RDS 3000 USER'S GUIDE

April 1982

Ikonas Graphics System, Inc.
a subsidiary of
Adage, Inc.

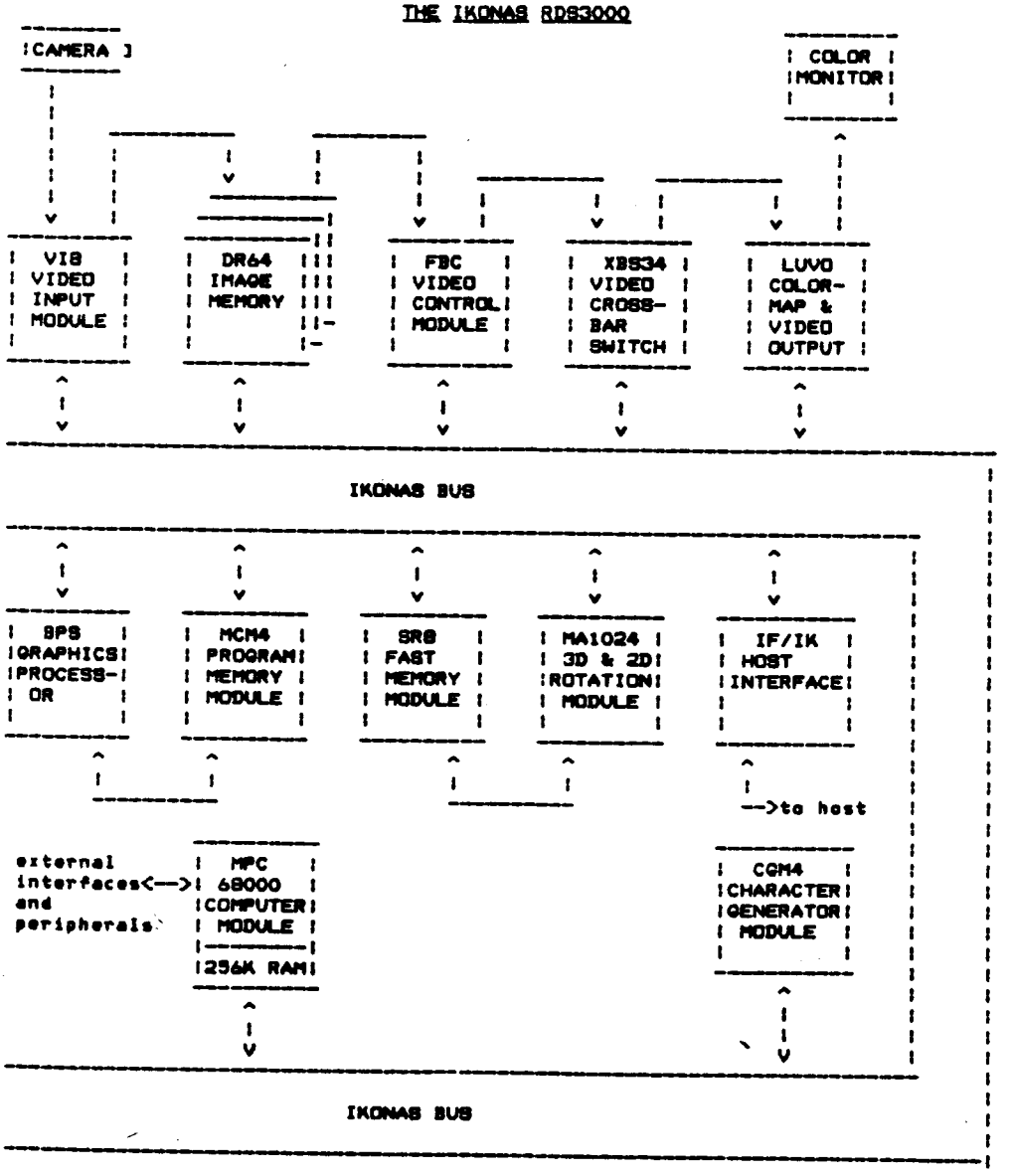
The information contained herein is subject to change without notice.

IKONAS GRAPHICS SYSTEMS, INC., makes no warranty of any kind, express or implied, with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IKONAS shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

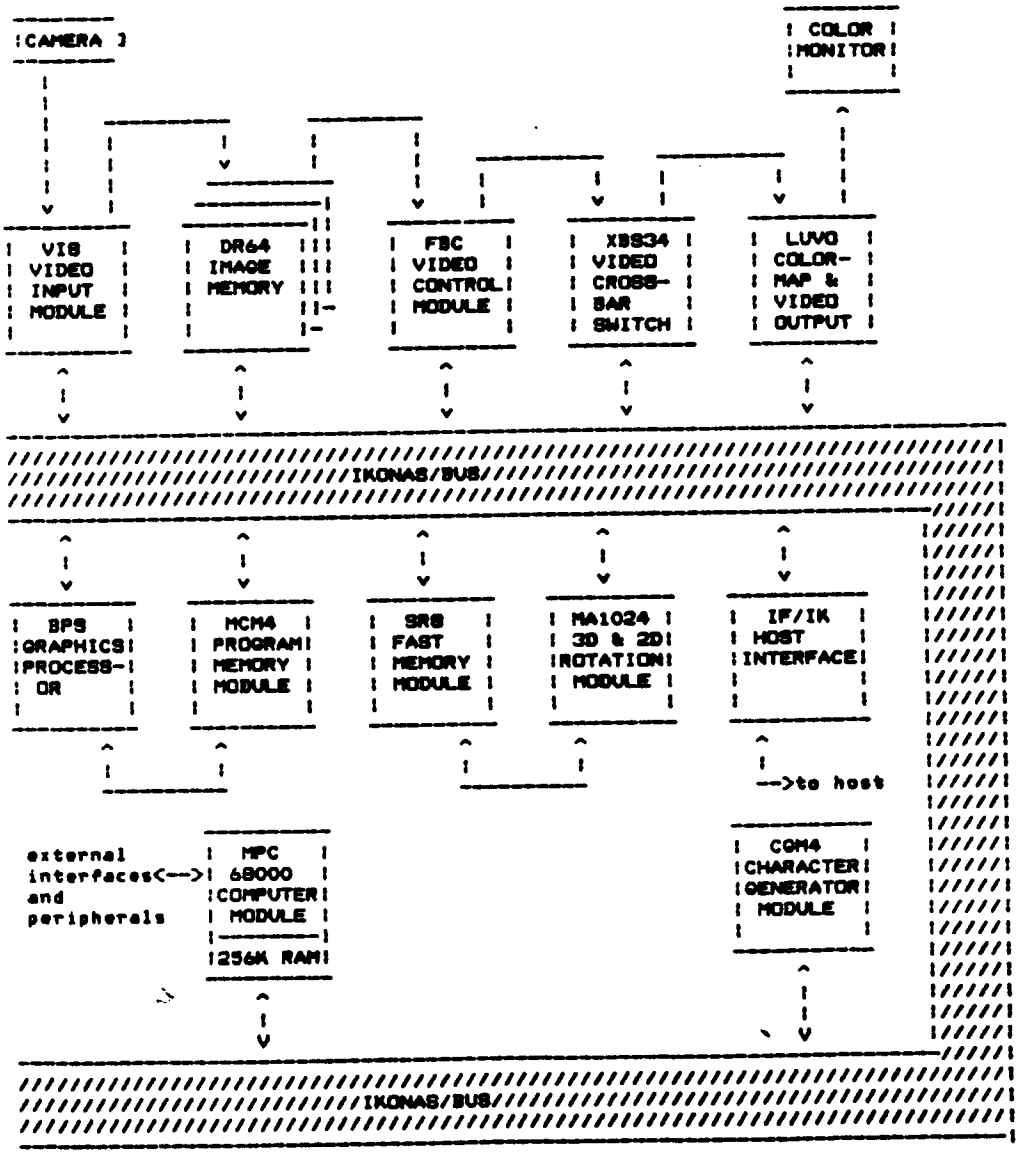
Part Number 10-301-095-10A

TABLE OF CONTENTS

THE IKONAS RDS3000



THE IKONAS BUS



THE IKONAS BUS

The IKONAS BUS connects all elements of the RDS3000 system.

The IKONAS bus:

- o is synchronous
- o starts a bus cycle every 200 ns
- o checks for a new bus master every cycle

Each IKONAS bus cycle transfers the following information:

- o 24 bits of address
- o 5 bits of function code
- o 32 bits of data

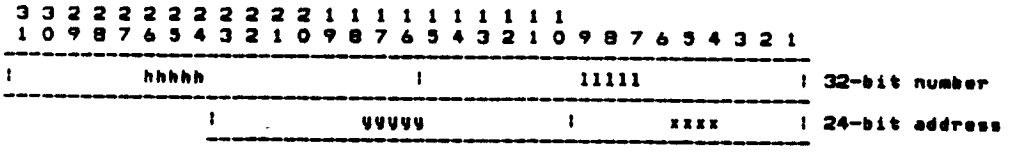
IKONAS Conventions

The IKONAS is a 32-bit machine, so we have some conventions for representing 24-bit addresses and 32-bit data items:

- yyyyy$xxxx to represent a 24-bit number whose upper 14 bits are yyyyy and whose lower 10 bits are xxxx
- hhhhh\11111 to represent a 32-bit number whose upper 16 bits are hhhhh and whose lower 16 bits are 11111

All numbers are octal. Examples:

20200$0, 100$100, 202\0



IKONAS Bus Addressing

The following rules apply to IKONAS addresses in general:

- The lower half of memory (bit 23=0) is for DR64s - image memory.
- The upper eight of memory (bit 21-23=111) is for MPC RAM and I/O.
- Everything in between is for IKONAS modules and fast RAM.
- The upper eight bits of address are used to distinguish between IKONAS modules

IKONAS MEMORY MAP

377000:1777	reserved for multiple-MPC systems	
341000		
340777:1777	MPC	
340400	(upper 16 bits unused by MPC)	I/O space
340377:1777		
340000		RAM
337000:1777	reserved for future use	
303000		
302000:1777	XBS34	
302000:42	reserved	
302000:41		
302000	control registers	
301000:1777	VIB/VI24	
301000:6	reserved	
301000:5		
301000	control registers	
300000:7777	FBC	
300000:1000	reserved	
300000:777		
300000:400	programmable curser	
300000:377		
300000:10	reserved	
300000:7		
300000	control registers	

(continued)

IKONAS MEMORY MAP (continued)

2777$1777 2070$0		reserved for future use
2067$1777 2060$2 2060$1 2060$0	CQM4	reserved address of CQMCB CQM4 Base Control Block
2057$1777 2050$0	BPS	writes as control register reads as current PC
2047$1777 2040$3 2040$2 2040$0 2040$1777 2040$0 2040$1777 2040$0	MA1024	reserved control registers multiplier microcode memory transformation matrix memory
2037$1777 2030$1 2030$0 2030$1777 2030$0	LUVO	reserved channel crossbar control colormap memory
2027$1777 2020$0	SR8	8K (10$0) words per SR8 - maximum of 8
2017$1777 2010$0		reserved
2007$1777 2000$0	MCM4	8K (10$0) words per MCM4 - maximum of 8
1777$1777 0$0	DR64	framebuffer memory - up to 12 DR64s each DR64 provides 512x512x8 pixels LORES 1024x1024x2 pixels HIRES

IKONAS BUS FUNCTION CODES

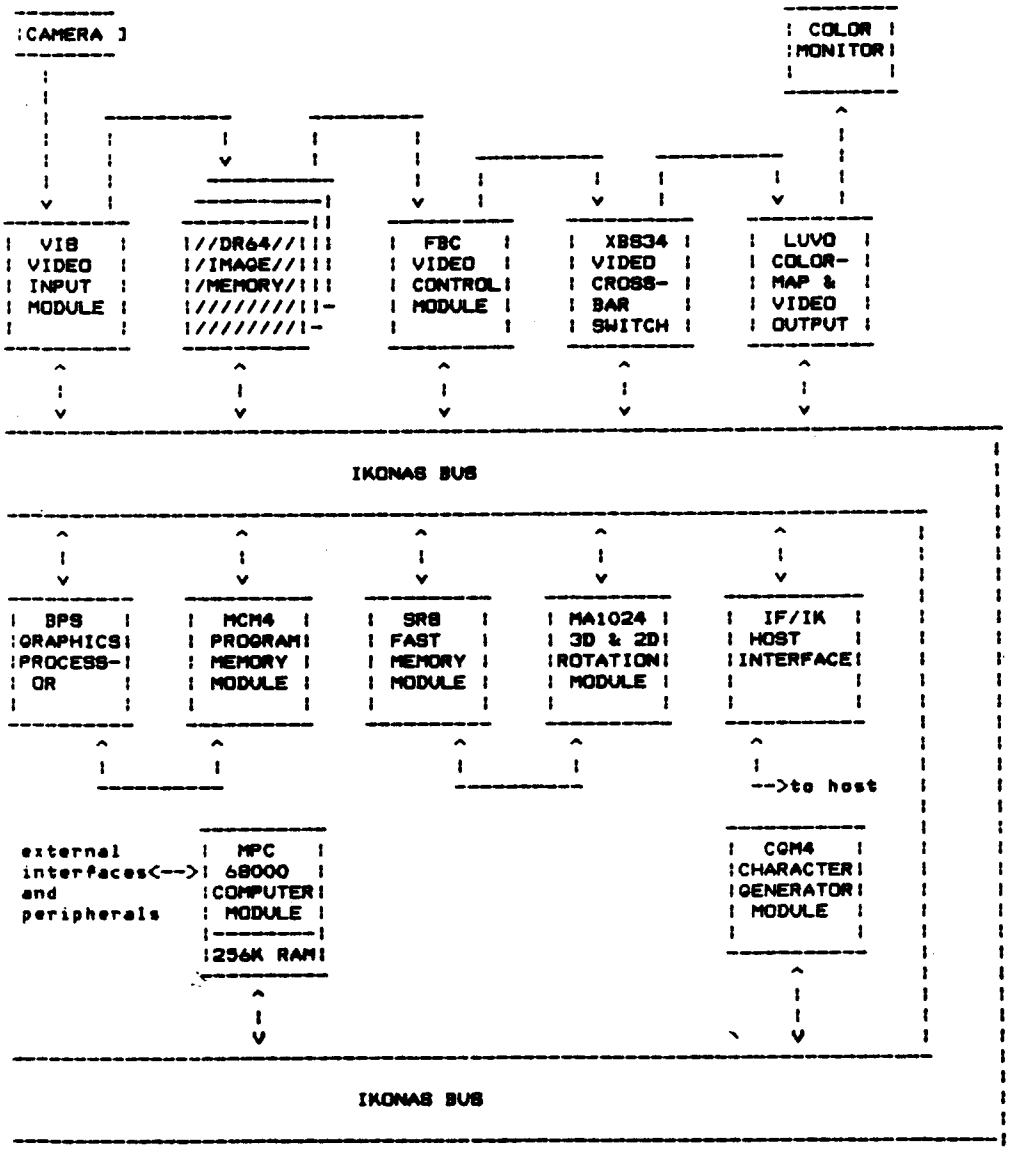
The 5-bit function code is present for each IKONAS bus operation. The function code tells:

- o whether the operation is a read or a write
- o how the address will be interpreted by the addressed module
- o what operation will be performed by the addressed module

for example, if the address is 100$100, then:

if the function code is:	the operation will be:
0	nothing - invalid address
2	LORES read from pixel (40,40)
3	HIRES read from pixel (100,100)
20	nothing - invalid address
22	LORES write to pixel (40,40)
23	HIRES write to pixel (100,100)
32	set write mask in all LORES cards

THE IKONAS DR64



THE DR64 IMAGE MEMORY

Features:

- provides 512x512 pixels of 8 bits each or 1024x1024 pixels of 2 bits each
- configurable into images of up to 512x512x32 or 1024x1024x24
- software selectable between 512x512 and 1024x1024 display
- may be used as 64Kx32 bulk memory
- separate port for input at video rate
- allows each bitplane to be protected from modification

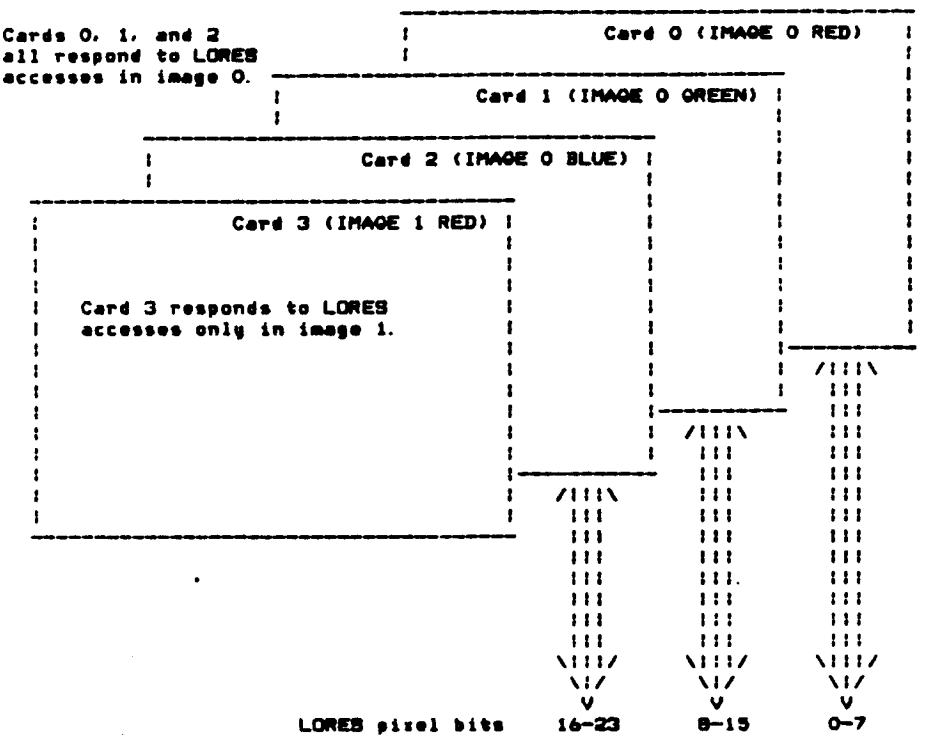
Configuring the DR64 Image Memory

A system is configured by specifying how each DR64 contributes its 8 bits at 512x512 or 2 bits at 1024x1024 to the pixel memory.

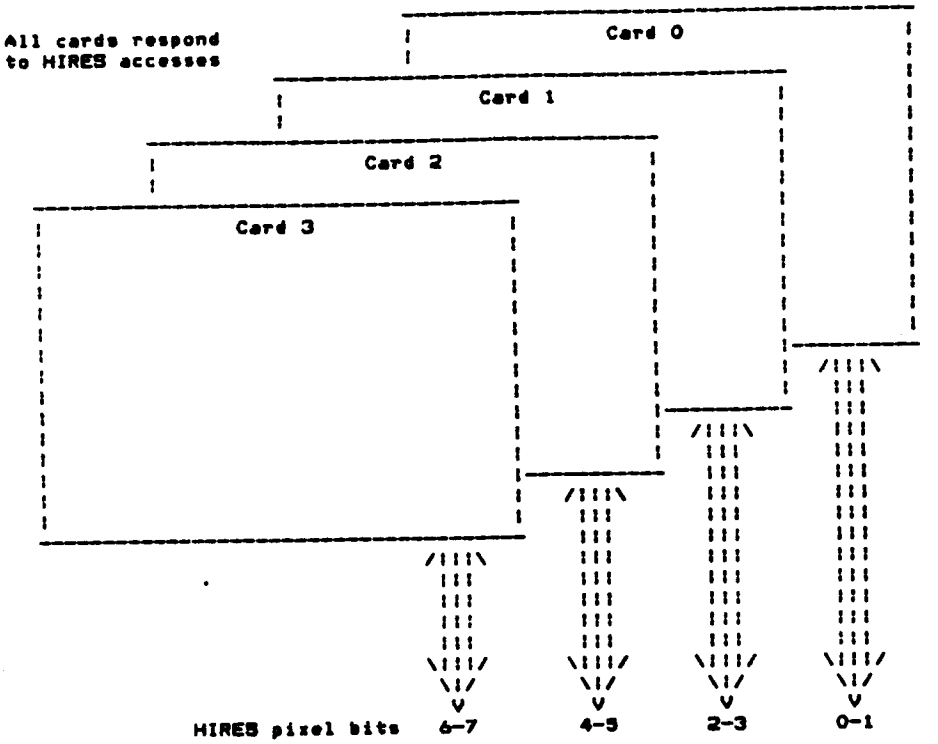
When a pixel is addressed, more than one DR64 will respond to the address. Each DR64 which is configured to contribute to the pixel will respond. All DR64s may respond to the same pixel address.

In 512x512 (LORES) display, three DR64s provide a 24-bit full-color image. If you have more than three DR64s, you can have more than one full 512x512 image stored.

Example of configuration for LORES



Example of configuration for HIRES



Addressing the DR64 in LORES mode

In 512x512 mode, pixels are addressed as (x,y) coordinates. The IKONAS address is y$x. In LORES mode the y and x coordinates must be multiplied by 2. The IKONAS bus function code is 2 (read) or 22 (write).

The following diagram gives the y$x addresses of pixels on the screen, as addressed and displayed in LORES mode.

Top left pixel

DR64 PIXEL ADDRESSES

000	002	004	017772	017774	017776
200	202	204	217772	217774	217776
400	402	404	417772	417774	417776
600	602	604	617772	617774	617776
177000	177002	177004	177017772	177017774	177017776
177200	177202	177204	177217772	177217774	177217776
177400	177402	177404	177417772	177417774	177417776
177600	177602	177604	177617772	177617774	177617776

Bottom right pixel

Addressing the DR64 in HIRES mode

In 1024x1024 mode, pixels are addressed as (x,y) coordinates. The IKONAS address is y$x. The IKONAS bus function code is 3 (read) or 23 (write).

The following diagram gives the y$x addresses of pixels on the screen, as addressed and displayed in HIRES mode.

Top left pixel

DR64 PIXEL ADDRESSES

000	001	002	001775	001776	001777
100	101	102	101775	101776	101777
200	201	202	201775	201776	201777
300	301	302	301775	301776	301777
177400	177401	177402	177401775	177401776	177401777
177500	177501	177502	177501775	177501776	177501777
177600	177601	177602	177601775	177601776	177601777
177700	177701	177702	177701775	177701776	177701777

Bottom right pixel

Accessing the DR64 in WORD mode

In WORD mode each DR64 is a 64Kx32-bit bulk memory. The WORD mode addresses start at 1000$0 and go for as far as you have DR64 memory.

Therefore, the first DR64 has word mode addresses from 1000$0 to 1077$1777; the second DR64 has addresses 1100$0 to 1177$1777; and so on.

The IKONAS bus function code is 0 (read) or 20 write).

Notes on different addressing modes

Remember, even though there are three different addressing modes for the DR64, only one set of data bits is stored. Addressing in the different modes only gives you different views of the same data.

In particular this means:

- when you store an image in HIRES you cannot meaningfully display it in LORES
- when you store an image in LORES you cannot meaningfully display it in HIRES
- when you store an image, you are overwriting any data you previously stored in those DR64s, even if you stored the previous data using a different addressing mode

In the configuration example given above, storing into LORES image 0 will destroy HIRES bits 0-5; storing into the HIRES image will destroy all LORES bits for images 0 and 1; storing into card 0 in word mode will destroy the red component of image 0 (bits 0-7).

The DR64 Write Mask

The write mask allows you to protect bitplanes of the image memory from modification.

There is one bit in the write mask to correspond to each bit of a pixel. When a write is performed, only those bits which have 1s in the corresponding bit positions of the write mask will be modified. Bits which have 0s in the corresponding positions of the write mask will be unchanged.

The write mask is always active.

Whatever is in the write mask will always be used to mask any write to a DR64.

To set the write mask:

1. use the IKONAS bus function code corresponding to the kind of writes you will be using: 32 for LORES, 33 for HIRES.

2. use the IKONAS address of any pixel in the image you want to set the write mask in. For example, 0$0 for image 0; 2000$0 for image 1; etc.

3. write the data which will become the write mask.

Example (8 bits per pixel):

Old data	11111111
Write mask	00001111
Data written	01100110
Result data	11110110

Because of the write mask, only bits 0-3 are modified; bits 4-7 are unchanged.

The GM64 Graphics Memory

The GM64 Graphics Memory Module is largely compatible with the DR64 Image Memory.

- Features compatible with the DR64
  - 512x512 8-bit pixels or 1024x1024 2-bit pixels per card
  - Programmably switched between 512x512 and 1024x1024 mode
  - HIRES and LORES read and write identical to DR64
  - WORD mode read identical to DR64
  - Video input port allows real-time image capture
- New features
  - Mask-mode writes: any or all of 32 pixels may be written simultaneously with a preset value
  - Enhanced video input port allows real-time generation of polygons
  - Z-buffer option: up to 24 bits may be used to store z; z-comparison and conditional write occur with no performance degradation.

Mask Mode Writes

Mask mode allows you to write up to 32 pixels with one operation. The operation of function code 20 (write word mode) is altered, and new function codes 24, 36, and 37 are added.

Mask mode consists of two steps:

1. Use a set shade function to store the data value to be used by subsequent mask-mode writes. Function codes to use are

   36 for LORES

   37 for HIRES

2. Use the mask-mode write function to write up to 32 pixels (HIRES) or 16 pixels (LORES). Function codes to use are

   24 for LORES. Bits 0-15 of the IKONAS bus data will be the mask.

   20 for HIRES. Bits 0-31 of the IKONAS bus data will be the mask.

   Each of the 32 data bits (for LORES, the low 16 bits) of the mask-mode write data corresponds to one pixel in the GM64 memory. A one-bit in a pixel position causes that pixel to remain unchanged; a zero-bit in a pixel position causes the previously set shade to be stored in that pixel. All GM64 cards respond to the set-shade and mask-mode write operations.

The Sender Id

The sender id is a three-bit field, set in each IKONAS bus write, indicating the originator of the write. Each IKONAS module appends its sender id to each write operation. Most modules allow the sender id to be set under program control; for example, the BPS32, the IF/IK, and the MPC allow the user to specify the sender id. Other modules use a fixed sender id: the FBC always uses sender id 0.

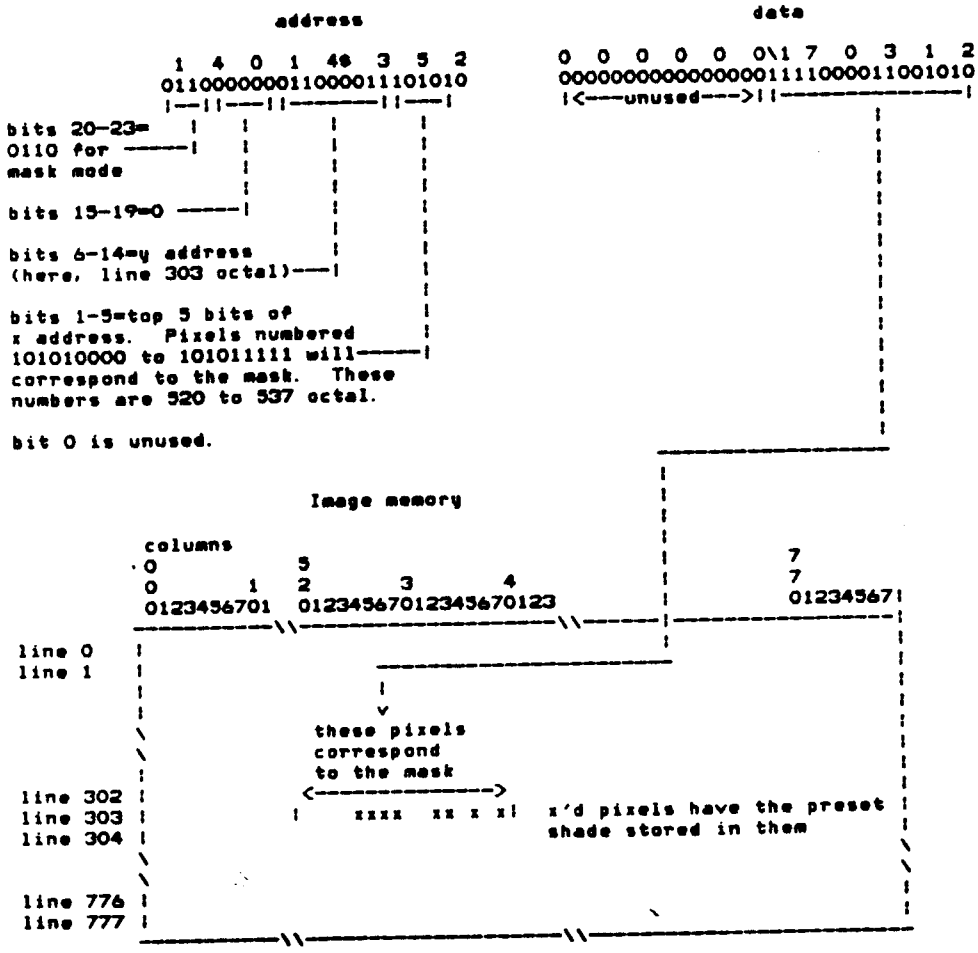
The sender id is useful in cases where an operation is a two-step process -- for example, in mask-mode writes or use of the write mask. The second step of the process uses the data stored by the first step which had the same sender id as the second step.

In the GM64, the sender id is used by all write operations. Each sender id has its own write mask and mask-mode shade. A write operation uses the write mask and shade corresponding to the sender id for that write.

This means that you may have up to eight write masks and mask-mode shade values active at one time. Each process using the memory may be alloted its own write mask and shade value, which will not interfere with any other.

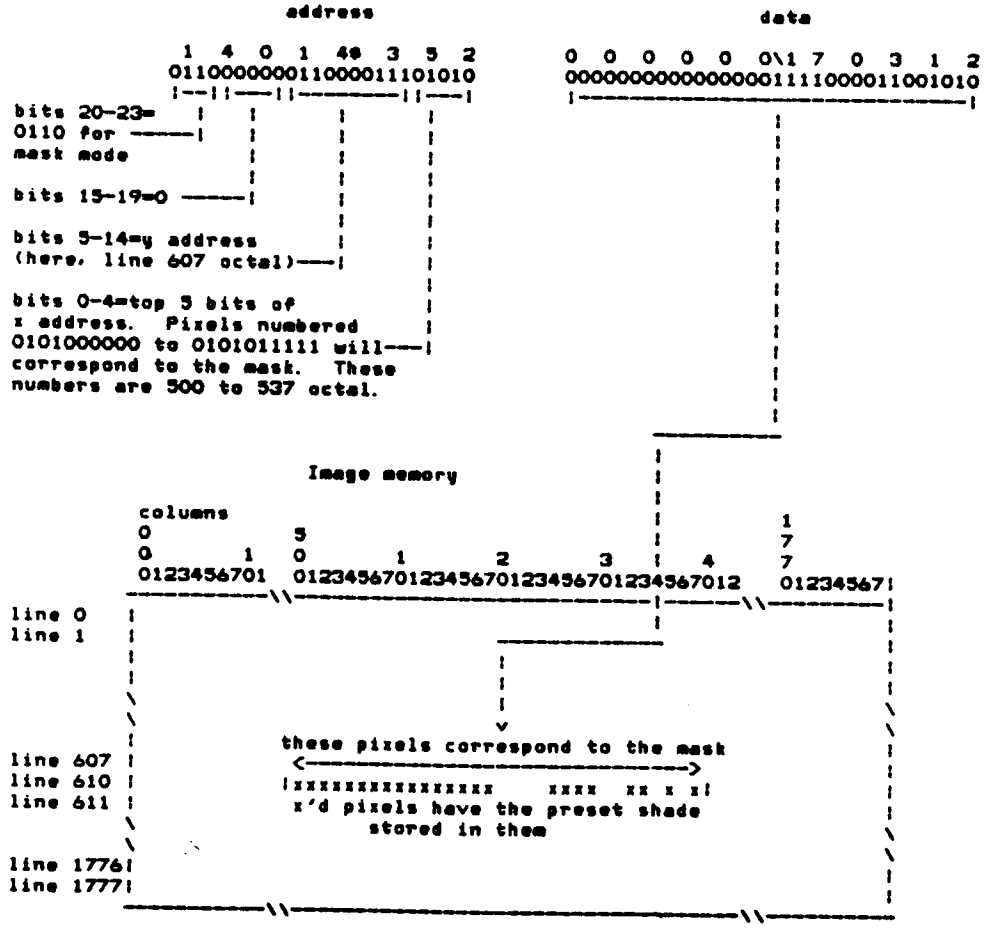
Example of Mask-mode Write - LORES

Step 1. Use function code 36 to set the shade.
Step 2. Issue the mask-mode write using function code 24. Note the setting of the top four address bits:



Example of Mask-mode Write - HIRES

Step 1. Use function code 37 to set the shade.
Step 2. Issue the mask-mode write using function code 20. Note the setting of the top four address bits:



Z-buffer Option

When the Z-buffer option is installed, an additional memory function is added: Certain bitplanes are configured as z-planes by wire-wrap jumpers on the GM64. When a conditional write is performed for a given function, the value to be written into those bitplanes is compared against the value currently stored there.

If the old value is less than the new value, the write is suppressed and the location remains unchanged.

If the old value is greater than or equal to the new value, the write is performed as a normal memory write operation.

The conditional write operation takes no more time than a normal write.

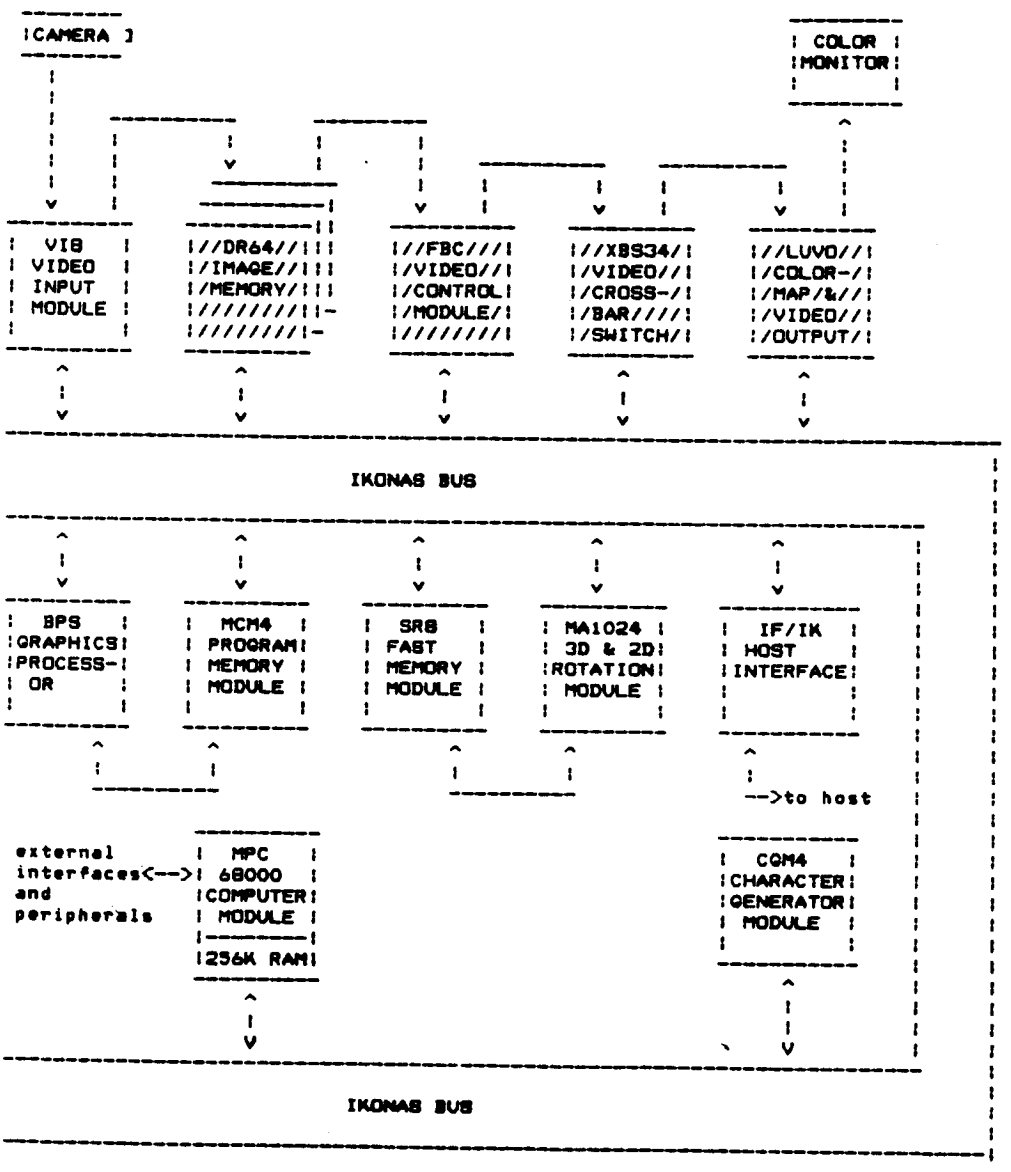
The GM64 is wire-wrap jumpered to allow LORES conditional write or HIRES conditional write. Use the proper function code for the cards in your system:

26 for LORES conditional write

27 for HIRES conditional write.

Normal operation of the GM64 is unaltered by the Z-buffer option. HIRES and LORES normal writes (unconditional) are unchanged.

THE VIDEO DATA PATH

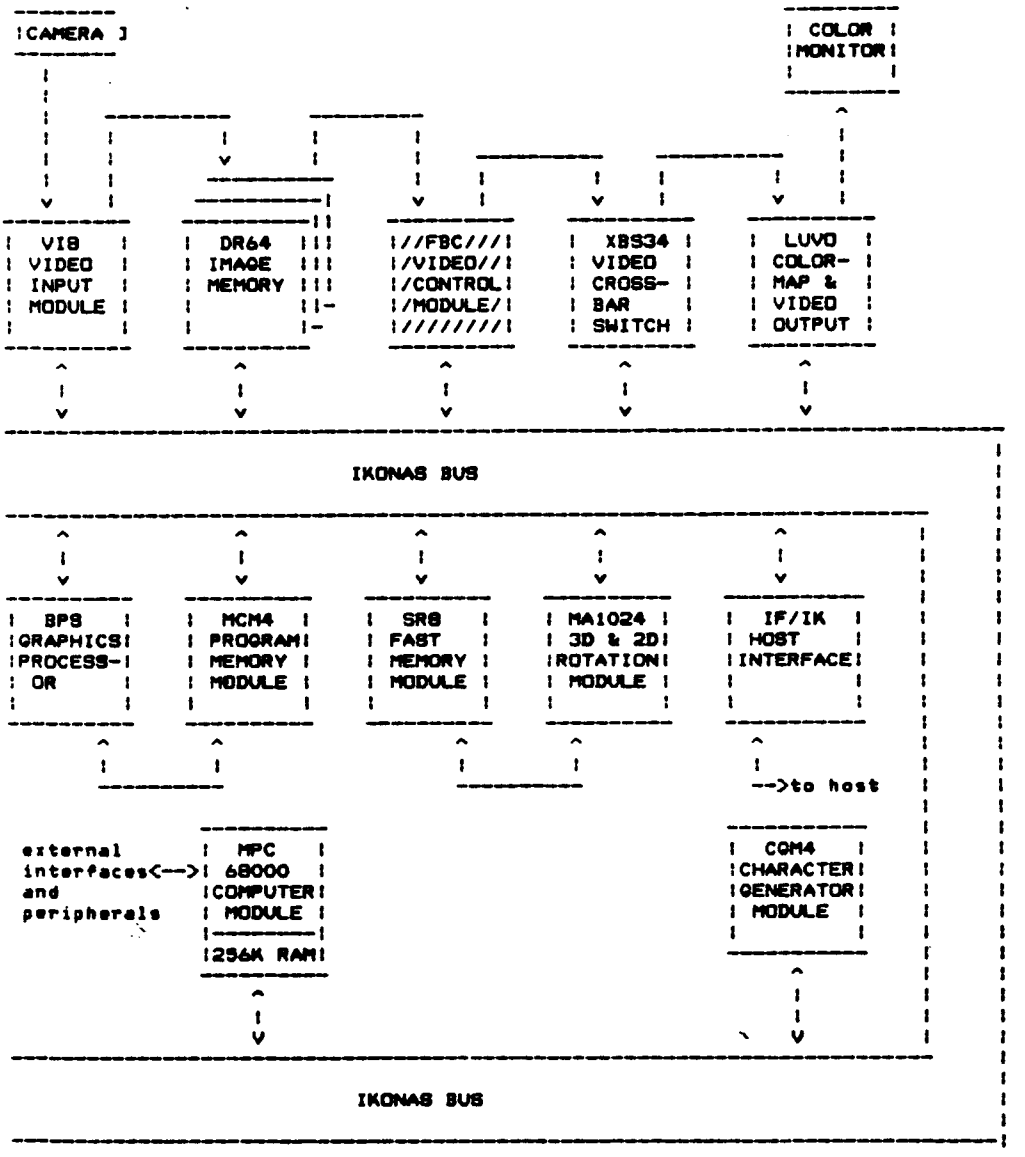


THE IKONAS VIDEO CHAIN

The video chain is the set of modules which process data at pixel rates. This consists of three modules:

- FBC - figures out what data to display, fetches it from the DR64, and starts it on its way through the video chain. Also inserts sync, blanking, cursor.
- XBS - rearranges bits of a pixel at pixel rates.
- LUVO - (colormap) performs a table lookup on each color of the pixel, and sends the result of the lookup to the digital-to-analog converters which produce standard video.

THE FRAME BUFFER CONTROLLER



The Frame Buffer Controller

The Frame Buffer Controller (FBC) controls the format and content of the display. Items controlled by the FBC are:

- o video sync rates - number of lines per field and time per line
- o viewport - the size and position on the screen of the displayed data
- o window - the position of the frame buffer data within the viewport
- o zoom - the magnification factor for each pixel
- o cursor - the shape and location of the 32x32-bit programmable cursor
- o aspect ratio - the ratio of the width of the picture to the height
- o automatic erase - automatically erases the screen after display
- o and others.

The FBC produces correct video sync information to conform to the EIA RS-170A standard or the RS-343 standard, selectable programmably.

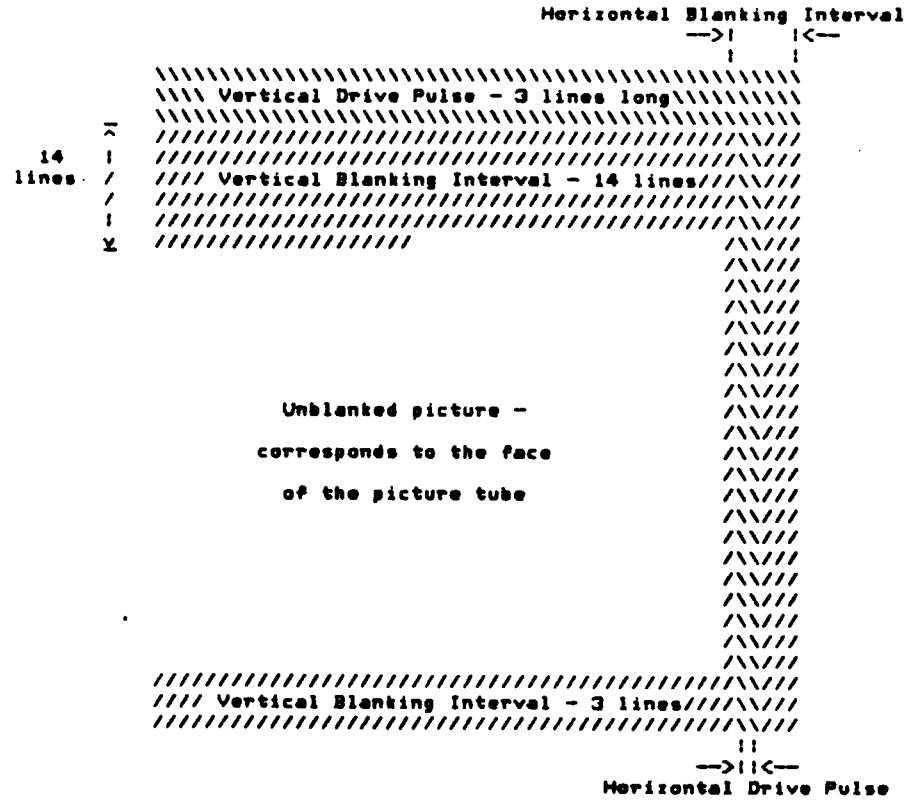
Definitions of Video Terms

Since the FBC produces a video signal, you must understand what the video signal looks like and how it can be controlled by the FBC.

The diagram on the next page shows the video signal as a sequence of lines, just as it appears on the color monitor. One such sequence, from the top of the monitor to the bottom, is called a field. Note that part of each line, and several lines at the top and bottom of the picture, are not displayed at all. This part of the picture is called the blanked portion of the picture. Blanked portions of the picture are denoted by / in the diagram.

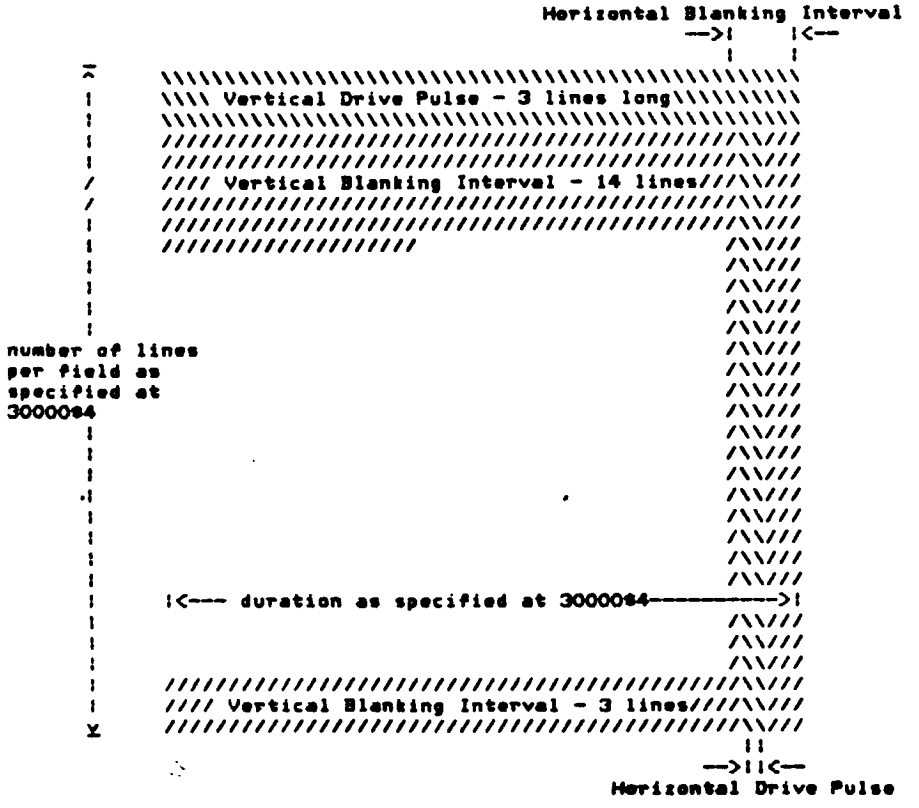
Embedded in the blanked portion of the picture are reference pulses used by the monitor. The horizontal drive pulse occurs during each line. The vertical drive pulse occurs during the top of each field. The horizontal and vertical drive pulses are denoted by \ in the diagram.

Video Terms Diagram



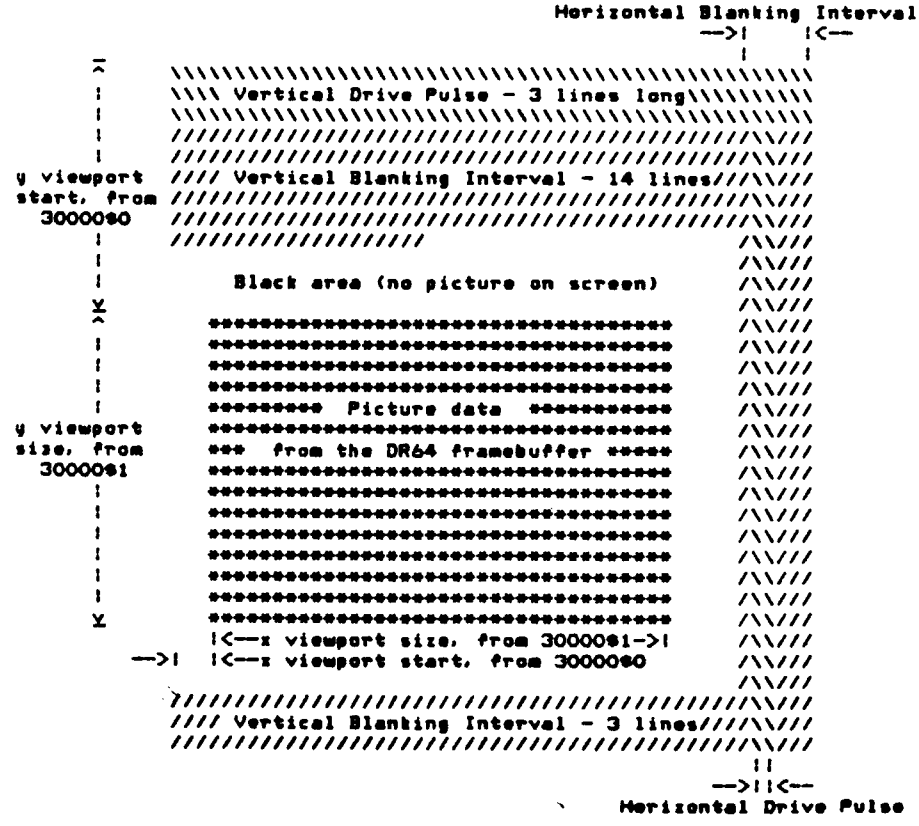
FBC Video Formatting - Sync Control

The FBC controls the sync information in the video signal by controlling the number of lines per field, and the length of time for each line. This information is in the FBC register at 30000$4.



FBC Video Formatting = Viewport

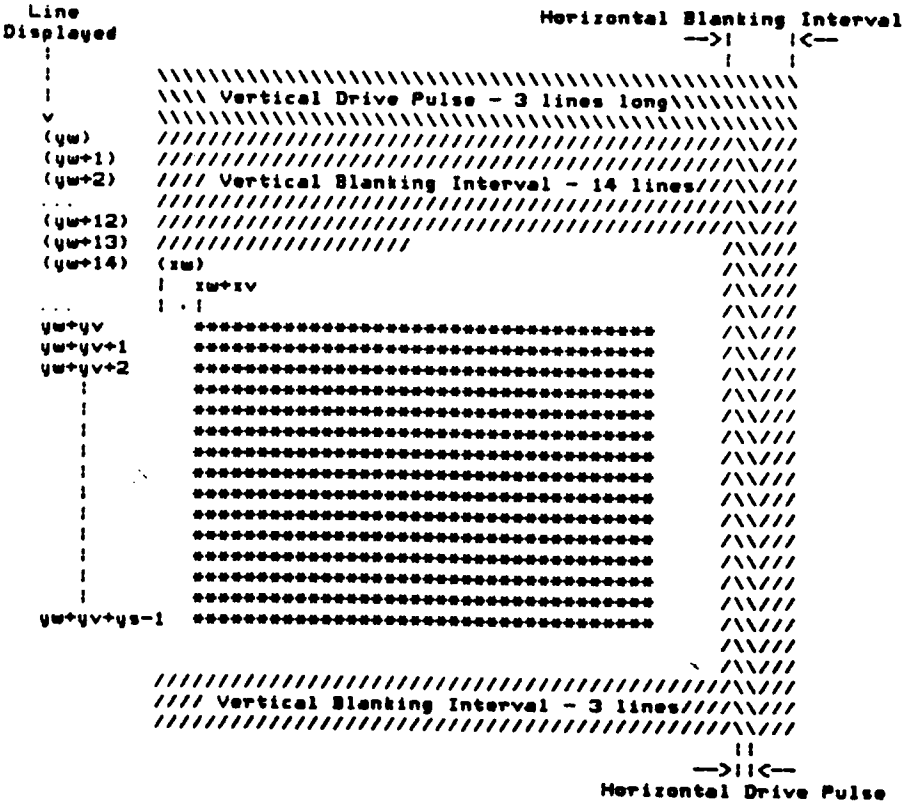
The unblanked portion of the picture corresponds to the face of the color monitor (roughly), but the FBC allows you to display less than the full 1024 lines and columns, and allows you to position the displayed data anywhere on the monitor. Two control registers, at 30000$0 and 30000$1, control the start of viewport and length of viewport respectively.



FBC Video Formatting - Window

The viewport registers, defined above, specify where on the color monitor the displayed picture will fall; the window controls what data from the DR64 framebuffer memory will be displayed in the viewport. The window is controlled by the register at 30000$2.

In the diagram, let yw be the y window setting, xw the x window setting, yv the y viewport setting, and xv the x viewport setting. The diagram shows the line which will appear on the screen at each location. Note that as the viewport changes, the visible picture moves, but the actual scene does not appear to move. Instead, data at the moving edges is revealed or obscured.



Zoom

Zoom is a magnification of the picture by repeating pixels and lines. The effect of zoom is to make each pixel grow, thus making the image grow.

On the IKONAS, zoom in the x and y directions is independent.

You may zoom only in integral multiples: so the smallest zoom is a factor of 2. The largest zoom is a factor of 256.

Zoom is controlled by the register at 30000$3.

Cursor

The IKONAS cursor is a 32x32 programmable overlay. There are two steps to displaying a cursor:

- first, define the cursor by specifying the 32x32 matrix
- then specify the location of the 32x32 overlay on the screen

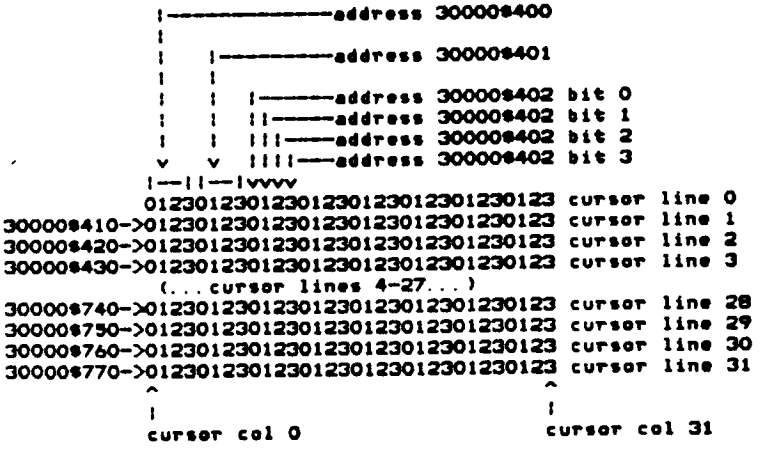
Once you define the cursor, you can move it around on the screen by changing the location register. You do not have to redefine the cursor unless you want the overlay pattern to change.

Defining the cursor

The cursor is a 32x32 matrix, for a total of 1024 bits. You define these 1024 bits by writing 256 4-bit numbers into the cursor definition table at 30000$400-30000$777.

The diagram below shows the correspondence of addresses and bits to pixels in the 32x32 cursor matrix.

The numbers 0, 1, 2, and 3 denote the bit position in the 4-bit word at the address specified. Each digit corresponds to one pixel of the cursor.



Positioning the Cursor

The cursor is positioned by setting the cursor position register, 30000$6, to the desired (x,y) coordinates. The cursor must be turned on in the flag register before it will be displayed.

When the cursor is active, the 32x32 cursor matrix appears as an overlay upon the framebuffer picture.

The cursor location register gives the location (x,y) of the top left point of the cursor. Pixels x to x+31 on lines y to y+31 then correspond to the 32x32 cursor matrix.

Any of these pixels which corresponds to a 1-bit in the 32x32 cursor matrix will be 'in the cursor'; any which corresponds to a 0 will 'outside the cursor'. Any pixel outside of the 32x32 overlay will also be outside the cursor.

Pixels in the cursor are denoted by a tag bit appended to them as they pass through the video chain. This bit is used by the LUVO to select the shade displayed at the pixel.

Example of cursor display

```
Cursor matrix: 10000000000000000000000000000001
               01000000000000000000000000000010
               00100000000000000000000000000100
               00010000000000000000000000001000
               00001000000000000000000000010000
                                etc.
               00001000000000000000000000010000
               00010000000000000000000000001000
               00100000000000000000000000000100
               01000000000000000000000000000010
               10000000000000000000000000000001
```

Cursor location set to (100,100)

Screen will be:

```
100$100
   |
   v
   *   *
    * *
     *
    * *
   *   *
       ^
       |
    137$137


        The *s represent
        pixels in the cursor.
```

FBC Flag Bits and Mode Settings

The register at 30000$5 contains flag bits which specify modes of operation of the FBC. This register is allocated as follows:

Bits	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
	| pixel clock | |R|3|P|G |X|E| |H|C|

Definitions of these bits are as follows:

C (bit 2) - ON to cause the cursor to be displayed, OFF to suppress the cursor

H (bit 3) - ON for 1024x1024 (HIRES) display, OFF for 512x512 display

E (bit 5) - ON for automatic erase, OFF for normal display. When automatic erase is in effect, the data in the DR64 framebuffer will be erased every field.

X (bit 6) - ON if external sync information (from the LUVO) is to be used, OFF if video sync is to be generated by the FBC.

PG (bits 7-8) - colormap page. These bits are appended to the pixel data as two tag bits for processing in the video chain.

3 (bit 9) - ON if RS-343 sync is to be generated, OFF if RS-170A sync is to be generated. This bit should be off for 512x512 30 Hz NTSC standard video, on otherwise.

R (bit 10) - ON for repeat field operation, OFF for interlaced.

pixel clock (bits 16-22) - number of nanoseconds per single pixel in LORES, or per two pixels in HIRES. This field may be used to change the aspect ratio.

Summary of FBC Control Registers

Bits	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
3000080	y viewport start | x viewport start
3000081	y viewport size | x viewport size
3000082	y window location | x window location
3000083	y zoom | x zoom
3000084	number lines per frame | time per line
3000085	pixel clock | R I3 P0 X E H C
3000086	y cursor position | x cursor position

Notes:

1. The two low-order bits of the x and y viewport start and size values are ignored, and 0 is used for them; so you can set the starting position and size in multiples of four lines. The size used is actually the size given+4, so the size ranges from 4 to 1024.

2. When the REPEAT FIELD bit is set in 30000$5, the y viewport size should be multiplied by two.

3. In LORES display, the x window should be multiplied by four.

4. Refer to the FBC Programming Guide for an exact discussion of the relationship between y window and y viewport.

5. Zoom counts should be set to one less than the magnification desired. Zoom count of 0 = unmagnified display, 1 = zoom by factor of 2, etc.

6. The number of lines per frame is more precisely the number of half-lines per field plus one. This number should be even for interlaced operation, odd for repeat-field operation.

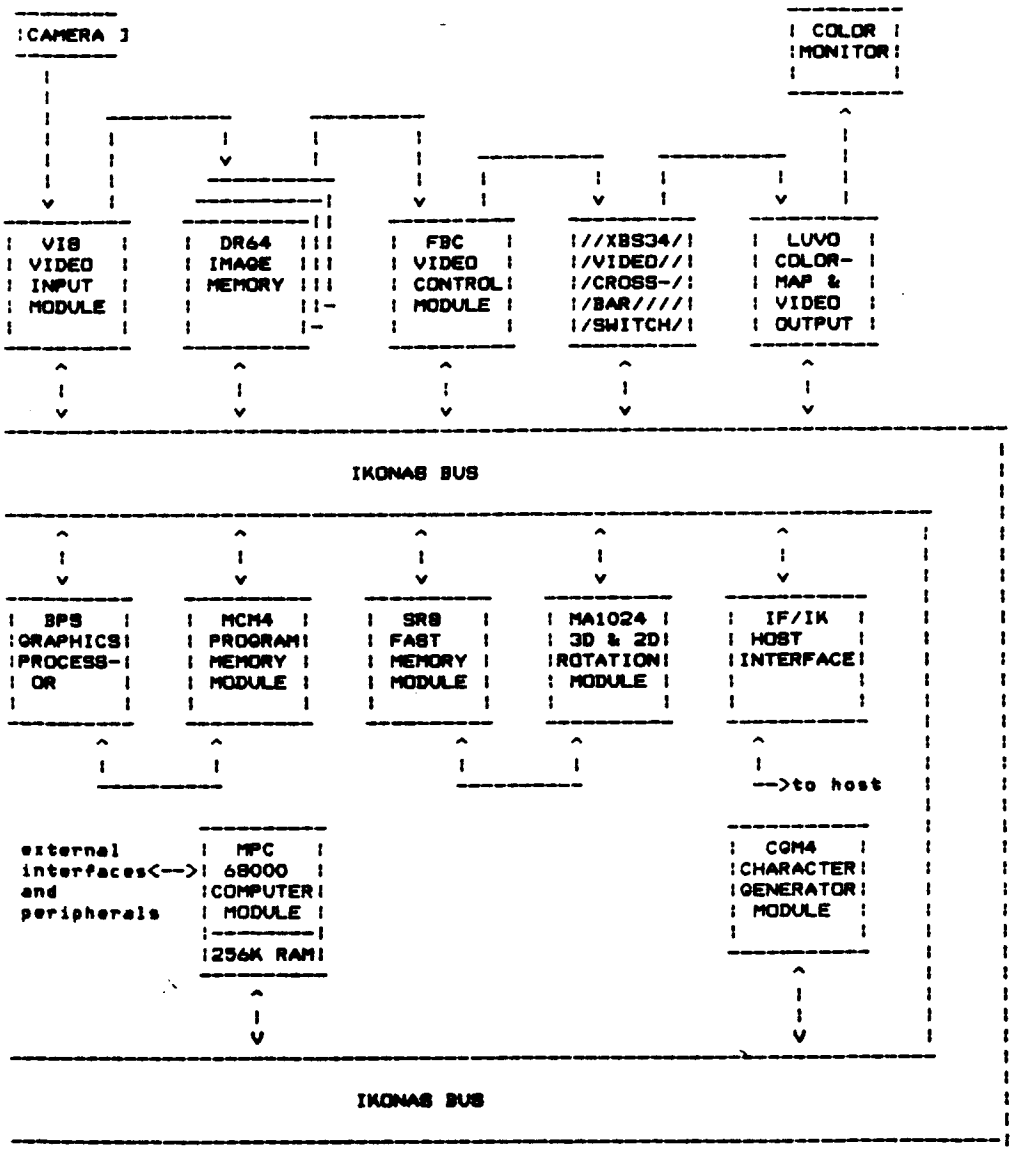
7. The pixel clock can range from 20 to 77. Operation at settings below 26 is not guaranteed by IKONAS.

The Video Data Path as it Leaves the FBC

The FBC starts data down the video chain. The pixel data as it leaves the FBC is as follows:

Bit number	Contents
0	Pixel data bit 0
1	Pixel data bit 1
2	Pixel data bit 2
3	Pixel data bit 3
4	Pixel data bit 4
5	Pixel data bit 5
6	Pixel data bit 6
7	Pixel data bit 7
8	Pixel data bit 8
9	Pixel data bit 9
10	Pixel data bit 10
11	Pixel data bit 11
12	Pixel data bit 12
13	Pixel data bit 13
14	Pixel data bit 14
15	Pixel data bit 15
16	Pixel data bit 16
17	Pixel data bit 17
18	Pixel data bit 18
19	Pixel data bit 19
20	Pixel data bit 20
21	Pixel data bit 21
22	Pixel data bit 22
23	Pixel data bit 23
24	Pixel data bit 24
25	Pixel data bit 25
26	Pixel data bit 26
27	Pixel data bit 27
28	Pixel data bit 28
29	Pixel data bit 29
30	Pixel data bit 30
31	Pixel data bit 31
32	In-cursor bit
33	Colormap page bit 0
34	Colormap page bit 1

THE IKONAS XBS34 VIDEO SWITCH



XBS34 VIDEO CROSSBAR SWITCH

- allows arbitrary rearrangement of pixel data bits
- allows bitplanes to be suppressed or repeated
- processes at video pixel rates
- switching path can be quickly modified by software for frame-to-frame changes
- handles 35 inputs and produces 34 outputs

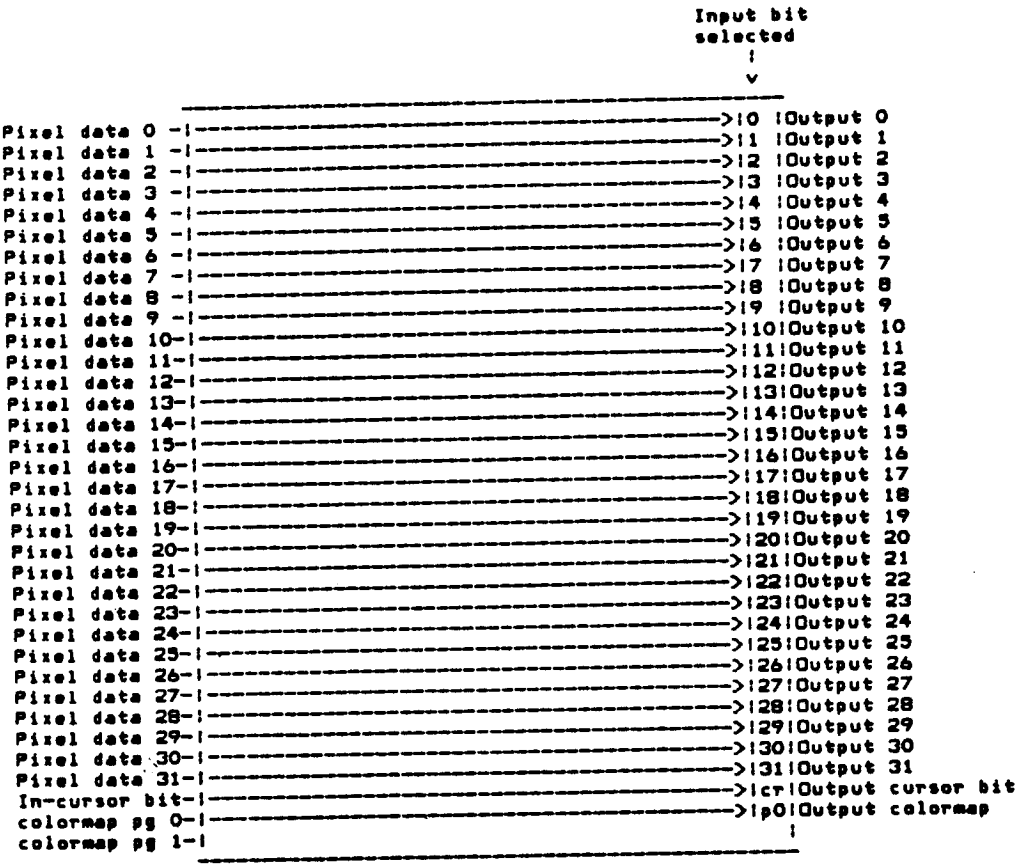
XBS34 Model

The XBS34 accepts up to 35 bits from the previous stage of the video chain and produces 34 bits of output. Each output bit may be selected from any of the 32 input pixel data bits or the constant zero, according to the setting in a register; in addition, the four most-significant outputs may be selected from any of the 35 input bits (which include the in-cursor bit and the colormap page bits).

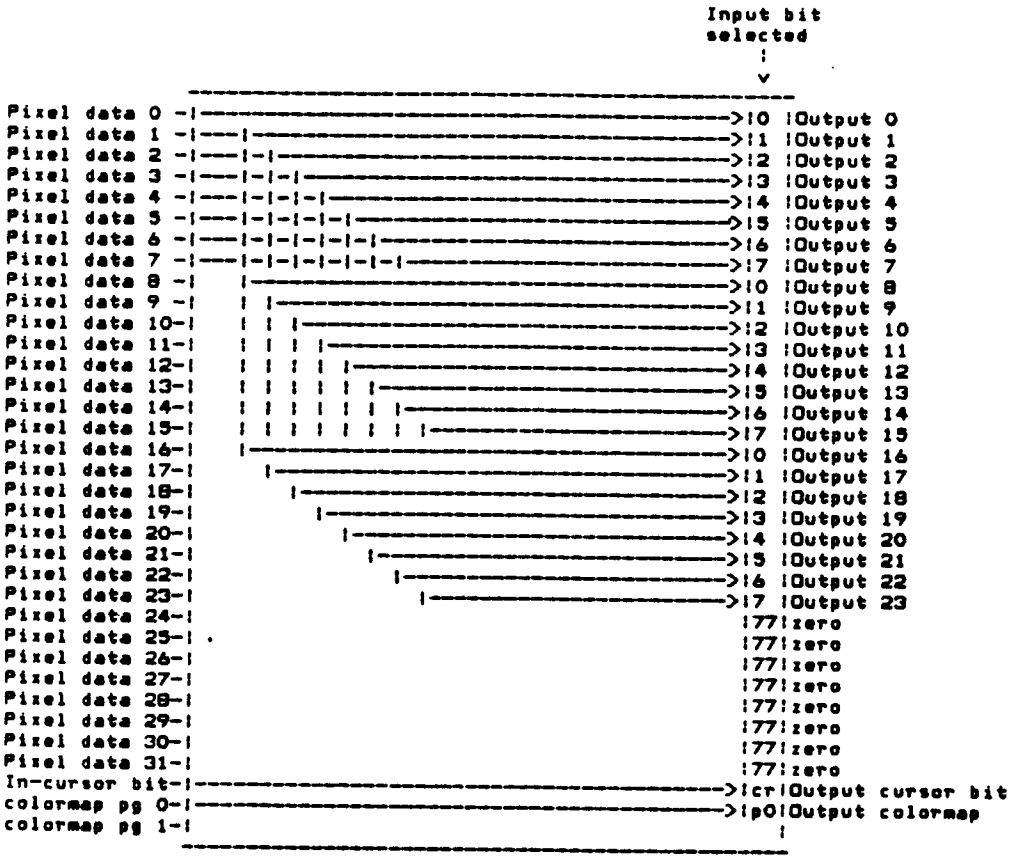
To set up the XBS34, you specify, for each output bit, the number of the input bits is to appear at that output bit. A number of 77 is used to specify that zero is to appear at the output bit.

For each output bit x, the number controlling that output is stored at 30200$x.

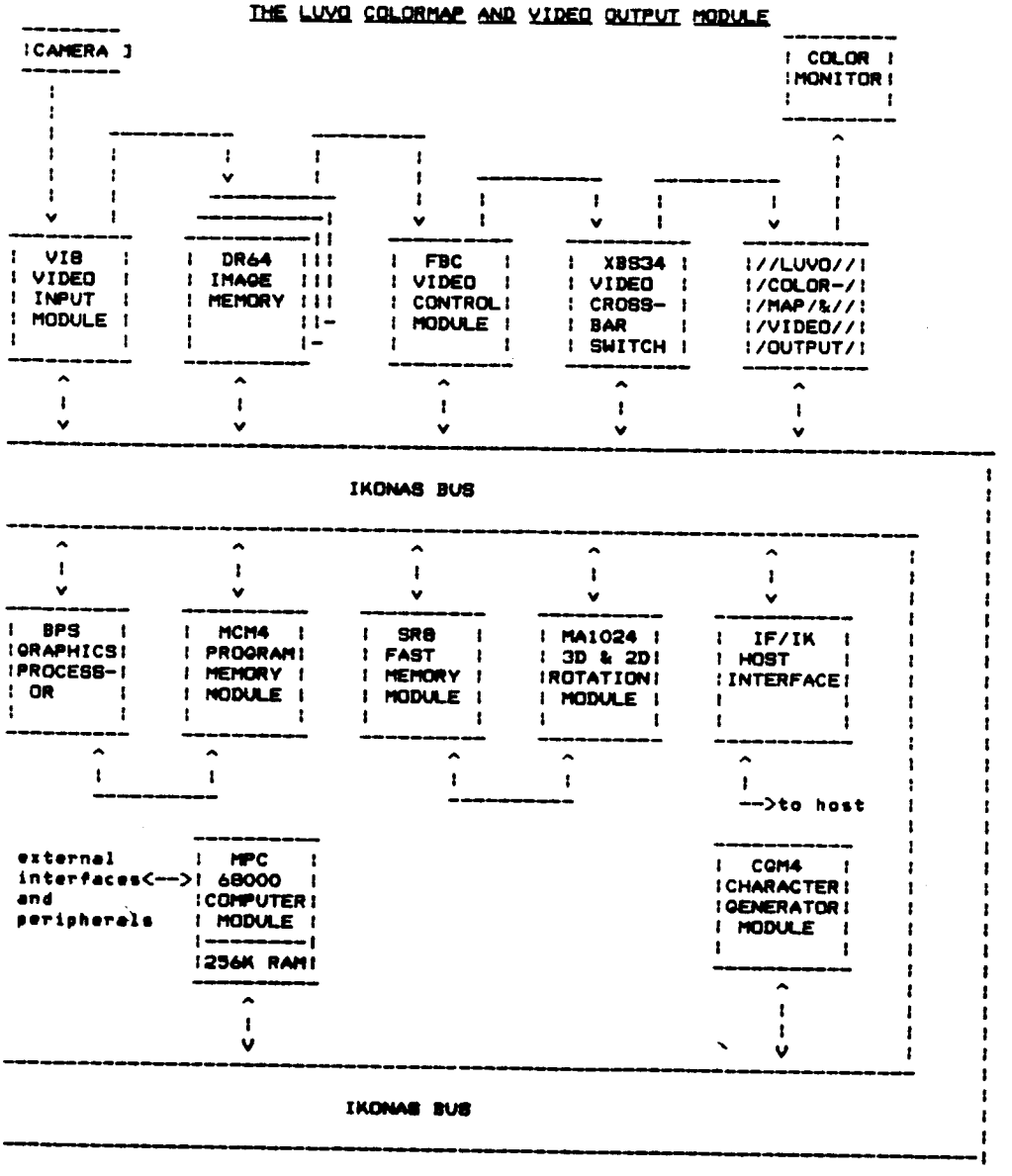
XBS34 Set Up in Transparent Mode (no change of pixel data)



XBS34 Set Up to Transfer Red Channel Input to All Three Outputs



THE LUVO COLORMAP AND VIDEO OUTPUT MODULE



The LUVQ Colormap and Video Output Module

- performs a color lookup on all three video channels, ten bits each
- full 30-bit output at video rates
- contains three independent lookup tables, one for each video channel
- fast access from IKONAS bus
- channel crossbar switch supports full-color images and pseudocolor
- NTSC standard video output and sync output
- accepts externally generated sync
- options to allow overlay bitplanes
- data-dependent cursor - allows solid, transparent, complementing, or more complex cursor functions

Colormaps

A colormap is a device which converts an input pixel value into an intensity to be used for one component (red, green, or blue) of the video output.

The LUVO contains three independent colormaps, one for each video output channel (red, green, and blue).

A colormap treats each pixel independently. Whenever a particular value appears at the input to the colormap, the output value is always the same.

The value produced by a colormap for each input value can be set by the programmer.

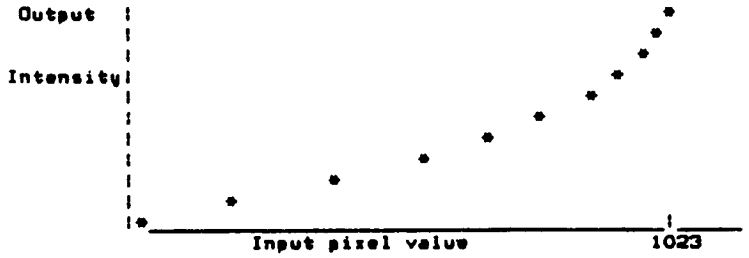
Colormaps have two main classes of use:

1. When the input pixels are in full color, with a red, green, and blue component, each component can be passed through a colormap which compensates for the characteristics of the display monitor and the observer's eye. This is called color correction.

2. When the input pixels are eight bits or fewer, each value of the input pixels may correspond to a different color. The same pixel value is passed to the three different colormaps, one for each output component (red, green, and blue); the resulting mixture of components gives the desired color for that input pixel value. This is called pseudocolor.

The colormap used for color correction

Since the colormap implements a function of the inputs, color correction can be shown in that way:



The pixel value for a particular channel consists of eight bits of pixel data, the in-cursor bit, and colormap page bit 0, as follows:

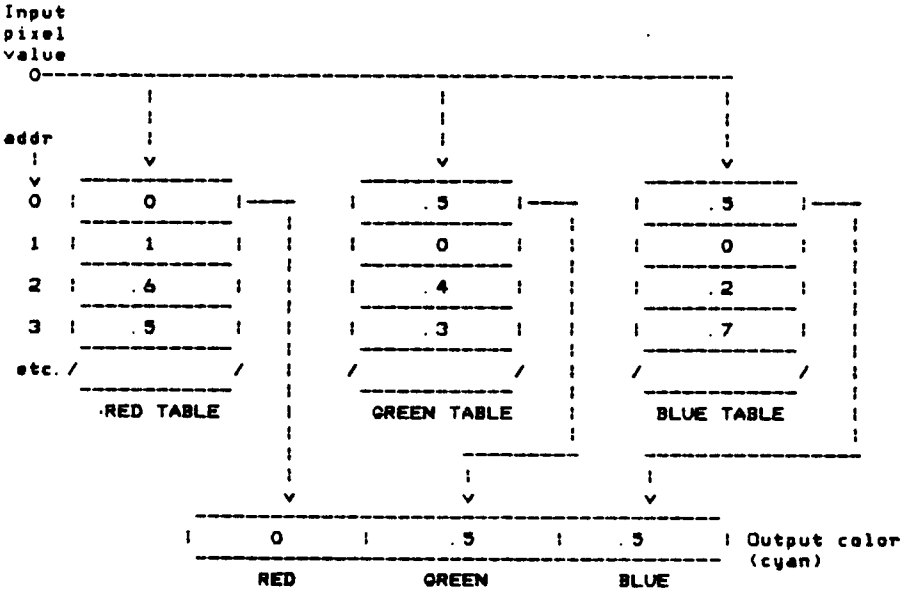
)=colormap page bit 0, C=in-cursor bit, both inserted by the FBC)

```
                bit 9 8 7 6 5 4 3 2 1 0
                    ---------------------
red channel:        |P|C| data bits 0-7 |
                    ---------------------

                bit 9 8 7 6 5 4 3 2 1 0
                    ---------------------
green channel:      |P|C|data bits 8-15 |
                    ---------------------

                bit 9 8 7 6 5 4 3 2 1 0
                    ---------------------
blue channel:       |P|C|data bits 16-23|
                    ---------------------
```

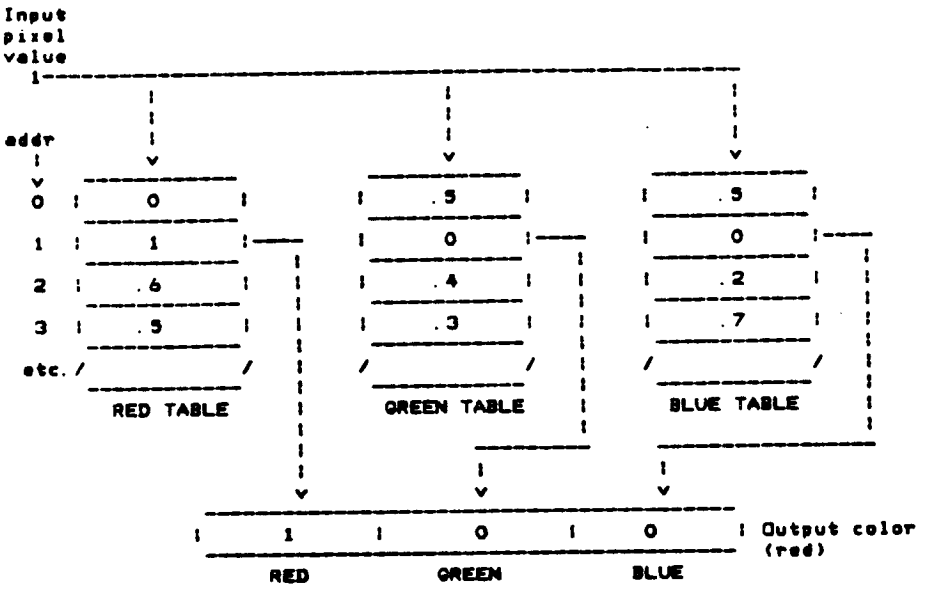
Colormaps used for pseudocolor

Colormaps used for pseudocolor resemble a translation table memory more than a mathematical function. The same pixel value is applied as the address of each memory, and the values stored at that address are used for the components of the color displayed.

In the example following, the colormap has been loaded with a palette of random colors.



Colormaps used for pseudocolor - continued



In pseudocolor, the pixel value is taken from some one of the red, green, or blue inputs, and the same value is sent to all three colormaps. The pixel value will be:

(P=colormap page bit 0, C=in-cursor bit, both inserted by the FBC)

bit 9 8 7 6 5 4 3 2 1 0

P	C	data bits ***

where *** is 0-7 for the red channel input, 8-15 for the green channel input, or 16-23 for the blue.

Colormap Pages and the Cursor

As can be seen from the format of the inputs to the colormaps, the colormap page bit and the in-cursor bit from the FBC become part of the pixel value applied to each colormap.

The in-cursor bit is applied as bit 8 of each colormap input. This bit is 1 for each pixel defined to be in the programmable cursor, and is 0 for all other pixels. In effect, you have two separate colormaps, one for pixels in the cursor and one for pixels not in the cursor. You may use this feature to provide a cursor whose value depends on the pixel value; for example, a complementing cursor, or a 'transparent' cursor which merely brightens the pixels.

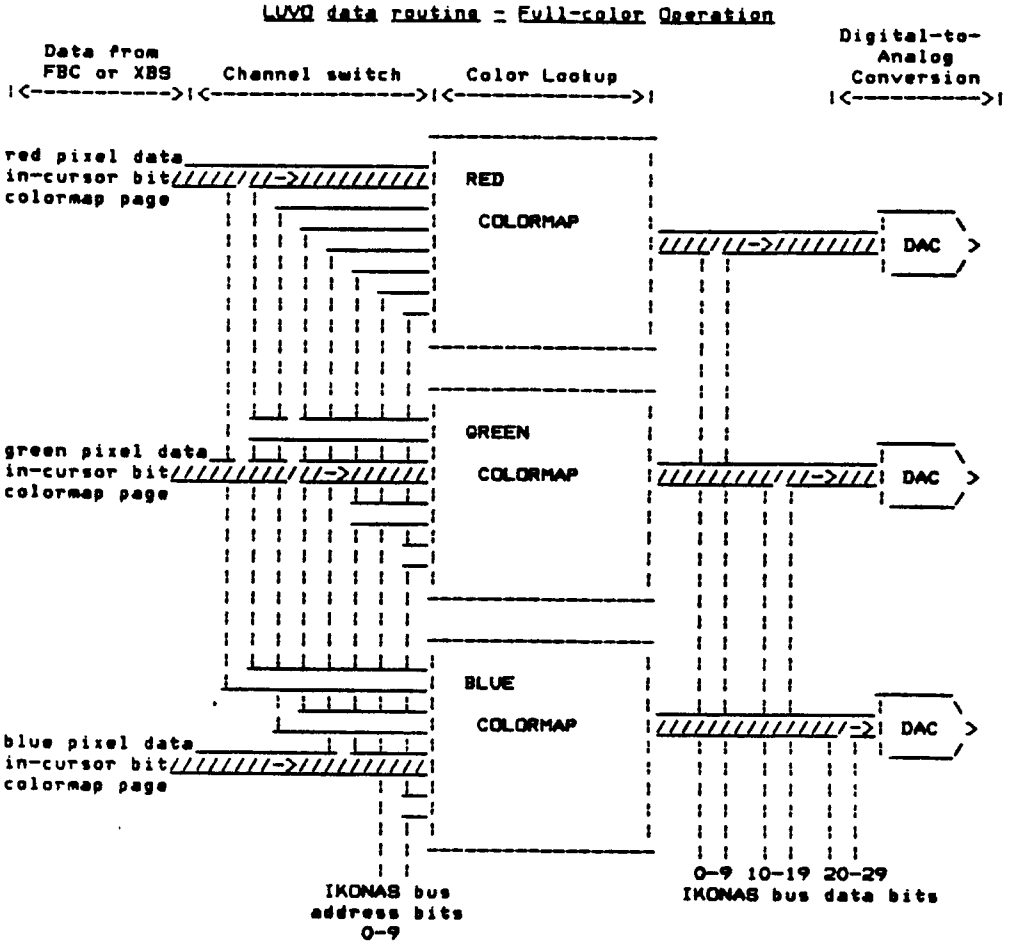
The colormap page bit (bit 7 of 30000$5) is applied as bit 9 of each colormap. This gives you the effect of having two colormaps, one with the page bit on and the other with the page bit off. You may switch between the colormaps merely by changing the value of the page bit in the FBC.

The channel switch

The channel switch on the LUVO allows you to select either full-color or pseudocolor operation. The channel switch is similar to the XBS34 crossbar switch, but you route entire video channels instead of single bits.

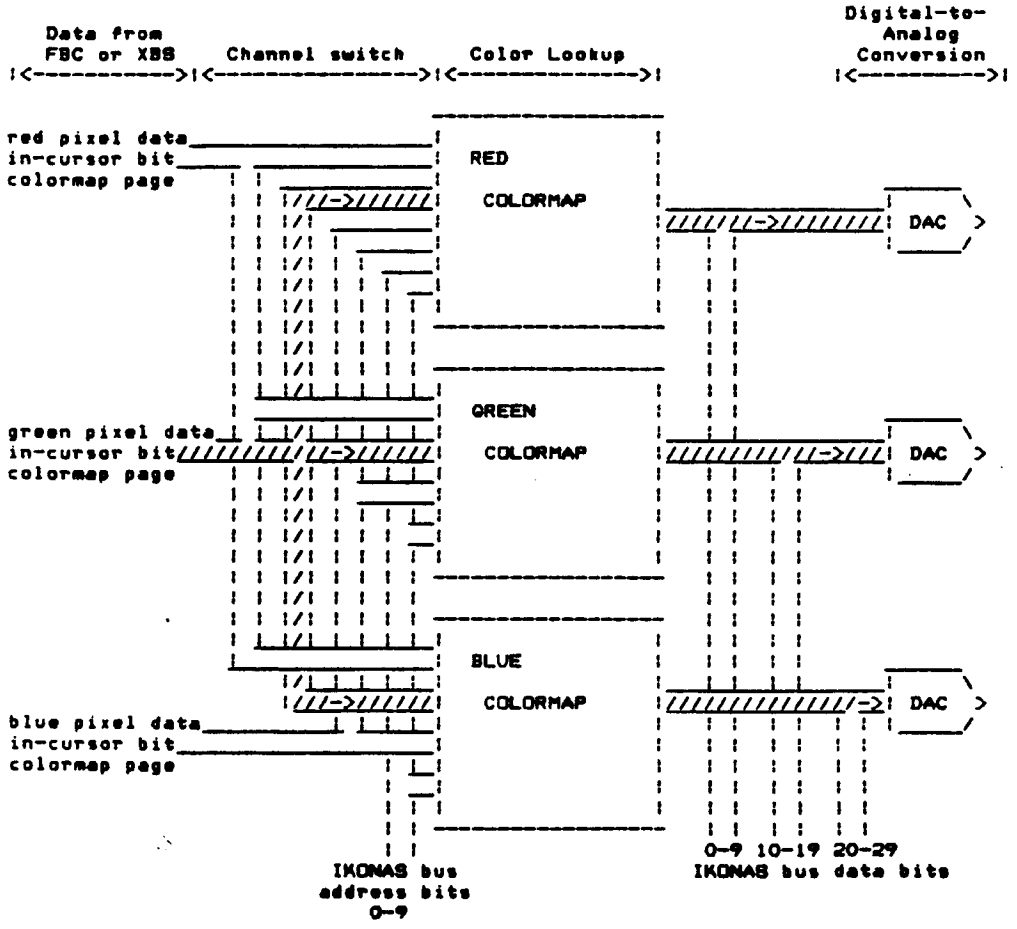
The channel switch has one output channel for each of the three colormaps; each output channel may be selected from any of the three input channels.

LUVO data routing - Full-color Operation



LUVO data routing - Pseudocolor Operation

(Example shows pseudocolor from the green channel)



Setting up the colormaps

The colormaps are initialized by writing for each input pixel value the output which the colormaps are to provide when that input is applied.

When a value is written to the LUVO, one entry in each of the three colormaps is modified with a single IKONAS bus write. When a value is read from the LUVO, one entry from each of the three colormaps is read with a single read.

The address used in the IKONAS operation is 20300$vvvv where vvvv is the input pixel value whose entries you want to read or modify.

Of the 32 IKONAS bus data bits, bits 0-9 correspond to the RED colormap; bits 10-19 correspond to the GREEN colormap; bits 20-29 correspond to the BLUE colormap; and bits 30-31 are reserved for future use.

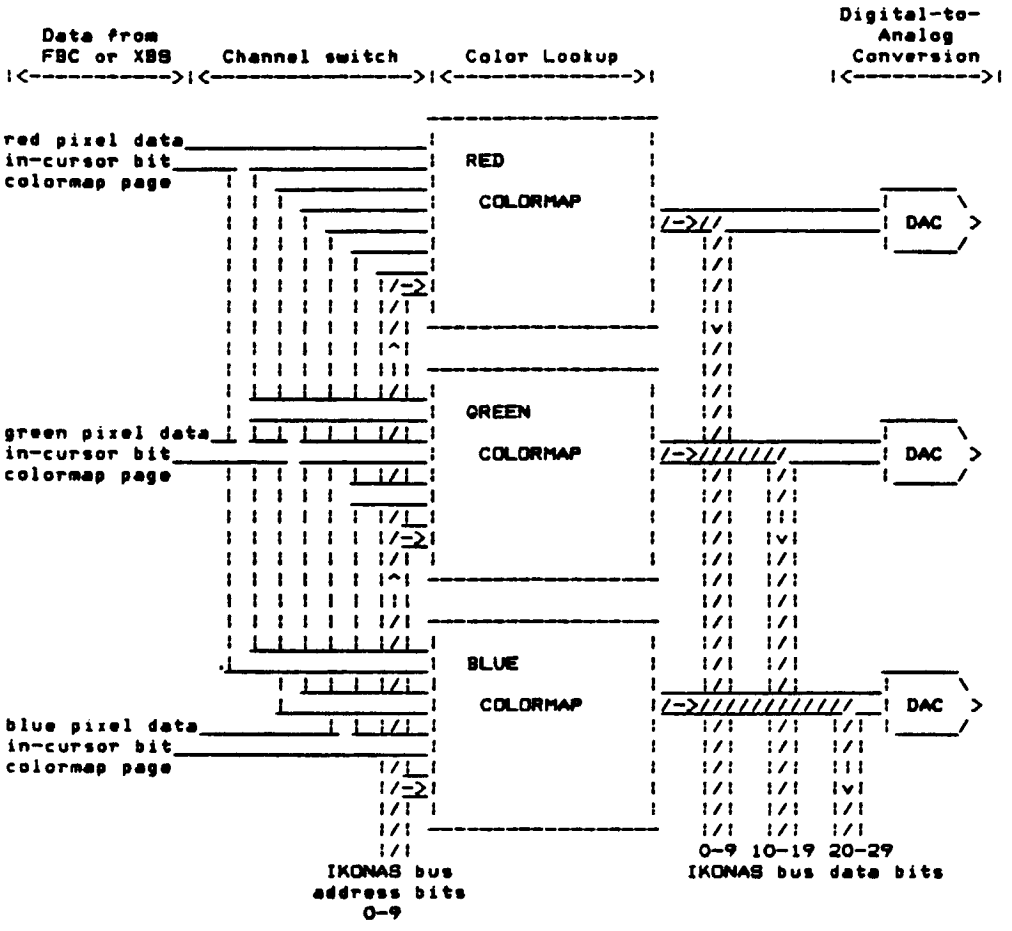
Example:

The address is 20300$50, the IKONAS bus data is 4402\57720. The entries in the colormaps corresponding to input pixel value 50 are:

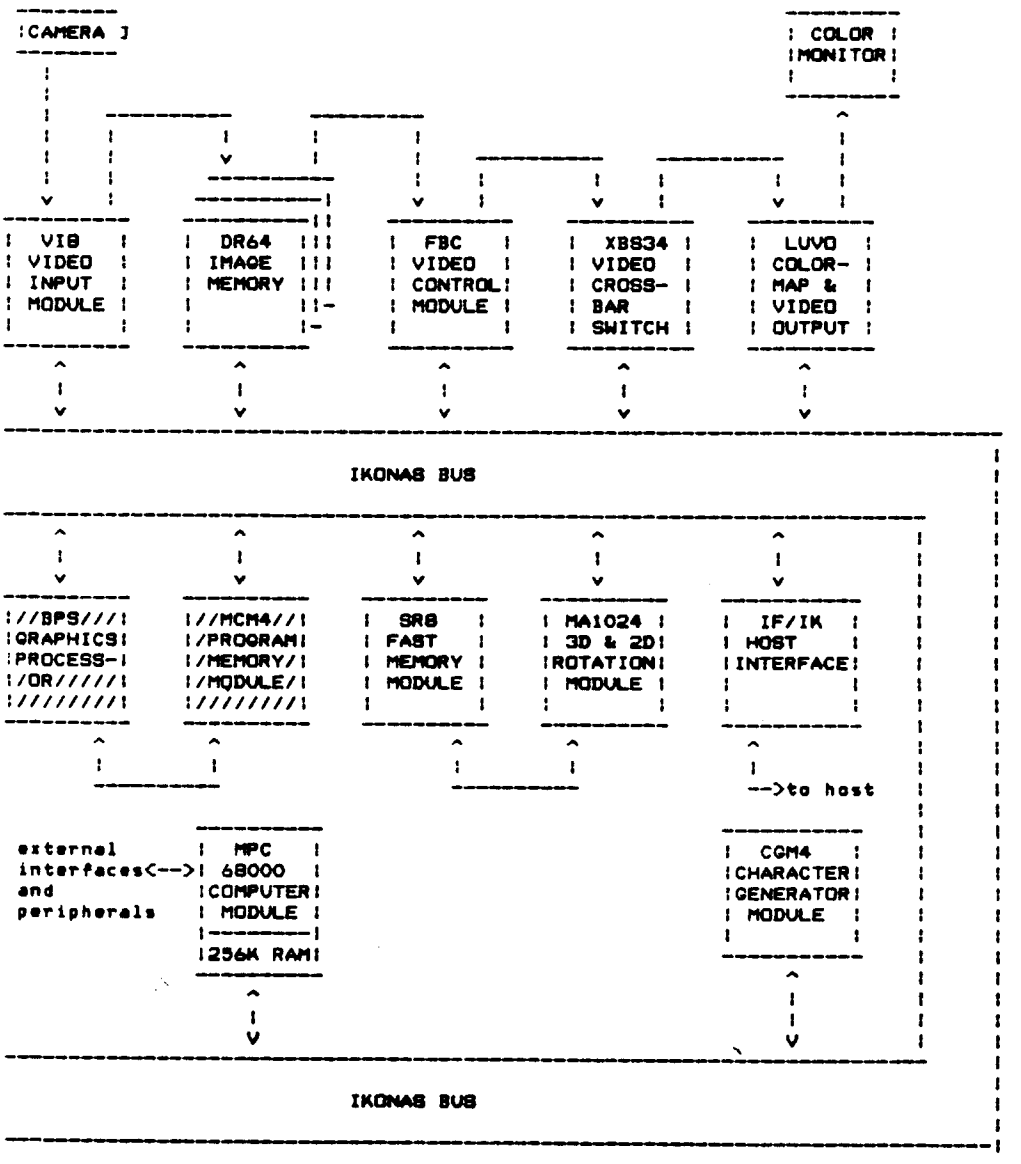
```
4402\57720 = 00001001000000100101111111010000
             |  blue  || green  ||  red   |
                220       227       1720
```

The entries in the colormaps range from 0 (dark) to 1777 (bright).

LUVO data routing - IKONAS bus access



THE IKONAS BPS32 GRAPHICS PROCESSOR



The BPS32 Graphics Processor

- o 32-bit processor for high precision
- o can simultaneously perform two 16-bit operations for high speed
- o 5 million instructions per second
- o writable control store - fully user-programmable
- o assembler is provided, written in standard FORTRAN
- o control store can be used as scratch memory
- o powerful instruction set allows overlapped arithmetic, branches, and I/O
- o fast multiply option allows 5 million multiplies per second

Microcode Development Tools

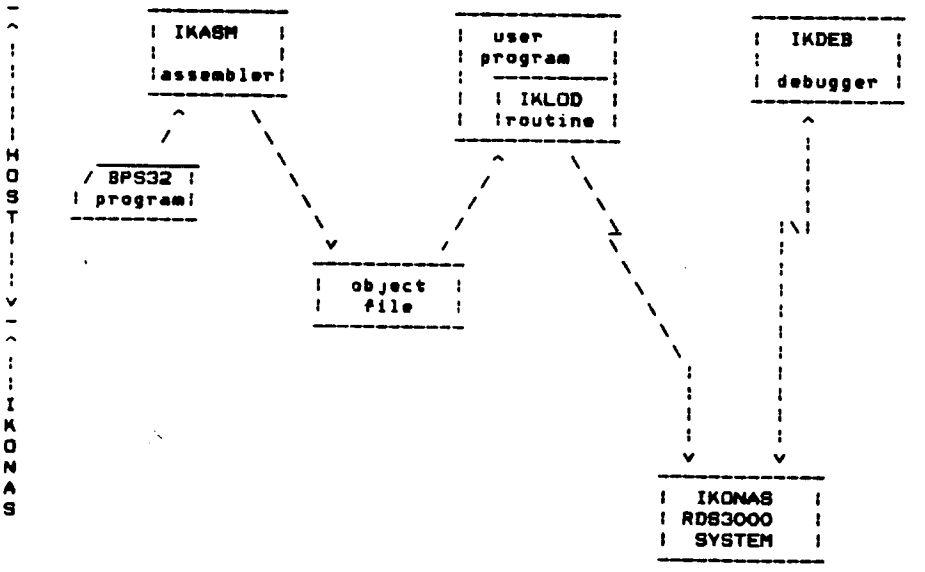
Microcode development tools consist of the BPS32 assembler (IKASM), the object-file loader (IKLOD), and the debugging tool (IKDEB).

All these programs run on the user's host computer.

IKASM converts a symbolic assembler-language file into an object file containing BPS32 machine code.

IKLOD is a subroutine which loads an IKASM object file into the IKONAS address space.

IKDEB allows you to start and start the processor, load files, and examine locations in the IKONAS address space.



IKASM Assembler Language Syntax

The IKASM assembler has the following syntax rules:

1. each statement corresponds to one BPS32 microcode instruction
2. one instruction per line
3. one line per instruction unless continued by slash in column 1 of next line
4. symbols are separated by spaces
5. symbols may be up to eight characters long
6. the first symbol may be followed by a colon, denoting a label; the value of the current location counter will be assigned to the label
7. fields may appear in any order
8. all characters after a semicolon are comments and are ignored by the assembler

The IKLOD subroutine

The object file output of IKASM is an unformatted FORTRAN file described in the IKONAS Loader Format and BPS32 Host Programming Guide documents.

To load an object file into the IKONAS, your FORTRAN program should do the following:

1. open the file for input
2. call the IKLOD subroutine, passing it the unit number of the file. On return your program should close the unit.

Example: CALL IKLOD(IUNIT)

IKASM Statements

IKASM statements fall into two categories:

1. Pseudo-operations: directives to the assembler which do not generate BPS32 instructions
2. Instructions: any line representing a BPS32 instruction.

The syntax of instructions and pseudo-operations is identical. Pseudo-operations are distinguished by the first symbol in the statement.

Pseudo-operations are:

1. assignment - explicitly assign a value to a symbol
2. ORG - set location counter
3. DEFAULT - establish default settings for fields
4. END - end the IKASM run

Instructions

Any IKASM statement which is not a pseudo-operation represents a BPS32 instruction.

In the hardware, BPS32 instruction words are composed of fields. Since the BPS32 is a microprogrammed machine, each field corresponds to some section of the hardware: a data path, a register, or a control signal.

An IKASM instruction is a specification of all the settings of all the fields.

Each field in the hardware microinstruction corresponds to a set of keywords in IKASM, one keyword for each possible setting of the field. For example, the hardware component called the 'R selector' can be loaded from the MDR or the MAR; the keywords 'RMAR' and 'RMDR' correspond to the appropriate settings of the 'R selector' field of the microinstruction.

In IKASM, you code the keywords which indicate the desired settings of the microinstruction fields. Eack keyword specifies simultaneously the field it refers to and the value for that field. You must avoid conflicts: two keywords which specify different value for the same field.

Symbols may be defined in your program, by the assignment pseudo-operation and by coding the symbol as the label of a statement. Symbols other than keywords may be used in instructions; they will be assembled in the data field of the instruction. The data field is a 16-bit field which holds immediate data and branch addresses.

If you code a symbol which is not defined (has not been defined in your program and is not a keyword), an error message is printed.

Examples of statements

```
LABEL:  RA1 RPS B2 BD JMPDF FRED  ; r2 <- r1+r2
```

The keywords are: RA1 - R operand from register 1; RPS - add R operand to S operand; B2 - S operand from register 2; BD - write result to register 2; JMPDF - branch to the value in the data field.

The symbol FRED, which must be defined elsewhere in the program, will be assembled into the data field as the branch target.

The symbol LABEL will be assigned a value equal to the location of this instruction. This label can be the target of branch instructions.

```
RA1 RA2 PR ALUMDR  ; MDR <- rxx
```

This statement contains a conflict: the R operand is specified as register 1 and register 2.

The Assignment Pseudo-operation

Example:

```
QBASE = 10 ;  define offset to base of queue
```

The assignment statement defines the first symbol, and gives it a value equal to the value of the symbol following the equals sign.

The symbol thus defined may be used as a data field in IKASM instructions.

A symbol need not be defined before it is referenced.

The DEFAULT Pseudo-operation

```
DEFAULT NANOP CCNOP LDNOP PR YD CARO SSO ALUBR SB MDRIKD MARIKA
```

The DEFAULT statement gives the default setting for fields in the BPS32 instruction.

The DEFAULT statement should be the first statement in a program.

When an IKASM instruction is assembled, all keywords appearing in the instruction will set the corresponding fields in the microinstruction; fields which are not set by a keyword in the instruction will assume the setting of the DEFAULT statement.

The DEFAULT statement saves a lot of typing, since you don't have to specify fields whose values equal the default.

The ORG Pseudo-operation

```
ORG value
ORG 256
ORG PGMSTRT
```

The ORG pseudo-instruction specifies a new value for the location counter. The location counter is automatically incremented by every IKASM instruction; the ORG statement may be used to skip a block of memory or start a program at a desired address.

The END Pseudo-operation

END

The END pseudo-operation is the last statement in an IKASM program.

BPS32 Organization

The BPS32 hardware consists of two components: the processor section and the sequencer section.

The processor section executes instructions, performs arithmetic and I/O, and saves results.

The sequencer section figures out what instructions to execute. During each instruction it computes the address of the next instruction.

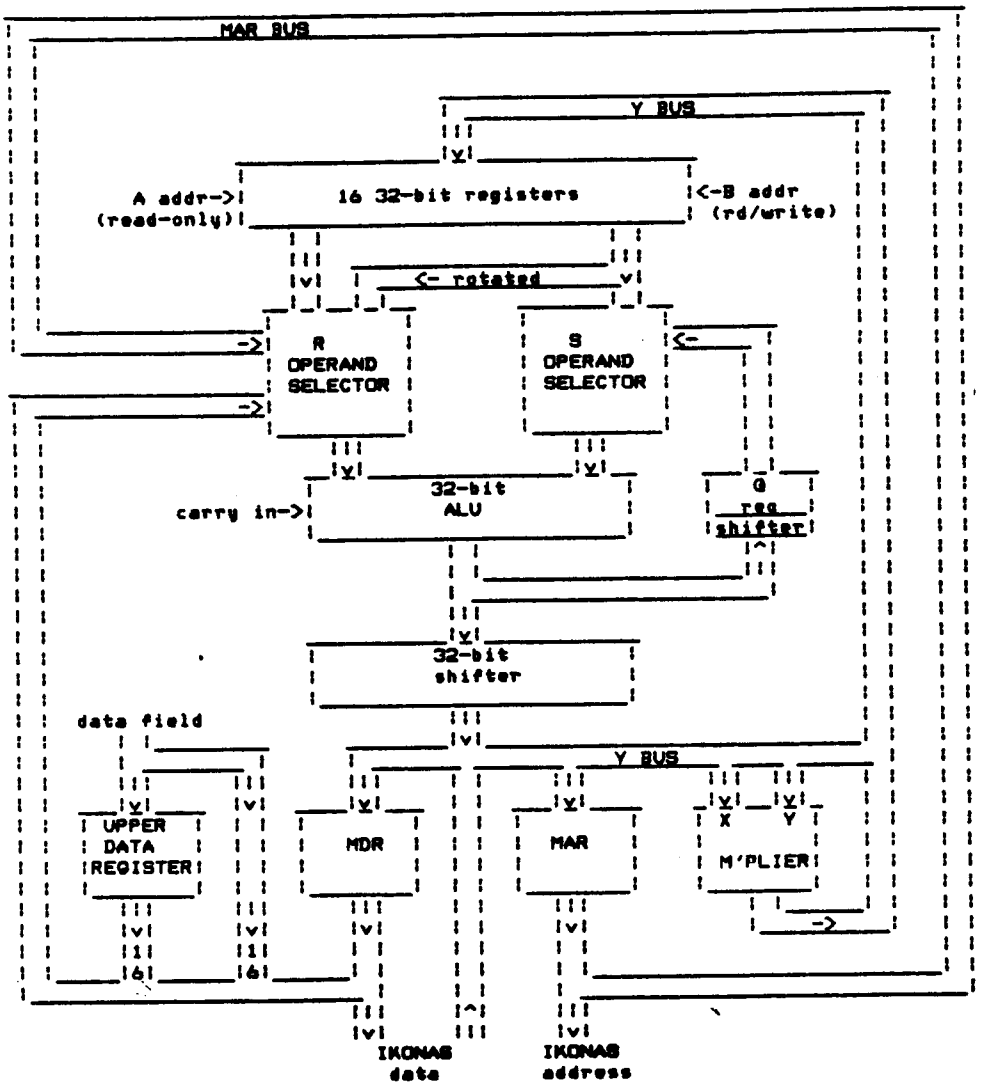
The BPS32 Processor Section

The processor section contains the following components:

1. 17 32-bit registers
2. a 32-bit ALU including shifter
3. a 32-bit Memory Data Register (MDR), to hold data to be written to the IKONAS bus
4. a 32-bit Memory Address Register (MAR), to hold the IKONAS bus address for IKONAS bus operations
5. (optional) a 16x16-bit two's complement multiplier

The connections within the processor secton are illustrated in the diagram on the next page.

The BPS32 Processor Section



The BPS32 Sequencer Section

The sequencer section consists of the following components:

1. a 16-bit program counter to hold the address of the next instruction
2. a 16-bit register to hold the data field of the previous instruction
3. a four-word by 16-bit stack to hold subroutine and loop addresses
4. a 16-bit loop counter
5. condition-code-testing logic

During each instruction, the microcode word and condition codes are examined to determine the address of the next instruction, which will be one of the following:

1. the current program counter plus one
2. the value in the data field of the current instruction (branch instruction)
3. the value at the top of the subroutine stack (subroutine return)
4. the value in the data field of the previous instruction (two-way branch)
5. the value on the IKONAS bus during the previous instruction (indirect branch)

Based on the microcode word, the sequencer may decrement the loop counter and set a flag if the count becomes zero.

The Sequencer Subroutine Stack

The subroutine stack in the sequencer has room to hold four addresses, to allow for subroutines nested up to four deep. If subroutines are nested more than four deep, the results are unpredictable.

A subroutine call consists of pushing the address of the current instruction plus one onto the stack (this will be the return address, the address of the instruction following the subroutine call), and branching to the subroutine. A return from subroutine consists of branching to the address contained in the top of the stack, and popping the stack. If the stack is empty when a return from subroutine is executed, the results are unpredictable.

The subroutine stack may be used to mark the top of loops by pushing the address of the top of the loop onto the stack. Special instructions may be used to branch to the address on the top of the stack without removing it from the stack.

You must be careful to remove any addresses pushed onto the stack, since no address except the one on the top of the stack can be referenced by the sequencer.

BPS32 QUICK REFERENCE CARD

NEXT ADDRESS SELECT

If CC false: I<-I+1 except as noted, CTR not incremented

NANOP	I<-I+1
JMPDF	I<-DF
JMPRDF	True: I<-DF
	False: I<-prev. DF
JMPIB	I<-previous IB
JSBxx	push I+1 & do JMPxx
NAPUSH	I<-I+1, push I+1
NALUP	True: I<-STK
	False: I<-I+1, pop STK
EXTLUP	I<-DF, pop STK
JMPCNT	I<-DF, incr. CTR
LUPCNT	I<-STK, incr. CTR
NARETN	I<-STK, pop STK
NAZERO	0

LOAD CONTROL

LDNOP	DF not loaded to any other register
LDCON	control reg<-DF
LDCNT	CTR<-DF
LDUDR	UDR<-DF

Y BUS CONTROL src & dst

ALUYD	
ALUMDR	ALUMAR
ALUMPX*	ALUMPY*
IKMPX*	IKMPY*
* wait 1 cycle for completion	
IKMDR	IKMAR
MPZYD	MPZMDR

IKONAS ADDRESS CONTROL

MARIKA	addr=MARO-23
MASHIKA	addr=MAR16-29$MARO-9

IKONAS BUS FUNCTIONS

IKRD	word mode read
IKWR	word mode write
HRESRD	HIRES read
LRESRD	LORES read
HRESWR	HIRES write
LRESWR	LORES write
HMASKWR	HIRES set write mask
LMASKWR	LORES set write mask
HSHADWR	HIRES set shade
LSHADWR	LORES set shade
WRSTRTMA	write & start MA1024
CONTMA	restart MA1024
STOPMA	stop MA1024

GLOSSARY

ALU	shifter output
BR	B register
CTR	loop counter
DF	data field
I	program counter
IB	IKONAS bus
MPX	MPLR X input
MPY	MPLR Y input
MPZ	MPLR output
Q	Q register
STK	stack
UDR	upper data register
~<operand>	one's complement
-<operand>	two's complement
^	AND
v	OR
v	EXCLUSIVE OR

COND CODE SELECT

Prefix N to negate condition

CCNOP	always true
CCCNTZ	CTR=0 after JMPCNT or LUPCNT
CCBLK	vertical drive pulse
CCMAAC	MA1024 busy
CCHOST	host request
CCOVR	ALU overflow
CCNEG	ALU negative
CCZERO	Y bus zero *
CCCAR	carry from ALU31
CCY16	Y-bus 15=1
CCGTZ	ALU >= 0 & Y bus = 0 *
*invalid during read from IB or MPLR	

SPECIAL FUNCTIONS

INCRS	BR<-BR+1+cry
SMTC	convert BR sign-magnitude to two's-complement
SLNML	single length normalize
DLNML	double length normalize
UMPY	unsigned multiply
TCMPY	signed multiply
TCMPL	end of signed multiply
TCDIV	divide
TCDIVC	end of divide

UNSIGNED MULTIPLY - Rx * Ry

Product = Rx\Q

```
ALUZ Bx BD
RAx PR QD LDCNT 1-32.
UMPY RAy Bx LSO JMPCNT NCCCNTZ .
```

TWO'S COMPLEMENT MULTIPLY - Rx * Ry

```
ALUZ Bx BD
RAx PR QD LDCNT 1-31.
TCMPY RAy Bx CARO LSO JMPCNT NCCCNTZ .
TCMPL RAy Bx CARZ LSO
```

TWO'S COMPLEMENT DIVISION

Ry\Q = dividend (64 bits)
Rx = divisor (32 bits)
Q = quotient, n bits; Ry = remainder
(n = degree of precision wanted)

```
DLNML RAx By LR LDCNT -1-n
TCDIV RAx By LR CARZ JMPCNT NCCCNTZ .
TCDIVC RAx By SS1 CARZ
```

ARITHMETIC COMPARISONS

Test	Fctn	CC (32 bits/16 bits)
R >= S	RMS	NCCNEG/NCCY16
R > S	SMR	CCNEG/CCY16
R <= S	SMR	NCCNEG/NCCY16
R < S	RMS	CCNEG/CCY16
R = S	REOS	CCZERO

LOGICAL COMPARISONS

R >= S	RMS	CCCAR
R > S	SMR	NCCCAR
R <= S	SMR	CCCAR
R < S	RMS	NCCCAR

R SELECTOR

RAx	REGx
RIMM	UDR\DF *
RLIMM	O\DF *
RMDR	MDR
RLMDR	O\MDRO-15
RMAR	MAR
RLMAR	O\MARO-15
RBHR	BRO-15\BR16-31
RBHS	O\BR16-31
RBBR	BRO-7,24-31\BR8-23
RBBS	BR24-31\BR8-23

* not valid during IKWR

S SELECTOR

Bx	REGx	SQ	Q

ALU OPERATIONS

SMR	S-R-1+cry	ALUZ	O
RMS	R-S-1+cry	NRAS	(~R)^S
RPS	R+S+cry	RENS	~(RvS)
PS	S+cry	REOS	RvS
MS	-S-1+cry	RAS	R^S
PR	R+cry	RNRS	~(RvS)
MR	-R-1+cry	RNAS	~(R^S)
		RORS	RvS

ALU SHIFT/DEST

BD = write BR, QD = write Q

BD	no shift
QD	no shift
YD	no shift
BDQD	no shift (write both)
RAFBD	single right arithmetic
RLFBD	single right logical
LAFBD	single left arithmetic
LLFBD	single left logical
RAFLQBD	double right arithmetic
RLFLQBD	double right logical
LAFLQBD	double left arithmetic
LLFLQBD	double left logical
LLQYD	Q left logical
RLQYD	Q right logical
LSXBD	ALU16-31<-ALU15

Note: logical = shift all bits; arith = leave ALU31 unchanged; single = shift ALU only; double = shift ALU(hi) & Q(lo)

SHIFT CONTROL (shifts only)

Single	Double	
SSO	LSO	Shift in 0
SS1	LS1	Shift in 1
SR	LR	Rotate

CARRY IN CONTROL

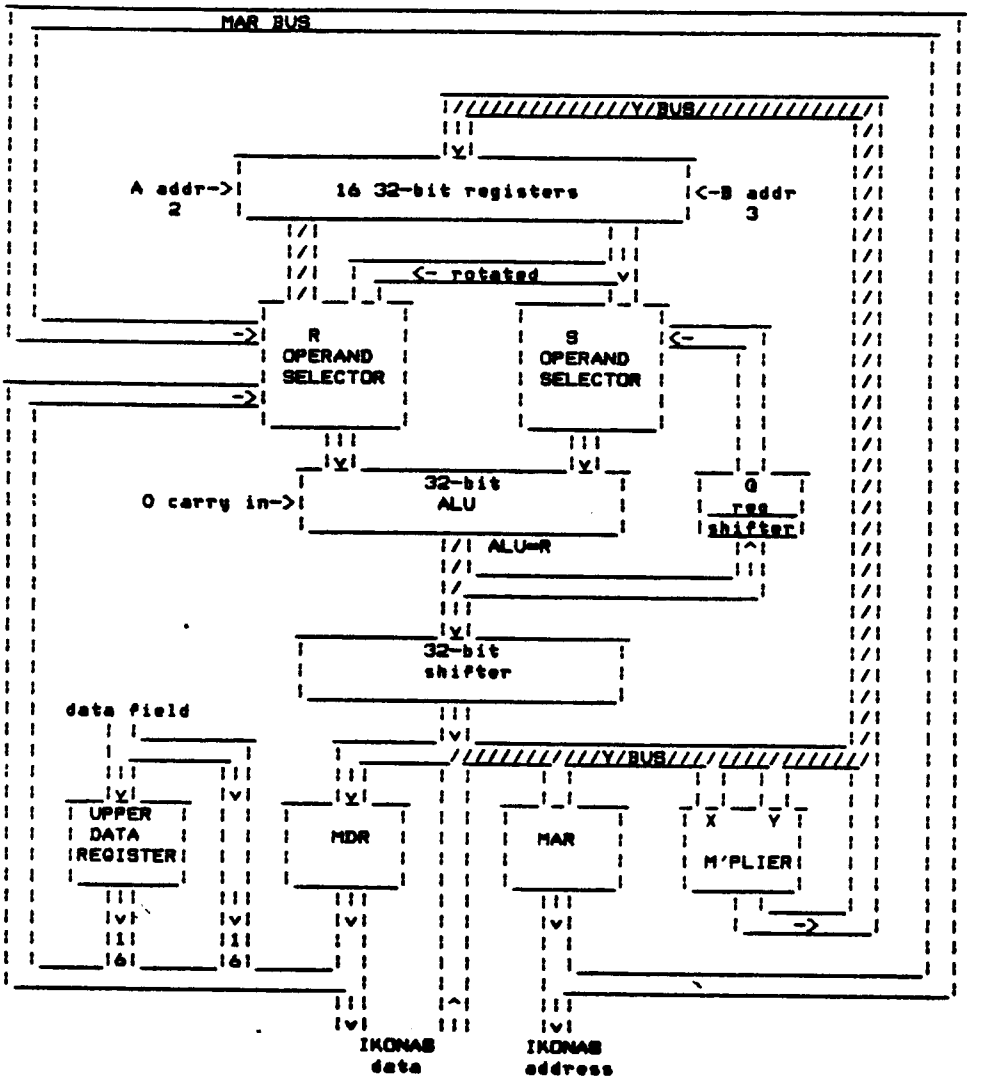
	bit0	bit16
CARO	0	C15
CAR1	1	C15
CAR31	C31	C15
CARN31	~C31	C15
CARZ	CCZ	C15
CARHO	0	0
CARH1	1	1

C15 = ALU carry from 15
C31 = ALU carry from 31
CCZ = 1 if ALU = 0

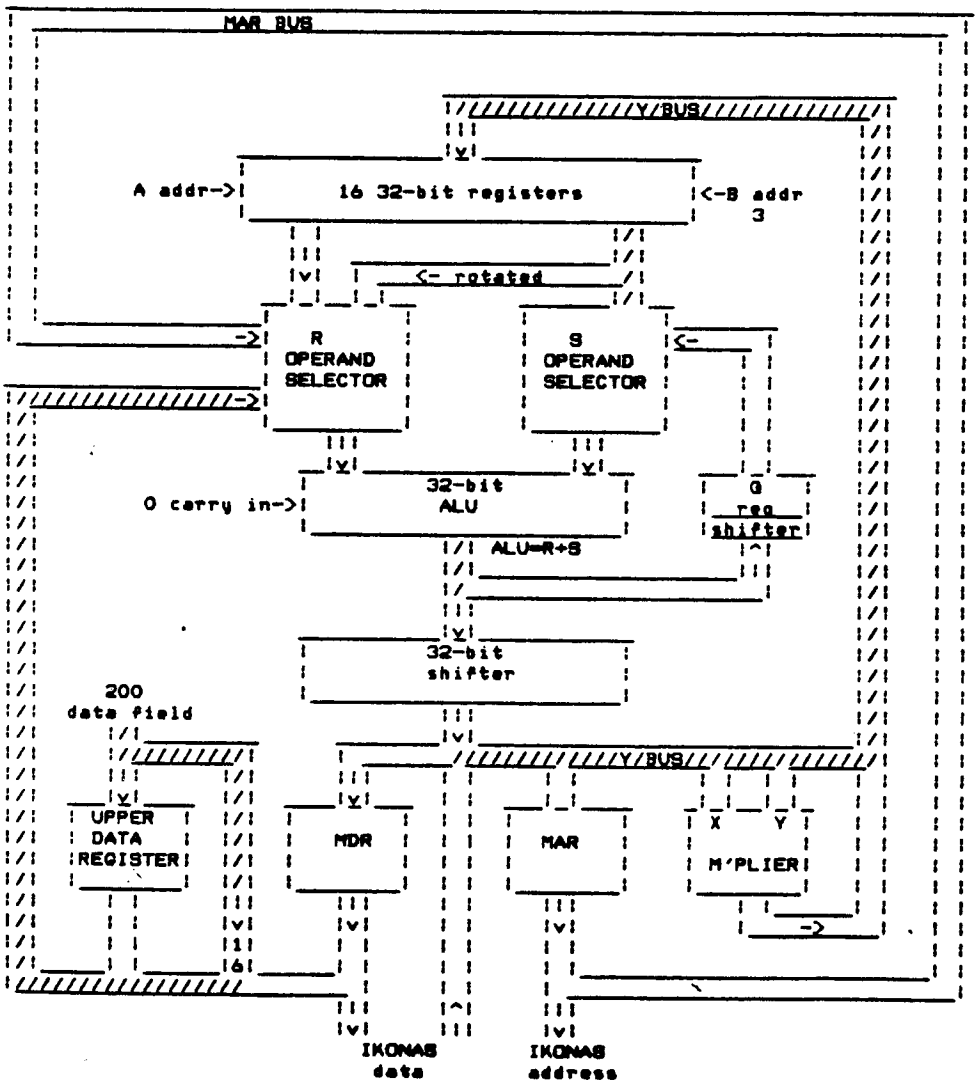
SAMPLE DEFAULT STATEMENT

```
DEFAULT NANOP CCNOP LDNOP PR YD CARO SSO ALUYD SB MDRIKD MARIKA
```

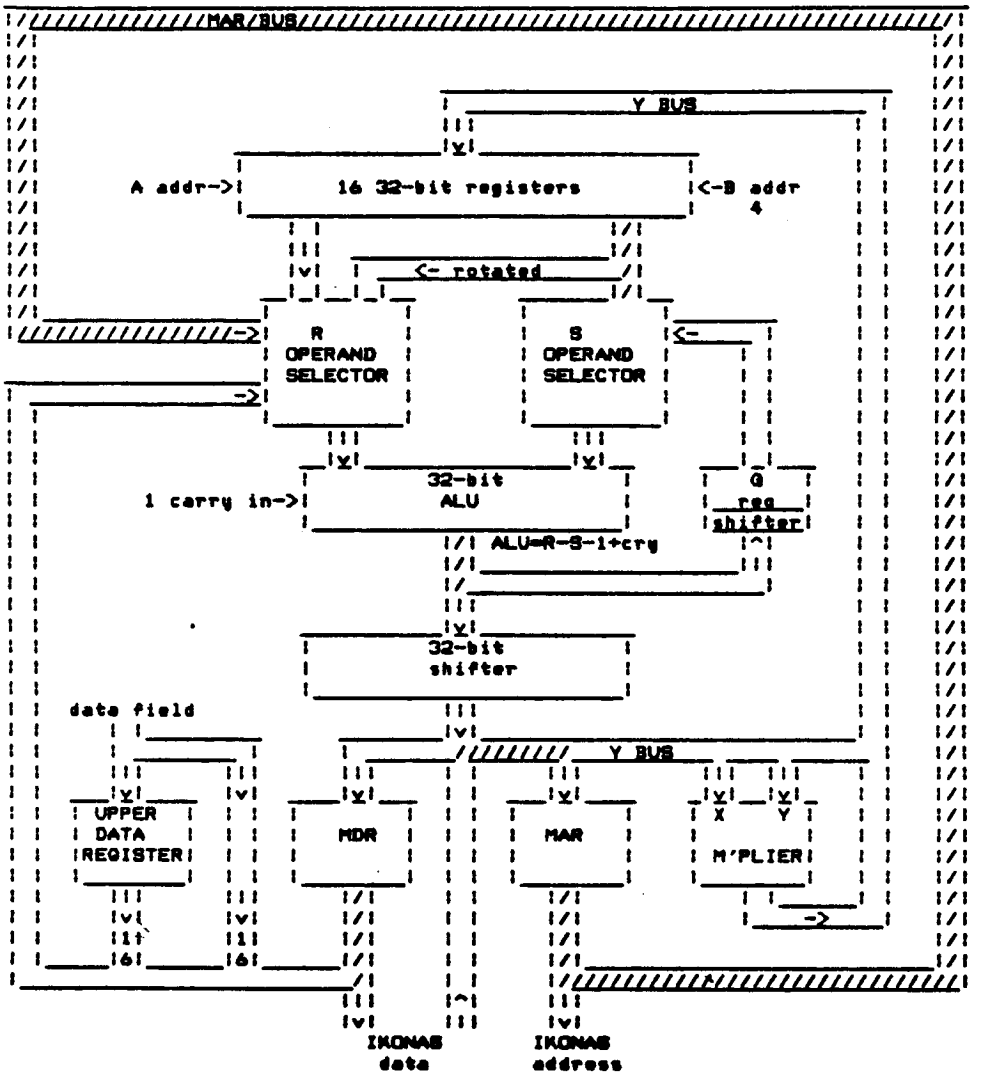
Operation: Move register 2 to register 3
nstruction: RA2 PR B3 BD



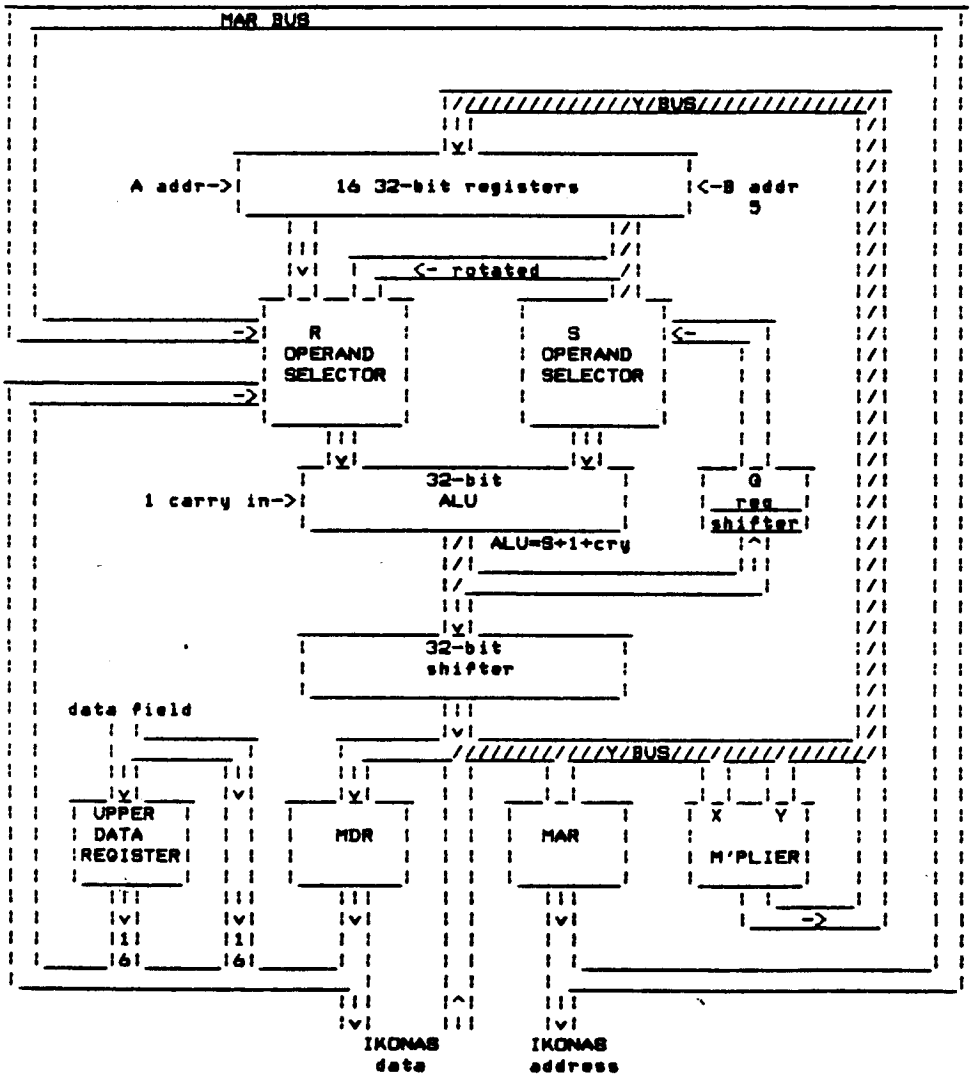
Operation: register 3 <- register 3 + 200
Instruction: RLIMM 200 RPS B3 BD



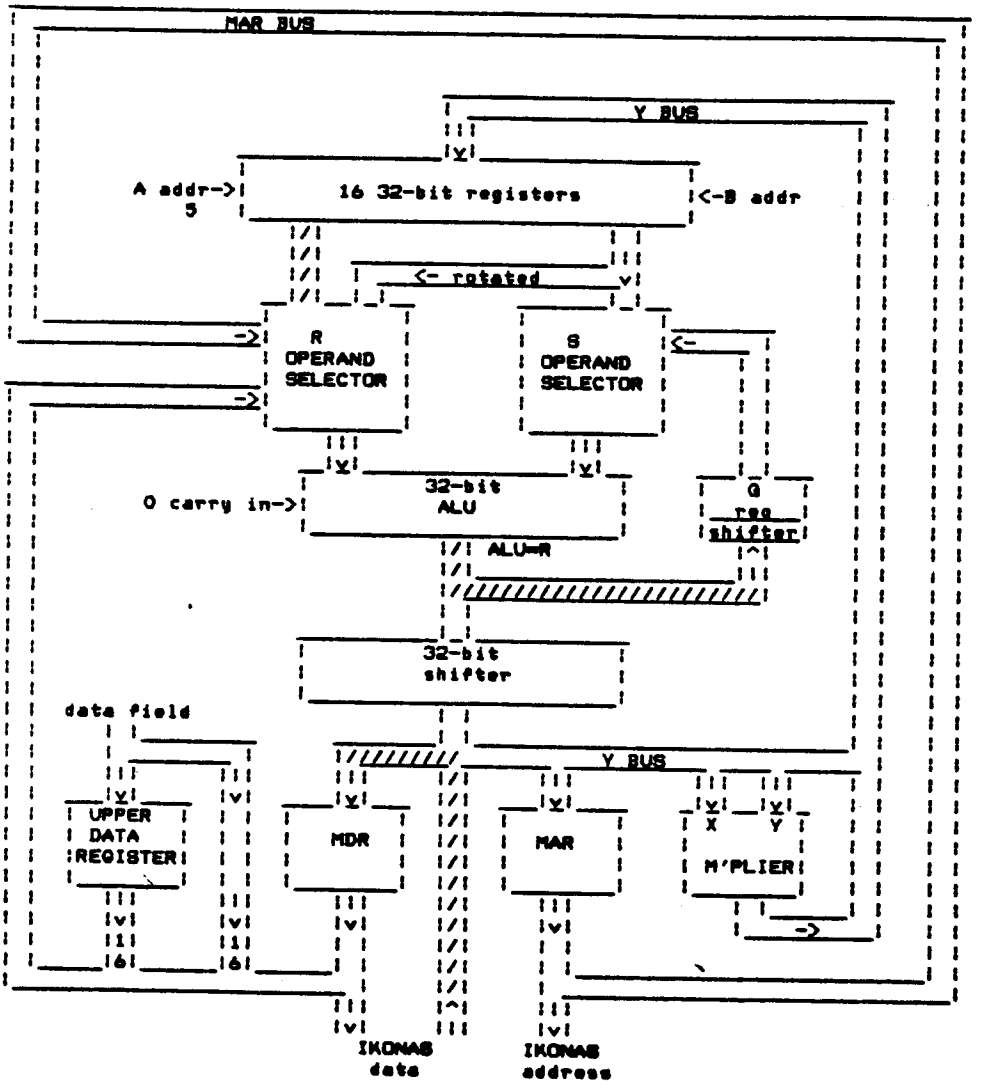
Operation: Write MDR to the IKONAS bus at addr in MAR; subtract R4 from MAR
truction: IKWR RMAR RMS B4 CAR1



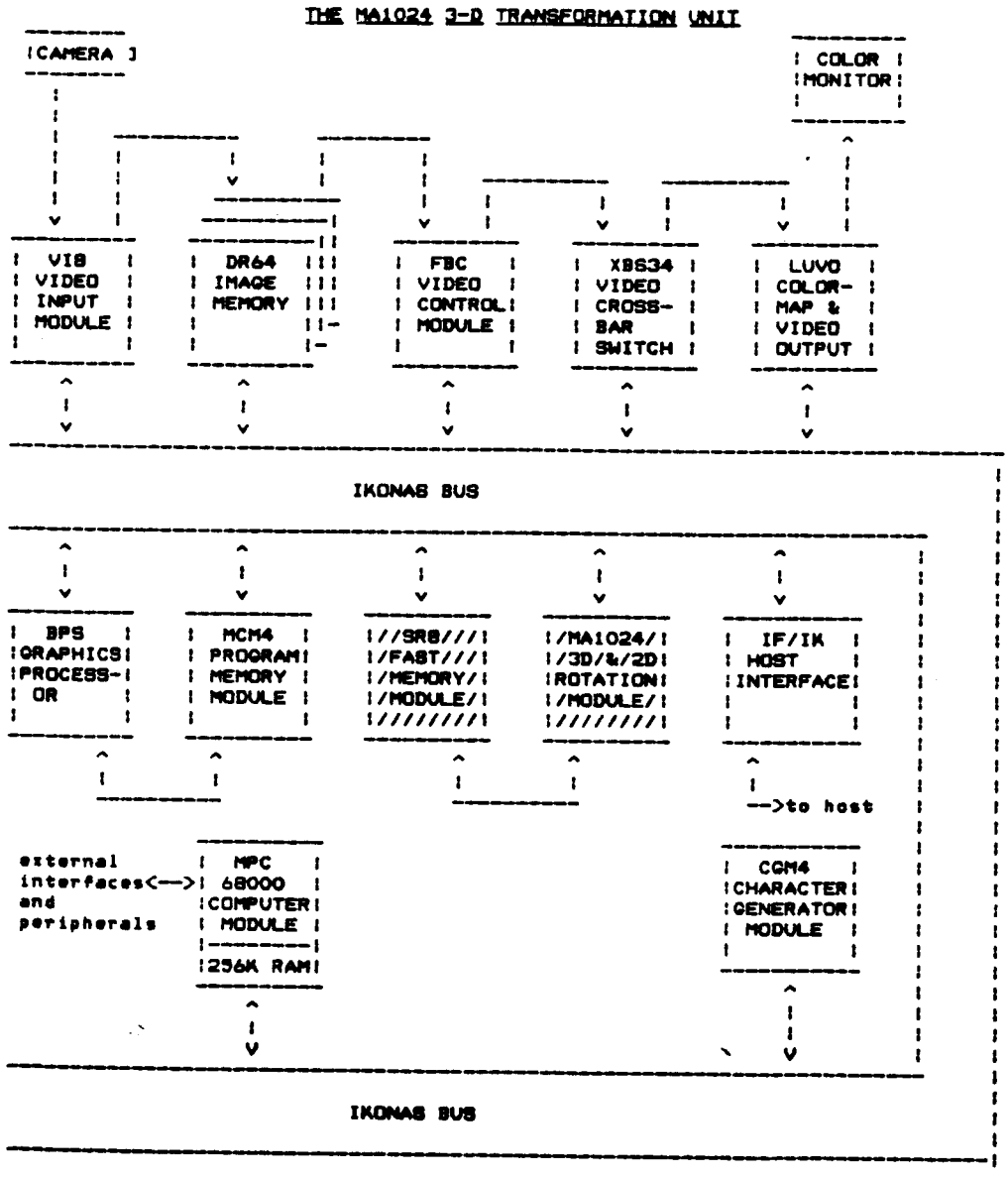
Operation: Increment R5 by 2
Instruction: B5 INCRS CAR1



Operation: Read IKONAS bus into MDR; load Q from R5
(second half of dynamic memory read)
Instruction: IKMDR RA5 PR QD



THE MA1024 3-D TRANSFORMATION UNIT



The MA1024 3-D Transformation Unit

- Performs 3-D rotation of vector endpoints and polygon vertices
- Fast - 300,000 transformations per second
- Allows storage of 64 rotation angles simultaneously
- Option allows rotation with perspective
- Calculates dot-product for lighting simulation
- Calculates cross-product of vectors for normal-vector computation
- Fully user-programmable with writable control store
- Full parallel processing - does not require access to IKONAS bus during operation
- Uses up to seven SR8 dual-port memories for data input and output

MA1024 Inputs and Outputs

- Input points can be 3-D [x y z] or homogeneous [x y z w]
- Output is same format as input
- 4x4 coefficient matrix controls rotation

The MA1024 accepts a list of input points, matrix multiplies them by a matrix representing the desired rotation (the coefficient matrix), and stores the resulting output points.

The MA1024 input and output points, which are stored in the SR8 memory, represent points in three-space or points in homogeneous coordinates. A point in three-space is a vector [x y z]; a point in homogeneous coordinates is a vector [x y z w].

The MA1024 always uses homogeneous coordinates in its multiplication, so a point in three-space is extended by setting w=1: [x y z 1].

The output points produced by the MA1024 are the same format as the inputs: either points in three-space or points in homogeneous coordinates. Input points in three-space carry with them a tag field which is copied unchanged to the output.

The examples below will all use input points in three-space.

The Rotation Operation in the MA1024

Each input point to the MA1024 is transformed to produce a corresponding output point by a matrix multiplication defined as follows:

$$[Xo\ Yo\ Zo\ 1] = [Xi\ Yi\ Zi\ 1] \begin{vmatrix} Cxx & Cxy & Cxz & 0 \\ Cyx & Cyy & Cyz & 0 \\ Czx & Czy & Czz & 0 \\ Tx & Ty & Tz & 1 \end{vmatrix}$$

[Xo Yo Zo] is the output point;

[Xi Yi Zi] is the input point;

Cij is the contribution of input coordinate i to output coordinate j;

Tk is the amount of translation (lateral movement) in coordinate k.

Computation of the coordinate matrix is the responsibility of the user. Helpful formulas can be found in Rogers & Adams, Mathematical Elements for Computer Graphics, pp. 47-48, 54-55, 207-208; and Newman & Sproull, Principles of Interactive Computer Graphics, pp. 333-352.

Data Formats

All numbers in the MA1024 are 16-bit two's-complement fractions ranging in value from -1.0 to +0.9999. The most-significant bit of such a number (bit 15) is the sign bit; bits 0-14 are the fractional part. The implied binary point lies between bits 14 and 15.

An [x y z] point or an [x y z w] point requires four 16-bit numbers, or two 32-bit IKONAS data words. The two words for a point occupy consecutive locations in the SR8 memory, as follows:

For [x y z] points:

Bits	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
first word	y coordinate	x coordinate
second word	E tag field	z coordinate

E (bit 30) is the end-of-list bit, set in the last point to be rotated.

For [x y z w] points:

Bits	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
first word	y coordinate	x coordinate
second word	w coordinate	z coordinate

Coefficient Matrix Format

The coefficient matrix to be used for a rotation is stored in the coefficient memory of the MA1024.

The MA1024 can hold up to 64 coefficient matrices simultaneously, though only one may be used for a single list of points.

Each coefficient matrix occupies 16 consecutive words of the coefficient memory, as follows (refer to the diagram above for the meaning of Cxx etc.):

Word 0 Cxx
Word 1 Cxy
Word 2 Cxx
Word 3 Tx
Word 4 Cyx
Word 5 Cyy
Word 6 Cyz
Word 7 Ty
Word 8 Czx
Word 9 Czy
Word 10 Czz
Word 11 Tz
Word 12 0
Word 13 0
Word 14 0 (used by the perspective division option)
Word 15 .9999

MA1024 Operating Sequence

The steps to use of the MA1024 are the following:

1. Load the correct MA1024 microcode routine into the program memory. Microcode routines to perform operations described in this document are supplied by IKONAS and may be loaded using the IKLOD subroutine. The routine to perform rotation is named MMEOL.

2. Load the coefficient matrix for the rotation into the coefficient memory.

3. Load the points to be transformed into the SR8. Be sure E (bit 30) is set in the last word.

4. Load the correct values into the MA1024 control registers for the following:

    1. MA1024 program address

    2. Coefficient matrix offset

    3. Input data address

    4. Output data address

    See below for the formats of these control registers.

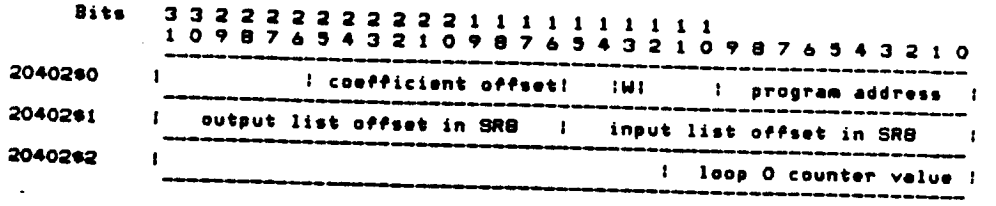
5. Start the MA1024 by storing 0 into address 20402$3 using a function code of 25.

Note

The procedure described above is the proper sequence for using the MA1024 to perform rotation. It does not use all the features of the MA1024. If you want to write custom MA1024 microcode, you should learn about the full power of the MA1024 by reading the MA1024 Programming Guide.

MA1024 Control Registers

The registers described below control the operation of the MA1024. All the registers may be accessed from the IKONAS bus in the address range 20402$0-20402$2.



program address is the offset to the MA1024 microprogram to be used. For the IKONAS rotation program MMEOL, this value will be 0. For custom microcode, the value will be the same as the value of the ORG statement in the IKASM assembly of the microcode.

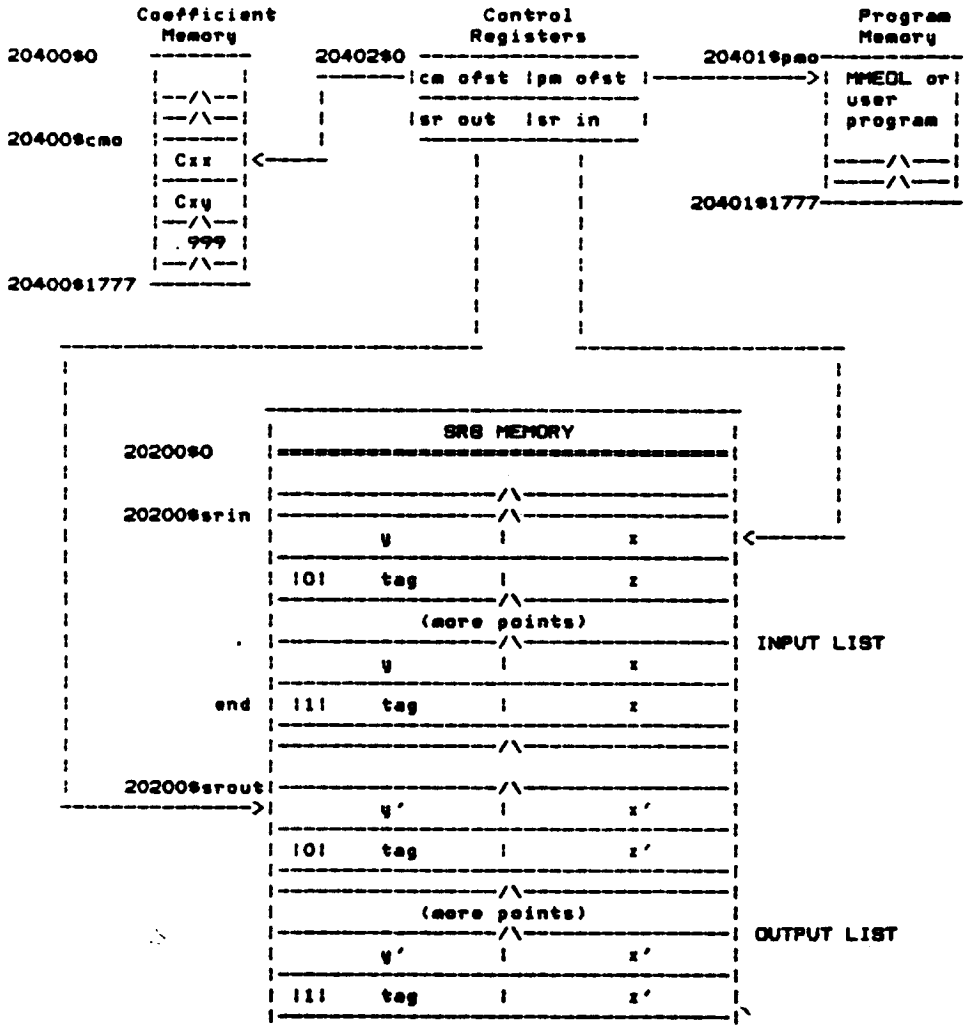
W (bit 13) should be set to 0 for MMEOL. When this bit is 1, the MA1024 will expect input points in homogeneous coordinates: [x y z w]. IKONAS does not supply microcode which requires W set to 1.

Coefficient address is the offset to the coefficient matrix to be used. If this offset is specified as cccc, the coefficient matrix must start at 20400$cccc.

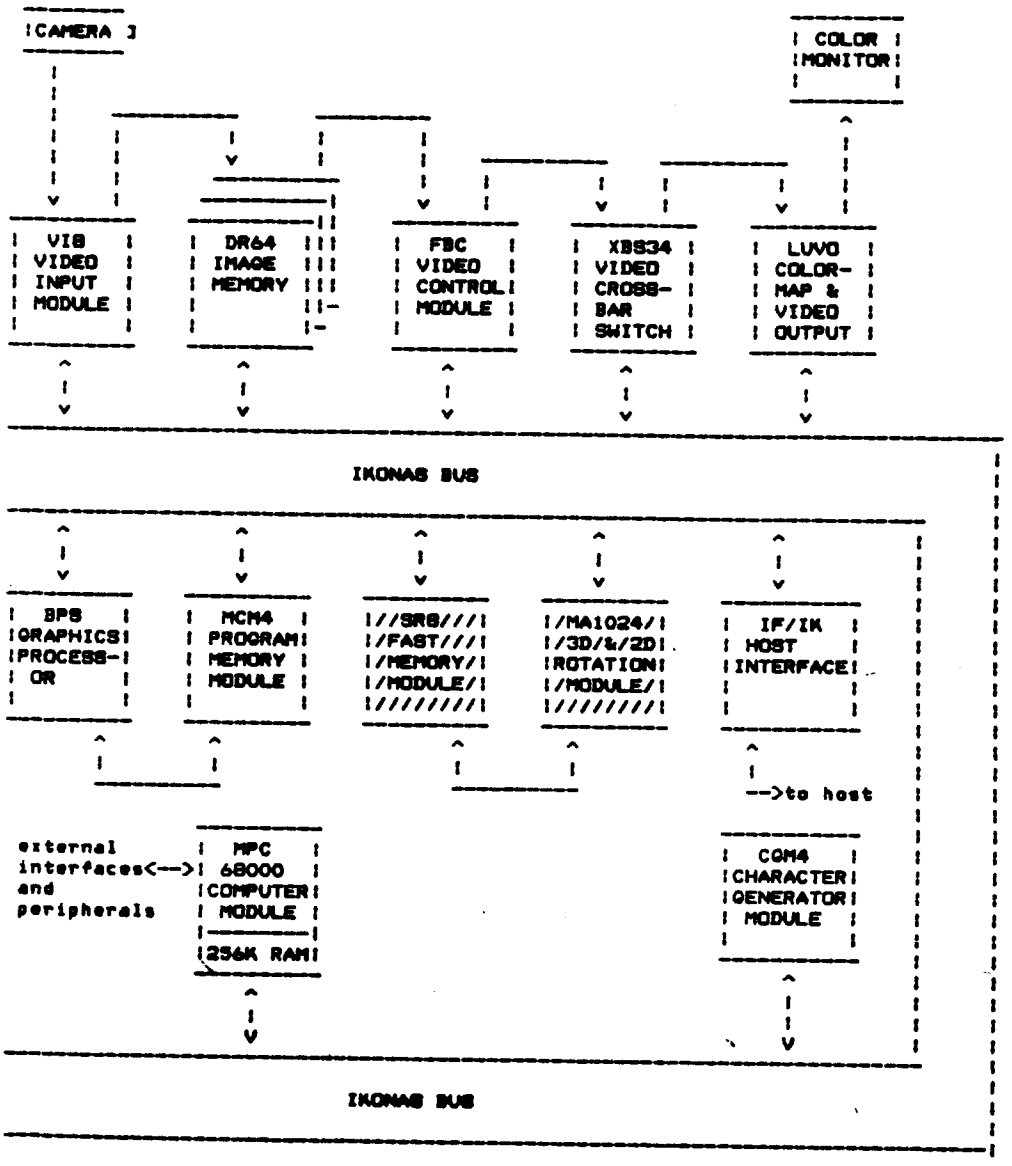
Input and output offsets are the offsets to the input and output areas in the SR8. If one of these offsets is specified as xxxx, the area at 20200$xxxx will be used for input or output. The last point in the input list must have the end-of-list (E) bit set.

Loop 0 counter value is a value which may be used by user microcode. IKONAS microcode does not use the loop 0 counter, which may be left uninitialized.

Diagram of MA1024 Control Flow



THE MA1024 3-D TRANSFORMATION UNIT WITH CLIPPING SUPPORT



The MA1024 3-D Transformation Unit with Clipping Support

- Performs 3-D rotation of vector endpoints and polygon vertices
- Fast - 300,000 transformations per second
- Allows storage of 64 rotation angles simultaneously
- Allows rotation with perspective
- High-precision perspective divider allows perspective rendering of complex scenes
- Clipping assist feature detects and flags points out of view in six clipping planes - allows savings of 50-80 percent of time spent in clipping
- Calculates dot-product for lighting simulation
- Calculates cross-product of vectors for normal-vector computation
- Fully user-programmable with writable control store
- Full parallel processing - does not require access to IKONAS bus during operation
- Uses up to seven SRB dual-port memories for data input and output

MA1024 Inputs and Outputs

- Input points can be 3-D [x y z] or homogeneous [x y z w]
- Output is same format as input
- 4x4 coefficient matrix controls rotation

The MA1024 accepts a list of input points, matrix multiplies them by a matrix representing the desired rotation (the coefficient matrix), and stores the resulting output points.

The MA1024 input and output points, which are stored in the SRB memory, represent points in three-space or points in homogeneous coordinates. A point in three-space is a vector [x y z]; a point in homogeneous coordinates is a vector [x y z w].

The MA1024 always uses homogeneous coordinates in its multiplication, so a point in three-space is extended by setting w=1: [x y z 1].

The output points produced by the MA1024 are the same format as the inputs: either points in three-space or points in homogeneous coordinates. Input points in three-space carry with them a tag field which is copied unchanged to the output.

The examples below will all use input points in three-space.

The Rotation Operation in the MA1024

Each input point to the MA1024 is transformed to produce a corresponding output point by a matrix multiplication defined as follows:

$$[Xo\ Yo\ Zo\ 1] = [Xi\ Yi\ Zi\ 1] \begin{vmatrix} Cxx & Cxy & Cxz & 0 \\ Cyx & Cyy & Cyz & 0 \\ Czx & Czy & Czz & 0 \\ Tx & Ty & Tz & 1 \end{vmatrix}$$

[Xo Yo Zo] is the output point;

[Xi Yi Zi] is the input point;

C_{ij} is the contribution of input coordinate i to output coordinate j;

T_k is the amount of translation (lateral movement) in coordinate k.

Computation of the coordinate matrix is the responsibility of the user. Helpful formulas can be found in Rogers & Adams, Mathematical Elements for Computer Graphics, pp. 47-48, 54-55, 207-208; and Newman & Sproull, Principles of Interactive Computer Graphics, pp. 333-352.

Data Formats

All numbers in the MA1024 are 16-bit two's-complement fractions ranging in value from -1.0 to +0.9999. The most-significant bit of such a number (bit 15) is the sign bit; bits 0-14 are the fractional part. The implied binary point lies between bits 14 and 15.

An [x y z] point or an [x y z w] point requires four 16-bit numbers, or two 32-bit IKONAS data words. The two words for a point occupy consecutive locations in the SRB memory, as follows:

For [x y z] points:

Bits	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
first word	y coordinate	x coordinate
second word	E clip flags tag field	z coordinate

E (bit 30) is the end-of-list bit, set in the last point to be rotated.

The clip flags are set by the clipping assist feature, if it is enabled. More about this feature under 'perspective transformation'

For [x y z w] points:

Bits	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
first word	y coordinate	x coordinate
second word	w coordinate	z coordinate

Coefficient Matrix Format

The coefficient matrix to be used for a rotation is stored in the coefficient memory of the MA1024.

The MA1024 can hold up to 64 coefficient matrices simultaneously, though only one may be used for a single list of points.

Each coefficient matrix occupies 16 consecutive words of the coefficient memory, as follows (refer to the diagram above for the meaning of Cxx etc.):

Word 0 Cxx
Word 1 Cxy
Word 2 Cxx
Word 3 Tx
Word 4 Cyx
Word 5 Cyy
Word 6 Cyz
Word 7 Ty
Word 8 Czx
Word 9 Czy
Word 10 Czz
Word 11 Tz
Word 12 0
Word 13 0
Word 14 0 (used by the perspective division option)
Word 15 .9999

The Perspective Transformation and Clipping

The MA1024 allows perspective transformation by dividing the coordinates by a value dependent upon the distance of the object from the observer.

The clipping assist feature of the MA1024 may be used to compare the transformed coordinates of each point against six clipping planes; the MA1024 will set six flags, one for each clipping plane, to show whether the point is beyond the plane or not.

The steps in perspective transformation with clipping are:

1. The input points are defined to be in 'world coordinates', with values ranging from -1 to +.9999.

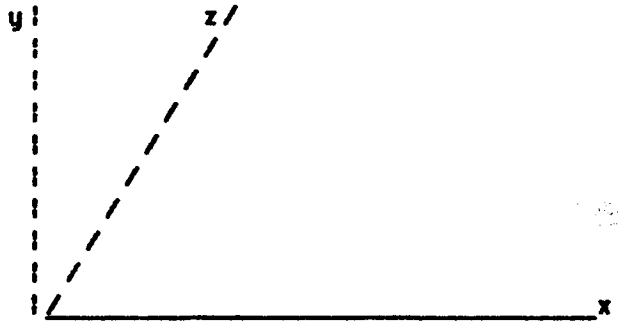
2. Use the MA1024 to rotate the input points and simultaneously convert then to 'clipping coordinates'. Clipping coordinates are similar to world coordinates, except that w (the perspective coordinate) is computed, and x, y, and w are scaled to allow clipping of x and y against w. Examples of this transformation follow. During this step the clipping assist feature is activated to cause the MA1024 to set flag bits if a coordinate is outside the clipping planes. The z-coordinate is left in world coordinates, except that the distance of the near clipping plane is subtracted from it.

3. Use the BPS32 to handle the clipped points. This may involve discarding them or replacing clipped vectors.

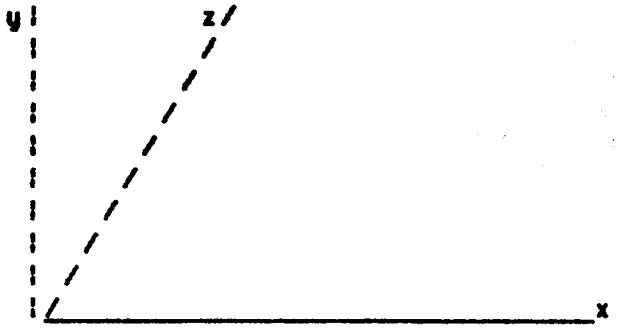
4. Use the MA1024 again the perform the perspective division by w. Since w is not stored in the point list, it is recalculated from z.

Perspective and Clipping Sequence

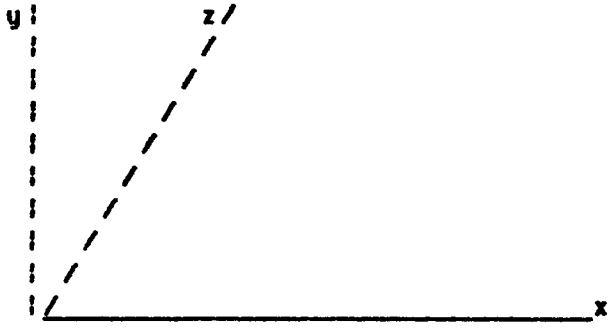
Initial World Coordinates



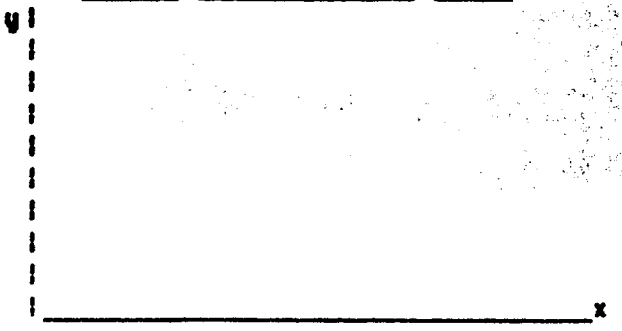
Clipping Coordinates



Out-of-bounds Points Clipped by the BPS32



Final Perspective View



The Conversion from World Coordinates to Clipping Coordinates

The MA1024's conversion from world coordinates to clipping coordinates consists of three components:

1. Calculation of the transformed x, y, and z coordinates in the same manner as described above for simple rotation
2. Calculation of w - the perspective coordinate - as a function of the transformed z and the desired viewing pyramid
3. Adjustment of z by subtracting the distance to the near clipping plane

W (the perspective coordinate) is significant because the final operation in the perspective transformation is division of x and y by w. W represents the apparent distance of the object from the observer. If the transformation is to simulate the view of a telescopic lens, i. e. if the object is to be made to appear close to the observer, w will be made a fraction of z. If the transformation is to give a wide-angle view, i. e. the object is to appear far away, x and y will be scaled down accordingly. X and y are scaled down instead of scaling up w, since it is impossible to scale up using the MA1024. Since x and y will be divided by w later, scaling down x and y is equivalent to scaling up w.

Z is the absolute distance from the object to the observer. Z is adjusted by subtracting a constant, which represents the nearest object which may be viewed.

The Clipping Assist Feature

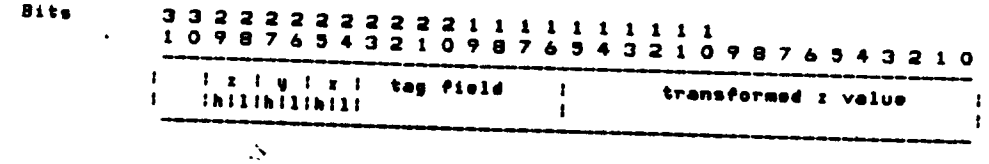
Conversion to clipping coordinates defines a viewing pyramid within which all points are visible, and outside of which all points are off the screen. This viewing pyramid is defined by the equations:

$$-w < x < +w$$
$$-w < y < +w$$
$$0 < z < \text{zrange}$$

That is, all points for which x or y are greater than w in absolute value are off the screen. When the viewing pyramid is defined in this way, the division by w results in numbers in the range -1 to +.9999, the legal range for the MA1024.

The zrange given in the equations above is the distance between the nearest object to be displayed and the farthest. It will be remembered that z was adjusted by subtracting the distance to the nearest object; so clipping against 0 and the zrange will discard all points which are either too near or too far away.

The clipping assist feature sets six flags in the tag field of each point. The six bits are defined as follows:



xl (bit 24) is set if x is less than -w

xh (bit 25) is set if x is greater than w

yl (bit 26) is set if y is less than w

yh (bit 27) is set if y is greater than w

zl (bit 28) is set if z is less than zero

zh (bit 29) is set if z is greater than zrange

Zrange must have been previously stored at 20402$3. The clipping assist bit (bit 15) must have been set in 20402$0. If w is negative, the sense of each test is reversed: xl is set if x is greater than -w, etc.

The Perspective Division

Perspective division is simply division of x, y, and z by w. The MBINW operation in the MA1024 is used to perform the division. The following observations apply:

1. W must be greater than the number being divided. Any x, y, or z values greater than w must be clipped by the BPS before division.

2. W is not stored with the point, but must be recalculated from z and rescaled. If w is scaled down from z, z must also be scaled so that it is less than w.

3. After the perspective division, a translation (to the center of the screen, usually) may be performed.

MA1024 Operating Sequence

The steps to use of the MA1024 are the following:

1. Load the correct MA1024 microcode routine into the program memory. Microcode routines to perform operations described in this document are supplied by IKONAS and may be loaded using the IKLOD subroutine. The routine to perform rotation is named MMEOL.

2. Load the coefficient matrix for the rotation into the coefficient memory.

3. Load the points to be transformed into the SRB. Be sure E (bit 30) is set in the last word.

4. Load the correct values into the MA1024 control registers for the following:

    1. MA1024 program address
    2. Coefficient matrix offset
    3. Input data address
    4. Output data address
    5. Clipping assist, if needed
    6. Zrange for clipping, if needed

    See below for the formats of these control registers.

5. Start the MA1024 by storing 0 into address 20402$3 using a function code of 25.

Note

The procedure described above is the proper sequence for using the MA1024 to perform rotation. It does not use all the features of the MA1024. If you want to write custom MA1024 microcode, you should learn about the full power of the MA1024 by reading the MA1024 Programming Guide.

MA1024 Control Registers

The registers described below control the operation of the MA1024. All the registers may be accessed from the IKONAS bus in the address range 20402$0-20402$2.

Bits	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
2040280	coefficient offset C N W program address
2040281	output list offset in SR8 | input list offset in SR8
2040282	loop 0 counter value
2040283	zrange for clipping assist

program address is the offset to the MA1024 microprogram to be used. For the IKONAS rotation program MMEOL, this value will be 0. For custom microcode, the value will be the same as the value of the ORG statement in the IKASM assembly of the microcode.

W (bit 13) should be set to 0 for MMEOL. When this bit is 1, the MA1024 will expect input points in homogeneous coordinates: [x y z w]. IKONAS does not supply microcode which requires W set to 1.

N (bit 14) is set to suppress normalization during perspective division and computation of w. N should be off unless perspective division is being performed; in particular, it should be off when the clipping assist feature is on (see bit 15 below). For compatibility with the IKONAS RDS-2000 MA1024, N should be off.

C (bit 15) is set to engage the clipping assist feature. When this bit is on, bits 24-29 of the tag field of each output point will contain the clipping flags, calculated as described under the 'perspective transformation' page.

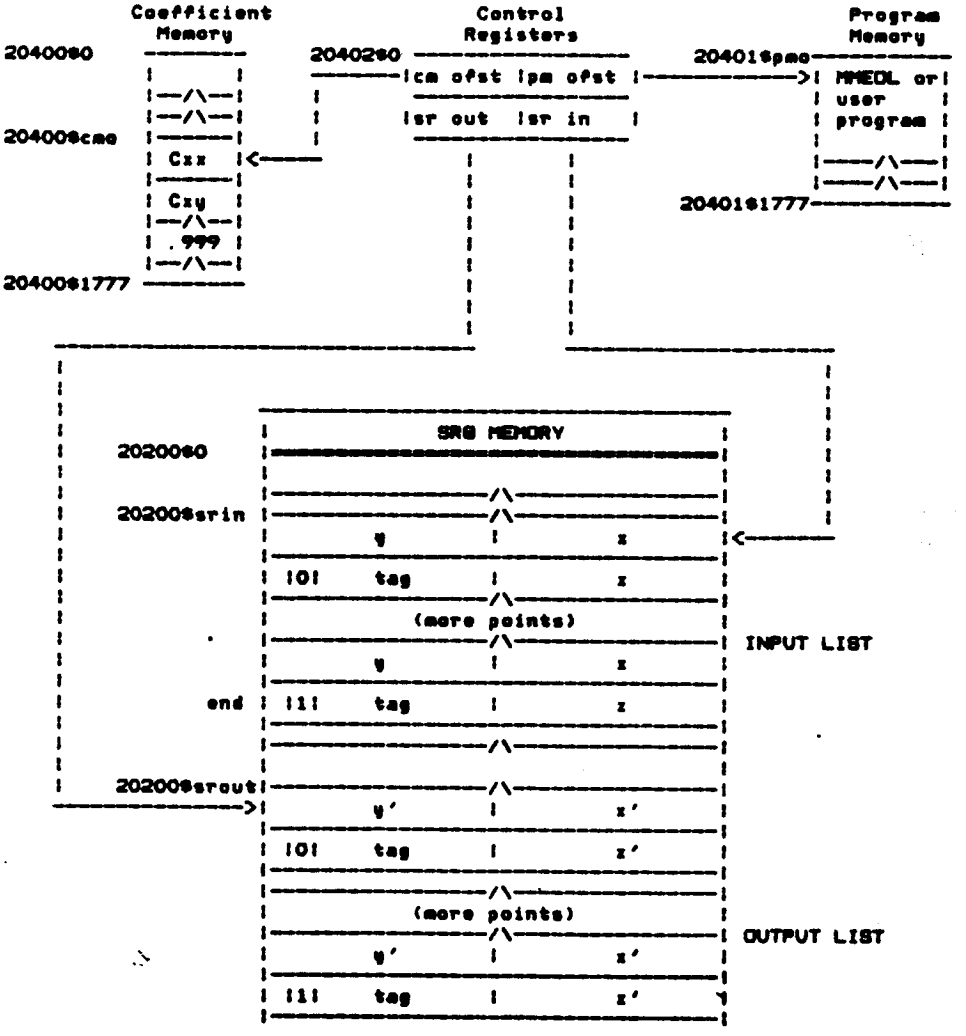
Coefficient address is the offset to the coefficient matrix to be used. If this offset is specified as cccc, the coefficient matrix must start at 20400$cccc.

Input and output offsets are the offsets to the input and output areas in the SR8. If one of these offsets is specified as xxxx, the area at 20200$xxxx will be used for input or output. The last point in the input list must have the end-of-list (E) bit set.

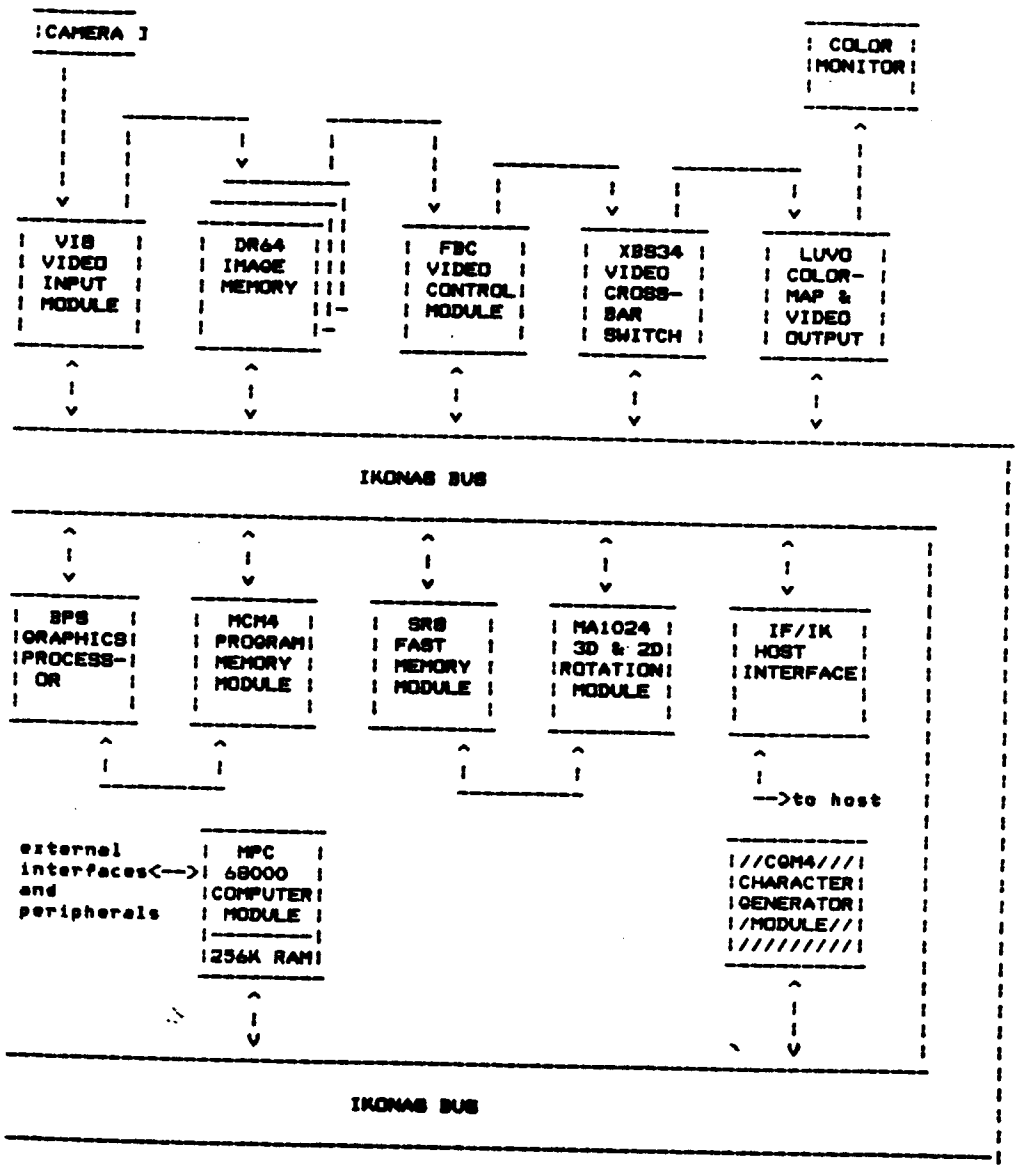
Loop 0 counter value is a value which may be used by user microcode. IKONAS microcode does not use the loop 0 counter, which may be left uninitialized.

Zrange for clipping assist is the distance between the nearest and farthest objects to be displayed. The setting is meaningful only if clipping assist is engaged.

Diagram of MA1024 Control Flow



THE CGM4 CHARACTER GENERATOR



The CQM4 Character Generator

- Writes characters directly into the DR64 framebuffer
- Character shade is set by user
- Fast - writes 1 million pixels per second after startup
- Allows variable character size - up to 32x32
- Allows variable spacing between characters
- Uses SR8 for font memory to allow user-programmed character set
- Up to 32 full ASCII fonts available at one time with one SR8
- Allows magnification of characters without changing font
- Allows variable direction: left, right, up, down
- Can fill background of character with specified shade, or leave it unchanged
- On-board-font option gives standard 7x9 font without requiring an SR8 or font programming

CGM4 Programming

The CGM4 converts a character string in memory into pixels in the DR64 framebuffer. The pixels drawn into the framebuffer will be a visible representation of the character string when viewed on the display monitor.

When the CGM4 starts to draw a character, it consults a font table which indicates which pixels should be written for the character code. The font table is stored in an SRB memory in coded form.

To draw characters with the CGM4, the following information is needed:

1. Description of the font:
    1. the font table - a map of each character
    2. the address of the font table
    3. the width of each character
    4. the height of each character
    5. the spacing between characters
2. Color of the characters, and background color if applicable
3. Magnification factor for the characters
4. Address of the character string to be displayed
5. Address in the DR64 framebuffer at which the characters are to be drawn
6. The character string itself
7. Mode settings, including the GO bit to start the operation (see CGM4 Base Control Block)

Coding the Font Table Entry for a Character

The font table consists of one entry for each character in the character set. Each entry indicate which pixels should be written for the corresponding character.

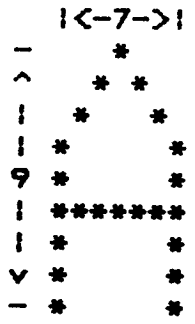
To create a font table entry for a character, follow this procedure:

1. Note the width and height (in pixels) of the characters in the font. Call the width W and the height H; in thi example W=7 and H=9. Each character in the font will occupy the same amount of space on the screen.

(continued)

Font-table Entry Example - continued

2. Draw a picture of the character, W pixels wide and H pixels high, noting which pixels should be written for the character:



3. Change each cell within the WxH grid to a 0 if it is a space, or a 1 if it is not:

0001000
0010100
0100010
1000001
1111111
1000001
1000001
1000001
1000001

4. Join the rows of the grid into one long string of bits. There will be WxH bits. The last bit in each row will be immediately followed by the first bit of the next row.

!row 1!!row 2!!row 3!!row 4!!row 5!!row 6!!row 7!!row 8!!row 9!
000100000101000100010100000111111111000001100000011000000110000001

5. Break the string into 32-bit words. If there is not an even multiple of 32 bits in the string, add bits to the end to fill out the last 32-bit word. Copy the string from left to right, and fill up the 32-bit words from high-order bit to low-order bit.

Word 0: 00010000010100010001010000011111
Word 1: 11110000011000001100000110000001x

6. You now have the font entry for the character. Install it in the font table at the proper address, as shown below.

Installing Font Table Entries Within the Font Table

After you have created the font table entry for a character, you must still store the entry into the font table at the address expected by the CQM4.

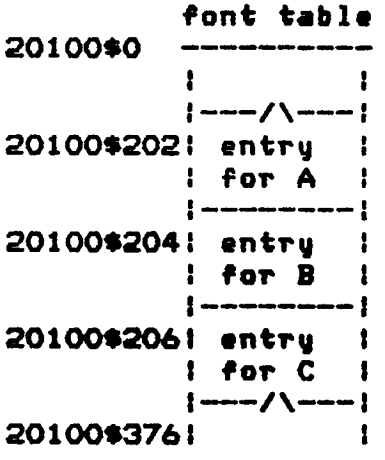
The CQM4 uses three pieces of information to calculate the address of the font table entry for a character it is about to display: the font table address, the number of words per font table entry, and the ASCII code for the character.

The address of the font table entry for a character should be equal to:

font table address+(number of words per entry * ASCII code)

In other words, the CQM4 uses the ASCII character code to index into the font table.

Example. If the base address of a 7x9 font is 20100$0, what is the address of the entry for uppercase A? Solution. The ASCII code for A is 101 octal. There are 2 words per font table entry, so the address of the entry for A will be 20100$0+(2*101)=20100$202.



Installing the Character String to be Displayed

The character string is stored into memory as a sequence of 8-bit characters, four characters per IKONAS 32-bit word.

Within each IKONAS word, the first character to be displayed is the low-order character (bits 0-7); then bits 8-15; etc.

When all four characters from an IKONAS word have been displayed, the next sequential IKONAS word is fetched and the characters in it are displayed.

The end of the character string is indicated by a character consisting of eight zero bits (character code 000, the ASCII NUL character). This may come in the middle of an IKONAS word, so that the display need not be a multiple of four characters.

The characters will be displayed in the order shown:

20200$200	4	3	2	1
20200$201	8	7	6	5
20200$202	12	11	10	9

etc.

Starting the CGM4

After the font has been loaded and the character string has been loaded, all that remains is to set up the CGM4 control blocks. There are two: the CGM4 base control block at 20600$0 and the CGM4 auxiliary control block (CGMCB), which can be anywhere in memory and is pointed to by the CGM4 base control block.

Format of the CGMCB:

Bits	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CGMCB+0	char. height (5 bits) | char. width (5 bits) | font table offset (22 bits)
CGMCB+1	shade to use for pixels in the character (32 bits)
CGMCB+2	shade to use for background pixels, if bit 12 of 20600$0=1 (32 bits)
CGMCB+3	char. spacing (16 bits) | starting pixel address for output (yyyyy$xxxx) (16 bits)
CGMCB+4	address of character string to display (32 bits)

char. height: height of a character in pixels

char. width: width of a character in pixels

font table offset: offset of the font table within the SR8 memory. If the SR8 base address is 20100$0, the font table will be at 20100$font-table-offset.

character shade: shade to be written to pixels written in the character

background shade: shade to use for background pixels. This applies only if NON-TRANSPARENT mode is selected (see the CGM4 base control block)

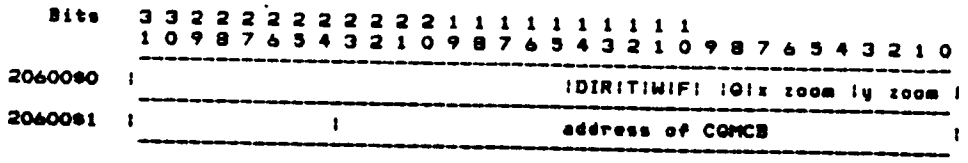
char. spacing: number of pixels to skip between characters

starting output address: pixel address to use for the top-left pixel of the first character

character string address: address of characters to be displayed

CGM4 Base Control Block

Format of the CGM4 base control block (except for bit 8 of 20600$0, these locations are read-only):



DIR (bits 13-14): gives the orientation to be used. 00=normal left-to-right, 01=top-to-bottom, 10=bottom-to-top, 11=right-to-left.

T (bit 12): if on, transparent mode is used: background pixels are left unchanged. If off, non-transparent mode is used: the background shade is written to background pixels.

W (bit 11): on for word mode writes, off (normal) for pixel mode.

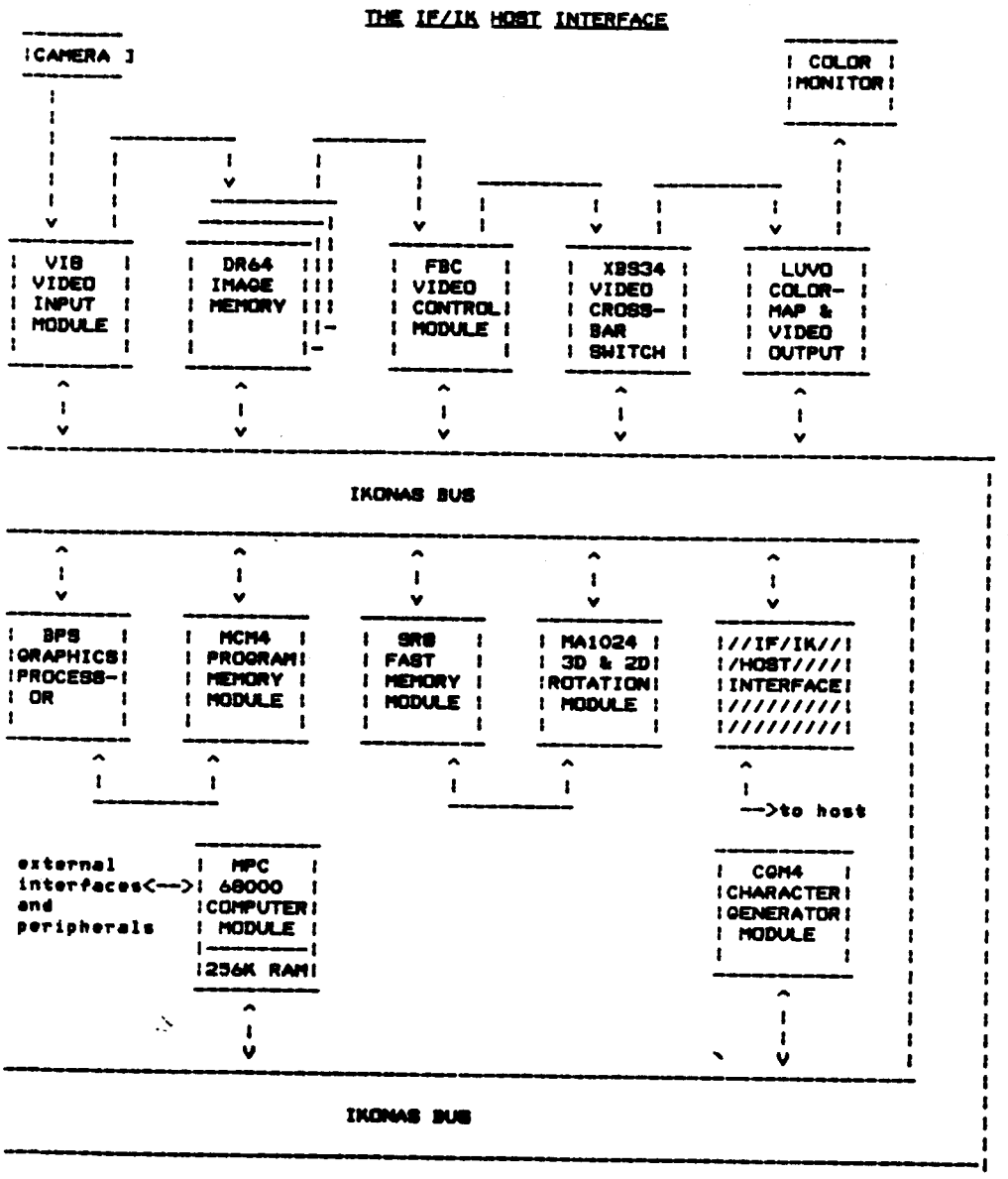
F (bit 10): on to use the optional on-board 7x9 font, off to use to SR8 font memory.

G (bit 8): the GO bit. When this bit is set, the CGM4 starts to draw the character string. All information needed by the CGM4 should be ready when the GO bit is set. When 20600$0 is read from, bit 8 will be on if the CGM4 is busy, off if it is idle and can start another operation.

x zoom (bits 4-7): the magnification factor in the x-direction: a setting of 1 (normal) is no magnification; a setting of 2-17 octal magnifies the x-direction by that amount; a setting of 0 magnifies the x-direction 16 times.

y zoom (bits 0-3): magnification in the y-direction, similar to x zoom.

THE IF/IK HOST INTERFACE



THE IF/IK HOST INTERFACE

- Direct DMA connection to user's host computer
- Fast - 2 million bytes per second transfer rate
- Allows non-DMA I/O of single words to avoid host DMA overhead
- Multiple modes: 32-bit word, 16-bit halfword, and 8-bit byte:
  - 32-bit word mode allows easy access to 32-bit data
  - 16-bit halfword mode allows efficient access of 16-bit and smaller items
  - 8-bit byte mode allows efficient access to a single red, green, or blue image component
- Vectored interrupts to host based on IKONAS events
- Allows host-programmable system reset
- Allows transfer to be inhibited except during blanking interval

IF/IK Transfer Modes

The IF/IK is the connection from the IKONAS to the user's host computer. Use the IF/IK to start a transfer between the IKONAS bus and the host computer's memory.

The examples here will assume a host computer with a 16-bit word; for example, a DEC PDP-11 or VAX-11 with a UNIBUS.

The IF/IK supports three transfer modes:

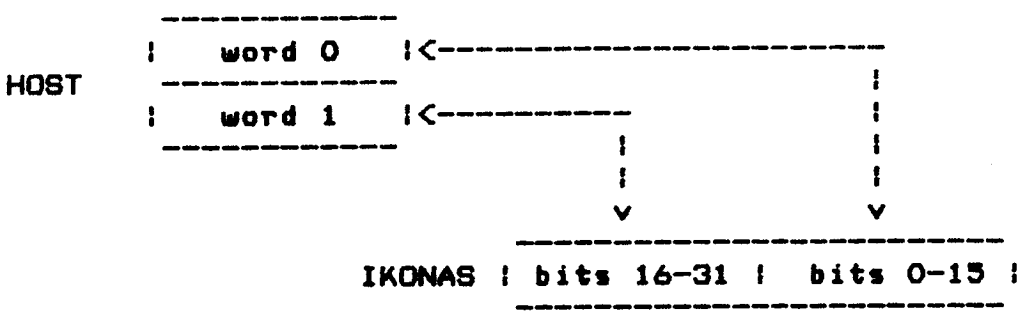
1. WORD mode: two host 16-bit words correspond to one IKONAS 32-bit words.

2. HALFWORD mode: one host 16-bit word corresponds to the low-order bits (bits 0-15) of one IKONAS 32-bit word. Bits 16-31 of the IKONAS 32-bit word are zero.

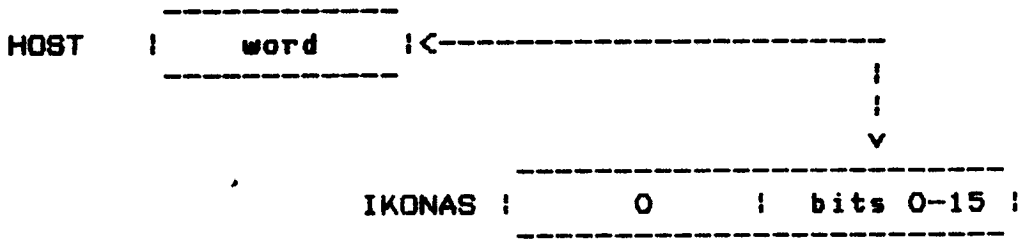
3. BYTE mode: each 8-bit byte of host data corresponds to one IKONAS 32-bit word. The IKONAS 32-bit word is addressed as four bytes (bits 0-7, 8-15, 16-23, and 24-31); the user specifies which byte receives of transmits with the host.

Transfer Mode Examples

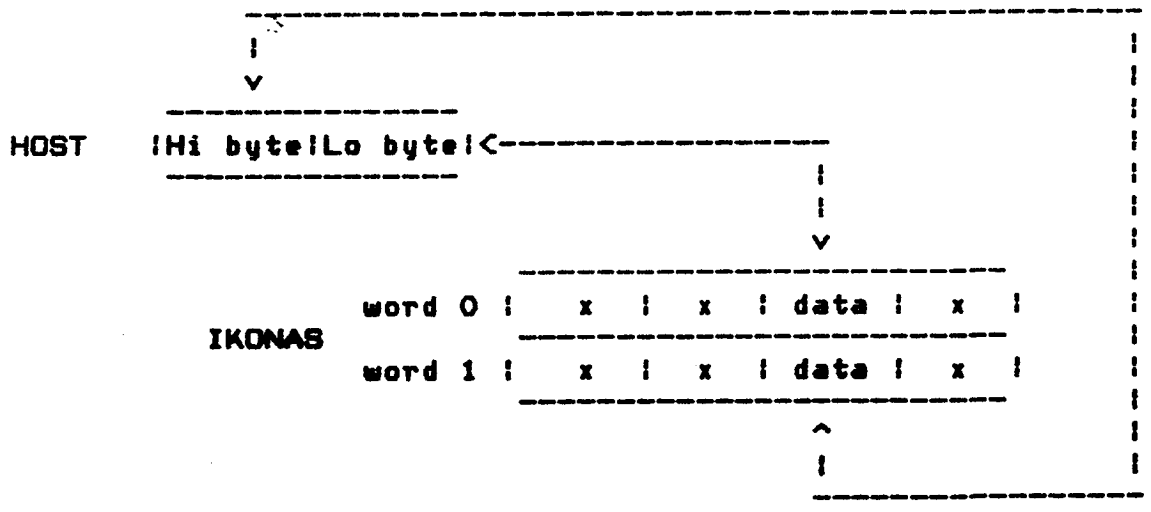
Word Mode



Halfword Mode



Byte Mode (example using byte 1)



The bytes marked x are undefined.

Notes on the Transfer Modes

1. In word mode, the even host words correspond to the low-order half of the IKONAS words; the odd host word correspond to the high-order half of the IKONAS words. This is in accordance with the host computer's numbering of words in INTEGER*4 variables.

2. In halfword mode, each host word corresponds to the Low-order half of an IKONAS word. However, during write operations, all 32 bits are written; bits 16-31 are written as 0.

3. In byte mode, each host byte corresponds to a selected byte of an IKONAS word. However, during write operations, all 32 bits are written; bits in bytes other than the selected byte are undefined. If you want to write to a single channel of the DR64 framebuffer (for example, the green component, byte 1), you should set the write mask to protect all bits except the channel to be written.

Starting an I/O Operation

The steps to starting an I/O operation with the IF/IK are:

1. Set up the host half of the interface, as specified in the programming guide for your host interface. (See the IKONAS PROGRAMMING REFERENCE MANUAL.)
2. Set up the IKONAS CONTROL REGISTER, described below
3. Start the I/O operation as described in the programming guide.
4. Wait for completion interrupt from the IF/IK.

The IKONAS Control Register

The IKONAS control register defines the transfer mode for all I/O operations using the IF/IK. The format is as follows:

```
Bits   1 1 1 1 1 1
       5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
      ---------------------------------
      !BYT!P!F!B!R!X!+!I!D!H!   func   !
      ---------------------------------
```

func (bits 0-4): the IKONAS bus function code. Bit 4 is always set for write operations, off for read operations. For most RDS3000 modules, the write function code is octal 20, read is 00; but each module may accept special function codes. Refer to the individual module descriptions. In particular, the DR64 uses many function codes.

H (bit 5): set for halfword mode, off for word or byte mode. When bit 5 is set, bit 11 must be reset.

D (bit 6): set for DMA transfers, reset for programmed I/O. the IKONAS I/O driver in the host will normally set this bit automatically without special action by the user.

I (bit 7): set for invisible I/O, reset normally. Invisible I/O occurs during the blanking portion of the picture. It is useful mainly for accesses to the LUVO, since any access to the LUVO during the picture causes an interruption of the video signal.

± (bit 8): set to increment the IKONAS address between words. This bit should always be set.

X (bit 9): set to allow the BPS32 to execute. When this bit is reset, the BPS32 will stop; setting the bit again will cause the BPS32 to resume from the point at which it stopped.

R (bit 10): set to force an IKONAS system reset.

(continued)

The IKONAS Control Register - continued

B (bit 11): set for byte mode, off for word or halfword mode.

Specific to read operations:

F (bit 12): this bit is read-only. When it is on, the video signal is in the vertical blanking interval.

P (bit 13): this bit is read-only. When it is on, the IKONAS system has requested service from the host. The request may be from the MPC or from the BPS32.

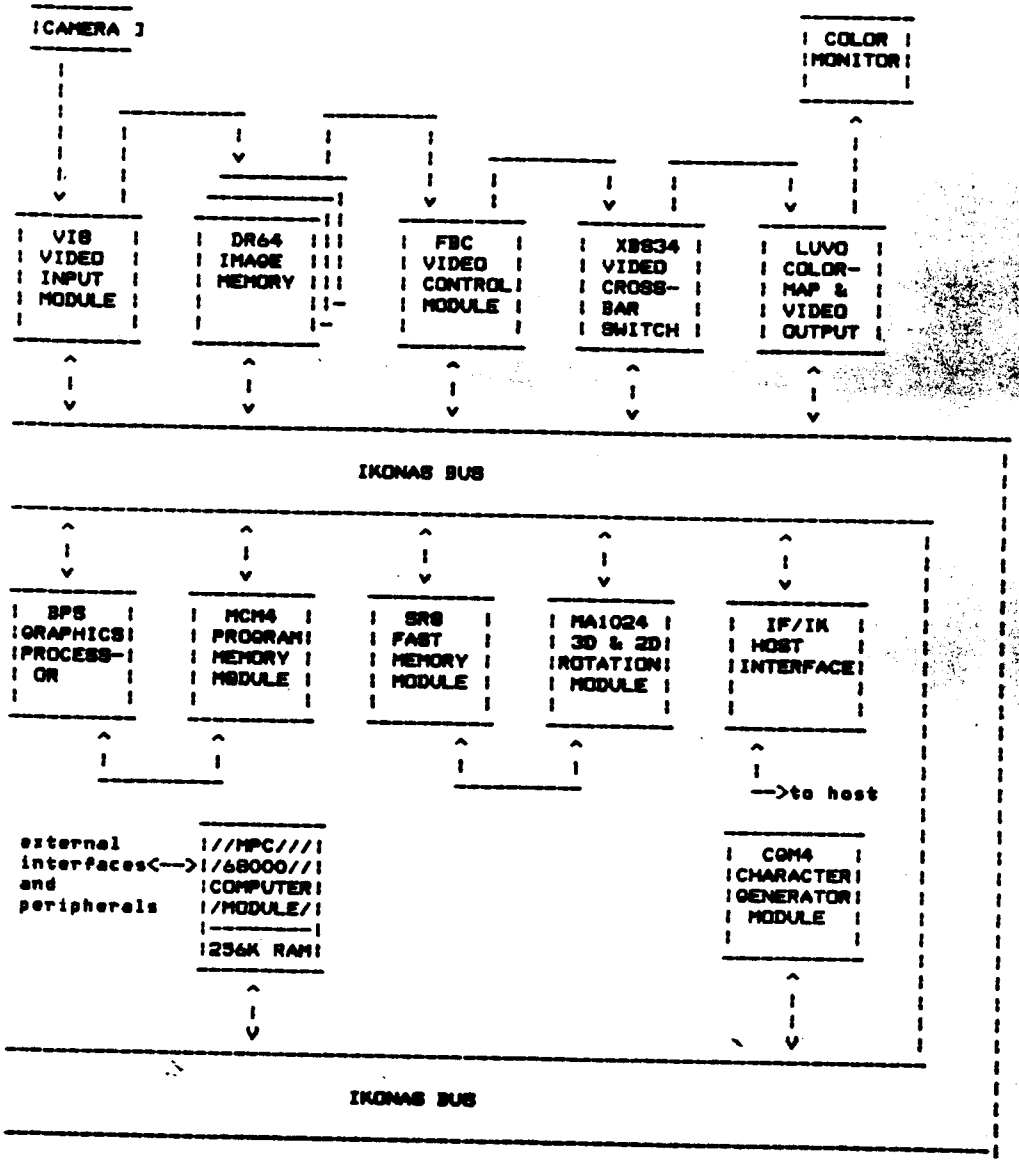
BYT (bits 14-15): the byte number to be selected for byte mode transfers. This field is meaningful only when bit 11 (B) is set. 00=byte 0 (bits 0-7), 01=byte 1 (bits 8-15), 10=byte 2 (bits 16-23), 11=byte 3 (bits 24-31). Set by user for read operations.

Specific to write operations:

(bits 12-14): these 3 bits contain the sender id for write operations; they specify the write mask to be written or used for the DR64 or DR256 and the write mask and shade register for the QM64. Set by user for write operations.

(bit 15): write-only; reserved for future use.

THE MPC 68000 COMPUTER

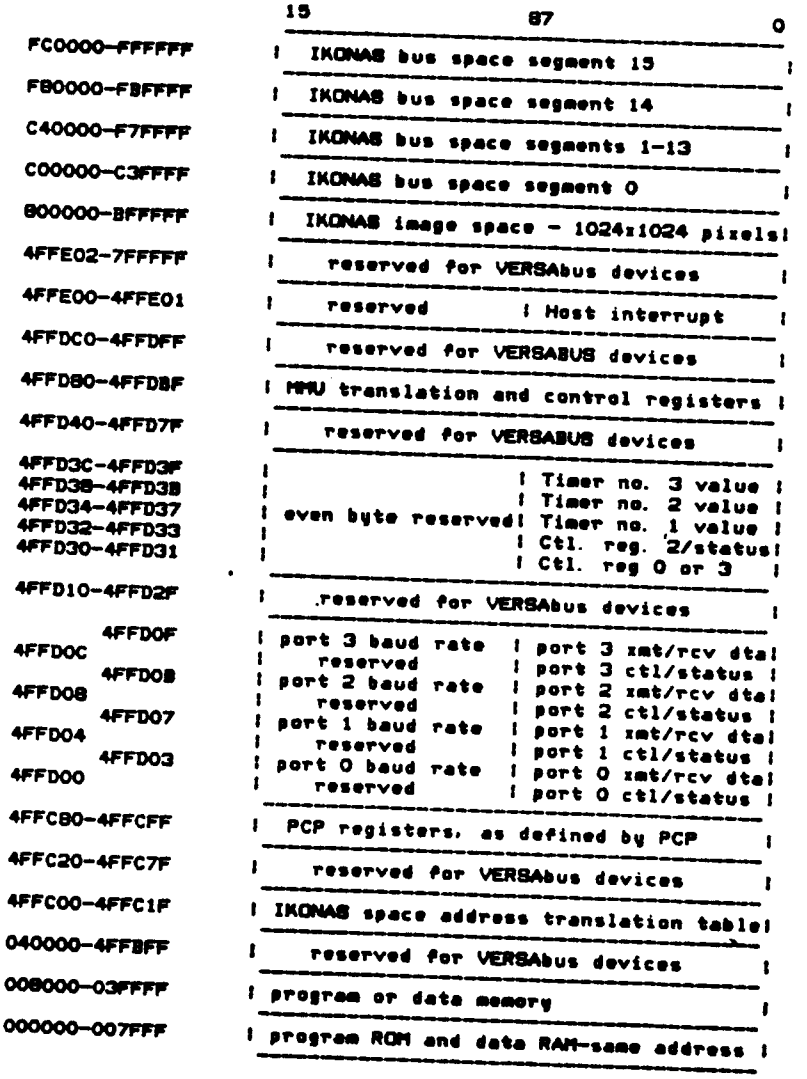


The MPC 68000 Computer Module

- Powerful 68000 processor to give minicomputer flexibility and power
- Allows standalone processing in the RDS3000 system
- 256K bytes of memory
- Up to 32K bytes of ROM for fixed-function systems
- Four RS-232 serial ports to allow terminals, data tablets, and other devices
- Programmable event timer
- Connects to VERSAbus and Multibus - gives full range of device support
- Connects to the IKONAS PCP:
    - 4 RS-232 ports
    - 16 analog ports (dials, joysticks, etc.)
    - 16 parallel ports (buttons, LEDs, etc.)

MPC Memory Map

(All Addresses in hex)



IKONAS Bus Access

- IKONAS bus is directly mapped to 68000 address space
- Two kinds of access:
  - Image mode: the image memory (1024x1024 pixels) is mapped to locations 800000-BFFFFF
  - Segment mode: C00000-FFFFFF is divided into 16 segments, each mapped to an area of IKONAS control space
- IKONAS bus function codes and sender IDs can be specified
- Fast interface allows 68000 to run while write cycles complete

IKONAS Address Translation Tables

There are 16 address translation table entries, one for image mode and segment 0, and one for each segment 1-15. Each translation table entry specifies:

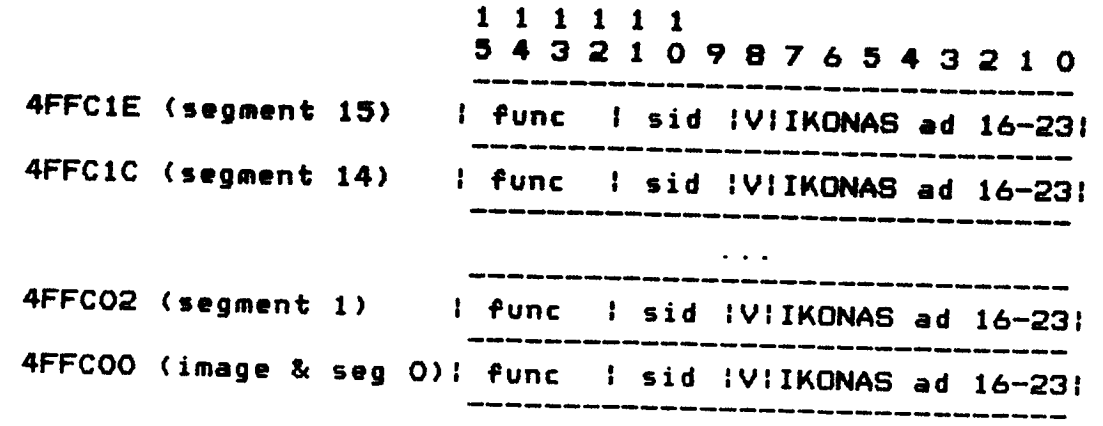
The upper IKONAS address bits to use

The IKONAS bus function code to use

The IKONAS bus sender id to use

Whether the segment is to be mapped to the IKONAS bus or the VERSAbus

The translation table looks like:



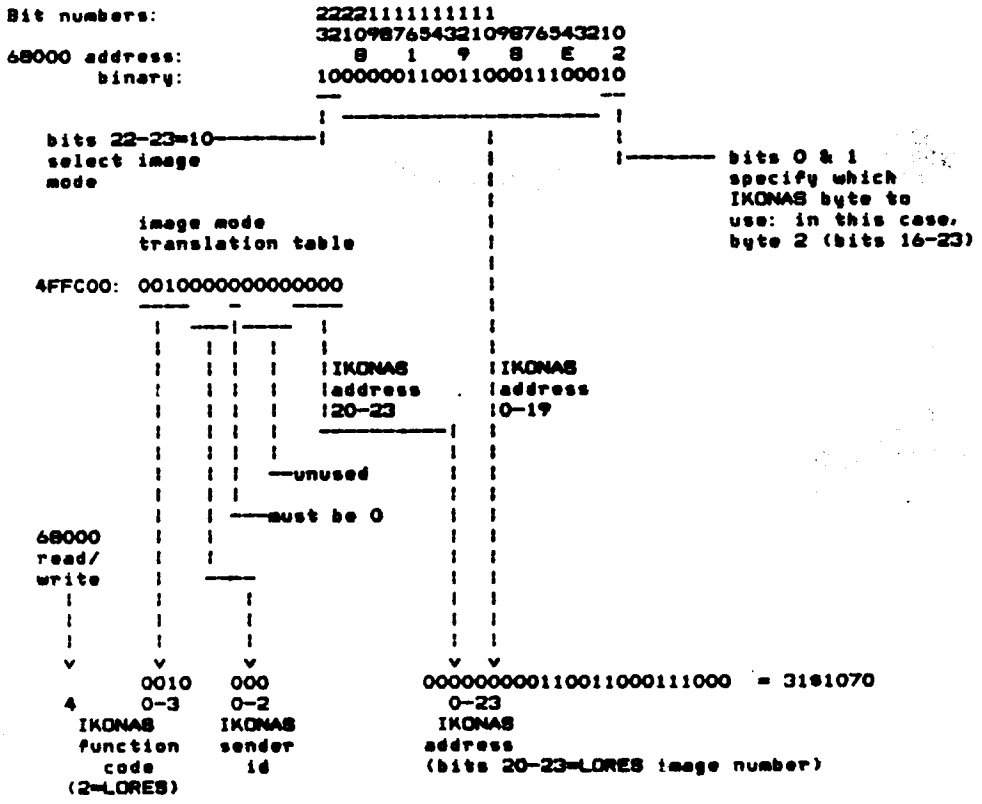
func will be used for IKONAS bus function bits 0-3

sid will be used for the IKONAS sender id

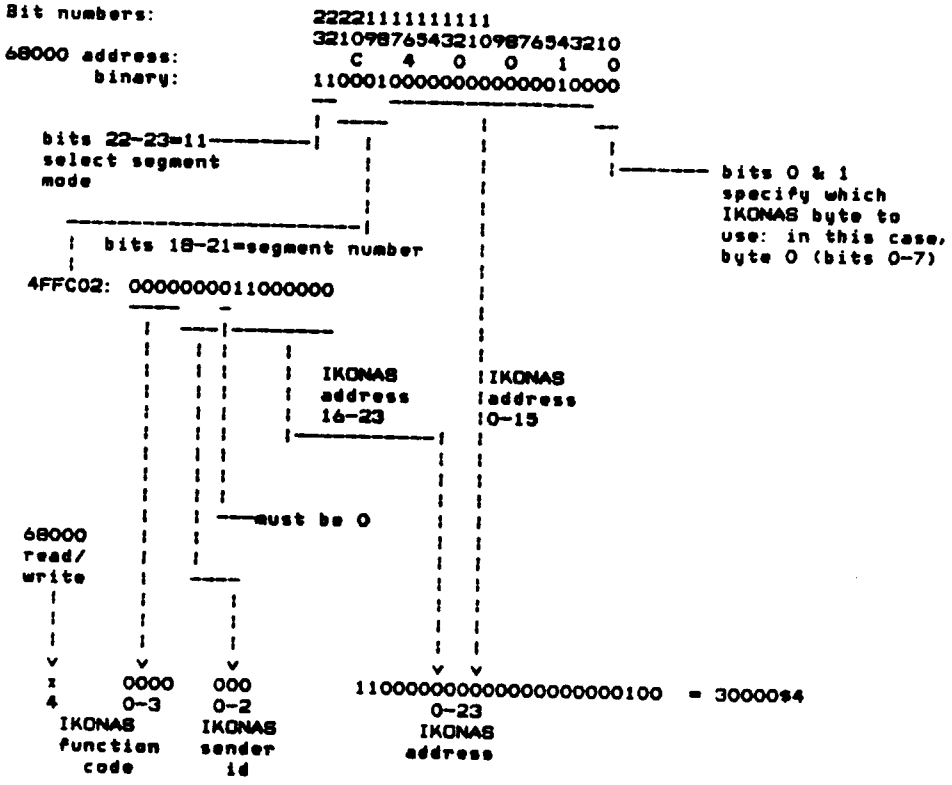
V should be set to cause this segment not to be mapped to the IKONAS bus. It should correspond to VERSAbus devices, or a bus error will result.

IKONAS ad 16-23 will be supplied as the upper bits of the IKONAS bus address

Image Memory Address Translation



Segment Memory Address Translation

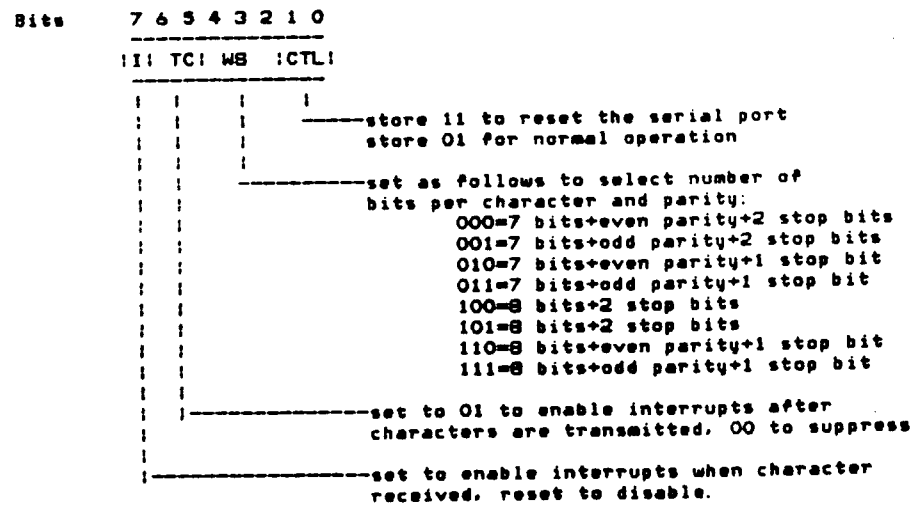


Serial Port Setup

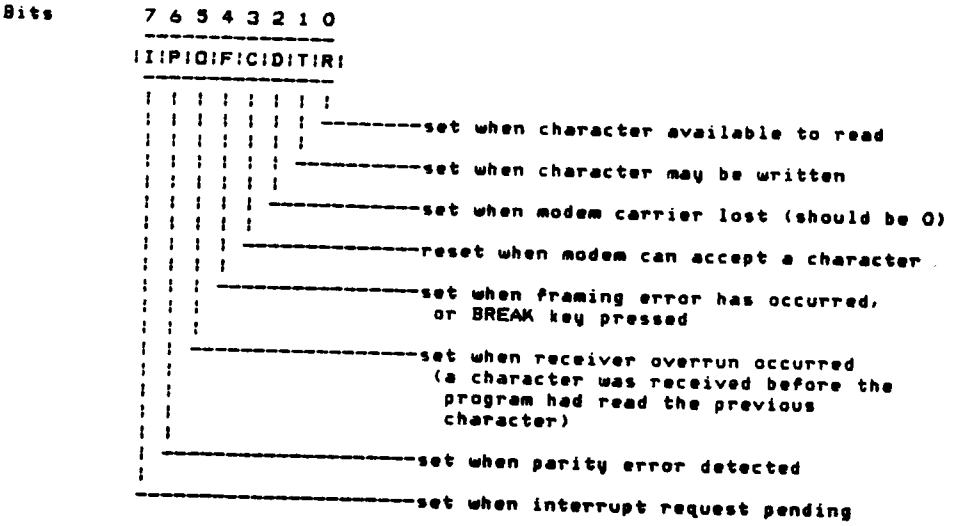
To start a serial port, you must:

- Write the baud rate to the baud rate address
- Issue reset through the control register
- Initialize the control register

Baud Rate	Data Bits 0-3	Baud Rate	Data Bits 0-3	Baud Rate	Data Bits 0-3	Baud Rate	Data Bits 0-3
50	1111	150	1011	1800	0111	4800	0011
75	1110	300	1010	2000	0110	7200	0010
110	1101	600	1001	2400	0101	9600	0001
134	1100	1200	1000	3600	0100	19200	0000



Serial Port Status Register



Serial Port Programming Sequence

1. Setup
    1. Store control register=03 to reset
    2. Set character length, receiver interrupt enable (example: control reg=8D for 7 bits, odd parity, one stop bit, all interrupts enabled)
    3. Set baud rate (example: baud rate generator=01 for 9600 baud)

2. Read sequence
    1. Interrupt to level 2 (serial port interrupt level)
    2. Read status registers to determine source of interrupt - detect a received-character interrupt
    3. Read the character from the data register. This clears the interrupt request
    4. Process the character
    5. Wait for interrupt

3. Write Sequence
    1. Write character to data register and enable transmitter interrupt
    2. Wait for interrupt
    3. Interrupt to level 2 - decode and detect transmitter interrupt
    4. If another character is ready to send, write it and wait; else disable transmitter interrupt

Programmable Timer (MPC256)

The programmable timer consists of three separate timers, each with several modes of operation possible. For a complete description of the timer, see the Motorola MC6840 data sheet. The following is an example of timer use.

Setting a real-time clock or interval timer

Problem: to create a timer interrupt every 16 milliseconds (60 Hz interrupt rate) using timer number 1.

1. Select control register 1 by storing 01 at 4FFD33 (control register 2).

2. Set the timer mode by storing C2 into control register 1, location 4FFD31. Bit 0 of each timer control register has special meaning; for example, bit 0 in the control register for timer 1 is a master-reset bit; so if you wanted to set a different timer, the setting of bit 0 in the timer control register would depend on the timer being set.

3. Set the timeout value: 16000 (the number of 1-microsecond clock ticks in 16 milliseconds), stored into the timer 1 value, locations 4FFD35 (upper byte) and 4FFD37 (lower byte). The MOVP instruction could be used for this.

4. The timer is started. Wait for an interrupt

5. When the timer expires, interrupt level 3 will be entered.

6. The interrupt routine should read the timer status register (4FFD33) to determine which timer interrupted. The three low-order bits of the status register indicate which timers have interrupted: bit 0 on=timer 1, bit 1=timer 2, bit 2=timer 3.

7. For each timer which has interrupted, the interrupt routine should read the counter value. Reading the counter value clears the interrupt from that counter.

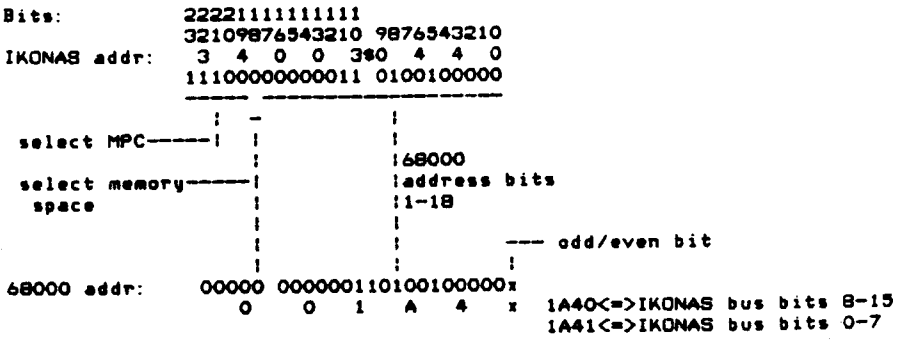
8. The interrupt routine performs its processing.

9. If a continuous real-time clock is needed, the interrupt routine exits; the timer will interrupt again when the time expires. If a single interval is needed, the interrupt should be disabled by storing 0 into control register 1.

IKONAS bus access to the MPC

- RAM occupies IKONAS addresses 34000$0-34377$1777
- I/O space occupies IKONAS addresses 34777$1000-34777$1777
- Each IKONAS access transfers 16 bits (two bytes) to bits 0-15 of the IKONAS bus
- IKONAS accesses pass through the MMU for translation

Accessing MPC memory from the IKONAS BUS



Accessing MPC I/O space from the IKONAS BUS

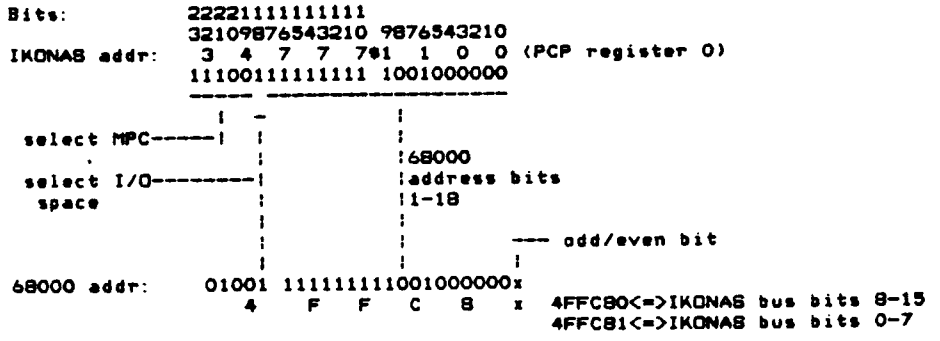


Table of I/O Addresses

Device	MPC address	IKONAS bus address
IKONAS translation unit	4FFC00-4FFC1F	34777$1000-34777$1037
PCP	4FFC80-4FFCFF	34777$1100-34777$1177
Serial ports	4FFD00-4FFD0F	34777$1200-34777$1217
Timer	4FFD30-4FFD3E	34777$1230-34777$1247
MMU	4FFD80-4FFDBF	34777$1300-34777$1337
Host interrupt	4FFE01	34777$1400