Title: WORKSTATION GRAPHICS ARCHITECTURE

Author: HENRY M. LEVY

Date: 7-November-1983

Version: 1.2

File: Dizzy::User$1:[Aurenz.wga.mem.wga]WGA.MEM

ABSTRACT

This document describes the software interface between a family of workstation display systems and a host computer. It defines commands to generate graphics and text, and commands to support keyboard, tablet, and mouse interaction. These commands are transmitted by the host driver and executed by a microprocessor-based controller in the workstation.

Revision:	Description	Author	Date
0.0-0.4	Prelim. Draft and Revisions	H. Levy	13-March-1982 to 10-January-1983
1.0	VS100 Release Base Level	H. Levy	1-March-1983
1.1	Updates based on VS100 implementation	L. Samberg	20-June-1983
1.2	Finalized Release for VS100 implementation	S. Aurenz	7-January-1984

PRELIMINARY

CONFIDENTIAL

1.0 INTRODUCTION

This document describes the command protocol supported by workstation display systems. These display systems implement five basic output commands:

1. General bitmap copy operation,
2. Text output,
3. Curve drawing,
4. Area fill,
5. Area flood,

and a collection of miscellaneous commands. In addition, they support a keyboard and one or more pointing devices (mouse or tablet). The Workstation Graphics Architecture is intended to be general and flexible enough to compatibly support future workstation products, such as multi-plane (color) and higher-functionality systems.

2.0 DISPLAY VOCABULARY

2.1 Display

For the purposes of this document, a physical device consisting of a high-resolution raster-scan monitor, keyboard, pointing device, control processor, microcode and firmware.

2.2 Firmware

A control program that is downloaded from the host into the display (see Microcode). It is code to be executed by the control microprocessor in the display.

2.3 Microcode

A control program which is fixed in hardware (i.e. not loadable), usually in ROMs or PROMs. It is code to be executed by the microcoded hardware in the display to perform high-speed BITBLT and graphical operations on framebuffer memory.

2.4 Pixel

A single picture element or addressable point on a display. Each pixel has a value, represented by one or more bits, that describes its state (i.e., the intensity or color of that point).

2.5 Two-Space

The pixels in a bitmap or on the display are organized in a Cartesian (rectangular) coordinate system of two dimensions. The origin of the coordinate grid is in the upper-left-hand corner, with the positive X axis extending right and the positive Y axis extending down. All points (pixels) are thus specified by their X and Y offsets (in pixels) from this origin.

2.6 Bitmap

A data structure consisting of a rectangular array of pixel values. The specification of a bitmap includes its starting address in memory and its dimensions. its Width is the X dimension in pixels, and Its Height is the Y dimension in pixels. Its Depth indicates how many planes, or bits/pixel, the bitmap occupies. The starting address points to the upper-left-hand corner of the bitmap.

2.7 Offset

Used to specify a point relative to some origin in two-space (See Two-Space). The coordinates are signed 16-bit X and Y components.

2.8 Extent

An extent (or, a rectangle extent) specifies the height and width, in pixels, of a rectangle in two-space. Both dimensions must be positive.

2.9 Rectangle

Used to specify a rectangular portion of a bitmap. A rectangle in two-space is specified by an Offset (of its upper-left-hand corner) and an Extent, with the bitmap's upper-left-hand corner as the origin.

2.10 Frame Buffer

The bitmap memory used to store the current value of each pixel and from which the physical display monitor is refreshed.

2.11 BITBLT

The transfer of a bit string or block from one location to another (bit block transfer, pronounced "bit blit").

2.12 Font

A collection of logically related images within a bitmap which may be individually addressed by index, e.g. the symbols of a character set.

2.13 Clipping Rectangle

A rectangle used to constrain an operation on a set of pixels in a bitmap. The intersection of a clipping rectangle with a destination bitmap rectangle forms a restricting boundary on the operation being executed.

2.14 Cursor

A small image that is displayed on the screen to indicate the current position of a selected pointing device. The cursor is automatically moved by the display hardware to reflect pointing device movement. The cursor image does not interfere with the current state of the frame buffer.

2.15 Halftone

A rectangular pattern used to tile a destination bitmap. That is, the halftone is replicated along its height and width to fill the destination. The point about which this replication is done is specified as an offset relative to the origin of the destination bitmap. Halftones are used to supply levels of shading or texture, as in newspaper printing.

2.16 Mouse

A small box-like device capable of sensing XY movement along a surface. It generally has one or more buttons, the function of which is defined by a particular application.

2.17 Digitizing Tablet

A flat rectangular surface capable of reporting the XY position of a special "pen" or "puck" placed on it. The pen/puck also has one or more buttons generally; again, their function is defined by a particular application.

3.0 ARCHITECTURAL OVERVIEW

The Workstation Graphics Architecture supports a family of display systems that connect to VAX or PDP-11 workstations. These displays differ in price, performance, size, and physical characteristics (e.g., color vs. black and white, resolution), however they all obey the same basic command set. The commands are defined by this document.

The workstation displays provide primitives to support the System Display Architecture (SDA), a high-level software interface to display programming. User application programs access the display via an SDA procedural interface. This interface allows for multiple applications to simultaneously access the display through separate windows on the display screen. Each application controls one or more virtual

displays, and the position of virtual display windows on the physical

screen is controlled by a screen manager responding directly to commands from the display operator. Thus, user application programs will never directly generate display commands as defined in this document; these commands are generated only by privileged operating system software.

The workstation display and associated input devices are thus multiplexed among several application processes. Therefore, the display hardware must supply primitives for window management, movement of windows on the display, partial covering of one window by another, and output to windows not currently visible on the screen. Basic to all such operations is the ability to efficiently copy bitmap rectangles from one area of the frame buffer or off-screen memory to another. In addition to the simple copy, the hardware also allows construction of the destination bitmap as a function of the source or source and destination bitmaps.

Each workstation graphics processor consists of a controlling microprocessor with its private instruction and data memory, a display monitor, a frame buffer memory from which the monitor is refreshed, and an interface to host memory. It may also have some additional memory for staging graphics operations or caching commonly used images, fonts, halftones, and so on. The display micro-processor typically has its private memory, the frame buffer memory, and some section of host memory mapped into its address space. Operations can be performed on data residing in any of these memories. That is, the workstation processor can operate on bitmaps stored in host memory as well as in local workstation memory. The instruction set of the display is formed by the microcode and firmware installed in it.

All addresses communicated between host and display are 32-bit values. Some hosts, displays, or interconnects may not support a full 32-bit address space and will therefore ignore some upper bits of the address. All coordinates are 16-bit signed values. The maximum size of a bitmap dimension (height or width) is therefore 2**15-1 pixels. In comparison, common frame buffer dimensions for existing monitors are on the order of 2**10 pixels on a side.

Host/display communications are handled through a shared-memory Control and Status Register interface. Graphics commands to the display processor are packet-oriented. The host transmits a command packet to the display; the display loads the command completion status in a control and status register and interrupts the host when done. Multiple command packets can be linked together to allow processing of several commands with a single device interaction. Other data may be transmitted spontaneously by the display to indicate the occurrence of various events. Command packets can contain the address of data in host memory to be read or written by the display. Any host data passed to the display by address must be locked in host memory for the duration of the command.

4.0 WORKSTATION OPERATION

This section describes the basic graphics operations of the workstation display system. These operations generally affect the contents of bitmaps stored in host or display memory. The workstation provides support for five basic graphics operations:

1. General bitmap copy (raster operations)
2. Text output
3. Curve drawing
4. Area fill
5. Area flood

4.1 PARAMETERS TO GRAPHICS OPERATIONS

The fundamental parameters to these basic operations are described below. Many terms are also discussed above in the Display Vocabulary. The components of each parameter will be described, as will its representation in the packet. Only the "generic" parameters are mentioned here; specialized parameters associated with only one command are mentioned in the section describing that command. Please refer to the section on Command Packet Formats for complete specification of all packets.

4.1.1 Value -

Used in this section to denote a single word. Groups of these words are built up to form the parameters.

4.1.2 Constant -

A constant is a 16-bit binary value passed directly in a command packet. In an N-plane bitmap, only the least significant N bits of the constant are used. The format of a Constant is:

Value<15:0> - Constant Value

4.1.3 Address -

A 32-bit address which points to an object (e.g. bitmap, font, string) in display-addressable memory. Note that the low word is given first. The format of an Address is:

Value<15:0> - Low word of address

Value<15:0> - High word of address

4.1.4 Offset -

An Offset specifies a point relative to some origin in two-space (See Two-Space). It usually designates the upper-left-hand corner of a rectangle or bitmap. The coordinates are signed 16-bit X and Y components.

Value<15:0> - X coordinate, in pixels
Value<15:0> - Y coordinate, in pixels

4.1.5 Extent -

An extent (or, a rectangle extent) specifies the height and width, in pixels, of a rectangle in two-space. Both dimensions must be positive.

Value<15:0> - Rectangle Width, in pixels
Value<15:0> - Rectangle Height, in pixels

4.1.6 Rectangle -

Used to specify a rectangular portion of a bitmap. A rectangle in two-space is specified by an Offset (of its upper-left-hand corner) and an Extent, with the bitmap's upper-left-hand corner as the origin.

Offset<31:0> - Coordinates of rectangle origin
Extent<31:0> - Width and height of rectangle

4.1.7 Bitmap -

A bitmap specifies a segment of memory containing the values of all of the pixels in a rectangle. The representation of the bitmap is device-dependent. A bitmap is specified by:

Address<31:0> - Base address of the bitmap memory
Value<15:0> - Width of the bitmap, in pixels
Value<15:0> - Height of the bitmap, in pixels
Value<15:0> - Number of bits per pixel in the bitmap

4.1.8 Sub-Bitmap -

A sub-bitmap specifies a rectangular subset of the pixels in a bitmap. It is thus specified by a bitmap, as above, plus the origin and extent of a rectangle within that bitmap:

Bitmap<79:0> - Specification of a bitmap memory
Offset<31:0> - Origin of subset rectangle within bitmap
Extent<31:0> - Extent of subset rectangle

4.1.9 Halftone -

A halftone is a rectangular pattern used to tile a bitmap (in the same way tiles are used to lay a design on a floor). The Halftone Alignment Offset specifies the origin of the halftone bitmap relative to the origin of the destination bitmap in two-space.

e.g. if the Offset is (0,0), the Halftone will be aligned to the upper-left-hand corner of the destination bitmap. A halftone is specified as a bitmap:

```
Bitmap<79:0>    - Specification of the bitmap
Offset<31:0>    - Alignment Offset of halftone pattern
```

4.1.10 List -

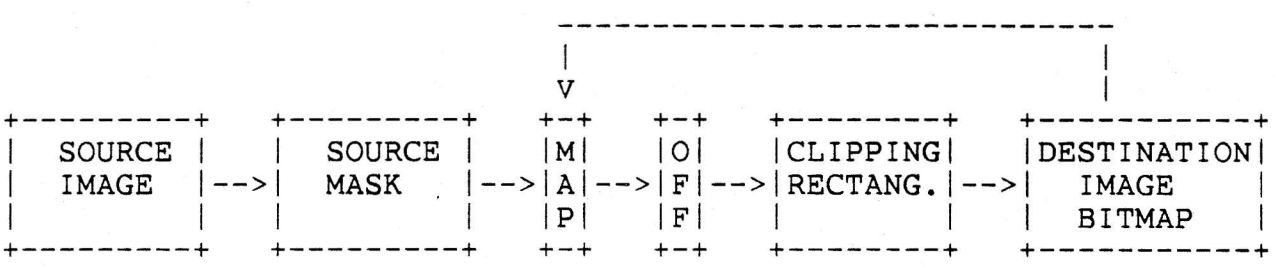
A list is any one-dimensional array of fixed-sized elements. Examples of lists are character strings, clipping rectangle lists, and line-drawing paths. The size of the individual elements is either known implicitly or specified by other parameters in the command. A list is specified by its base address and its length (a COUNT of the elements).

```
Address<31:0>   - Address of the start of the list
Value<15:0>     - Length of the list (COUNT)
```

4.2 DISPLAY GRAPHICS COMMANDS

As previously stated, the workstation display executes a set of five basic commands; each command is capable of processing some number of different parameter types or formats. In this section we describe the commands and their parameters in more detail.

Before presenting the commands, it is necessary to describe a model of the machine implemented by the workstation micro-processor. At its most basic level, this machine inputs a source bitmap and uses it to modify a destination bitmap in a specified way. However, in the most general case, only certain pixels in the source may be used, and only certain pixels in the destination may be available for modification. Our model indicates how these sources and destinations are specified. A picture of this machine is shown below:



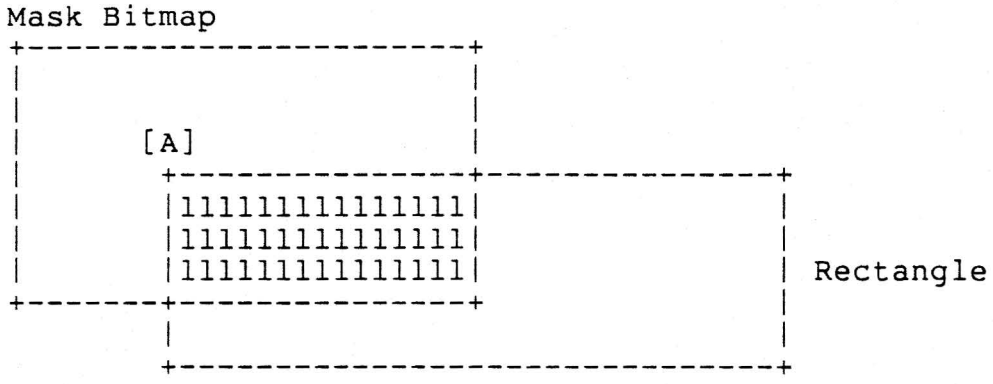
This machine is logically divided into three parts. The first part, consisting of the source image and source mask, determines what pixels of the source will be used to update the destination. The last part, consisting of the clipping rectangles and destination image bitmap, determines what pixels of the destination are available for modification. The map box in the center specifies a function with which source pixel values may be transformed before replacing destination pixels. Or, the destination pixels may be replaced with a function of both source and destination pixels, as shown by the arrow leading from the destination to the map. The destination offset box (OFF) in the center specifies where the source pixels are to be placed relative to the origin of the destination bitmap.

4.2.1 THE CLIPPING MODEL -

Here is a graphical illustration of the "Implied Clipping" which may take place when processing a source bitmap through the above machine. Only a subset of these operations need take place in most cases, depending on the user's selection of parameters. The operation proceeds in four steps:

[Step 1]

First, the Mask parameter is processed.
It may be either a Sub-Bitmap or Rectangle mask.



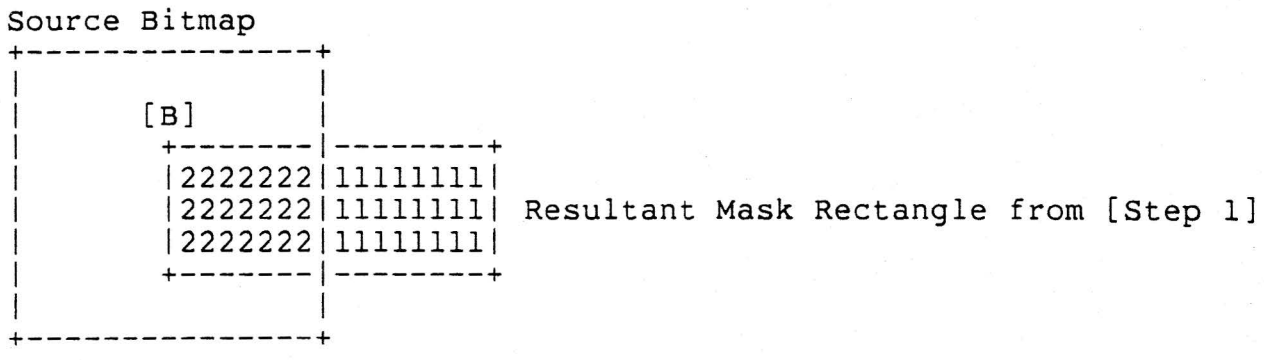
- o A Sub-Bitmap Mask specifies a Rectangle (offset and extent) within a Mask Bitmap. The Mask Offset specifies Point [A], and the selected intersection (the Sub-Bitmap Mask) is shown in "1's".

- o If a Rectangle Mask is used instead of a Sub_Bitmap Mask, no clipping is needed and [Step 1] is skipped. That is, the dimensions of the 'selected intersection' are taken directly from the Rectangle Extent in the packet.

- o The resulting Mask Rectangle (the Extent of the result) is placed on the source as below in step 2:

[Step 2]

Next, the Source is processed.
It may be either Bitmap, Constant, or Halftone.



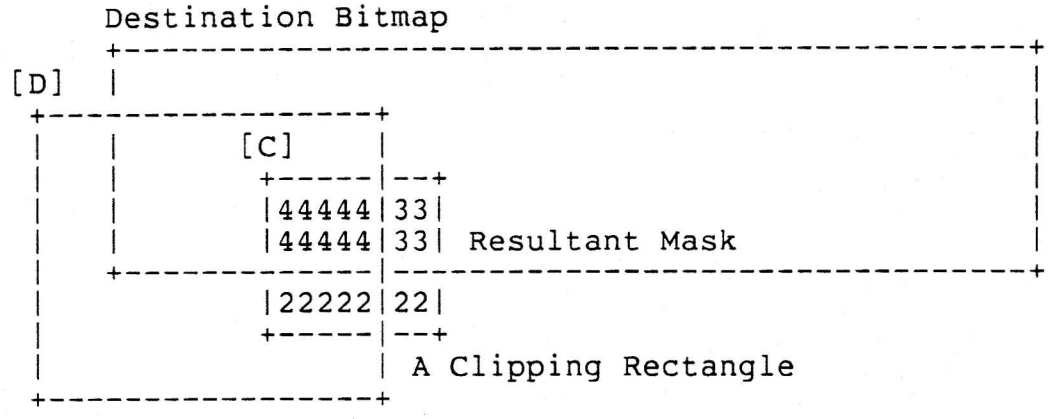
- The Source Offset, Point [B], specifies where the resultant Mask is placed relative to the Source Bitmap's origin. The "2's" represent the selected part of the Source Bitmap. Note that they now also represent the selected part of a Sub-Bitmap mask.

- If the source is a Constant or Halftone, no clipping is necessary and [Step 2] is skipped. The Resultant Mask is passed on unchanged to [Step 3].

  The Resulting Mask is then placed on the destination bitmap below in step 3:

[Step 3]

Next, the Destination bitmap and Clipping Rectangles are processed. There may be several clipping rectangles. [Step 3] is repeated for each.



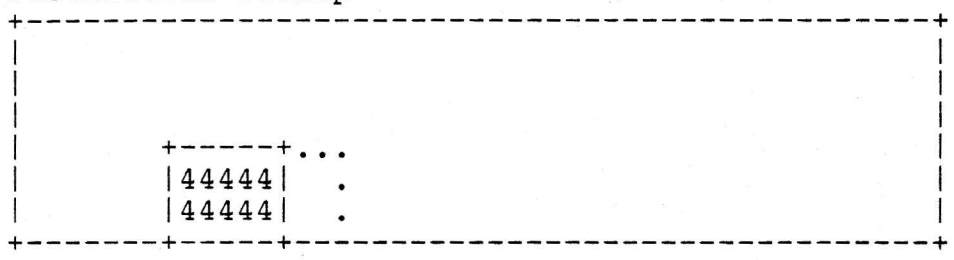
- The Resultant Mask Rectangle is placed on the Destination Bitmap according to the Destination Offset Point [C]. Then, one or more Clipping Rectangles may be applied to further restrict this selected area.
  The Clipping Rectangles are specified by an Offset Point [D] and an Extent (height and width).

- In this example, the "2's" were eliminated when the Resultant Ma was clipped to the Destination, the "3's" were eliminated when the Clipping Rectangle was applied, and the "4's" represent the resultant area available for modification.

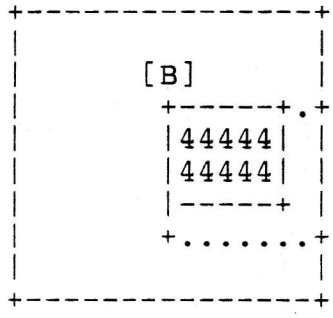
[Step 4]

o Now, these accumulated offsets and extents are "traced back" to find the corresponding selected areas of the Source Bitmap and Sub-Bitmap Mask:

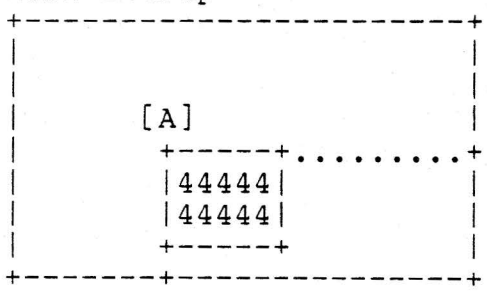
Destination Bitmap



Source Bitmap



Mask Bitmap



4.2.2 Copy Area Command -

The copy area command is the fundamental operation of the display system. Other operations are extensions of this basic copy area function. In its simplest form, copy area merely moves a source bitmap to a destination bitmap. That is, the pixels in a selected rectangular area of a destination bitmap are replaced by corresponding pixels in an identically-sized rectangular area of a source bitmap. Both bitmaps must be in display-addressable memory. In addition to simple replacement, the values moved to the destination can be either (1) a function of the source pixel values, or (2) a function of both source and destination pixel values. A Mask parameter may also be used to select some subset of the source. Finally, the user may specify one or more Clipping Rectangles to further alter the area affected by the operation.

The seven parameters accepted by the copy area command are:

- o Source Image
- o Source Offset
- o Source Mask
- o Destination Image Bitmap
- o Destination Offset
- o Map
- o Clipping Rectangles

These parameters are described in detail in the following sections. The formats of the parameters in the command packet are those shown in Section 4.1 above. Complete packets are pictured in Section 6.

4.2.2.1 SOURCE IMAGE -

The source image parameter specifies a bitmap whose pixel values are used to update the destination. The source image can be specified in three ways.

1. First, the source parameter can be a single constant value that is used to replace all pixels in the destination. This would be used, for example, to set the entire destination to a single value (color or shade).

2. Second, the source can be specified as a bitmap in display-addressable memory. If the source and destination bitmaps overlap, the copy operation sequences through the pixels so that source pixels are not modified before they are used to update the destination.

3. Third, the source can specify a halftone to be used to tile the destination as specified by the remaining parameters. A halftone is represeted as a bitmap. The bitmap is conceptually replicated horizontally and vertically as needed to fill the destination.

4.2.2.2 SOURCE OFFSET -

The source offset parameter consists of a point (two 16-bit signed values) specifying where the source mask parameter, described below, is to be placed relative to the source.

4.2.2.3 SOURCE MASK -

The source mask parameter defines a bounded subset of the source to be used to modify destination pixels. That is, the source mask can restrict the set of source bitmap pixels used in the copy operation. Two mask formats are available, depending on how the source subset is to be determined.

1. First, the mask can select a rectangular subset of the source bitmap. Only a rectangle extent need be specified (its height and width), since the source offset parameter, described above, determines where the origin of the rectangle will be placed relative to the source.

2. Second, the mask can be specified as a bitmap of zero and one values. This is the most general form, in which the one-valued elements of the bitmap select an arbitrary set of pixels from the source to be used in the operation. That is, the mask bitmap is used as a template. Once again, the source offset determines how the mask is to be placed relative to the source bitmap. The mask is specified as a sub-bitmap, as defined in the previous section.

Either source mask format can be selected with any of the possible source image parameters. For example, selection of a rectangle mask with a constant or halftone source generates a rectangle filled with that color or halftone for use as the source. The copy area command can thus be used as a simple rectangle fill. More complex fill operations can be performed with the fill area command, to be described later.

4.2.2.4 DESTINATION IMAGE BITMAP -

The destination image is always a display-addressable bitmap to be modified.

4.2.2.5 DESTINATION OFFSET -

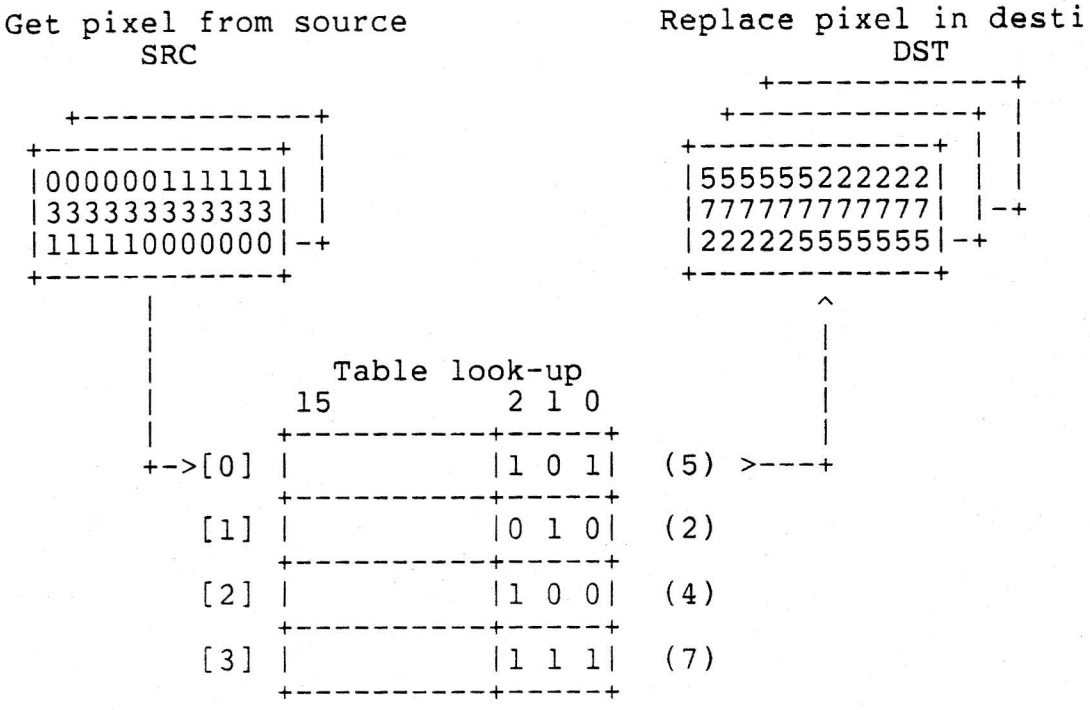
The destination offset parameter determines the placement of source pixels in the destination bitmap. Refer to the Clipping Model; after the source image and mask have been processed, the Destination Offset is used to specify the origin of the result. The result is then placed on the destination bitmap.

4.2.2.6 MAP - A set of source pixel values can be mapped into a different set of destination pixel values by using a form of the map parameter. A map can have three forms:

1. Identity Map. No map is specified. The source pixels directly replace the destination pixels.

2. Source Table-Lookup Map. In this case, each source pixel is used as an index into a table of destination pixel values. To map the pixels in an N-plane bitmap to pixels in an M-plane bitmap, the table would have 2**N entries. Each of these entries is a 16-bit word, of which the least-significant M bits are defined.

Consider, for example, the case below. The source bitmap has two planes (Z = 2), and the destination bitmap has three planes (Z = 3). The source pixels may therefore have values 0 through 3, and the destination pixels may have values 0 through 7. In this example, then, the source map would have 2**2 = 4 locations, which contain the destination pixel values, stored in the least-significant 3 bits of each location.



If the source Z parameter (the number of bits/pixel) is equal to 1, then the table may be included in the WGA command packet (2**1 = 2 16-bit word entries). If the source Z parameter is greater than 1, then the lookup table will not fit in the packet, and must be referred to via a pointer to the table's base address.

3. Function Code. The map may be a function code which designates one of a number of logical operations to be performed on the destination, using the source and destination pixel values as operands.

   This form of map assumes that the pixels in both the source and destination planes have the same number of significant bits.

   Since only one word is needed to specify the function code value, it may be passed either directly (literal function code) or indirectly (pointer to function code), even if the Z parameters of the source and destination are > 1.

   N-plane-source/N-plane-destination operations are performed

on corresponding bits between source and destination pixel values.

The user may specify any 2-operand logical function by its "Boolean Characteristic Number". This number is generated as below:

Consider a two-input truth table, where the source and destination pixel values are the inputs, and the new destination pixel value is the output.

```
Source Pixel Value
|   Destination Pixel Value
|   |    Output Pixel Value
|   |    |
v   v  | v
-------+---
0   0  | a
0   1  | b
1   0  | c
1   1  | d
       |
       +-----> abcd
```

Since there are two inputs, there are four possible combinations. The result of these combinations may be arranged, as shown above, into a four-bit number - the Characteristic Number. This defines a logical mapping function to the display:
Map Function Code Parameter:

```
+--------------------+
|             |a b c d|
+--------------------+
 15            3 2 1 0
```

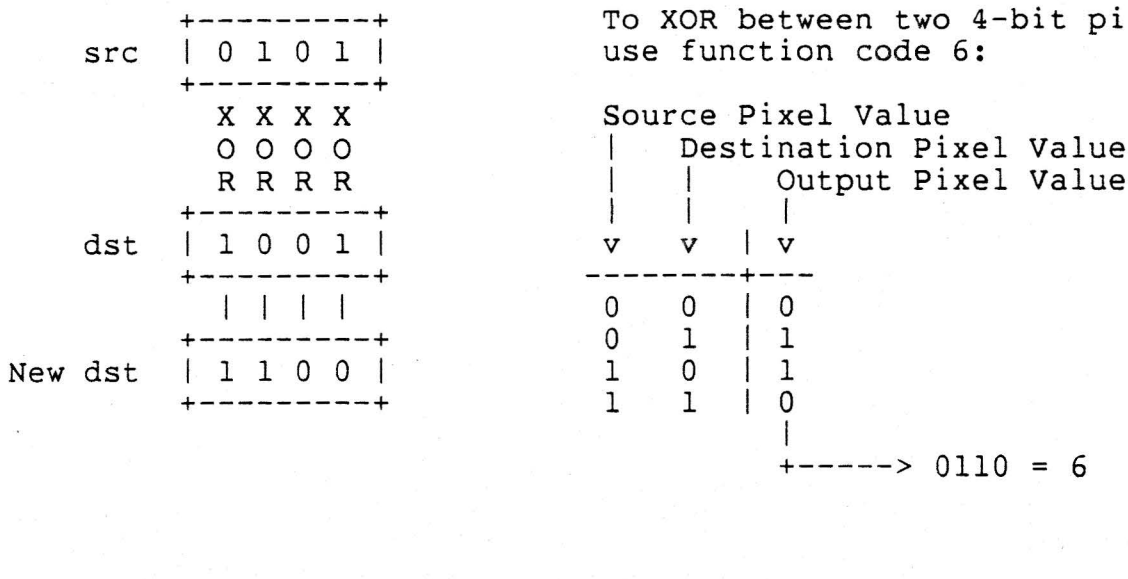
The Characteristic Number is a 4-bit value, and so it may define a set of 16 functions:

Function Code	Logical Function
0	0
1	src AND dst
2	src AND NOT dst
3	src
4	NOT src AND dst
5	dst
6	src XOR dst
7	src OR dst
8	NOT src AND NOT dst
9	NOT src XOR dst
A	NOT dst

B	src OR NOT dst
C	NOT src
D	NOT src OR dst
E	NOT src OR NOT dst
F	1

Note that since these are "bitwise" operations, two N-bit pixels may also be operands to the same 16 functions. The selected function is applied to corresponding bits in the operands. Hence, the same Characteristic Number/Function Code may be used to describe an operation between any two operands, no matter what their size.

Example:



4.2.2.7 CLIPPING RECTANGLES -

As shown in the Clipping Model, Copy Area operations are clipped to the boundaries of the destination bitmap. To clip the operation within the destination, one or more Clipping Rectangles may be used. A Clipping Rectangle specifies an area of the destination available for modification. A Clipping Rectangle is defined by an Offset (relative to the origin of the Destination Bitmap) and an Extent. The union of all of the Clipping rectangles Provides a boundary defining the area of the destination that can be modified. The clipping rectangles should not overlap. A single Clipping Rectangle may be given directly in a command packet. If more than one is required, a List of Clipping Rectangles must be specified.

4.2.3 Draw Curve Command -

The Draw Curve command is used to paint a source bitmap along a Path defined by a given sequence of points. The path between any two points may be either straight or curved. The Draw Curve command uses the same parameters as the Copy Area command, and has additional parameters to specify the path and pattern string.

The command proceeds as follows. First, the source is determined by the Source Bitmap, Source Mask, and Source Offset. Second, this source image is painted through the curve by successively moving its origin through the sequence of points within the Destination Bitmap as determined by the Path parameter. The source is combined with the destination as specified by the Map.

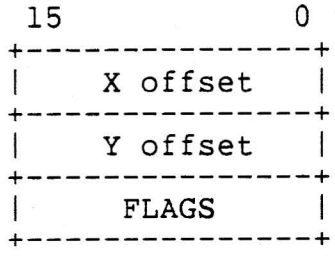
The parameters to the Draw Curve command are:

- o Source Image
- o Source Offset
- o Source Mask
- o Destination Image Bitmap
- o Destination Offset
- o Map
- o Clipping Rectangles
- o Path
- o Pattern String
- o Pattern Multiplier
- o Pattern State
- o Secondary Source
- o Secondary Source Offset

Specification of the first seven parameters is identical to the copy area command. The next sections describe the curve-specific parameters.

4.2.3.1 PATH -

A path is specified as a list of segments. Each path segment is described by its ending point and a Flag Word. The starting point is the end of the previous segment. The format of a path element (point) is:



The Flag Word describes the characteristics of that segment, and contains the following indicator bits:

1. Bit 0 - the Offset-Relative / Path-Relative bit, indicates how the ending point coordinates are to be interpreted. If bit 0 is clear (Offset-Relative mode), this point is interpreted as a point relative to the Destination Offset parameter specified above. If bit 0 is set (Path-Relative mode), this point is interpreted relative to the end of the previous segment in the list.

2. Bit 1 - the Draw / Move bit, indicates whether or not the segment should actually be drawn. If bit 1 is clear, the segment will be drawn. If bit 1 is set, the segment will not be drawn; the position is simply advanced to the next point. With this bit set (Move), several disconnected segments may be drawn with a single Draw Curve command. This bit would be set (Move), for example, to define the starting point of the first segment in a Path. Invisible segments can be also be used to provide additional information to the cubic spline algorithm. An invisible segment does not advance the pattern string, if one is specified.

3. Bit 2 - the Straight / Curved bit, indicates how the source should be moved from the starting to the ending point. If bit 2 is clear, a straight line is painted to the current point. If bit 2 is set, a curve is drawn using a cubic spline algorithm. For a curve to be drawn through a point, it must have a preceeding segment and succeeding segment to define the point's tangents. Then the curve will merge smoothly with the preceeding and following segments without abrupt bends.

4. Bit 3 - the Start Closed Figure bit, when set, indicates that this point is the first in a series of segments that defines

a closed figure (i.e. circle, square, polygon).

5. Bit 4 - the End Closed Figure bit, when set, indicates that this is the last point in the closed figure. The first and last points (that is, the start and end points) for a closed figure must be identical.

6. Bit 5 - the Draw Last Point bit, when set, indicates that the final point in the segment will be drawn. If bit 5 is clear, the endpoint will not be drawn.

Therefore, by manipulating the bits in the Flag Word, The Path may consist of both straight-line segments and curved (cubic spline) segments. These segments may be drawn "brush up" or "brush down". Using this mechanism, additional points can be supplied before the first visible point and after the final visible point, to define a curve more accurately.

When drawing a multi-segment path using a mapping function such as XOR, end-point-writing can be disabled for each segment so that its end-points are written only once.

The flag word MUST be used to specify the starting point and ending point of closed figures.

For example, a circle would be specified by the following five-point path:

```
point 1: move to P1, start closed figure
point 2: draw curve to P2
point 3: draw curve to P3
point 4: draw curve to P4
point 5: draw curve to P1, end closed figure
```

A segment of length zero (i.e., same starting and ending point) causes a single point to be drawn.

If the source image is specified as a halftone, the halftone pattern is always aligned to a fixed point (the Halftone Alignment Offset). This gives the effect of painting with a halftone.

4.2.3.2 PATTERN STRING -

The Pattern String Parameter is used to draw dashed or patterned segments in the Path. As mentioned above, segments may be either "straight" or "curved". A dashed segment alternates between writing and not writing pieces of the segment based on the sequence of set and clear bits in the pattern string. Patterned segments require the specification of a secondary source parameter. A patterned segment alternates between the source and the secondary source based on the sequence of set and clear bits in the pattern string. This would be used, for example, to draw a line that alternates between two colors.

The Pattern String parameter is a 16-bit word which contains the drawing pattern. The pattern may be 0 to 16 bits long. The Least Significant Bit is used first, and the pattern rotates to the right, wrapping around as needed.

The pattern is only applied to segments that have the Draw flag bit set as described above; invisible (Draw flag clear) segments do not advance the pattern. A Pattern String of length 0 draws a solid segment (as though there were no pattern).

The Draw Curve command causes an iterated copy area of the source to each pixel in the destination between segment endpoints. The Pattern String specifies how or whether the destination is to be modified at each pixel. A command modifier bit determines the mode in which the pattern string parameter will be used. The two modes are:

1. Single Source Mode - In this mode, a set bit in the pattern causes the source to be copied to the current destination pixel along the curve path, and a cleared bit causes the destination pixel to be skipped. This format lets the existing background "show through" the spaces in the pattern. The result is called a Dashed Line.

2. Alternate Source Mode - In this mode, a set bit in the pattern causes the source to be copied to the current destination pixel along the curve path, and a cleared bit causes a different source to be copied to the destination.

   This Secondary Source is passed in the command packet when

   alternate source mode is used. The result is called a Patterned Line.

Thus, Single Source mode alternates between writing and not writing of a single source. Alternate Source Mode alternates between the writing of two different sources.

4.2.3.3 PATTERN MULTIPLIER -

The Pattern Multiplier parameter specifies the number of times each bit in the Pattern String should be repeated before moving on to the next bit. For example, if the Pattern String is '10' and the Pattern Multiplier is 3, the pattern '111000' will be used to generate the patterned curves and lines in the draw command.

4.2.3.4 PATTERN STATE -

The Pattern State parameter allows a draw curve command to continue at the Pattern String position at which a previous draw curve completed. In this way, Pattern Strings can be used across multiple Draw Curve commands.

The Pattern State consists of two 16-bit words, the Pattern Position and Pattern Count. Pattern Position specifies the starting bit within the pattern string, and must be less than the size of the string (that is, it runs from zero to length-1). Pattern Count specifies the starting count to be used for that bit on its first use, and must be less than or equal to the pattern multiplier (that is, it runs from one to the multiplier count).

The Pattern State parameter can be updated by the display following the completion of a Draw Curve command to indicate where in the Pattern String the next Draw Curve command should begin. The Pattern State can be specified as a literal in the command packet or as the address of the two-word parameter to be updated.

If a Pattern State is specified in the Draw Curve command, scanning of the Pattern String begins at the specified Pattern Position. The specified starting bit is then repeated PATTERN MULTIPLIER - PATTERN COUNT times. The scan then continues at the following bit, with each bit used PATTERN MULTIPLIER times.

4.2.3.5 SECONDARY SOURCE -

In alternate source mode, the secondary source parameter specifies the source that will be copied to the destination whenever a zero bit in the pattern string is encountered. The secondary source can be specified as any of the formats allowed for a source in the copy area command.

4.2.3.6 SECONDARY SOURCE OFFSET -

The secondary source offset specifies how the source mask will be applied to the secondary source, when the secondary source is selected in pattern string alternate source mode.

4.2.4 Print Text Command -

The Print Text command is the basic character string output operation for the Workstation Graphics Architecture. The command scans along a given text string, looking up the character indexes in a user-defined font. The selected character cells are written horizontally, one after the other, in the destination bitmap.

For special formatting of output strings, such as horizontal and vertical adjustments (e.g. subscripts), an optional control string may be included.

To support simple string justification, the user may set the spacing between character cells by specifying a fixed inter-character pad. The width of the space character may be adjusted by specifying a fixed additional space pad.

The Print Text command is an extension of the basic Copy Area. The text string is used to select bitmap cells in the font that become the source parameters to the copy operations.

The images of the characters or symbols in a font are stored in a bitmap. This bitmap, plus some other indexing information, forms the WGA font data structure. The font is specified as the address of such a data structure in display-addressable memory, and may be used either as a bitmap source image or as a bitmap mask.

The parameters to Print Text are:

- o Source Image
- o Mask Font
- o Destination Image Bitmap
- o Initial Destination Offset
- o Map
- o Clipping Rectangles
- o Text String
- o Control String
- o Inter-character Pad
- o Space Pad

The sections below describe the parameters in more detail.

4.2.4.1 SOURCE IMAGE -

The source image may be specified in one of three ways:

1. First, it can be the address of a font data structure, as defined below. The character cells in the font bitmap are copied directly to the destination bitmap, with no mask involved. The pixel values copied may, however, be modified with a mapping function.

2. Second, the source image can be a constant pixel value. This value is used as the writing color for characters whose symbols are defined by a mask font.

3. Third, the source image can be a halftone. The halftone is used as the writing color for characters whose symbols are defined by a mask font.

4.2.4.2 MASK FONT - -

The mask font parameter is the address of a one-bit-per-pixel font data structure that defines the symbol shapes of the characters in the font. The mask font is used in conjunction with a constant or halftone source to define the shape and writing color of the characters in the text string.

4.2.4.2.1 FONT DATA STRUCTURE - -

The font parameter is specified as the address of a font data structure in display-addressable memory. This structure has three parts:

- The Header
- The Image Bitmap
- The Leftarray [optional]

The Header contains central information about the font, the Bitmap contains the character images, and the Leftarray is a table of offsets used to find the character cells in the Bitmap.

The Font parameter in the command packet is the base address of the Header. All "pointers" in the Header, e.g. the LEFTARRAY pointer and the pointer in the BITMAP specification, are byte offsets relative to this address. So to compute the base address of the Leftarray, for example, one must add the Leftarray pointer (offset) to the Font parameter (Header address). In normal usage the data structure is contiguous.

4.2.4.2.1.1 Header Format -

Bitmap<79:0>	FONT BITMAP. Specification of the bitmap containing the character images
Value<15:0>	FIRSTCHAR. The first valid character index in the font.
Value<15:0>	LASTCHAR. The last valid character index in the font. Any character not within the range (FIRSTCHAR to LASTCHAR) inclusive is an error. The display terminates processing and reports the error to the host.
Address<31:0>	LEFTARRAY. A pointer to an array of 16-bit elements. There is one element for each character in the font between FIRSTCHAR and LASTCHAR. Each element contains the X offset (see Offset in Display Vocabulary) of the left edge of a character cell in the bitmap. If the WIDTH parameter is non-zero, then all cells have the same width, no Leftarray is needed, and this pointer need not be defined.
Value<15:0>	BASELINE. The Y offset of the character baseline from the top of the character cells (since origin is upper left).
Value<15:0>	SPACE. The index of the space character in the font (32 decimal for ASCII fonts).
Value<15:0>	WIDTH. If all characters in a font have the same width (a fixed-width font), then this parameter specifies that width (in pixels), and no Leftarray is needed. For variable-width fonts (ones with a Leftarray), this parameter must be zero.

4.2.4.2.1.2 The Bitmap -

The actual character images are stored in "strike" format; that is, all the character images are concatenated to form a horizontal bitmap strip. The first character is on the left and the last is on the right. The characters are aligned so that they have a common baseline. The height of the bitmap extends (at a minimum) from the bottom of the lowest descender to the top of the tallest character. There is no restriction on the height of a font, or the width of the characters within it, save for certain practical restrictions such as memory address space. Each character in the font is represented in the bitmap; absent characters have images of zero width.

4.2.4.2.1.3 The Leftarray -

Each character in a variable-width font is indexed through the Leftarray. If the WIDTH parameter is non-zero, this array is not needed.

The following symbols are defined for the figure and explanation below:

F = The index of the first defined character in the font.
L = The index of the last defined character in the font.
R = L - F, The range of defined characters in the font.
N = index - F, The "normalized" index of the character.

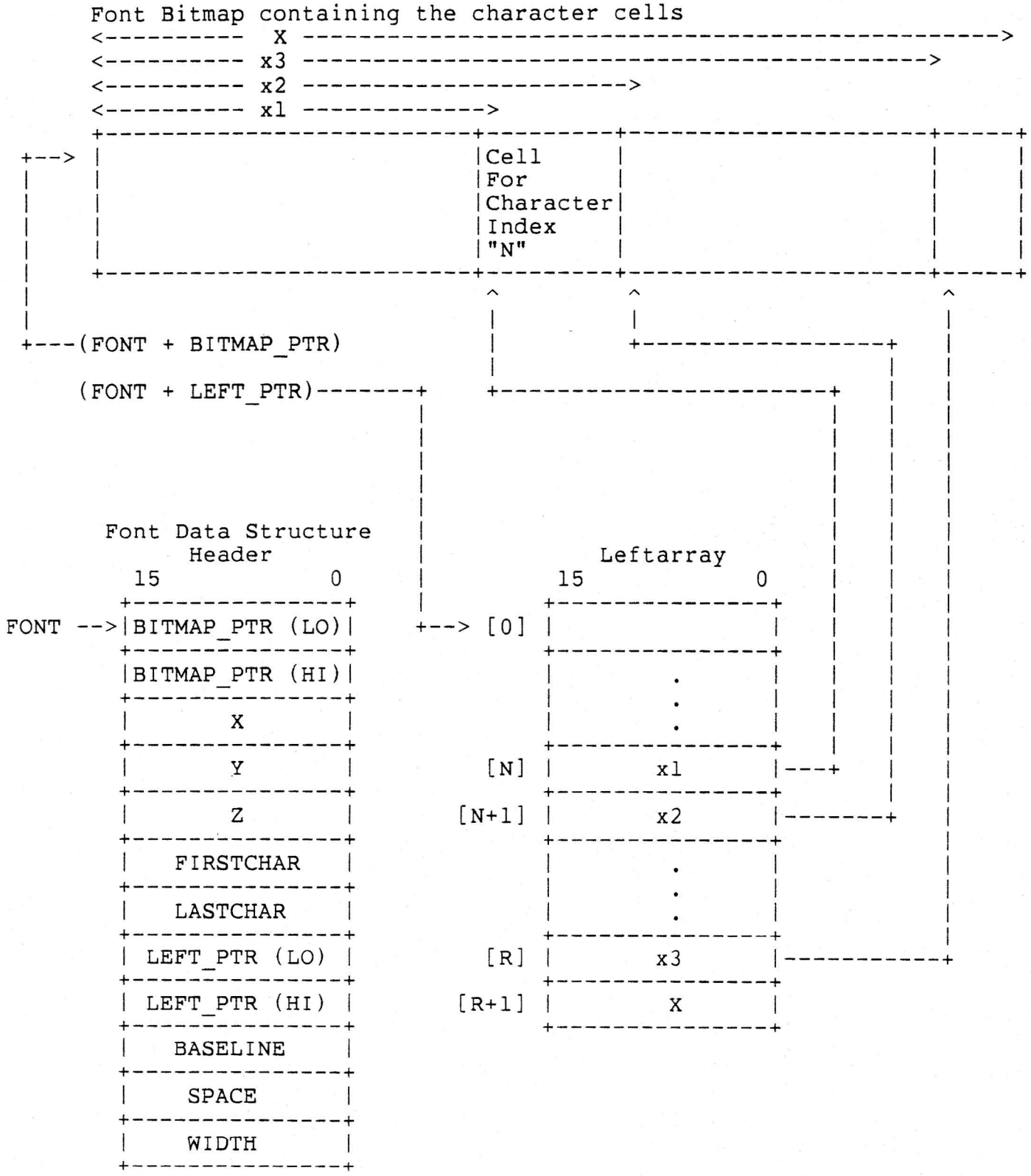
When a character index is fetched from the text string, it is "normalized" to N, which is in the range 0 to R. Note that there are therefore R+1 characters in the font. And the Leftarray has R+2 elements.

Now the offset of the left edge of character N (x1 in the figure) is found in the N'th element of Leftarray, and the left edge of the next character (x2 in the figure) is found in the next element. Subtracting these two numbers gives the width (in pixels) of the character N.

All of the characters have the same height - the height of the Bitmap.

Note the case for the last character in the font. There are no characters following it, but there is a location in Leftarray containing the proper 'left edge' so that the last character's width may be correctly computed as before.

ACCESSING CHARACTER CELLS IN A VARIABLE-WIDTH WGA FONT



A font may be used in two ways to specify the character images. First, a font may be used as a source image. In this case, each rectangular character cell is copied to the destination. The cell contains both the character image and a background. A map parameter can be used to transform the pixel values for the image and background. Multiplaned systems can use a source image font to contain anti-aliased characters (characters with grey scale).

Second, the font may be used as a source mask, or, mask font. A mask font is always one bit per pixel, independent of the number of bits per pixel in the destination bitmap. The cells of the mask font simply define the shapes of the characters. When a mask font is used, the source image parameter specifies the writing color. The source image can be either a constant or a halftone. When characters are output using a mask font, only the character shapes themselves are written, that is, there is no background rectangle for the characters. Mask fonts thus allow for writing of characters over other images without obliterating the underlying rectangular area, or for writing overstruck characters. They also provide a storage-efficient mechanism for writing halftone or single-colored characters.

4.2.4.3 DESTINATION IMAGE BITMAP -

The destination image bitmap is identical to the destination image bitmap specified in the copy area command.

4.2.4.4 INITIAL DESTINATION OFFSET -

The Initial Destination Offset is a point in the destination image bitmap where character writing begins. It is the same as the Destination Offset mentioned in the Clipping Model, but may be used in one of four modes. There are two modifier bits associated with the Initial Destination Offset, named Update and Indirection, respectively. They work as follows:

Initial Destination Offset
Modifier Bits
U = Update, I = Indirection

U	I	mode
0	0	literal
0	1	indirect
1	0	update literal
1	1	update indirect

1. literal
   The (x,y) offset is used as given in the WGA command packet.

The updated offset is not saved when the command is finished.

2. indirect
   The (x,y) offset is stored in a 2 word block pointed to by an address given in the WGA command packet. The offset in the block will not be updated when the command is finished.

3. update literal
   The (x,y) offset is used as given in the WGA command packet. The updated offset is saved by storing it back in the WGA com packet. An offset saved in this way may be referenced by re-u the same WGA packet. This is the case in single-character ech (see below).

4. update indirect
   The (x,y) offset is stored in a 2 word block pointed to by an address given in the WGA command packet. The updated offset w be saved by writing it back to this same block. The address p in the WGA packet will remain unchanged.

(Note: For efficient echoing of keyboard input characters, a print text command packet specifying a single character string can be cached in display local memory. This packet will specify the address of a single character buffer (in host memory) and the address of the initial x-y destination offset (also in host memory). The host need only load the character and issue the print text command, assuming that the x-y position is maintained by the display as indicated above. Storing the packet in display memory saves the time required to copy the command over the host/device interface.)

4.2.4.5 MAP - This parameter is specified as in the copy area command.

4.2.4.6 CLIPPING RECTANGLES - The clipping rectangles are identical to the clipping rectangles in the copy area command.

4.2.4.7 TEXT STRING - The text string is a list of 8-bit or 16-bit character indices. The size of the characters is determined by an opcode modifier.

4.2.4.8 CONTROL STRING - The control string is a list of 16-bit words that specify how the text string is to be output. Normally, characters are written on a single horizontal line with the spacing determined by the width of each character as stored in the font. However, the control string provides a series of commands that allow for x-y position adjustment as well as text string character skipping. Each control string consists of a 16-bit opcode followed by zero to two 16-bit operands. The control commands are defined as follows:

1. OUT(N) (opcode=0) outputs the next N characters of the text string, where N is the single operand.

2. OUTALL (opcode=1) outputs all remaining characters of the text string.

3. SKIP(N) (opcode=2) skips the next N characters of the text string, where N is the single operand.

4. ADJUST(X,Y) (opcode=3) adjusts the current character position by X and Y, where X and Y are signed horizontal and vertical adjustments specified in pixels, where X and Y are the two operands.

If the text string is exhausted before the control string, processing of the control string continues with the following interpretation: (1) any ADJUST commands are obeyed, (2) any OUTALL commands are ignored, and (3) any SKIP(N) or OUT(N) commands with N > 0 cause termination of the command with error status. If the control string is exhausted before the text string, the command terminates successfully at that point.

4.2.4.9 INTER-CHARACTER PAD -

The inter-character pad parameter is a constant value specifying the number of pixels to be added to the current destination offset following each character printed (including the last character in the text string).

4.2.4.10 SPACE PAD -

The space pad parameter is a constant value specifying the number of pixels to be added to the current destination offset following each space character printed.

4.2.5 Fill Area Command -

The fill area command is used to construct a source consisting of one or more closed shapes, where each of the closed shapes is filled with a single color or halftone value. The generated source image is copied to the specified location in a destination bitmap. For example, fill can be used to place a circle filled with a single color or shade within a destination bitmap.

The fill command is used when the boundary or boundaries of the area to be filled are known and can be defined by a list of straight or curved segments. The flood area command, on the other hand, is used when the boundary is not completely known, but the user can specify one internal point of the closed area.

The source parameter specifies the pixel value or halftone to be used to fill the bounded area or areas. A path list, specified as in the draw curve command, specifies the boundaries of the areas to be filled. The path list defines the bounded areas within the specified destination bitmap. Thus, this command causes one or more areas of a destination bitmap to be filled with one or more similarly-colored shapes.

At most one clipping rectangle can be used with fill area.

The parameters to the fill area command are:

- o Source Image
- o Destination Image Bitmap
- o Destination Offset
- o Map
- o Clipping Rectangle
- o Path

The parameters are described as follows:

4.2.5.1 SOURCE IMAGE -

The source image specifies the constant pixel value or halftone with which the closed area specified by the path parameter will be filled. The constant or halftone is specified as in the copy area command. Bitmap source image is not allowed.

4.2.5.2 DESTINATION IMAGE BITMAP -

The destination image bitmap is identical to the destination image bitmap in the copy area command. It specifies the bitmap in which the filled shape will be placed.

4.2.5.3 DESTINATION OFFSET -

The destination offset is identical to the destination offset in the copy area command, and specifies the placement of the filled image in the destination.

4.2.5.4 MAP -

The Map parameter is identical to the map in the copy area command.

4.2.5.5 CLIPPING RECTANGLE -

The clipping rectangle parameter for the fill area command specifies a single clipping rectangle to constrain the fill operation. If no clipping rectangle is supplied, the operation is constrained by the size of the destination bitmap.

4.2.5.6 PATH -

The path specifies one or more closed areas to be filled within the destination bitmap by the pixel values specified by the source parameter. A path is specified as a list of segments. The segments are described by a path list specified exactly as in the draw curve command. Each segment of the path can be a straight or curved line. Multiple disjoint closed areas are generated through use of the move flag bit. If the path describes an open figure, the results are unpredictable, but the operation will always be confined to the given bounding box (either the clipping rectangle or the destination bitmap border).

4.2.6 Flood Area Command -

The flood area command is used to fill bounded areas of a destination with a single color (pixel value) or halftone pattern. Moreover, the flood command is similar to copy area with two particular differences: (1) the source pixels are not mapped, but are copied directly, and (2) the destination pixels that are modified are determined by a flood algorithm, and lie within a closed area bounded by one or more pixel values.

In more detail, the flood algorithm proceeds in several steps. First, the algorithm determines the area of the destination to be flooded; that is, it locates the inside and outside portion of the closed area. Moreover, the algorithm builds a binary mask which when applied to the destination, will select those pixels on the inside of the flooded area. The determination of the bounded area requires two parameters: a seed point and a map. The seed specifies a single pixel within the destination; this point is known to be within the bounded area. The boundary map is a table of zeros and ones, used to determine whether points are internal or boundary points. The algorithm searches every pixel adjacent to the seed pixel. The value of each pixel is mapped through the boundary map table. If the mapped value of the pixel is zero, then this point is an inside point, otherwise it is a boundary point. If an inside point is found, the algorithm continues to examine all of its neighbors. Processing continues until the neighbors of all internal points have been examined. If the seed point is found to lie on a boundary, the algorithm terminates immediately.

Once the boundary has been determined, the inside area is flooded with the source. The parameters to flood area are:

- o Source Image
- o Destination Image Bitmap
- o Seed Point
- o Clipping Rectangle
- o Boundary Map

The parameters are described in more detail in the following sections.

4.2.6.1 SOURCE IMAGE -

The source image can be specified in two of the three formats available for the copy area command. For flooding, the source will be either (1) a constant, flooding the bounded area with a single color, or (2) a halftone, flooding the area with a halftone pattern.

4.2.6.2 DESTINATION IMAGE BITMAP -

The destination image bitmap is the bitmap containing the image to be flooded.

4.2.6.3 SEED POINT -

The seed point specifies the coordinates of a single point in the destination bitmap that lies within the internal region to be flooded.

4.2.6.4 CLIPPING RECTANGLE -

The clipping rectangle parameter for the flood area command specifies a single clipping rectangle to constrain the flood operation. If no clipping rectangle is supplied, the operation is constrained by the size of the destination image bitmap.

4.2.6.5 BOUNDARY MAP -

The Boundary Map tells the command which pixels of the closed figure in the destination bitmap are internal (to be flooded) and which are on the boundary (to be left alone). The Boundary Map parameter is a table whose one-bit elements are indexed by the pixel values in the destination. If the element is set, the corresponding pixel value is an external point, and if the element is clear, the pixel value is an internal point.

Note that unlike the previous commands, the Flood Area command map is used to map pixels in the destination.

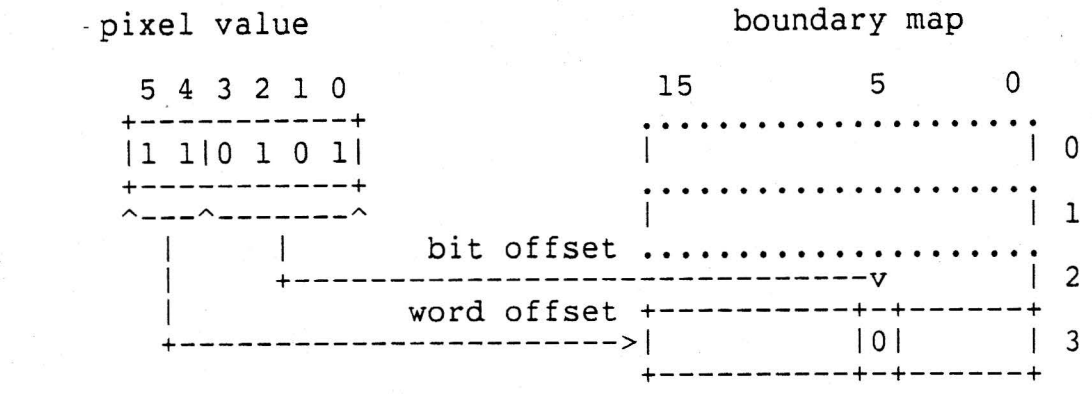
For an N-bit-per-pixel destination bitmap, 2**N pixel values are possible, and so a table with 2**N entries is needed. However, since each table entry contains either a zero (indicating an internal point) or a one (indicating a point on the boundary), only one bit is needed for each entry.

If N is less than or equal to four, then the table only uses one word, and only 2N bits of it are defined. For example, in a system with 3 bits per pixel, the 8-bit boundary map 00101110 indicates that pixel values 1, 2, 3, and 5 are boundary values, while pixel values 0, 4, 6, and 7 are interior values.

If N is greater than four, however, more than one word is needed. The least-significant four bits are used to select the proper bit within a word as before, and the remaining most-significant bits are used to select the proper word in the boundary map.

Example: 6 bit-per-pixel destination bitmap

pixel value = 35 (hex)



and pixel value 35 (hex) is an internal point.

4.3 DISPLAY CURSOR COMMANDS

The display cursor is a small image or icon that is automatically displayed on the screen at a point determined by the position of the default pointing device. The default pointing device is usually a mouse, but may also be a graphics tablet. The cursor is automatically moved by the display micro-processor to reflect movements of the default pointing device. The maximum size of the cursor icon is 64x64 pixels. The icon is cached by the display and modification of the icon bitmap in memory will not affect the image on the screen.

4.3.1 Load Cursor Command -

The load cursor command initializes the cursor to a new image. The image is specified by source image, source mask, and map parameters, as in the copy area command. The display device places the origin of the cursor rectangle at the current position of the pointing device. The command has the following parameters:

- o Cursor Source Image
- o Cursor Source Offset
- o Cursor Source Mask
- o Cursor Map
- o Cursor Attributes

whose definition follows.

4.3.1.1 CURSOR SOURCE IMAGE -

This cursor source image specifies the image of the cursor icon. This parameter is specified as in the copy area command.

4.3.1.2 CURSOR SOURCE OFFSET -

This parameter is specified as in the copy area command.

4.3.1.3 CURSOR SOURCE MASK -

This parameter is specified as in the copy area command, and generally defines the shape of the cursor icon.

4.3.1.4 CURSOR MAP -

This parameter is specified as in the copy area command. The map parameter defines how the cursor is added to the screen. If no map is specified, the pixel values specified by source and source mask are used to replace the pixels in the frame buffer at a destination determined by the cursor reference point. That is, the cursor image is written to the screen. Or, some systems may use a cursor that is XOR'ed or OR'ed onto the screen, in which case a full map would be specified. In any case, the screen state is not affected by the cursor; when the cursor is moved, the original pixel values in the area previously occupied by the cursor are restored.

4.3.1.5 CURSOR ATTRIBUTES -

The cursor attributes parameter defines how the cursor is displayed. The bits are defined as follows:

ATTRIBUTES<0> This bit specifies whether the cursor should blink or not. When set to 0, no blinking of the cursor occurs; when set to 1, the cursor is blinked at an implementation-defined interval.

4.4 DEVICE ORIENTED COMMANDS

4.4.1 Attach Cursor Command -

At any point in time, the cursor may "track" the position and movements of a single pointing device connected to the display. The Attach Cursor command tells the display which device to use. If "No Device" is selected, the cursor is "disconnected" from all devices, but its position may still be changed with the "Set Cursor Position" command.

The command has only one parameter:

- o Device Type

4.4.1.1 DEVICE TYPE -

This specifies which device will control the movements of the cursor. The defined parameter values are:

- 0 = no device
- 1 = mouse
- 3 = tablet

4.4.2 Set Cursor Position Command -

This command moves the cursor to the point specified by the parameter:

- o Location

If the cursor is attached to the mouse, the mouse's position will also be set. The mouse only reports relative movement from a given point, and so its 'position' may be defined by the user.

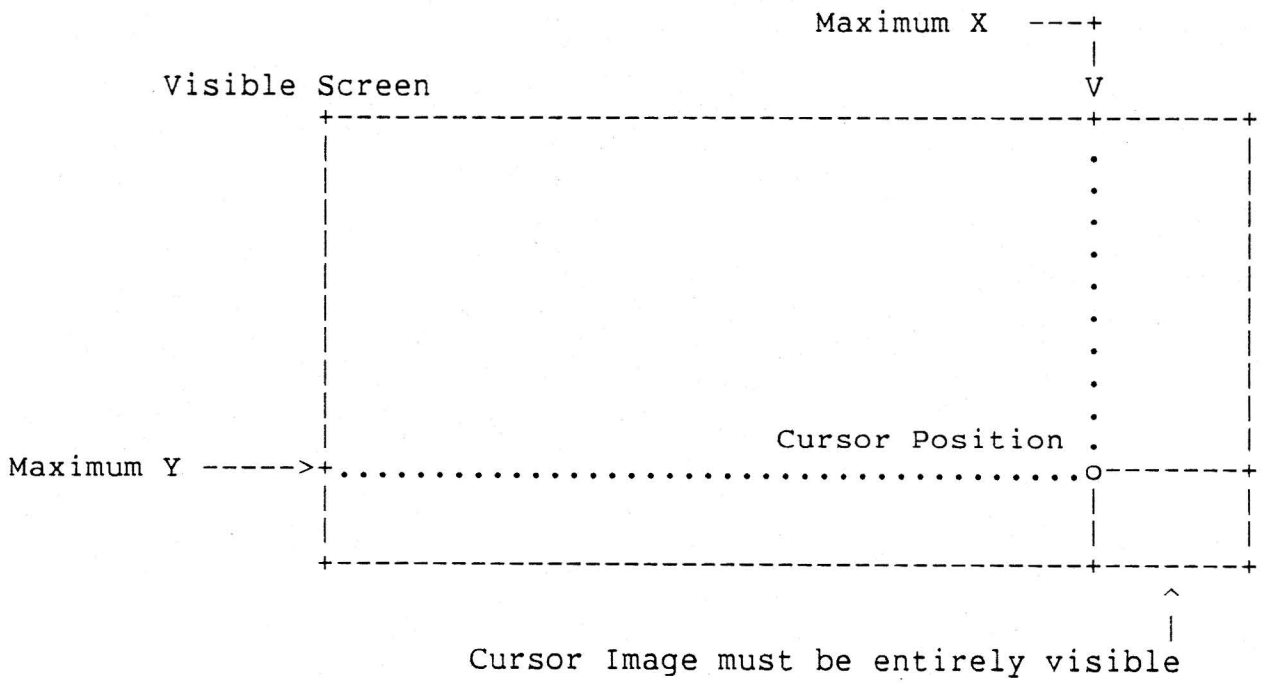
The tablet, however, reports an absolute position on the tablet surface, and this position may not be redefined by the user. Therefore, an error will occur if the user attempts to set the position of the cursor when it is attached to the graphics tablet.

While the cursor may not be moved off the screen, it can be made invisible by loading a zeroed mask bitmap.

4.4.2.1 LOCATION -

The Location parameter specifies an XY offset from the origin (upper-left-hand-corner) of the visible screen. The cursor's origin is aligned to this point.

Cursor Movement Within the Visible Screen Area



If the Location is not within the rectangle bounded by Maximum X and Maximum Y, the Location will be clipped to the rectangle's boundaries, such that the entire cursor is on the screen.

4.4.3 Get Cursor Position Command -

This command requests the current position of the cursor. The cursor position is returned in a two-word field in the command packet. There are no input parameters; however, the command packet contains a two-word field in addition to the header.

4.4.4 Get Mouse Position Command -

This command requests the current position of the mouse. The position is returned in a two-word field in the command packet. There are no input parameters; however, the command packet contains a two-word field in addition to the header. Normally, the reported position will be identical to that accumulated by the device as it moves about its origin point. However, if the mouse is attached to the cursor (via the "Attach Cursor" command), then the position reported will conform to the position of the cursor (and thus will be constrained to the boundaries of the visible screen, etc.).

4.4.5 Set Mouse Characteristics Command -

This command is used to specify how the cursor tracks mouse movement when the cursor is attached to the mouse. The mouse has no absolute position; its position is reported as relative movements from a given point.

The mouse reports CHANGE in position as it moves. Hence, the mouse position is updated by adding the accumulated change on to the current position. Of course, the change could be positive or negative.

It is frequently useful to "scale" this accumulated change in order to alter the mouse's "action" or "responsiveness". The WGA has two scaling algorithms from which to choose.

This incremental movement may be scaled in one of two ways: Linearly or Exponentially. A command modifier is used to select which one will be used.

The parameters to Set Mouse Characteristics are either:

- o Tracking Ratio

or:

- o Threshold
- o Scale Factor

4.4.5.1 TRACKING RATIO -

The tracking ratio specifies the distance relationship between device movement and cursor movement on the screen. It consists of two 16-bit integers, a multiplier and a divisor. Each unit of device movement corresponds to

cursor_movement = (device_movement*multiplier)/divisor

units of movement on the screen. Thus, the cursor tracks the device linearly.

4.4.5.2 THRESHOLD And SCALE FACTOR -

When an exponentially tracking (accelerating) cursor is selected, tracking is linear (with a Tracking Ratio of 1:1) up until the Threshold is reached. When the incremental mouse movement exceeds the threshold, the move is scaled by the Scale Factor. For example:

Threshold = 2
Scale Factor = 3

Mouse Movement	Resulting Cursor Movement
-5	-11
-4	-8
-3	-5
-2	-2
-1	-1
0	0
1	1
2	2
3	5
4	8

4.4.6 Set Tablet Characteristics Command -

This command is used to specify how cursor tracking is handled when the cursor is attached to the tablet. The tablet reports an absolute position which is scaled/modified by the Tracking Ratio or Quantization Ratio. As with the Set Mouse Characteristics command, the scaling algorithm is selected by a command modifier. Since the tablet always reports absolute positions (which correspond to well-defined locations on the tablet surface), exponential mapping is not defined.

The parameters to Set Tablet Characteristics are either:

- Tracking Ratio

or:

- Quantization Ratio

4.4.6.1 TRACKING RATIO -

This parameter specifies a linear scaling ratio as in the Set Mouse Characteristics command.

4.4.6.2 QUANTIZATION RATIO -

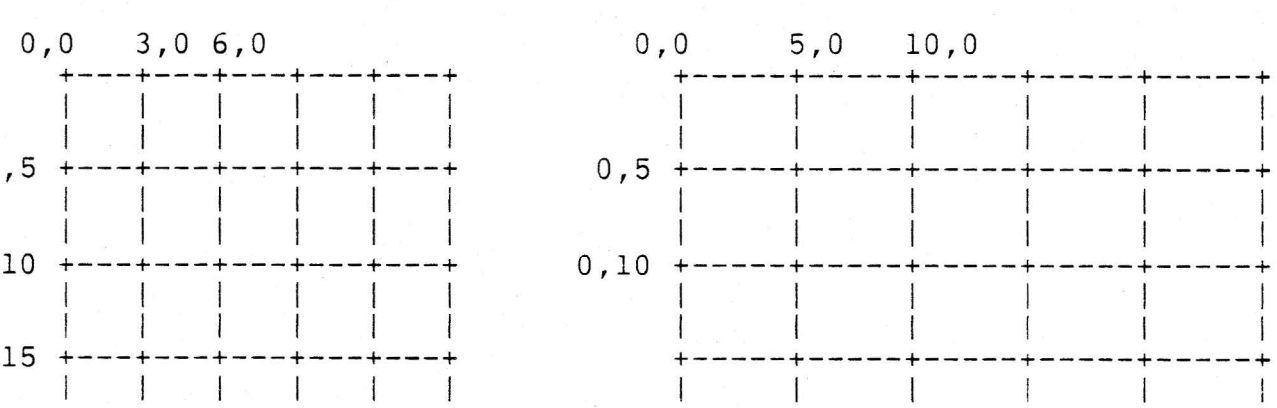
This scaling algorithm divides, or quantizes, the surface of the tablet and display screen into rectangular cells. When the tablet pointer is moved from one "input cell" to another, the output position is moved to the corresponding "output cell". Movement within a cell is ignored; only when the pointing device moves from one cell to another is a position change reported to the host. There are two components in this Quantization Ratio: the Input Grid SPacing (I.GSP) and the Output Grid SPacing (O.GSP). The I.GSP specifies the cell size on the tablet surface, and the O.GSP specifies the cell size on the display surface.

Each GSP parameter has X and Y components, so the user may specify different aspect ratios for the input and output. This is useful for "correcting" the geometrical differences between different devices.

The position reported is the upper-left-hand corner of the output cell; it is an Offset in Two-Space.

This scaling algorithm may be used to eliminate the output-coordinate "jitter" so common in high-resolution graphics tablets. It can also facilitate many graphical design applications.

Example: I.GSP = 3 X 5
O.GSP = 5 X 5



4.4.7 Get Tablet Position Command -

This command requests the current position of the graphics tablet pointing device. The position is returned in a two-word field in the command packet. There are no input parameters; however, the command packet contains a two-word field in addition to the header. Normally, the reported position will be identical to that transmitted by the physical tablet device. However, if the tablet device is attached to the cursor (via the "Attach Cursor" command), then the position reported will conform to the position of the cursor (and thus will be constrained to the boundaries of the visible screen, etc.).

4.4.8 Set Pointing Device Event Reporting -

If the host wants to be notified (via an interrupt) when one or more pointing devices moves, it may enable "event-reporting" for the desired devices. If event reporting is enabled, the display will interrupt the host at most once every 1/60'th of a second for each device that has moved since the last interval. There is one parameter:

- o Enable Flag

4.4.8.1 ENABLE FLAG -

The enable flag parameter enables or disables movement event reporting for pointing devices. The flag is bit-encoded by device. A cleared bit disables and a set bit enables event reporting for the corresponding device. This parameter is encoded as follows:

- All bits 0 = no device
- bit 0 set = mouse
- bit 2 set = tablet

4.5 MISCELLANEOUS COMMANDS

4.5.1 Move Object Command -

The move object command is used to move arbitrary data from one part of the system to another. Some examples are:

- Loading fonts into display memory
- Sending control strings to the keyboard

The sections of the system involved are:

- VAX and Display Memory
- Peripherals (Keyboard, USART ports, etc.)

Parameters for the move object command are:

- Object Type
- Object Length
- Source
- Destination

4.5.1.1 OBJECT TYPE -

The object type parameter indicates what format the data is in and the source and destination types.

Type	Data Format	Source	Destination
0	reserved		
1	data words	memory	memory
2	character strings	memory	peripheral
3	character strings	peripheral	memory

4.5.1.2 OBJECT LENGTH -

The object length parameter indicates the length in bytes of the object to be moved.

4.5.1.3 SOURCE -

Depends on object type:

- Memory: Address of word-aligned and word padded buffer
- Peripheral: Device type value

4.5.1.4 DESTINATION -

Depends on object type:

- o Memory: Address of word-aligned and word padded buffer
- o Peripheral: Device type value

4.5.1.5 ERRORS -

Invalid Device Error: Reference to an invalid peripheral

device type or an illegal operation for a specific device.

4.5.1.6 NOTES -

4.5.1.6.1 DEVICE TYPES -

Device type values:

- 0 = no device
- 1 = mouse
- 2 = keyboard
- 3 = tablet
- 4 = USART - AUX 1
- 5 = USART - console
- 6,7 = reserved

4.5.1.6.2 BUFFERS -

Buffers must be word-aligned and word padded. Padding might be needed because all buffer must be an even number of bytes long and an extra byte might be needed at the end of the buffer. (The VS100 only supports word transfers.)

4.5.1.6.3 DATA FORMATS -

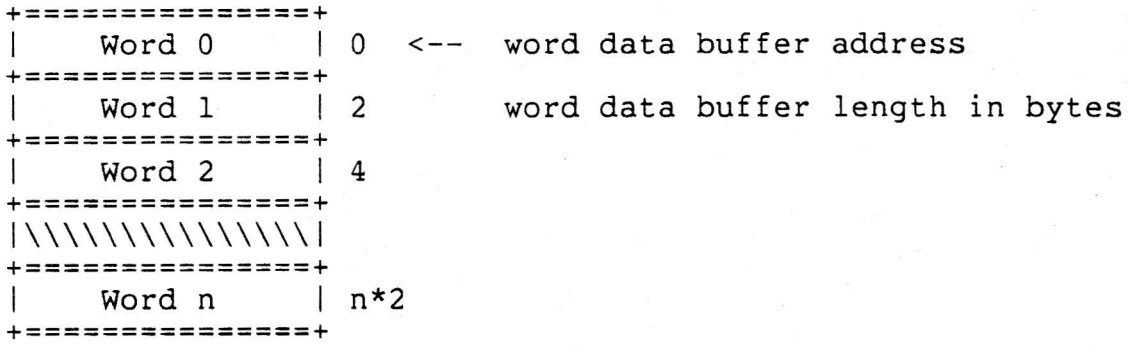
Data formats that Move Object supports:

- Word Data
- Character String

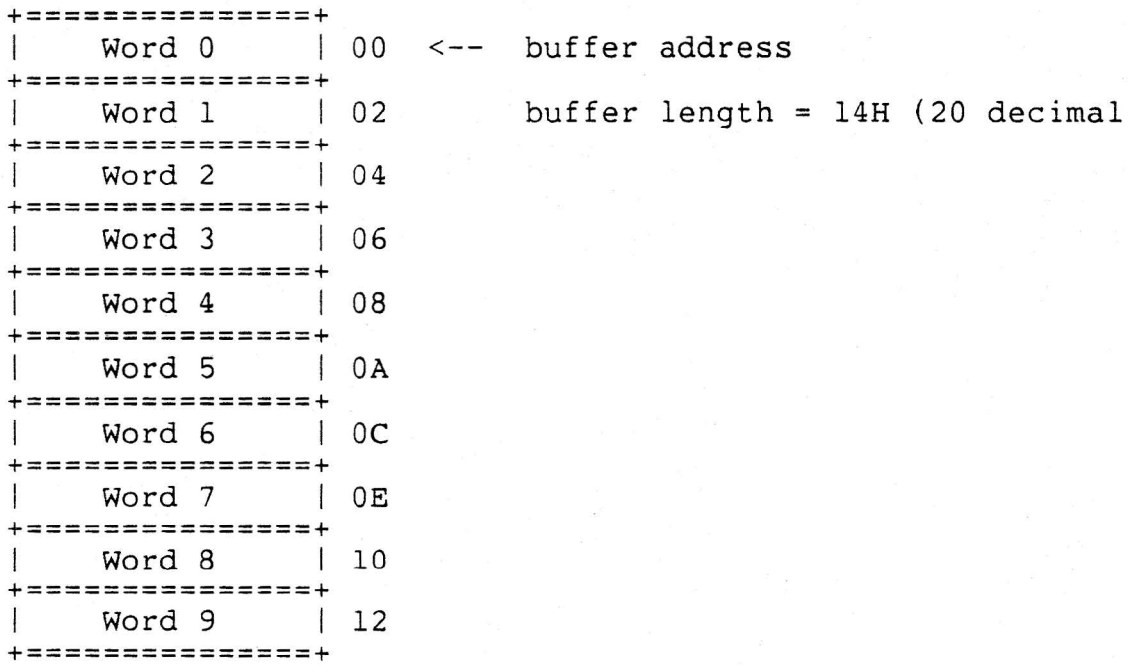
4.5.1.6.3.1 WORD DATA -

Format of Word Data Buffer

Object Type 1



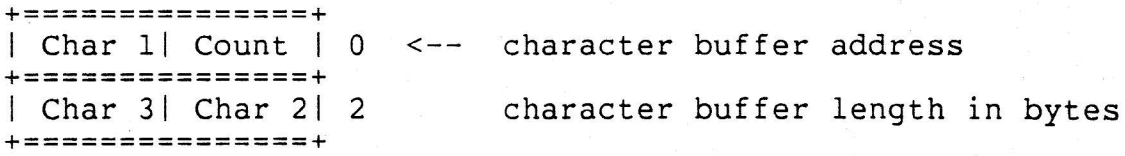
Example of a ten word buffer:



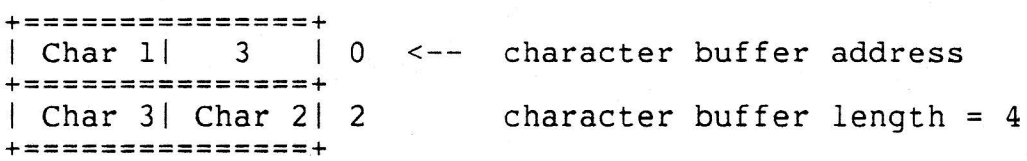
4.5.1.6.3.2 CHARACTER STRING -

Format of Character String Buffer

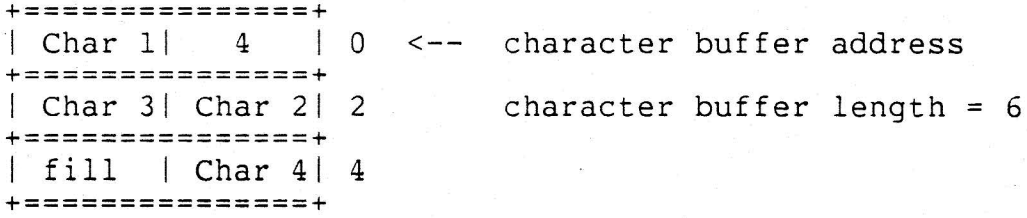
Object Type 2 and 3



Example of an odd number of characters:



Example of an even number of characters: (padding)



Sending Control Strings to the Keyboard from the Host Processor

The WGA Move Object Command will be used to send keyboard control strings to the LK201 keyboard. With this support the host will be able to control the keyboard; for example:

- Enable/Disable keyclicks
- Ring the bell
- Set the volume of the bell
- Control the LEDs

Example #1:

Ring the bell (A7H).

Move Object Parameters:

```
        Object Type  =  2  (memory to peripheral)
      Object Length  =  2  (length of character buffer)
             Source  =  Address of character buffer
        Destination  =  2  (Keyboard)
```

Character Buffer:

```
+================+
+   A7   |   1    + 0
+================+
```

Example #2:

Enable the bell (23H) and set the bell volume to 4 (84H).

Move Object Parameters:

```
        Object Type  =  2  (memory to peripheral)
      Object Length  =  4  (length of character buffer)
             Source  =  Address of character buffer
        Destination  =  2  (Keyboard)
```

Character Buffer:

```
+================+
+   23   |   2    + 0
+================+
+    0   |  84    + 2
+================+
```

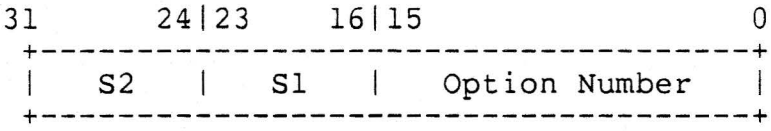
4.5.2 Report Status Command -

The Report Status command requests the return of information about the current status and configuration of the display processor. The information returned includes:

- o Device Type
- o Device Version
- o Micro-code Version
- o Visible Screen Frame Buffer Bitmap
- o Free Frame Buffer Memory and Byte Length
- o Free Program Memory Space Address and Byte Length
- o Host Memory Space Base Address and Byte Length

All addresses are in the physical address space of the display microprocessor. The host uses the base addresses reported by this command to map addresses in its space to addresses in the display's space. The returned parameters are described in more detail below.

4.5.2.1 DEVICE TYPE - A 32-bit value that indicates the type of device connected. This parameter is encoded as follows:



The low order word is a 16 bit binary option number and the high order two bytes are the option suffix encoded in ASCII (i.e. the "AA" in VS100-AA). Note that the suffix is encoded in standard VAX-11 address order (an even address points to the low-order byte).
Example:

VS100-AB	binary	decimal	hexadecimal
Option	0110 0100	100	64
ASCII "A"	0100 0001	65	41
ASCII "B"	0100 0010	66	42

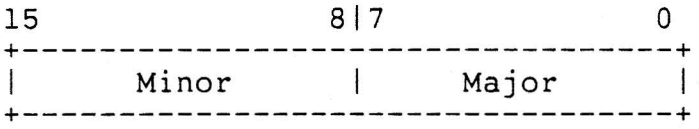
Device type is: 42410064 Hex

4.5.2.2 DEVICE VERSION -

A 16-bit binary number indicating what changes have been made to the display hardware (the revision level).

4.5.2.3 FIRMWARE VERSION -

Two (2) 8-bit binary values representing the major and minor version numbers of the firmware currently loaded in the display. Again, they are stored in standard address order.



When the Report Status command is processed by the ROM during intialization, this field is returned as zero.

4.5.2.4 VISIBLE SCREEN FRAME BUFFER BITMAP -

A bitmap specification indicating the address and size of the frame buffer bitmap from which the visible screen is refreshed.

4.5.2.5 FREE FRAME BUFFER MEMORY -

The base address and length of storage in the frame buffer memory not used by the visible screen. This memory is available for use by the host software.

4.5.2.6 FREE PROGRAM MEMORY SPACE -

The base address of additional memory in the microprocessor's program/data space that is available for use by the host software. When the ROM-based report status command is executed during intialization (see below, section on initialization and initial display requirements), the base address of microprocessor program/data space is returned. This space could be used for storing frequently used data. The size of this space will likely be much smaller than the off-screen framebuffer space.

4.5.2.7 HOST MEMORY SPACE BASE ADDRESS -

The base address, in the display microprocessor's space, of the memory space shared with the host. Host addresses passed to the display must be biased by this value. The display also returns this value to the host in CSRs during initialization.

4.5.3 No Operation Command -

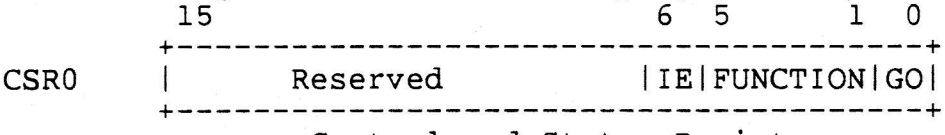
The no operation command simply causes the display to read the packet and respond with a command completion interrupt. There are no parameters.

5.0 CONTROL AND STATUS REGISTERS

The workstation graphics display communicates with the host via an interface that utilizes a set of control and status registers (CSRs). These CSRs are implemented as shared memory and not as hardware registers. Therefore, a strict protocol must be obeyed to avoid race conditions when using the CSRs. The CSRs and their programming are described in the following sections.

5.1 Control And Status Register (CSR0)

The control and status register is the main control register for transmitting commands to the display processor. Its format is:



Control and Status Register

where the fields have the following uses:

GO = CSR0<0> The GO bit is set by the host to indicate that the next command has been set up and the device should process the command.

FUNCTION = CSR0<5:1> The function field is set by the host to indicate the display command to execute, i.e., this is the command opcode. Functions 0 through 15 are implementation-independent definitions and 16 through 31 are implementation-dependent.

IE = CSR0<6> The interrupt enable bit, when set, allows the display to interrupt the host when a command is completed, a mouse event occurs, a keyboard event occurs, or an error occurs. There is only one interrupt vector; the host determines the event by scanning the interrupt reason flags. (Note: when the host initializes or reboots, it should clear IE until it is capable of receiving display interrupts.)

Use of CSR0 must follow a strict protocol. When GO is set to zero, the device is ready for a new command. Only the host can modify FUNCTION and GO when GO is zero. When GO is set, the device is processing a command. Only the device can modify FUNCTION once GO is set.

At initialization, the device sets GO and FUNCTION to zero to indicate that it is ready for processing. Host can then load FUNCTION and set GO to perform an operation. The device will clear both when it is done.

5.2 Interrupt Reason Register

The interrupt reason register is written by the display to indicate the event causing an interrupt. Interrupt reasons are bit-encoded; each bit indicates a different condition. Bit 15 of the interrupt reason register is an error bit. If set, the low-order 15 bits contain a display error code. Errors that also have bit 14 set are diagnostic errors.

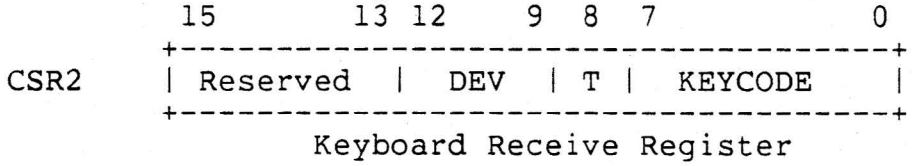
The format of the Interrupt Reason Register is:

Interrupt Reason Register

Like CSR0, use of CSR1 must be synchronized. When interrupt reason is zero, the device can complete a request. Only the device can write CSR1 when it is set to zero. When interrupt reason is nonzero, the host has not yet examined an event and the display may not write to CSR1. After the host has processed an interrupt, it sets CSR1 to zero. The device can then write the interrupt reason for the next event.

5.3 Device Event Register

The Device Event Register indicates events on keyboard or pointing device buttons attached to the display. The events are generally reported as up or down transitions on numbered keys. The format of the Device Event Register is:



Keyboard Receive Register

where:

KEYCODE=CSR2<7:0> This field is written by the display to indicate on which key a transition has occurred. The key codes are integers from 0 to 255. The meaning of these codes depends on the particular device involved.

T=CSR2<8> The transition indicator specifies whether an up transition (T=0) or a down transition (T=1) has occurred. Not all of the button-event devices connected to the display may operate in exactly the same fashion, though. Some keys on the keyboard, for example, may only report down-transitions, while the buttons on the mouse

report both up and down transitions. The user should refer to device-specific documentation for the particulars.

DEV=CSR2<12:9> The device field specifies the device responsible for generating the event. This field is encoded as follows:

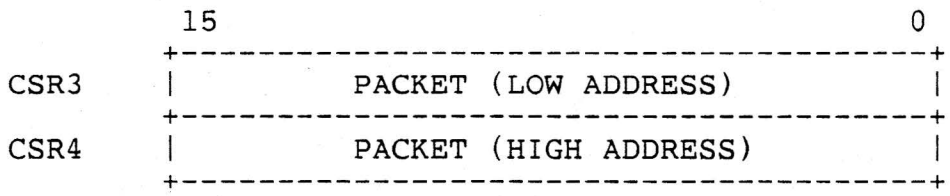
- 0 = no device
- 1 = mouse
- 2 = keyboard
- 3 = tablet
- 4 = USART - AUX #1
- 5 = USART - console

5.4 Function Parameter Register

The function parameter register consists of two 16-bit CSRs, CSR3 and CSR4. This 32-bit register pair is loaded with the address of a packet to be transmitted to the display.

When a command packet (or list of linked commands) is completed (or aborted), the number of packets successfully completed is written into this register pair.

During initialization, the device writes in CSR3/CSR4 the location at which host memory is mapped within its local address space. The host must then offset all host memory addresses by this amount when communicating with the display.



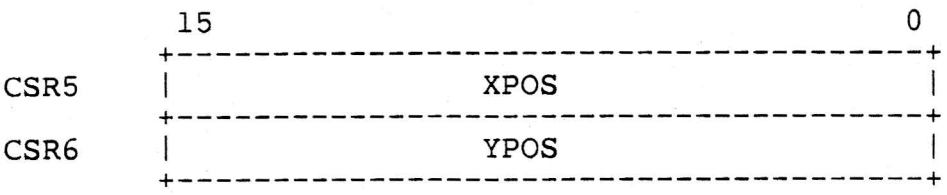
Function Parameter Register

5.5 Device Position Register

The Device Position Register contains the current position of the device whose movement generated the most recent interrupt. If the device is attached to the cursor, the interrupt reason will be "Cursor Moved"; the device position will be the same as the cursor position. If the device is not attached to the cursor, then the interrupt will indicate movement of that particular device. See also the sections on:

- Interrupt Reason Register
- Set Cursor Position
- Get Cursor Position
- Get Mouse Position
- Get Tablet Position
- Set Pointing Device Event Reporting

The format of the Device Position Register is:

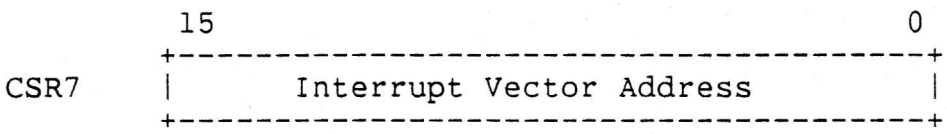


Current Device Position

The Device Position Register should only be examined immediately following a device event interrupt. These interrupts will occur at most once every 1/60'th of a second. The display will update the Device Position Register prior to interrupting. Because the update requires two non-interlocked 16-bit writes, it would be possible for the host to read a new XPOS with an old YPOS if the registers were read at random times. Therefore, when servicing a device interrupt, the host should copy the device position to a local storage area.

5.6 Interrupt Vector Address Register

The Interrupt Vector Address Register is set by the host to the address of an entry in the UNIBUS interrupt vector block which contains the PC and PSL of the device interrupt service routine. When the device wants to interrupt the host, these values are used to invoke interrupt servicing.



5.7 Initialization And Initial Display Requirements

The workstation graphics display must contain a ROM with code capable of processing a minimum of five immediate commands; that is, five function codes that can be loaded into the function field of CSR0. These commands are:

1. INIT (FUNCTION = 1). The INIT command causes the display micro-processor, if currently in operation, to halt execution and re-enter the initial ROM routine. An identical function

is performed on power-up of the display module. When the ROM is entered, it automatically stores the location at which host memory is mapped within the microprocessor's address space in the Function Parameter Registers, CSR3 and CSR4. When the host communicates a host memory address to the display, it must first add this value to the host address. This initial address is thus required by the host in order to transmit any host-mapped command packets to the display.

2. SEND PACKET (FUNCTION=2). The SEND PACKET command causes the display to read and process a command packet; the address of the command packet is loaded by the host into the Function Parameter Registers, CSR3 and CSR4, prior to setting FUNCTION CSR0 .

3. START DISPLAY (FUNCTION=3). The START DISPLAY command starts execution of the display micro-processor at the address specified in the Function Parameter Register (CSR3/CSR4). The start display command is executed during initialization to begin execution of display microcode. It causes the initial display ROM to invoke the firmware at the specified address.

   Start display is an immediate function and not a message. It is executed by loading the start display opcode into the CSR0 function field and setting the GO bit. The firmware responds with the "Started Executing" interrupt reason. The firmware start address is contained in the Function Parameter Register pair.

4. ABORT (FUNCTION=4). The ABORT function code stops any command being executed by the display. The display responds with the "command aborted" interrupt reason, and returns the number of successfully completed command packets in the Function Parameter Register.

5. EXECUTE_POWER_UP_SEQUENCE (FUNCTION=5). This function code causes the display to branch to its power-up sequence. This is a more "drastic" initialization than INIT. Some implementations may, for example, go in to a "self-test" mode here.

In addition, the initial display code must be capable of reading and processing two command packet opcodes as transmitted by a send packet request. These commands are:

1. Report Status - returns information about the display's status and addressing environment to the host,

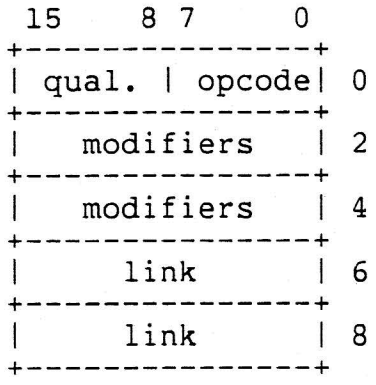
2. Move Object - allows down-line loading of display micro-processor code into display local memory.

5.8 Aborting A Request

Since some commands can take seconds or minutes to complete and commands can be linked together, it must be possible to abort the current command sequence. To abort an operation, the host loads the ABORT function code into the CSR0 function field. Note that this violates the rules for access to CSR0. It is possible that after the ABORT is written and before the function field is examined by the device, the command will complete and the device will write a zero in the CSR0 function code. Therefore, when processing an abort, the host driver must be capable of accepting two interrupt reasons: aborted and command completion. The driver must also be able to ignore an aborted interrupt reason when no command is in progress.

6.0 COMMAND PACKET FORMATS

This section describes the formats of the display command packets. Each command packet has two parts, a header and a command message. The header format is common among all commands and has the following format:

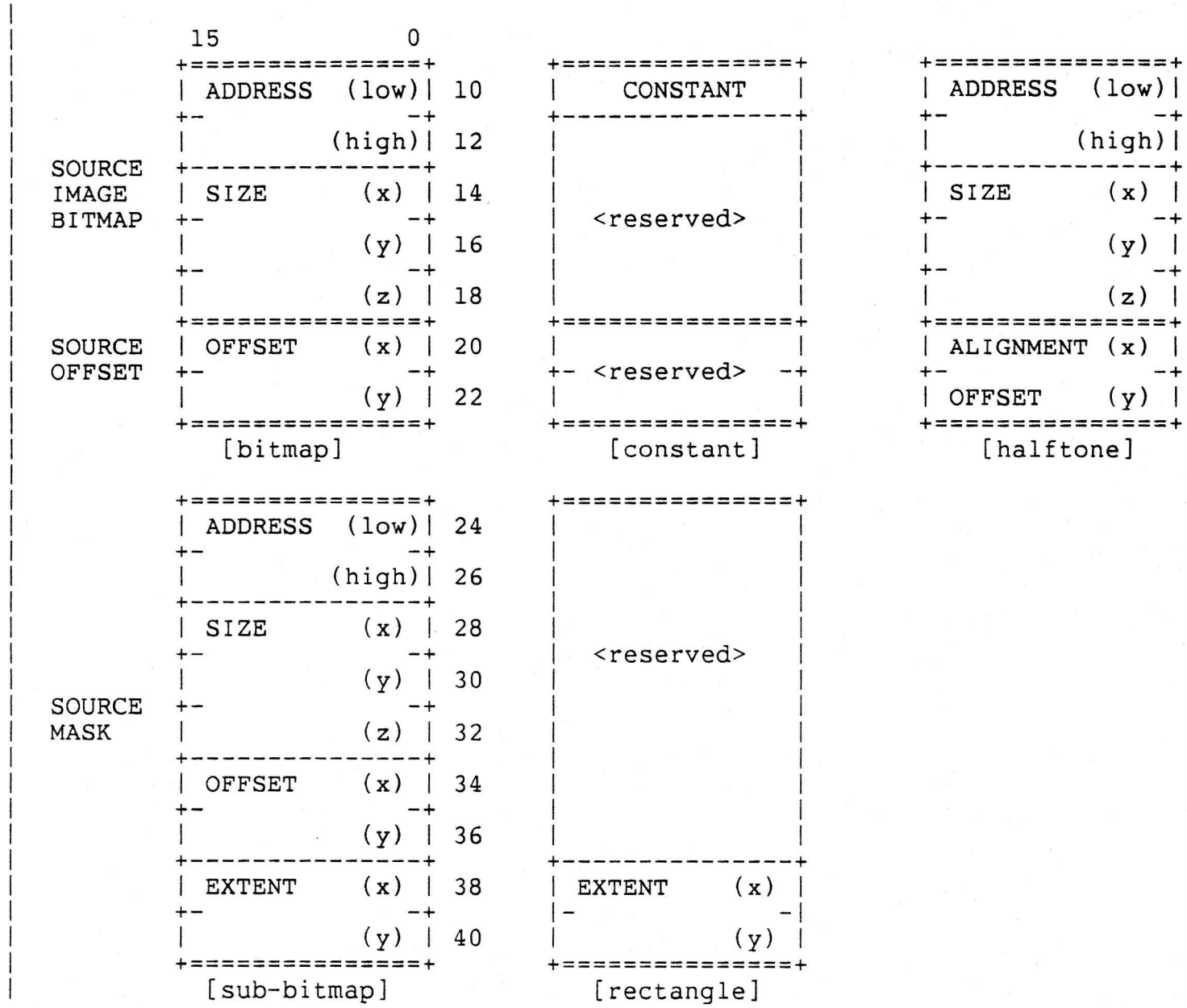


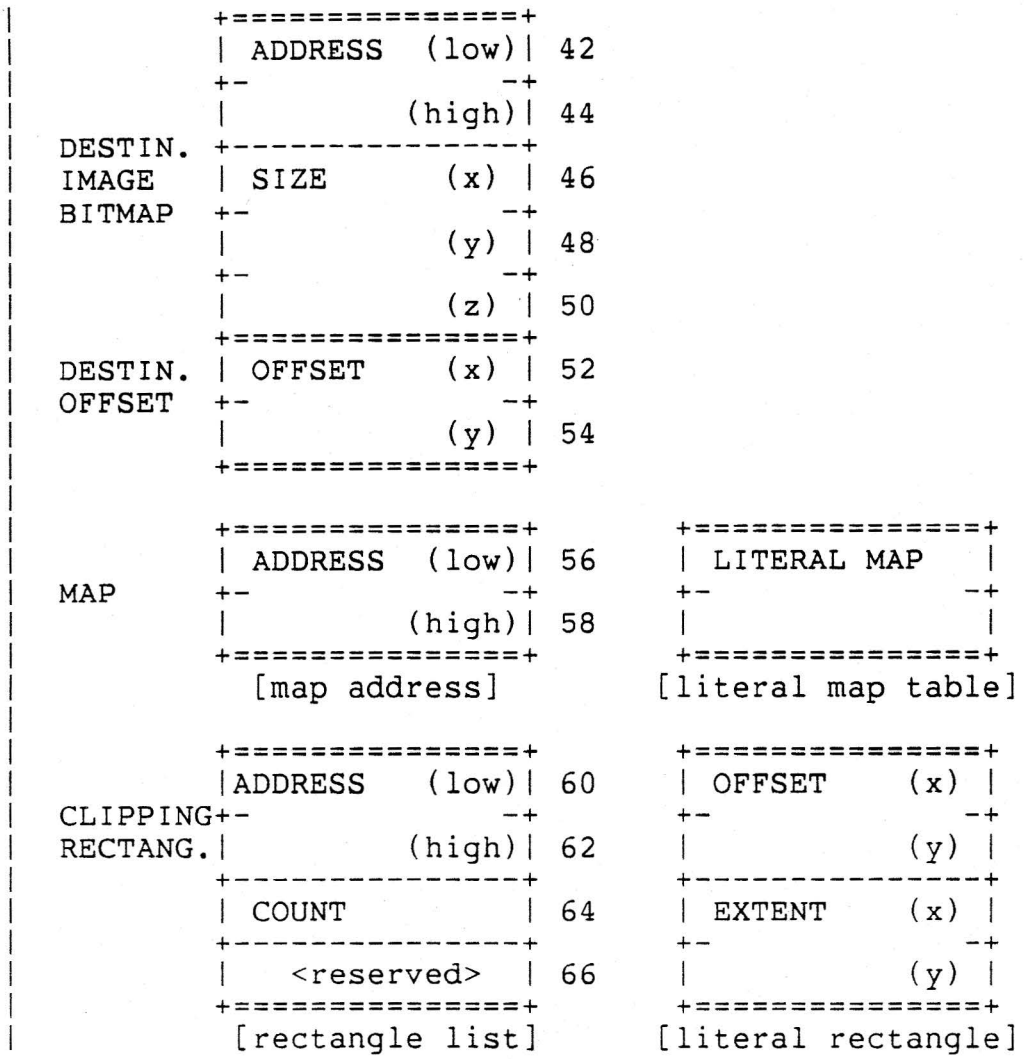
The opcode is an 8-bit value. Associated with the opcode is an 8-bit qualifiers field that contains command-independent qualifiers. In addition, many of the commands have command-dependent options in the specification of one or more parameters in the packet. For each parameter with specification options, the 32-bit opcode modifiers field indicates which option was chosen for that command. Although different options for a single parameter may require different size specifications, all packets for each opcode are of fixed size. Space is left in the packet for the largest size of each parameter offering several options. Thus, for each opcode, each parameter always starts at the same offset from the top of the packet, independent of the specification options chosen for the parameters.

The single command-independent qualifier currently defined is the Wait For Refresh qualifier, specified by bit zero of the 8-bit qualifiers field. If this bit is set to 1, the packet is processed as normal; however, physical screen operations are syncronized with the vertical blanking interrupt.

The link field is the 32-bit address of the next packet to be processed, if any. By linking packets, the host can initiate several commands with a single interaction. A zero value in the link field terminates the list. If any error occurs, the list is aborted at the point of error. The device returns the count of the number of packets successfully processed in the function parameter register, CSR3. For example, if the first packet is in error, a zero value will be returned, if the second is an error, a one will be returned, and so on.

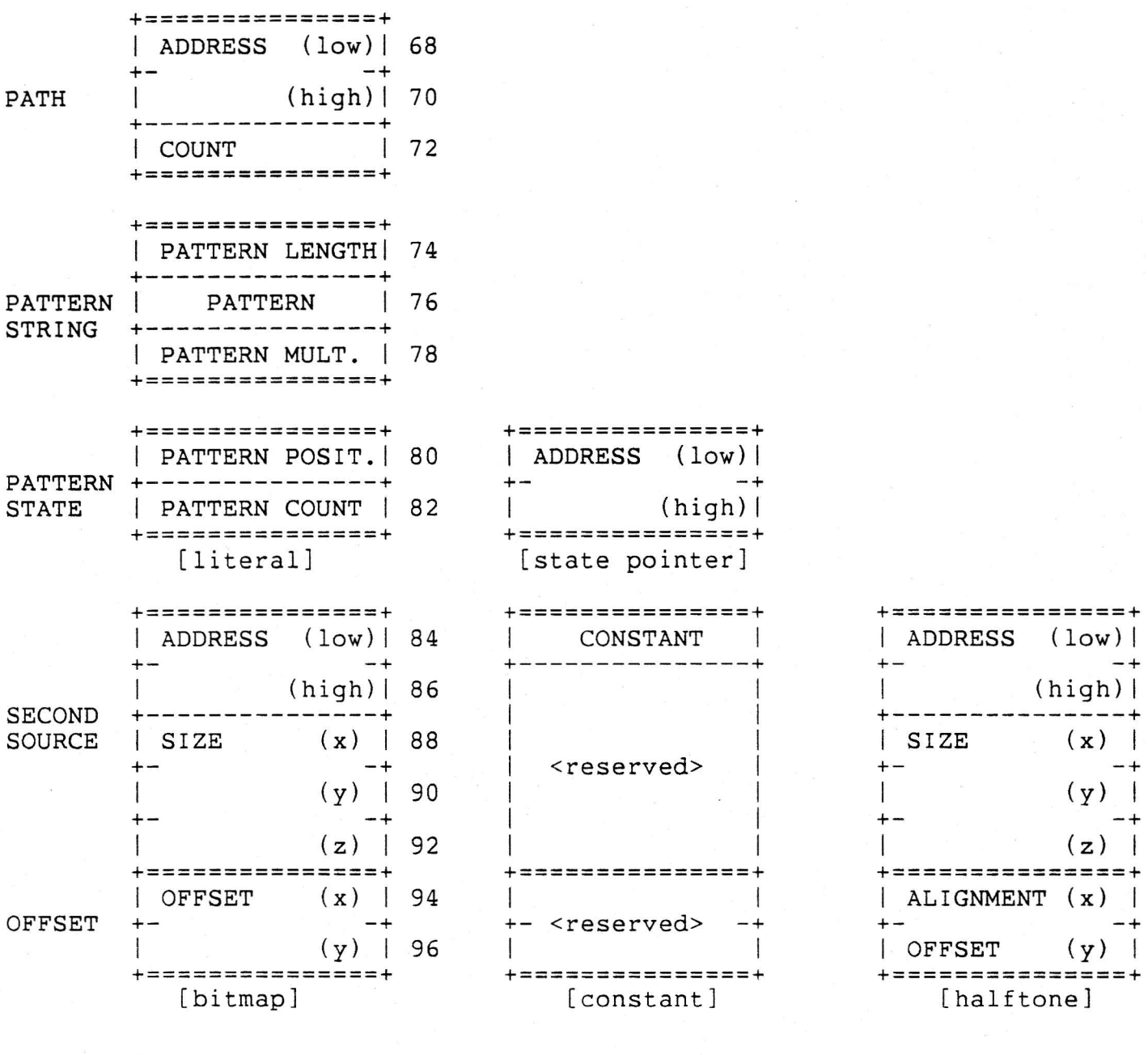
6.1 Copy Area Command Packet




ADDRESS (low)
(high)
DESTIN. IMAGE BITMAP
SIZE (x)
(y)
(z)
DESTIN. OFFSET
OFFSET (x)
(y)
ADDRESS (low)
(high)
MAP
[map address]
LITERAL MAP
[literal map table]
ADDRESS (low)
(high)
CLIPPING RECTANG.
COUNT
<reserved>
[rectangle list]
OFFSET (x)
(y)
EXTENT (x)
(y)
[literal rectangle]

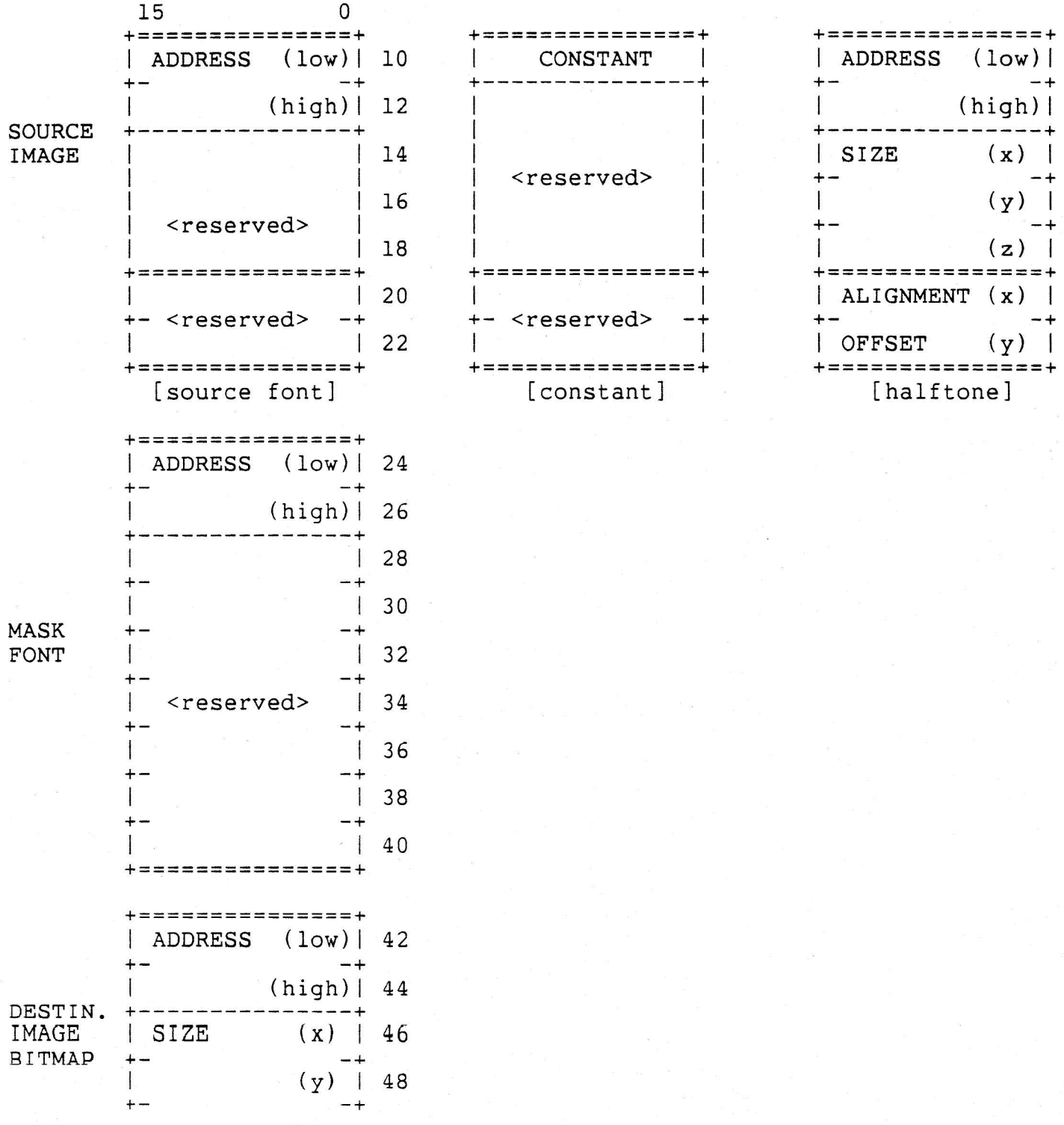
6.2 Draw Curve Command Packet

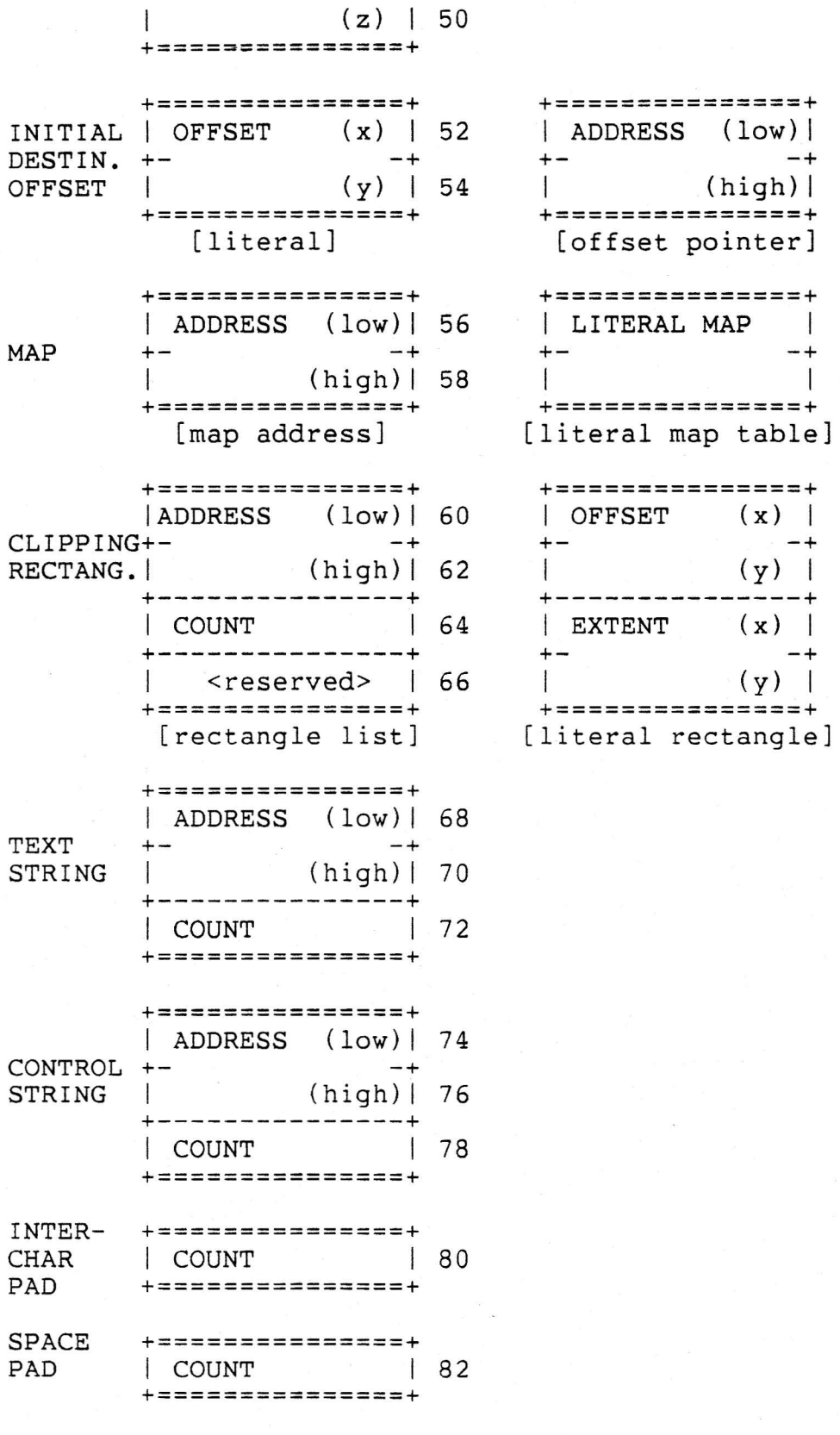
The draw curve command packet is identical to the copy area command packet with the following additional fields:



6.3 Print Text Command Packet

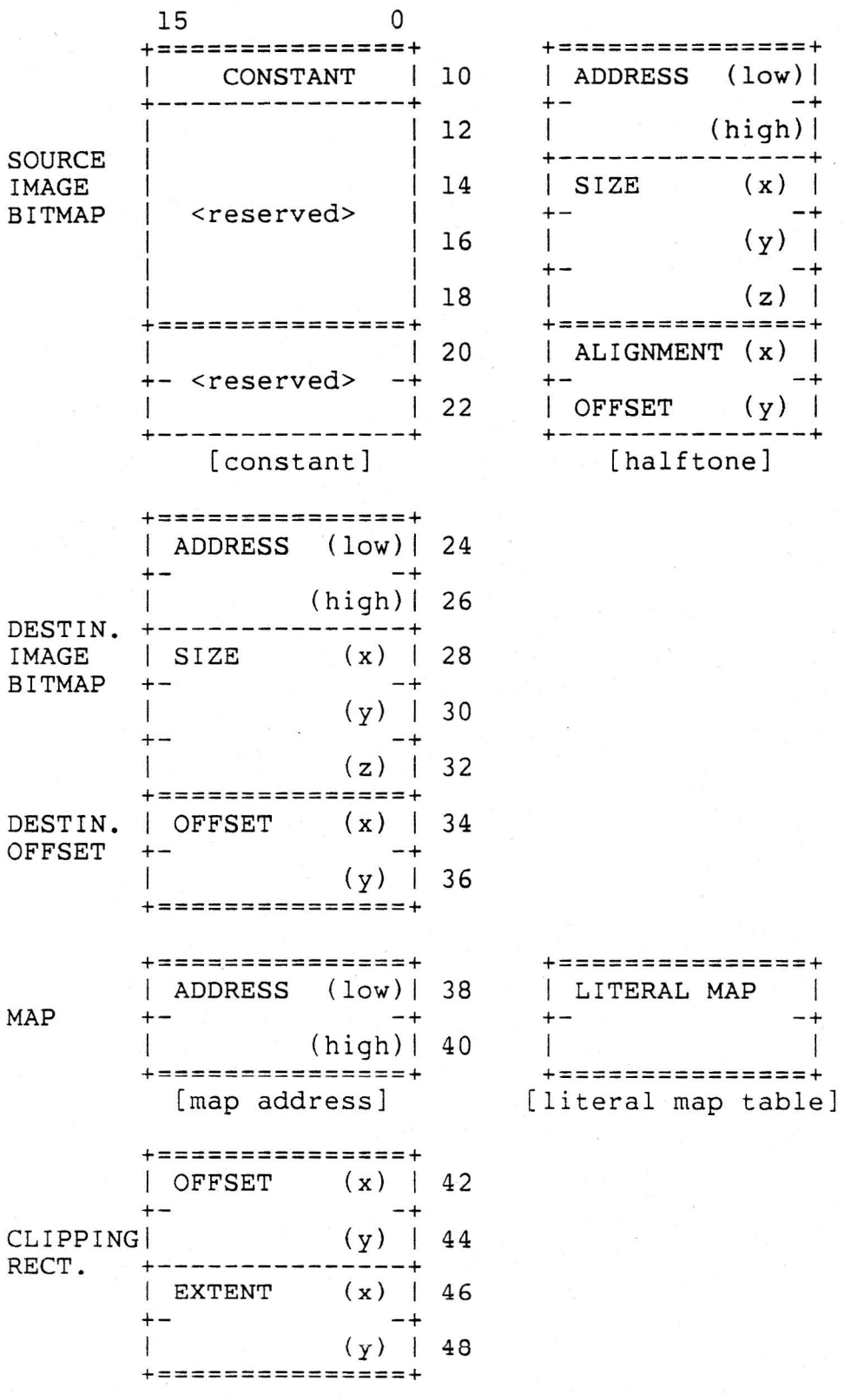
The print text command is similar to the copy area command, and has the following format. The differences are in specification of font (instead of source mask), the alternative specification of initial destination offset (instead of destination offset), and the additional parameters.

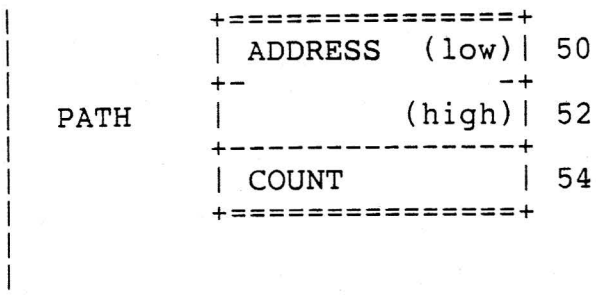



(z)
50
INITIAL DESTIN. OFFSET
OFFSET (x)
52
(y)
54
[literal]
ADDRESS (low)
(high)
[offset pointer]
MAP
ADDRESS (low)
56
(high)
58
[map address]
LITERAL MAP
[literal map table]
CLIPPING RECTANG.
ADDRESS (low)
60
(high)
62
COUNT
64
<reserved>
66
[rectangle list]
OFFSET (x)
(y)
EXTENT (x)
(y)
[literal rectangle]
TEXT STRING
ADDRESS (low)
68
(high)
70
COUNT
72
CONTROL STRING
ADDRESS (low)
74
(high)
76
COUNT
78
INTER-CHAR PAD
COUNT
80
SPACE PAD
COUNT
82

6.4 Fill Area Command Packet

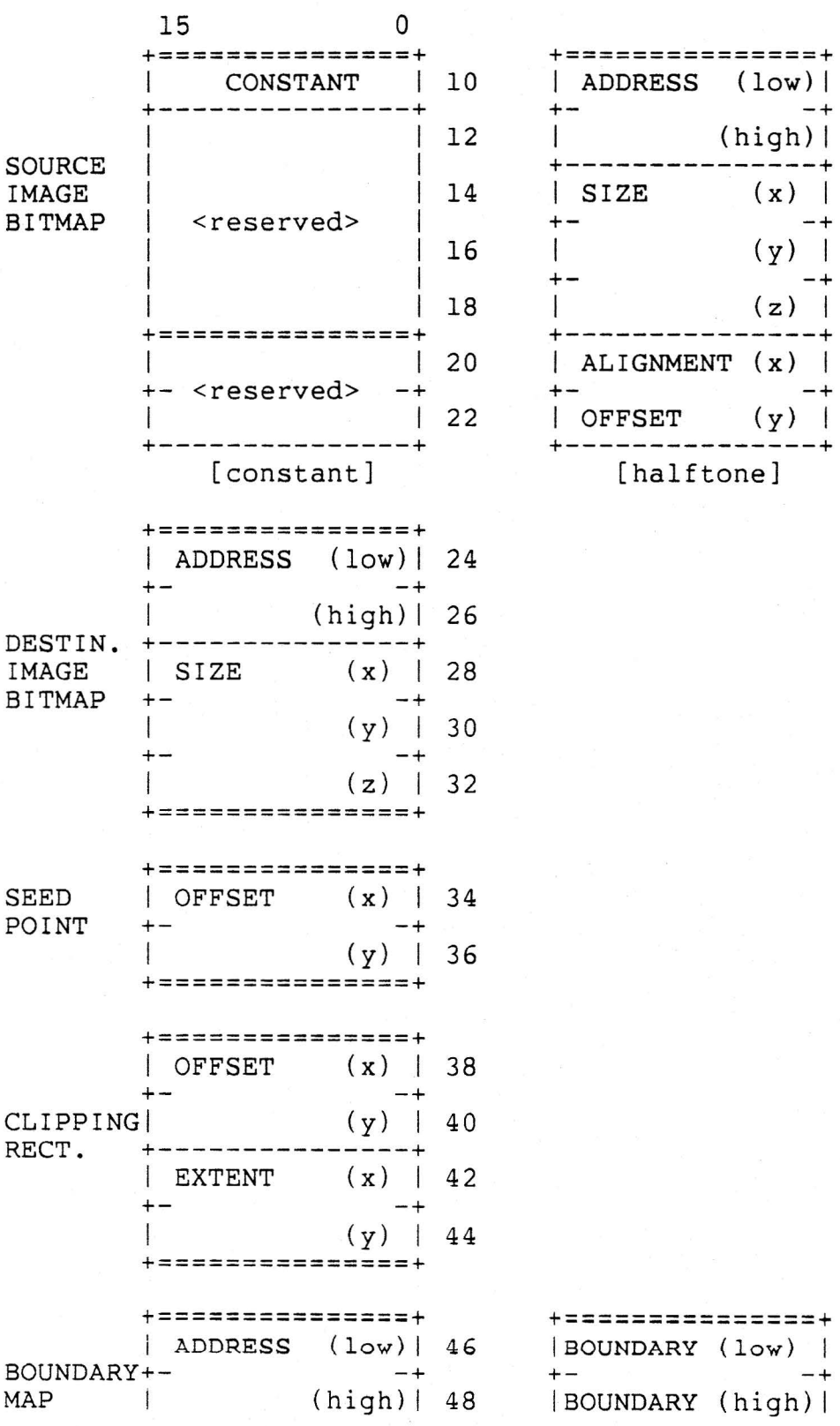
The command packet for the fill area command has the following format:

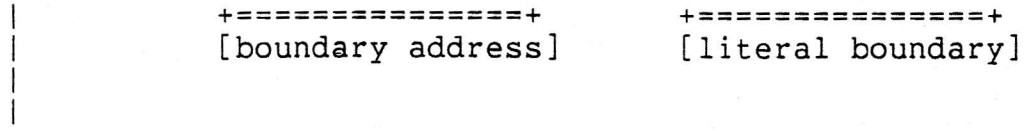



PATH
ADDRESS (low)
(high)
COUNT
50
52
54

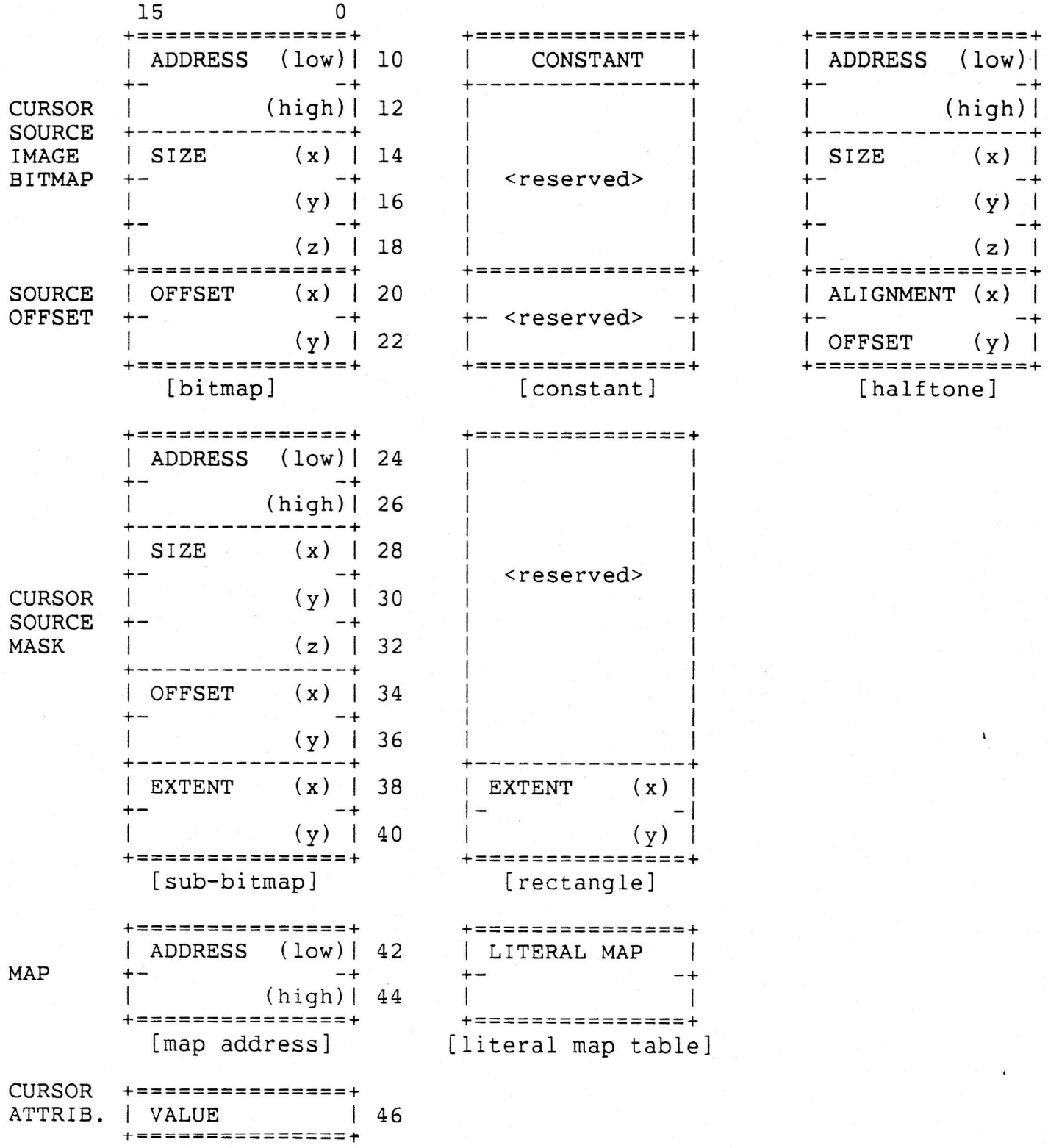
6.5 Flood Area Command Packet

The command packet for the flood area command has the following format:

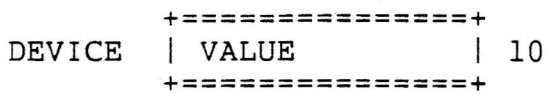



[boundary address]
[literal boundary]

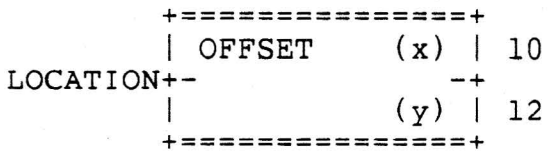
6.6 Load Cursor Command Packet



6.7 Attach Cursor Command Packet

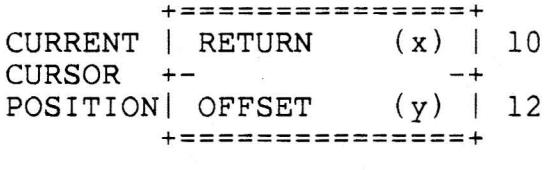


6.8 Set Cursor Position Command Packet



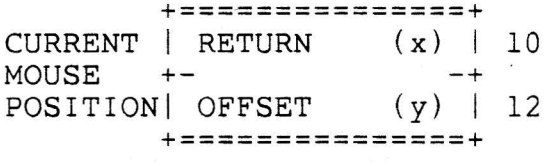
6.9 Get Cursor Position Command Packet

This command has a two-word field in which the display returns the current position of the cursor.

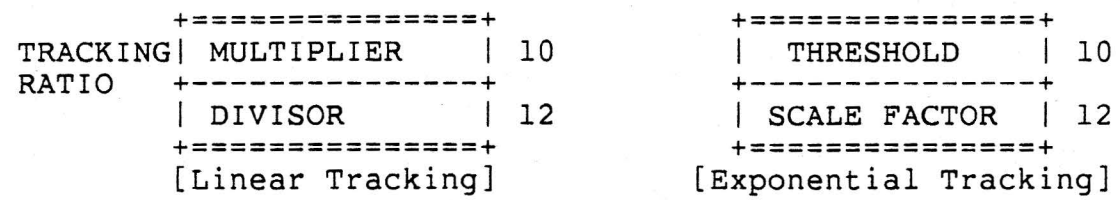


6.10 Get Mouse Position Command Packet

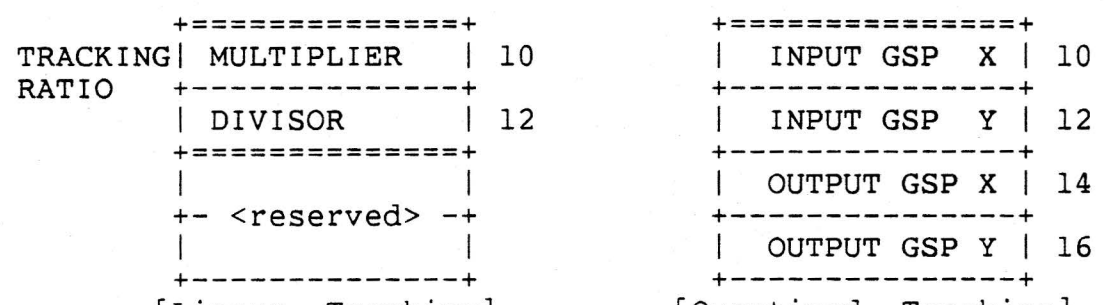
This command has a two-word field in which the display returns the current position of the mouse.



6.11 Set Mouse Characteristics Command Packet

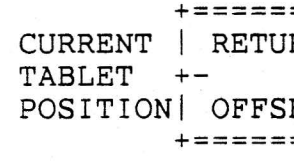


6.12 Set Tablet Characteristics Command Packet



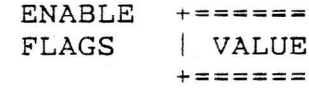
6.13 Get Tablet Position Command Packet

This command has a two-word field in which the display returns the current position of the tablet.

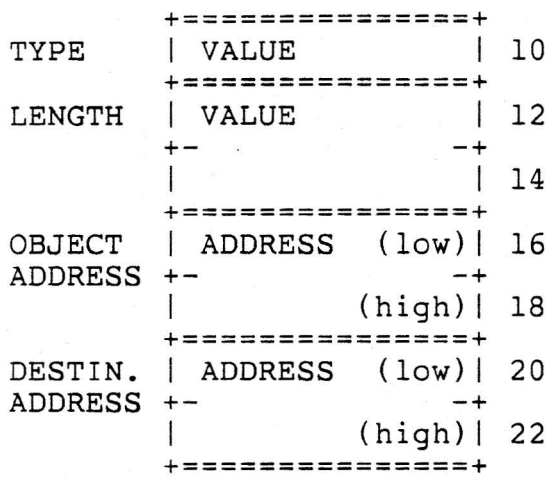


6.14 Set Pointing Device Event Reporting

This command enables or disables periodic reporting of mouse and/or tablet movement.

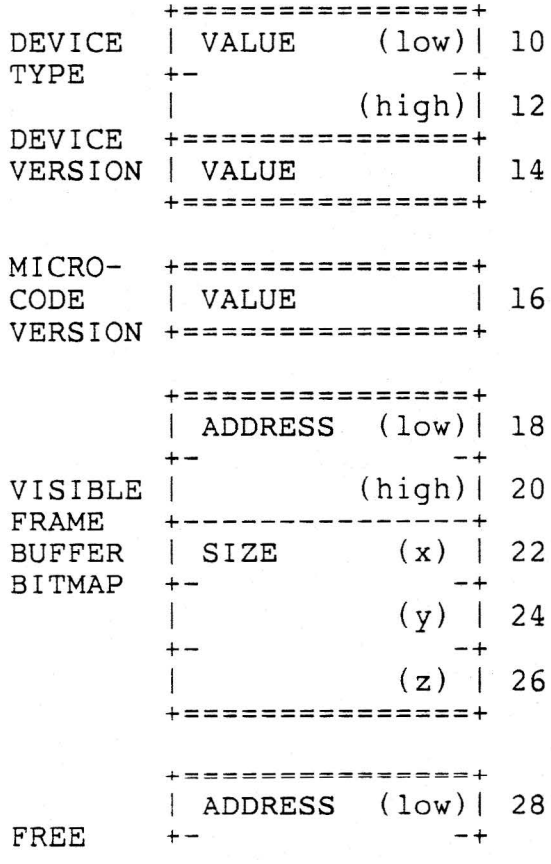


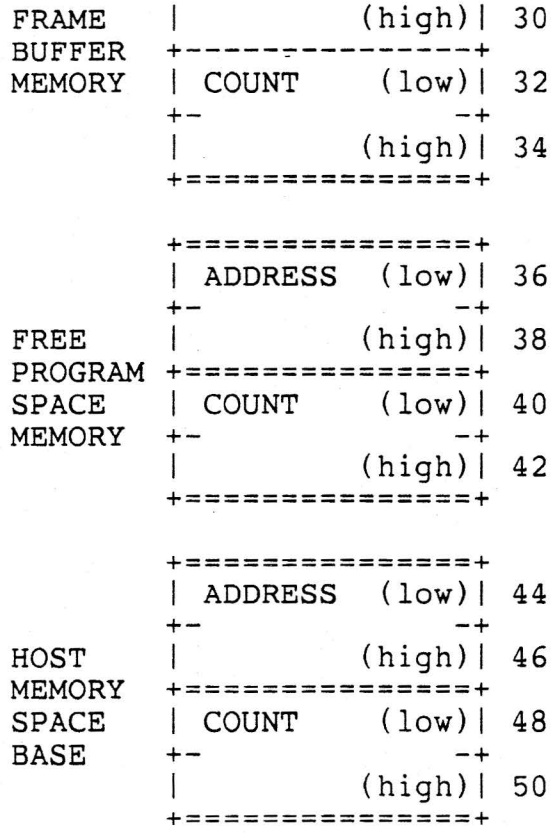
6.15 Move Object Command Packet



6.16 Report Status Command Packet

For the report status command, the packet contains no input parameters except for the opcode header. However, the packet contains space for the following information that is filled by the display.





6.17 No Operation Command Packet

This command has no parameters, and consists only of the packet opcode header.

7.0 CONSTANTS, OPCODES, MODIFIERS, AND ERROR CODES

This section specifies the constants used as opcodes, modifiers, and error codes for onyx commands and return codes.

7.1 Control And Status Register 0 Function Codes

The following function codes are defined for CSR0 functions.

Implementation-independent functions:

```
INIT = 1
SEND_PACKET = 2
START_DISPLAY = 3
ABORT = 4
EXECUTE_POWERUP_SEQUENCE = 5
```

VS100 Implementation-dependent functions:

```
BBA_ON = 16
BBA_OFF = 17
SET_INFINITE_RETRIES = 18
SET_FINITE_RETRIES = 19
```

7.2 Command Packet Operation Codes

The following are the values of the 8-bit operation codes specified in the low byte of the first word transmitted in a device command. These commands are all interperted by display microcode or firmware.

```
NO_OPERATION                = 0
COPY_AREA                   = 1
DRAW_CURVE                  = 2
PRINT_TEXT                  = 3
FLOOD_AREA                  = 4
LOAD_CURSOR                 = 5
SET_CURSOR_POSITION         = 6
ATTACH_CURSOR               = 7
GET_CURSOR_POSITION         = 8
MOVE_OBJECT                 = 9
REPORT_STATUS               = 10
FILL_AREA                   = 11
GET_MOUSE_POSITION          = 12
SET_MOUSE_CHARACTERISTICS   = 13
GET_TABLET_POSITION         = 14
SET_POINTING_DEV_REPORTING  = 15
SET_TABLET_CHARACTERISTICS  = 16
```

The following commands are interpreted only by display ROM, and are thus given different operation codes, even though there is some overlapping of the commands. This prohibits accidental interpretation of a command intended for a different machine state.

```
MOVE_OBJECT = 128
REPORT_STATUS = 129
```

7.3 Command Packet Operation Modifiers

The modifiers field specifies, for each command, which optional parameters are specified and what format is present for parameters with multiple formats. The following modifiers are defined below, listed separately for each command.

7.3.1 Copy Area Command Modifiers -

PARAMETER	FIELD	VALUE	MEANING
SOURCE	mod<2:0>	0	constant
		1	bitmap
		2	halftone
SOURCE MASK	mod<5:3>	0	rectangle
		1	bitmap
DEST. OFF.	mod<8:6>		(not used)
MAP	mod<11:9>	0	identity map
		1	source map address
		2	source map literal
		3	function code address
		4	function code literal
CLIPPING RECTANGLES	mod<14:12>	0	none
		1	literal rectangle
		2	rectangle list addr.

7.3.2 Draw Curve Command Modifiers -

PARAMETER	FIELD	VALUE	MEANING
SOURCE	mod<2:0>	0	constant
		1	bitmap
		2	halftone
SOURCE MASK	mod<5:3>	0	rectangle
		1	bitmap
DEST. OFF.	mod<8:6>		(not used)
MAP	mod<11:9>	0	identity map

		1	source map address
		2	source map literal
		3	function code address
		4	function code literal
CLIPPING RECTANGLES	mod<14:12>	0	none
		1	literal rectangle
		2	rectangle list addr.
DRAWING MODE	mod<15>	0	Solid Segment
		1	Dashed/Patterned Segment
PATTERN STATE	mod<17:16>	0	Literal pattern state
		1	Indirect pattern state
		2	Update literal pattern sta
		3	Update indirect pattern st
PATTERN MODE	mod<19:18>	0	No secondary source (Dashe
		1	Constant secondary source
		2	Bitmap secondary source
		3	Halftone secondary source

7.3.3 Print Text Command Modifiers -

PARAMETER	FIELD	VALUE	MEANING
SOURCE	mod<2:0>	0	constant
		1	source font
		2	halftone
MASK FONT	mod<5:3>	0	no mask
		1	mask font supplied
DEST. OFF.	mod<8:6>	0	dest offset literal
		1	dest offset indirect
		2	update dest literal
		3	update dest indirect
MAP	mod<11:9>	0	identity map
		1	source map address
		2	source map literal
		3	function code address
		4	function code literal
CLIPPING RECTANGLES	mod<14:12>	0	none
		1	literal rectangle
		2	rectangle list addr.
TEXT STRING	mod<15>	0	8 bit characters
		1	16 bit characters
CONTROL STRING	mod<16>	0	no control string
		1	control string

7.3.4 Fill Area Command Modifiers -

PARAMETER	FIELD	VALUE	MEANING
SOURCE	mod<2:0>	0	constant
		1	(not used)
		2	halftone
SOURCE MASK	mod<5:3>		(not used)
DEST. OFF.	mod<8:6>		(not used)
MAP	mod<11:9>	0	identity map
		1	source map address
		2	source map literal
		3	function code address
		4	function code literal
CLIPPING RECTANGLE	mod<14:12>	0	none
		1	literal rectangle

7.3.5 Flood Area Command Modifiers -

PARAMETER	FIELD	VALUE	MEANING
SOURCE	mod<2:0>	0	constant
		1	(not used)
		2	halftone
SOURCE MASK	mod<5:3>		(not used)
DEST. OFF.	mod<8:6>		(not used)
MAP	mod<11:9>		(not used)
CLIPPING RECTANGLE	mod<14:12>	0	none
		1	literal rectangle
BOUNDARY MAP	mod<15>	0	literal
		1	pointer

7.3.6 Load Cursor Command Modifiers -

PARAMETER	FIELD	VALUE	MEANING
SOURCE	mod<2:0>	0	constant
		1	bitmap
		2	halftone

SOURCE MASK	mod<5:3>	0 1	rectangle bitmap
DEST. OFF.	mod<8:6>		(not used)
MAP	mod<11:9>	0 1 2 3 4	identity map source map address source map literal function code address function code literal

7.3.7 Set Mouse Characteristics Command Modifiers -

PARAMETER	FIELD	VALUE	MEANING
TRACKING	mod<2:0>	0 1	Linear Exponential

7.3.8 Set Tablet Characteristics Command Modifiers -

PARAMETER	FIELD	VALUE	MEANING
TRACKING	mod<2:0>	0 1	Linear Quantized

7.4 Interrupt Reason Values

The following are the values returned by the display in the Interrupt Reason Register (CSR1) following display interrupt. Interrupt reasons (bit 15 = 0) are unary encoded. Errors (bit 15 = 1) are binary encoded.

```
*
* WGA Completion Codes (15 reserved)
*

INT_ID          equ     $0001           ; Initialisation Done
INT_CD          equ     $0002           ; Command Done
INT_SE          equ     $0004           ; Started Executing
INT_BE          equ     $0008           ; Button Event
INT_CM          equ     $0010           ; Cursor Moved
INT_TM          equ     $0020           ; Tablet Moved
INT_MM          equ     $0040           ; Mouse Moved
INT_PD          equ     $0080           ; Powerup Done

* These are the error messages generated by the
* VS100, VS125 and VS300. Some errors are specific to
* one device. The following key indicates the error code's
* status for each device:
*
*       *   IS generated by this device
*       -   IS NOT generated by this device
*       +   New, was not defined in previous versions
*       !   Changed, Redefined from previous versions
*
* The key contains two characters. The first indicates the status
* for the VS100/125, and the second indicates the status for the VS300.
* For example:
*
*       *-  Error code is generated by VS100/125 only
*       -*  Error code is generated by VS300 only
*       **  Both VS100/125 and VS300 generate this code
*       !-  VS100/125 only, code has new definition
*       -+  VS300 only, new error code added
*       ++  VS100/125 and VS300, new error code added


* WGA Hardware Error Codes (32 reserved)
*
* Error Mnemonic        Value           Key  Description

NO_ERROR        equ     0               ; -+  Normal Successful Completion
ERR_BASE        equ     $8000           ; **  Error-Encountered bit
ERR_NYI         equ     ERR_BASE+0      ; **  Not Yet Implemented
ERR_IFC         equ     ERR_BASE+1      ; **  Invalid Function Code
ERR_ICC         equ     ERR_BASE+2      ; **  Invalid Command Code
ERR_RN          equ     ERR_BASE+3      ; **  Bus Error: Non-Existant Memo
ERR_RO          equ     ERR_BASE+4      ; *-  Bus Error: Retry Overflow
ERR_LD          equ     ERR_BASE+5      ; *-  Bus Error: Link Down
ERR_BE          equ     ERR_BASE+6      ; *-  Bus Error: Unexplained
```

```
ERR_AE          equ     ERR_BASE+7      ; **  Address Error
ERR_SI          equ     ERR_BASE+8      ; **  Spurious Interrupt
ERR_II          equ     ERR_BASE+9      ; *-  Illegal Instruction
ERR_BN          equ     ERR_BASE+10     ; *-  BBA NXM (Non-Existant Memory
ERR_BNI         equ     ERR_BASE+11     ; *-  BBA Not Installed
ERR_KBO         equ     ERR_BASE+12     ; *-  Keyboard Buffer Overflow
ERR_TBO         equ     ERR_BASE+13     ; *-  Tablet Buffer Overflow
ERR_BBO         equ     ERR_BASE+14     ; *-  Button Buffer Overflow
ERR_ITP         equ     ERR_BASE+15     ; *-  Invalid Tablet Packet


* WGA Packet Error Codes (32 reserved)
*
* Error Mnemonic        Value           Key  Description

ERR_ISRCMB      equ     ERR_BASE+32     ; **  Invalid SRC Modifier Bits
ERR_ISRCBW      equ     ERR_BASE+33     ; **  Invalid SRC Bitmap Width
ERR_ISRCBH      equ     ERR_BASE+34     ; **  Invalid SRC Bitmap Height
ERR_ISRCC       equ     ERR_BASE+35     ; *-  Invalid SRC Constant
ERR_ISRCBD      equ     ERR_BASE+36     ; **  Invalid SRC Bitmap Depth
ERR_ISRCD       equ     ERR_BASE+37     ; -+  Invalid SRC Bitmap Dimension

ERR_IMSKMB      equ     ERR_BASE+38     ; **  Invalid MSK Modifier Bits
ERR_IMSKBW      equ     ERR_BASE+39     ; **  Invalid MSK Bitmap Width
ERR_IMSKBH      equ     ERR_BASE+40     ; **  Invalid MSK Bitmap Height
ERR_IMSKBD      equ     ERR_BASE+41     ; **  Invalid MSK Bitmap Depth

ERR_IDSTMB      equ     ERR_BASE+44     ; **  Invalid DST-Offset Modifier

ERR_IDSTBW      equ     ERR_BASE+45     ; **  Invalid DST Bitmap Width
ERR_IDSTBH      equ     ERR_BASE+46     ; **  Invalid DST Bitmap Height
ERR_IDSTBD      equ     ERR_BASE+47     ; **  Invalid DST Bitmap Depth

ERR_NOAREA      equ     ERR_BASE+48     ; -+  No Resultant Area

ERR_IMAPMB      equ     ERR_BASE+50     ; **  Invalid Map Modifier Bits
ERR_IMAPFC      equ     ERR_BASE+51     ; -+  Invalid Map Function Code
ERR_ZIMAP       equ     ERR_BASE+52     ; -+  Depth Incompatible with Map
ERR_ZCIMAP      equ     ERR_BASE+53     ; -+  Depth Combination Incompatib

ERR_ICLPMB      equ     ERR_BASE+54     ; **  Invalid ClipR Modifier Bits
ERR_ICLPRC      equ     ERR_BASE+55     ; **  Invalid ClipR Count

ERR_SMC_ITC     equ     ERR_BASE+56     ; +-  Invalid Tracking Ratio
ERR_ITC_MULT    equ     ERR_BASE+57     ; -!  Invalid Tracking Multiplier
ERR_ITC_DIV     equ     ERR_BASE+58     ; -+  Invalid Tracking Divisor

ERR_ICD         equ     ERR_BASE+59     ; **  Invalid Cursor Device
ERR_MO_IBC      equ     ERR_BASE+60     ; **  Invalid Byte Count
ERR_MO_IOT      equ     ERR_BASE+61     ; **  Invalid Object Type
ERR_MO_IDT      equ     ERR_BASE+62     ; **  Invalid Device Type
ERR_IPC         equ     ERR_BASE+63     ; **  Invalid Path Count

*
* WGA Draw_Curve Error Codes (16 reserved)
```

```
*
* Error Mnemonic          Value              Key  Description

ERR_DC_IPC       equ     ERR_BASE+64      ; **  Invalid Path Count
ERR_DC_IPSL      equ     ERR_BASE+65      ; **  Invalid Pattern Length
ERR_DC_IPSM      equ     ERR_BASE+66      ; **  Invalid Pattern Multiplier
ERR_DC_ICF       equ     ERR_BASE+67      ; **  Invalid Closed Figure
ERR_DC_IPSP      equ     ERR_BASE+68      ; **  Invalid Pattern Position
ERR_DC_IPSMB     equ     ERR_BASE+69      ; **  Invalid Pattern String Modif
ERR_DC_IPMMB     equ     ERR_BASE+70      ; **  Invalid Pattern Mode Modifie
ERR_DC_IPSC      equ     ERR_BASE+71      ; **  Invalid Pattern Count
ERR_DC_ISSRCBW   equ     ERR_BASE+72      ; **  Invalid Second SRC Bitmap Wi
ERR_DC_ISSRCBH   equ     ERR_BASE+73      ; **  Invalid Second SRC Bitmap He
ERR_DC_ISSRCBD   equ     ERR_BASE+74      ; **  Invalid Second SRC Bitmap De
ERR_DC_ISSRCC    equ     ERR_BASE+75      ; *-  Invalid Second SRC Constant
ERR_DC_IDPM      equ     ERR_BASE+76      ; ++  Incompatible Drawing/Pattern

*
* WGA Print_Text Error Codes (16 reserved)
*
* Error Mnemonic          Value              Key  Description

ERR_PT_ICSL      equ     ERR_BASE+80      ; **  Invalid Control String Lengt
ERR_PT_ICSO      equ     ERR_BASE+81      ; **  Invalid Control String Opcod
ERR_PT_ICSP      equ     ERR_BASE+82      ; **  Invalid Control String Param
ERR_PT_ITSL      equ     ERR_BASE+83      ; **  Invalid Text String Length
ERR_PT_ICI       equ     ERR_BASE+84      ; **  Invalid Character Index
ERR_PT_TSE       equ     ERR_BASE+85      ; **  Text String Exhausted
ERR_PT_NFP       equ     ERR_BASE+86      ; **  No Font Present
ERR_PT_ISRCFW    equ     ERR_BASE+87      ; **  Invalid SRC Font width
ERR_PT_ISRCFH    equ     ERR_BASE+88      ; **  Invalid SRC Font height
ERR_PT_ISRCFD    equ     ERR_BASE+89      ; **  Invalid SRC Font depth
ERR_PT_IMSKFW    equ     ERR_BASE+90      ; **  Invalid MSK Font width
ERR_PT_IMSKFH    equ     ERR_BASE+91      ; **  Invalid MSK Font height
ERR_PT_IMSKFD    equ     ERR_BASE+92      ; **  Invalid MSK Font depth
ERR_PT_CSMF      equ     ERR_BASE+93      ; +*  Conflicting SRC/MSK Fonts

*
* WGA Flood_Area Error Codes (16 reserved)
*
* Error Mnemonic          Value              Key  Description

ERR_FA_ISRCB     equ     ERR_BASE+96      ; **  Invalid SRC Bitmap
ERR_FA_SPIOB     equ     ERR_BASE+98      ; **  Seed Point is on boundary
ERR_FA_SO        equ     ERR_BASE+99      ; **  Stack Overflow
ERR_FA_IBMMB     equ     ERR_BASE+100     ; **  Invalid Boundary Map Modifie

*
* WGA Fill_Polygon Error Codes (16 reserved)
*
* Error Mnemonic          Value              Key  Description

ERR_FP_ISRCB     equ     ERR_BASE+112     ; **  Invalid SRC Bitmap
ERR_FP_SO        equ     ERR_BASE+113     ; **  Stack Overflow
ERR_FP_IPC       equ     ERR_BASE+114     ; **  Invalid Point Count
```

```
ERR_FP_ICF      equ     ERR_BASE+115    ; **  Invalid Closed Figure

*
* WGA Powerup Error Codes (32 reserved)
*
* Error Mnemonic        Value           Key  Description

ERR_PASS        equ     ERR_BASE+128    ; *-  Base for test numbers
ERR_68K         equ     ERR_BASE+129    ; *-  68000 CPU
ERR_RC          equ     ERR_BASE+130    ; *-  ROM Checksum
ERR_PR          equ     ERR_BASE+131    ; *-  Program RAM
ERR_CRT         equ     ERR_BASE+132    ; *-  CRTC Register
ERR_TU          equ     ERR_BASE+133    ; *-  Tablet USART
ERR_KU          equ     ERR_BASE+134    ; *-  Keyboard USART
ERR_FOE         equ     ERR_BASE+135    ; *-  FOTR Electrical Loop Back
ERR_VTO         equ     ERR_BASE+136    ; *-  Vsync Time Out
ERR_SB          equ     ERR_BASE+137    ; *-  Screen Buffer
ERR_BS          equ     ERR_BASE+138    ; *-  BBA Scratchpad RAM
ERR_BC          equ     ERR_BASE+139    ; *-  BBA Copyarea Command
ERR_TTO         equ     ERR_BASE+140    ; *-  Tablet Time Out
ERR_FOO         equ     ERR_BASE+141    ; *-  FOTR Optical Loop Back
ERR_KTO         equ     ERR_BASE+142    ; *-  Keyboard Time Out
ERR_KST         equ     ERR_BASE+143    ; *-  Keyboard Self-Test

*
* WGA Load Cursor Error Codes (16 reserved)
*
* Error Mnemonic        Value           Key  Description

ERR_LDC_IATRV   equ     ERR_BASE+160    ; ++  Invalid Cursor Attribute Val
ERR_LDC_ICH     equ     ERR_BASE+161    ; ++  Invalid Cursor Height
ERR_LDC_ICW     equ     ERR_BASE+162    ; ++  Invalid Cursor Width
ERR_NOVALCUR    equ     ERR_BASE+163    ; ++  No Valid Cursor Defined
```

8.0 VAXSTATION 100 RESTRICTIONS

The following architectural restrictions apply to the VAXstation 100 implementation of the Workstation graphics architecture.

8.1 Number Of Planes

Since the VS100 has only one bit plane of framebuffer memory, the Z parameter may only have the value 1.

8.2 Halftone Representation

The VAXstation 100 uses only a single format for a halftone bitmap. The halftone pattern must be specified as a square bitmap 16 pixels on a side. Therefore, to use a "standard" four-by-four halftone pattern, the pattern is simply replicated horizontally and vertically to form a 16x16 pattern.

9.0 GENERAL IMPLEMENTATION RESTRICTIONS

The following restrictions apply to all VAX/UNIBUS implementations of the Workstation graphics architecture. Hence, the term "display" is used instead of "VS100".

9.1 Word Access I/O

All 16-bit word parameters must be word-aligned. Additionally, the display may only access host memory by words. Any byte strings of odd length, then, should be padded with an extra byte so that no "undefined" data is accessed.

9.2 UNIBUS Window Mapping

It is often the case that the display is asked to perform an operation between a source rectangle and a destination rectangle which overlap. That is, both source and destination bitmaps occupy the same (or nearly the same) area of memory. In this case, the display firmware must determine the proper memory-copy direction, so that no data is overwritten. It does this by comparing the base addresses of the two memory blocks in question. For this reason, it is important that the driver software (responsible for mapping UNIBUS memory to the display) maintain a one-to-one correspondence between areas of memory on each side of the UBW. Specifically, an area of VAX memory must never be made to appear to the display as two different areas of memory.

9.3 Bitmap Storage Requirements

The display represents a bitmap as a sequence of horizontal scan lines stored in contiguous memory locations. Each scan line must begin on a 16-bit word boundary. That is, although a bitmap can have any horizontal width in pixels, the storage in which the bitmap is kept must have sufficient space so that each horizontal line can be word-aligned. If the horizontal width in pixels is not evenly divisible by 16, the last bits in the last word of the storage for each horizontal line will be unused. For any bitmap of dimensions (X,Y) on the display, the storage requirement is ((X+15)/16)Y words.

9.4 Device Coordinate Management For WGA

The Device Position Registers, defined by the WGA, are used to hold the current XY position of the cursor. They are also used to report the current XY position of any pointing devices, e.g. mouse and/or tablet, which may be attached to the VAXstation.

The WGA also defines a phenomenon known as Event Reporting. If reporting is enabled for a particular device, it will interrupt the host at a maximum rate of 60 Hz. It places its XY position in the Cursor Position Registers and issues a "'device' moved" interrupt reason.

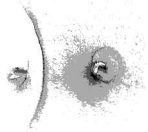
The following paragraphs enumerate the possible Event-Reporting situations, and how they are implemented.

[1] State: No Device Attached to the Cursor

[1a] Mouse Event Reporting set

On the Vsync interrupt, a service routine within the VS100 will read the current mouse XY position. If it has changed from the last time, it will be placed in the Device Position Registers. An interrupt is then sent to the host indicating "MOUSE MOVED". If the mouse has not moved, nothing happens. Note that since the mouse is not connected to anything here, its position is not confined to the visible screen. Its coordinates may vary from 32767 to -32768.

[1b] Tablet Event Reporting set



On the Vsync interrupt, a service routine will read the current tablet XY position. If it has changed from the last time, it will be placed in the Device Position Registers. It then issues an interrupt to the host indicating "TABLET MOVED". If the tablet has not moved, nothing happens. Note that the tablet position is not clipped to the visible screen boundaries unless it's attached to the cursor.

[1c] Both Mouse and Tablet Event Reporting set

In this case, both case [1a] and [1b] happen concurrently. However, only one device will be reported on any one Vsync. The devices will be polled in a round-robin-like fashion. On one Vsync, device (i) will be checked FIRST. If no action, device (i+1) is checked, and so on, until an 'active' one is found (or until there are no more devices). On the next Vsync, device (i+1) will be the first one checked, and so on.

This method has the following properties:

- o No more than 60 interrupts per second are sent to no matter how many devices are reporting.
- o No device will be 'locked out' by a higher-priorit or a more-rapidly-interrupting device.
- o Each individual device may still interrupt at 60Hz it has no 'competition'.
- o The worst case for device acknowledgement is 60/N N reporting devices.

[2] State: Mouse is Attached to the Cursor

[2a] Mouse Event Reporting set

All happens as in [1a], except that now the interrupt reason will be "CURSOR MOVED" instead of "MOUSE MOVED". Of course, since the mouse and cursor are attached, their XY positions will

always be the same.

This implies, of course, that the mouse XY position will now be clipped to the visible screen.

[2b] Tablet Event Reporting set

All happens as in case [1c], except that cases [2a] and [1b] are happening concurrently, instead of cases [1a] and [1b].

[2c] Both Mouse and Tablet Event Reporting set

This is the same as case [2b] above.

[3] State: Tablet is Attached to the Cursor

[3a] Mouse Event Reporting set

All happens as in case [1c], except that cases [1a] and [3b] are happening concurrently, instead of cases [1a] and [1b].

[3b] Tablet Event Reporting set

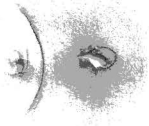
All happens as in [1b], except that now the interrupt reason will be "CURSOR MOVED" instead of "TABLET MOVED".

Since the tablet and cursor are attached, their XY positions will always be the same as long as they are within the visible screen.

Should the tablet position stray from the visible screen, both the tablet and the cursor XY position will be clipped to the screen's boundaries.

[3c] Both Mouse and Tablet Event Reporting set

This is the same as case [3a] above.



9.5 Keyboard Interface

If the input data is a keycode, a transition bit is generated, the keyboard device code added, and the result is returned, where it will become a button event to the VAXen host. If the key is in Autorepeating/Down-Only mode, no up-transition events will be generated. An autorepeating key will thus appear as a succession of down-events. If the key is in Up/Down mode, both up and down transitions will be generated.

If the input data is an error message, or an acknowledgement, it is ignored by the VS100.

There are a few codes which are treated specially:

- The Metronome character, for example, is not passed to the host, but is used to repeat the most recently depressed character.

- The state of the control-key code is monitored. When the 'control' key is depressed, the firmware will automaticall issue a 'temporary autorepeat inhibit' to the keyboard to keep control keys from autorepeating.

- The 'Allups' keycode will generate up-transitions for all up/down-mode keys which are down at the time.

- Other special codes not related to actual keys will be discard

  Since the user is not allowed to change the mode of the keyboard divisions, Prefix-To-Keys-Down should never occur.

The keyboard divisions are initialized
According to the following defaults:

Division	Mode
1. Main array	Autorepeat down
2. Numeric keypad	Autorepeat down
3. Delete character	Autorepeat down
4. Return and Tab characters	Down only
5. Lock, Compose, and A10	Up down
6. Shift keys and Control key	Up down
7. Horizontal arrow keys	Autorepeat down
8. Vertical arrow keys	Autorepeat down
9. Edit keypad keys	Autorepeat down
10. Local function keys (G99-G04)	Autorepeat down
11. Second function key set (G05-G09)	Autorepeat down
12. Third function key set (G10-G14)	Autorepeat down
13. HELP and MENU keys (G15-G16)	Down only
14. Fifth function key set (G20-G23)	Autorepeat down

In Up/Down mode, both up- and down-transitions are reported.
In Autorepeat and Down-Only mode, only down-transitions are reported.