CHAPTER XI

TPM-II CONFIGURATION AND ALTERATION GUIDE

Our final chapter, "customizing" your QX-10, will probably surprise you. As you'll see, there are very few applications in which customization is necessary. The QX-10 is a state-of-the-art single-board computer. It contains as many features as all but the most ambitious bus-oriented microcomputers (such as S-100 based systems). Unlike its bus-oriented counterpart, the entire QX-10 system was "known" to its creators when TPM-II was modified to run on it. The TPM-II drivers were designed in concert with the hardware engineers, and they are fully integrated into PIOS.

HOOK, LINE, AND SINKER

At this point you are probably wondering if we forgot about the five expansion slots. Suddenly our single-board world is shattered by the introduction of a "random element". Who knows what kind of expansion boards might be plugged into the QX-10 and expected to work with TPM-II?

Enter the "hook".

Provisions have already been made for integrating unknown cards into TPM-II. These software hooks provide a simple interface for a full contingent of unknown expansion cards.

Of course there are already several expansion cards available for the QX-10. Some of the more popular ones have drivers in the standard TPM-II release. Therefore, you simply plug in the new board and turn on the QX-10. Fairly simple stuff. (Consult your local dealer for a list of the currently supported line, as well as their logical and physical I/O or disk assignments.)

Finally we come to the sinker. As you are undoubtedly aware, the memory in the QX-10 is organized into four banks. As luck would have it, PIOS is spread out over two memory banks. For the unfortunate soul whose application is so unique that none of the interfacing methods described during the remainder of this chapter will work, the best bet is to consult either Rising Star Industries or Computer Design Labs.

OVERVIEW OF INTERFACE METHODS

There are several methods of interfacing with TPM-II. The method of choice will depend on the nature of the expansion device, the level to which it must be integrated into TPM-II, and the limitations placed on it by other system components. Let's explore the possibilities.

Avoidance

A curious way to interface with TPM-II, but a valid method just the same. Certain expansion devices don't need to interface with TPM-II. Instead, they will come supplied with a program that performs all input, output, and control functions. However, in order to successfully integrate such a device into the QX-10, care must be taken that the device or its support software doesn't interfere or conflict with TPM-II.

Any peripheral which uses interrupts, DMA, a memory bank other than Bank 0, or is supported by software that doesn't reside completely within the bounds of the TPA falls into this category. You'll need to read the rest of this chapter to familiarize yourself with the TPM-II interface to make certain that you don't inadvertently interfere with TPM-II.

A User-defined I/O Device

TPM-II supports four logical and four physical I/O devices. The logical devices are identical to those in CP/M: Console, Punch, Reader, and List devices. Of the physical devices, three have built-in drivers: the keyboard/CRT, the serial port, and the printer ports. A fourth, the User device, is undefined. Hooks are provided to allow you to attach a custom driver. The standard TPM-II mechanism described in Chapter 8 can be used to assign this driver to any logical I/O device.

Interrupt Driven Devices

The QX-10 supports fifteen interrupt sources. As part of the power-on initialization the hardware vectors are set to a second set of vectors located in the last page of memory. These vectors may be altered easily to point to user-supplied service routines.

INTERBANK GYMNASTICS

Outside of the realm of the TPA, your QX-10 has no available free memory -- every page is spoken for. However, three pages of memory (E700H - E7FFH, FD00H - FEFFH) can be reclaimed, although to do so will require that you give up one or more of the subsidiary TPM-II functions. Unfortunately, most drivers you might

add won't fit into three pages, which brings us to the topic of interbank gymnastics.

In order to launch an attack on another bank, a "beachhead," or common area, must be established. Although the three banks we just alluded to aren't always suitable for additional drivers, any one of them will serve for this common area. Let's review their functions and characteristics:

Area 1 E700H - E7FFH: Area used by TPM whenever either of the expanded features are desired. It contains "linkage" code to link the function #39 and function #40 TPM LIOS calls to the appropriate drivers. If you don't need the expanded functions, this area is the nicest to use, as it can be loaded on a "Cold Boot". It is located at 2800H through 28FFH in the "SYSGEN" image. You can place the code onto the system tracks, and have it there on boot-up.

Area 2 FD00H - FDFFH: Area used as the entry point for the function call #40 software, i.e., the spooler handler, buffered serial port handler, and clock display handler. This area, as well as area 3, is used during the boot-up procedure, and may be overwritten by an application AFTER the system has come up.

Area 3 FE00H - FEFFH: Area used as the interface into function call #39. This is the expanded bit-mapped video driver. The area is a continuation of area 2, going from 0FE00H through 0FEFFH.

The following routines should help you call subroutines in other banks. The two routines must both be located in the common memory areas (E000H - FFFFH). The XFER routine is JMPed to after the HL register pair has been pushed on the stack, HL loaded with the address on the other bank which you wish to call, and NEWBANK primed with the byte necessary to enable the desired bank. BACK is JMPed to at the end of the subroutine. Note that a separate stack must be established in the common RAM area.

```
XFER:   DI              ;CAN'T STAND INTERRUPTS NOW
        SHLD JMPADR     ;LOAD THE DESTINATION ADDRESS INTO THE
                        ;"PSEUDO CALL" INSTRUCTION
        POP H           ;RESTORE HL
        SSPD STSAV      ;SAVE THE CURRENT BANK'S STACK
        LXI SP,NEWSTACK ;SET UP NEW STACK
        PUSH PSW        ;GET PSW OUT OF THE WAY
        IN 30H          ;DETERMINE WHAT BANK WE'RE CURRENTLY IN
        ANI 0F0H        ;STRIP OFF FLOPPY DISKETTE STATUS IN THE
                        ;LOWER NIBBLE
        STA OLDBANK     ;SAVE IT FOR POSTERITY
        LDA NEWBANK     ;PLOT THE NEW COORDINATES
        OUT 18H         ;..AND HYPERSPACE
        POP PSW         ;RESTORE PSW
        EI              ;ALLOW INTERRUPTS AGAIN
        DB C3H          ;PSEUDO CALL INSTRUCTION
JMPADR:DS 2
                        ;
BACK:   DI              ;DON'T ALLOW INTERRUPTS
        PUSH PSW        ;GET PSW OUT OF THE WAY
        LDA OLDBANK     ;GET THE BYTE TO RETURN TO THE OLD BANK
        OUT 18H         ;SWAP TO ORIGINAL BANK
        POP PSW         ;RESTORE PSW
        LSPD STSAV      ;RESTORE OLD STACK
        EI              ;ALLOW INTERRUPTS AGAIN
        RET             ;BACK TO REALITY
```

ATTACHING A USER I/O ROUTINE

A user-supplied I/O routine can be accessed for any one of the four logical I/O devices by properly setting the I/O byte. (Remember, under TPM-II the I/O byte can only be set using the function call, not by simply writing a new value to 0004H as under CP/M). TPM-II LIOS will branch to a jump table containing vectors to the physical I/O routines. A different routine can be supplied for each logical device, so even though we will call them all USER, they needn't be the same.

If you install a USER routine for any of the logical I/O devices, these vectors should point to routines, such as the one shown above, that provide access to the actual device driver(s), which will usually be on another bank. This jump table resides at 0FF00H - 0FF1EH. It includes two vectors for each logical device (an input and output routine), as well as a vector for the Console Status and the STOP key. The vectors are listed below:

Address	Vector
0FF00	USER CONSOLE INPUT
0FF03	USER CONSOLE OUTPUT
0FF06	USER READER #1
0FF09	USER READER #2
0FF0C	USER PUNCH #1
0FF0F	USER PUNCH #2
0FF12	NOT USED
0FF15	USER LIST OUTPUT
0FF18	USER CONSOLE STATUS
0FF1B	STOP KEY VECTOR

The STOP KEY vector is four bytes in length, instead of the three allocated to the others, which means that an enable interrupts instruction (EI) can be inserted before the vector.

INTERFACING INTERRUPT-DRIVEN DEVICES

The QX-10 contains two 8259A interrupt controllers arranged in a Master/Slave arrangement, which provides for 15 levels of interrupts. Any device may generate an interrupt, whether it's an I/O device, Disk device, or one that has nothing to do with TPM-II. Several of these interrupts are assigned to hardware functions contained within the QX-10. However, a sufficient number are "unassigned" or reserved that each of the five option slots has its own unique interrupt.

When TPM-II is booted, the vectors contained within the 8259A's are initialized to a jump table starting at 0FF20H. If you wish to install an interrupt driver, you must patch a vector

to point to your interrupt service routine. It is strongly suggested that you change the vectors contained in the jump table rather than the ones in the 8259A's.

Priority	Address	Vector
1	0FF20H	POWER FAIL INTERRUPT
2	0FF24H	SOFTWARE TIMER #1
3	0FF28H	COMMON OPTION INTERRUPT #1
4	0FF2CH	COMMON OPTION INTERRUPT #2
5	0FF30H	QX-10 KEYBOARD/RS-232
6	0FF34H	GRAPHICS DISPLAY CONTROLLER INTERRUPT
6	0FF38H	FLOPPY DISK CONTROLLER INTERRUPT
7	0FF3CH	NOT USED (USED INTERNALLY FOR 8259A
8		MASTER/SLAVE INTERFACE)
9	0FF40H	PRINTER PORT
10	0FF44H	OPTION CARD #1 INTERRUPT
11	0FF48H	REAL-TIME CLOCK ALARM
12	0FF4CH	OPTION CARD #2 INTERRUPT
13	0FF50H	OPTION CARD #3 INTERRUPT
14	0FF54H	SOFTWARE TIMER #2
15	0FF58H	OPTION CARD #4 INTERRUPT
16	0FF5CH	OPTION CARD #5 INTERRUPT

You'll notice that each vector is four bytes in length, and is initialized so that each starts with a No Op instruction (00H), followed by a JMP. Please consult the QX-10 Technical Manual for further information about the relationship between the common and unique option card interrupts.

BYPASSING LIOS

At times it is necessary and desirable to access one or more of the physical I/O driver routines (both character and disk I/O) without using the normal function call access. All function calls are routed first through LIOS before the actual driver in PIOS is executed. TPM-II provides a jump table at the beginning of PIOS which gives user programs (as well as LIOS) direct access to the various physical I/O routines.

The address stored in memory locations 0001H & 0002H is used to determine the location of this jump table. Since this is the Warm Boot vector, which is second in the table, 0003H is subtracted from it. In the current version of TPM-II, the address stored in 0001H & 0002H is 0E803H, which places the beginning of the PIOS jump table at 0E800H. This could change with future releases of TPM-II, so it is important that your program access these vectors by reading 0001H and 0002H, subtracting 0003H, and then adding the appropriate offset (provided in the table below), rather than by calling an absolute address.

The first 17 vectors are identical to CP/M versions 1.4 and 2.2, providing complete TPM-II compatibility with programs that utilize this facet of CP/M. In addition, several other vectors are also available, providing access to features that are unique to TPM-II. The conventions used to pass data to and from these PIOS routines are given below:

Data Transfer	Register To Use
Pass an 8-bit value To PIOS	Place value in "C"
Pass a 16-bit value To PIOS	Place value in "BC"
Get an 8-bit value From PIOS	Place value in "A"
Get a 16-bit value From PIOS	Place value in "HL"

Every rule has its exceptions, and here's the exception to the preceding caveats. "SELDK" also returns the disk definition tables in "IX" and "IY". "TINMS" returns the time (in seconds past midnight) in two register pairs, "HL" and "BC".

(See next page)

```
Pios
offset     LABEL                              Description

          0000        $BOOT:     COLD START
          0003        $WBOOT:    WARM START
          0006        $CONST:    CONSOLE STATUS
          0009        $CONIN:    CONSOLE CHARACTER IN
          000C        $CONOT:    CONSOLE CHARACTER OUT
          000F        $LIST:     LIST CHARACTER OUT
          0012        $PUNCH:    PUNCH   CHARACTER OUT
          0015        $READR:    READER CHARACTER IN
                    ;
                    ; DISK        ORIENTED VECTORS
                    ;
          0018        $HOME:     HOME THE HEADS
          001B        $SELDK:    SELECT A DISK
          001E        $STTRK:    SELECT A TRACK
          0021        $STSEC:    SELECT A SECTOR
          0024        $STDMA:    SET THE DMA ADDRESS
          0027        $READ:     DO THE READ
          002A        $WRITE:    DO THE WRITE
                    ;
                    ; CP/M        2.X ADDITIONS
                    ;
          002D        $LISTS:    LIST DEVICE STATUS
          0030        $SECTR:    SECTOR XLATE FOR CP/M GUYS
                    ;
                    ; TPM-II ADDITIONS
                    ;
          0033        $IIOS:     GET IOBYTE
          0036        $AIOS:     SET IOBYTE
          0039        $DATE:     DATE ROUTINES
          003C        $TIME:     TIME ROUTINES
          003F        $TINMS:    GET TIME IN SECONDS
          0042        $INITD:    INITIALIZE SYSTEM FLAGS
          0045        $SETUP:    SET X   & Y INDEX VECTORS
          0048        $CALCK:    LIOS INTERCEPT
          004B        $ERROR:    ERROR   PRINTING ROUTINE
          004E        $TRAP:     ERROR   TRAP HANDLER
```

END CHAPTER 11

APPENDIX A

CP/M TO TPM-II CROSS REFERENCE

CP/M Command	TPM-II Equivalent	Description
DIR	DIR	Type a directory of the current disk.
DIR filename	DIR filename	Check for a file or group of files on the current disk.
DIR d:	DIR d:	Type a directory of disk d:
*	DIR n:	Type a directory of user group n.
*	DIR n:filename	Check for a file or group of files on the current disk in user group n.
*	DIR n:d:filename	Check for a file or group of files on disk d: in user group n.
ERA filename	ERA filename	Erase a file(s) from the current disk.
ERA d:filename	ERA d:filename	Erase a file(s) from disk d:
*	ERA n:d:filename	Erase a file(s) from disk d: in user group n.
REN new=old	REN new=old	Rename one or more files on the current disk. The new filename is to the left of the equals sign and old filename to the right.
SAVE n filename	SAVE n filename	Save n 256 byte blocks as filename on the current disk.
TYPE filename	TYPE filename	Type the contents of filename on the current disk to the console.
USER	USER	Return the current user group for the current disk.
USER n	n:	Set the current user group for the current disk to n:.
STAT	SP	Returns the amount of empty space on the current disk.
STAT d:	SP d:	Returns the amount of empty

STAT filename	*	space on disk d:. Returns the number of bytes, records, and extents of a file(s) on the current disk.

CP/M Command	TPM-II Equivalent	Description
STAT d:filename	*	Returns the number of bytes, records, and extents of a file(s) on the disk d:.
*	LIST	Returns the names, number of bytes, user group, and protection level of all files on the current disk in current user group.
*	LIST d:	Returns the names, number of bytes, user group, and protection level of all files on disk d: in current user group.
*	LIST filename	Returns the name(s), number of bytes, user group, and protection level of a file(s) on the current drive.
*	LIST d:filename	Returns the name(s), number of bytes, user group, and protection level of a file(s) on drive d:.
*	LIST n:	Returns the names, number of bytes, user group, and protection level of all files on the current disk in all user groups.
STAT d:=R/O	*	Set drive d: to read-only.
STAT DEV:	IOMOD	Returns the current logical to physical I/O assignments.
STAT ld:=pd:	IOMOD ld=pd	Assigns physical device pd to logical device ld.
PIP y:=x:filename	PIP y:=x:filename	Transfer a file(s) from drive x: to drive y:.
MOVCPM	*	Relocate the CP/M memory image.
SYSGEN	SYSGEN	Place the DOS on a disk.
ED	*	CP/M Editor.

ASM	*	CP/M Assembler.
LOAD	*	CP/M Loader.
DDT	DDT	Debugger.
SUBMIT	@	Batch command processor.
CP/M Command	TPM-II Equivalent	Description
XSUB	*	Extended Batch command processor.

Definitions:

* = No equivalent command or utility
d: = Drive label
n: = User group number
filename = Legal ambiguous or unambiguous file name

END APPENDIX A

APPENDIX B

QX-10 VIDEO TERMINAL EMULATION

Part of the function of the PIOS bit-mapped video display drives is to convert the characters to be displayed on the QX-10's screen to their corresponding bit pattern, and to turn the appropriate pixels on or off. This allows you to interface with the video display (through the Console output function calls) as if it were a character-oriented display device, similiar to a standard CRT terminal.

To further enhance this compatibility, PIOS also supports the control and escape codes of the Televideo 920 terminal. This emulation has been included because many application programs rely on some degree of display enhancement (high/low intensity, etc.) and/or cursor positioning. The 920 was selected due to its popularity, and hence its wide support and implementation.

The following table lists the control and escape codes currently supported, along with a summary of their function. For a more detailed explanation, you are encouraged to consult a Televideo 920 manual or your application program documentation.

(Table starts on next page)

Code	Function
CTRL-G	Ring Bell
CTRL-H	Cursor Left
CTRL-I	Tab
CTRL-J	Cursor Down
CTRL-K	Cursor Up
CTRL-L	Cursor Right
CTRL-Q	Blink Start/Stop
CTRL-R	Reverse Video Start/Stop
CTRL-S	High Intensity Start/Stop
CTRL-Z	Clear All To Space
CTRL-^	Home Cursor
ESC =xy	Load Cursor Address
ESC T	Erase Line To Space
ESC Y	Erase Page To Space
ESC j	Reverse Video Start
ESC k	Reverse Video Stop
ESC .0	Cursor Off
ESC .1	Cursor On
ESC)	High Intensity Start
ESC (	High Intensity Stop
ESC ^	Blink Start
ESC q	Blink Stop
ESC E	Insert Line
ESC R	Delete Line
ESC ?	Read Cursor Address

Code	Function
ESC l	Underline Start
ESC m	Underline Stop

END APPENDIX B

APPENDIX C

VIDEO BIT DRIVER FUNCTION CALL

The QX-10 has a range of sophisticated video display capabilities. The system utilizes the 7220 video display controller IC, which provides such advanced features as 640 X 400 pixel graphics resolution and multiple windows in a low-cost microcomputer. In addition to the QX-10's native hardware capabilities, TPM-II contains a special video driver that enhances the repertoire to include vector, circle, and arc drawing, multiple character fonts, etc.

The video driver is accessed in the same manner as other TPM-II function calls, i.e., a function call number is placed in Reg C, data and/or parameters are placed in the other registers, and a CALL 0005H instruction is executed. However, due to the sophistication of the video bit driver, its interface is far more complicated than the other TPM-II function calls. For this reason, it has been included here, as a separate Appendix, rather than in Chapter 10. This Appendix should be read in conjunction with the QX-10 Technical Manual (Pub # Q8290001-x) and any manufacturer's hardware documentation on the NEC 7220 Graphics Display Controller IC. We will not repeat the information already covered in those documents, in order to minimize the length of this Appendix and to enhance its readability.

OVERVIEW

Not unlike TPM-II, the QX-10 video bit driver contains its own set of function calls. When you call TPM-II, 39 decimal is placed in Reg C. The bit driver function code (also referred to as a D-opcode, for "drawing opcode") is placed in B, and data and parameters are placed in Reg pairs DE and HL. Status information is returned in Reg A. Either 00H is returned to indicate successful completion, or an error code appears.

Pictures, Windows, And Screens

Before we launch into the specific D-opcodes, three terms should be defined: picture, window, and screen.

Picture: The actual screen image which is made up of 640 pixels (dots) per line horizontally and 400 lines vertically. Each pixel occupies a bit of video RAM storage, for a total of 32,768 bytes for an entire screen. The QX-10 Video board contains 128K of video RAM or the equivalent of four pictures.

Window: The picture is made up of one or two windows. The first window (Window 1) extends from the upper left-hand corner of the screen down to the top of the second window (Window 2). Window 2 starts at the bottom of Window 1 and continues to the last line of the picture. The size of the windows is expressed in lines (a horizontal row of 640 pixels), and the total length of Windows 1 and 2 may not exceed 400 lines (the size of the Picture). Only the starting line number of Window 2 must be specified in order to break a picture into two windows; the size of Window 1 is imputed.

Screens: A user-definable segment of the video RAM. The QX-10 contains sufficient RAM to store 1,638 lines (640 pixels). This RAM may be segmented into a maximum of 16 screens. A screen is defined by its starting address in RAM, expressed in lines, and its length, also expressed in lines. Screens may overlap one another, and do not need to be placed in RAM contiguously, or in any order. The one exception to the above involves screens with the scroll option activated. The actual number of lines in video RAM will be double the number specified to accommodates the scrolling feature. For example, a screen which is specified as being 100 lines in length will actually require 200 lines of RAM if the scroll feature is activated.

From Screen To Picture

So, "How do we actually display something on the monitor?". The first step is to define both the screens and windows you will use. Next, the image to be displayed must be stored in the RAM assigned to one of the screens. Finally, one or more screens must be "attached" to the window(s), causing them to actually be displayed in the picture.

The purpose, then, of the D-opcodes is to accomplish these tasks in an orderly manner. As you will see shortly, the repertoire consists of commands to define screens and windows, fill screens with lines, circles, rectangles, and characters, and to attach various screens to a window for display.

Passing Data And Parameters

Depending on the D-opcode, up to 8 bytes of data and/or parameters can be passed to or from the video bit driver by the calling program. The following Table summarizes the number of bytes and the register assignments. The upper three bits of the D-opcode indicate the number of bytes to be passed, as well as the register convention, also listed in the Table.

**** TABLE C - 1 ****

UPPER BITS	NUMBER OF BYTES	REGISTERS
000	0	D-opcode only
001	1	E
010	2	D & E
011	3	D, E, & L
100	4	D, E, H, & L
101	2 + Field	D & E (Bytes) H & L (Field)
110	2 Fields	D & E (Field 1) H & L (Field 2)
111	D-opcode returns data	

****

CHARACTER VS. GRAPHICS MODE

The QX-10 supports two different display modes: character and graphic. In the graphic mode, the screen must be treated as 256K individual pixels. The character mode, on the other hand, perceives the display as number of rows, each row containing some number of characters. This usually means 25 rows of 80 characters, although the combination of windows and the QX-10's ability to mix characters and graphics on the same screen means that a different format will often be encountered.

One of the unique characteristics of the QX-10 is its ability to display different fonts in the character mode. Unlike many display units, the character generator is not a PROM, but rather RAM. Thus, different character fonts can be loaded dynamically, while retaining the simplicity of the character display mode.

D-OPCODES

The various D-opcodes can be separated into four related groups: those concerned with initializing the QX-10's display hardware; those that define the screens, windows, fonts, and other video parameters; those pertaining to commands that support drawing; and those that deal with "attaching" screens for actual display. The following table lists all of the available D-opcodes by category, along with their D-opcodes.

**** TABLE C - 1 ****

INITIALIZATION

D-Opcode	Name	Function
01	RESET	Resets video display.
02	SET-UP	Sets up the 7220 video display chip.
04	INITIAL	Set screen values to original condition.
08	CADS	Clear all drawing screens.
09	CCSDS	Clear currently selected drawing screen.
24	CLRSCR	Clear drawing screen n. Drawing screen number is passed in register E.
25	SET VECTOR TYPE	E register contains the vector drawing type. 0 = replace 1 = complement 2 = reset 3 = set
26	SET PITCH	The E register contains the number of display memory words in the horizontal direction.

DEFINITIONS

D-Opcode	Name	Function
23	SELSCR	Select drawing screen. Screen number is in register E.
2E	DEFNS	Define screen as non-scrolling. The screen number is in register E.
2F	DEFSS	Define screen as scrolling. The screen number is in register E.
32	SET FONT	Set the current character font. The E register contains the font number (0 - 7).
4E	ATTFNT	Attach font n (stored in the D register) to screen n (stored in the E register).
4F	DDW#2L	Define display window # 2's length. The number of lines is stored as a word in register pair DE.
61	DEFSSA	Define screen n's starting address. The address is stored in register pair DE while the screen number is in register L.
62	DEFSLN	Define screen n's length in lines. The number of lines is contained in register pair DE,

		and the screen number in register L.
92	NEWFNT	Sets up new font parameters. The number of bytes required to store the font set is stored in register pair HL, the font's vertical size (in bits) is stored in register D, and the font's horizontal size (also in bits) is stored in register E.

DRAWING

D-Opcode	Name	Function
03	HOME	Home the cursor.
05	BLANK SCREEN	Blank the screen.
06	UNBLANK	Unblank the screen.
21	ASC	Output the character in Register E.
38	SCRLN	Scroll n lines. Register E contains the number of lines. If the MSB bit is set the direction of the scroll is up, to a maximum of 127. If the MSB isn't set, the direction is down, to a maximum of 127 lines.
3F	CHAR FUNCTION	Special character functions. Register E holds the function number. The definition of each function number is listed in Table C - 2.
41	DRAW CIRCLE	Draw a circle. The DE register pair contain the circle's radius.
42	SET LINE	Set the line pattern. The DE register pair contain the pattern number.
43	CIRCLE ARC	Draw a circle arc. The DE register pair contain the radius of the arc. The D-Opcode must be preceded by a START/END ANGLE call (see below).
81	VECTOR RECTANGLE	The delta X coordinates are stored in register pair DE, while the delta y coordinates are in register pair HL.
82	VECTOR MOVE	The delta X coordinates are stored in register pair DE, while the delta y coordinates are

		in register pair HL.
83	VECTOR DRAW	The delta X coordinates are stored in register pair DE, while the delta y coordinates are in register pair HL.
84	START/END ANGLE	The DE register pair contains the starting angle times 10, and the HL register pair contains the ending angle times 10.
85	DRAW BASIC ARC	Draw a basic arc. The upper nibble of register D contains the direction. The lower nibble of register D plus register E contains the angle of the arc in degrees times 10. Register pair HL contain the radius of the arc.

ATTACHING

D-Opcode	Name	Function
07	PCP	Print current picture.
28	ATTDW1	Attach drawing screen n to window 1. The number of the drawing screen is stored in register E.
28	ATTDW2	Attach drawing screen n to window 2. The number of the drawing screen is stored in register E.

****

**** TABLE C - 2 ****

SPECIAL CHARACTER FUNCTIONS

CODE (HEX)	FUNCTION
01	Write complement block at current location.
02	Cursor on special.
03	Cursor off special.
04	Cursor attach. Cursor moves to last place it was attached.
05	Cursor detach. Leaves cursor in current location, but turns cursor off.
06	Turn overstrike on.
07	Turn overstrike off.
08	Auto-scroll on.
09	Auto-scroll off.
0A	Auto-linefeed/carriage return on.
0B	Auto-linefeed/carriage return off.
0C	Reverse video on.
0D	Reverse video off.
0E	Clear to end of line.
0F	Clear to end of screen.
10	Underline on.
11	Underline off.
12	C-TYPE 'COMPLEMENT' on.
13	C-TYPE 'COMPLEMENT' off.
14	C-TYPE 'RESET' on.
15	C-TYPE 'RESET' off.

****

END APPENDIX C