patch Console Input vector #1
0107 06F3               DW      OLDINP+OFFSET   ;Place to put current contents of vector
0109 00F3               DW      LWRCASE+OFFSET  ;New contents of vector
                ;
                ; Check to make sure he isn't using an external terminal:
                ;
010B 3ABBF3     LWRCASE:LDA     SLTTYP+2        ;Is there a card in slot 3?
010E FE03               CPI     3               ;Is he using a Com Card as a terminal?
0110 CA0000             JZ      0000            ;Dummy address
0111 =          OLDINP  EQU     $-2             ;Place to put normal input routine addr
                ;
                ; Get a character from the Apple keyboard:
                ;
0113 3A00E0     KBLOOP: LDA     KEYBD           ;See if char available at keyboard
0116 B7                 ORA     A               ;Set condition codes on keybd loc
0117 F208F3             JP      KBLOOP+OFFSET   ;Loop if char not available
011A 3210E0             STA     KEYBD+10H       ;Clear keyboard strobe
011D E67F               ANI     7FH             ;Mask high bit of char
011F 4F                 MOV     C,A             ;Save character in [C]
                ;
0120 0615               MVI     B,SHFCHR        ;Shift character into [B]
0122 213DF3             LXI     H,STATE+OFFSET  ;Point to shift state
0125 7E                 MOV     A,M             ;Get state.
0126 FE01               CPI     1               ;Determine state
0128 79                 MOV     A,C             ;Get typed character into [A]
0129 DA36F3             JC      STATE0+OFFSET   ;Carry set - state 0
012C CA2EF3             JZ      STATE1+OFFSET   ;State 1
                ;
                ; Here if in lower case input mode.
                ; All alphabetic characters are converted
                ; to lower case, unless the shift character is
                ; typed, which enters 'shift next character' mode
                ;
012F B8         STATE2: CMP     B               ;for shift char.
0130 CA32F3             JZ      SETONE+OFFSET   ;It was, set state = 1
0133 FE40               CPI     64              ;It wasn't, so convert all
0135 D8                 RC                      ;alphabetic chars to lower case
0136 EE20               XRI     00100000B       ;This does the conversion
0138 C9                 RET                     ;All done
                ;
                ; Here if in 'shift next character' mode, entered
                ; by typing the shift char once in lower case
                ; input mode.  If shift character is typed again,
                ; upper case shift lock mode will be entered.
                ;
0139 34         STATE1: INR     M               ;Reset state = 2 = lower case mode
013A B8                 CMP     B               ;Hit shift character?
013B C0                 RNZ                     ;Let upper case character go.
013C 35                 DCR     M               ;set state to zero: upper shift lock
013D 35         SETONE: DCR     M
013E C300F3             JMP     LWRCASE+OFFSET  ;Get another character
                ;
                ; Here if in upper case shift lock mode.
                ; Shift character must be typed once to enter lower
                ; case input mode.
                ;
```

```
0141 B8         STATE0: CMP     B               ;Did he type shift char?
0142 C0                 RNZ                     ;Not shift, return upper case char.
0143 3602               MVI     M,2             ;Set state = 2 = lower case input mode
0145 C300F3             JMP     LWRCASE+OFFSET  ;and get another character
                ;
0148 00         STATE:  DB      0               ;Shift state.  Default = upper lock
                ;
                PRGEND:
0149                    END

0100 01 00 F3 3E 00 02 02 06 F3 00 F3 3A BB F3 FE 03
0110 CA 00 00 3A 00 E0 B7 F2 08 F3 32 10 E0 E6 7F 4F
0120 06 15 21 3D F3 7E FE 01 79 DA 36 F3 CA 2E F3 B8
0130 CA 32 F3 FE 40 D8 EE 20 C9 34 B8 C0 35 35 C3 00
0140 F3 B8 C0 36 02 C3 00 F3 00
-
-
-
-
```

Calling of 6502 Subroutines

As discussed in the Hardware Details section of this manual, the 6502 processor is enabled from Z-80 mode by a *write* to the slot-dependent location 0EN00H, where N is the slot location of the SoftCard, Z-80 mode is selected from 6502 mode with a *write* to the same slot dependent location, which is addressed at $CN00 in 6502 mode. (See the 6502 / Z-80 address translation table on page 2-5). Since the SoftCard may be plugged into any unused slot except zero, the location of the SoftCard will vary from system to system.

However, when the system is booted, the location of the SoftCard is determined by CP/M and its address is stored in the I/O Configuration Block. This address is thus available to CP/M software for calling 6502 subroutines. See the "Hardware Details" section of this manual.

Calling 6502 subroutines is a simple matter. The programmer simply sets up the address of the subroutine to be called, and then does a write to the address of the SoftCard explained above. It is also possible to pass parameters to and from 6502 subroutines through the 6502 A, X, Y, and P (status) registers. The 6502 stack pointer is also available after a 6502 subroutine call. Remember that 6502 and Z-80 addresses are not equivalent – See the 6502/Z-80 Address Translation Table on page 2-30.

Z-80 ADDR	6502 ADDR	PURPOSE
0F045H	$45	6502 A register pass area
0F046H	$46	6502 Y register pass area
0F047H	$47	6502 X register pass area
0F048H	$48	6502 P (status) register pass area
0F049H	$49	Contains 6502 stack pointer on exit from subroutine
0F3DEH		Address of SoftCard held here–low byte = 0 followed by high byte of form 0ENH where N is the slot occupied by the SoftCard.

0F3D0H	Address of 6502 subroutine to be called is stored here in low-high order.
$3C0	Start address of 6502 to Z-80 mode switching routine. 6502 RESET, NMI, and BREAK vectors point here. A JMP to this address puts the 6502 on "hold" and returns to Z-80 mode.

NOTE: Locations $800-$FFF are NOT available for use by a 6502 subroutine. The Apple disk driver software and disk buffers reside here.

Special Note for Language Card Users:
When in Z-80 mode, the Language Card RAM is both read- and write-enabled. When a 6502 subroutine is called, the Apple's on-board ROM is automatically enabled, making the Apple Monitor available to the 6502 subroutine. However, the Language Card RAM is write-enabled during a 6502 call, which means that a write to any location above 6502 $D000 will write in the Language Card RAM.

A side effect of read-enabling the on-board Apple ROMs is that the Z-80 memory from 0C000H to 0EFFFH ($D000-$FFFF on 6502) cannot be *read* by the 6502 unless the appropriate Language Card addresses are accessed.

The first of the two available 4K banks for the 6502 $D000-$DFFF area is not used by Apple CP/M.

Below is a short segment of 8080 assembly language code to illustrate the use of the above addresses to call a 6502 subroutine:

```
;
; Subroutine to read the value of
; Paddle zero into register A.
; Demonstrates 6502 subroutine
; calling conventions and parameter
; passing. (NK 5/80)
;
;          Equates
Z$CPU    EQU     0F3DEH    ;Location of SoftCard stored here
A$VEC    EQU     0F3D0H    ;Addr of 6502 sub. to call goes here
A$ACC    EQU     0F045H    ;6502 A register goes here
A$XREG   EQU     0F046H    ;6502 Y register pass area
PREAD    EQU     0FB1EH    ;Apple Monitor paddle read routine
;
PDL:     XRA     A         ;Clear A register
         STA     A$XREG    ;Read paddle zero
         LXI     H,PREAD   ;Get addr of subroutine
         SHLD    A$VEC     ;And store it for 6502 caller
```

```
        LHLD    Z$CPU       ;Get SoftCard addr...
        MOV     M,A         ;Go do it! (Must be a write)
;
;       Execution resumes here after 6502 does a RTS
        LDA     A$ACC       ;A = paddle value.
        RET                 ;All done – return
```

Indication of Presence and Location of Peripheral Cards.

The Card Type Table

When Apple CP/M is booted, each of the slots of the Apple is checked to see if a standard Apple I/O card is installed. This is done by checking to see if there is ROM present in the slot-dependent memory space allocated to peripheral card driver ROMs, and then comparing two signature bytes to those of the standard Apple I/O peripheral cards.

This information is then stored in the Card Type Table, which is located in the I/O Configuration Block. There are seven bytes in the Card Type Table, each corresponding to the seven slots from 1 to 7.

The value of a table entry may range from 0 to 5. The meaning of each value is as follows:

VALUE	EXPLANATION
0	No peripheral card ROM was detected (Usually means that no card is installed in the slot)
1	A peripheral card ROM was detected, but it was of an unknown type.
2	An Apple Disk II Controller card is installed in the slot.
3	An Apple Communications Interface or CCS 7710A Serial Interface is installed in the slot.
4	An Apple High-Speed Serial Interface, Videx Videoterm, M&R Sup-R-Term or Apple Silentype printer interface is installed in the slot.
5	An Apple Parallel Printer Interface is installed in the slot.

This information can be useful to the programmer. For instance, if the third entry (slot 3 – console device) of the Card Type Table is either 3 or 4, a program can assume that the user is using an 80 column external terminal of some kind. In this way, it is possible to write software that configures itself for 40 or 80 column terminals automatically.

The Card Type Table is located at 0F3B9H. The entry for a given slot is located at 3B8H + S, where S is an integer from 1 to 7.

Disk Count Byte
The Disk Count Byte is a single byte equal to the number of disk controller cards in the system times two. This value does not reflect an odd number of disk drives (i.e., only one drive plugged into a controller card).

The Disk Count Byte is located at 0F3B8H.

To Boot a Diskette Without Powering Down
The following program will allow you to boot diskettes from CP/M without having to turn the Apple's power off. This program *is not necessary;* it simply bypasses the power-off step.

1. Use the DDT "S" command to enter the following data at 100 hex.

```
0100 0E 01 CD 05 00 21 77 C7 22 00 30 21 00 C6 22 D0
0110 F3 2A DE F3 C3 00 30
-
```

2. Type Control-C to exit DDT.

3. Type SAVE 1 BOOT.COM

The program is now saved on disk. To use it, just type BOOT and press RETURN. Wait a few seconds, then insert the disk you wish to boot. Press any key to reboot the disk. Your system will reboot exactly as if you had typed PR#6 in Applesoft or Integer BASIC.

CHAPTER 3
HARDWARE DESCRIPTION

- **Introduction**
- **Timing Scheme**
- **SoftCard Control**
- **Address Bus Interface**
- **Data Bus Interface**
- **6502 Refresh**
- **DMA Daisy Chain**
- **Interrupts**
- **SoftCard Parts List**
- **SoftCard Schematic**

This chapter describes the SoftCard itself, both physically and operationally. You won't need this information for normal use of the SoftCard; it is included here to satisfy your curiosity and in case you have an unusual application in which this information would be needed.

Introduction

The Microsoft SoftCard is a peripheral card for the Apple family of computers. The SoftCard contains the necessary hardware to interface a Z-80 microprocessor (contained on the card) to the Apple bus. This permits the direct execution of 8080 and Z-80 programs, including Digital Research's CP/M operating system and all of the programs written to execute in the CP/M software environment.

The SoftCard plugs into any Apple slot except slot zero, and will work in the Apple II, Apple II Plus, or either machine with the Apple Language System. When the Language System is used, the additional memory of the Language Card is made available for use by CP/M or any program operating under CP/M.

Timing Scheme

The Z-80 microprocessor on the SoftCard is synchronized and phase locked to the Apple clocks. This is accomplished by generating a syncopated clock for the Z-80 from the Apple clocks.

During each video refresh period (01), the seven MHz Apple clock is divided down to provide three half clock periods of 135 nsec. The first half-clock is always high, the second always low, and the third always high again. After the end of the third half clock, the signal goes low and stays low until the start of the next 01. This means that the Z-80 clock is low during all of 02 plus a small part of 01. This fourth half-cycle is typically 563 nsec long. (This time is stretched by 69 nsec at the end of each video line.) The effective Z-80 clock rate is 2.041 MHz.

Each kind of machine cycle always contains one memory access period (02). The read/write line is constructed by synchronizing the leading edge of the write transition to the SoftCard clock, thus ensuring that write will only go low during the time that the SoftCard clock is high.

Because all address transitions from the Z-80 occur when its clock is high, they all must occur during 01, when the video update accesses are occurring. Therefore, each 02 cycle has stable addresses for the entire duration of the cycle.

The clock generation is performed by U4 and parts of U1 and U9. The circuit is arranged so that it will still work if the seven MHz clock occurs just prior to the start of 01, or vice-versa. Q1 and the associated components form an analog buffer to provide the high speed switching to within a few tenths of a volt of the supply voltage.

SoftCard Control

The SoftCard is controlled by write commands to the area of memory that normally contains peripheral read-only-memory. It is important to use a write instruction to ensure that the 6502 will not perform two accesses in succession (which would prevent switching back to the 6502).

When the Apple is powered up, the Apple reset signal forces the SoftCard to the off state. The reset signal is synchronized to the Apple clocks to ensure that a write operation cannot be interrupted. The Z-80 is immediately placed in a wait mode, and remains there until the SoftCard is activated.

Upon receipt of a write to the proper area of memory, the SoftCard is activated, and the red LED is turned on. The Z-80 remains in a wait mode until one memory cycle occurs with SoftCard address information. At this point, the Z-80 is released from the wait mode and allowed to run with no further wait cycles required.

Receipt of another write to the same area of memory (this time from the SoftCard itself) will de-activate the SoftCard.

The table below shows the memory addresses used to control the SoftCard as a function of slot location:

SLOT	CONTROL ADDRESSES
1	$C100-$C1FF
2	$C200-$C2FF
3	$C300-$C3FF
4	$C400-$C4FF
5	$C500-$C5FF
6	$C600-$C6FF
7	$C700-$C7FF

Address Bus Interface

The SoftCard address bus is interfaced to the Apple I/O bus through a bank translation circuit. This circuit, consisting of U7, U8, U11, and half of U12, resolves the memory address conflicts that exist between the 6502

architecture and the conventions used by both CP/M and the Z-80 microprocessor. When enabled by S1-1 turned off, the translator adds $1000 to all addresses. This effectively shifts the Z-80 interrupt addresses and CP/M starting addresses out of the 6502 zero page of memory. In addition, addresses in the range of $C000-$EFFF are shifted to allow apparent contiguous memory for CP/M. The table below shows exactly how the translator functions:

Z-80 ADDRESS	APPLE ADDRESS
$0000-$0FFF	$1000-$1FFF
$1000-$1FFF	$2000-$2FFF
$2000-$2FFF	$3000-$3FFF
$3000-$3FFF	$4000-$4FFF
$4000-$4FFF	$5000-$5FFF
$5000-$5FFF	$6000-$6FFF
$6000-$6FFF	$7000-$7FFF
$7000-$7FFF	$8000-$8FFF
$8000-$8FFF	$9000-$9FFF
$9000-$9FFF	$A000-$AFFF
$A000-$AFFF	$B000-$BFFF
$B000-$BFFF	$D000-$DFFF
$C000-$CFFF	$E000-$EFFF
$D000-$DFFF	$F000-$FFFF
$E000-$EFFF	$C000-$CFFF
$F000-$FFFF	$0000-$0FFF

Notice that when the Language Card is installed, the Z-80 can address contiguous memory from $0000-$DFFF, without accessing the 6502 zero page of memory or the Apple peripheral area.

When the translator is disabled (S1-1 turned on) addresses presented by the Z-80 are buffered and appear at the Apple I/O bus unchanged.

All of the address buffers are tri-state buffers capable of sinking or sourcing 24 mA of current. All of the buffers are turned off whenever the SoftCard relinquishes control of the bus. The timing at turn-on and turn-off is arranged to prevent the SoftCard buffers from driving the address bus when the Apple is driving the bus.

The timing of the SoftCard forces all address transitions to occur during the time that the video display (and dynamic memory) is being refreshed by the Apple. Because for each memory access the address lines are stable at the start of the cycle, no wait states are used for memory accesses.

Data Bus Interface

The data from the SoftCard to the Apple (memory writes) is buffered by the same high current driver type as used by the address bus interface. It is only enabled when the following two conditions occur:

1. The SoftCard has control of the bus
2. The SoftCard is attempting to write

When the SoftCard is reading memory, the data is buffered and latched by U15. The outputs of U15 are tri-state, and only enabled when the SoftCard is performing a read. The latch is needed to save data not latched by the Apple (such as the keyboard characters) until the Z-80 can look at it.

Because the SoftCard timing is synchronous and phase locked with the Apple, the timing signals generated by the Z-80 can be used to drive the buffers and the latch.

When an interrupt is recognized by the Z-80 (assuming they are enabled in both hardware and software) the pull-up resistors guarantee that a predictable response is generated for any of the interrupt modes of the Z-80. The byte of data read during an interrupt sequence will be $FF.

6502 Refresh

The 6502 is a dynamic microprocessor, meaning that it requires clock cycles to maintain the contents of its internal registers. The Apple DMA circuitry interrupts operation of the 6502 by turning its clock off. Occasionally, this clock must be turned back on if the 6502 is to remain ready to operate.

This is accomplished by holding the 6502 in a non-ready state (by holding the "RDY" line low) and allowing one memory fetch to be controlled by the 6502. The data fetched is not used by the 6502, and control of the bus reverts back to the SoftCard immediately after the "refresh" memory cycle.

The Z-80 dynamic refresh control lines are used to implement this function. Therefore, the 6502 "refresh" occurs immediately after an op code fetch, and is thus transparent to the SoftCard and the user. No wait cycles have to be added to any Z-80 machine cycles, because the 6502 refresh time is used by the Z-80 to decode the op code. While the 6502 has control of the bus again, the SoftCard address and data buffers are placed in the tri-state mode.

If higher priority DMA devices are allowed to interrupt operation of the SoftCard, the 6502 refresh does not continue. Therefore if it is important to retain the register contents of the 6502 during a DMA cycle, the length of the cycle must be limited to a few microseconds (less than 5).

During a normal mixture of instructions, the 6502 refresh occurs every 4-5 microseconds, well under the data sheet maximum of 40 microseconds. The longest instruction will allow 11.25 microseconds to elapse between refreshes.

DMA Daisy Chain

The Apple DMA daisy chain is fully supported, to the extent that a higher priority DMA device may cause the SoftCard to relinquish control of the bus. Switch S1-2 (when on) enables DMA requests to interrupt the SoftCard. If this switch is on, and the DMA daisy chain input (pin 27) is driven low, the Z-80 will finish the current machine cycle, then the SoftCard will give up control of the bus by raising the DMA control line on pin 22 of the I/O bus. At this time another device may assume control by lowering pin 22. Control must not commence sooner, because the SoftCard buffers will still be driving the bus.

If S1-2 is off, the daisy chain is preserved if the SoftCard is off. When the SoftCard is turned on, the daisy chain output (pin 24) indicates to lower priority devices that DMA activity is in progress. The lower priority devices are therefore locked out of doing any DMA. Likewise, the higher priority devices are also locked out because the SoftCard will not relinquish control of the bus.

Interrupts

Hardware has been included to allow interrupts to be recognized by the Z-80 on the SoftCard as well as by the 6502 microprocessor. When S1-4 is on, the Z-80 will respond to interrupts occurring in the Apple. The interrupt handler program should not attempt to service the interrupt. Instead, control should be passed back to the 6502 for the actual processing. This permits the 6502, which also sees the interrupt, to clear itself of the interrupt status.

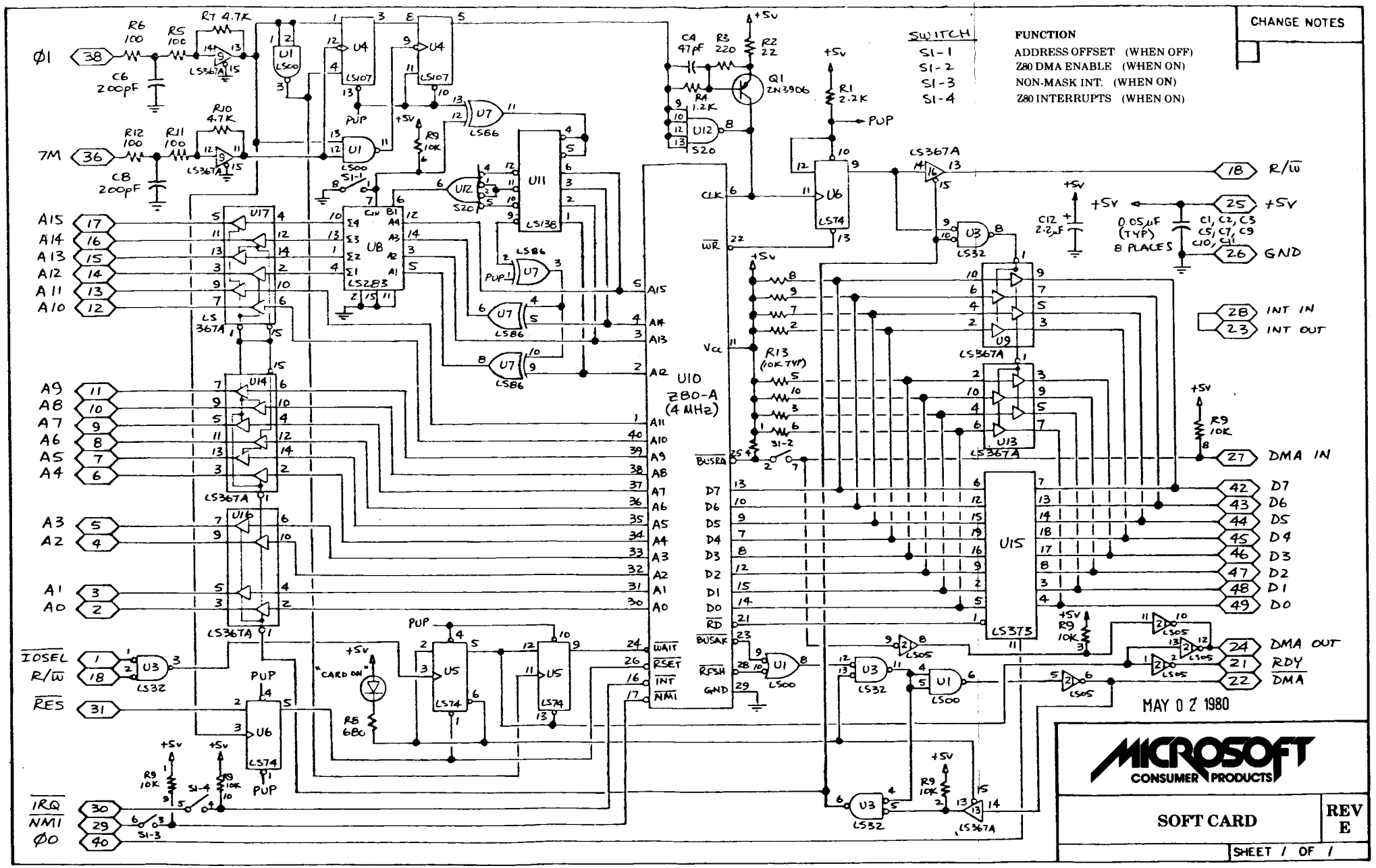
Regardless of the interrupt mode selected for the Z-80, the data byte read during the interrupt sequence will always be $FF. This may be used to vector to a particular memory location for the interrupt handling routine.

Switch S1-3 performs the same function for the non-maskable interrupt.

Parts List SoftCard

Component Identifier	Part No.	Description
U1	74LS00	Quad Nand
U2	74LS05	Hex Inverter
U3	74LS32	Quad Or

U4	74LS107	Dual Flip-Flop
U5	74LS74A	Dual Flip-Flop
U6	74LS74A	Dual Flip-Flop
U7	74LS86	Quad Ex-Or
U8	74LS283	4 Bit Adder
U9	74LS367A	Hex Buffer
U10	Z-80A	Z-80A (4 MHz)
U11	74LS138	Octal Decoder
U12	74S20 (must be "S" part)	Dual Nand
U13	74LS367A	Hex Buffer
U14	74LS367A	Hex Buffer
U15	74LS373	Octal Latch
U16	74LS367A	Hex Buffer
U17	74LS367A	Hex Buffer
Q1	2N3906	PNP Transistor
R1		2.2KΩ, 5%, ¼ watt
R2		22Ω, 5%, ¼ watt
R3		220Ω, 5%, ¼ watt
R4		1.2KΩ, 5%, ¼ watt
R5		100Ω, 5%, ¼ watt
R6		100Ω, 5%, ¼ watt
R7		4.7KΩ, 5%, ¼ watt
R8		680Ω , 5%, ¼ watt
R9		Resistor Pack, 10KΩ,
R10		4.7KΩ, 5%, ¼ watt
R11		100Ω, 5%, ¼ watt
R12		100Ω, 5%, ¼ watt
R13		Resistor Pack, 10KΩ
C1		Capacitor, 0.05 μF
C2		Capacitor, 0.05 μF
C3		Capacitor, 0.05 μF
C4		Capacitor, 47 ρF, 10%, 1000V
C5		Capacitor, 0.05 μF
C6		Capacitor, 200 ρF, 10%, 1000V
C7		Capacitor, 0.05 μF
C8		Capacitor, 200 ρF, 10%, 1000V
C9		Capacitor, 0.05 μF
C10		Capacitor, 0.05μF
C11		Capacitor, 0.05μF
C12		Capacitor, Solid Tant., 2.2 μF, 20%, 35V
"Card On"		Light Emitting Diode
S1		Dip Switch – Quad
	-1	Printed Circuit Card


CHANGE NOTES
SWITCH
SI-1
SI-2
SI-3
SI-4
FUNCTION
ADDRESS OFFSET (WHEN OFF)
Z80 DMA ENABLE (WHEN ON)
NON-MASK INT. (WHEN ON)
Z80 INTERRUPTS (WHEN ON)
U10
Z80-A
(4 MHz)
MAY 0 2 1980
MICROSOFT
CONSUMER PRODUCTS
SOFT CARD
REV
E
SHEET 1 OF 1