INTERACT™ Graphics Language Manual

Version 4.0

VMI

INTERACT™ Graphics Language Manual

Version 4.0

Vermont Microsystems, Inc.
One Main Street, POB 236
Winooski, Vermont 05404

October 1984
VM 2001 001-07

While, to the best of our knowledge, this manual contains accurate information and graphics, it is subject to change without notice. VMI cannot be held responsible for any errors, omissions, or misinformation that may appear in this manual. Changes made to the information contained in this document will be incorporated in new editions of this publication.

Table of Contents

Table of Contents (Cont.)

Table of Contents (Cont.)

Appendices

Tables and Illustrations

1
Introduction

This manual introduces the reader to INTERACT, explains the main architectural features of the language, and serves as a programming reference. Section 2 describes the environment provided to execute INTERACT commands. Section 3 introduces the function and use of the various INTERACT graphics primitives. Section 4 supplies individual INTERACT command descriptions and syntax rules. Section 5 describes the operation of each of the possible system interfaces to INTERACT.

This manual encompasses all versions of INTERACT. Footnotes and text notations indicate which sections or commands apply to which version of INTERACT.

2
Graphics Environment

2.1 - Coordinate Space

A two-dimensional cartesian system serves as the coordinate space for INTERACT commands. Each coordinate contains an **x**-component and a **y**-component. The **x**-component indicates displacement along an axis parallel to the bottom of the display screen; the **y**-component corresponds to displacement along an axis parallel to the left edge of the screen. Positive values for **x** and **y** indicate right-hand and upward displacement respectively. Both **x**- and **y**-components appear within INTERACT as two's complement 16-bit integers. Therefore, both **x**- and **y**-displacement values range from -32,768 to +32,767. We refer to this **x,y** system as the "virtual" coordinate space since it is entirely addressable but not entirely physically implemented in memory. Refer to Figure 2.1 for more detail while reading the next several sections.

2.2 - Image Memory

The image memory, composed of actual pixel buffers, physically implements a selected subset of the virtual coordinate space. Only graphics command output which falls within the image memory has potential to display to the screen. To position image memory in virtual coordinate space, place the desired center coordinate into the coordinate origin register (CREG 3). Thus, if the coordinate 0,0 appears in CREG 3, the image memory centers horizontally and vertically about the coordinate 0,0. The actual extent of the image memory depends on the amount of pixel RAM available in the graphics processor. If dual image memories become available, they both center about the coordinate origin (CREG 3).

For other hardware installations, reconfigure INTERACT using appropriate commands immediately following cold starts. Refer to Section 4.2 for additional details.

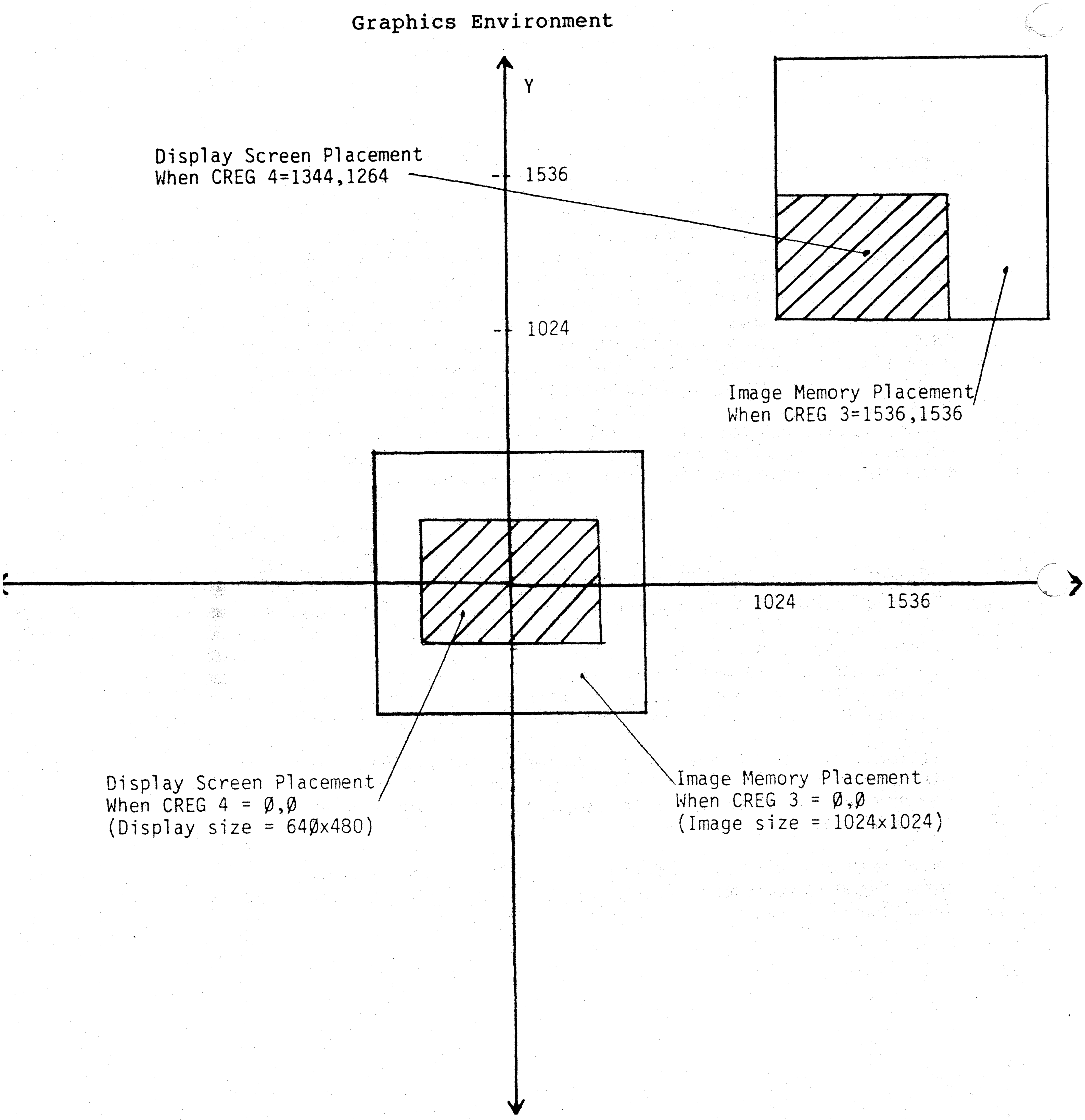


Figure 2.1 : Coordinate Environment

2.3 - Display Screen

The display screen presents image data scanned from the image memory. The display screen information can come from any of the image memories, if available, and, with the screen origin register (CREG 4), can "pan" relative to the selected image memory. The screen origin register specifies the **x,y** coordinate of the pixel to appear at the center of the display screen. Therefore, if the content of CREG 4 differs from that of CREG 3, the display screen will offset vertically and/or horizontally.

2.4 - Clipping

INTERACT graphic output falling outside the image memory is clipped; only graphic output which falls within the current clipping window writes to the image memory. In INTERACT Version 2.0, the boundary of image memory forms the clipping window. Version 3.0 allows the user to define several clipping windows and window formats and to move between windows during a session. Refer to Section 3.4.3 for further explanation.

2.5 - Current Point

Most INTERACT commands use the "current point" to implement their respective functions. The current point register, CREG 0, denotes the starting, or center, point for the generation of a primitive. Coordinate registers 5 and 6 each store the coordinates of one crosshair. Placing the contents of CREG 0 into either register displays that crosshair on the screen at the current point. The current point may lie anywhere in virtual image space.

2.6 - Current Value

All draws to image memory access the current color stored in value register VREG0. Use the VALUE command to change the current drawing color.

2.7 - Coordinate Registers

The coordinate registers (CREGs 0 to 63) provide temporary holding areas for coordinate values. The INTERACT software defines specific functions for 13 of the CREGS, reserves 7 for future definition, and leaves 44 available to the user for applications programming. The CLOAD command stores coordinate values within a specified CREG. Use the READCR command to determine the contents of a CREG. Move the contents of CREGs from one CREG to another with the CMOVE command. The CADD and the CSUB commands perform addition and subtraction operations respectively on the contents of named registers. Appendix B lists the default values for the CREGs. Those CREGs specifically defined by INTERACT follow:

CREG	Name	Description
0	Current Point	Starting, or center, point for graphics primitives
1	Reserved	
2	Locator position	Coordinate of the locator device
3	Coordinate Origin	Coordinate of the center of image memory in virtual space
4	Screen Origin	Coordinate of the pixel at the center of the display screen
5	Crosshair 0	Coordinate of crosshair 0
6	Crosshair 1	Coordinate of crosshair 1
7	Text endpoint	End of string coordinates for TEXT1 and TEXT0 (0,1)
8	Locator Adjustment	Coordinate calibration factor for locator hardware
9,10	Clipping boundary	Current clipping window coordinates
11,12	Device boundary	Coordinates of the rectangle used by the printer driver and the digitizing tablet
13-19	Reserved	
20-63	Unassigned	

2.8 - Value Registers

The value registers (VREGs 0 to 15) serve as temporary holding areas for pixel values. The INTERACT software assigns specific functions to 7 VREGS and leaves 9 for use in applications programming. The command VLOAD stores pixel values into VREGs, while READVR queries the contents of a VREG. Move the contents of VREGs to other VREGs with the VMOVE command. The VADD and VSUB commands allow addition and subtraction operations respectively using the contents of the registers. Appendix B lists the default values for the VREGS. The VREGS specifically defined by INTERACT follow:

VREG	Name	Description
0	Current Value	Pixel value used by all graphics primitives
1	Crosshair 0 Color	Pixel value for crosshair 0
2	Crosshair 1 Color	Pixel value for crosshair 1
3	Area Fill Mask	Pixel mask for random area fills
4	LUT Mask	Value mask for color lookup
5	Text background color	Background color for text
6	Bit Plane Mask	Color mask used by all graphics primitives Contents are logically AND'd with current value before drawing
7-15	Unassigned	

2.9 - Color Look-up Tables

The color look-up tables (LUTs) hold the color values available for drawing. A red, green, and blue intensity level combine to display a single color. LUT commands alter the contents of the tables. Use these commands to change hues or intensities assigned to any index. Reprogramming the LUTs can also change existing colors on the screen. The default LUTs for a color system follow an HLS color model. For a list of these values, refer to Appendix D.

2.10 - Monochrome Look-up Tables

INTERACT defaults to a one-to-one correlation for 8-bit monochrome LUTs where the index value equals the entry value, that is index 7 contains the value 7, etc. A 4-bit system still uses eight bits of output. In this case the 16 entries for monochrome LUTs use evenly spaced values:

Entry	Value
0	00H
1	11H
2	22H
.	.
.	.
.	.
F	0FFH

To redefine LUT values in a monochrome system, user the LUTG command.

2.11 - Power-up Screen

After a power-on reset or a DSPSIZ command, INTERACT draws its power-up screen. This screen allows the user to visually check for proper color channel connections and monitor adjustments. The three color system shows blocks of the three colors unsaturated as well as white, black, and gray. The monochrome system displays the gradations of black to white of the gray scale.

2.12 - Video Generation

The following section describes the video generation process, controlled by the video scanner of the graphics board. This description presents the capabilities of INTERACT. Refer to Figure 2.2 for further illustration. For the following discussion, refer to the table of variables listed below:

	Board Product	
	VM-8850A	VM-8851
bit planes (**bp**)	4	8
simultaneous colors (**sc**)	16	256
bits/color in each LUT (**bc**)	4	4
color palette (**cp**)	4096	4096

The video generation process begins when the video scanner reads a new pixel value from image memory. The pixel value consists of **bp** bits, each read from one of the **bp** bit-planes in the image memory. Next, the pixel value serves as a simultaneous index into the three look-up tables (LUTs). The pixel value selects one of **2bp** entries in each of those three tables, resulting in an ability to display **sc** simultaneous colors. The output values from each of the three LUTs represent the red, green, and blue intensities required to compose the target dot. Since the tables consist of **bc** bits for each of the three colors, the **sc** simultaneous colors are selected from a color palette of **2bc+bc+bc** or **cp** values. The **bc** bit digital color values from the look-up tables are converted to analog intensities in high-speed D/A converters before passing to the video monitor. Refer to the appendices for the default values of the LUTs. The look-up table programming synchronizes to VSYNC so that the palette selections may change "on the fly". During a series of INTERACT commands sent to the graphics board to change the LUT entries, the first command delays execution until the advent of vertical blanking.

The surface functions manipulated by LUTMSK and SURFAC work by reprograming the hardware LUTs. Neither scheme affects the values which are written to display RAM, but both affect the colors which are displayed on the screen. This is accomplished by altering the values in the hardware LUTs in a fashion which is transparent to the user.

LUTMSK works by disabling particular bit planes specified in its **mask** parameter. To disable a bit plane, set the corresponding bit in **mask** to 0. For example, a **mask** of 00001011 would cause the value 00001110 to be displayed as the color represented by value 00001010. Masking is handled before any specified surface priority scheme.

SURFAC allows for the definition of a surface priority scheme in which certain bit planes are assigned priority over other bit planes. Bit planes are assigned priority in the order in which they appear in the surface parameters of SURFAC. If a pixel's value has any bits set in a priority surface, then all of the bits in the non-priority surface are considered to be zero. For example, the following sets up two surfaces:

```
SURFAC 2 0FH 0F0H
```

With this scheme, a pixel of value 42H (01000010B) would be displayed as a pixel of value 02H, since the presence of a set bit in the lower nibble (higher priority) of the pixel value overlays any value in the higher nibble (lower priority). The display may be considered as two separate surfaces in which any color (except value 0) in the higher priority surface "overlays" any color in the lower priority surface.

2.13 - Elements of State

While the result of each INTERACT command depends on the values of its associated parameters, the graphic output may also depend on the current values of the elements of state (see Appendix E). The elements of state which influence each command are detailed in the "Affected by" section of each command description. The elements of state which are influenced by each command are detailed in the "Affects" section of each command description.

Graphics Environment

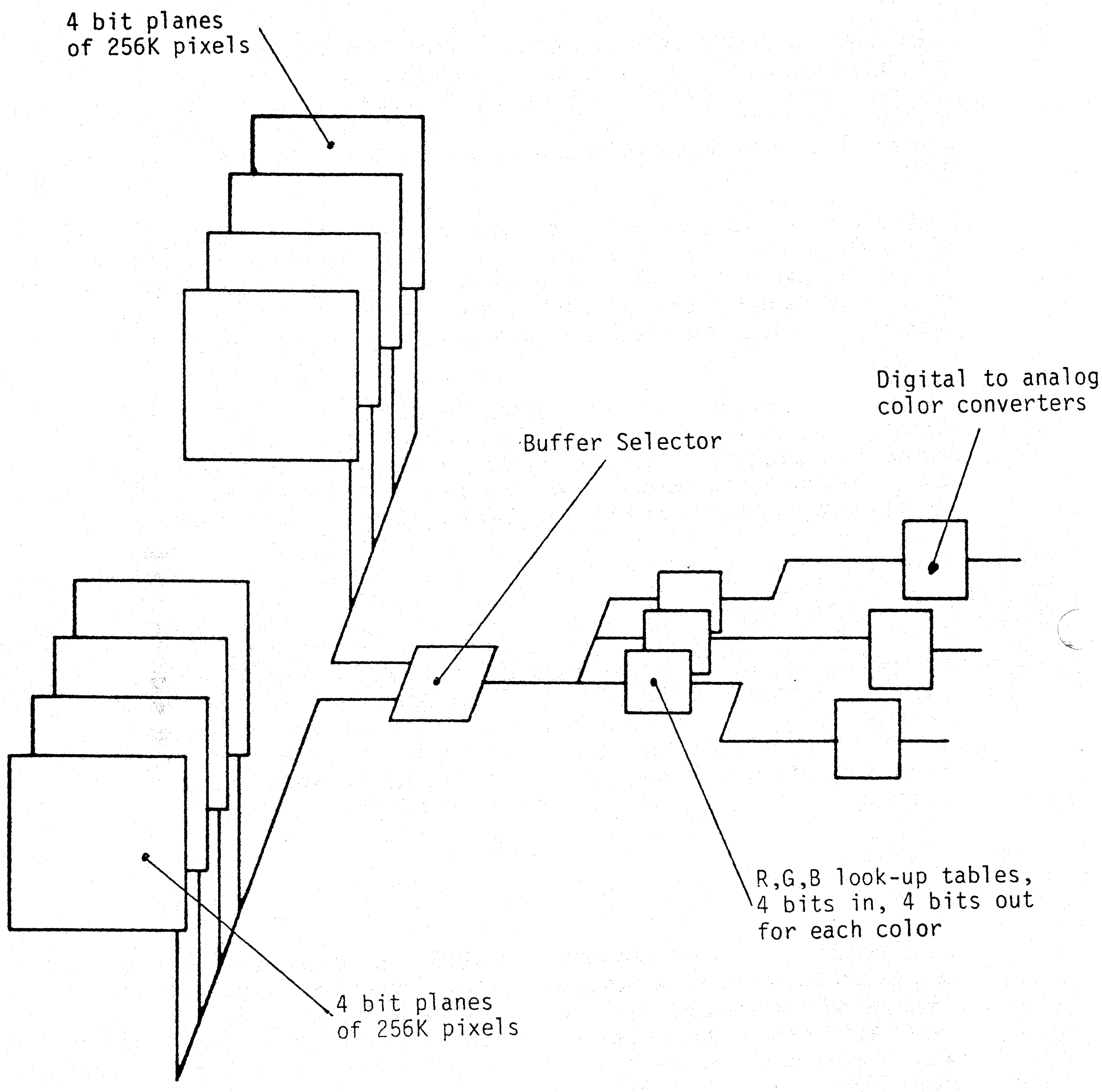


Figure 2.2 : Video Generation

3
General Description

3.1 - Drawing Primitives

Drawing primitives create basic geometric shapes in image memory. Certain display control commands affect draws to the screen or image memory. Refer to Section 3.4 for more information about those commands.

3.1.1 - Moves

Move commands update the current point location stored in coordinate register CREG 0. Change the current point by specifying absolute coordinates (MOVABS) or relative displacement (MOVREL), or by indirectly using absolute coordinates stored in registers (MOVI).

3.1.2 - Points

The POINT command, the simplest INTERACT graphics primitive, places a single pixel of given value anywhere in the image memory. POINT will place a pixel of the value contained in VREG 0 (current color) into image memory at the absolute coordinate contained in CREG 0 (current point.)

3.1.3 - Vectors

Use the vector commands to draw lines. The draw absolute (DRWABS) command will draw a vector in the current value (VREG 0) from the current point (CREG 0) to the x,y point specified by the command parameters. An "absolute" vector defines the endpoints as x,y coordinates. In the draw indirect (DRWI) command, also an absolute vector operation, the parameter specifies a CREG containing the endpoint coordinate x,y. The draw relative (DRWREL, DRW2R, and DRW3R) commands, however, draw a vector which begins at the current point but ends at a particular dx,dy offset from the current point. All vector commands update the current point to the last pixel drawn. This update method facilitates the drawing of concatenated vectors. INTERACT clips a vector as though the line continues off the screen toward the specified endpoint. The DRWABS, DRWI, DRWREL, DRW2R, and DRW3R commands draw line patterns determined by the VECPAT command.

3.1.4 - Linear Forms

The rectangle commands draw right-angled, four-sided figures into image memory. The rectangle relative (RECREL) command draws a rectangle where the coordinate contained in CREG 0 defines one corner coordinate. The parameters dx and dy indicate the relative displacement of the corner diagonally opposite from the current point. The rectangle (RECTAN) command draws a rectangle with one corner located at the current point and the diagonally opposite corner identified by the absolute x,y parameters. The rectangle indirect (RECTI) command also draws an "absolute" rectangle where a specified CREG contains the opposite corner coordinate.

The polygon commands draw a multisided polygon defined by its vertices. A single command can produce any specified number of polygons, each defined by a respective vertex list. The absolute command (POLYGN) interprets its parameters in absolute coordinates. INTERACT connects each vertex to the following coordinate with a vector drawn in the current color. The final named coordinate connects to the initial coordinate, completing the polygon. The polygon relative command (POLYRL) also connects the vertices in the order specified. Each vertex, however, lies at a particular dx, dy distance displaced from the current point (CREG 0). Both polygons will draw "degenerate" shapes, that is, one where one side crosses another side of the same polygon creating multiple enclosed spaces.

3.1.5 - Non-Linear Forms

The CIRCLE command draws a circle defined by a center point and a radius. The center of the circle will lie at the current point. The command defines the radius of the circle in virtual dimensions. The circle indirect (CIRCI) and circle x,y (CIRCXY) commands draw a circle defined by the current point as its center and a specified coordinate to lie on its circumference. The CIRCXY command names the circumferential point in its parameters; the CIRCI command obtains that point from an identified coordinate register.

The ARC command draws arcs. The center of curvature for the arc lies at the current point. The parameters provide the value for the radius of curvature, as well as the starting and ending angles for the arc. These angles reference the current point, drawing counter-clockwise (positive values) from an imaginary line which extends horizontally to the right of the current point. INTERACT interprets the angular specifications as integer degrees employing modulo-360. Refer to Figure 3.1 for an example of an ARC command specification.

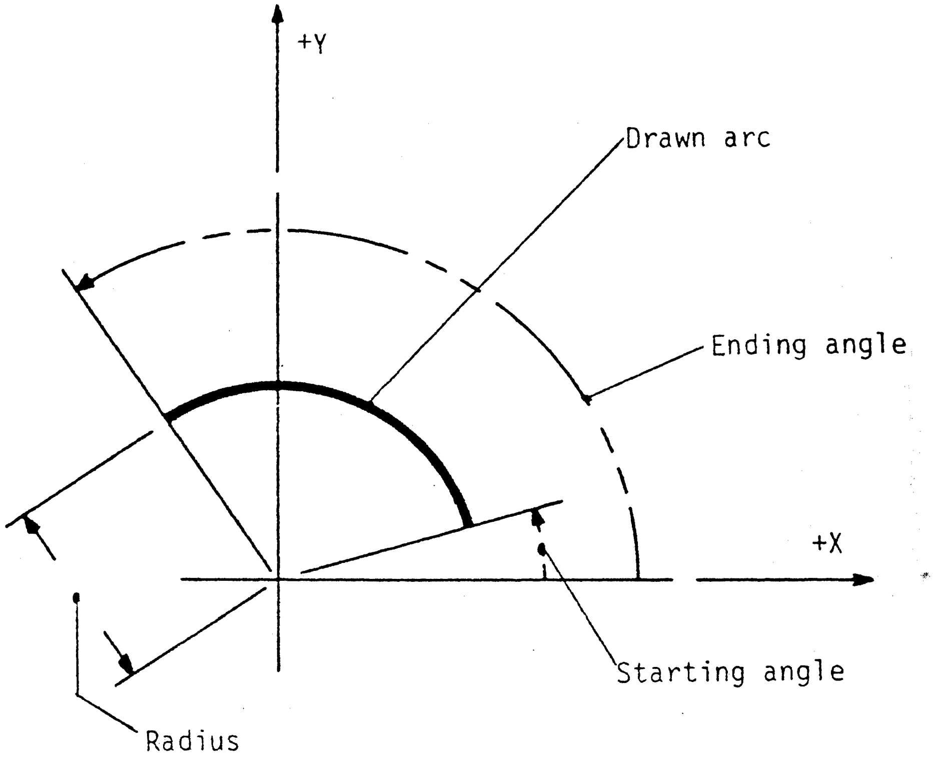


Figure 3.1 : ARC Definition Example

3.1.6 - Flood

The FLOOD command sets all pixels in the current update buffer to the current pixel value. The update parameter of the most recently executed BUFFER command specifies the current update buffer.

3.1.7 - Text

Draw text with the TEXT0, TEXT1 and TEXT2 commands. These commands draw horizontal text only. The TEXT0 command uses two expandable fixed fonts, each containing a full ASCII character set featuring true descenders and smooth, expanded characters. The TEXTC command controls the size of the font used by the TEXT0 command. Size 0 refers to 5x7 characters contained within 6x9 cells. Size 1 corresponds to a 7x9 font in an 8x12 cell size. Size 2 doubles the size 0 characters. For sizes 3 through 255, use the following algorithm to determine the size, in pixels, of each character:

```
((n-1)x7) x ((n-1)x9)    [character size]
((n-1)x8) x ((n-1)x12)   [cell size]
                          where n = size
```

For example, Size 4 uses 21x27 characters in a 24 x 36 pixel cell. The TEXT1 command also uses a fixed font containing the full ASCII character set with 5x7 format in 8x8 cells.

The TEXT2 command draws in a variable-cell font defined using the TEXTDN command. The TEXTDN command allows the definition of any character format in variable cells of any size. Only the amount of system RAM allocated to text font storage by a CONFIG command limits the space available for a TEXTDN command. Thus, the TEXT2 font may define and combine characters as small as 1x1 pixels, or as large as 512x512 pixels and more. The variable-cell capability of TEXT2 can simulate proportional-spacing techniques, or can implement complex fonts such as Chinese characters. The TEXT2 font may also store "building block" graphic images, e.g. an OR-gate for CAD applications. Up to 255 separate characters may be defined with TEXTDN and drawn with TEXT2. In source mode, these characters may be described as "char" or by their equivalent ASCII value in decimal or hexidecimal format. Thus, in source mode (see INTERACT Interpreter) the following are identical commands:

```
TEXT2 "A"
TEXT2 65
TEXT2 041H
```

General Description

The TEXT2 font defines a character as an array of pixels. The bytes in the **fntlst** parameter of the TEXTDN command define the pixel array starting at the lower left corner of the cell and working to the right and upward. One byte represents each 8 bits, or fraction of 8 bits, required to define one horizontal line of the cell. Additional bytes define each successive line of the cell. Thus, a cell which is 14 x-direction by 20 y-direction pixels in size will require 2 bytes of definition for each horizontal line, and 40 bytes of total definition in the TEXTDN command. The definition stores internally in a compressed format. Use the following equation to determine the number of bytes of memory, (M), needed for a given character:

$$M = INT\ ((x*y)/8) + 6$$

where INT represents the integer function. Figure 3.2 illustrates the definition process through an example.

Text characters for all fonts display into the image memory using the current point as the coordinate of the lower left corner of the character cell. The current point (CREG0) does not change. CREG 7 holds the coordinates for the text endpoint, that is, the coordinates of the lower right-hand corner of the last cell written plus one pixel in the positive x-direction. Place the contents of CREG 7 into CREG 0 to continue a text string. TEXT1 wraps around with a downward shift of one cell upon exceeding the right edge of image memory. Due to ambiguities in character size, TEXT0 and TEXT2 truncate excessive character string lengths at the image memory boundary.

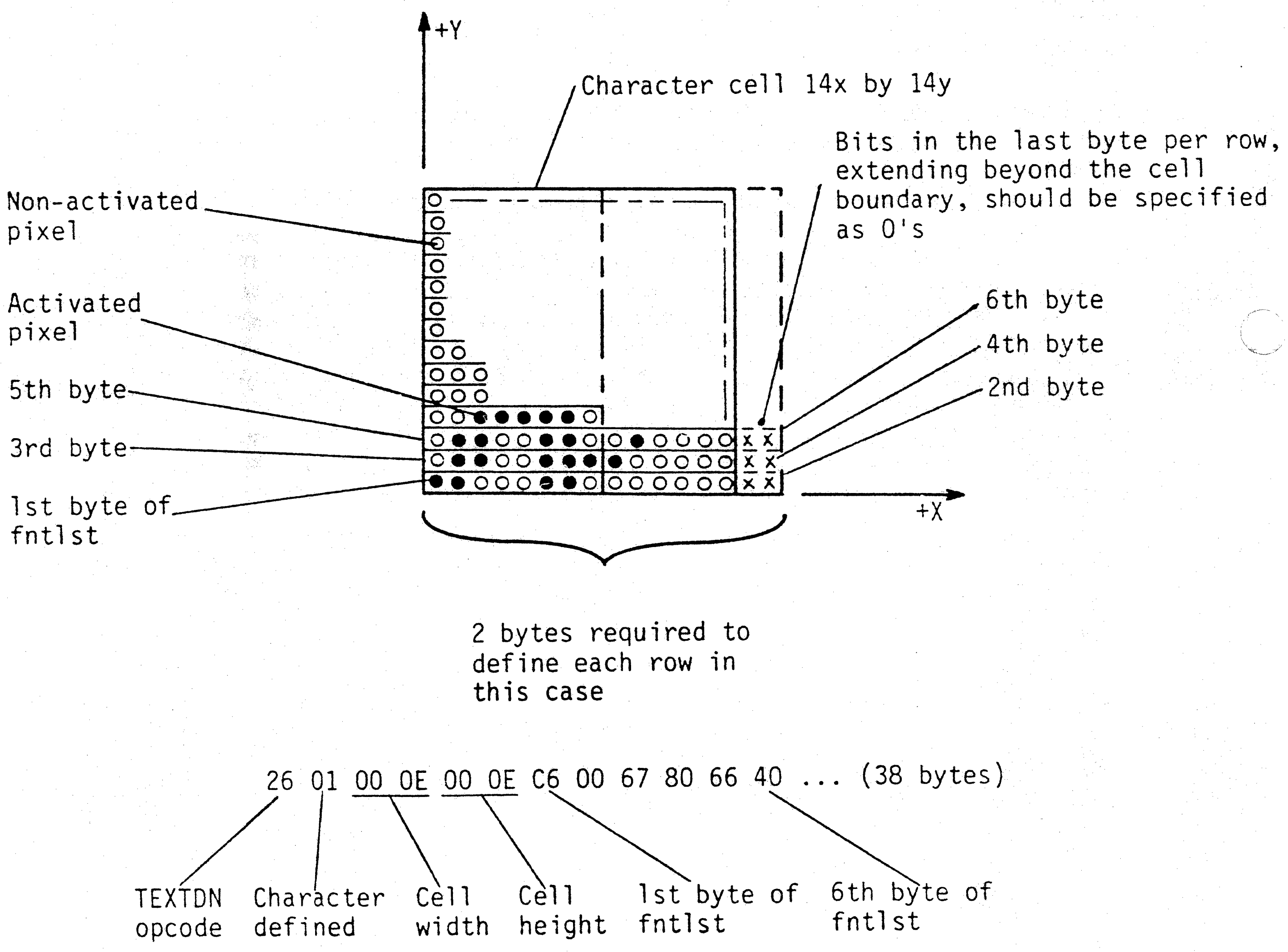

+Y
Character cell 14x by 14y
Bits in the last byte per row, extending beyond the cell boundary, should be specified as 0's
Non-activated pixel
Activated pixel
5th byte
3rd byte
1st byte of fntlst
6th byte
4th byte
2nd byte
+X
2 bytes required to define each row in this case
26 01 00 0E 00 0E C6 00 67 80 66 40 ... (38 bytes)
TEXTDN opcode
Character defined
Cell width
Cell height
1st byte of fntlst
6th byte of fntlst

Figure 3.2 : TEXT2 Definition Example

3.2 - Macro Commands

Macros involve a series of INTERACT commands executed by a single command. INTERACT provides up to 256 simultaneously defined macros. The MACDEF and MACEND commands mark the beginning and end of a macro definition respectively. The MACRUN command executes the specified macro, while MACREP repeats the execution invoked with MACRUN of a particular macroa designated number of times. Macros may be nested up to 16 levels. Allow two levels for macros invoked with MACREP, or BUTTON or BUTCON.

The macro capability is used to define a list of commands for later execution. The VM88xx allows the definition of 256 MACROs with a nesting depth of 15. There are five macro commands available:

MACDEF **macnum**
MACEND
MACRUN **macnum**
MACREP **macnum, count**
MACERA **macnum**

The MACDEF command defines a macro, where **macnum** is between 0 and 255. The commands following the MACDEF command and ending with the MACEND command define a macro. The commands can consist of any combination of valid INTERACT commands (commands and parameters), with the exception of the commands WARM, COLD, and CONFIG. Only the available memory space limits the length of the MACDEF command string (refer to the CONFIG command).

The user can redefine any previously defined macro by defining another MACDEF command with the macro number of the macro which is to be redefined. The MACERA command erases the definition of a specified macro thereby freeing space in the macro buffer.

The MACEND command ends the macro definition at the current nesting level. If no macro definition is in progress, no action occurs.

The MACRUN command executes a previously defined macro. The MACREP command runs a previously defined macro a number of times (as defined by count). If count = 0, then the macro repeats indefinitely.

3.3 - Button Commands

The BUTTON and BUTTBL commands allow the user to access macros through a reconfigurable table. When INTERACT is initialized, each button number (0 - 31) is associated with its respective macro; this association can be changed with the BUTTBL command. The button number specified in the BUTTON command indexes the button table, invoking the macro associated with that button. Thus, the BUTTON and BUTTBL commands provide dynamic access to a set of on-board macros.

The BUTREC and BUTCON commands allow the user to conditionally invoke the BUTTON command. BUTREC associates a rectangular area of virtual memory with a particular button. BUTCON, the conditional button command, has as its parameter a coordinate register. If the value of the specified coordinate register falls within a rectangle specified by the BUTREC command, a BUTTON command is invoked.

Button commands may also be accessed through other devices. The optionallight pen invokes BUTTON 0 when pressed to the screen. The optional digital tablet can run up to 16 different buttons from a hand held cursor.

3.4 - Display Control Commands

Display control commands affect the way subsequent commands draw to the screen. They can also alter an existing display.

3.4.1 - Bit-Plane Control and Masks

The number of bits used to define the colors of a graphics system also specifies the number of bit planes. With masks and look-up table (LUT) commands, these planes can create non-destructive backgrounds and dynamic foregrounds. The contents of the bit plane mask, VREG 6, are logically AND'd with current value before drawing to image memory. The LUT mask (LUTMSK) acts on the LUT index. Thus several indices can use the same bit designation, but the mask can produce different colors. The masks can create background or foreground color without changing the LUTs.

The blank (BLANK) command blanks the displayed image without affecting image memory. Commands sent to the board during a blank command will appear as part of the restored image when the blank flag is turned off.

3.4.2 - Primitive Fills and Drawing Patterns

The primitive fill (PRMFIL) command instructs all subsequent drawing primitives which produce an enclosed space to fill that area with the current color. Otherwise, primitive draw commands draw only an outline.

The area fill commands, AREA1 and AREA2, fill the interiors of "closed" graphic outlines with pixels of the current value. Both types of area fill require the use of a "seed point" coordinate, provided in CREG 0. Use any point in the interior of the target area as a seed point. The AREA1 command finds the boundary color by moving horizontally to the left until encountering a pixel value different from the starting value. AREA1 will fill the inside of the outline by tracing along the entire boundary, drawing to the right from each boundary point while inside the figure. The AREA2 command functions similarly to AREA1 except that the named VREG holds the value of the boundary color. From the seed point, AREA2 moves to the left until finding a pixel of this value. The command identifies this pixel as part of the boundary. This command then fills within the boundary as in an AREA 1 command. Both types of fill employ the fill mask (VREG 3) in their respective boundary comparisons. The fill mask ANDs with both the seed point value and the current pixel value before any boundary comparison occurs. Therefore, the fill mask can disable comparison on certain bit plane positions.

The vector pattern (VECPAT) command specifies the pattern of the line drawn in graphics primitives. All lines use a single pixel width but may specify any dash or dot combination. VECPAT masks the draw made to the screen, repeating the pattern every 16 pixels. The 16-bit number, providing one bit for each pixel, sets an on/off pattern for the drawn vector. A one in the pattern draws a pixel in the current color, while a zero does not affect the screen. The first pixel (FIRSTP) command sets a flag to draw or not draw the first pixel in a vector.

The area pattern (AREAPT) command specifies the pattern of an area filled by a graphics primitive. All filled areas use the specified area pattern, which is composed of 16 words of parameters, defining a square area 16 pixels long and 16 pixels high. Each of the 256 pixels in this area corresponds to a bit in the 16 word pattern. A "1" in the pattern allows the corresponding pixel to be drawn in a filled primitive, while a "0" masks out the corresponding pixel in the area being filled.

3.4.3 - Clipping

INTERACT clips any pixels drawn outside of image memory. The clipping window definition (CLIPDF) command defines a clipping boundary. The clipping (CLIP) command enables that boundary. The clipping window only affects subsequent commands. Existing displays remain unaffected by an enabled window.

3.4.4 - Highlighting

The blink commands control highlighting of image portions. These commands enable blinking by alternating the LUT values for a particular pixel value between two specified values. As many as 256 independent types of blinking fields may occur in the image by using all the pixel values. All types of blinking fields must blink at the same rate, but may alternate between any two of the possible display colors available in the palette.

The BLINKE command enables blinking of a particular pixel value, in one, two, or all three of the LUTs, between two specified entries. The BLINKR command sets the blink rate in vertical retrace interval units. The BLINKC command clears all previous blinking set-ups and returns all fields to entry 1 of the BLINKE command. The BLINKD command disables the blinking of only a specified pixel value.

3.5 - Register Operations

INTERACT provides two types of storage registers: value registers (VREGs) and coordinate registers (CREGs). Refer to Sections 2.7 and 2.8 for more information on reserved registers and their designations. Both types of registers allow similar operations.

Use the register load (VLOAD and CLOAD) to load color values and coordinates into a specified register. Copy the contents of one register to another using the move (CMOVE and VMOVE) commands. Other operations include adding (VADD and CADD) and subtracting (VSUB and CSUB) register contents.

3.6 - Readback Commands

Readback commands provide information stored in various registers to the user. Read the contents of coordinate and color value registers using read (READCR and READVR) commands. Read the value at the current point using the read pixel (READP) command. The RDPIXR command reads the value of the current point and places that value in VREG 0 as the current color.

BUTTBL

BUTTBL index,macnum Load button table.

Assign a macro **macnum** to button number **index** in the button table. The value **index** varies from 0 to 31. The value **macnum** varies between 0 and 255.

Example :

```
MACDEF 51                  ;Begin macro definition
VALUE 0                    ;Set current pixel value to 0
FLOOD                      ;Flood current update buffer with
                           ;current pixel value
VALUE 1                    ;Set current pixel value to 1
CIRCLE 25                  ;Draw a circle of radius 25
MACEND                     ;End macro definition
BUTTBL 8 51                ;Assign macro 51 to button location 8
BUTTON 8
```

Object Code Format :

[AAH][**index**][**macnum**] (3 bytes)

Affected by : NONE

Affects : Button Table

Command available Version ≥ 2.0

BUTREC

BUTREC **butnum,x1,y1,**
x2,y2 — Assign a rectangular area to a button number

Assign a rectangular area to button **butnum.** The rectangular area is defined as having a lower left corner of (**x1,y1**) and an upper right corner of (**x2,y2**). If the two corners are equivalent, the rectangle is reduced to a point. If **x2** is less than **x1** or **y2** is less than **y1,** then no area is assigned to button **butnum.** This prevents button number **butnum** from being invoked by a BUTCON. The same area may be assigned to more than one button. This command is used with the BUTCON command to conditionally execute buttons.

Example:

```
MACDEF 2                   ;Begin definition of macro 2
VALUE 0                    ;Set current pixel value to 0
FLOOD                      ;Flood current update buffer with
                           ;current pixel value
VALUE 1                    ;Set current pixel value to 1
CIRCLE 100                 ;Draw circle of radius 100
MACEND                     ;End definition of macro 2
BUTTBL 3 2                 ;Run macro 2 if button 3 requested
BUTREC 3 0 0 100 100       ;Associate rectangle (0,0),
                           ;(100,100) with button 3
CLOAD 20 50 50             ;Load CREG 20 with (50,50)
CLOAD 21 -10 -20           ;Load CREG 21 with (-10,-20)
BUTCON 20                  ;Draw circle of radius 100
BUTCON 21                  ;Does not execute macro 2
```

Object Code Format:

[B9H][**butnum**][high**x1**][low**x1**][high**y1**][low**y1**][high**x2**][low**x2**]
[high**y2**][low**y2**] (10 bytes)

Affected by : NONE

Affects : Conditional Button Execution Table

Command available Version ≥ 4.0

BUTCON

BUTCON **creg** Run a conditional button.

Run each button whose defined rectangular area (see BUTREC) contains the coordinates stored in coordinate register **creg.**

Example:

```
MACDEF 2                      ;Begin definition of macro 2
VALUE 0                       ;Set current pixel value to 0
FLOOD                         ;Flood current update buffer with
                              ;current pixel value
VALUE 1                       ;Set current pixel value to 1
CIRCLE 100                    ;Draw circle of radius 100
MACEND                        ;End definition of macro 2
BUTTBL 3 2                    ;Run macro 2 if button 3 requested
BUTREC 3 0 0 100 100          ;Associate rectangle (0,0),
                              ;(100,100) with button 3
CLOAD 20 50 50                ;Load CREG 20 with (50,50)
CLOAD 21 -10 -20              ;Load CREG 21 with (-10,-20)
BUTCON 20                     ;Draw circle of radius 100
BUTCON 21                     ;Does not execute macro 2
```

Object Code Format:

[BAH][**creg**] (2 bytes)

Affected by : Conditional Button Execution Table
Button Table

Affects : Button FIFO Event Queue

Command available Version ≥ 4.0

BUFFER **update,display**   Select buffer usage.

Display image buffer **display** to the screen. Subsequent graphics commands operate on the **update** buffer. This command synchronizes with vertical retrace. The number of buffers allowed depends on the image size and amount of available memory. (Refer to the hardware manual.)

Enabled crosshairs appear in the **display** buffer.

Example :

```
BUFFER 0 1                  ;Update buffer 0, and display buffer 1
BUFFER 0 0                  ;Update and display buffer 0
```

Object Code Format :

[E0H][**update**][**display**] (3 bytes)
224

Affected by : NONE

Affects :   Updated Buffer
               Display Buffer

Command available Version ≥ 1.0

BLKMOV

BLKMOV x1,y1,x2,y2 Move block to current point.

Move the rectangular block with one corner at **x1,y1** and the opposite corner at **x2,y2,** to the current point. The pixel **x1,y1** is placed at the current point.

Example :

```
                                 14
MOVABS 0 20     01,00,00,00,~~20~~   ;Move current point to 0,20
PRMFIL 1        1F,01            ;Set primitive fill flag
VALUE 1         06,02            ;Set current pixel value to 1
CIRCLE 50       0E,00,32         ;Draw circle centered at 0,20
MOVABS 55 75    01,00,37,00,4B   ;Move current point to 55 75
BLKMOV 0 20 50 70                ;Move defined block such that data at
    E5  000,00  00  00           ;point 0,20 appears at point 55,75.
            14  32  46           ;The orientation of pixels within the
                                 ;block will not change.
```

Object Code Format :

[E5H][high**x1**][low**x1**][high**y1**][low**y1**][high**x2**][low**x2**][high**y2**][low**y2**] (9 bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Bit Plane Mask
Update Buffer

Affects : NONE

Command available Version ≥ 3.0

BLINKR **frames**	Set blink rate to **frames** vertical synch intervals.

Set the rate at which LUT entries will alternate after enabling a blink command. The command defines this rate as the number of vertical sync intervals between swapping. The value of **frames** ranges from 0 to 255.

Example :

```
BLINKR 60                ;Set blink rate to 1 swap per second
                         ;for a 60 Hz configuration
```

Object Code Format :

[22H][**frames**] (2 bytes)

34

Affected by :   NONE

Affects :   Blink Rate

Command available Version ≥ 1.0

BLINKE

BLINKE **lut,index** Enable blink of specified **lut,index.**
entry1,entry2

Enable blinking of a specified LUT location. The value **lut** specifies the RGB enable mask. **Index** specifies the value code to be blinked for all requested LUTs. The value **index** ranges from 0 to (2(pixel depth)-1). Setting the least significant bit of **lut** (bit 0) enables the blue LUT value for that **index,** setting bit 1 of **lut** enables the green LUT value, while setting bit 2 of **lut** enables the red LUT value. More than one bit in the RGB enable mask may be set in a single BLINKE command. For example, setting**lut**=7 enables all look-up table values for the specified **lut. Entry1** and **entry2** will alternate at a rate set by the BLINKR command. The values **entry1** and **entry2** range from 0 to 2(bits/color in each LUT). (See Section 2.12.) This command synchronizes to vertical retrace.

Example :

```
VALUE 3                    ;Set current pixel value to 3
FLOOD                      ;Flood current update buffer with
                           ;current pixel value
BLINKE 4 3 7 15            ;Enable blink of pixel value 3 in the
                           ;red LUT only.  Pixels of this value
                           ;alternate between red content of 7 and
                           ;15
```

Object Code Format :

[20H][**lut**][**index**][**entry1**][**entry2**] (5 bytes)

Affected by : NONE

Affects : Blink Status
Blink Tables

Command available Version ≥ 1.0

BLINKD **lut,index** Disable blink of specified **lut,index.**

Disable blinking of a specified LUT location. The value **lut** specifies the RGB enable mask. **Index** specifies the value code to be disabled for all requested LUTs. The value **index** ranges from 0 to (2(pixel depth)-1). Setting the least significant bit of **lut** (bit 0) disables the blue LUT value for that **index**, setting bit 1 of **lut** disables the green LUT value, while setting bit 2 of **lut** disables the red LUT value. More than one bit in the RGB enable mask may be set in a single BLINKD command. For example, setting**lut**=7 disables all look-up table values for the specified **lut.** At a blink disable command, the disabled entries in the LUTs revert to the original values they contained before receiving the most recent BLINKE command.

Example :

```
VALUE 5                    ;Set current pixel value to 5
PRMFIL 1                   ;Enable filled figures
CIRCLE 30                  ;Draw filled circle of radius 30
BLINKE 7 5 7 15            ;Enable blink of color 5 from dark
                           ;gray to white
BLINKD 7 5                 ;Disable blinking of color 5,
                           ;returning to cyan
```

Object Code Format :

[21H][**lut**][**index**] (3 bytes)

Affected by : NONE

Affects : Blink Status
Lookup Tables

Command available Version ≥ 1.0

BLINKC

BLINKC Clear blink table.

Disable blinking of all LUT locations. All blinking LUT entries reset to **entryl** of their blink values. This command synchronizes to vertical retrace.

Example :

```
VALUE 5                   ;Set current pixel value to 5
PRMFIL 1                  ;Enable filled figures
CIRCLE 30                 ;Draw filled circle of radius 30
BLINKE 7 5 7 15           ;Enable blink of color 5 from dark
                          ;gray to white
BLINKC                    ;Clear blink table, returning color 5
                          ;to dark gray
```

Object Code Format :

[23H] (1 byte)

Affected by : Blink Status

Affects : Lookup Tables
Blink State

Command available Version ≥ 1.0

BLANK **flag** Blank the screen when **flag**=1; if **flag**=0, unblank the screen.

Set the blank flag to the value **flag.** If **flag**=1, the command blanks the screen, no longer displaying image data. If **flag**=0, the screen displays image data.

Example :

```
VALUE 1                  ;Set current pixel value to 1
CIRCLE 50                ;Draw circle of radius 50
BLANK 1                  ;Blank screen
CIRCLE 100               ;Draw circle of radius 100
BLANK 0                  ;Unblank screen
```

Object Code Format :

[31H][**flag**] (2 bytes)

Affected by : NONE

Affects : Blank Flag

Command available Version ≥ 1.0

For more information on these drivers, refer to Section 5, System Interfacing. User-written drivers require a separate opcode.

Example :

```
ASSIGN 1 2                  ;Load the interpreter onto channel 1
ASSIGN 5 0FH                ;Load the light pen onto channel 5
```

Object Code Format :

[B8][**chan**][**dev**] (3 bytes)

Affected by : NONE

Affects : NONE

Command available Version ≥ 4.0

ASSIGN **chan,dev** Assign a device to a channel.

Load the device driver **dev** onto the channel **chan.** Values for both **chan** and **dev** correspond to a specific channel or device. Use any of the following as valid channels:

Value	Channel
0	MULTIBUS
1	first iSBX port
2	second iSBX port
5	input only port (light pen, touch screen)

Use any of the following as valid devices:

Value	Device
0	dummy (no action)
1	binary (INTERACT object code)
2	interpreter (INTERACT memnonics)
3	printer
5	bitpad
15	light pen (channel 5 only)

Affected by :	Current Point Coordinate Origin Clipping Boundary Area Fill Mask Current Color Bit Plane Mask Update Buffer Area Pattern
Affects :	NONE

Command available Version $\geq$ 1.0

18

AREA2 **vreg** — Area fill. Boundary pixel value given in **vreg.**

Set all pixels within a closed region to the current value (VREG 0). A boundary consists of any pixel whose value matches the value of VREG **vreg.** The value in VREG **vreg** must differ from the current color. The current point must lie within the target area. This area extends from the current point outward to an encountered boundary. The boundary color must differ from the current value. INTERACT Version 2.0 limits area fills to continuous regions. The region may not contain any pixels whose value also matches the value in VREG **vreg,** i.e. the command requires a single, contiguous boundary. This restriction does not hold true for Version 3.0. The boundary pixel values and the value specified by value register **vreg** are ANDed with the fill mask (VREG 3) and the bit plane mask (VREG 6) before the comparison is made. The AREA2 command differs from AREA1 in that AREA2 seeks a boundary of a specific pixel value placed in **vreg** before execution of the area fill.

Example :

```
VALUE 15                ;Set current pixel value to 15
MOVABS 0 0              ;Move current point to 0,0
CIRCLE 20               ;Draw circle of radius 20
VALUE 14                ;Set current pixel value to 14
CIRCLE 25               ;Draw circle of radius 25
VLOAD 9 14              ;Load VREG 9 with value 14
FILMSK 15               ;Set all mask bits to 1
VALUE 2                 ;Fill color
AREA2 9                 ;Begin area fill. Boundary pixel value
                        ;is found in VREG 9. (Inner circle is
                        ;over-written because it is not drawn
                        ;in boundary pixel value.)
```

Object Code Format :

[14H][**vreg**] (2 bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Bit Plane Mask
Current Color
Area Fill Mask
Area Pattern
Update Buffer

Affects : NONE

Command available Version $\geq$ 1.0

AREA1 Area fill. Any pixel different from start and current value defines a boundary.

Setall pixels within a closed region to the current value (VREG 0). A boundary consists of any pixel whose value differs from the value of the current point and the value of the current color. The current point must lie within the target area. This area extends from the current point to an encountered boundary. INTERACT Version 2.0 area fills work only for continuous regions. The region may not contain any "holes," i.e. the command requires a single, contiguous boundary (e.g., AREA1 will not fill the area between concentric circles). This limitation does not apply to Version 3.0. The boundary colors must differ from the current value. The boundary pixel values and the original pixel value are ANDed with the fill mask (VREG 3) and the bit plane mask (VREG 6) before the comparison is made.

Example :

```
VALUE 5                  ;Set current pixel value to 5
MOVABS 16 16             ;Move current point to 16,16
CIRCLE 30                ;Draw circle of radius 30
VALUE 6                  ;Set current pixel value to 6
FILMSK 15                ;Set all mask bits to 1
AREA1                    ;Fill previous circle with value 6
```

Object Code Format :

[13H] (1 byte)

AREAPT

AREAPT **pattern** Define area pattern mask.

The 16 **pattern** mask words define a 16x16 pixel array to be repeated horizontally and vertically when drawing filled figures. The least significant bit of the first word appears in the lower left-hand corner when displayed. Setting all bits in the mask (sending 16 words of 65535) will cause areas to be filled in solid, and is the default at power up or following a COLD.

Example :

```
VALUE 1  FFFF  FFFF          ;Set current pixel value to 1
AREAPT 65535,65535,0,0       ;Define area pattern as 2 pixel wide
       65535,65535,0,0       ;horizontal stripes
       65535,65535,0,0
       65535,65535,0,0
PRMFIL 1                     ;Engage primitive fill flag
CIRCLE 50                    ;Draw filled circle with a striped
                             ;pattern
```

Object Code Format :

[2D][high**p0**][low**p0**]...[high**p15**][low**p15**] (33 bytes)

Affected by : NONE

Affects : Area Pattern

Command available Version ≥ 4.0

32768
32768
65536

ARC rad,a1,a2 — Draw arc of radius **rad**, starting angle **a1**, and ending angle **a2**.

Draw a circular arc with its center at the current point (CREG 0) and with a radius of **rad**. The parameters **a1** and **a2** specify the starting angle and ending angle respectively. These parameters define the angle in integer degrees measured counter-clockwise. An angle of 0 specifies horizontal to the right from the current point. The arc draws counter-clockwise from the start angle to the end angle. The values **a1** and **a2** range from -32,768 to +32,767. The parameter **rad** may not exceed 8191 pixels.

Example :

```
VAL 1           06 01               ;Set current color to 1
MOVABS 0 0      01,00 00 00 00      ;Move current point to location 0,0
ARC 75 45 135                       ;Draw circular arc of radius 75,
     00 00 00
11   4B 2D 87                       ;starting at 45 degrees and ending at
                                    ;135 degrees
ARC 100 -30 60                      ;Draw circular arc of radius 100,
     00  FF 00                      ;starting at -30 degrees and ending at
11   64  D0 3C                      ;60 degrees
```

Object Code Format :

[11H][highrad][lowrad][higha1][lowa1][higha2][lowa2] (7 bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Current Color
Bit Plane Mask
Update Buffer
Vector Pattern

Affects : NONE

Command available Version ≥ 1.0

```
Affected by:
                    Area Pattern
                    Bit Plane Mask
                    Current Color
                    Current Point
                    Coordinate Origin
                    Clipping Boundary
                    First Pixel Flag
                    Pixel Function
                    Update Buffer
                    Vector Pattern
               -------------------------
                         \
                          \
                           \ -------------------------------
                            | Affected by Elements of state |
                             -------------------------------

Affects:
                    None
               ----------------
                      \
                       \
                        \ ---------------------------
                         | Affects Elements of state |
                          ---------------------------
```

Command available Version ≥ 2.0

Note: In the Example section, all commands are issued immediately after power-on reset.

Figure 4.1 - Command Format (Cont.)

```
                                                             CIRCI
                                                             -----
                                                   ________________/
                                                  |Command mnemonic|
                                                   ----------------


CIRCI creg                        Draw circle given a point on
----- ----                        circumference.
\       \                         ----------------------------
 \       \____________               \
  \       |Parameter(s)|              \
   \       ------------                \
    \_______                            \_____________________________
    |Command|                           |Concise command description|
     -------                             ----------------------------


Draw a circle (filled for PRMFIL enabled) with the center located
at  the  current point such that the circumference  contains  the
point specified in CREG creg.
-----------------------------------------------------------------
    \____________________________________
    |Detailed command description and use|
     ------------------------------------

Example :

MOVABS 0 0                        ;Current point becomes 0,0
CLOAD 37 25 60                    ;Load CREG 37 with 25,60
CIRCI 37                          ;Draw circle of radius 65
--------------                    -------------------------
    \________                              \_____________________________
    |Commands|                             |Comments describing commands|
     --------                               ----------------------------

Object Code Format :

[10H][creg] (2 bytes)
----------- ---------
 \                   \_____________________________
  \                  |Object code size requirements|
   \                  -----------------------------
    \__________________
    |Object code syntax|
     ------------------
```

Figure 4.1 - Command Format

Graphics Commands

Refer to Figure 4.1 for the command format.

Use this section as a programmer's reference guide. A summary of the INTERACT commands appears in Appendix C.

4
Graphics Commands

4.1 - Syntax

The hosting hardware processes INTERACT graphics commands in one of two formats: "source" format, using an on board interpreter, or "object" format for high-speed machine-to-machine communication. The following paragraph describes the syntax rules for each format.

Invoke the INTERACT "object" format for inter-processor communication of commands. All VMI card-level graphics processors use it as standard command format. The command descriptions in Section 4.2 provide the syntax of the "object" format for each command. The "object" format consists of a stream of 8-bit bytes written to the graphics processor by the system CPU. The processor supplies all bytes in binary format. The first byte sent for any command corresponds to the opcode for that command. Based on the specific command, a variable number of parameter bytes follows the opcode. Send the opcode for the next command immediately following the last parameter byte of a given command. The board will accept commands whenever the Programmed I/O status byte indicates XMIT ready. If a transmission error causes the INTERACT input processor to get "out of sync," a reset command to the Programmed I/O port reinitializes communications. Section 5.1 provides details on this procedure and all other aspects of the Programmed I/O.

4.2 - Descriptions

This section presents descriptive information on the commands for all versions of INTERACT. Each command starts on a new page. The information provided for each command includes the following:

- Command Mnemonic
- Source Format Syntax
- One-line Description
- Descriptive Paragraph
- Examples of Usage
- Object Format Syntax
- Object Format Byte Length
- Affected by Elements of State
- Affects Elements of State
- Version Reference

3.7 - Image Transmission

The PIXELS command defines an image pixel-by-pixel. The parameters specify the number of pixel rows and columns to be defined. Supply the pixel values starting at the lower left cornerofthe array and working to the right and upward. In a similar way, use the READP command to read an image portion in a pixel-by-pixel fashion. The PIXELS and READP commands facilitate the storage and retrieval of entire graphic images.

3.8 - Run-Length Encoding

Run-length encoding compresses image data by giving a repeat factor where data of the same value occurs in consecutive horizontal locations. This value repetition very commonly takes place in graphing applications. For more complex patterns, the scheme used by INTERACT avoids inefficiency by providing a code to turn off the run-length encoding. Permitting the user to specify how many bits from each pixel to transmit achieves further compression. This process proves useful when employing fewer than eight bit-planes. Another application involves using some planes to hold overlay information, and transmitting only the background. Note that the background planes occur as the less significant bits. On the other hand, to allocate extra bits, set the depth parameter to a value larger than the number of physical bit planes used (up to 32). The upper bits get filled with zeros. A repeat count of zero is neccessary and sufficient to end the command.

The PIXDMP command produces data in the form of a PIXLOD command. That is, F1H (the PIXLOD op code) appears as the first byte in the data stream followed by depth, dx, and dy as specified in the PIXDMP command. The remainder of the data occurs asrun-length encoded pixel data in a bit stream form. While successive bytes appear logically adjacent to each other, their boundaries may not correspond to any logical boundary in the data. The bit stream consists of multiple blocks where each block begins with an 8-bit **count.** If **count** equals zero, no more data will follow, i.e., a zero **count** signifies the last block. For **count** positive, the following **depth** bits define a pixel value which occurs **count** times in the source image. For **count** negative, the following (**depth** * significant bits) specify count pixels. Within each byte, the most significant bit (MSB) occurs first. Blocks of this form cover the specified image window beginning from the lower left-hand corner of the rectangle space and moving left to right and bottom to top. The remaining lower bits in the last block get set to zero, and a 0-length block follows as the last block.

BUTTON index Execute macro defined for cursor button.

Execute the macro assigned to button number **index.** The value **index** varies from 0 to 31.

Example :

```
MACDEF 17                ;Begin macro definition
VALUE 2                  ;Set current pixel value to 2
FLOOD                    ;Flood current update buffer with
                         ;current pixel value
VALUE 3                  ;Set current pixel value to 3
CIRCLE 25                ;Draw a circle of radius 25
MACEND                   ;End macro definition
BUTTBL 5 17              ;Assign macro 17 to button location 5
BUTTON 5                 ;Simulate pressing button 5 on cursor
```

Object Code Format:

[ABH][**index**] (2 bytes)

Affected by : Button Table

Affects : Button FIFO Event Queue

Command available Version ≥ 2.0

CADD

CADD **csum,creg** Add the contents of one CREG to another.

Add the x- and y-coordinates in the CREG specified by **creg** to the x- and y-coordinates in CREG **csum,** leaving the result in CREG **csum.**

Example :

```
CLOAD 22 50 25          ;Load CREG 22 with 50,25
CLOAD 24 15 30          ;Load CREG 24 with 15,30
CADD 22 24              ;Adds x-,y-values of CREGs 22 and 24
                        ;Places result (65,55) in CREG 22
```

Object Code Format :

[A2H]**[csum]**[**creg**] (3 bytes)

Affected by : NONE

Affects : CREG **csum**

Command available Version ≥ 2.0

CIRCI **creg** Draw circle given a point on circumference.

Draw a circle in the current color with the center located at the current point such that the circumference includes the point specified in CREG **creg.** The radius may not exceed 8191 pixels.

Example :

```
MOVABS 0 0              ;Current point becomes 0,0
VALUE 1                 ;Set current pixel value to 1
CLOAD 37 25 60          ;Load CREG 37 with 25,60
CIRCI 37                ;Draw circle containing point 25,60
                        ;on its circumference
```

Object Code Format :

[10H][**creg**] (2 bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Primitive Fill Flag
Current Value
Bit Plane Mask
Area Pattern
Update Buffer
Vector Pattern

Affects : NONE

Command available Version ≥ 2.0

CIRCLE

CIRCLE **rad** Draw a circle of radius **rad.**

Draw a circle of radius **rad** in the current color. The center of the circle lies at the current point (CREG 0). The radius **rad** can range from -8191 to +8191. A circle of radius zero sets the current point to the current pixel value.

Example :

```
MOVABS 100 150          ;Move current point to 100,150
VALUE 1                 ;Set current pixel value to 1
CIRCLE 30               ;Draw circle of radius 30 centered at
                        ;100,150
MOVREL 10 0             ;Move current point by 10,0 to 110,150
CIRCLE 20               ;Draw circle of radius 20 centered at
                        ;110,150
CIRCLE 10               ;Draw circle of radius 10 centered at
                        ;110,150
```

Object Code Format :

[0EH][high**rad**][low**rad**] (3 bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Primitive Fill Flag
Current Value
Bit Plane Mask
Area Pattern
Update Buffer
Vector Pattern

Affects : NONE

Command available Version ≥ 1.0

CIRCXY

CIRCXY x,y — Draw a circle given a point on the circumference.

Draw a circle in the current color with the center located at the current point such that the circumference includes the point (**x,y**). The radius may not exceed 8191 pixels.

Example :

```
MOVABS 20 32                ;Move current point to 20,32
VALUE 1                     ;Set current pixel value to 1
CIRCXY 40 80                ;Draw a circle with the center at 20,32
                            ;and point 40,80 on its circumference
```

Object Code Format :

[0FH][high**x**][low**x**][high**y**][low**y**] (5 bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Primitive Fill Flag
Current Value
Bit Plane Mask
Area Pattern
Update Buffer
Vector Pattern

20
32

Affects : NONE

Command available Version ≥ 2.0

30 H

CLIP

CLIP **num** Select current clipping window.

Enable the current clipping window to the clipping window format **num.** The value **num** may range from 0 to 4. Set the clipping window format with the CLIPDF command. If **num**=0 the current clipping window is set to the power-on reset default clipping window format. The x,y coordinates specified by the format **num** are loaded into coordinate registers CREG9 and CREG10.

Example :

```
CLIPDF 1 -10 -10 30 20    ;Define clipping window
CLIP 1                    ;Invoke clipping window 1
MOVABS -8 0               ;Move current point to -8,0
VALUE 2                   ;Set current pixel value to 2
TEXT1  "Write in          ;Write in window
       window only"
CLIP 0                    ;Invoke default window
```

Object Code Format :

[EAH][**num**] (2 bytes)

Affected by : Clip Window Definitions

Affects: Clipping Boundary

Command available Version ≥ 3.0

CLIPDF **num,x1,y1,x2,y2** Define clipping window.

Set the clipping window format **num** to the rectangular region defined by the corners **x1,y1** and **x2,y2**. Four clipping window formats can be defined; **num** ranges from 1 to 4. The coordinates of the clipping windows are specified in virtual coordinates. The coordinate values range from -32,768 to +32,767. Coordinate registers CREG 9 and CREG 10 are loaded with the coordinates **x1,y1** and **x2,y2** respectively.

Example :

```
CLIPDF 1 -10 -10 30 20     ;Define clipping window
CLIP 1                     ;Invoke clipping window 1
MOVABS -8 0                ;Move current point to -8,0
VALUE 2                    ;Set current pixel value to 2
TEXT1  "Write in           ;Write in window
       window only"
CLIP 0                     ;Invoke default window
```

Object Code Format :

[EBH][**num**][high**x1**][low**x1**][high**y1**][low**y1**][high**x2**][low**x2**]
[high**y2**][low**y2**] (10 bytes)

Affected by : NONE

Affects : Clip Window Definitions

Command available Version ≥ 3.0

CLOAD

CLOAD creg,x,y — Load coordinate register **creg** with **x,y.**

Load the coordinate register **creg** with the value **x,y.** The value **creg** ranges from 0 to 63. The range of **x** and **y** extends from -32,768 to +32,767.

Example :

```
           x,x,  y,y
A0 CLOAD 17 100 150        ;Load CREG 17 with 100,150
   CLOAD 17 50 -50         ;Load CREG 17 with 50,-50
```

Object Code Format :

```
        160
11,12  [A0H][creg][highx][lowx][highy][lowy] (6 bytes)
```

Affected by : NONE

Affects : Coordinate Register **creg**

Command available Version ≥ 1.0

CMOVE **cdst,csrc** Move contents of **csrc** into **cdst.**

Load the coordinate register **cdst** with the data contained in the coordinate register **csrc.** The values **cdst** and **csrc** range from 0 to 63.

Example :

```
CLOAD 25 100 150          ;Load CREG 25 with 100,150
CLOAD 26 20 -50           ;Load CREG 26 with 20,-50
CMOVE 26 25               ;Move contents of CREG 25 into CREG 26
```

Object Code :

[A1H][**cdst**][**csrc**] (3 bytes)

Affected by : NONE

Affects : Coordinate Register **cdst**

Command available Version ≥ 1.0

COLD

```
COLD                        Perform cold start.
```

Reset INTERACT. COLD erases all pending commands.

Example :

```
COLD                        ;Execute a cold start
```

Object Code :

[FDH] (1 byte)

Affected by : None

Affects : All Elements of Board State

Command available Version ≥ 1.0

CONFIG **fifo,macbuf, txtfnt** Configure processor local memory.

Configure local RAM space. Reserve **fifo** bytes for the internal FIFO, **macbuf** bytes for the macro definition area, and **txtfnt** bytes for the TEXT2 font area. Specify the number of bytes to be configured. If the CONFIG command exceeds available local RAM, the various lengths will remain at their previous values. Reconfiguring local RAM erases all pending INTERACT command bytes (not neccessarily whole commands), all macro definitions, and all text definitions. Increasing the size of the internal FIFO allows the graphics processor to buffer more INTERACT commands.

Example :

```
CONFIG 2048 4096 1024     ;Configure RAM for 2K bytes of FIFO,
                          ;4K of macro space, and 1K of space
                          ;for the TEXT2 font definition
```

Object Code Format :

[24H][high**fifo**][low**fifo**][high**macbuf**][low**macbuf**]
[high**txtfnt**][low**txtfnt**] (7 bytes)

Affected by : NONE

How big is the fifo, etc.

Affects : RAM Configuration

Command available Version ≥ 1.0

CSUB

CSUB **cdif,creg** Subtract the contents of one CREG from another.

Subtract the x- and y-coordinates in the CREG specified by **creg** from the x- and y-coordinates in CREG **cdif,** leaving the result in CREG **cdif.**

Example :

```
CLOAD 22 50 25          ;Load CREG 22 with 50,25
CLOAD 24 15 30          ;Load CREG 24 with 15,30
CSUB 22 24              ;Subtract x- and y-values of CREG 24
                        ;from x-,y-values in CREG 22.  Place
                        ;result, (35,-5), in CREG 22.
```

Object Code Format :

[A3H][**cdif**][**creg**] (3 bytes)

Affected by : NONE

Affects : Coordinate Register **cdif**

Command available Version ≥ 2.0

DRWABS **x,y** Draw a vector to the point **x,y.**

Draw a vector from the current point (CREG 0) to the point **x,y.** The command updates the current point to the value **x,y.** For the FIRSTP flag set, the beginning point of the vector will not store to image memory. The values **x** and **y** range from -32,768 to +32,767. The command draws in the current pixel value (VREG 0).

Example :

```
VALUE 1                  ;Set current pixel value to 1
MOVABS 50 50             ;Move current point to 50,50
DRWABS 60 50             ;Draw line to 60,50 (horizontal line 11
                         ;pixels long)
MOVABS 60 60             ;Move current point to 60,60
DRWABS 60 70             ;Draw line to 60,70 (vertical line 11
                         ;pixels long)
DRWABS 70 70             ;Draw diagonal line to 70,70, connected
                         ;to previous line at point 60,60
DRWABS 80 100            ;Draw line to 80,100
```

Object Code Format :

[81H][high**x**][low**x**][high**y**][low**y**] (5 bytes)

Affected by : Bit Plane Mask
Clipping Boundary
Coordinate Origin
Current Point
Current Value
First Pixel Flag
Pixel Function
Update Buffer
Vector Pattern

Affects : Current Point

Command available Version ≥ 1.0

DRWI

DRWI **creg**	Draw a vector to the location specified in **creg**.

Draw a vector from the current point (CREG 0) to the point stored in coordinate register **creg.** The current point (CREG 0) updates to the new point. The value of **creg** ranges from 0 to 63.

Example :

```
VALUE 2                  ;Set current pixel value to 2
CLOAD 40 -120 10         ;Load CREG 40 with coordinates -120,10
MOVABS -100 -50          ;Move current point to -100,-50
DRWI 40                  ;Draw vector from -100,-50 to location
                         ;given in CREG 40
MOVABS -30 -60           ;Move current point to -30,-60
CLOAD 33 100 150         ;Load CREG 33 with 100,150
DRWI 33                  ;Draw vector from -30,-60 to 100,150
```

Object Code Format :

[85H][**creg**] (2 bytes)

Affected by :	Bit Plane Mask Clipping Boundary Coordinate Origin Current Point Current Value First Pixel Flag Pixel Function Update Buffer Vector Pattern
Affects :	Current Point

Command available Version ≥ 1.0

DRWREL

DRWREL **dx,dy** — Draw a vector relative by **dx,dy.**

Draw a vector beginning at the current point (CREG 0) and ending at a point displaced relative to the current point **dx** pixels in the x-direction and **dy** pixels in the y-direction. The values **dx** and **dy** range from -32,768 to +32,767. The current point updates to the sum of the x-component of the previous current point plus **dx** and the sum of the y-component of the previous current point plus **dy.** Setting the value **dx,dy** equal to 0,0 writes only the current point.

Example :

```
VALUE 1                   ;Set current pixel value to 1
MOVABS 50 30              ;Move current point to 50,30
DRWREL 10 20              ;Draw line from 50,30 to 60,50
DRWREL 10 0               ;Draw line from 60,50 to 70,50
DRWREL 0 -10              ;Draw line from 70,50 to 70,40
```

Object Code Format :

[82H][high**dx**][low**dx**][high**dy**][low**dy**] (5 bytes)

Affected by :
- Bit Plane Mask
- Clipping Boundary
- Coordinate Origin
- Current Point
- Current Value
- First Pixel Flag
- Pixel Function
- Update Buffer
- Vector Pattern

Affects : Current Point

Command available Version ≥ 1.0

DRW2R

DRW2R **dxdy** Draw short vector relative.

Draw a vector from the current point to a point offset in the x direction by **dx** and in the y direction by **dy.** The most significant nibble of **dxdy** specifies **dx;** the least significant four bits specify **dy.** The current point updates to the endpoint of the drawn vector. DRW2R requires only two bytes, but the command restricts the range of **dx** and **dy** from -8 to +7.

Example :

```
VALUE 3                  ;Set current pixel value to 3
MOVABS -25 -25           ;Move current point to -25,-25
DRW2R 5 5                ;Draw relative to -20,-20
```

Object Code Format :

[84H][**dxdy**] (2 bytes)

Affected by : Bit Plane Mask
Clipping Boundary
Coordinate Origin
Current Point
Current Value
First Pixel Flag
Pixel Function
Vector Pattern

Affects : Current Point

Command available Version ≥ 2.0

DRW3R **dx,dy** Draw short vector relative.

Draw a vector from the current point to a point offset in the x direction by **dx** and in the y direction by **dy.** The current point then updates to the endpoint of the drawn vector. DRW3R requires only three bytes, but the command restricts the range of **dx** and **dy** from -128 to +127.

Example :

```
VALUE 3                  ;Set current pixel value to 3
MOVABS -25 -25           ;Move current point to -25,-25
DRW3R 50 50              ;Draw the relative distance to
                         ;point 25,25
```

Object Code Format :

[83H][**dx**][**dy**] (3 bytes)

Affected by : Bit Plane Mask
Clipping Boundary
Coordinate Origin
Current Point
Current Value
First Pixel Flag
Pixel Function
Vector Pattern

Affects : Current Point

Command available Version ≥ 2.0

DSPSIZ

DSPSIZ **x,y,freq,screen**   Select screen display format.

Change the screen display to the format specified. Refer to the Graphics Processor Manual for valid parameter values for individual boards. If **screen** = 0 no screen will be drawn. If **screen** = 1, the power-on-reset screen will be drawn.

Example :

```
DSPSIZ 512 512 60 1      ;Select a 512 x 512 display screen
                         ;at 60Hz and draw the power-on-reset
                         ;test screen
```

Object Code Format :

68 02 00 02 00 60 1

[44H][high**x**][low**x**][high**y**][low**y**][**freq**][**screen**] (7 bytes)

3E

Affected by : NONE

Affects :   Display Size

Command available Version ≥ 3.0

FILMSK **mask** Set fill mask for area fills.

Set the fill mask (VREG 3) to **mask**. During fill commands, the bitwise mask "ANDs" with pixel values before boundary comparisons. The value **mask** ranges from 0 to (2(pixel depth)-1).

Example :

```
FILMSK 7                  ;Set fill mask to value 7. Boundary
                          ;comparisons will thus be made only on
                          ;bits 0 to 2 of each pixel value.
```

Object Code Format :

[9FH][**mask**] (2 bytes)

Affected by : NONE

Affects : Area Fill Mask

Command available Version ≥ 1.0

FIRSTP

FIRSTP flag First pixel on vectors is inhibited when **flag**=1.

Inhibit writing the first pixel of vectors if **flag**=1. The inhibited mode of operation eliminates writing shared endpoints of concatenated lines twice into image memory.

Example :

```
VALUE 2                 ;Set current pixel value to 2
POINT                   ;Set current point to
                        ;current pixel value
VALUE 1                 ;Set current pixel value to 1
FIRSTP 1                ;Disable writing first pixel on vectors
DRWABS 10 20            ;Draw vector from current point to
                        ;point 10,20. The pixel at the current
                        ;point will not be included in the draw.
```

Object Code Format :

[2FH][**flag**] (2 bytes)

Affected by : NONE

Affects : First Pixel Flag

Command available Version $\geq$ 1.0

FLOOD — Flood current update buffer with current pixel value.

Change all pixels in the current update buffer to the current pixel value (VREG 0).

Example :

```
VALUE 8                  ;Change current pixel value to 8
FLOOD                    ;Flood the current update buffer to
                         ;value 8
VALUE 3                  ;Change current pixel value 3
FLOOD                    ;Flood the current update buffer to
                         ;value 3
VALUE 7                  ;Change current pixel value to 7
FLOOD                    ;Flood the current update buffer to
                         ;value 7
```

Object Code Format :

[07H] (1 byte)

```
Affected by :  Bit Plane Mask
               Current Value
               Update Buffer
```

Affects :   NONE

Command available Version ≥ 1.0

IMGSIZ

IMGSIZ **x,y,depth** Configure image memory.

Configure image memory into one of various image sizes. The number of buffers possible for a given image size will depend on available memory. Refer to Appendix D in the appropriate Graphics Processor Manual for valid parameter values.

Example :

```
IMGSIZ 512 512 4             ;Set the image to 512x512 resolution
                             ;with four bits per pixel
```

Object Code Format :

[45H][high**x**][low**x**][high**y**][low**y**][**depth**] (6 bytes)

Affected by : NONE

Affects :

Command available Version ≥ 3.0

LUTB **index,entry** Make entry in blue look-up table.

Change an entry in the blue look-up table (LUT). At the offset **index** in the blue LUT, load the blue LUT with **entry.** The value **index** ranges from 0 to (2(pixel depth)-1). Beginning with the next vertical retrace, the color value **index** will be displayed using the new **entry** as the blue intensity. For the range of the value **index** refer to Appendix D in the Graphics Processor Manual.

Example :

```
VALUE 8                    ;Set current pixel value to 8
FLOOD                      ;Flood the current update buffer to
                           ;current pixel value
LUTB 8 7                   ;Change entry in blue LUT location
                           ;8 to 7 (half intensity)
LUTB 8 15                  ;Change entry in blue LUT location
                           ;8 to 15 (full intensity)
VALUE 0                    ;Change current pixel value to 0
FLOOD                      ;Flood the current update buffer to
                           ;current pixel value
LUTB 0 14                  ;Change entry in blue LUT location
                           ;0 to 14
```

Object Code Format :

[1AH][**index**][**entry**] (3 bytes)

Affected by : Blink Status

Affects : Lookup Tables

Command available Version ≥ 1.0

LUTG

LUTG **index,entry**   Make entry in green look-up table.

Change an entry in the green look-up table (LUT). At the offset **index** in the green LUT, load the green LUT with **entry.** The value **index** ranges from 0 to (2(pixel depth)-1). Beginning with the next vertical retrace, the color value **index** will be displayed using the new **entry** as the green intensity. For the range of the value **index** refer to Appendix D in the Graphics Processor Manual. Use this command to influence monochrome LUT values.

Example :

```
VALUE 8                  ;Set current pixel value to 8
FLOOD                    ;Flood the current update buffer to
                         ;current pixel value
LUTG 8 0                 ;Change entry in green LUT location
                         ;8 to 0 (zero intensity)
LUTG 8 15                ;Change entry in green LUT location
                         ;8 to 15 (full intensity)
VALUE 0                  ;Change current pixel value to 0
FLOOD                    ;Flood the current update buffer to
                         ;current pixel value
LUTG 0 14                ;Change entry in green LUT location
                         ;0 to 14
```

Object Code Format :

[19H][**index**][**entry**] (3 bytes)

Affected by : Blink Status

Affects :   Lookup Tables

Command available Version ≥ 1.0

LUTMSK **mask** Mask the LUT values.

Mask the values sent to the look-up tables. A zero bit-value disables that bit within the pixel to zero. A one-value in the mask leaves the color bit unchanged. For example, if a pixel has the value of 0111 binary and the mask was 1011 then the pixel appears as a 0011 binary on the screen.

Example :

```
LUTMSK 7                 ;Set the LUT mask to 0111 binary
```

Object Code Format :

[F7H][**mask**] (2 bytes)

Affected by : NONE

Affects : Lut Mask

Command available Version ≥ 4.0

LUTR

LUTR **index,entry** Make entry in red look-up table.

Change an entry in the red look-up table (LUT). At the offset **index** in the red LUT, load the red LUT with **entry.** The value **index** ranges from 0 to (2(pixel depth)-1). Beginning with the next vertical retrace, the color value **index** will be displayed using the new **entry** as the red intensity. For the range of the value **index** refer to Appendix D in the Graphics Processor Manual.

Example :

```
VALUE 8                    ;Set current pixel value to 8
FLOOD                      ;Flood the current update buffer to
                           ;current pixel value
LUTR 8 0                   ;Change entry in red LUT location
                           ;8 to 0 (Black)
LUTR 8 15                  ;Change entry in red LUT location
                           ;8 to 15 (full intensity)
VALUE 0                    ;Change current pixel value to 0
FLOOD                      ;Flood the current update buffer to
                           ;current pixel value
LUTR 0 14                  ;Change entry in red LUT location 0
                           ;to 14
```

Object Code Format :

[18H][**index**][**entry**] (3 bytes)

Affected by : Blink Status

Affects : Lookup Tables

Command available Version ≥ 1.0

LUTRST   Reset LUT values.

Reset the LUTs to the default values. Refer to Appendix D for a list of these values. Turns off blinking.

Example :

```
LUT8 2 555                ;Set color 2 to gray
LUTRST                    ;Reset the default LUT values (sets
                          ;color 2 to red)
```

Object Code Format :

[F6H] (lbyte)

Affected by : NONE

Affects :	Blink Status Blink Tables Lookup Tables

Command available Version ≥ 4.0

LUT8

LUT8 **index,rentry,**
gentry,bentry		Make entry in all three LUTS.

Changetheentries in the red, green and blue look-up tables (LUTs). At the offset **index** in each LUT, load the red LUT with **rentry**, the green LUT with **gentry**, and the blue LUT with **bentry.** Thevalue **index** ranges from 0 to (2(pixel depth)-1). Beginning with the next vertical retrace, the color value **index** will be displayed as a combination of the intensities **rentry, gentry,** and**bentry.** For the range of the value **index** refer to Appendix D in the Graphics Processor Manual.

Example :

```
VALUE 8                     ;Change current pixel value to 8
FLOOD                       ;Flood the current update buffer to the
                            ;current pixel value
LUT8 8 6 8 4                ;Change location 8 in red LUT to 6
                            ;in green LUT to 8, and blue LUT to 4
```

Object Code Format :

[1CH][**index**][**rentry**][**gentry**][**bentry**] (5 bytes)

Affected by : Blink Status

Affects : Lookup Table

Command available Version ≥ 1.0

MACDEF **macnum** Define a macro.

Define INTERACT macro **macnum**, where the value **macnum** varies between 0 and 255. The string following the MACDEF command and ending with the MACEND command specifies a macro. The string can consist of any combination of valid INTERACT command strings (commands and parameters), excluding the commands WARM, COLD, and CONFIG. Only the available memory space limits the length of the MACDEF command string. (Refer to the CONFIG command.) Macro definitions may nest up to 16 levels deep. Definition of a previously defined macro will result in automatic erasure of the original definition.

Example :

```
MACDEF 23                 ;Begin macro definition
MOVABS 0 0                ;Move current point to 0,0
VALUE 4                   ;Set current pixel value to 4
CIRCLE 25                 ;Draw a circle of radius 25
MOVABS -25 -25            ;Displace current point to -25,-25
VALUE 2                   ;Set current pixel value to 2
RECREL 50 50              ;Draw a square around the circle
MACEND                    ;End macro definition
MACRUN 23                 ;Run this macro
```

Object Code Format :

[8BH][**macnum**] (2 bytes)

Affected by : RAM Configuration

Affects : Macro Definition Table

Command available Version ≥ 1.0

MACEND

MACEND   End of macro definition.

End a macro definition. If no MACDEF command has preceded a MACEND command, no action will occur. A MACEND command must occur for each MACDEF command.

Example :

```
MACDEF 23                ;Begin macro definition
MOVABS 0 0               ;Move current point to 0,0
VALUE 1                  ;Set current pixel value to 1
CIRCLE 25                ;Draw circle of radius 25

MACDEF 16                ;Define macro 16
VALUE 5                  ;Set current pixel value to 5
FLOOD                    ;Flood the current update buffer to
                         ;current value
MACEND                   ;End definition of macro 16

                         ;Continue with MACDEF 23
MOVABS -25 -25           ;Displace current point to perimeter
RECREL 50 50             ;Draw a square around the circle
MACEND                   ;End definition of macro 23

MACRUN 16                ;Run macro 16
MACRUN 23                ;Run macro 23
MACRUN 16                ;Run macro 16
```

Object Code Format :

[0CH] (1 byte)

Affected by : NONE

Affects : NONE

Command available Version ≥ 1.0

MACERA **macnum** Erase macro.

Erase the definition of macro **macnum.** The space in the macro buffer used by macro **macnum** becomes available for another macro definition.

Example :

```
MACDEF 18               ;Begin macro definition
MOVABS 0 0              ;Move current point to 0,0
VALUE 0                 ;Set current pixel value to 0
FLOOD                   ;Flood current update buffer with
                        ;current pixel value
VALUE 1                 ;Set current pixel value to 1
CIRCLE 25               ;Draw a circle of radius 25
MOVABS -25 -25          ;Displace current point to -25,-25
RECREL 50 50            ;Draw a square around the circle
MACEND                  ;End macro definition
MACRUN 18               ;Run this macro
MACERA 18               ;Erase this macro
MACRUN 18
```

Object Code Format :

[8CH][**macnum**] (2 bytes)

Affected by : NONE

Affects : NONE

Command available Version ≥ 2.0

MACREP

MACREP **macnum,count**   Repeat macro.

Execute the previously defined macro **macnum** **count** times. If **count**=0, repeat indefinitely. This command may appear within a macro definition.

Example:

```
MACDEF 17                   ;Begin macro definition
MOVREL 1 1                  ;Move current point one pixel
                            ;diagonally
VALUE 4                     ;Set current pixel value to 4
CIRCLE 25                   ;Draw a circle of radius 25
MACEND                      ;End macro definition
MACREP 17 500               ;Repeat macro number 17 500 times
```

Object Code Format:

[BBH][**macnum**][high**count**][low**count**] (4 bytes)

Affected by :   NONE

Affects :   NONE

Command available Version ≥ 2.0

MACRUN **macnum** Execute macro.

Execute the previously defined macro **macnum**.

Example :

```
MACDEF 18                ;Begin macro definition
MOVABS 0 0               ;Move current point to 0,0
VALUE 1                  ;Set current pixel value to 1
CIRCLE 25                ;Draw a circle of radius 25
MOVABS -25 -25           ;Displace current point to perimeter
VALUE 4                  ;Set current pixel value to 4
RECREL 50 50             ;Draw a square around the circle
MACEND                   ;End macro definition
MACRUN 18                ;Run this macro
```

Object Code Format :

[0BH][**macnum**] (2 bytes)

Affected by : NONE

Affects : NONE

Command available Version ≥ 1.0

MOVABS

MOVABS **x,y** Move absolute to the point **x,y.**

Move from the current point (CREG 0) to the point **x,y.** The values **x** and **y** range from -32,768 to +32,767.

Example :

```
MOVABS 50 70                  ;Move current point to 50,70
VALUE 1                       ;Set current pixel value to 1
DRWABS 100 -10                ;Draw line from 50,70 to 100,-10
CIRCLE 15                     ;Draw a circle of radius 15
                              ;centered at 100,-10
VALUE 2                       ;Set current pixel value to 2
MOVABS 0 0                    ;Move current point to 0,0
CIRCLE 20                     ;Draw a circle of radius 20
                              ;centered at 0,0
```

Object Code format :

[01H][high**x**][low**x**][high**y**][low**y**] (5 bytes)

6 0 700

Affected by : NONE

Affects : Current Point

Command available Version ≥ 1.0

MOVI

MOVI **creg** Move to the point specified in **creg.**

Move from the current point (CREG 0) to the point stored in coordinate register **creg.** The value **creg** ranges from 0 to 63. This command effectively performs the command "CMOVE 0 **creg**" which transfers a given coordinate register into CREG 0.

Example :

```
CLOAD 15 100 150            ;Load 100,150 into CREG 15
VALUE 5                     ;Set current pixel value to 5
MOVI 15                     ;Move to location given in CREG 15
DRWABS 140 100              ;Draw line from 100,150 to 140,100
MOVI 2                      ;Move to the location given in CREG 2
CIRCLE 25                   ;Draw circle of radius 25 at current
                            ;point
```

Object Code Format :

[05H][**creg**] (2 bytes)

Affected by : Coordinate Register **creg**

Affects : Current Point

Command available Version $\geq$ 1.0

MOVREL

MOVREL **dx,dy**   Move relative by **dx,dy.**

Move from the current point (CREG 0) to a point displaced in the x-direction by **dx** and in the y-direction by **dy.** The values of **dx** and **dy** range from -32,768 to +32,767. The new current point updates to the sum of the x-component of the previous current point plus **dx** and the sum of the y-component of the previous current point plus **dy.**

Example :

```
MOVABS 100 -130          ;Move current point to 100,-130
MOVREL 50 100            ;Move current point by 50,100 to 150,-30
VALUE 3                  ;Set current pixel value to 3
CIRCLE 30                ;Draw circle of radius 30 centered
                         ;at current point
MOVREL 20 20             ;Move current point by 20,20 to 170,-10
CIRCLE 10                ;Draw circle of radius 10 centered
                         ;at current point
MOVREL -20 -20           ;Move current point by -20,-20 to 150,-30
CIRCLE 25                ;Draw circle of radius 25 centered
                         ;at current point
```

Object Code Format :

[02H][high**dx**][low**dx**][high**dy**][low**dy**] (5 bytes)

Affected by : Current Point

Affects : Current Point

Command available Version ≥ 1.0

MOV2R **dxdy** Move short relative.

Move from the current point to a point offset in the x direction by **dx** and in the y direction by **dy.** MOV2R requires three fewer bytes than MOVREL, but the command restricts the range of **dx** and **dy** from -8 to +7. The most significant nibble of **dxdy** specifies **dx** and the least significant four bits specify **dy.**

Example :

```
MOVABS 0 0                ;Move current point to 0,0
MOV2R 5 5                 ;Move relative to 5,5
```

Object Code Format :

[04H][**dxdy**] (2 bytes)

Affected by : Current Point

Affects : Current Point

Command available Version $\geq$ 2.0

MOV3R

MOV3R **dx,dy** Move short relative.

Move from the current point to a point offset in the x direction by **dx** and in the y direction by **dy.** MOV3R requires only three bytes than MOVREL, but the command restricts the range of **dx** and **dy** from -128 to +127.

Example :

```
MOVABS 0 0                ;Move current point to 0,0
MOV3R 50 60               ;Move relative to 50,60
```

Object Code Format :

[03H][**dx**][**dy**] (3 bytes)

Affected by : Current Point

Affects : Current Point

Command available Version $\geq$ 2.0

PIXDMP **depth,dx,dy** Output pixels of defined window.

The current point defines the lower left corner of a rectangle with dimensions **dx, dy.** Beginning with this corner and proceeding left to right and bottom to top, each pixel in the current update buffer gets read, compressed by run-length encoding, and transmitted to the host. The output appears as a bit stream where each **depth** bits represents a new pixel. Run-length data, however, always consists of full, eight-bit lengths. (See Section 3.7 for the run-length encoding description.)

Example :

```
MOVABS -40 60             ;Move to lower left corner of rectangle
PIXDMP 4 120 80           ;Read four least significant bits of
                          ;each pixel in a 120 x 80 pixel
                          ;rectangle
```

Object Code Format :

[F0H][**depth**][high**dx**][low**dx**][high**dy**][low**dy**] (6 bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Update Buffer
Bit Plane Mask

Affects : NONE

Command available Version $\geq$ 2.0

PIXELS

PIXELS **x,y,color,...**   Load a rectangular array of pixels in image memory.

Load a rectangular array of pixels with the values in the string **color,....** The current point specifies the lower left corner of the array. The **x** and **y** values define the width and height dimensions of the array. The pixel array is written left to right, bottom to top.

Example :

```
PIXELS 1 2 7 10           ;Load a pixel array, consisting of the
                          ;current point and the point above it,
                          ;to value 7 at the current point, and
                          ;value 10 on the other
```

Object Code Format :

[28H][high**x**][low**x**][high**y**][low**y**][**color**]...   (5+**x*****y** bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Bit Plane Mask

Affects :   NONE

Command available Version ≥ 1.0

PIXFUN **mode** Set pixel processor mode.

Set the mode of operation executed by the pixel processor. All operations performed by the pixel processor affect image memory. The **mode** parameter specifies the operation performed by the pixel processor. The values for **mode** are 0 , 1, and 2. INTERACT defines the mode values as follows:

Function	Mode	Operation
INSERT	0	Insert new data directly (Default)
COMPLEMENT	1	Complement image data
XOR	2	XOR new data to image data

Example :

```
VLOAD 6 15                ;Load VREG 6 with color value 15
VALUE 5                   ;Set current pixel value to 5
PRMFIL 1                  ;Enable filled figures
CIRCLE 30                 ;Draw a cyan circle with radius 30
VALUE 7                   ;Set current pixel value to 7
PIXFUN 2                  ;XOR new dat to image data
CIRCLE 30                 ;Draw a red circle with radius 30
PIXFUN 1                  ;Complement image data
CIRCLE 30                 ;Draw a magenta circle with radius 30
```

Object Code Format :

[3BH][**mode**] (2 bytes)

Affected by : NONE

Affects : Pixel Function

Command available Version $\geq$ 2.0

PIXLOD

PIXLOD **depth,dx,dy,** **bitstream**	Load a stream of pixels into the specified window.

The current point defines the lower left corner of a rectangle with dimensions **dx, dy.** The **bitstream** defines a group of **depth**-deep pixels which produce the rectangle starting at the lower left corner and proceeding left to right and bottom to top. (See Section 3.7 for the run length encoding description.)

Example :

```
MOVABS 20 80                   ;Define lower left corner of
                               ;rectangle
PIXLOD 8 10 10 20 2 20
       1 20 2 20 1 20 2 0      ;Draw red and white horizontal
                               ;stripes
```

Object Code Format :

[F1H] [**depth**] [high**dx**] [low**dx**] [high**dy**] [low**dy**] [**bitstream**]
(6 bytes + length of bitstream)

Affected by :	Current Point Coordinate Origin Clip Window Update Buffer Pixel Function
Affects :	NONE

Command available Version $\geq$ 2.0

POINT   Set current point to current pixel value.

Set the pixels located at the current point (CREG 0) to the current pixel value (VREG 0). The current point and the current pixel value remain unchanged.

Example :

```
VALUE 8                  ;Set current pixel value to 8
MOVABS 100 100           ;Move current point to location 100,100
POINT                    ;Set pixel at location 100,100 to 8
MOVREL 1 0               ;Move current point by 1,0 to 101,100
POINT                    ;Set pixel at location 101,100 to 8
VALUE 2                  ;Set current pixel value to 2
MOVREL 1 1               ;Move current point by 1,1 to 102,101
POINT                    ;Set pixel at 102,101 to 2
```

Object Code Format :

[88H] (1 byte)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Current Value
Bit Plane Mask

Affects : NONE

Command available Version $\geq$ 1.0

POLYGN

POLYGN **npoly,nvert1,**
x1,y1,x2,y2,
x3,y3,...,xnvert,ynvert

Draw polygons in current color with specified vertices.

Draw a polygon with verticies at the absolute coordinates **x1, y1,...,xnvert,ynvert.** Each x- and y-value may range from -32,768 to +32,767. The value **nvert** specifies the number of vertices for each polygon. The list progresses in a "connect-the-dots" fashion, with the last point connected back to the first. The value **npoly,** which may vary between 0 and 255, determines the number of multiple polygons the command will draw. For unfilled polygons, **nvert** ranges from 0 to 32768, but for filled polygons, the maximum value of **nvert** depends on the amount of free memory available on the VM-885x (see CONFIG). For multiple filled polygons, the areas to be filled are determined by an algorithm which scans the figure from left to right at each horizontal line. If the leftmost edge is designated as edge number 1, the filling algorithm fills the area between each odd left edge and even right edge, but leaves unfilled the area between each even left edge and odd right edge.

Example :

```
VALUE 1                 ;Set current pixel value to 1 (white)
PRMFIL 1                ;Enable filled figures
POLYGN 1 3 0 0 40 0     ;Draw filled triangle
       20 20
PRMFIL 0
POLYGN 2 4 -100 -100
       100 -100 100 100
       -100 100
       4 -50 -50 50 -50
       50 50 -50 50     ;Draw outlines of two squares
```

Object Code Format :

[12H][**npoly**]{[high**nvert1**][low**nvert1**]
    ([high**x1**][low**x1**][high**y1**][low**y1**]...)
  [high**nvert2**][low**nvert2**]
    ([high**x2**][low**x2**][high**y2**][low**y2**]...)}
(2 bytes + (2***npoly** + 4(**nvert1** + **nvert2** +...)) bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Vector Pattern
Current Value
Bit Plane Mask
RAM Configuration
Update Buffer
Primitive Fill Flag

Affects : NONE

Command available Version ≥ 2.0

POLYRL

POLYRL npoly,nvertl, dxl,dyl,... Draw relative polygon in current color.

Draw a polygon with verticies **xl, yl,...,xnvert,ynvert** relative to the current point. Each x- and y-value may range from -32,768 to +32,767. The value **nvert** specifies the number of vertices for each polygon. The list progresses in a "connect-the-dots" fashion, with the last point connected back to the first. The value **npoly,** which may vary between 0 and 255, determines the number of multiple polygons the command will draw. For unfilled polygons, **nvert** ranges from 0 to 32768, but for filled polygons, the maximum value of **nvert** depends on the amount of free memory available on the VM885x (see CONFIG). For multiple filled polygons, the areas to be filled are determined by an algorithm which scans the figure from left to right at each horizontal line. If the leftmost edge designated as edge number 1, the filling algorithm fills the area between each odd left edge and even right edge, but leaves unfilled the area between each even left edge and odd right edge.

Example :

```
MOVABS 0 0                    ;Move the current point to 0,0
VALUE 2                       ;Set current pixel value to 2 (red)
POLYRL 1 3 25 0               ;Draw a triangle
       25 25 0 25
```

Object Code Format :

```
[E6H][npoly]{[highnvertl][lownvertl]
             ([highxl][lowxl][highyl][lowyl]...)
           [highnvert2][lownvert2]
             ([highx2][lowx2][highy2][lowy2]...)}
(2 bytes + (2*npoly + 4(nvertl + nvert2 +...)) bytes)
```

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Vector Pattern
Current Value
Bit Plane Mask
RAM Configuration
Update Buffer
Primitive Fill Flag

Affects : NONE

Command available Version ≥ 4.0

PRMFIL

PRMFIL **flag** Set primitive fill flag.

If **flag**=0, subsequent polygon, rectangle, and circle commands draw vectors describing an outline. If **flag** = 1 or 2, subsequent commands describe filled figures. If flag=2, filled polygons will be drawn using a "quick" algorithm, but degenerate polygons will not draw properly.

Example :

```
VALUE 2                     ;Set current pixel value to 2 (red)
PRMFIL 1                    ;Set primitive fill flag
POLYGN 1 3 0 0 40 0         ;Draw red, filled triangle
       20 20

VALUE 3                     ;Set current pixel value to 3 (green)
PRMFIL 0                    ;Clear fill flag
POLYGN 1 3 0 0 40 0         ;Green outline around the same
       20 20                ;polygon
```

Object Code Format :

31

[1FH][**flag**] (2 bytes)

Affected by : NONE

Affects : Primitive Fill Flag

Command available Version ≥ 2.0

RDPIXR **vreg**	Place the pixel value found in image memory at the current point in **vreg.**

Read the pixel value from image memory at the current point (CREG 0) and place the value into VREG **vreg.**

Example :

```
VALUE 8                  ;Change current pixel value to 8
POINT                    ;Set current point to current value
RDPIXR 13                ;Read current point and place value in
                         ;VREG 13
READVR 13                ;Read VREG 13
```

Object Code Format :

[AFH][**vreg**] (2 bytes)

Affected by : Current Point
Coordinate Origin
Update Buffer

Affects : NONE

Command available Version ≥ 1.0

READBU

READBU **flag,cflag** Read button number.

Read values from the button FIFO event queue. Eight events compose the queue, each event consisting of a button number, the crosshair coordinate (CREG 5), and the input device coordinate (CREG 2). These coordinates are recorded as the button command starts to execute. Reading back an event will erase the event from the queue. If **flag=0**, the oldest event (least recent) gets read. If there are no events in the queue, a **butnum** of 0FFH is returned. Setting **flag**=1 clears the queue and sends the values for the next button after execution of the next button command. Setting **cflag**=0 sends the coordinate of the crosshair (CREG 5), while **cflag**=1 sends the coordinate of the locator device, (CREG 2).

Example :

```
READBU 0 1                ;Read back from the next event (least
                          ;recent) in the event queue the button
                          ;number and the coordinates saved for
                          ;CREG 2
```

Object Code Format :

[9AH][**flag**][**cflag**] (3 bytes)

Response :

[**butnum**][highx][lowx][highy][lowy] (5 bytes)

Affected by : Button FIFO Event Queue

Affects : Button FIFO Event Queue

Command available Version ≥ 2.0

READCR **creg** Read the coordinate register **creg.**

Send the contents of coordinate register **creg** to the port available for readback by the host. The value of **creg** ranges from 0 to 63.

Example :

```
CLOAD 15 120 340          ;Load CREG 15 with 120 340
READCR 15                 ;Read CREG 15
```

Object Code Format :

[98H][**creg**] (2 bytes)

Response :

[high**x**][low**x**][high**y**][low**y**] (4 bytes)

Affected by : NONE

Affects : NONE

Command available Version ≥ 1.0

READP

READP   Read pixel value.

Read back the value of the pixel at the current point.

Example :

```
MOVABS 10 50                 ;Move current point to 10,50
VALUE 9                      ;Set current value to 9
POINT                        ;Set pixel at 10,50 to value 9
READP                        ;Read the value of the pixel at 10,50
```

Object Code Format :

[95H] (1 byte)

Response :

[**value**] (1 byte)

Affected by : Current Point
               Coordinate Origin
               Update Buffer

Affects :   NONE

Command available Version ≥ 1.0

READVR **vreg** Read the value register **vreg.**

Read back the contents of value register **vreg** specified. The value of **vreg** ranges from 0 to 15.

Example :

```
VLOAD 15 7                ;Load VREG 15 with 7
READVR 15                 ;Read VREG 15
```

Object Code Format :

[99H][**vreg**] (2 bytes)

Response :

[**value**] (1 byte)

Affected by : NONE

Affects : NONE

Command available Version ≥ 1.0

RECREL

RECREL **dx,dy** Draw rectangle relative.

Draw a rectangle in image memory with one corner at the current point (CREG 0) and a diagonally opposite corner displaced relative to the current point by **dx** in the x-direction and by **dy** in the y-direction. The rectangle draws in the current color (VREG 0). The values **dx** and **dy** range from -32,768 to 32,767. The current point remains fixed.

Example :

```
MOVABS 100 150          ;Move current point to 100,150
VALUE 6                 ;Set current pixel value to 6
RECREL 10 10            ;Draw rectangle with diagonally
                        ;opposite corner displaced by 10,10
                        ;to 110,160
VALUE 7                 ;Set current pixel value to 7
RECREL -20 -30          ;Draw rectangle with diagonally
                        ;opposite corner displaced by -20,-30
                        ;to 80,120
```

Object Code Format :

[89H][high**dx**][low**dx**][high**dy**][low**dy**] (5 bytes)

Affected by :	Current Point Coordinate Origin Clipping Boundary First Pixel Flag Pixel Function Primitive Fill Flag Vector Pattern Current Value Bit Plane Mask Area Pattern Update Buffer
Affects :	NONE

Command available Version $\geq$ 1.0

RECTAN x,y — Draw rectangle. Point **x,y** specifies diagonal corner.

Draw a rectangle with one corner located at the current point (CREG 0) and the diagonally opposite corner located at the point **x,y**. The values **x** and **y** range from -32,768 to +32,767.

Example :

```
VALUE 6                   ;Set current pixel value to 6
MOVABS 30 50              ;Move current point to 30,50
RECTAN 70 100             ;Draw rectangle whose corners are
                          ;located at 30,50  30,100  70,100  70,50
VALUE 7                   ;Set current pixel value to 7
MOVABS -20 -10            ;Move current point to -20,-10
RECTAN -25 15             ;Draw rectangle
```

Object Code Format :

[8EH][high**x**][low**x**][high**y**][low**y**] (5 bytes)
142

Affected by : Current Point
Coordinate Origin
Clipping Boundary
First Pixel Flag
Pixel Function
Primitive Fill Flag
Vector Pattern
Current Value
Bit Plane Mask
Area Pattern
Update Buffer

Affects : NONE

Command available Version $\geq$ 1.0

RECTI

RECTI creg — Draw rectangle. Location in **creg** is diagonal corner.

Draw a rectangle with one corner located at the current point (CREG 0) and the diagonally opposite corner located at the point stored in coordinate register **creg.** The value **creg** ranges from 0 to 63. Version 2.0 "clips" any portion of the rectangle which falls outside of the display boundary.

Example :

```
   VALUE 12                 ;Set current pixel value to 12
01 MOVABS -20 -100          ;Move current point to -20,-100
A0 CLOAD 17 50 70           ;Load 50,70 into CREG 17
8F RECTI 17                 ;Draw rectangle whose corners are 50,70
                            ;50,-100 -20,-100 -20,70
   VALUE 13                 ;Set current pixel value to 13
   CLOAD 18 40 60           ;Load 40,60 into CREG 18
   RECTI 18                 ;Draw rectangle whose corners are 40,60
                            ;40,-100 -20,-100 -20,60
```

Object Code Format :

[8FH][**creg**] (2 bytes)

Affected by :
Current Point
Coordinate Origin
Clipping Boundary
First Pixel Flag
Pixel Function
Primitive Fill Flag
Vector Pattern
Current Value
Bit Plane Mask
Area Pattern
Update Buffer

Affects : NONE

Command available Version ≥ 1.0

SURFAC **count,p1,p2,...** Establish surface priorities.

For a discussion of surface priorities, see Section 2.12. See the appropriate Graphics Processor Manual for acceptable parameters.

Example :

```
                                ;Example is specific to 8 bit plane
                                ;graphics processor
SURFAC 2 0F0H 0FH               ;Set surface priority to front half
VALUE 0C0H                      ;Value 0C0H draws only into
                                ;upper bit planes
TEXT1 "TEST"                    ;Draw and display text
VLOAD 6 0FH                     ;Mask upper bit planes
VALUE 3                         ;Value 3 draws only into
                                ;lower bit planes
PRMFIL 1                        ;Enable filled figures
CIRCLE 100                      ;Draw circle
SURFAC 2 0FH 0F0H               ;Text disappears; color of circle
                                ;has priority over color of text
```

Object Code Format :

[F5]**[count]**[**p1**][**p2**]...[**pn**] ((2 + n) bytes)

Affected by : NONE

Affects : Surface Priorities

Command available Version ≥ 4.0

Have 3.4

TRY

TEXTB

TEXTB **flag**   Set flag to select background attribute

The TEXTB command selects the background attribute of text drawn with the TEXT1 and TEXT0 commands. If **flag** = 1, the background of each text cell is filled with the color value specified in VREG5 before the text character is drawn. If **flag** = 0 , no background color is drawn.

Example :

```
    VALUE 1                   ;Set current pixel value to 1    06,01
147 TEXT0 "This is a test"    ;Draw text  with no background  147,2,65,66
    MOVABS 0 20               ;Move the current point to 0,20  01,00,00,00,20
    TEXTB 1                   ;Select a background to be drawn 148,1
164 VLOAD 5,3                 ;Select value 3 as background color 164,5,3
    TEXT0 "Test background"   ;Draw text with background       147,1,65
    147
```

Object Code Format :

[94H][**flag**] (2 bytes)
148

Affected by :   NONE

Affects :   Text Background Flag

Command available Version ≥ 4.0

TEXTC

TEXTC size,angle Set **size** and **angle** for TEXT0 command.

The TEXTC command should occur before a TEXT0 command to specify the size of character desired. The **size** parameter may vary from 0 to 255 with zero corresponding to a 5 x 7 pixel character font. The **angle** parameter may vary from -32,768 to +32,767. It specifies the rotation angle in degrees for TEXT0. INTERACT V4.0 does not support rotation.

Example :

```
VALUE 11                     ;Set current pixel value to 11
TEXTC 2 0                    ;Set size to 2 (10 x 14)
TEXTO "This is a test"       ;Draw large text
MOVABS -220 -150             ;Move the current point to -220,-150
VALUE 10                     ;Set current pixel value to 10
TEXTC 20 0                   ;Set size to 20 (133 x 171)
TEXTO "BIG!"                 ;Draw enormous text
```

Object Code Format :

[92H][**size**][high**angle**][low**angle**] (4 bytes)

14

Affected by : NONE

Affects : Text Size

Command available Version ≥ 2.0

TEXTDN

TEXTDN **char,x,y,fntlst**   Define fonts for TEXT2.

Define the character image for the character **char** in font 2. The parameters **x** and **y** define the width and height of the character cell respectively. The bytes in the **fntlst** define the pixel information needed to construct the character. The value **char** ranges from 0 to 255. The values **x** and **y** range from 0 to 32,767. Refer to Section 3 of this manual for further detail on the format of **fntlst.** If a character definition exceeds available RAM, the definition will be ignored.

Example :

```
TEXTDN 65 5 5 32 32 248 32 32     ;Define the character "A" in
                                  ;font2 to be a small cross
VALUE 7                           ;Set current pixel value to 7
TEXT2 "A"                         ;Draw a small cross
```

Object Code Format :

[26H][**char**][high**x**][low**x**][high**y**][low**y**][**fntlst**]...
(6+**y***INT((**x**+7)/8) bytes)

Affected by : RAM Configuration

Affects :   NONE

Command available Version ≥ 1.0

TEXTO **string** Draw string in current size characters.

This command draws the given character string at the current location and in the current color and size. The TEXTC command sets the size. The value **string** specifies the text. The first byte of **string** contains the number of characters in the string (**strlen**) followed by **strlen** bytes containing the ASCII characters to be drawn.

The command produces larger characters by expanding the basic font definitions and then algorithmically smoothing the edges to avoid "blocky" looking characters. The current location defines the lower left corner of the first character cell. Each subsequent character appears to the right on a horizontal line. (INTERACT does not support the angle parameter of TEXTC.) The first byte of **string** gives the length of the text string in bytes and may range from 0 to 255.

Example :

```
VALUE 1           96,             ;Set current pixel value to 1
TEXTC 2 0         196, [illegible]  ;Set size to 2 (10 x 14)
TEXTO "This is a test"            ;Draw large text  147,
MOVABS -250 -100                  ;Move current point to -250,-100   01, [illegible]
VALUE 2           96, [illegible] ;Set the current pixel value to 2
TEXTC 20 0        [illegible] 0   ;Set size to 20 (133 x 171)
TEXTO "BIG!"      147, 4          ;Draw enormous text
```

Object Code Format :

[93H][strlen][char1][char2]... ((2+**strlen**) bytes)

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Text Background Color
Text Background Flag
Text Size
Current Value
Bit Plane Mask
Update Buffer

Affects : Text Endpoint

Command available Version ≥ 2.0

TEXT1 **string** Draw text string with font 1.

Draw horizontal text into image memory using font 1. Text drawn with font 1 appears as 5x7 dot matrix characters in 8x8 cells. The value **string** specifies the text. The first byte of **string** contains the number of characters in the string (**strlen**) followed by **strlen** bytes containing the ASCII characters to be drawn. The current point (CREG 0) specifies the lower left corner of the first character cell and remains unchanged. Subsequent characters are placed horizontally to the right at 8 pixel increments. Strings which cross the right clipping boundary will wrap around and continue at the left margin with a downward shift of one cell. CREG7 updates to the new end point of the text, ie., the lower left hand corner of the next cell space.

Example :

```
VALUE 1                  ;Set current pixel value to 1
TEXT1 "12345"            ;Draw text string 12345
MOVABS 0 20              ;Move current point to 0,20
TEXT1 "wxyz"             ;Draw text string wxyz
MOVABS 20 0              ;Move current point to 20,0
TEXT1 041H 042H 043H     ;Draw text string "ABC"
```

Object Code Format :

[90H][**strlen**][**char1**][**char2**]...[**charn**] ((2+**strlen**) bytes)

144

Affected by : Current Point
Coordinate Origin
Clipping Boundary
Pixel Function
Current Value
Text Background Color
Text Background Flag
Bit Plane Mask
Update Buffer

Affects : Text Endpoint

Command available Version ≥ 1.0

TEXT2

TEXT2 **string** Draw text string with font 2.

Draw variable-cell text into image memory using font 2. The TEXTDN command defines the text drawn with font 2. The value **string** specifies the text. The first byte of **string** contains the number of characters in the string (**strlen**) followed by **strlen** bytes containing the ASCII characters to be drawn. The current point (CREG 0) specifies the lower left corner of the first character cell and remains unchanged. Subsequent characters appear horizontally adjacent to the right. Strings exceeding the image width are clipped. CREG7 updates to the new end point of the text, i.e., the lower left hand corner of the next cell space.

Example :

```
TEXTDN 65 5 5 32 32 248 32 32    ;Define the character "A" in
                                 ;font2 to be a small cross
VALUE 7                          ;Set current pixel value to 7
TEXT2 "A"                        ;Draw a small cross
```

Object Code Format :

[91H][**strlen**][**char1**][**char2**]...[**charn**] ((2+**strlen**) bytes)

145

Affected by :	Current Point
	Coordinate Origin
	Clipping Boundary
	Pixel Function
	Current Value
	Bit Plane Mask
	Update Buffer

Affects : Text Endpoint

Command available Version ≥ 1.0

VADD **vsum,vreg** Add the contents of one VREG to another.

Add the value in the VREG specified by **vreg** to the value in VREG **vsum,** leaving the result in VREG **vsum.**

Example :

```
VLOAD 14 5                ;Load VREG 14 with 5
VLOAD 15 3                ;Load VREG 15 with 3
VADD 15 14                ;Add values of VREGs 14 and 15;
                          ;place result (8) in VREG 15
```

Object Code Format :

[A6H][**vsum**][**vreg**] (3 bytes)

Affected by : NONE

Affects : Value Register **vreg**

Command available Version ≥ 2.0

VALUE

VALUE color Set the current pixel value to **color.**

Change the current pixel value (VREG 0) to the value **color.** The value **color** is a byte. All graphics primitives which write into image memory use VREG 0, the current pixel value.

Example :

```
VALUE 8                 ;Set current pixel value to 8
MOVABS -10 25           ;Move current point to -10,25
DRWABS 50 -30           ;Draw line from current point to 50,-30
                        ;in current pixel value
VALUE 10                ;Set current pixel value to 10
MOVABS 50 100           ;Move current point to 50,100
CIRCLE 50               ;Draw circle of radius 50 at current
                        ;point
```

Object Code Format :

[06H][**color**] (2 bytes)

Affected by : NONE

Affects : Current Color

Command available Version ≥ 1.0

VECPAT **mask** Set vector pattern mask.

Set the 16-bit vector pattern to the value given. The bits of the pattern are drawn for bits set to "1" while bits set to "0" do not appear. The value for **mask** ranges between 0 to 65,535.

Example:

```
VALUE 1                  ;Set current pixel value to 1
VECPAT 0F0F0H            ;Set vector pattern to four pixels
                         ;on, four pixels off, four pixels
                         ;on, four pixels off
CIRCLE 100               ;Draw a circle with radius 100
DRWABS 250,0             ;Draw a patterned horizontal line of
                         ;length 250 pixels
```

Object Code Format:

[2EH][high**mask**][low**mask**] (3 bytes)

Affected by : NONE

Affects : Vector Pattern

Command available Version $\geq$ 2.0

VLOAD

VLOAD vreg,color Load value register **vreg** with **color.**

Load the value register **vreg** with the pixel value **color.** The parameter **vreg** ranges from 0 to 15.

Example :

```
VLOAD 13 8                 ;Load VREG 13 with pixel value 8
CIRCLE 20                  ;Draw a circle in value 8
```

Object Code Format :

[A4H][**vreg**][**color**] (3 bytes)

Affected by : NONE

Affects : Value Register **vreg**

Command available Version $\geq$ 1.0

VMOVE **vdst,vsrc** Move contents of **vsrc** into **vdst.**

Load the value register **vdst** with the pixel value stored in the value register **vsrc.** The parameters **vdst** and **vsrc** range from 0 to 15.

Example :

```
VLOAD 10 8                ;Load VREG 10 with 8
VMOVE 11 10               ;Move contents of VREG 10 into VREG 11
```

Object Code Format :

[A5H][**vdst**][**vsrc**] (3 bytes)

Affected by : Value Register **vreg**

Affects : VREG **vdst**

Command available Version ≥ 1.0

VSUB

VSUB **vdif,vreg**	Subtract the contents of one VREG from another.

Subtract the value in the VREG specified by **vreg** from the value in VREG **vdif,** leaving the result in VREG **vdif.**

Example :

```
VLOAD 15 5                ;Load VREG 15 with 5
VLOAD 14 3                ;Load VREG 14 with 3
VSUB 15 14                ;Subtract value of VREG 14
                          ;from value in VREG 15.  Place
                          ;result in VREG 15.
```

Object Code Format :

[A7H][**vdif**][**vreg**] (3 bytes)

Affected by :	Value Register **vdif** Value Register **vreg**
Affects :	Value Register **vdif**

Command available Version ≥ 2.0

WAIT frames Wait specified time before continuing.

Wait for **frames** frame times (each frame time equals one vertical sync period) before continuing command execution. Use this command to choreograph graphic displays and to synchronize updates with vertical blanking. The value **frames** ranges from 0 to 65,535.

Example :

```
WAIT 600                  ;Pause for 10 seconds before
                          ;continuing command execution
```

Object Code Format :

[3DH][high**frames**][low**frames**] (3 bytes)

Affected by: NONE

Affects : NONE

Command available Version ≥ 1.0

WARM

WARM Warm start the graphics processor.

Terminate execution of the current command. Reset the serial input and output buffer pointers on the current channel and jump to the INTERACT command processor, to await further input. This command is useful only when invoked by an asynchronous warm start. (See Sections 5.1 and 5.3.)

Example :

```
WARM                        ;Reset INTERACT communication link
```

Object Code Format :

[FEH] (1 byte)

Affected by : NONE

Affects : NONE

Command available Version ≥ 1.0

WINDOW **x1,y1,x2,y2**   Set current clipping window.

Set the current clipping window to the rectangle specified by **x1,y1,x2,y2.** One corner of the window is specified by **x1,y1,** the other corner by **x2,y2.** The coordinate register CREG 9 is loaded with the **x1,y1** coordinates, coordinate register CREG 10 is loaded with the **x2,y2** coordinates. All graphics primitives are clipped to the current window. The x,y-values range from -32,768 to +32,767. Those limits also serve as the default values for **x1,y1** and **x2,y2** respectively.

Example:

```
WINDOW 0 0 50 50          ;Define window
VALUE 1                   ;Set current pixel value to 1
CIRCLE 50                 ;Draw a circle of radius 50
```

Object Code Format:

[3AH][high**x1**][low**x1**][high**y1**][low**y1**][high**x2**][low**x2**][high**y2**][low**y2**]
(9 bytes)

Affected by : NONE

Affects :   Clipping Boundary

Command available Version ≥ 3.0

XHAIR

XHAIR **num,flag** Enable or disable crosshair **num.**

For **flag**=1, enable crosshair number **num.** If **flag**=0, disable crosshair number **num.** The value **num** equals 0 or 1. The crosshair positions for crosshairs 0 and 1 originate from CREG 5 and 6 respectively. The center of each crosshair remains unfilled to allow the user to locate individual pixels.

Example :

```
VLOAD 1 1                  ;Load XHAIR color
XHAIR 0 1                  ;Enable crosshair 1
CLOAD 5 100 100            ;Move XHAIR
```

Object Code Format :

[9CH][**num**][**flag**] (3 bytes)

```
Crosshair draw
affected by :   Coordinate origin
                Crosshair 0 Location
                Corsshair 1 Location
                Crosshair 0 Color
                Crosshair 1 Color
                Display Buffer
                Xhair Enable Flags
```

```
Affects :       Xhair Enable Flags
```

Command available Version ≥ 1.0

ZOOM **fact,bdst,bsrc**   Buffer to buffer ZOOM copy.

Copy source buffer to destination buffer with magnification fact. The buffer selected by **bsrc** becomes the source image. The buffer **bdst** receives the adjusted image. The value **fact** can equal 1, 2, 4, or 8. The values **bsrc** and **bdst** can be any valid buffer numbers (**bsrc** is not equal to **bdst**).

Example :

```
45  IMGSIZ 512 512 8          ;Set image size
65H DSPSIZ 512 512 60 1       ;Draw power-up screen into buffer 0
    ZOOM 4 1 0                ;Change scale on buffer 0, and place
                              ;scaled image in buffer 1
    BUFFER 1 1                ;Update into buffer 1, and display
                              ;zoomed image
```

Object Code Format :

[34H][**fact**][**bdst**][**bsrc**] (4 bytes)

52

Affected by : Current Point
               Coordinate Origin

Affects :   NONE

Command available Version ≥ 1.0

62 02 00 02 00 08

224 01

5
System Interfacing

The interface to INTERACT depends on the graphics hardware environment in which the software executes. Available interfaces include Programmed I/O, DMA, and RS-232C. The following sections describe the software protocols used to drive these interfaces.

5.1 - Programmed I/O Interface

Summary: Write data to the board for status bit 0 or bit 2 set; read data from the board for status bit 1 set.

The Programmed I/O Interface allows the host processor to view the graphics board as a standard hardware USART. The graphics processor uses two contiguous bytes of MULTIBUS I/O or memory space for this interface (see Figure 5.1). Refer to the configuration information supplied with each board to obtain the preset base address of this 2 byte communications area. The board uses the base address as the destination for data writes from the host CPU, and the source for data reads from the graphics processor. The base address location + 1 serves as the destination for communications channel commands from the host CPU, and the source for status information from the graphics processor.

After the INTERACT power up screen is drawn the VM885x is ready to execute INTERACT commands. Poll the status byte to check programmed I/O status. For bit 0 or 2 of the status byte set to 1, one byte of an INTERACT command may be written to the data port (offset 0). For some jumper configurations (see Graphics Processor Manual) more than one byte may be written when transmit ready status is detected. The command port will accept commands (see below) even if bits 0 and 2 of status read zero. For bit 1 of the status byte set to 1, read one byte of an INTERACT reply, in object form, from the data port (offset 0) of the board.

When the host CPU expects a response to its previous INTERACT command, it should poll the status register until bit 1 of the status byte reads 1. When the host detects the data ready condition, it should read one byte from the data register. The host should continue the poll and read loop until the required number of bytes have been collected.

The PI/O communications interrupt (see Graphics Processor Manual for this jumper selectable option) can become active if either a transmit or receive ready condition exists. This interrupt parallels the status bits described above for transmit ready and receive ready. The activity of this interrupt can be controlled by writing a mask to the PI/O command byte. Setting bit 0 to 1 in the command byte enables the transmit interrupt, while clearing bit 0 to 0 masks (disables) the transmit interrupt. Similarly, setting or clearing bit 2 controls the receive ready interrupt. With both bit 0 and bit 2 of the control byte cleared to 0 no MULTIBUS interrupt is generated regardless of jumper position. If interrupt is unmasked for both conditions, the status byte may be read upon interrupt to determine its cause. For some jumper configurations (see Graphics Processor Manual) more than one byte may be written when the transmit ready interrupt is activated. Communications throughput may be increased if the host processor can send a block of data to the graphics processor for each MULTIBUS interrupt.

During normal operation of the PI/O interface, no bytes need be written to the command register (offset 1). However, for disrupted communications or after an incorrect command, a WARM start (see WARM INTERACT command) may be executed by writing 040H to the USART emulator's command register, even in the absence of an XMIT ready status. During the handling of this WARM start interrupt, both receive ready and transmit ready status are cleared. On the VM-885x, the interruption of the command stream with a WARM start may cause unpredictable results, depending on the exact state of processing at the instant of the interrupt, however communication will be reestablished. The WARM start interrupt should not be used during power on reset.

```
                         MULTIBUS
                         I/O Space

       Read                                      Write

 |--------------|                          |----------------|
 |   status     |     Base address +1      |    control     |
 |------------- |                          |----------------|
 |    data      |     Base address         |     data       |
 |--------------|   of communication       |----------------|
                         area

     Status                                   Control
     -----------------                        -----------------
   7 |1|D|0|0|0|A|B|A| 0                    7 |X|C|X|X|X|R|X|T| 0
     -----------------                        -----------------

     Data                                     Data
     -----------------                        -----------------
   7 | |b|i|n|a|r|y| | 0                    7 | |b|i|n|a|r|y| | 0
     -----------------                        -----------------

     A = Ready for data byte                  C = Reset communications
     B = Data byte ready                      X = Don't care
     D = DMA busy                             R = Receive interrupt
                                                  enable
                                              T = Transmit interrupt
                                                  enable

** all bits active high
```

Figure 5.1 : Programmed I/O Registers

49

45 dec

255

At one Time

255
150
167
231

247
182
134

215

5.2 - DMA Interface

The DMA interface allows the VM-885x to fetch INTERACT commands and to output data directly to and from host memory. INTERACT reserves a communication area located at the memory-mapped base address (supplied by the user). This area contains the DMA Control Byte and DMA Block Pointer used in initiating and controlling DMA transfers. Refer to Figure 5.2 for the specification of these bytes.

Host memory contains INTERACT commands and input areas arranged in designated DMA blocks. Each DMA block contains a header listing a status byte, various data bytes, and pointers which direct processing. Refer to Figure 5.3 for specifications on these bytes. The Chain Pointer allows the user to link these blocks together. All write blocks, i.e. ,those containing INTERACT commands, are arranged in the write chain, while all input buffers are arranged in the read chain. The commands sent to the DMA Control Byte in the communications area control the processing of these chains. The DMA address bytes, allocated as the DMA Block Pointer in the dedicated communications area, specifies the location of the lead block of a chain sent to the VM-885x. Since the read and write chains function separately, the VM-8851 can allow both DMA writes and Programmed I/O reads or DMA reads and Programmed I/O writes. The VM-8850A, however, does not allow this option since the DMA and Programmed I/O interfaces require different daughter boards.

5.2.1 - Address Space

Both the VM-8850A and the VM-8851 can generate only 24 bits of address. Thus the DMA block headers and data area must exist in the first 16 Mb of host address space. Additionally, bits 18 through 23 on the VM-8850A are hardware configurable, not software selectable. This restriction limits all DMA headers and data to the single 256 Kb space determined by the hardware configuration. Also note that only 3 bytes (24 bits) are allocated in the dedicated communications area to point to the first block in a chain. (Refer to the DMA Block Pointer in Figure 5.2.)

5.2.2 - Dedicated Communication Area (DCA)

The user provides a memory mapped address for Programmed I/O and DMA interfaces. That base address plus the next consecutive seven bytes compose the dedicated communication area. Refer to Figure 5.2 for an illustration of these bytes. For a description of the first two bytes, refer to the Programmed I/O section of this manual, Section 5.1. This area also contains a DMA Block Pointer and a DMA Control Byte, each described in the following subsections.

5.2.2.1 - Protocol for Writing to DMA DCA

Bytes 4 - 7 of the DCA compose the DMA portion of the dedicated communications area. Before writing a sequence of DMA address and control bytes to the DCA, read the status byte (offset 1) to determine the state of the DMA BUSY bit (see Section 5.1). The DMA BUSY bit will be set after bytes are written to the DMA portion of the DCA, and will be cleared after the DMA control byte is processed. The protocol for writing the the four DMA locations is as follows:

1) Wait for DMA BUSY to go low.
2) Write DMA Block Pointer bytes in order, if needed.
3) Write DMA Control Byte. DMA BUSY will be cleared after the DMA command has been processed and the VM-885x is ready for another DMA command.

5.2.2.2 - DMA Block Pointer

The DMA Block Pointer references the DMA block header of the initial DMA block. (Refer to Figure 5.3 for the organization of the DMA block header.) Bytes located at base address + 4, 5 and 6 must be written sequentially for the pointer to access the proper location.

5.2.2.3 - DMA Control Byte

The DMA Control Byte receives instructions from the host to control DMA operations. Each instruction is identified as a specific binary value. The user writes the value of the requested operation to this byte for execution during the DMA procedure. For a list and description of the available commands, refer to Section 5.2.3, DMA Commands.

```
                   ----------------------------
                 7 |    DMA Control Byte      |
                   ----------------------------
                 6 |                          |  MSB
                   |-                        -|
                 5 |    DMA Block Pointer     |
                   |-                        -|
                 4 |                          |  LSB
                   ----------------------------
                 3 |       Reserved           |
                   ----------------------------
                 2 |       Reserved           |
                   ----------------------------
                 1 |     Command Status       |
                   ----------------------------
Base Address + 0 |   Programmed I/O Data    |
                   ----------------------------
```

Figure 5.2 : VM-885x Dedicated Communication Area

5.2.3 - DMA Commands

The following DMA commands are executed by writing the values shown to the DMA control byte. (Refer to Figure 5.2.)

Command	Value
Read Init	00
Write Init	01
Read Halt	02
Write Halt	03
Read Continue	04
Write Continue	05
Read PI/O	06
Write PI/O	07
Interrupt Acknowledge	08

Writing data to the DMA control byte causes an internal interrupt on the VM-885x. Thus, this byte is processed as soon as possible.

5.2.3.1 - Read Init

This command initializes the first block in the read chain. The address of this first block equals the last address written to the DMA Block Pointer. If nothing has been written to this area, a default address of 0 is used. The initialized block is marked as active. If the ENABLE BLOCK bit is set, then the current Count is set equal to Data Length, and processing begins. This command is only valid if the VM-885x is in Programmed I/O mode or if the read chain has been halted (either by a DMA command or by a Halt Request). If the state changes from Programmed I/O to DMA, then the current INTERACT command is completed before the initiation of a DMA read.

5.2.3.2 - Write Init

Perform the function of INIT (as above) for the write chain. If the write data contains _any_ INTERACT read command, then the read chain should be initialized before the write chain. The DMA write command waits until the completion of the current INTERACT command.

5.2.3.3 - Read Halt

Mark the currently active DMA block in the read chain as inactive. This change stops all processing of this DMA block by the VM-885x until a Read Continue command resets the HALT bit in the status bytes.

5.2.3.4 - Write Halt

Mark the currently active block inactive and halted. This command halts all processing of the DMA write block until a Write Continue, Write Init, or Write Programmed I/O command is issued.

Note that this command is issued asynchronously with processing of INTERACT commands. Thus, the command being fetched from the currently active block may not be complete. If a Write Init or Write Programmed I/O is then issued, the INTERACT command stream will be misinterpreted. This problem can be avoided by issuing a Warm Start command following the Halt.

5.2.3.5 - Read Continue

Continue processing of the currently active block. If the currently active block is marked as COMPLETE and contains no CHAIN REQUEST, then the block is re-initialized. If the block is complete and does contain a chain request, then the Chain Pointer is followed to the next block. If the active block is not halted then no action takes place.

5.2.3.6 - Write Continue

As above for currently active write block.

5.2.3.7 - Programmed I/O Read

The Read Programmed I/O command returns read operations to Programmed I/O mode. Execution of this command is delayed until the currently executed INTERACT command is finished. This command is only valid when the currently active read block is in a HALT state.

5.2.3.8 - Programmed I/O Write

The Write Programmed I/O returns write operations to Programmed I/O mode.

5.2.3.9 - Interrupt Acknowledge

When interrupted, the user may issue an interrupt acknowledge comand to reset the interrupt sent by VM-885x.

5.2.4 - DMA Block Header

The DMA block header is the building block of the DMA interface. This section describes each part of the header and its function. Refer to Figure 5.3 while reading the following information.

5.2.4.1 - Block Command Byte

The Block Command byte directs processing both before processing of the data area begins and after the data area is exhausted. If the CHAIN REQUEST bit is set, then processing continues. The Chain Pointer points to the next block, which then becomes active. If the INTERRUPT REQUEST bit is set, the VM-885x generates an interrupt when the block data area is exhausted. Finally the HALT REQUEST bit forces the HALT bit to be set in the Status byte. A chain request is not honored until this HALT bit has been cleared by a continue command.

The BLOCK ENABLE bit ensures that processing of a block does not commence until the user has indicated a ready state. This bit is checked on initialization of a block, accomplished using anInit command or through a chaining operation. While this bit equals zero, no processing of the block occurs. Processing beins when the bit equals one. Since the VM-885x polls the ENABLE BLOCK bit, a block in an active but disabled state implies numerous MULTIBUS accesses by the VM-885x. For an example on the use of this bit, refer to Section 5.2.5.

5.2.4.2 - Status Byte

The Status byte indicates the current status of its respective DMA Block. The ACTIVE bit, if set, indicates that the block is currently active and is being accessed by the VM-885x. The HALT bit indicates that either the processing of this block has been halted by a DMA Halt command (Section 5.2.3) or this block has completed processing and no completion request bits were set. The CHAINED bit indicates that the block has completed processing and has honored a chain request. The COMPLETE bit indicates that processing of the block has been completed. Note that the host system should treat the status byte as read only.

5.2.4.3 - Data Area Pointer

The Data Area Pointer is a 32-bit pointer to the data area associated with the block. If the block is in the write chain, this data contains INTERACT commands. If the block is in the read chain, then this data area will be written to by the VM-885x in response to "read" INTERACT commands.

```
       ------------------
    13 |                |  MSB
       |-              -|
    12 |     Chain      |
       |-              -|
    11 |    Pointer     |
       |-              -|
    10 |                |  LSB
       ------------------
     9 |    Current     |  MSB
       |-              -|
     8 |     Count      |  LSB
       ------------------
     7 |  Data Length   |  MSB
       |-              -|
     6 |                |  LSB
       ------------------
     5 |                |  MSB
       |-              -|        ----------------------- ACTIVE
     4 |   Data Area    |        |
       |-              -|        |              --------- HALTED
     3 |    Pointer     |        |              |  ------ COMPLETE
       |-              -|        |              |  |  --- CHAINED
     2 |                |  LSB   |              |  |  |
       ------------------       -------------------------
     1 |    Status      |  ---> |  |           |  |  |  |
       ------------------     -------------------------
Byte 0 |    Command     |  ---> |  |           |  |  |  |
       ------------------       -------------------------
                                 |              |  |  |
                                 |              |  |  --- CHAIN REQ
                                 |              |  ------ INTERRUPT REQ
                                 |              --------- HALT REQ
                                 |
                                 ----------------------- ENABLE BLOCK
```

Figure 5.3 : DMA Block Header

5.2.4.4 - Data Length

Data Length is a 16-byte area which indicates the number of bytes in the data area. Data Length may not exceed 65280.

5.2.4.5 - Current Count

Current Count is a 16-bit area used by the VM-885x to monitor progress of the processing of the block. The VM-885x initializes this area with Data Length when processing of a given block starts, then decrements to 0. The host should treat the Current Count as read only.

5.2.4.6 - Chain Pointer

The Chain Pointer is a 32-bit address pointing to the next DMA block header in the chain.

5.2.5 - DMA Examples

Refer to the DMA State Diagrams, Figures 5.4 and 5.5, for further illustration of these examples.

5.2.5.1 - Single Write Block

The following is a simple example of the DMA interface.

1) All INTERACT commands to be executed are assembled sequentially into some known data area and the length computed.

2) Create a DMA block header and place the address of the data area previously established in the Data Area Pointer location.

3) Initialize the Data Length location with the length of the data area.

4) In this example no chaining or interrupt is needed. Clear the completion request byte to 0. This request means that when processing is finished, the block will be marked as complete and the process halted.

5) Clear Status byte.

6) Write the address of the block header to the DMA Block Pointer.

7) Write a write init to the DMA Control Byte.

8) Wait for the block to be completed by polling the completion bit. Note that the block can be re-executed by issuing a Write Continue command.

5.2.5.2 - Cyclic Write Blocks

In this example, three blocks link together in a static cycle as shown:

```
          -----
     ---->| A |-----
     |    -----    |
     |             v
   -----         -----
   | C | <-------| B |
   -----         -----
```

In this situation, the host could update one block while the VM-885x accesses another block. To achieve this state, the user must complete certain steps. First, the host must create and initialize the block headers and link them together, chaining A to B, B to C, and C to A, as shown above. All blocks should be labeled as not enabled, i.e. the ENABLE BLOCK bit should equal zero for each block. For this example, let block header C HALT and generate an interrupt upon completion. Processing begins when the host updates the data area associated with block A. When the update operation is complete, the host will update the data length field in header A and mark that block as enabled.

The host initiates DMA by writing the address of block header A to the DMA Block Pointer in the Dedicated Communication Area. The host must also send a Write Init command to the DMA Command Byte in the same area. The host can now begin updating block B data area. On completion of this operation, the host marks block B as enabled, updates the Data Length field and proceeds to block C.

After completing the data update and enabling Block C, the host may resume other processing. When the VM-885x finishes processing Block C, an interrupt will be issued and the write chain process will be halted. The host, after acknowledging the interrupt with an Acknowledge command, can then disable all three blocks. When block A is updated, a Write Continue command will resume write chain processing, and the cycle repeats.

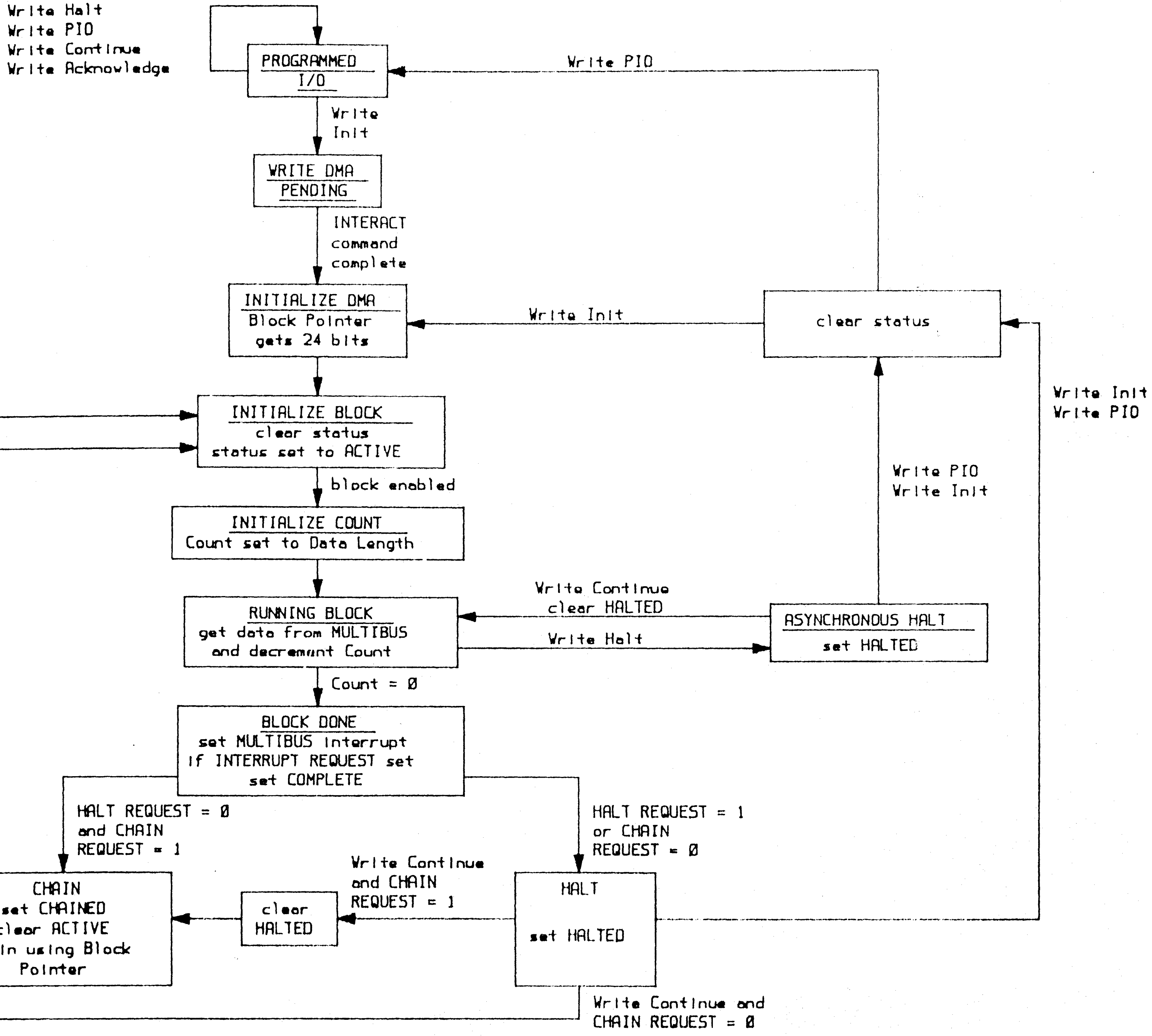


Figure 5.4 : DMA Write State Diagram

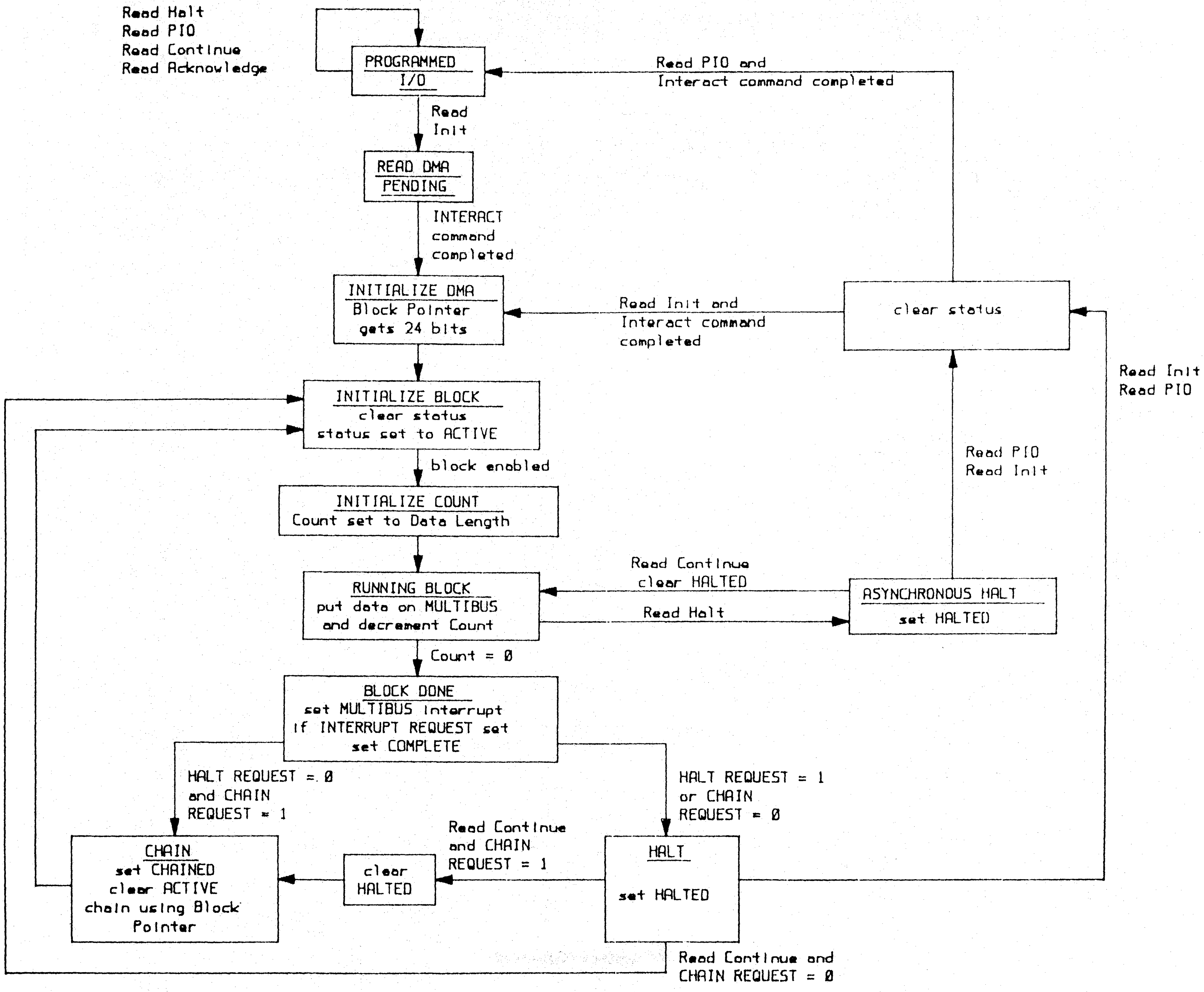


Figure 5.5 : DMA Read State Diagram

5.3 - INTERACT Interpreter

The ASSIGN command can invoke the interpreter using the following format:

```
        ASSIGN  chan 2
```

Invoking the interpreter will result in the response:

```
        I>
```

Certain Interpreter commands allow the user to define how the Interpreter should accept INTERACT commands. All interpreter commands start with "%". Following is a list of some of the valid interpreter commands:

Command	Mode	Command	Mode
%SRC	(Source)	%OBJ	(Object)
%DEC	(Decimal)	%HEX	(Hex)
%ECHO	(Echo)	%QUIET	(Quiet)
%WSIGN	(Words signed)	%WPOS	(Words positive)
%BPOS	(Bytes positive)	%BSIGN	(Bytes signed)
%LZHEX	(Lead zeros for hex)	%NZHEX	(No lead zeros for hex)
%NHSUP	(NOT H suppress)	%HSUP	(H suppress)

The above table lists the commands in a one-to-one correspondence. The interpreter defaults to all the commands in the left-hand column. The right-hand column lists the optional modes for each command on the left. For example, the interpreter can operate in either source mode or object mode.

The term "command line" will refer to a user-supplied string of ASCII characters followed by a carriage return. A command line cannot exceed 255 characters and the resulting object code stream cannot exceed 255 bytes for any one command.

The interpreter will accept only spaces, commas or angle brackets as delimiters between parameters.

The interpreter ignores commands in lines after a delimiter followed by a semicolon (;).

5.3.1 - Modes of Operation

5.3.1.1 - SOURCE Mode

The interpreter defaults to SOURCE mode. To specify SOURCE mode, use the %SRC interpreter command. If the interpreter is in

SOURCE mode and prompts are not being suppressed (refer to Section 5.3.1.6, QUIET mode), then the user will receive either an "I>" or an "M>" as a prompt. The prompt signifies the interpreter as ready to accept INTERACT mnemonics as commands. For example, to load CREG 20 with the values 2695, 35, the user would enter:

```
        CLOAD 20 2695 35
```

In SOURCE mode, the interpreter will try to match the mnemonic entered by the user to a mnemonic listed in the table of valid commands. If the user were to type CLOA 20 1023 35, the interpreter would search its table for mnemonics beginning with "CLOA". If CLOAD is the only command which begins "CLOA", then the interpreter will assume CLOA to mean CLOAD. If the interpreter finds more than one mnemonic in its table that matches the mnemonic typed in, it will return an error message to the user. For example, "MOV" is not a valid mnemonic because both MOVABS and MOVREL begin with "MOV".

In SOURCE mode, the interpreter determines the number of parameters needed for any given command. The command line is scanned for the number of parameters designated in the command specification. For any parameters missing on the line, the interpreter will supply additionally needed zeroes. Each command or series of commands and associated parameters must be completely contained within a single command line. A carriage return terminates each command. The interpreter takes no action on a command until a carriage return has been typed.

The "I>" prompt indicates the interpreter is ready to process another command. During a macro definition, the prompt changes to "M>". The "M>" prompt indents from the left margin on the screen and continues until execution of a MACEND command.

If a readback command is executed in SOURCE mode,then both word and byte readback parameters are converted to an 8-character ASCII stream. An example of a terminal display after a readback command follows:

```
        I>VLOAD 10,15
        I>CLOAD 20,2695,35
        I>READCR 20
            2695    1743
        I>READVR 10
              15
        I>
```

5.3.1.2 - OBJECT Mode

Entering the %OBJ Interpreter command puts the interpreter into OBJECT mode. In this mode, if prompts are not suppressed (refer to Section 5.3.1.6, QUIET mode), then the user will receive a "#>" as a prompt. In OBJECT mode, the interpreter accepts only numeric parameters (i.e., no mnemonics) and each parameter is interpreted as a byte. The requirement that all numbers begin with a digit is relaxed in object mode, where all input is assumed to be numbers. This aspect implies that word parameters must be entered as two byte parameters. Thus, the CLOAD example above could be entered in OBJECT mode as (with Hex mode on):

```
A0, 14, A, 87, 0, 23
```

Note that the high bytes of words are entered first.

In this mode, the interpreter does not check opcodes for validity or calculate the parameter string length required for each command. Each command and associated parameters may extend over more than a single command line. Thus a command longer that 255 bytes which could not be entered in source mode may be spread over multiple command lines in object mode. The restriction on command line size, however, still holds true. Also, the interpreter executes none of the commands on a command line until detecting a carriage return.

If a readback command is executed in OBJECT mode, then the interpreter treats readback parameters as byte parameters, i.e., word parameters will be read back as two bytes. An example of a terminal display after a readback command follows:

```
I>VLO 8 3
I>CLO 20 15 5
I>%OBJ
#>98H 20T
        0      15       0      5
#>99H 8
        3
#>%SRC
I>
```

5.3.1.3 - DECIMAL Mode

The Interpreter defaults to DECIMAL mode. The user can select DECIMAL mode by using the %DEC interpreter command. In DECIMAL mode, the Interpreter assumes all numbers to be decimal numbers (base 10) unless they are followed by a trailing "H". Numbers may also be followed by a trailing "T" to specify decimal.

When doing readbacks in DECIMAL mode, leading zeros are blank filled with the exception of the rightmost digit.

5.3.1.4 - HEX Mode

To change to HEXADECIMAL mode, use the %HEX Interpreter command. In HEXADECIMAL mode, the Interpreter assumes all numbers to be hexadecimal (base sixteen) numbers unless they are followed by a trailing "T" (for base ten). A trailing "H" specifies hexadecimal.

When doing readbacks in HEX mode, the Interpreter assumes all parameters are unsigned.

5.3.1.5 - ECHO Mode

The Interpreter defaults to ECHO mode. The user can invoke ECHO mode by using the %ECHO interpreter command. In ECHO mode, the Interpreter echoes all commands back to the channel where it received them and includes the appropriate prompts.

Readback data in ECHO mode has a carriage return and a line feed before for the parameter data.

5.3.1.6 - QUIET Mode

The user can invoke the QUIET mode by using the %QUIET Interpreter command. In QUIET mode, the Interpreter does not echo entered commands. All prompts, including line feeds and carriage returns, are suppressed.

Error messages are returned for Interpreter errors. Readbacks are also returned.

5.3.1.7 - WORDS SIGNED Mode

The Interpreter defaults to WORDS SIGNED mode. Invoke WORDS SIGNED mode by using the %WSIGN Interpreter command. In WORDS SIGNED mode, all word parameters read back will be interpreted as signed integers.

5.3.1.8 - WORDS POSITIVE Mode

Change to WORDS POSITIVE mode by using the %WPOS interpreter command. If the Interpreter is in WORDS POSITIVE mode, the interpreter assumes all word parameters read back to be unsigned (positive) integers.

5.3.1.9 - BYTES POSITIVE Mode

The interpreter defaults to BYTES POSITIVE mode. Invoke BYTES POSITIVE mode by using the %BPOS Interpreter command. In BYTES POSITIVE mode, all byte parameters read back will be interpreted as unsigned (positive) integers.

5.3.1.10 - BYTES SIGNED Mode

The user can attain BYTES SIGNED mode by using the %BSIGN interpreter command. In BYTES SIGNED mode, all byte parameters read back will be interpreted as signed integers.

5.3.1.11 - LEAD ZEROS FOR HEX Mode

The Interpreter defaults to LEAD ZEROS FOR HEX Mode. Invoke LEAD ZEROS FOR HEX Mode by using the %LZHEX Interpreter command. This mode allows the interpreter to distinguish mnemonics from parameters. It requires that hex numbers always start with a digit from 0 to 9. The hex number FFH would thus be entered as 0FFH. Hex readbacks in LEAD ZEROS FOR HEX mode will always have a leading zero.

5.3.1.12 - NO LEAD ZEROS FOR HEX Mode

Change to NO LEAD ZEROS FOR HEX Mode by using the %NZHEX Interpreter command. This mode relaxes the restriction that hex numbers must start with a digit from 0 to 9. Operating in this mode can result in mnemonics being interpreted as parameters. For example, if the interpreter were in SOURCE Mode, HEX Mode, and NO LEAD ZEROS FOR HEX Mode and the user typed in "MOVABS CADD 5", the user may want that to mean "MOVABS 0 0 CADD 5 0" but it would be interpreted as "MOVABS 0CADDH 0".

5.3.1.13 - NOT H SUPPRESS Mode

The Interpreter defaults to NOT H SUPPRESS mode. Invoke NOT H SUPPRESS Mode by using the %NHSUP Interpreter command. In NOT H SUPPRESS Mode, all readbacks done in HEX Mode will have a trailing H.

5.3.1.14 - H SUPPRESS Mode

Change to H SUPPRESS Mode by using the %HSUP Interpreter command. In H SUPPRESS Mode, all readback done in HEX Mode will not have a trailing H.

5.3.2 - Editing

The interpreter accepts INTERACT commands in either upper or lower case letters.

The <DEL> key (7FH) deletes the character preceding the cursor and moves the cursor back one position.

The backspace key (08H) will move the cursor back one position but will not delete any characters.

A <CTRL> X deletes the entire line.

5.3.3 - Interrupt

A <CTRL> R sends a warm start to the graphics processor.

5.4 - AM94/1530 Dual Channel SBX Module

The optional dual channel SBX module offers two additional channels for the VM885x graphics processor. These logical channels, designated channel 1 and channel 2, support the same software functions as the standard MULTIBUS interface, channel 0. The channels function independently, although high level drivers, such as the INTERACT Interpreter, may not be loaded on more than one channel simultaneously. The channel are scanned sequentially, with one complete INTERACT command executed on the current channel (if available) before the next channel is scanned. Since MACRUN and MACDEF are each INTERACT commands, a complete macro must be executed or defined on the current channel before the next channel is scanned. The input/output handlers of each channel operate independently of the currently scanned channel, so that communications is not functionally affected by graphics tasks.

5.4.1 - Cable Connection to the RS-232C SBX Module

The AM94/1530 SBX module offers two (male) 26 pin edge connectors labeled P2 and P3, which respectively correspond to INTERACT channels 1 and 2. (Refer to the ASSIGN command in the INTERACT software manual.) The MULTIBUSTM interface corresponds to INTERACT channel 0.

The SBX Module is a Data Set device which will interface to a standard Data Terminal device according to the following specifications:

Baud Rate	9600 Baud
Word Length	8 bits
Parity	none
Stop Bits	2
Protocol	Xon/Xoff or DTR/DSR

The protocol listed above depends on the driver assigned using the ASSIGN command. If the driver uses the ASCII communication format, the default protocol is Xon/Xoff; for binary communication format the default protocol is Data Terminal Ready/Data Set Ready (DTR/DCD). Refer to the Graphics Processor manual for specification of the particular driver.

For ASCII communications, only three lines are required over an RS-232C cable: TxD, RxD, and signal ground. For binary communication formats, two additional lines are needed: DATA SET READY (DSR) and DATA TERMINAL READY (DTR/DCD). If DTR is not supplied by the device, the SBX can be used for ASCII communications only by connecting DTR to DSR on the header of the SBX module. CLEAR TO SEND (CTS) and REQUEST TO SEND (RTS) should be connected on the header if CTS is not supplied by the data terminal device. The SBX will always assert CTS and will ignore RTS. To connect a Data Terminal device, these seven lines may be brought straight through on the SBX header. To connect a Data Set device, each element in the pairs of signals must be crossed; TxD/RxD, DTR/DSR, and CTS/RTS. (Refer to Figure 5.6.) The VM-885x is factory configured for a seven line RS-232C cable to connect to data terminal devices.

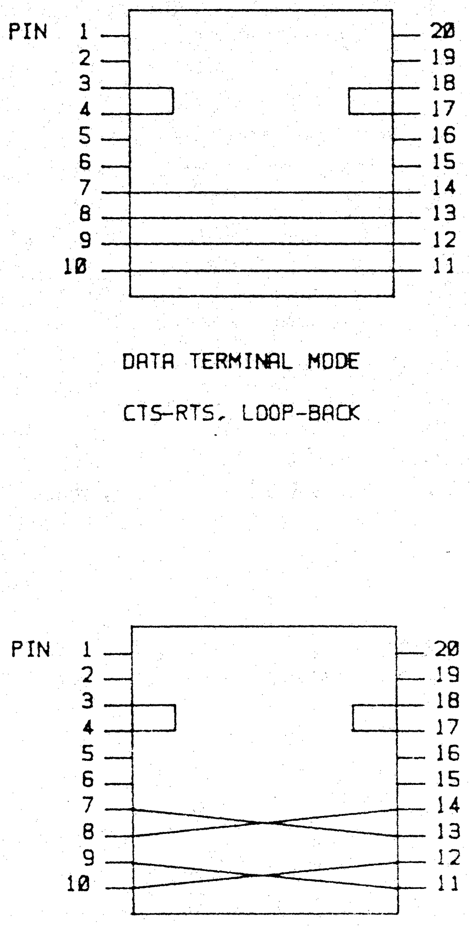
By default, the INTERACT interpreter is ASSIGNed to channel 1 and the transparent mode (Interact binary) is ASSIGNed to channel 0 at power-on, reset, and COLD starts.

184, 1, 3
↑ channel [illegible] try channel 2

160, 11, 0,0,0,0

160, 12, 0,100,0,100

[illegible] painted rectangle



DATA TERMINAL MODE

CTS-RTS, LOOP-BACK

DATA SET MODE

Figure 5.6 : SBX Header Configuration

5.4.2 - Digitizing Tablet

A digitizing tablet can be assigned to a channel with the ASSIGN command. An example would be:

```
ASSIGN 2 5
```

The above example assigns the digitizing tablet to channel 2. The contents of CREG 11 and CREG 12, at the time of the ASSIGN command, define the rectangle covered by the digitizing tablet. Load CREG 11 with the coordinates of the lower left-hand corner of the defined area and CREG 12 with the coordinates of the upper right-hand corner of the coordinate space. The coordinate space actually covered by the digitizing tablet may be slightly larger than the coordinate space requested. The magnitude of this discrepancy will depend on the digitizing tablet used and the values chosen for CREG 11 and CREG 12.

5.4.3 - Printer

A printer can be assigned to a channel with the ASSIGN command. An example would be:

```
ASSIGN 2 3
```

The above example assigns the printer to channel 2. The contents of CREG 11 and CREG 12, at the time of the ASSIGN command, define rectangle to be printed. Load CREG 11 with the coordinates of the lower left-hand corner of the designated area and CREG 12 with the coordinates of the upper right-hand corner of the rectangle to be printed.

5.4.4 - Light Pen

The optional light pen can be enabled by:

```
ASSIGN 5 15
```

Once enabled, placing the light pen on the display sceen causes the virtual coordinate under the pen to be placed in CREG 2. If the light pen button is pressed (this may be the tip of the pen), the INTERACT command

```
BUTCON 2
```

is run, which allows macros to be accessed by the light pen.

Appendix A
Related Documents

Document Number	Description
VM 2001 1101-02	INTERACT^{TM} Language Reference Card
VM 1013 0001-01	VM-8850A Graphics Processor Manual
VM 1018 0001-00	VM-8851 Graphics Processor Manual

Appendix B
Cold Start Default Values

A COLD start INTERACT command, a power-on, or a reset initializes INTERACT software. During initialization, the board issues the following INTERACT commands:

```
CONFIG    0,128,256

VLOAD     n,0                 ; where n ranges from 0 through 15
VLOAD     6,255
VLOAD     3,255
VLOAD     4,255

CLOAD     n,0,0               ; where n ranges from 0 through 63

LUTRST                        ; Reset all LUT entries

ASSIGN    0,1                 ; ASSIGN commands are set to
ASSIGN    1,2                   defaults for any board with
ASSIGN    2,0                   an RS-232C SBX connector

BUFFER    0,0
FIRSTP    0
BLINKR    30
BLANK     0
PIXFUN    0
PRMFIL    0
SURFAC    0
BUTTBL    n,n                 ; where n ranges from 0 to 31

WINDOW    -32768,-32768,32767,32767

CLIPDF    n,-32768,-32768,32767,32767
                              ; where n ranges from 1 to 4

BUTREC    n,32767,32767,-32768,-32768
                              ; where n ranges from 0 to 31

DSPSIZ    (consult hardware manual)
IMGSIZ    (consult hardware manual)
TEXTB     0
TEXTC     0,0
XHAIR     0,0
XHAIR     1,0
VECPAT    FFFF
AREAPT    FFFF,FFFF,....,FFFF
VREG 14   i                   ; i = 2 for 8850A, i = 3 for 8851
VREG 15   j                   ; j = 3 for INTERACT Version 4.0
```

Appendix Cl
Command Summary by Opcode

The following listing provides a summary of the INTERACT commands in ascending order of opcode. For each command, the hex opcode, mnemonic, and parameters are given.

Opcode	Mnemonic	Parameters
00	NULL	
01	MOVABS	x,y
02	MOVREL	dx,dy
03	MOV3R	dx,dy
04	MOV2R	dxdy
05	MOVI	creg
06	VALUE	color
07	FLOOD	
0B	MACRUN	macnum
0C	MACEND	
0E	CIRCLE	rad
0F	CIRCXY	x,y
10	CIRCI	creg
11	ARC	rad,al,a2
12	POLYGN	npoly,nvertl,xl,yl,...
13	AREAl	
14	AREA2	vreg
18	LUTR	index,entry
19	LUTG	index,entry
lA	LUTB	index,entry
lC	LUT8	index,rentry,gentry,bentry
lF	PRMFIL	flag
20	BLINKE	lut,index,entryl,entry2
21	BLINKD	lut,index
22	BLINKR	frames
23	BLINKC	
24	CONFIG	fifo,macbuf,txtfnt
26	TEXTDN	char,x,y,fntlst
28	PIXELS	x,y,color,...
2D	AREAPT	pattern
2E	VECPAT	mask
2F	FIRSTP	flag
31	BLANK	flag
34	ZOOM	fact,bdst,bsrc
3A	WINDOW	xl,yl,x2,y2
3B	PIXFUN	mode
3D	WAIT	frames
44	DSPSIZ	x,y,freq,screen
45	IMGSIZ	x,y,depth
81	DRWABS	x,y

Opcode	Mnemonic	Parameters
82	DRWREL	dx,dy
83	DRW3R	dx,dy
84	DRW2R	dxdy
85	DRWI	creg
88	POINT	
89	RECREL	dx,dy
8B	MACDEF	macnum
8C	MACERA	macnum
8E	RECTAN	x,y
8F	RECTI	creg
90	TEXT1	string
91	TEXT2	string
92	TEXTC	size,angle
93	TEXT0	string
94	TEXTB	flag
95	READP	
98	READCR	creg
99	READVR	vreg
9A	READBU	flag,cflag
9C	XHAIR	num,flag
9F	FILMSK	mask
A0	CLOAD	creg,x,y
A1	CMOVE	cdst,csrc
A2	CADD	csum,creg
A3	CSUB	cdif,creg
A4	VLOAD	vreg,color
A5	VMOVE	vdst,vsrc
A6	VADD	vsum,vreg
A7	VSUB	vdif,vreg
AA	BUTTBL	index,macnum
AB	BUTTON	index
AF	RDPIXR	vreg
B8	ASSIGN	chan,dev
B9	BUTREC	butnum,x1,y1,x2,y2
BA	BUTCON	creg
BB	MACREP	macnum,count
E0	BUFFER	update,display
E5	BLKMOV	x1,y1,x2,y2
E6	POLYRL	npoly,nvert1,dx1,dy1,...
EA	CLIP	num
EB	CLIPDF	num,x1,y1,x2,y2
F0	PIXDMP	depth,dx,dy
F1	PIXLOD	depth,dx,dy,bitstream
F5	SURFAC	count,p1,p2,...
F6	LUTRST	
F7	LUTMSK	mask
FD	COLD	
FE	WARM	

Appendix C2
Command Summary by Mnemonic

The following listing provides a summary of INTERACT commands in alphabetical order of the mnemonic.

Opcode	Command	Parameters
11	ARC	rad,al,a2
2D	AREAPT	pattern
13	AREA1	
14	AREA2	vreg
B8	ASSIGN	chan,dev
31	BLANK	flag
23	BLINKC	
21	BLINKD	lut,index
20	BLINKE	lut,index,entry1,entry2
22	BLINKR	frames
E5	BLKMOV	x1,y1,x2,y2
E0	BUFFER	update,display
BA	BUTCON	creg
B9	BUTREC	butnum,x1,y1,x2,y2
AA	BUTTBL	index,macnum
AB	BUTTON	index
A2	CADD	csum,creg
10	CIRCI	creg
0E	CIRCLE	rad
0F	CIRCXY	x,y
EA	CLIP	num
EB	CLIPDF	num,x1,y1,x2,y2
A0	CLOAD	creg,x,y
A1	CMOVE	cdst,csrc
FD	COLD	
24	CONFIG	fifo,macbuf,txtfnt
A3	CSUB	cdif,creg
81	DRWABS	x,y
85	DRWI	creg
82	DRWREL	dx,dy
84	DRW2R	dxdy
83	DRW3R	dx,dy
44	DSPSIZ	x,y,freq,screen
9F	FILMSK	mask
2F	FIRSTP	flag
07	FLOOD	
45	IMGSIZ	x,y,depth
1A	LUTB	index,entry
19	LUTG	index,entry
F7	LUTMSK	mask
18	LUTR	index,entry
F6	LUTRST	

Opcode	Command	Parameters
1C	LUT8	index,rentry,gentry,bentry
8B	MACDEF	macnum
0C	MACEND	
8C	MACERA	macnum
BB	MACREP	macnum,count
0B	MACRUN	macnum
01	MOVABS	x,y
05	MOVI	creg
02	MOVREL	dx,dy
04	MOV2R	dxdy
03	MOV3R	dx,dy
00	NULL	
F0	PIXDMP	depth,dx,dy
28	PIXELS	x,y,color,...
3B	PIXFUN	mode
F1	PIXLOD	depth,dx,dy,bitstream
88	POINT	
12	POLYGN	npoly,nvert,x1,y1,...
E6	POLYRL	npoly,nvert1,dx1,dy1,...
1F	PRMFIL	flag
AF	RDPIXR	vreg
9A	READBU	flag,cflag
98	READCR	creg
95	READP	
99	READVR	vreg
89	RECREL	dx,dy
8E	RECTAN	x,y
8F	RECTI	creg
F5	SURFAC	count,p1,p2,...
94	TEXTB	flag
92	TEXTC	size,angle
26	TEXTDN	char,x,y,fntlst
93	TEXT0	string
90	TEXT1	string
91	TEXT2	string
A6	VADD	vsum,vreg
06	VALUE	color
2E	VECPAT	mask
A4	VLOAD	vreg,color
A5	VMOVE	vdst,vsrc
A7	VSUB	vdif,vreg
3D	WAIT	frames
FE	WARM	
3A	WINDOW	x1,y1,x2,y2
9C	XHAIR	num,flag
34	ZOOM	fact,bdst,bsrc

Appendix D
Look-up Table Default Values

INDEX	VALUE	COLOR
	RGB	
0	0000H	BLACK
1	0FFFH	WHITE
2	0F00H	RED
3	00F0H	GREEN
4	000FH	BLUE
5	00FFH	CYAN
6	0F0FH	MAGENTA
7	0FF0H	YELLOW
8	0F80H	RED-YELLOW
9	08F0H	YELLOW-GREEN
10	00F8H	GREEN-CYAN
11	0080H	CYAN-BLUE
12	080FH	BLUE-MAGENTA
13	0F08H	MAGENTA-RED
14	0555H	DARK GRAY
15	0AAAH	LIGHT GRAY

INDEX	VALUE
	RGB
16	0FF5H
17	0AF6H
18	06F8H
19	03FAH
20	04BBH
21	069BH
22	0A7BH
23	0F5DH
24	08D6H
25	05D7H
26	03D8H
27	038DH
28	036FH
29	065FH
30	0A3FH
31	0D3FH
32	0BF3H
33	09F4H
34	06F5H
35	03F6H
36	03B8H
37	039AH
38	037BH
39	046DH
40	055DH
41	083DH
42	0B3BH
43	0D59H
44	0B68H
45	0B76H
46	0D85H
47	0D94H
48	0FD3H
49	0BF3H
50	08F3H
51	06F4H
52	03D6H
53	03B7H
54	0399H
55	037AH
56	035DH
57	054DH
58	073DH
59	0B3AH
60	0F39H
61	0F48H
62	0F56H
63	0F65H
64	0FB3H
65	0AF0H
66	07F3H
67	05F3H
68	03F5H
69	03B6H
70	03A6H
71	0379H
72	036AH
73	035BH
74	034DH
75	033FH
76	053FH
77	072DH
78	0A0BH
79	0D0AH
80	0DA2H
81	09D2H
82	07D3H
83	04D4H
84	03D5H
85	03A5H
86	0396H
87	0377H
88	0369H
89	035AH
90	034BH
91	053BH
92	073AH
93	0A38H
94	0D37H
95	0B46H
96	0BA0H
97	07B2H
98	06D2H
99	05D3H
100	03D4H
101	03B4H
102	03A5H
103	0386H
104	0367H
105	0367H
106	0359H
107	0449H
108	0647H
109	0946H
110	0B45H
111	0F45H
112	0A90H
113	07A2H
114	05B2H
115	04D2H
116	03D3H
117	03B3H
118	03A4H
119	0395H
120	0375H
121	0366H
122	0357H
123	0457H
124	0656H
125	0755H
126	0955H
127	0B54H
128	0980H
129	0682H
130	0593H
131	0493H
132	0394H
133	0285H
134	0265H
135	0356H
136	0356H
137	0238H
138	0339H
139	0537H
140	0636H
141	0736H
142	0A35H
143	0D35H
144	0762H
145	0682H
146	0573H
147	0383H
148	0274H
149	0365H
150	0255H

INDEX	VALUE
	RGB
151	0355H
152	0346H
153	0436H
154	0437H
155	0536H
156	0636H
157	0735H
158	0935H
159	0B34H
160	0662H
161	0562H
162	0580H
163	0382H
164	0382H
165	0373H
166	0363H
167	0264H
168	0354H
169	0355H
170	0245H
171	0236H
172	0237H
173	0308H
174	0506H
175	0605H
176	0A50H
177	0950H
178	0752H
179	0553H
180	0454H
181	0445H
182	0435H
183	0535H
184	0635H
185	0734H
186	0A24H
187	0B04H
188	0F03H
189	0F33H
190	0D32H
191	0B42H
192	0840H
193	0650H
194	0453H
195	0453H
196	0344H
197	0335H
198	0435H
199	0525H
200	0624H
201	0823H
202	0B03H
203	0D03H
204	0F02H
205	0F33H
206	0B30H
207	0A40H
208	0630H
209	0550H
210	0362H
211	0253H
212	0253H
213	0244H
214	0335H
215	0325H
216	0405H
217	0604H
218	0803H
219	0A03H
220	0B03H
221	0A22H
222	0832H
223	0830H
224	0530H
225	0350H
226	0250H
227	0242H
228	0233H
229	0234H
230	0235H
231	0205H
232	0304H
233	0303H
234	0503H
235	0503H
236	0503H
237	0622H
238	0630H
239	0330H
240	0430H
241	0330H
242	0330H
243	0232H
244	0233H
245	0033H
246	0223H
247	0303H
248	0303H
249	0303H
250	0303H
251	0403H
252	0502H
253	0630H
254	0430H
255	0330H

Appendix E
Elements of INTERACT State

AREA FILL MASK
-VREG3- Pixel mask for random area fills.

AREA PATTERN
Pattern used to implement texturing of filled figures. Set with **AREAPT.**

BIT PLANE MASK
-VREG6- Color mask used by all graphics primitives.

BLANK FLAG
Screen is blank when enabled. Set with **BLANK.**

BLINK RATE
Rate at which blinking occurs. Set with **BLINKR.**

BLINK STATUS
Three bits for each (red, green, and blue) LUT entries. Set with **BLINKE.**

BLINK TABLES
Two tables which provide color information for blinking LUTs. Loaded with **BLINKE.**

BUTTON FIFO
EVENT QUEUE
Eight event FIFO, where each event consists of an executed button number, **CREG2,** and **CREG5** at the time of button execution.

BUTTON TABLE
Table which associates button numbers with macro numbers. Set with **BUTTBL.**

CLIPPING BOUNDARY
-CREG9,CREG10- Current clipping window virtual coordinates.

CLIP WINDOW DEFINITIONS
Four definitions, each consisting of a pairof coordinates, which define a rectangular clipping window. Set by **CLIPDF.**

CONDITIONAL BUTTON
EXECUTION TABLE
One entry for each of the 32 buttons. Each entry is a pair of virtual coordinates defining a rectangular area that will cause that button to be executed if the **CREG** coordinates given in a **BUTCON** is contained within that rectangular area. Set by **BUTREC.**

COORDINATE ORIGIN -CREG3- Coordinate of the center of image memory in virtual space.

CROSSHAIR 0 COLOR -VREG1- Pixel value for crosshair 0.

CROSSHAIR 0 LOCATION -CREG5- Virtual coordinate of crosshair 0.

CROSSHAIR 1 COLOR -VREG2- Pixel value for crosshair 1.

CROSSHAIR 1 LOCATION -CREG6- Virtual coordinate of crosshair 1.

CURRENT COLOR -VREG0- Pixel value used by all graphics primitives.

CURRENT POINT -CREG0- Starting, or center, point for graphics primitives.

DEVICE BOUNDARY -CREG11,CREG12- Coordinates of the rectangle used by the printer driver and the digitizer driver.

DISPLAY BUFFER Buffer to be displayed on the video screen. Set by **BUFFER.**

DISPLAY SIZE Format of display; for example, 640 x 480 pixels. Set by **DSPSIZ**.

FIRST PIXEL FLAG Flag to inhibit drawing of first pixel of vectors. Set by **FIRSTP.**

IMAGE SIZE Organization of physical memory. Given in x, y, and depth dimensions. Set by **IMGSIZ.** Image size determines the number of buffers available.

LOCATOR ADJUSTMENT -CREG8- Coordinate calibration factor for screen dependent locator hardware.

LOCATOR POSITION -CREG2- Virtual coordinate returned by locator device

LOOKUP TABLES Color lookup tables used to convert value codes into actual R, G, and B color intensities for display. Set with **LUTR, LUTG, LUTB,** and **LUT8**.

LUT MASK -VREG4- Mask applied to pixel values before indexing into LUTs.

MACRO DEFINITION TABLE
Table which contains INTERACT macros, which are defined with **MACDEF** and erased with **MACERA.**

PIXEL FUNCTION
Drawing mode. Insert, complement, or XOR functions currently allowed.

PRIMITIVE FILL FLAG
When set, closed primitives are drawn filled. When cleared, primitives draw in outline. Set with **PRMFIL.**

RAM CONFIGURATION
Allocation of scratch pad RAM among FIFOs, **TEXT** font definition table and macro definition table. Set with CONFIG following power up or **COLD.**

SCREEN ORIGIN
-CREG4- Virtual coordinate of the pixel at the center of the display screen.

SURFACE PRIORITIES
Priorities given to certain bit planes to provide the appearence of one surface covering another. Set by **SURFAC.**

TEXT BACKGROUND COLOR
-VREG5- Color for background of text.

TEXT BACKGROUND FLAG
When set, causes text command to draw background underneath text.

TEXT ENDPOINT
-CREG7- End of string virtual coordinates for TEXT PRIMITIVES.

TEXT FONT DEFINITION TABLE
Table which contains text fonts used by **TEXT2.** These fonts are specified using **TEXTDN.**

TEXT SIZE
Size of characters drawn with **TEXT0.** Set by **TEXTC.**

UPDATE BUFFER
Buffer affected by draw commands. Set by **BUFFER.**

VECTOR PATTERN
Pattern used to implement dotted or dashed outline figures. Set by **VECPAT.**

XHAIR ENABLE FLAGS
Flags set to enable display of the two possible crosshairs. Set by **XHAIR.**

VMI

INTERACT Graphics Language Manual
Version 4.0
VM 2001 001-07

READER'S COMMENTS

NOTE: The Technical Publications Department attempts to provide documents that meet the needs of all VMI product users.

Please restrict your comments to the contents of this document. VMI will use comments submitted on this form at the company's discretion.

1. Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

2. Did you find errors in this manual? If so, specify the error and the page number.

3. Please indicate the type of reader that you most nearly represent.

□ Assembly language programmer
□ Higher-level language programmer
□ Hardware engineer
□ Other (please specify) ____________

NAME ____________ DATE ____________
TITLE ____________
COMPANY NAME/DEPARTMENT ____________
ADDRESS ____________
CITY ____________ STATE ____________ ZIP CODE ____________

WE'D LIKE YOUR COMMENTS...

Your comments on the back of this form will be used to help us produce better manuals. All comments and suggestions become the property of Vermont Microsystems, Inc.

Vermont Microsystems, Inc.
Attn: Technical Publications Dept.
11 Tigan Street
Winooski, Vermont 05404

VMI

11 Tigan Street
Box 236
Winooski, VT 05404
Tel. (802) 655-3800