CHAPTER 5

THE VIDEO DRIVER PROGRAM
(Video Driver Version 3.40 for Valdocs Version 1.18)

INTERFACING to TPM-II

Interfacing to TPM-II is accomplished by a system call to location 0005H, with the bit driver call code located in the C register. The bit driver function code is located in the B register. The D, E, and L registers may contain additional data, depending upon the specified D-opcode operation. (Vdriver drawing operations will be referred to as D-opcodes.)

When control is passed back to the calling program, the A register will contain either a 00, indicating that the operation was successful, or an error code specifying the failing condition will be returned.

GRAPHICS COMPATIBILITY

The D-opcode system was developed to be compatible with the various types of graphic hardware systems in use, and, with some modifications, will adapt to future systems, including the following:

1. Where the host processor and the graphics processor are the same, as in the case of the Epson QX-10.

2. Where the host and graphics processors are independent and communicate via parallel or serial ports.

3. Where the host and graphics processors are independent, but the graphics processor has access to the host memory to retrieve the D-opcode.

BIT DEFINITIONS

Bits 5, 6, and 7 of the operations code have specific meaning and use: they define how many bytes of data will be needed in order to perform particular operations. The binary value of these three bits is the number of bytes of data being passed. The following table indicates in which Z-80 registers the data bytes are passed.

BIT CODE	REGISTERS USED
-765-	
-000-	D-opcode only.
-001-	1 byte in E register.
-010-	2 bytes in D and E registers.
-011-	3 bytes in D, E, and L registers.
-100-	4 bytes in D, E, H, and L registers.
-101-	Reserved (not currently implemented).
-110-	Reserved (not currently implemented).
-111-	D-opcode returns data to caller.

D-OPCODE ONLY

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
0	00	NOP0	NOP	(No operation)
1	01	SETUP	SETUP	Sets up graphics hardware (used by operating system only).
2	02		RESERVED	
3	03	HOME	HOME	Home the cursor.
4	04	INIT	INITIAL	Set screen values to original condition.
5	05	BLANK	BLANK PICTURE	
6	06	UBLANK	UNBLANK PICTURE	
7	07		RESERVED	
8	08	CAS	CLEAR ALL SCREENS	Clears all screens
9	09	CLRSCR	CLEAR CURRENT SELECTED SCREEN	Clears only the current screen.
10	0A	VWAIT	VERTICAL WAIT	Waits for vertical sync pulse to be active.

OP-CODES 11-31, (HEX 0B-1F) are RESERVED.

D-OPCODE + E REGISTER: (1 BYTE)

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
32	20	NOP1	NOP	(No operation)
33	21	CHROUT	CHARACTER OUTPUT	E Register = Character
34	22		RESERVED	
35	23	SELSCR	SELECT SCREEN	E Register = Screen #. 16 screens valued 00-0F
36	24	CLRDS	CLEAR DESIRED SCREEN	E Register = Screen # cleared.
37	25	VTYPE	SET VECTOR TYPE	E Register contains type: E = 0 (replace) E = 1 (complement) E = 2 (reset) E = 3 (set)
38	26	PITCH	SET PITCH	E Register = Pitch
39	27		RESERVED	
40	28		RESERVED	
41	29		RESERVED	
42	2A	BITMOD	BIT-MAPPED MODE	E Register = Screen # .
43	2B	CHRMOD	CHARACTER MODE	E Register = Screen # .
44	2C		RESERVED	
45	2D		RESERVED	
46	2E	DEFNS	DEFINE SCREEN NON-SCROLLABLE	E Register = Screen #.
47	2F	DEFSS	DEFINE SCREEN SCROLLABLE	E Register = Screen # scrollable.

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
48	30		RESERVED	
49	31		RESERVED	
50	32	SELFNT	SELECT FONT	E Register = Font #.
51	33	INITFNT	INITIALIZE FONT	E Register = Font #.
52	34	LDFNT	LOAD FONT	E Register = Font data.
53	35	ENDFNT	END FONT	E Register = Font #.
		OP-CODES 54 & 55, (HEX 36 & 37) are RESERVED.		
56	38	SCRLN	SCROLL LINES	E Register = # of lines scrolled.
		OP-CODES 57-62, (Hex 39-3E) are RESERVED.		
63	3F	CHRFUN	CHARACTER FUNCTIONS	E Register = Character function. See table on character functions.

D-OPCODE + D & E REGISTERS: (2 BYTES)

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
64	40	NOP2	NOP	(No operation)
65	41	CIRCLE	DRAW CIRCLE	D & E Registers = Radius.
66	42	SETPAT	SET LINE PATTERN	D & E Registers = Pattern
67	43	ARC	CIRCLE ARC	D & E Registers = Radius.
		OP-CODES 68-76, (Hex 44-4B) are RESERVED.		
76	4C	ATTSCR	ATTACH SCREEN	D Register = Window # E Register = Screen #
77	4D		RESERVED	
78	4E	ATTFNT	ATTACH FONT	D Register = Font # E Register = Screen #

OP-CODES 79-95, (HEX 4F-5F) are RESERVED.

D-OPCODE + D, E, & L REGISTERS: (3 BYTES)

DECIMAL	HEX	MNEMONIC	DEFINITION	REFERENCE	PARAMETERS
96	60	NOP3	NOP		(No operation)
97	61	DEFSSA	DEFINE SCREEN START ADDRESS		D & E Registers =Starting address. L Register = Screen # .
98	62	DEFSLN	DEFINE SCREEN LENGTH		D & E Registers = Length. L Register = Screen #.
		OP-CODES 99-102, (HEX 63-66) are RESERVED.			
103	67	DEFWIN	DEFINE WINDOW LENGTH		D & E registers = Window Length L Register = Window #.
		OP-CODES 104-127, (HEX 68-7F) are RESERVED.			

D-OPCODE + D, E, H, & L REGISTERS: (4 BYTES)

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
128	80	NOP4	NOP	(No operation)
129	81	VRECT	VECTOR RECTANGLE	D & E Registers = Delta X H & L Registers = Delta Y
130	82	VMOVE	VECTOR MOVE	D & E Registers = Delta X H & L Registers = Delta Y
131	83	VDRAW	VECTOR DRAW	D & E Registers = Delta X H & L Registers = Delta Y
132	84	SETANG	SET START/ END ANGLE	D & E Registers = Start angle H & L Registers = End angle
133	85		RESERVED	
134	86	ABSXY	SET X, Y LOCATION	D & E Registers = Absolute X position H & L Registers = Absolute Y position
135	87	FILREC	FILL RECTANGLE	D & E Registers = Delta X H & L Registers = Delta Y
136	88	FILPAT	SET RECTANGLE FILL PATTERN	D & E, H & L Registers = Pattern

OP-CODES 137-159, HEX 89-9F are RESERVED.

D-OPCODE : (5 BYTES)

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
160	A0	NOP5	NOP	(No operation)

OP-CODES 161-191, (HEX A1-BF) are RESERVED.

D-OPCODE : (6 BYTES)

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
192	C0	NOP6	NOP	(No operation)

OP-CODES 193-223, (HEX C1-DF) are RESERVED.

D-OPCODE RETURNS DATA TO HOST

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
224	E0		RESERVED	
225	E1	RETPS	RETURN PHYSICAL SPECIFICATIONS	B Register = X (X,Y Aspect Ratio) C Register = Y D&E Registers = X Dimension H&L Registers = Y Dimension
226	E2		RESERVED	
227	E3		RESERVED	
228	E4	RSFS	RETURN SELECTED FONT SPECS.	On Return: B & C Registers = Fonts loaded. E Register= Font selected H Register = Vertical. L Register = Horizontal. D Register is reserved.

OP-CODES 229-238, (HEX E5-EE) are RESERVED.

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
239	EF	RETVER	RETURN GRAPHIC DRIVER DATA	On return: C Register = Board type. D Register = Driver type. E Register = Version #. H Register = Revision #. L Register = Modification

DECIMAL	HEX	MNEMONIC	DEFINITION	PARAMETERS
240	F0	RSD	RETURN SELECTED DATA	E Register = Data returned (see special section).
241	F1	RETXY	RETURN X,Y LOCATION	Returns device or cursor X,Y location. D & E Registers = X position. H & L Registers = Y position. On Calling: E Register specifies type of data: Bit# Definition 0 = Nop 1 = Cursor 2 = Light Pen 3 = Reserved

OP-CODES 242-255, (HEX F2-FF) are RESERVED.

SPECIAL CHARACTER FUNCTIONS

E REGISTER CODE HEX	MNEMONIC	FUNCTION
00		NOP (No operation)
01	WCB	Write complement block at current location. (Bit-mapped only.)
02	SCON	Cursor on special. (Bit-mapped only.)
03	SCOF	Cursor off special. (Bit-mapped only.)
04	ATTCUR	Cursor attach (goes to last place cursor was detached).
05	DETCUR	Cursor detach (leaves cursor at current location and turns cursor off).
06	CSETON	C-TYPE 'SET' on (Overstrike on.)
07	CREPON	C-TYPE 'REPLACE'on (Overstrike off.)
08	ASON	Auto-scroll on.
09	ASOFF	Auto-scroll off.
0A	CRLFON	Auto-Carriage Return / linefeed on.
0B	CRLFOFF	Auto-Carraige Return / linefeed off.
0C	REVON	Reverse video on.
0D	REVOFF	Reverse video off.
0E	CLEOL	Clear to end of line.
0F	CLEOS	Clear to end of screen.
10	UNDON	Underline on. (Bit-mapped only.)
11	UNDOFF	Underline off. (Bit-mapped only.)
12	CCMPON	C-TYPE 'COMPLEMENT' on.
13	SECBLK	SECRET and BLINK off.
14	CRESON	C-TYPE 'RESET' on.
15	------	RESERVED
16	ITLON	Italics on. (Character mode only.)
17	ITLOFF	Italics off. (Character mode only.)
18	BNKON	Blink on. (Character mode only.)
19	BNKOFF	Blink off. (Character mode only.)
1A	HION	Highlight on. (Character mode only.)
1B	HIOFF	Highlight off. (Character mode only.)
1C	SECON	Secret on.
1D	SECOFF	Secret off.
1E	STATON	Stationary on.
1F	STATOF	Stationary off.

RETURN SELECTED DATA

E REGISTER CODE HEX	MNEMONIC	DATA RETURNED
01	DWOP	Returns display window # 0 parameters . As defined by DEFSSA D-opcode # 61H & 62H. D & E Registers = Starting address in lines. H & L Registers = Length in lines.
02	DW1P	Returns display window # 1 parameters. As defined by DEFSSA D-opcode # 61H & 62H. D & E Registers = Starting address in lines. H & L Registers = Length in lines.
03	CCXY	Column & Row X,Y position of current cursor. Returns in D & E registers.
04	CCPXY	Absolute pixel X,Y position in screen. (Bit-mapped mode only.)
05	RPITCH	Return memory pitch in E register.
20	RETSCR	Returns currently selected screen # in E register.
21	RETATT	Returns attached screens to display window #0 and display window #1. D Register = Screen # attached to DW #0. E Register = Screen # attached to DW #1. H Register = Mode of DW # 0 L Register = Mode of DW # 1 (Bit-mapped mode = 00) (Character mode = -1)
22	RETFLG	Returns Flags. C Register = Is reserved D Register = XFlags E Register (high nibble) = V type E Register (low nibble) = C-Type H Register = Flag1 L Register = Flag2

E REGISTER CODE HEX	STARTING ADDRESS & LENGTH DATA OF SCREEN RETURNED
E0	Screen # 0
E1	Screen # 1
E2	Screen # 2
E3	Screen # 3
E4	Screen # 4
E5	Screen # 5
E6	Screen # 6
E7	Screen # 7
E8	Screen # 8
E9	Screen # 9
EA	Screen # 10
EB	Screen # 11
EC	Screen # 12
ED	Screen # 13
EE	Screen # 14
EF	Screen # 15

*F0-FF RESERVED FOR FUTURE SCREENS

RETURN FLAGS

FLAG0

BIT#	FUNCTION
7	Cursor on.
6	Cursor detached.
5	Auto-carriage return & linefeed active.
4	Auto-scroll active.
3	'.' (period) active.
2	Escape, 1st / 2nd Flag.
1	'=' active.
0	Escape sequence active.

FLAG1

BIT#	FUNCTION
7	Screen scrollable.
6	Reverse video flag.
5	Underline flag.
4	Font loading active.
3	Secret flag.
2	Highlight flag.
1	Blink flag (character mode only).
0	Italics set (character mode only).

FLAG2

BIT#	
3	Character mode flag.
2	Blinking cursor.
1 & 0	00 = Block style. 01 = Underline style. 10 = Half high/Half size. 11 = Half intensity (bit-mapped only).

C-Type

HEX	FUNCTION
00	Replace (normal mode).
01	Complement.
02	Reset.
03	Set (overstrike).

V-Type

HEX	FUNCTION
00	Replace (normal mode).
01	Complement.
02	Reset.
03	Set (overstrike).

ERRORS IN A REGISTER RETURN CODE

A REGISTER CODE DEC.	HEX	MEANING
-1	FF	Illegal D-pcode function.
-2	FE	Attempted to attach cursor when not detached.
-3	FD	Light pen data not available.
-4	FC	Attempted to select a screen that does not exist.
-5	FB	Window length too large.
-6	FA	Attempted to define a screen that does not exist.
-7	F9	Attempted to define current screen.
-10	F6	Attempted to do a vector rectangle with the delta X, delta Y, or both, having a 0 value.
-11	F5	Attempted circle arc without doing Set Start/End instruction.
-14	F2	Type specified is incorrect.
-19	ED	Attempted to select a font not loaded.
-20	EC	The two character escape sequence invalid. Only valid when using the D-OPCODE system.
-21	EB	The escape "." "(any)" sequence invalid.
-22	EA	Cannot evaluate command.
-23	E9	Start/End angle greater than 3,599 (or 359.9 * ten).
-26	E6	Tried to set or attach a font to a character defined screen.
-27	E5	Attempted a bit-mapped command on a character defined screen. No can do.
-30	E2	Cannot change existing screen to character mode. Either the hardware is bit-mapped only (like the QX-10 Colorboard) or the screen size selected is not modular 16 (divisible by 16).
-41	D7	Window # not valid.

SCREEN HANDLING OPERATIONS

The Valdocs video driver software allows the video screen to be manipulated in various powerful and versatile ways. Let us first define some terms:

1. Picture. The picture that the user sees on the video screen. On the QX-10 the picture contains 640 pixels (dots) horizontally and 400 pixels vertically, for a total of 256,000 pixel points.

2. Window. The picture may be divided into two separate sections called windows. Window 0 starts at the top of the screen and extends downward. Window 1 begins after the last line of window 0, and extends to the bottom of the picture.

3. Screen. Sixteen drawing screens are available to the user. Each screen corresponds to a specific area in graphics memory. A screen may be attached to either (or both) of the display windows. When this is done, anything that has been, or is written to that screen will be displayed on the picture.

4. Character Mode. When a screen has been defined as being in character mode, the data written to the screen memory is the ASCII value of each character to be displayed. These values are then used by the character generator hardware to create the display.

5. Bit-mapped Mode. When a screen is in bit-mapped mode, the actual bit pattern for each portion of the display is stored in the screen memory.

6. Memory Pitch. Pitch is the horizontal size of the display screens, and is measured in 16-bit words. Normally, and initially, the pitch is set at 40 words to match the 640 pixel width of the picture.

7. Video Line. A video line is one row of pixels across the picture. It is also called one horizontal scan line.

Each screen has a parameter table that defines the screen and its current status. Four items in the screen parameter table are required to define the screen: the screen starting address, the screen length, whether the screen is scrollable or non-scrollable, and the screen mode (bit-mapped or character). The rest of the screen parameters are indicators that may change during the use of the screen. The following is a list of all the screen parameters:

Parameter name	Description
Start address	Starting address of the screen in video lines.
Screen length	Number of video lines the screen uses.
Position	Current cursor position.
Font	Font currently in use.
Ctype	Character type in use.
Vtype	Vector drawing type in use.
Flags	Bits which tell the on/off condition of the following:

cursor on/off	cursor detached	auto cr/lf
auto scroll	escape "." active	escape "=" active
escape active	screen scrollable	reverse video
underline	loading font	secret
highlite	blink	italic
stationary	character mode	blinking cursor

Cursor type	Two flag bits that define what kind of cursor will be displayed.

All of these screen parameter values may be defined or changed by the user through the use of the various D-opcode function calls. The screens can e attached to the drawing windows at will, without losing the data or screen parameters of the previously attached screen. Each screen retains all status information, such as cursor position, font size and style, and row and column.

A screen may be written or drawn on when it is not attached to any window. This allows the user to put a drawing or text on a screen and then bring it into view instantly.

The following descriptions are of the D-opcode functions that relate to screen handling operations:

INIT Function number 4

The INIT function sets memory pitch to 40 and sets up all 16 screens to the following initial parameter values:

Parameter	Value or condition
Cursor position	Home
Cursor	Off
Auto cr/lf	On
Auto scroll	On
Reverse video	Off
Underline	Off
Secret	Off
Highlight	Off
Blink	Off
Italics	Off
C-type	Replace
V-type	Replace
Vector line pattern	OFFFFH
Rectangle fill pattern	OFFFFFFFFFFFFFFFH (64 bits)
Selected font	Font # zero

The following chart shows the initial definition of each of the 16 screens for all bit-mapped screens (as used for a color video board).

SCREEN #	START ADDRESS (Video line)	LENGTH (lines)	MODE	SCROLLABLE	RESERVED FOR
0	0	400	Bit	Yes	
1	800	32	Bit	No	
2	832	16	Bit	No	
3	848	128	Bit	No	
4	976	128	Bit	No	
5	1,104	16	Bit	No	
6	1,104	16	Bit	No	
7	1,104	16	Bit	No	
8	1,104	16	Bit	No	
9	1,104	16	Bit	No	
10	1,104	16	Bit	No	
11	1,104	16	Bit	No	
12	1,120	128	Bit	No	SYSTEM
13	1,248	128	Bit	No	MAIL
14	1,376	128	Bit	No	SCHEDULER
15	1,504	128	Bit	No	SPOOLER

Note that screens 12 through 15 are reserved for Valdocs use.

The following chart shows the initial definition of each of the 16 screens for video boards that support character generators (ie, QX-10 without color board).

SCREEN #	START ADDRESS (Video line)	LENGTH (lines)	MODE	SCROLLABLE	RESERVED FOR
0	0	400	Bit	Yes	
1	800	32	Bit	No	
2	832	16	Bit	No	
3	1,331	128	Char	No	
4	1,347	128	Char	No	
5	1,104	16	Bit	No	
6	1,104	16	Bit	No	
7	1,104	16	Bit	No	
8	1,104	16	Bit	No	
9	1,104	16	Bit	No	
10	1,104	16	Bit	No	
11	1,104	16	Bit	No	
12	1,363	128	Char	No	SYSTEM
13	1,379	128	Char	No	MAIL
14	1,395	128	Char	No	SCHEDULER
15	1,411	128	Char	No	SPOOLER

Note that screens 12 through 15 are reserved for Valdocs use.

PITCH Function number 38, 26H

The PITCH function sets the horizontal size of video display memory for all screens. Initially this is set at 40 words to match the 640 pixel width of the QX-10 picture. If the pitch is set greater than 40, the horizontal size of the drawing screens will be greater than the picture size, which makes it possible to put text or draw into an area not normally displayed. The purpose of doing this is for horizontal panning, a feature that will be included in a future release of the video driver software. The minimum pitch is 40 words, and maximum pitch is 128 words. When pitch is changed, all screens must be redefined because the size of a memory line is directly affected by the pitch setting. The pitch value to be set is passed in the E register.

Example: Set memory pitch to 40.

```
MVI E, 40           ;PITCH IN E REG.
MVI B, PITCH
MVI C, 39
CALL 5
```

SETWIN Function number 103, 67H

SETWIN sets the length f the display windows. The number of the window to be set is passed in the L register and the window length in video lines in the DE registers. This function can only be used to set the length of window one and above. The size of window number zero will be 400 minus the combined length of all other windows. If the size of all other windows is set to zero then window number zero will be the whole picture (400 lines). Currently only windows zero and one are available on the QX-10.

Example: Set display window #1 to display eight lines of characters having a 16-bit font height. Since each character line will require 16 video lines, the correct number of video lines for window #1 is 128. Window #0 will have 400-128 = 272 video lines, or space for 17 character lines.

```
LXI D, 128          ;SIZE
MVI L, 1            ;WINDOW #1
MVI B, SETWIN
MVI C, 39
CALL 5
```

DEFSSA Function number 97, 61H

The DEFSSA function defines the starting address of a screen. The starting address is the decimal line number in video memory, and is passed in the DE registers. The screen number being defined is passed in the L register. Normally the starting address of a screen will be greater than the starting address of the previous screen plus the length of the previous screen so that the screens do not overlap. It is possible to define two screens with the same starting addresses, or with starting addresses and lengths assigned so that the screens overlap each other in various ways.

Example: Define screen number 4 to start at video line number 990.

```
LXI D, 990          ;STARTING ADDRESS
MVI L, 4            ;SCREEN NUMBER
MVI B, DEFSSA
MVI C, 39
CALL 5
```

DEFSLN Function number 98, 62H

Define the length of a screen. DEFSLN sets the number of video lines in a screen. The number of lines to assign for each line of characters is the height of the character font being used.

The length of a screen is also used in calculating the starting address of the next screen. If the screen is bit-mapped then the starting address of the next screen is simply the starting address of the current screen plus the length. In character mode it is necessary to reserve two video lines for each line of characters desired. Thus the length of the screen for this calculation would be the number of video lines divided by one half the font height.

If a screen is to be defined as scrollable then the length reserved must be twice that required for the display. The maximum length of any screen is 1638 video lines. If more than 1638 lines are assigned, graphic memory wrapping will occur. The DE registers contain the screen length and the L register the screen number when calling this function.

Example: Set the length of screen number 7 to display and scroll 16 lines of text. The font vertical size is 16 bits. The calculation is: 16 bits per character line * 16 lines = 256 lines.

```
LXI D, 256          ;DE = NR OF LINES
MVI L, 7            ;SCREEN NUMBER
MVI B, DEFSLN
MVI C, 39
CALL 5
```

BITMOD Function number 42, 2AH

The BITMOD function defines the specified screen to be a bit-mapped screen. When in bit mode, the exact bit pattern to be displayed is written to the screen memory. The E register contains the screen number being set to bit mode.

Example: Set screen number 2 to bit-mapped mode.

```
MVI E, 2            ;E = SCREEN NUMBER
MVI B, BITMOD
MVI C, 39
CALL 5
```

CHRMOD Function number 43, 2BH

CHRMOD defines the specified screen to be in the character mode. When in character mode, the ASCII character value of text to appear on the display is written to the screen memory. The screen number to set is passed in the E register.

Example: Set screen number 1 to be in character mode.

DEFNS Function number 46, 2EH

DEFNS defines the specified screen as non-scrollable. No scrolling operations will work. The screen number to set is passed in the E register.

Example: Set screen number 9 to be non-scrollable.

```
MVI E, 9              ;E = SCREEN NUMBER
MVI B, DEFNS
MVI C, 39
CALL 5
```

DEFSS Function number 47, 2FH

DEFSS defines the specified screen as scrollable. All scrolling operations will work. The screen number to set is passed in the E register.

Example: Set screen number 9 to be scrollable.

```
MVI E, 9              ;E = SCREEN NUMBER
MVI B, DEFSS
MVI C, 39
CALL 5
```

CAS Function number 8

The CAS function clears the video memory of all 16 screens. Screens that are defined as character mode are cleared to contain all 20 hex (ASCII space). Bit-mapped screens are cleared to all zero unless reverse video is set, in which case all bits will be set.

Example:
```
MVI B, CAS            ;B = FUNCTION NUMBER
MVI C, 39
CALL 5
```

CLRDS Function number 36, 24H

CLRDS clears the desired screen, in the same manner as the CAS function, described above: in character or bit-mapped mode. The screen number to clear is passed in the E register.

```
Example:  MVI E, 2              ;E=SCREEN # TO CLEAR
          MVI B, CLRDS
          MVI C, 39
          CALL 5
```

SELSCR Function number 35, 23H

The SELSCR function selects the screen that will be drawn to. Only one screen may be selected at a time and all screen parameters defined for that screen will be in effect. The screen selected does not have to be attached to a window, so writing to the screen is not necessarily displayed when being done. The screen number to select is passed in the E register.

Example: Select screen number 10 for drawing.

```
          MVI E, 10             ;E = SCREEN #
          MVI B, SELSCR
          MVI C, 39
          CALL 5
```

CLRSCR Function number 9

CLRSCR clears the currently selected screen according to whether it is a bit-mapped or a character-defined screen, as explained in the CAS function description.

```
Example:  MVI B, CLRSCR         ;B = FUNCTION #
          MVI C, 39
          CALL 5
```

ATTSCR Function number 40, 28H

ATTSCR attaches the screen specified in the E register to the display window specified in the D register. If the screen being attached is the same size as the display window, then all of the previous display in that window will be replaced by the data in the newly attached screen. If the screen being attached is smaller than the window, then the bottom of the display window will show the data in the graphics memory immediately following the screen. If the screen is scrollable, part of the data shown at the end would be from the scrolling mechanism. If the screen being attached is larger than the display

window, the bottom portion of the screen will not be displayed.

Example: Attach screen number 11 to display window number 0.

```
MVI E, 11           ;E = SCREEN # TO ATTACH
MVI D, 0            ;D = WINDOW 0
MVI B, ATTSCR       ;B = FUNCTION #
MVI C, 39
CALL 5
```

HOME Function number 3

Positions the cursor to the upper left corner of the selected screen.

Example:
```
MVI B, HOME         ;B = FUNCTION #
MVI C, 39
CALL 5
```

BLANK Function number 5

BLANK turns off the picture (makes the entire display blank). Striking any key or calling the UBLANK function turns the picture back on.

Example:
```
MVI B, BLANK        ;B = FUNCTION #
MVI C, 39
CALL 5
```

UBLANK Function number 6

UBLANK turns the screen on after it has been turned off with the blank function.

SCRLN Function number 56, 38H

If the screen selected has been defined as scrollable, then the SCRLN function will scroll the screen by the number of lines specified in the E register. The screen will scroll up if the value passed is positive in the range from 1 to 127, or down if a negative value from -1 to -128 is passed. The scroll will be the number of lines of characters specified.

Example: Scroll the currently elected screen up six lines.
```
MVI E, 6            ;E = # LINES
MVI B, SCRLN
MVI C, 39
CALL 5
```

Some functions that pertain to screen handling, such as auto-scroll, reverse-video, and cursor control, are handled by functions CHRFUN, RSD, and RETXY in the character and return data section of this manual.

VECTOR DRAWING OPERATIONS

Vector drawing operations are used to draw straight or curved lines on the video screen. These operations require functions to set starting and ending points, and to determine the length, direction, and curvature of the lines to be drawn.

The video screen consists of a number of tiny dots, called pixels, which may be lighted or dark to form the characters or other patterns you see on the screen. The picture is 640 pixels wide and 400 pixels high, and each pixel has a corresponding bit in the video memory space which has been defined for the screen. By writing ones and zeros to the bits in the video memory, the corresponding pixels on the screen are illuminated or dark.

In order to provide reference points for line drawing operations, a coordinate system of XY axes, and a zero or home position for the screen must be defined. Home for the video driver is the pixel in the top left corner of the screen, located at XY position 0,0. The absolute X axis extends horizontally from home to the right side of the screen, and is numbered from 0 to 639. The absolute Y axis extends from home down the left side of the screen to the bottom, and is numbered from 0 to 99. Each XY point of this coordinate system represents one screen pixel (and one bit of video memory). Note that the positive Y direction is defined as down, rather than up, as in the standard Cartesian coordinate system.

Another reference point used in drawing is the cursor position at the beginning of the draw operation. Since all drawing distances, directions, and angles are referenced to the cursor position, a sub set of coordinate axes (x,y) is used. These axes are dimensioned the same as the absolute XY axes, but the origin is at the current cursor position rather than at home. For the rest of this document, when the origin is mentioned it is the origin of the subcoordinate axes (small x,y) at the cursor position that is being referenced. For the vector drawing operations, movement in the coordinate axes is specified as a change in the x direction and a change in the y direction. The amount of these changes is called "delta x" and "delta y". These are passed to the video driver routines as sign-extended 14-bit values in one or more of the Z-80 register pairs. Since only 14 bits are used, the maximum values passed are plus/minus 16,384.

The following paragraphs detail the operation and use of the vector drawing and related functions contained in the video driver:

ABSXY Function number 134, 86H

The ABSXY function sets the origin (cursor position) of the subcoordinate axes to any absolute (X,Y) location on the absolute axes. The X and Y values are passed to the function as two 16-bit values, the X value in the DE register pair and the Y value in the HL registers.

Example: This code would set the origin to the center of the screen. Point (320,200) absolute.

```
LXI D, 320          ;delta x
LXI H, 200          ;delta y
MVI B, ABSXY        ;function #
MVI C, 39
CALL 5
```

VMOVE Function number 130, 82H

VMOVE moves the origin to a new position specified as the point (x,y) relative to the current origin. The 16-bit values x and y are passed in the DE and HL register pairs.

Example: This code would move the origin to a new point, 100 pixels to the right and 10 pixels down from the current origin.

```
LXI D, 100          ;delta x
LXI H, 10           ;delta y
MVI B, VMOVE
MVI C, 39
CALL 5
```

VDRAW Function number 131, 83H

VDRAW draws a line on the screen from the origin to the point (x,y) which is passed in the DE and HL register pairs. The line drawn will include the point of origin, but not the point (x,y) that was passed. Upon completion of a VDRAW, the origin will be moved to the next pixel past the end of the line drawn. (Point (x,y)).

Example: Draw a line from the origin, upward and to the right, to a point 50 pixels over and 50 pixels up.

```
LXI D, 50           ;delta x
LXI H, -50          ;delta y
MVI B, VDRAW
MVI C, 39
CALL 5
```

VRECT Function number 129, 81H

VRECT draws a rectangle which is delta x pixels wide and delta y pixels high. The origin is left where it started. Delta x and delta y are passed in DE and HL respectively.

Example: Draw a rectangle which is 100 pixels wide and 75 pixels high. The origin is at the upper left corner.

```
LXI D, 100                    ;delta x
LXI H, 75                     ;delta y
MVI B, VRECT
MVI C, 39
CALL 5
```

CIRCLE Function number 65, 41H

This function draws a circle whose center is the origin. The radius of the circle is the positive 14-bit value passed in the DE register pair. The origin is not moved.

Example: Draw a circle with a radius of 150 pixels.

```
LXI D, 150                    ;DE = RADIUS
MVI B, CIRCLE
MVI C, 39
CALL 5
```

SETANG Function number 132, 84H

Function SETANG sets the starting and ending angle of the arc to be drawn by the ARC function. The starting and ending angle values are 16-bit integers equal to the desired angle values in degrees, times ten. The start angle is passed in DE, and the end angle in HL. The angles passed may not be less than zero or greater than 359.9 degrees. The use of these angles is further explained in the ARC function descriptions.

Example: Set the starting angle to 20 degrees, and the ending angle to 35.5 degrees.

```
LXI D, 200                    ;20= * 10
LXI H, 355                    ;35.5= * 10
MVI B, SETANG
MVI C, 39
CALL 5
```

ARC Function number 67, 43H

ARC draws an arc of the circle with a radius equal to the 16-bit value passed in the DE register pair, with its center at the origin. SETANG must first be used to set the start and end angles to draw from and to. These angles are the number of degrees of arc, from the point on the negative y axis where the circle would intersect the axis, measured in the clockwise direction. This is a polar coordinate system where zero degrees is north (or up) and 90 degrees is east (or right).

Example: Draw a 120 degree arc from 250 degrees to 10 degrees. The radius of the arc is 150 pixels.

```
LXI D, 2500          ;250= *10 (START)
LXI H, 100           ;10= * 10 (END)
MVI B, SETANG        ;SET THE ANGLE
MVI C, 39
CALL 5
LXI D, 150           ;DE = RADIUS
MVI B, ARC
MVI C, 39
CALL 5
```

SETPAT Function number 66, 42H

SETPAT is used to specify the line drawing pattern that will be used by all of the vector draw operations. A 16-bit pattern value passed in the D & E registers is placed into a pattern register in the graphics controller hardware. If the pattern bits are all ones then the lines drawn will be the usual solid lines but the user may alter the pattern to create any combination of dots and spaces desired. The pattern setting operation applies to the currently selected screen, and does not alter the pattern that is set for any other screen.

The first time a drawing routine writes a bit into the screen memory, bit 0 of the pattern is used to determine whether a 0 or a 1 is to be written. The next bit written uses bit 1 of the pattern, etc. After bit 15 of the pattern has been used, the operation continues, starting over again at bit 0. This process can also be viewed as always using bit 0 of the pattern register, and then rotating the register bits to the right by one bit after each write.

Note that the determination of writing a one or a zero bit to the screen may be modified by the vector drawing type (VTYPE).

VTYPE Function number 37, 25H

VTYPE is used in vector drawing operations only. It specifies one of four possible modifications to be performed upon the pattern data as it is written to the screen memory. The bits written depend on the VTYPE, the data in the pattern register, and the data already at the screen locations to be written. VTYPE modes are selected by the value placed in the E register. The four possible values are:

0 - Replace	2 - Reset
1 - Complement	3 - Set

As each bit in screen memory is to be written, it is modified by the value of the next bit in the pattern register according to the following rules:

REPLACE: The value of the bit in the screen memory is made the same as the value of the bit in the pattern register.

COMPLEMENT: If the pattern bit is zero, the screen bit is left unchanged. If the pattern bit is one, the screen bit is complemented.

RESET: If the pattern bit is zero, the screen bit is not changed. If the pattern bit is one, the screen bit is made zero.

SET: If the pattern bit is zero, the screen bit is not changed. If the pattern bit is one, the screen bit is made one.

FILPAT Function number 136, 88H

FILPAT sets up an 8-bit by 8-bit square pattern to be used by the FILREC function in drawing a filled rectangle. Eight bytes of data are passed in two successive FILPAT calls, and these bytes are put into a pattern register which can be visualized as an 8 deep push-down, pop-up stack. Upon initilization, the register is filled with all OFFH bytes. To set a new fill pattern, FILPAT is called with the pattern bits in the DE and HL register pairs. The registers are pushed down on the stack (register) in a E D L H order. To load another 4 bytes, the FILPAT function is called again, and the first four bytes are pushed down as the E D L and H registers are again pushed onto the stack. To make further changes, call FILPAT again, and the first four bytes are pushed out the bottom, the second four are now the bottom four, etc.

FILREC Function number 135, 87H

FILREC draws a rectangle which is filled with the pattern set up by the FILPAT function. The size of the rectangle will be delta x pixels by delta pixels. Delta x is passed in DE and delta y in HL. (See the example under the VRECT function description.)

If the eight bytes in the pattern register are all OFFH, the rectangle drawn will be solid (all pixels within the boundaries of the rectangle will be illuminated). Otherwise the pattern seen will depend upon the quadrant of the current subcoordinate (x,y) axes the rectangle is drawn in. If the rectangle is drawn in the +x, +y quadrant, the pixels will be turned on or left off according to the way the bytes of the pattern register are shown in figure 6.

To see how the fill pattern register bytes modify the screen pixels for each of the other three quadrants, rotate figure 6 so that the filled rectangle shown is in the quadrant desired. Then reassign the axes back to normal--the one pointing down is the positive y axis, etc. The placement of the bytes and bits shown in the fill pattern are now correct for this quadrant.

If the rectangle to be filled is larger than the 8-by 8-bit size of the fill pattern then the pattern repeats itself vertically and horizontally. The bits actually written to the screen are further odified by the VTYPE mode that has been set. (See the description of the VTYPE function for further explanation.)

FONT AND STYLE DESIGN

A font will be defined as any single set of characters, e.g., normal text, italics, or graphics. One or more of these fonts (up to a total of ten) may be combined into a "style." The standard style is HASCI.STL, consisting of the standard HASCI fonts in addition to several others, which are selected by various shift and typestyle keys. Style is selected via the Style key.

When the Style key is selected, a menu of various styles will appear. Each style will have a 128-byte style header, whose main purpose is to make a style appear when the Version program is run. The first three bytes are an absolute jump to zero, 0C3h 00h 00h. The next byte, offset 3, is a start of text indicator, 0AAh, which is followed by 80 bytes or less of text, delimited by a null. Byte 070h contains the number of fonts in the style. Bytes 071h to 073h contain the letters STL. The header ends at 07Fh.

Each font begins with 16 bytes of data followed by text. Individual fonts will be of the form *filename*.FNT. Fonts will not appear under the Version program. A utility called MATRIX will construct each individual font and allow for modification and storage.

The first two bytes of each font are the length of data that must be sent to the video driver, which includes all data except the header. The length is stored in the order of low byte, high byte.

Bytes 2 and 3 decimal are the size of the characters in the fonts. Byte 2 is the X value, which is usually 8. As Valdocs stands now, this byte must be 8. The next byte is the Y value, usually 16. The compressed, superscripts, and subscripts are all, at present, various 8 by 8 fonts. Normal text is 8 by 16.

The next byte in the sequence is the type byte, which has an offset of 4 decimal. This byte describes the characteristics of the font according to the following table:

NORMAL	COMPRESSED
0 normal text	8 normal compressed
1 bold	9 Reserved
2 italics	10 italics compressed
3 bold-italics	11 Reserved
4 subscript	12 subscript compressed
5 superscript	13 superscript compressed
6 graphics	14 graphics compressed
7 system graphics	15 system graphics compressed

Byte offset 5 is the font device type, which is used to determine for which device the font is designed. A 00h in this byte means it is a screen font. An FX-80 or FX-100 printer font is selected by a 01h in this byte. Byte six is undefined.

The next data byte, offset 7, contains the number of the style in which the font is contained. A 00H is defined as the HASCI style and an FFH is defined as not belonging to a style. This byte would be set when a style is built out of several fonts. Style numbers 0 through 128 are reserved. The numbers 128 to 254 are user-defined.

The last two bytes contain character codes. The first, byte offset 8, contains the value of the first character in the font. This byte is usually space character, a 20H. Byte 9 contains the value of the last character in the font, usually a 127 decimal.

The next record starts the vector table, which is a list of pointers into the character font. Each vector points at the horizontal width byte of the character it represents. The horizontal width byte is followed by the character itself. The vector is calculated from the beginning of the vector table. There are 96 vectors in a normal font. The first vector points at the first character. The vector is in the order of low byte, high byte.

The font starts just after the vector table, at offset 192 decimal. Normally, each font contains 96 characters, the first starting just after the vector table. The first byte of each character is the horizontal width of that character, which allows varible width characters for line justification in the text editor. The next few bytes form the character itself. Each assigned bit of a byte is a dot on the screen. An 8 by 16 font would be 16 bytes of character, plus one byte for the horizontal width, for a total of 17 bytes. In hexadecimal, the letter "A" of the HASCI font would look like this: 08 1C 22 41 41 41 41 7F 41 41 41 41 41 00 00 00 00. Bit 0 is sent to the screen first, so the Hex bit pattern is reversed from the way the character would appear. The following depicts the character diagrammatically (08 is the width):

	num. value	character
1	1C	..***...
2	22	.*...*..
3	41	*.....*.
4	41	*.....*.
5	41	*.....*.
6	41	*.....*.
7	7F	*******.
8	41	*.....*.
9	41	*.....*.
10	41	*.....*.
11	41	*.....*.
	41	*.....*.
	00	....
	00	....
	00	....
	00	....

The font generally follows the ASCII character set in both uppercase and lowercase. The first character is the space (20H). The position of a character would be calculated with the formula:

(*character value* - 32) * 2 = vector location. Remember that the vector points at the horizontal width byte, not the first row of dots. A font that is 17 dots tall by 8 dots wide (8 by 17) would consist of 18 bytes, one width byte, and seventeen 8-dot bytes. A font that is wider than 8 bits is handled differently. In a 9 by 16 font, the first 16 bytes would contain the front 8 dots of the character; the next 16 bytes would contain the remaining dots.

We'll use the example of the letter "A" moved right one position. If this were an 8-bit wide font, the right leg of the letter "A" would be lost. Here is how the letter would appear, moved right, in a 9 by 16 font: 09 70 88 04 04 04 04 FE 04 04 04 04 04 00 00 00 00 00 00 01 01 01 01 01 01 01 01 01 01 00 00 00 00. As you can see, increasing the width by one bit doubles the size of the font, but increasing the width by 8 dots sill only doubles it. The diagram below illustrates this example of the letter "A", shifted two places to the right, in a 9 by 16 font. Remember that bit 0 is sent first.

09 is the width

Num.		Divider		Num.
1	0E	....***.!.	00	17
2	11	...*...*!.	00	18
3	20	..*.....!*	01	19
4	20	..*.....!*	01	20
5	20	..*.....!*	01	21
6	20	..*.....!*	01	22
7	3F	..******!*	01	23
8	20	..*.....!*	01	24
9	20	..*.....!*	01	25
10	20	..*.....!*	01	26
11	20	..*.....!*	01	27
12	20	..*.....!*	01	28
13	00	........!.	00	29
14	00	........!.	00	30
15	00	........!.	00	31
16	00	........!.	00	32