# RAINBOW GRAPHICS OPTION

# PROGRAMMER'S REFERENCE GUIDE

AA-AE36A-TV

```
****************
* 20 April '84 *
* REVIEW DRAFT *
****************
```

| | | |
|---|---|---|
| DIGITAL | MASSBUS | UNIBUS |
| DEC | PDP | VAX |
| DECmate | P/OS | VMS |
| DECsystem−10 | Professional | VT |
| DECSYSTEM−20 | Rainbow | Work Processor |
| DECUS | RSTS | DIBOL |
| RSX | | |

# CONTENTS

# CHAPTER 1

# PREFACE

## THE INTENDED AUDIENCE

The Rainbow Graphics Option Programmer's Reference Guide is written for the experienced systems programmer who will be programming applications that display graphics on Rainbow video monitors. It is further assumed that the system programmer has had both graphics and 8088 programming experience.

The information contained in this document is not unique to any operating system; however, it is specific to the 8088 hardware and 8088-based software.

## ORGANIZATION OF THE MANUAL

The Graphics Option Programmer's Reference Guide is subdivided into four parts containing fifteen chapters and two appendixes as follows:

- PART I - OPERATING PRINCIPLES contains the following four chapters:

  - Chapter 1 provides an overview of the Graphics Option including information on the hardware, logical interface to the CPU, general functionality, color and monochrome ranges, and model dependencies.

  - Chapter 2 describes the monitor configurations supported by the Graphics Option.

  - Chapter 3 discusses the logic of data generation, bitmap addressing, and the GDC's handling of the screen display.

  - Chapter 4 describes the software components of the Graphics Option such as the control registers, maps, and buffer areas accessible under program control.

- PART II - PROGRAMMING GUIDELINES contains the following eight chapters:
  - Chapter 5 discusses programming the Graphics Option for initialization and control operations.
  - Chapter 6 discusses programming the Graphics Option for setting up bitmap write operations.
  - Chapter 7 discusses programming the Graphics Option for area write operations.
  - Chapter 8 discusses programming the Graphics Option for vector write operations.
  - Chapter 9 discusses programming the Graphics Option for text write operations.
  - Chapter 10 discusses programming the Graphics Option for read operations.
  - Chapter 11 discusses programming the Graphics Option for scroll operations.
  - Chapter 12 contains programming notes and timing considerations.

- PART III - REFERENCE MATERIAL contains the following three chapters:
  - Chapter 13 provides descriptions and contents of the Graphics Option's registers, buffers, masks, and maps.
  - Chapter 14 provides descriptions and contents of the GDC's status register and FIFO buffer.
  - Chapter 15 provides a description of each supported GDC command arranged in alphabetic sequence within functional grouping.

- PART IV - APPENDIXES contains the following two appendixes:
  - Appendix A contains the Graphics Option's Specification Summary.
  - Appendix B is a fold-out sheet containing a block diagram of the Graphics Option.

# PREFACE

## SUGGESTIONS FOR THE READER

For more information about the Graphics Display Controller refer to the following:

- The uPD7220 GDC Design Manual---NEC Electronics U.S.A. Inc.
- The uPD7220 GDC Design Specification---NEC Electronics U.S.A. Inc.

For a comprehensive tutorial/reference manual on computer graphics, consider "Fundamentals of Interactive Computer Graphics" by J. D. Foley and A. Van Dam published by Addison--Wesley Publishing Company, 1982.

## Terminology

| | |
|---|---|
| ALU/PS | Arithmetic Logical Unit and Plane Select (register) |
| Bitmap | Video display memory |
| GDC | Graphics Display Controller |
| Motherboard | A term used to refer to the main circuit board where the processors and main memory are located -- hardware options, such as the Graphics Option, plug into and communicate with the motherboard |
| Nibble | A term commonly used to refer to a half byte (4 bits) |
| Pixel | Picture element when referring to video display output |
| Resolution | A measure of the sharpness of a graphics image -- usually given as the number of addressable picture elements for some unit of length (pixels per inch) |
| RGB | Red, green, blue -- the acronym for the primary additive colors used in color monitor displays |
| RGO | Rainbow Graphics Option |
| RMW | Read/Modify/Write, the action taken when accessing the bitmap during a write or read cycle |
| VSS | Video Subsystem |

# PART I

# OPERATING PRINCIPLES

# CHAPTER 1

# OVERVIEW

## 1.1 HARDWARE COMPONENTS

The Graphics Option is a user-installable module that adds graphics and color display capabilities to the Rainbow system. The graphics module is based on a NEC uPD7220 Graphics Display Controller (GDC) and an 8 X 64K dynamic RAM video memory that is also referred to as the bitmap.

The Graphics Option is supported, with minor differences, on Rainbow systems with either the model A or model B motherboard. The differences involve the number of colors and monochrome intensities that can be simultaneously displayed and the number of colors and monochrome intensities that are available to be displayed (see Table 1). Chapter 5 includes a programming example of how you can determine which model of the motherboard is present in your system.

| | | Med. Resolution | | High Resolution | |
|---|---|---|---|---|---|
| Config. | Model | Color | Mono. | Color | Mono. |
| Monochrome Monitor Only | 100-A | N/A | 4/4 | N/A | 4/4 |
| | 100-B | N/A | 16/16 | N/A | 4/16 |
| Color Monitor Only | 100-A | 16/1024 | N/A | 4/1024 | N/A |
| | 100-B | 16/4096 | N/A | 4/4096 | N/A |
| Dual Monitors | 100-A | 16/4096 | 4/4 | 4/4096 | 4/4 |
| | 100-B | 16/4096 | 16/16 | 4/4096 | 4/16 |

Table 1. Colors and Monochrome Intensities - Displayed/Available

The GDC, in addition to performing the housekeeping chores for the video display, can also:

- o Draw lines at any angle
- o Draw arcs of specified radii and length
- o Fill rectangular areas
- o Transfer character bit-patterns from font tables in main memory to the bitmap

### 1.1.1 Video Memory (Bitmap)

The CPUs on the motherboard have no direct access to the bitmap memory. All writes are performed by the external graphics option hardware to bitmap addresses generated by the GDC.

The bitmap is composed of eight 64K dynamic RAMs. This gives the bitmap a total of 8x64K of display memory. In high resolution mode, this memory is configured as two planes, each 8 X 32K. In medium resolution mode, this memory is configured as four planes, each 8 X 16K. However, as far as the GDC is concerned, there is only one plane. All plane interaction is transparent to the GDC.

Although the bitmap is made up of 8x64K bits, the GDC sees only 16K of word addresses in high resolution mode (2 planes X 16 bits X 16K words). Similarly, the GDC sees only 8K of word addresses in medium resolution mode (4 planes X 16 bits X 8K words). Bitmap address zero is displayed at the upper left corner of the monitor screen.

### 1.1.2 Additional Hardware

The option module also contains additional hardware that enhances the performance and versatility of the basic GDC. This additional hardware includes:

- o A 16 X 8-bit Write Buffer used to store byte-aligned or word-aligned characters for high performance text writing or for fast block data moves from main memory to the bitmap

- An 8-bit Pattern Register and a 4-bit Pattern Multiplier for improved vector writing performance
- Address offset hardware (256 X 8-bit Scroll Map) for full and split-screen vertical scrolling
- ALU/PS register to handle bitplane selection and the write functions of Replace, Complement, and Overlay
- A 16 X 16-bit Color Map to provide easy manipulation of pixel color and monochrome intensities
- Readback hardware for reading a selected bitmap memory plane into main memory

## 1.2 RESOLUTION MODES

The Graphics Option operates in either of two resolution modes:

- Medium Resolution Mode
- High Resolution Mode

### 1.2.1 Medium Resolution Mode

Medium resolution mode displays 384 pixels horizontally by 240 pixels vertically by four bitmap memory planes deep. This resolution mode allows up to 16 colors to be simultaneously displayed on a color monitor. Up to sixteen monochrome shades can be displayed simultaneously on a monochrome monitor.

### 1.2.2 High Resolution Mode

High resolution mode displays 800 pixels horizontally by 240 pixels vertically by two bitmap memory planes deep. This mode allows up to four colors to be simultaneously displayed on a color monitor. Up to four monochrome shades can be simultaneously displayed on a monochrome monitor.

## 1.3 OPERATIONAL MODES

The Graphics Option supports the following write modes of operations:

- WORD MODE to write 16-bit words to selected planes of the bitmap memory for character and image generation

- VECTOR MODE to write pixel data to bitmap addresses provided by the GDC

- SCROLL MODE for full- and split-screen vertical scrolling and full- screen horizontal scrolling

- READBACK MODE to read 16-bit words from a selected plane of bitmap memory for special applications, hardcopy generation or diagnostic purposes

# CHAPTER 2

# MONITOR CONFIGURATIONS

In the Rainbow system with the Graphics Option installed, there are three possible monitor configurations: Monochrome only, Color only, and Dual (color and monochrome). In all three configurations, the selection of the option's monochrome output or the motherboard VT102 video output is controlled by bit two of the system maintenance port (port 0A hex). A zero in bit two selects the motherboard VT102 video output while a one in bit two selects the option's monochrome output.

## 2.1 MONOCHROME MONITOR ONLY

As shown in Figure 1, the monochrome monitor can display either graphics option data or motherboard data depending on the setting of bit two of port 0Ah. Writing an 87h to port 0Ah selects the Graphics Option data. Writing an 83h to port 0Ah selects the motherboard VT102 data. The red, green and blue data areas in the Color Map should be loaded with all F's to reduce any unnecessary radio frequency emissions.

```
Blue Intensities

Red Intensities

Green Intensities

Monochrome Intensities----->|\
                            | \
                            |  >------->Monochrome Monitor
                            | /
Motherboard  Data---------->|/
                            ^
                            |
Port 0Ah (bit 2)------------'
```

Figure 1.  Monochrome Monitor Only System

## 2.2 COLOR MONITOR ONLY

When the system is configured with only a color monitor, as in Figure 2, the green gun does double duty. It either displays the green component of the graphics output or it displays the monochrome output of the motherboard VT102 video subsystem. Because the green gun takes monochrome intensities, all green intensities must be programmed into the monochrome data area of the Color Map. The green data area of the Color Map should be loaded with all F's to reduce any unnecessary radio frequency emissions.

When motherboard VT102 data is being sent to the green gun, the red and blue output must be turned off at the Graphics Option itself. If not, the red and blue guns will continue to receive data from the option and this output will overlay the motherboard VT102 data and will also be out of synchronization. Bit seven of the Mode Register is the graphics option output enable bit. If this bit is a one, red and blue outputs are enabled. If this bit is a zero, red and blue outputs are disabled.

As in the monochrome only configuration, bit two of port 0Ah controls the selection of either the graphics option data or the motherboard VT102 data. Writing an 87h to port 0Ah enables the option data. Writing an 83h to port 0Ah selects the motherboard VT102 data.

```
Blue Intensities----------------------->Blue Gun

Red Intensities------------------------>Red Gun

Green Intensities

Monochrome Intensities----->|\
    (Green Data)            | \
                            |  >------->Green Gun
                            | /
Motherboard Data----------->|/
                            ^
                            |
Port 0Ah (bit 2)------------'
```

Figure 2.  Color Monitor Only System

## 2.3 DUAL MONITORS

In the configuration shown in Figure 3, both a color monitor and a monochrome monitor are available to the system. Motherboard VT102 video data can be displayed on the monochrome system while color graphics are being displayed on the color monitor. If the need should arise to display graphics on the monochrome monitor, the monochrome intensity output can be directed to the monochrome monitor by writing an 87h to port 0Ah . Writing an 83h to port 0Ah will restore motherboard VT102 video output to the monochrome monitor.

When displaying graphics on the monochrome monitor, the only difference other than the the lack of color is the range of intensities that can be simultaneously displayed on systems with model A motherboards.

Systems with model A motherboards can display only four monochrome intensities at any one time. Even though sixteen entries can be selected when operating in medium resolution mode, only the two low-order bits of the monochrome output are active. This limits the display to only four unique intensities at most. On systems with the model B motherboard, all sixteen monochrome intensities can be displayed.

```
Blue Intensities----------------------->Blue Gun

Red Intensities------------------------>Red Gun

Green Intensities---------------------->Green Gun

Monochrome Intensities----->|\
                            | \
                            |  >------->Monochrome Monitor
                            | /
Motherboard Data----------->|/
                            ^
                            |
Port 0Ah (bit 2)------------'
```

Figure 3. Dual Monitor System

# CHAPTER 3

# SOFTWARE LOGIC

## 3.1 GENERAL

The Graphics Display Controller (GDC) can operate either on one bit at a time or on an entire 16-bit word at a time. It is, however, limited to one address space and therefore can only write into one plane at a time. The Graphics Option is designed in such a manner that while the GDC is doing single pixel operations on just one video plane, the external hardware can be doing 16-bit word operations on up to four planes of video memory.

Write operations are multi-dimensioned. They have width, depth, length and time.

- o Width refers to the number of pixels involved in the write operation.
- o Depth refers to the number of planes involved in the write operation.
- o Length refers to the number of read/modify/write cycles the GDC is programmed to perform.
- o Time refers to when the write operation occurs in relation to the normal housekeeping operations the GDC has to perform in order to keep the monitor image stable and coherent.

## 3.2 SCREEN LOGIC

The image that appears on a video screen is generated by an electron beam performing a series of horizontal scan lines in the forward direction (to the right). At the end of each horizontal scan line, the electron beam reverses its direction and moves to the beginning of the next scan line. At the end of the last scan line, the electron beam does a series of scan lines to position itself at the beginning of the first scan line.

The GDC writes to the bitmap (display memory) only during the screen’s horizontal and vertical retrace periods. During active screen time, the GDC is busy taking information out of the bitmap and presenting it to the video screen hardware. For example, if the GDC is drawing a vector to the bitmap, it will stop writing during active screen time and resume writing the vector at the next horizontal or vertical retrace.

In addition to the active screen time and the horizontal and vertical retrace times, there are several other video control parameters that precede and follow the active horizontal scans and active lines. These are the Vertical Front and Back Porches and the Horizontal Front and Back Porches. The relationship between all the video control parameters is shown in Figure X. Taking all the parameters into account, the proportion of active screen time to bitmap writing time is approximately 4 to 1.

Figure X.  GDC Video Control Parameters

(full page figure)

## 3.3 DATA LOGIC

The Graphics Option can write in two modes: word mode (16 bits at a time) and vector mode (one pixel at a time).

In word mode, the data patterns to be written into the bitmap are based on bit patterns loaded into the Write Buffer, Write Mask, and the Foreground/Background Register, along with the type of write operation programmed into the ALU/PS Register.

In vector mode, the data patterns to be written to the bitmap are based on bit patterns loaded into the Pattern Register, the Pattern Multiplier, the Foreground/Background Register, and the type of write operation programmed into the ALU/PS Register.

In either case, the data will be stored in the bitmap at a location determined by the addressing logic.

## 3.4 ADDRESS LOGIC

The addressing logic of the Graphics Option is responsible for coming up with the plane, the line within the plane, the word within the line, and even the pixel within the word under some conditions.

The display memory on the Graphics Option is one-dimensional. The GDC scans this linear memory to generate the two dimensional display on the CRT. The video display is organized similarly to the fourth quadrant of the Cartesian plane with the origin in the upper left corner. Row addresses (y coordinates of pixels) start at zero and increase downwards while column addresses (x coordinates of pixels) start at zero and increase to the right (see Figure X). Pixel data is stored in display memory by column within row.

```
                           Column (x)

 Row (y)     0       1       2     ...    n
         +-------------------------//---------+
    0    | (0,0) | (1,0) | (2,0) |    | (n,0) |
         +------------------------    --------+
    1    | (0,1) | (1,1) | (2,1) |    | (n,1) |
         +------------------------    --------+
    2    | (0,2) | (1,2) | (2,2) |    | (n,2) |
    .    +------------------------    --------+
    .    /                                    /
    .    /                                    /
         +------------------------    --------+
    m    | (0,m) | (1,m) | (2,m) |    | (n,m) |
         +-------------------------//---------+
```

Figure X. Rows and Columns in Display Memory

The GDC sees the display memory as a number of 16-bit words where each bit represents a pixel. The number of words defined as well as the number of words displayed on each line is dependent on the resolution. The relationship between words and display lines is shown in Figure X.

```
          |<------------ words/line defined -------------->|
          |<----- words/line displayed ---------->|         |

          +------+-----+-----+---+---+---+-------+---+-----+
 line 0   |   0  |  1  |  2  | --------- |  Q-1  |---| P-1 |
          +------+-----+-----+---+---+---+-------+---|-----+
 line 1   |   P  | P+1 | P+2 | --------- | P+Q-1 |---|2P-1 |
          +------+-----+-----+---+---+---+-------+---|-----+
 line 2   |  2P  |2P+1 | --------------- |2P+Q-1 |---|3P-1 |
          +------+-----+-----+---+---+---+-------+---|-----+
    .     |  3P  | --------------------- |3P+Q-1 |---|4P-1 |
          +------+-----+-----+---+---+---+-------+---+-----+
    .     |  4P  | --------------------- |4P+Q-1 |---+5P-1 |
          +------+-----+-----+---+---+---+-------+---+-----+
    .     /                                                /
          +------+-----+-----+---+---+---+-------+---+-----+
          |(m-1)P| --------------------------------- |mP-1 |
          +------+-----+-----+---+---+---+-------+---+-----+
          /                                                /
          +------+-----+-----+---+---+---+-------+---+-----+
 line n-1 |(n-1)P| --------------------------------- |nP-1 |
          +------+-----+-----+---+---+---+---+---+---+-----+
```

where:

| | |
|---|---|
| P = words/line defined | - 32 in medium resolution.<br>- 64 in high resolution. |
| Q = words/line displayed | - 24 in medium resolution<br>- 50 in high resolution |
| n = no. of lines defined | - 256 |
| m = no. of lines displayed | - 240 |

The GDC requires the word address and the pixel location within that word to address specific pixels. The conversion of pixel locations to memory is accomplished by the following formulas:

Given the pixel (x,y):

```
Word Address of pixel = (words/line defined * y) + integer(x/16)
Pixel Address within word  = remainder(x/16) * 16
```

Because the Graphics Option is a multi-plane device, a way is provided to selectively enable and disable the reading and writing of the individual planes. This function is performed by the ALU/PS and Mode registers. More than one plane at a time can be enabled for a write operation; however, only one plane can be enabled for a read operation at any one time.

The entire address generated by the GDC does not go directly to the bitmap. The low-order six bits address a word within a line in the bitmap and do go directly to the bitmap. The high-order eight bits address the line within the plane and these bits are used as address inputs to a Scroll Map. The Scroll Map acts as a translator such that the bitmap location can be selectively shifted in units of 64 words. In high resolution mode, 64 words equate to one scan line; in medium resolution mode, they equate to two scan lines. This allows the displayed vertical location of an image to be moved in 64-word increments without actually rewriting it to the bitmap. Programs using this feature can provide full and split screen vertical scrolling. The Scroll Map is used in all bitmap access operations: writing, reading, and refreshing.

If an application requires addressing individual pixels within a word, the two 8-bit Write Mask Registers can be used to provide a 16-bit mask that will write-enable selected pixels. Alternately, a single pixel vector write operation can be used.

There is a difference between the number of words/line defined and the number of words/line displayed. In medium resolution, each scan line is 32 words long but only 24 words are displayed (24 words $\overset{1}{16}$ bits/word = 384 pixels). The eight words not displayed are unusable. Defining the length of the scan line as 24 words would be a more efficient use of memory but it would take longer to refresh the memory. Because display memory is organized as a 256 by 256 array, it takes 256 bytes of scan to refresh the entire 64K byte memory. Defining the scan line length as 32 words long enables the entire memory to be refreshed in 4 line scan periods. Defining the scan line length as 24 words long would require 5 line scans plus 16 bytes.

Similarly, in high resolution, each scan line is 64 words long but only 50 words are displayed. With a 64 word scan line length, it takes 2 line scan periods to refresh the entire 64K byte memory. If the scan line length were 50 words, it would take 2 lines plus 56 bytes to refresh the memory.

Another advantage to defining scan line length as 32 or 64 words is that cursor locating can be accomplished by a series of shift instructions which are considerably faster than multiplying.

## 3.5 DISPLAY LOGIC

Data in the bitmap does not go directly to the monitor. Instead, the bitmap data is used as an address into a Color Map. The output of this Color Map, which has been preloaded with color and monochrome intensity values, is the data that is sent to the monitor.

In medium resolution mode there are four planes to the bitmap; each plane providing an address bit to the Color Map. Four bits can address sixteen unique locations at most. This gives a maximum of 16 addressable Color Map entries. Each Color Map entry is 16 bits wide. Four of the bits are used to drive the color monitor's red gun, four go to the green gun, four go to the blue gun, and four drive the output to the monochrome monitor. In systems with the Model 100-A motherboard, only the two low-order bits of the monochrome output are used. Therefore, although there are 16 possible monochrome selections in the Color Map, the number of unique intensities that can be sent to the monochrome monitor is four.

In high resolution mode there are two planes to the bitmap; each plane providing an address bit to the Color Map. Two bits can address four entries in the Color Map at most. Again, each Color Map entry is sixteen bits wide with 12 bits of information used for color and four used for monochrome shades. In systems with the Model 100-A motherboard, only the two low-order bits of the monochrome output are used. This limits the number of unique monochrome intensities to four.

Although the Color Map is 16 bits wide, the color intensity values are loaded one byte at a time. First, the 16 pairs of values representing the red and green intensities are loaded into bits 0 through 7 of the map. Then, the 16 pairs of values representing the blue and monochrome intensities are loaded into bits 8 through 15 of the map.

## 3.6 GDC COMMAND LOGIC

Commands are passed to the GDC command processor from the Rainbow system by writing command bytes to port 57h and parameter bytes to port 56h. Data written to these two ports is stored in the GDC's FIFO buffer, a 16 X 9-bit area that is used to both read from and write to the GDC. The FIFO buffer operates in half-duplex mode -- passing data in both directions, one direction at a time. The direction of data flow at any one time is controlled by GDC commands.

When commands are stored in the FIFO buffer, a flag bit is associated with each data byte depending on whether the data byte was written to the command address (57h) or the parameter address (56h). A flag bit of one denotes a command byte; a flag bit of zero denotes a parameter byte. The command processor tests this flag bit as it interprets the contents of the FIFO buffer.

The receipt of a command byte by the command processor signifies the end of the previous command and any associated parameters. If the command is one that requires a response from the GDC such as RDAT, the FIFO buffer is automatically placed into read mode and the buffer direction is reversed. The specified data from the bitmap is loaded into the FIFO buffer and can be accessed by the system using read operations to port 57h. Any commands or parameters in the FIFO buffer that followed the read command are lost when the FIFO buffer’s direction is reversed.

When the FIFO buffer is in read mode, any command byte written to port 57h will immediately terminate the read operation and reverse the buffer direction to write mode. Any read data that has not been read by the Rainbow system will be lost.

# CHAPTER 4

# SOFTWARE COMPONENTS

## 4.1 I/O PORTS

The CPUs on the Rainbow system's motherboard use a number of 8-bit I/O ports to exchange information with the various subsystems and options. The I/O ports assigned to the Graphics Option are ports 50h through 57h. They are used to generate and display graphic images, inquire status, and read the contents of video memory (bitmap). The function of each of the Graphics Option's I/O ports is as follows:

| Port | Function |
|---|---|
| 50h | Graphics option software reset. Any write to this port also resynchronizes the read/modify/write memory cycles of the Graphics Option to those of the GDC. |
| 51h | Data written to this port is loaded into the area selected by the previous write to port 53h. |
| 52h | Data written to this port is loaded into the Write Buffer. |
| 53h | Data written to this port provides address selection for indirect addressing (see Indirect Register). |
| 54h | Data written to this port is loaded into the low-order byte of the Write Mask. |
| 55h | Data written to this port is loaded into the high-order byte of the Write Mask. |
| 56h | Data written to this port is loaded into the GDC's FIFO Buffer and flagged as a parameter. |

| | |
|---|---|
| | Data read from this port reflects the GDC status. |
| 57h | Data written to this port is loaded into the GDC's FIFO Buffer and flagged as a command. |
| | Data read from this port reflects information extracted from the bitmap. |

## 4.2 INDIRECT REGISTER

There are more registers and storage areas on the Graphics Option module than there are address lines (ports) to accommodate them. The option uses indirect addressing to solve the problem. Indirect addressing involves writing to two ports. A write to port 53h loads the Indirect Register with a bit array in which each bit selects one of eight areas.

The Indirect Register bits and the corresponding areas are as follows:

| Bit | Area Selected |
|---|---|
| 0 | Write Buffer (*) |
| 1 | Pattern Multiplier |
| 2 | Pattern Register |
| 3 | Foreground/Background Register |
| 4 | ALU/PS Register |
| 5 | Color Map (*) |
| 6 | Mode Register |
| 7 | Scroll Map (*) |
| (*) | Also clears the associated index counter |

After selecting an area by writing to port 53h, you access and load data into most selected areas by writing to port 51h. For the Write Buffer however, you need both a write of anything to port 51h to access the buffer and clear the counter and then a write to port 52h to load the data.

## 4.3 WRITE BUFFER

An 16 X 8-bit Write Buffer provides the data for the bitmap when the Graphics Option is in Word Mode. You can use the buffer to transfer blocks of data from the system's memory to the bitmap. The data can be full screen images of the bitmap or bit-pattern representations of font characters that have been stored in main or mass memory. The buffer has an associated index counter that is cleared when the Write Buffer is selected.

Although the CPU sees the Write Buffer as sixteen 8-bit bytes, the GDC accesses the buffer as eight 16-bit words. (See Figure 4.) A 16-bit Write Mask gives the GDC control over individual bits of a word.

```
                 As the CPU sees it                 As the GDC sees it

      byte    High byte     Low byte     word              Word
               7         0   7         0          15                        0
              +-----------+ +-----------+        +---------------------------+
       0,1    |           | |           |    0   |                           |
              +-----------+ +-----------+        +---------------------------+
       2,3    |           | |           |    1   |                           |
              +-----------+ +-----------+        +---------------------------+
       4,5    |           | |           |    2   |                           |
              +-----------+ +-----------+        +---------------------------+
       6,7    |           | |           |    3   |                           |
              +-----------+ +-----------+        +---------------------------+
       8,9    |           | |           |    4   |                           |
              +-----------+ +-----------+        +---------------------------+
      10,11   |           | |           |    5   |                           |
              +-----------+ +-----------+        +---------------------------+
      12,13   |           | |           |    6   |                           |
              +-----------+ +-----------+        +---------------------------+
      14,15   |           | |           |    7   |                           |
              +-----------+ +-----------+        +---------------------------+
```

Figure 4. Write Buffer as Seen by the CPU and the GDC

The output of the Write Buffer is the inverse of its input. If a word is written into the buffer as FFB6h, it will be read out of the buffer as 0049h. To have the same data written out to the bitmap as was received from the CPU requires an added inversion step. You can exclusive or (XOR) the CPU data with FFh to pre-invert the data before going through the Write Buffer. Or, you can write zeros into the Foreground Register and ones into the Background Register to re-invert the data after it leaves the Write Buffer and before it is written to the bitmap. Use one method or the other, not both.

In order to load data into the Write Buffer, you first write an FEh to port 53h and any value to port 51h. This not only selects the Write Buffer but also clears the Write Buffer Index Counter to zero. The data is then loaded into the buffer by writing it to port 52h in high-byte low-byte order. If more than 16 bytes are written to the buffer the first 16 bytes will be overwritten.

If you load the buffer with less than 16 bytes (or other than a multiple of 16 bytes for some reason or other) the GDC will find an index value other than zero in the counter. Starting at a location other than zero will alter the data intended for the bitmap. Therefore, before the GDC is given the command to write to the bitmap, you must again clear the Write Buffer Index Counter so that the GDC will start accessing the data at word zero.

## 4.4 WRITE MASK REGISTERS

When the Graphics Option is in Word Mode, bitmap operations are carried out in units of 16-bit words. A 16-bit Write Mask is used to control the writing of individual bits within a word. A zero in a bit position of the mask allows writing to the corresponding position of the word. A one in a bit position of the mask disables writing to the corresponding position of the word.

While the GDC sees the mask as a 16-bit word, the CPU sees the mask as two of the Graphic Option's I/O ports. The high-order Write Mask Register is loaded with a write to port 55h and corresponds to bits 15 through 8 of the Write Mask. The low-order Write Mask Register is loaded with a write to port 54h and corresponds to bits 7 through 0 of the Write Mask. (See Figure 5.)

```
                 As seen by
                  the CPU
        Port 55h             Port 54h
           |                    |
           V                    V
    7------------------0 7------------------0

    +--------------------+--------------------+
    | Write Mask (high) | Write Mask (low)  |
    +-----------------------------------------+

    15--------------------------------------0

              Word As Seen By GDC
```

Figure 5. Write Mask Registers

## 4.5 PATTERN GENERATOR

When the Graphics Option is in vector mode, the Pattern Generator provides the data to be written to the bitmap. The Pattern Generator is composed of a Pattern Register and a Pattern Multiplier.

The Pattern Register is an 8-bit recirculating shift register that is first selected by writing FBh to port 53h and then loaded by writing an 8-bit data pattern to port 51h.

The Pattern Multiplier is a 4-bit register that is first selected by writing FDh to port 53h and then loaded by writing a value of 0-Fh to port 51h.

NOTE

You must load the Pattern Multiplier before loading the Pattern Register.

Figure 6 shows the logic of the Pattern Generator. Data destined for the bitmap originates from the low-order bit of the Pattern Register. That same bit continues to be the output until the Pattern Register is shifted. When the most significant bit of the Pattern Register has completed its output cycle, the next bit to shift out will be the least significant bit again.

```
                                              Option
                                               Clock
                                                 |
                                                 |
                                              3  V  0
      Pattern Multiplier             +-------+-------+
      (Loaded from CPU) ----->       |       |M'plier|
                                     +-------+-------+
                                                    |
                                               Shift|
                                               Clock|
                                                    V
                                      7             0
       Pattern Register              +---------------+
       (Loaded From CPU) ---->   +-->| Data Pattern  |--+
                                 |   +--------------v+  |
                                 |                  |   |
                                 +---------<--------+   |
                      Shifted Bits Recirculated         |
                                                        V
                                                 Data Bit Output
                                                To Write Circuitry
```

Figure 6. Pattern Generator

The shift frequency is the write frequency from the option clock divided by 16 minus the value in the Pattern Multiplier. For example, if the value in the Pattern Multiplier is 12, the shift frequency divisor would be 16 minus 12 or 4. The shift frequency would be one fourth of the write frequency and therefore each bit in the Pattern Register would be replicated in the output stream four times. A multiplier of 15 would take 16 - 15 or 1 write cycle for each Pattern Register bit shifted out. A multiplier of 5 would take 16 - 5 or 11 write cycles for each bit in the Pattern Register.

## 4.6 FOREGROUND/BACKGROUND REGISTER

The Foreground/Background Register is an eight-bit write-only register. The high-order nibble is the Foreground Register; the low-order nibble is the Background Register. Each of the four bitmap planes has a Foreground/Background bit-pair associated with it (see Figure 7). The bit settings in the Foreground/Background Register, along with the write mode specified in the ALU/PS Register, determine the data that is eventually received by the bitmap. For example; if the write mode is REPLACE, an incoming data bit of zero is replaced by the corresponding bit in the Background Register. If the incoming data bit is a one, the bit would be replaced by the corresponding bit in the Foreground Register.

```
   Foreground      Background
|7  Register   4|3  Register   0|
+-------------------------------+
| 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
+-------------------------------+
  |   |   |   |   |   |   |   |
  +---)---)---)---'   |   |   |
  |   |   |   |       |   |   |
  |   +---)---)-------'   |   |
  |   |   |   |           |   |
  |   |   +---)-----------'   |
  |   |   |   |               |
  |   |   |   +---------------'
  |   |   |   |
  |   |   |   V
  |   |   | +------------------------+
  |   |   V | PLANE 0                |
  |   | +------------------------+   |
  |   V | PLANE 1                |   |
  | +------------------------+   |
  V | PLANE 2                |   |
+------------------------+   |
| PLANE 3                |   |
|                        |
|                        |
```

Figure 7.  Foreground/Background Register

Each bitmap plane has its own individual Foreground/Background bit pair.  Therefore, it is possible for two enabled planes to use the same incoming data pattern and end up with different bitmap patterns.

## 4.7  ALU/PS REGISTER

The ALU/PS Register has two functions.

Bits 0 through 3 of the ALU/PS Register are used to inhibit writes to one or more of the bitmap planes.  If this capability was not provided, each write operation would affect all available planes.  When a plane select bit is set to one, writes to that plane will be inhibited.  When a plane select bit is cleared to zero, writes to that plane will be allowed.

NOTE

During a readback mode operation, all plane select bits should be set to ones to prevent accidental changes to the bitmap data.

Bits 4 and 5 of the ALU/PS Register define an arithmetic logic unit function. The three logic functions supported by the option are REPLACE, COMPLEMENT, and OVERLAY. These functions operate on the incoming data from the Write Buffer or the Pattern Generator as modified by the Foreground/Background Register as well as the current data in the bitmap and generate the new data to be placed into the bitmap.

When the logic unit is operating in REPLACE mode, the current data in the bitmap is replaced by the Foreground/Background data selected as follows:

- An incoming data bit "0" selects the Background data.
- An incoming data bit "1" selects the Foreground data.

When the logic unit is operating in COMPLEMENT mode, the current data in the bitmap is modified as follows:

- An incoming data bit "0" results in no change.
- An incoming data bit "1" results in the current data being exclusive or'ed (XOR) with the appropriate Foreground bit. If the Foreground bit is a "0", the current data is unchanged. If the Foreground bit is a "1", the current data is complemented by binary inversion. In effect, the Foreground Register acts as a plane select register for the complement operation.

When the logic unit is operating in OVERLAY mode, the current data in the bitmap is modified as follows:

- An incoming data bit "0" results in no change.
- An incoming data bit "1" results in the current data being replaced by the appropriate Foreground bit.

## 4.8 COLOR MAP

The Color Map is a 16 X 16-bit RAM area where each of the 16 entries is composed of four 4-bit values representing color intensities. These values represent, from high order to low order, the monochrome, blue, red, and green outputs to the video monitor. Intensity values are specified in inverse logic. At one extreme, a value of zero represents maximum intensity (100% output) for a particular color or monochrome shade. At the other extreme, a value of 15 (Fh) represents minimum intensity (zero output).

Bitmap data is not directly displayed on the monitor, each bitmap plane contributes one bit to an index into the Color Map. The output of the Color Map is the data that is passed to the monitor. Four bitmap planes (medium resolution) provide four bits to form an index allowing up to 16 intensities of color or monochrome to be simultaneously displayed on the monitor. Two bitmap planes (high resolution) provide two bits to form an index allowing only four intensities of color or monochrome to be simultaneously displayed on the monitor.

In Figure 8, a medium resolution configuration, the bitmap data for the display point x,y is 0110b (6 decimal). This value, when applied as an index into the Color Map, selects the seventh entry out of a possible sixteen. Each Color Map entry is sixteen bits wide. Four of the bits are used to drive the color monitor’s red gun, four go to the green gun, four go to the blue gun, and four drive the output to the monochrome monitor. The twelve bits going to the color monitor support a color palette of 4096 colors; the four bits to the monochrome monitor support 16 shades. (In systems with the Model 100−A motherboard, only the two low−order bits of the monochrome output are active. This limits the monochrome output to four unique intensities.)

```
                          Bitmap          /                       Color Map
                   +---------------------/-----+   Bitmap Data    +-------+
                   | Plane 0            0      |-----\          0 |       |
                +----------------------/----+  |      |         . |       |
                | Plane 1             1     |---------|         . |       |
             +-----------------------/---+  |          >-0110b->6 |. . . .|
             | Plane 2              1    |------------|         . || | | ||
          +------------------------/--+  |            |         . || | | ||
          | Plane 3               0   |--------------/          . || | | ||
          |                      /    |                         . || | | ||
          |                (x,y)   |                        15 || | | ||
                                                                  +|-|-|-|+
                                                                   | | | |
               4(*) bits of Monochrome level to Mono. Monitor<-----' | | |
                                                                     | | |
                        4 bits of Blue level to Color Monitor<-------' | |
                                                                       | |
                         4 bits of Red level to Color Monitor<---------' |
                                                                         |
                       4 bits of Green level to Color Monitor<-----------'

          (*) 2 low-order bits on Model 100-A systems
```

Figure 8. Bitmap/Color Map Interaction (medium resolution)

In Figure 9, a high resolution configuration, the bitmap data for point (x,y) is 10b (2 decimal). This value, when applied as an index into the Color Map, selects the third entry out of a possible four. Again, each Color Map entry is sixteen bits wide and 12 bits of information are used for color and four are used for monochrome. (In systems with the Model 100-A motherboard, only the two low-order bits of the monochrome output are active. This limits the monochrome output to four unique intensities.)

```
                         Bitmap          /                       Color Map
                  +---------------------/-----+   Bitmap Data    +-------+
                  | Plane 0            0      |                0 |       |
                +---------------------/----+  |------\         1 |       |
                | Plane 1            1     |          >--10b-->2 |. . . .|
                |                   /      |---------/         3 || | | ||
                                 (x,y)                         . || | | ||
                                                               . || | | ||
                                                               . || | | ||
                                                               . || | | ||
                                                              15 || | | ||
                                                                 +|-|-|-|+
                                                                  | | | |
             4(*) bits of Monochrome level to Mono. Monitor<-----' | | |
                                                                    | | |
                       4 bits of Blue level to Color Monitor<-------' | |
                                                                      | |
                        4 bits of Red level to Color Monitor<---------' |
                                                                        |
                      4 bits of Green level to Color Monitor<-----------'

            (*) 2 low-order bits on Model 100-A systems
```

Figure 9. Bitmap/Color Map Interaction (high resolution)

## 4.8.1 Loading The Color Map

From the graphic option's point of view, the Color Map is composed of 16 sixteen-bit words. However, from the CPU's point of view the Color Map is composed of 32 eight-bit bytes. The 32 bytes of intensity values are loaded into the Color Map one entire column of 16 bytes at a time. The red and green values are always loaded first, then the monochrome and blue values. (See Figure 10.)

| | 2nd 16 bytes loaded by the CPU | | 1st 16 bytes loaded by the CPU | | | |
|---|---|---|---|---|---|---|
| address value | mono. data | blue data | red data | green data | color displayed | monochrome displayed |
| 0 | 15 | 15 | 15 | 15 | black | black |
| 1 | 14 | 15 | 0 | 15 | red | . . . |
| 2 | 13 | 15 | 15 | 0 | green | gray |
| 3 | 12 | 0 | 15 | 15 | blue | |
| 4 | 11 | 0 | 0 | 15 | magenta | shades |
| 5 | 10 | 0 | 15 | 0 | cyan | |
| 6 | 9 | 15 | 0 | 0 | yellow | |
| . . . . | | | | | . . . . | . . . |
| 15 | 0 | 0 | 0 | 0 | white | white |

Figure 10. Sample Color Map With Loading Sequence

Writing the value DFh to port 53h selects the Color Map and also clears the Color Map Index Counter to zero. To load data into the Color Map requires writing to port 51h. Each write to port 51h will cause whatever is on the 8088 data bus to be loaded into the current Color Map location. After each write,the Color Map Index Counter is incremented by one. If 33 writes are made to the Color Map, the first Color Map location will be overwritten.

## 4.9 MODE REGISTER

The Mode Register is an 8-bit multi-purpose register that is loaded by first selecting it with a write of BFh to port 53h and then writing a data byte to port 51h. The bits in the Mode Register have the following functions:

- o Bit 0 determines the resolution mode:

  0 = medium resolution mode (384 pixels across)
  1 = high resolution mode (800 pixels across)

- o Bit 1 determines the write mode:

  0 = word mode, 16 bits/RMW cycle, data comes from Write Buffer
  1 = vector mode, 1 bit/RMW cycle, data comes from Pattern Generator

- o Bits 3 and 2 select a bitmap plane for readback mode operation:

  00 = plane 0
  01 = plane 1
  10 = plane 2
  11 = plane 3

- o Bit 4 determines the option's mode of operation:

  0 = read mode, plane selected by bits 3 and 2 is enabled for readback
  1 = write mode, writes to the bitmap allowed but not mandatory

- o Bit 5 controls writing to the Scroll Map:

  0 = writing is enabled (after selection by the Indirect Register)
  1 = writing is disabled

- o Bit 6 controls the interrupts generated by the Graphics Option every time the GDC issues a vertical sync pulse:

  0 = interrupts to the CPU are disabled (if an interrupt has already occurred when this bit is set to zero, the pending interrupt is cleared)
  1 = interrupts to the CPU are enabled

- o Bit 7 controls the video data output from the option:

  0 = output is disabled (all other operations on the graphics board still take place)
  1 = output is enabled

## 4.10 SCROLL MAP

The Scroll Map is a 256 X 8-bit recirculating ring buffer that is used to offset scan line addresses in the bitmap in order to provide full and split-screen vertical scrolling. The entire address as generated by the GDC does not go directly to the bitmap. Only the low-order six bits of the GDC address go directly to the bitmap. They represent one of the 64 word addresses that are the equivalent of one scan line in high resolution mode or two scan lines in medium resolution mode. The eight high-order bits of the GDC address represent a line address and are used as an index into the 256-byte Scroll Map. The eight bits at the selected location then become the new eight high-order bits of the address that the bitmap sees. (See Figure 11.) By manipulating the contents of the Scroll Map, you can perform quick dynamic relocations of the bitmap data in 64-word chunks.

```
      GDC address                  word address
        bits 0-5 -------------------------------------+
         (word)                                       |
                      7        0                      |
                     +-----------+             +------v-----------+
                    0|           |             |      .           |
                     |           |             |      .           |
      GDC address    |           |             |      .           |
       bits 6-13 --->| xxxxxxxx  |             |      .           |
         (line)      |    .      |             |      .           |
                     |    .      |             |      .           |
                     |    .      |         +--->.....wd...........|
                     |    .      |         |   |                  |
                  255|    .      |   offset|   |                  |
                     +-----------+    scan |   +------------------+
                          |           line |
                          +----------------+

                      Scroll Map                      Bitmap
```

Figure 11.  Scroll Map Operation

### 4.10.1 Loading The Scroll Map

Start loading the offset addresses into the Scroll Map at the beginning of a vertical retrace. First set bit 5 of the Mode Register to zero to enable the Scroll Map for writing. Write a 7Fh to port 53h to select the Scroll Map and clear the Scroll Map Index Counter to zero. Then do a series of writes to port 51h with the offset values to be stored in the Scroll Map. Loading always begins at location zero of the Scroll Map. With each write, the Scroll Map Index Counter is automatically incremented until the write operations terminate. If there are more than 256 writes, the index counter loops back to Scroll Map location zero. This also means that if line 255 requires a change, lines 0-254 will have to be rewritten first.

All 256 scroll map entries should be defined even if all 256 addresses are not displayed. This is to avoid mapping undesirable data onto the screen. After the last write operation, bit 5 of the Mode Register should be set to one to disable further writing to the Scroll Map.

The time spent in loading the Scroll Map should be kept as short as possible. During loading, the GDC’s address lines no longer have a path to the bitmap and therefore memory refresh is not taking place. Delaying memory refresh can result in lost data.

While it is possible to read out of the Scroll Map, time constraints preclude doing both a read and a rewrite during the same vertical retrace period. If necessary, a shadow image of the Scroll Map can be kept in some area in memory. The shadow image can be updated at any time and then transferred into the Scroll Map during a vertical retrace.

# PART II

# PROGRAMMING GUIDELINES

# CHAPTER 5

# INITIALIZATION AND CONTROL

The examples in this chapter cover the initialization of the Graphics Display Controller (GDC) and the Graphics Option , the control of the graphics output, and the management of the option's color palette.

## 5.1 TEST FOR OPTION PRESENT

Before starting any application, you should ensure that the Graphics Option has been installed on the Rainbow system. Attempting to use the Graphics Option when it is not installed can result in a system reset that may in turn result in the loss of application data. The following code will test for the option's presence.

### 5.1.1 Example Of Option Test

```
;*******************************************************************
;                                                                  *
;       p r o c e d u r e   o p t i o n _ p r e s e n t _ t e s t  *
;                                                                  *
;       purpose:        test if Graphics Option is present.        *
;       entry:          none.                                      *
;       exit:           dl = 1          option present.            *
;                       dl = 0          option not present.        *
;       register usage: ax,dx                                      *
;*******************************************************************
cseg    segment byte    public  'codesg'
        public  option_present_test
        assume  cs:cseg,ds:nothing,es:nothing,ss:nothing
option_present_test     proc    near
        mov     dl,1            ;set dl for option present
        in      al,8            ;input from port 8
        test    al,04h          ;test bit 2 to see if option present
        jz      opt1            ;if option is present, exit
        xor     dl,dl           ;else, set dl for option not present
opt1:   ret
option_present_test     endp
cseg    ends
        end
```

## 5.2 TEST FOR MOTHERBOARD VERSION

When you initially load or subsequently modify the Color Map, it may be necessary to know what version of the motherboard is installed in the Rainbow system. The code to determine this is operating system dependent. The examples in the following sections are written for CP/M, MS−DOS, and Concurrent CP/M.

### 5.2.1 Example Of Version Test For CP/M System

```
;*********************************************************************
;                                                                   *
;       p r o c e d u r e   t e s t _ b o a r d _ v e r s i o n     *
;                                                                   *
;       purpose:        Test motherboard version                    *
;       restriction:    This routine will work under cp/m only.     *
;       entry:          none.                                       *
;       exit:           flag :=         0 = 'A' motherboard         *
;                                       1 = 'B' motherboard         *
;       register usage: ax,bx,cx,dx,di,si,es                        *
;*********************************************************************
;
;
        dseg
flag    db      000h
buffer  rs      14              ;reserve 14 bytes
        cseg
test_board_version:
        push    bp
        mov     ax,ds           ;clear buffer, just to be sure
        mov     es,ax           ;point es:di at it
        mov     di,0
        mov     cx,14           ;14 bytes to clear
        xor     al,al           ;clear clearing byte
```

```
opt1:   mov     buffer[di],al   ;do the clear
        inc     di
        loop    opt1            ;loop till done
        mov     ax,ds           ;point bp:dx at buffer for
        mov     bp,ax           ; int 40 call
        mov     dx,offset buffer
        mov     di,1ah          ;set opcode for call to get hw #
        int     40
        mov     si,0
        mov     cx,8            ;set count for possible return ASCII
opt2:   cmp     buffer[si],0
        jne     opt3            ;got something back, have rainbow 'B'
        inc     si
        loop    opt2            ;loop till done
        mov     flag,0          ;no ASCII, set rainbow 'A' type
        jmp     opt4
opt3:   mov     flag,1          ;got ASCII, set rainbow 'B' type
opt4:   pop     bp
        ret
```

## 5.2.2 Example Of Version Test For MS-DOS System

```
;*********************************************************************
;                                                                    *
;       p r o c e d u r e   t e s t _ b o a r d _ v e r s i o n      *
;                                                                    *
;       purpose:        test motherboard version                     *
;       restriction:    this routine will work under MS-DOS only     *
;       entry:          none                                         *
;       exit:           flag :=         0 = 'A' motherboard          *
;                                       1 = 'B' motherboard          *
;       register usage: ax,bx,cx,dx,di,si                            *
;*********************************************************************
;
cseg    segment byte    public  'codesg'
        public  test_board_version
        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
;
test_board_version      proc    near
        push    bp                      ;save bp
        mov     di,0                    ;clear buffer to be sure
        mov     cx,14                   ;14 bytes to clear
        xor     al,al                   ;clear clearing byte
tb1:    mov     byte ptr buffer[di],al  ;do the clear
```

```
        inc     di
        loop    tb1                     ;loop till done
        mov     ax,ds                   ;point bp:dx at buffer for
        mov     bp,ax                   ; int 18h call
        mov     dx,offset buffer
        mov     di,1ah          ;set opcode for call to get hw #
        int     18h             ;int 40 remapped to 18h under MS-DOS
        mov     si,0
        mov     cx,8            ;set count for possible return ASCII
tb2:    cmp     byte ptr buffer[si],0
        jne     tb3             ;got something back, have rainbow 'B'
        inc     si
        loop    tb2
        mov     flag,0          ;no ASCII, set rainbow 'A' type
        jmp     tb4
tb3:    mov     flag,1          ;got ASCII, set rainbow B type
tb4:    pop     bp              ;recover bp
        ret
test_board_version      endp
cseg    ends
dseg    segment byte    public  'datasg'
        public  flag
flag    db      0
buffer  db      14      dup (?)
dseg    ends
        end
```

## 5.2.3 Example Of Version Test For Concurrent CP/M System

```
;*********************************************************************
;                                                                    *
;       p r o c e d u r e   t e s t _ b o a r d _ v e r s i o n     *
;                                                                    *
;       purpose:        test motherboard version                     *
;       restriction:    this routine for Concurrent CP/M only        *
;       entry:          none                                         *
;       exit:           flag :=         0 = 'A' motherboard          *
;                                       1 = 'B' motherboard          *
;       register usage: ax,bx,cx,dx,si                               *
;*********************************************************************
;
test_board_version:
        mov     control_b+2,ds
        mov     di,offset biosd
        mov     bx,3
        mov     [di+bx],ds
        mov     dx,offset biosd         ;setup for function 50 call
        mov     cl,32h
        int     0e0h                    ;function 50
        mov     flag,0                  ;set flag for rainbow 'A'
        mov     bx,6                    ;offset to array_14
        mov     si,offset array_14
        mov     al,'0'
        cmp     [si+bx],al              ;'0', could be a rainbow 'A'
        jne     found_b                 ;no, must be rainbow 'B'
        inc     bx                      ;next number...
        mov     al,'1'                  ;can be either 1...
        cmp     [si+bx],al
        je      test_board_exit
        mov     al,'2'                  ;or 2 ...
        cmp     [si+bx],al
        je      test_board_exit
        mov     al,'3'                  ;or 3 if its a rainbow 'A'
        cmp     [si+bx],al
        je      test_board_exit
found_b:
        mov     flag,1                  ;its a rainbow 'B'
test_board_exit:
        ret
        dseg
biosd           db      80h
                dw      offset control_b
                dw      0
control_b       dw      4
                dw      0
                dw      offset array_14
array_14        rs      14
flag            db      0
        end
```

## 5.3 INITIALIZE THE GRAPHICS OPTION

Initializing the Graphics Option can be separated into the following three major steps:

- o Reset the GDC to the desired display environment.
- o Initialize the rest of the GDC's operating parameters.

- o Initialize the Graphic Option's registers, buffers, and maps.

## 5.3.1 Reset The GDC

To reset the GDC, give the RESET command with the appropriate parameters followed by commands and parameters to set the initial environment. The RESET command is given by writing a zero byte to port 57h. The reset command parameters are written to port 56h.

The GDC Reset Command parameters are the following:

| Parameter | Value | Meaning |
|---|---|---|
| 1 | 12h | The GDC is in graphics mode<br>Video display is noninterlaced<br>No refresh cycles by the GDC<br>Drawing permitted only during retrace |
| 2 | 16h<br>30h | For medium resolution<br>For high resolution<br><br>The number of active words per line, less 2. There are 24 (18h) active words per line in medium resolution mode and 50 (32h) words per line in high resolution mode. |
| 3 | 61h<br>64h | For medium resolution<br>For high resolution<br><br>The lower-order five bits are the horizontal sync width in words, less one (med. res. HS=2, high res. HS=5). The high-order three bits are the low-order three bits of the vertical sync width in lines (VS=3 lines). |
| 4 | 04h<br>08h | For medium resolution<br>For high resolution<br><br>The low-order two bits are the high-order two bits of the vertical sync width in lines. The high-order six bits are the horizontal front porch width in words, less one (med. res. HFP=2, high res. HFP=3). |
| 5 | 02h<br>03h | For medium resolution<br>For high resolution |

| | | |
|---|---|---|
| | | Horizontal back porch width in words, less one (med. res. HBP=3, high res. HBP=4). |
| 6 | 03h | Vertical front porch width in lines (VFP=3). |
| 7 | F0h | Number of active lines per video field (single field, 240 line display). |
| 8 | 40h | The low-order two bits are the high-order two bits of the number of active lines per video field. The high-order six bits are the vertical back porch width in lines (VBP=16). |

## 5.3.2 Initialize The GDC

Now that the GDC has been reset and the video display has been defined, you can issue the rest of the initialization commands and associated parameters by writing to ports 57h and 56h respectively.

Start the GDC by issuing the START command (6Bh).

ZOOM must be defined; however, since there is no hardware support for the Zoom feature, program a zoom magnification factor of one by issuing the ZOOM command (46h) with a parameter byte of 00.

Issue the WDAT command (22h) to define the type of Read/Modify/Write operations as word transfers - low byte, then high byte. No parameters are needed at this time because the GDC is not being asked to do a write operation; it is only being told how to relate to the memory.

Issue the PITCH command (47h) with a parameter byte of 20h for medium resolution or 40h for high resolution to tell the GDC that each scan line begins 32 words after the previous one for medium resolution and 64 words after the previous one for high resolution. Note, however, that only 24 or 50 words are displayed on each screen line. The undisplayed words left unscanned are unusable.

The GDC can simultaneously display up to four windows. The PRAM command defines the window display starting address in words and its length in lines. The Graphics Option uses only one display window with a starting address of 0000 and a length of 256 lines. To set this up, issue the PRAM command (70h) with four parameter bytes of 00,00,F0,0F.

Another function of the GDC's parameter RAM is to hold soft character fonts and line patterns to be drawn into the bitmap. The Graphics Option, rather than using the PRAM for this purpose, uses the external Character RAM and Pattern Generator. For the external hardware to work properly, the PRAM command bytes 9 and 10 must be loaded with all ones. Issue the PRAM command (78h) with two parameter bytes of FF,FF.

Issue the CCHAR command (4Bh) with three parameter bytes of 00,00,00, to define the cursor characteristics as being a non-displayed point, one line high.

Issue the VSYNC command (6Fh) to make the GDC operate in master sync mode.

Issue the SYNC command (0Fh) to start the video refresh action.

The GDC is now initialized.

### 5.3.3 Initialize The Graphics Option

First you must synchronize the Graphics Option with the GDC's write cycles. Reset the Mode Register by writing anything to port 50h and then load the Mode Register.

Next, load the Scroll Map. Wait for the start of a vertical retrace, enable Scroll Map addressing, select the Scroll Map, and load it with data.

Initialize the Color Map with default data kept in a shadow area. The Color Map is a write-only area and using a shadow area makes the changing of the color palette more convenient.

Set the Pattern Generator to all ones in the Pattern Register and all ones in the Pattern Multiplier.

Set the Foreground/Background Register to all ones in the foreground and all zeros in the background.

Set the ALU/PS Register to enable all four planes and put the option in REPLACE mode.

Finally, clear the screen by setting the entire bitmap to zeros.

### 5.3.4 Example Of Initializing The Graphics Option

The following example is a routine that will initialize the Graphics Option including the GDC. This initialization procedure leaves the bitmap cleared to zeros and enabled for writing but with gzaphics output turned off. Use the procedure in the next section to turn the graphics output on. Updating of the bitmap is independent of whether the graphics output is on or off.

```
;*******************************************************************
;                                                                  *
;       p r o c e d u r e   i n i t _ o p t i o n                  *
;                                                                  *
;       purpose:        initialize the graphics option             *
;                                                                  *
;       entry:          dx = 1     medium resolution               *
;                       dx = 2     high resolution                 *
;       exit:           all shadow bytes initialized               *
;       register usage: none, all registers are saved              *
;*******************************************************************
cseg    segment byte    public  'codesg'
extrn   alups:near,pattern_register:near,pattern_mult:near,fgbg:near
        public  init_option
        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
init_option     proc    near
        push    ax              ;save the registers
        push    bx
        push    cx
        push    dx
        push    di
        push    si
        cld                     ;make sure that stos incs.
;
;First we have to find out what the interupt vector is for the
;graphics option.  If this is a Model 100-A, interrupt vector
;22h is the graphics interrupt.  If this is a Model 100-B, the
;interrupt vector is relocated up to A2.  If EE00:0F44h and
;04<>0, we have the relocated vectors of a Model 100-B and need
;to OR the msb of our vector.
;
        mov     ax,ds
        mov     word ptr cs:segment_save,ax
        push    es              ;save valid es
        mov     bx,0ee00h       ;test if vectors are relocated
        mov     es,bx
        mov     ax,88h               ;100-A int. vector base addr
        test    es:byte ptr 0f44h,4  ;relocated vectors?
        jz      g0                   ;jump if yes
        mov     ax,288h              ;100-B int. vector base addr
g0:     mov     word ptr g_int_vec,ax
        pop     es
        cmp     dx,1            ;medium resolution?
        jz      mid_res         ;jump if yes
        jmp     hi_res          ;else is high resolution
mid_res:
        mov     al,00           ;medium resolution reset command
        out     57h,al
        mov     gbmod,030h      ;mode = med res, text, no readback
        call    mode            ;turn off graphics output
        mov     al,12h          ;p1. refresh, draw enabled during
        out     056h,al         ;retrace
        mov     al,16h          ;p2. 24 words/line minus 2
        out     056h,al         ;384/16 pixels/word=24 words/line
        mov     al,61h          ;p3. 3 bits vs/5 bits hs width - 1
        out     056h,al         ;vs=3, hs=2
        mov     al,04           ;p4. 6 bits hfp-1, 2 bits vs high
        out     056h,al         ;byte, 2 words hfp, no vs high byte
        mov     al,02           ;p5. hbp-1, 3 words hbp
        out     056h,al
        mov     al,03           ;p6. vertical front porch, 3 lines
        out     056h,al
        mov     al,0f0h         ;p7. active lines displayed
        out     056h,al
        mov     al,40h          ;p8. 6 bits vbp/2 bits lines/field
        out     056h,al         ;high byte, vbp=16 lines
        mov     al,047h         ;pitch command, med res, straight up
        out     057h,al
        mov     al,32           ;med res memory width for vert. pitch
        out     056h,al
        mov     word ptr nmritl,3fffh
        mov     word ptr xmax,383       ;384 pixels across in med res
        mov     byte ptr num_planes,4   ;4 planes in med res
        mov     byte ptr shifts_per_line,5 ;rotates for 32 wds/line
        mov     byte ptr words_per_line,32 ;words in a line
        jmp     common_init
hi_res: mov     al,00           ;high resolution reset command
        out     57h,al
        mov     gbmod,031h      ;mode = high res, text, no readback
        call    mode            ;disable graphics output
        mov     al,12h          ;p1. refresh, draw enabled during
        out     056h,al         ;retrace
        mov     al,30h          ;p2. 50 words/line - 2
        out     056h,al
        mov     al,64h          ;p3. hsync w-1=4(low 5 bits), vsync
        out     056h,al         ;w=3(upper three bits)
        mov     al,08           ;p4. hor fp w-1=2(upper 2 bits),
        out     056h,al         ;vsync high byte = 0
        mov     al,03           ;p5. hbp-1. 3 words hbp
        out     056h,al
        mov     al,03           ;p6. vertical front porch, 3 lines
        out     056h,al
        mov     al,0f0h         ;p7. active lines displayed
        out     056h,al
        mov     al,40h          ;p8. 6 bits vbp/2 bits lines per field
        out     056h,al         ;high byte. vbp=16 lines
        mov     al,047h         ;pitch command, high res, straight up
        out     057h,al
        mov     al,64           ;high res pitch is 64 words/line
        out     056h,al
        mov     word ptr nmritl,7fffh
        mov     word ptr xmax,799       ;800 pixels across
        mov     byte ptr num_planes,2   ;2 planes in high res
        mov     byte ptr shifts_per_line,6 ;shifts for 64 wds/line
        mov     byte ptr words_per_line,64 ;number of words/line
common_init:
        mov     al,00           ;setup start window display for memory
        mov     startl,al       ;location 00
        mov     starth,al
        mov     al,06bh         ;start command
        out     057h,al         ;start the video signals going
        mov     al,046h         ;zoom command
        out     057h,al
        mov     al,0            ;magnification assumed to be 0
        out     056h,al
        mov     al,22h          ;setup R/M/W memory cycles for
        out     57h,al          ;figure drawing
;
;Initialize PRAM command. Start window at the address in startl,
;starth.  Set the window length for 256 lines. Fill PRAM parameters
;8 and 9 with all ones so GDC can do graphics draw commands without
;altering the data we want drawn.
;
        mov     al,070h         ;issue the pram command, setup
        out     057h,al         ;GDC display
        mov     al,startl       ;p1. display window starting address
        out     056h,al         ;low byte
        mov     al,starth       ;p2. display window starting address
        out     056h,al         ;high byte
        mov     al,0ffh         ;p3. make window 256 lines
        out     056h,al
        mov     al,0fh          ;p4. high nibble display line on
        out     056h,al         ;right, the rest = 0
        mov     al,078h         ;issue pram command pointing to p8
        out     057h,al
        mov     al,0ffh         ;fill pram with ones pattern
        out     056h,al
        out     056h,al
        mov     al,04bh         ;issue the cchar command
        out     057h,al
        xor     al,al           ;initialize cchar parameter bytes
        mov     cchp1,al        ;graphics cursor is one line, not
        out     056h,al         ;displayed, non-blinking
        mov     cchp2,al
        out     056h,al
        mov     cchp3,al
        out     056h,al
        mov     al,06fh         ;vsync command
        out     057h,al
        out     050h,al         ;reset the graphics board
        mov     al,0bfh
        out     53h,al
        mov     al,byte ptr gbmod  ;enable, then disable interrupts
        or      al,40h                  ;to flush the interrupt hardware
        out     51h,al                  ;latches
        mov     cx,4920                 ;wait for a vert sync to happen
g1:     loop    g1
        mov     al,0bfh                 ;disable the interrupts
        out     53h,al
        mov     al,byte ptr gbmod
        out     51h,al
        call    assert_colormap         ;load colormap
        call    inscrl                  ;initialize scroll map
        mov     bl,1                    ;set pattern multiplier to 16-bl
        call    pattern_mult            ;see example "pattern_mult"
        mov     bl,0ffh                 ;set pattern data of all bits set
        call    pattern_register        ;see example "pattern_register"
        mov     bl,0f0h                 ;enable all foreground registers
        call    fgbg                    ;see example "fgbg"
        mov     bl,0                    ;enable planes 0-3, REPLACE logic
        call    alups                   ;see example "alups"
        mov     di,offset p1            ;fill the p table with ff's.
        mov     al,0ffh
        mov     cx,16
        rep     stosb
        mov     al,0                    ;enable all gb mask writes.
        mov     gbmskl,al
        mov     gbmskh,al
        mov     al,0ffh                 ;set GDC mask bits
        mov     gdcml,al
        mov     gdcmh,al
        mov     word ptr curl0,0        ;set cursor to top screen left
        mov     ax,word ptr gbmskl      ;fetch and issue the graphics
        out     54h,al                  ;option text mask
        mov     al,ah
        out     55h,al
        call    setram                       ;then set ram to p1 thru p16 data
        mov     word ptr ymax,239
        mov     al,0dh
        out     57h,al                  ;enable the display
        pop     si                      ;recover the registers
        pop     di
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
init_option     endp
;
;*********************************************************************
;*                                                                   *
;*      g r a p h i c s   s u b r o u t i n e s                      *
;*                                                                   *
;*********************************************************************
```

```
;
gsubs   proc near
public  setram,assert_colormap,gdc_not_busy,imode,color_int,scrol_int
public  cxy2cp,mode
;
;*********************************************************************
;                                                                   *
;       s u b r o u t i n e    a s s e r t _ c o l o r m a p        *
;                                                                   *
;       colormap is located at clmpda which is defined in           *
;       procedure "set_color"                                       *
;                                                                   *
;       entry:          clmpda = colormap to be loaded              *
;       exit:           none                                        *
;       register usage: ax,bx                                       *
;*********************************************************************
;
;
assert_colormap:
        cld
        call    gdc_not_busy    ;make sure nothing's happening
;
;The graphics interrupt vector "giv" is going to be either 22h or
;A2h depending on whether this is a Model 100-A or a Model 100-B
;with relocated vectors. Read the old vector, save it, then
;overwrite it with the new vector.
;
        push    es
        xor     ax,ax
        mov     es,ax
        mov     bx,word ptr g_int_vec   ;fetch address of "giv"
        cli                             ;temp. disable interrupts
        mov     ax,es:[bx]              ;read the old offset
        mov     word ptr old_int_off,ax
        mov     ax,es:[bx+2]            ;read the old segment
        mov     word ptr old_int_seg,ax
        mov     word ptr es:[bx],offset color_int ;load new offset.
        mov     ax,cs
        mov     es:[bx+2],ax            ;load new int segment
        sti                             ;re-enable interrupts
        pop     es
        mov     byte ptr int_done,0     ;clear interrupt flag
        or      byte ptr gbmod,40h      ;enable graphics interrupt
        call    mode
ac1:    test    byte ptr int_done,0ffh  ;has interrupt routine run?
        jz      ac1
        push    es                      ;restore interrupt vectors
        xor     ax,ax
        mov     es,ax
        mov     bx,word ptr g_int_vec   ;fetch graphics vector offset
        cli
        mov     ax,word ptr old_int_off ;restore old interrupt vector
        mov     es:[bx],ax
        mov     ax,word ptr old_int_seg
        mov     es:[bx+2],ax
        sti
        pop     es
        cld                             ;make lods inc si
        ret
color_int:
        push    es
        push    ds
        push    si
        push    cx
        push    ax
        mov     ax,word ptr cs:segment_save ;can't depend on es or ds
        mov     ds,ax                       ;reload segment registers
        mov     es,ax
        cld
        and     byte ptr gbmod,0bfh     ;disable graphics interrupts
        call    mode
        mov     si,offset clmpda        ;fetch color source
        mov     al,0dfh                 ;get the color map's attention
        out     053h,al
        mov     cx,32           ;32 color map entries
ci1:    lodsb                   ;fetch current color map data
        out     051h,al         ;load color map
        loop    ci1             ;loop until all color map data loaded
        mov     byte ptr int_done,0ffh  ;set "interrupt done" flag
        pop     ax
        pop     cx
        pop     si
        pop     ds
        pop     es
        iret
;
;*******************************************************************
;                                                                  *
;       s u b r o u t i n e   c x y 2 c p                          *
;                                                                  *
;       CXY2CP takes the xinit and yinit numbers, converts them to  *
;       an absolute memory location and puts that location into    *
;       curl0,1,2.  yinit is multiplied by the number of words per  *
;       line.  The lower 4 bits of xinit are shifted to the left    *
;       four places and put into curl2. xinit is shifted right four *
;       places to get rid of pixel information and then added to    *
;       yinit times words per line.  This result becomes curl0,     *
;       curl1.                                                     *
;                                                                  *
;       entry:          xinit = x pixel location                   *
;                       yinit = y pixel location                   *
;       exit:           curl0,1,2                                  *
;       register usage: ax,bx,cx,dx                                *
```

```
;*********************************************************************
;
cxy2cp: mov     cl,byte ptr shifts_per_line
        mov     ax,yinit        ;compute yinit times words/line
        shl     ax,cl           ;ax has yinit times words/line
        mov     bx,xinit        ;calculate the pixel address
        mov     dx,bx           ;save a copy of xinit
        mov     cl,4            ;shift xinit 4 places to the left
        shl     bl,cl           ;bl has pixel within word address
        mov     curl2,bl        ;pixel within word address
        mov     cl,4            ;shift xinit 4 places to right
        shr     dx,cl           ;to get xinit words
        add     ax,dx
        mov     word ptr curl0,ax   ;word address
        ret
;*********************************************************************
;                                                                   *
;       s u b r o u t i n e    g d c _ n o t _ b u s y              *
;                                                                   *
;       gdc_not_busy will put a harmless command into the GDC and   *
;       wait for the command to be read out of the command FIFO.    *
;       This means that the GDC is not busy doing a write or read   *
;       operation.                                                  *
;                                                                   *
;       entry:          none                                        *
;       exit:           none                                        *
;       register usage: ax                                          *
;*********************************************************************
;
gdc_not_busy:
        push    cx              ;use cx as a time-out loop counter
        in      al,056h         ;first check if the FIFO is full
        test    al,2
        jz      gnb2            ;jump if not
        mov     cx,8000h        ;wait for FIFO not full or reasonable
gnb0:   in      al,056h         ;time, whichever happens first
        test    al,2            ;has a slot opened up yet?
        jz      gnb2            ;jump if yes
        loop    gnb0            ;if loop count exceeded, go on anyway
gnb2:   mov     al,0dh          ;issue a screen-on command to GDC
        out     057h,al
        in      al,056h         ;did that last command fill it?
        test    al,2
        jz      gnb4            ;jump if not
        mov     cx,8000h
gnb3:   in      al,056h         ;read status register
        test    al,2            ;test FIFO full bit
        jnz     gnb4            ;jump if FIFO not full
        loop    gnb3            ;loop until FIFO not full or give up
gnb4:   mov     ax,40dh         ;issue another screen-on,
        out     057h,al         ;wait for FIFO empty
        mov     cx,8000h
gnb5:   in      al,056h         ;read the GDC status
        test    ah,al           ;FIFO empty bit set?
        jnz     gnb6            ;jump if not.
        loop    gnb5
gnb6:   pop     cx
        ret
;*********************************************************************
;                                                                    *
;       s u b r o u t i n e   i m o d e                              *
;                                                                    *
;       issue Mode command with the parameters from register gbmod  *
;                                                                    *
;       entry:          gbmod                                        *
;       exit:           none                                         *
;       register usage: ax                                           *
;*********************************************************************
;
;
imode:  call    gdc_not_busy
        mov     al,0bfh         ;address the mode register through
        out     53h,al          ;the indirect register
        mov     al,gbmod
        out     51h,al          ;load the mode register
        ret
mode:   mov     al,0bfh         ;address the mode register through
        out     53h,al          ;the indirect register
        mov     al,gbmod
        out     51h,al          ;load the mode register
        ret
;*********************************************************************
;                                                                    *
;       s u b r o u t i n e   i n s c r l                            *
;                                                                    *
;       initialize the scroll map                                    *
;                                                                    *
;       entry:          none                                         *
;       exit:           none                                         *
;       register usage: ax,bx,cx,dx,di,si                            *
;*********************************************************************
;
;
inscrl: cld
        mov     cx,256          ;initialize all 256 locations of the
        xor     al,al           ;shadow area to desired values
        mov     di,offset scrltb
insc0:  stosb
        inc     al
        loop    insc0
;
;The graphics interrupt vector is going to be either 22h or A2h
;depending on whether this is a Model 100-A or a Model 100-B with
;relocated vectors.  Read the old vector, save it, and overwrite it
```

```
;with the new vector.  Before we call the interrupt, we need to
;make sure that the GDC is not in the process of writing something
out to the bitmap.
;
ascrol: call    gdc_not_busy          ;check if GDC id busy
        push    es
        xor     ax,ax
        mov     es,ax
        mov     bx,word ptr g_int_vec
        cli                           ;temporarily disable interrupts
        mov     ax,es:[bx]            ;read the old offset
        mov     word ptr old_int_off,ax
        mov     ax,es:[bx+2]          ;read the old segment
        mov     word ptr old_int_seg,ax
        mov     word ptr es:[bx],offset scrol_int ;load new offset
        mov     ax,cs
        mov     es:[bx+2],ax          ;load new interrupt segment
        sti                           ;re-enable interrupts
        pop     es
        mov     byte ptr int_done,0   ;clear interrupt flag
        or      byte ptr gbmod,40h    ;enable graphics interrupt
        call    mode
as1:    test    byte ptr int_done,0ffh ;has interrupt routine run?
        jz      as1
        push    es                    ;restore the interrupt vectors
        xor     ax,ax
        mov     es,ax
        mov     bx,word ptr g_int_vec ;fetch graphics vector offset
        cli
        mov     ax,word ptr old_int_off ;restore old interrupt vector
        mov     es:[bx],ax
        mov     ax,word ptr old_int_seg
        mov     es:[bx+2],ax
        sti
        pop     es
        ret
;
;Scrollmap loading during interrupt routine.
;Fetch the current mode byte and enable scroll map addressing.
;
scrol_int:
        push    es
        push    ds
        push    si
        push    dx
        push    cx
        push    ax
        cld
        mov     ax,word ptr cs:segment_save ;can’t depend on ds
        mov     ds,ax                        ;reload it
        mov     es,ax
        and     byte ptr gbmod,0bfh   ;disable graphics interupts
        mov     al,gbmod        ;prepare to access scroll map
        mov     gtemp1,al       ;first save current gbmod
        and     gbmod,0dfh      ;enable writing to scroll map
        call    mode            ;do it
        mov     al,07fh         ;select scroll map and reset scroll
        out     53h,al          ;map address counter
        mov     dl,51h          ;output port destination.
        xor     dh,dh
        mov     si,offset scrltb  ;first line's high byte address=0
        mov     cx,16             ;256 lines to write to
        test    byte ptr gbmod,1  ;high resolution?
        jnz     ins1              ;jump if yes
        shr     cx,1            ;only 128 lines if medium resolution
ins1:   lodsw                   ;fetch two scrollmap locations
        out     dx,al           ;assert the even byte
        mov     al,ah
        out     dx,al           ;assert the odd byte
        lodsw                   ;fetch two scrollmap locations
        out     dx,al           ;assert the even byte
        mov     al,ah
        out     dx,al           ;assert the odd byte
        lodsw                   ;fetch two scrollmap locations
        out     dx,al           ;assert the even byte
        mov     al,ah
        out     dx,al           ;assert the odd byte
        lodsw                   ;fetch two scrollmap locations
        out     dx,al           ;assert the even byte
        mov     al,ah
        out     dx,al           ;assert the odd byte
        lodsw                   ;fetch two scrollmap locations
        out     dx,al           ;assert the even byte
        mov     al,ah
        out     dx,al           ;assert the odd byte
        lodsw                   ;fetch two scrollmap locations
        out     dx,al           ;assert the even byte
        mov     al,ah
        out     dx,al           ;assert the odd byte
        lodsw                   ;fetch two scrollmap locations
        out     dx,al           ;assert the even byte
        mov     al,ah
        out     dx,al           ;assert the odd byte
        lodsw                   ;fetch two scrollmap locations
        out     dx,al           ;assert the even byte
        mov     al,ah
        out     dx,al           ;assert the odd byte
        loop    ins1
        mov     al,gtemp1       ;restore previous mode register
        mov     gbmod,al
        call    mode
        mov     byte ptr int_done,0ffh  ;set interrupt-done flag
        pop     ax
        pop     cx
        pop     dx
        pop     si
        pop     ds
        pop     es
        iret                    ;return from interrupt
;*********************************************************************
;                                                                   *
;       s u b r o u t i n e    s e t r a m                          *
;                                                                   *
;       set video ram to a value stored in the p table              *
;                                                                   *
;       entry:          16 byte p1 table                            *
;       exit:           none                                        *
;       register usage: ax,bx,cx,dx,di,si                           *
;*********************************************************************
;
;
setram: mov     byte ptr twdir,2  ;set write direction to --->
        call    gdc_not_busy      ;make sure that the GDC isn't busy
        mov     al,0feh           ;select the write buffer
        out     053h,al
        out     051h,al         ;reset the write buffer counter
        mov     si,offset p1    ;initialize si to start of data
        mov     cx,10h          ;load 16 chars into write buffer
setr1:  lodsb                   ;fetch byte to go to write buffer
        out     52h,al
        loop    setr1
        mov     al,0feh         ;select the write buffer
        out     053h,al
        out     051h,al         ;reset the write buffer counter
        mov     al,049h         ;issue GDC cursor location command
        out     57h,al
        mov     al,byte ptr curl0 ;fetch word location low byte
        out     56h,al            ;load parameter
        mov     al,byte ptr curl1 ;fetch word location high byte
        out     56h,al            ;load parameter
        mov     al,4ah            ;set the GDC mask to all F's
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al
        mov     al,04ch           ;issue figs command
        out     57h,al
        mov     al,byte ptr twdir ;direction to write.
        out     56h,al
        mov     al,nmritl         ;number of GDC writes, low byte
        out     56h,al
        mov     al,nmrith         ;number of GDC writes, high byte
        out     56h,al
        mov     al,22h            ;wdat command
        out     57h,al
        mov     al,0ffh       ;p1 and p2 are dummy parameters
        out     56h,al        ;the GDC requires them for internal
        out     56h,al        ;purposes - no effect on the outside
        ret
segment_save    dw      0         ;ds save area for interrupts
gsubs   endp
        cseg    ends
dseg    segment byte    public  'datasg'
extrn   clmpda:byte
public  xmax,ymax,alu,d,d1,d2,dc
public  curl0,curl1,curl2,dir,fg,gbmskl,gbmskh,gbmod,gdcml,gdcmh
public  nmredl,nmredh,nmritl,nmrith,p1,prdata,prmult,scrltb,startl
public  gtemp3,gtemp4,starth,gtemp,gtemp1,gtemp2,twdir,xinit,xfinal
public  yinit,yfinal,ascrol,num_planes,shifts_per_line
public  words_per_line,g_int_vec
;
;variables to be remembered about the graphics board states
;
alu     db      0       ;current ALU state
cchp1   db      0       ;cursor/character
cchp2   db      0       ;    size definition
cchp3   db      0       ;        parameter bytes
curl0   db      0       ;cursor          - low byte
curl1   db      0       ;  location      - middle byte
curl2   db      0       ;    storage     - high bits & dot address
dc      dw      0       ;figs command dc parameter
d       dw      0       ;figs command d parameter
d2      dw      0       ;figs command d2 parameter
d1      dw      0       ;figs command d1 parameter
dir     db      0       ;figs direction.
fg      db      0       ;current foreground register
gbmskl  db      0       ;graphics board mask register - low byte
gbmskh  db      0       ;                              - high byte
gbmod   db      0       ;graphics board mode register
gdcml   db      0       ;GDC mask register bits - low byte
gdcmh   db      0       ;                        - high byte
g_int_vec       dw      0   ;graphics option's interrupt vector
gtemp   dw      0           ;temporary storage
gtemp1  db      0           ;temporary storage
gtemp2  db      0           ;temporary storage
gtemp3  db      0           ;temporary storage
gtemp4  db      0           ;temporary storage
int_done        db      0   ;interrupt-done state
nmredl  db      0         ;number of read operations - low byte
nmredh  db      0         ;                          - high byte
nmritl  db      0         ;number of GDC writes - low byte
nmrith  db      0         ;                     - high byte
num_planes      db      0 ;number of planes in current resolution
old_int_seg     dw      0 ;old interrupt segment
old_int_off     dw      0 ;old interrupt offset
```

```
p1      db      16 dup (?) ;shadow write buffer & GDC parameters
prdata  db      0         ;pattern register data
prmult  db      0         ;pattern register multiplier factor
scrltb  db      100h dup (?) ;scroll map shadow area
si_temp dw      0
startl  db      0         ;register for start address of display
starth  db      0
twdir   db      0         ;direction for text mode write operation
shifts_per_line db      0 ;shift factor for one line of words
words_per_line  db      0 ;words/scan line for current resolution
xinit   dw      0         ;x initial position
yinit   dw      0         ;y initial position
xfinal  dw      0         ;x final position
yfinal  dw      0         ;y final position
xmax    dw      0
ymax    dw      0
dseg            ends
        end
```

## 5.4 CONTROLLING GRAPHICS OUTPUT

There will be occasions when you will want to control the graphics output to the monitors. The procedure varies according to the monitor configuration. The following two examples illustrate how graphics output can be turned on and off in a single monitor system. The same procedures can be used to turn graphics output on and off in a dual monitor system. However, in a dual monitor configuration, you may want to display graphics output only on the color monitor and continue to display VT102 VSS text output on the monitor. This can be accomplished by loading an 83h into 0Ah instead of an 87h.

### 5.4.1 Example Of Enabling A Single Monitor

```
;*******************************************************************
;                                                                  *
;       p r o c e d u r e   g r a p h i c s _ o n                  *
;                                                                  *
;       purpose:        enable graphics output on single           *
;                       color monitor                              *
;                                                                  *
;       entry:          gbmod contains mode register shadow byte   *
;       exit:           none                                       *
;       register usage: ax                                          *
;*******************************************************************
;
dseg    segment byte    public  'datasg'
extrn   gbmod:byte      ;defined in procedure 'init_option'
dseg    ends
cseg    segment byte    public  'codesg'
extrn   imode:near      ;defined in procedure 'init_option'
        public  graphics_on
        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
;
graphics_on     proc    near
        mov     al,87h
        out     0ah,al          ;enable graphics on monochrome line
        or      byte ptr gbmod,080h  ;enable graphics output in gbmod
        call    imode           ;assert new mode register
        ret                     ;
graphics_on     endp
cseg    ends
        end
```

## 5.4.2 Example Of Disabling A Single Monitor

```
;*******************************************************************
;                                                                   *
;       p r o c e d u r e    g r a p h i c s _ o f f                *
;                                                                   *
;       purpose:        disable graphics output to single           *
;                       (color) monitor                             *
;                                                                   *
;       entry:          gbmod contains mode register shadow byte    *
;       exit:           none                                        *
;       register usage: ax                                          *
;*******************************************************************
;
dseg    segment byte    public  'datasg'
extrn   gbmod:byte      ;defined in procedure 'init_option'
dseg    ends
cseg    segment byte    public  'codesg'
extrn   imode:near      ;defined in procedure 'init_option'
        public  graphics_off
        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
;
graphics_off    proc    near
```

```
        and     byte ptr gbmod,07fh ;disable graphics output in gbmod
        call    imode           ;assert new mode register
        mov     al,83h
        out     0ah,al          ;turn off graphics on monochrome line
        ret
graphics_off    endp
cseg    ends
        end
```

## 5.5 MODIFYING AND LOADING THE COLOR MAP

For an application to modify the Color Map, it must first select the Color Map by way of the Indirect Register (write DFh to port 53h). This will also clear the Color Map Index Counter to zero so loading always starts at the beginning of the map.

Loading the Color Map is done during vertical retrace so there will be no interference with the normal refreshing of the bitmap. To ensure that there is sufficient time for the load, you want to catch the beginning of a vertical retrace. First, check for vertical retrace going inactive (bit 5 of the GDC Status Register = 0). Then, look for the vertical retrace to start again (bit 5 of the GDC Status Register = 1).

To modify only an entry or two, the use of a color shadow map is suggested. Changes can first be made anywhere in the shadow map and then the entire shadow map can be loaded into the Color Map. The next section is an example of modifying a color shadow map and then loading the data from the shadow map into the Color Map.

### 5.5.1 Example Of Modifying And Loading Color Data In A Shadow Map

```
;*************************************************************************
;*                                                                       *
;*              p r o c e d u r e   c h a n g e   c o l o r m a p        *
;*                                                                       *
;*  purpose: change a color in the colormap.                             *
;*  entry:   ax = new color                                              *
;*                al = high nibble = red data
;*                     low nibble  = green data
;*                ah = high nibble = grey data
;*                     low nibble  = blue data
;*           bx = palette entry number                                   *
;*                                                                       *
;*  exit:                                                              *
;*                                                                     *
;***********************************************************************
extrn   fifo__empty:near
cseg    segment byte    public  'codesg'
        public  change__colormap
        public  load__colormap
        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing

change__colormap proc    near
        mov     si,offset clmpda        ;colormap shadow.
        mov     [si+bx],al              ;store the red and green data.
        add     bx,16
        mov     [si+bx],ah              ;store the grey and blue data.
        jmp     load__colormap           ;asssert the new colors.
change__colormap endp




;***********************************************************************
;*                                                                     *
;*              p r o c e d u r e   l o a d  c o l o r m a p           *
;*                                                                     *
;*  purpose: move the data currently in clmpda into the graphics       *
;*              option's colormap.                                     *
;*  entry:   si points to a list of 32 bytes to be loaded into the     *
;*           graphics option colormap.                                 *
;*  exit:                                                              *
;*                                                                     *
;*                                                                     *
;***********************************************************************


load__colormap   proc    near

        mov     si,offset clmpda        ;assume clmpda contains color map
;wait for a vertical retrace to start. because of the way the hardware is
;constructed it is best if we load the colormap during a time when the gdc is
;not trying to apply addresses to it from the bitmap. we could have set up
;an interrupt but this is an easier way of doing things and, under the
;curcumstances, good enough. we want to make sure that we catch the beginning
;of a vertical retrace so first we check for vertical retrace inactive and
;then look for the retrace to start.

        mov     bl,20h          ;wait for no retrace.
here1:  in      al,56h          ;read gdc status register
        test    al,bl           ;verticle sync active?
        jnz     here1           ;keep jumping until it isn't.

here2:  in      al,56h          ;now wait vert retrace to start.
```

```
        test    al,bl           ;keep looping until vert sync goes active.
        jz      here2

;3)enable colormap writes by enabling it through an access to the indirect
;register select port 53h.

        mov     al,0dfh         ;get the color map’s attention
        out     53h,al

;4)now the 16 words composing the entire colormap will be transfered from
;the 32 byte table that si is pointing to. the 16 words are transfered as
;32 bytes, first the 16 bytes containing the red and green information and
;then the 16 bytes containing the grey and blue data.

        cld                     ;make sure that the lods increments si.
        mov     dx,51h
        mov     cx,32           ;32 color map entries
here3:  lodsb                   ;fetch current color map data
        out     dx,al           ;load color map
        loop    here3           ;loop if not all 32 color map datas loaded
        call    fifo__empty      ;gdc status check, see example 03
        ret
load__colormap   endp
cseg    ends
dseg    segment byte    public  ’datasg’
public  clmpda


;colormaps:
;---------
;in general, colormap format is 16 bytes of red and green data,then
;16 bytes of grey and blue data. 0 specifies full intensity, while 0fh
;specifies zero intensity. an possible color map for a 100b, monochrome
;monitor only system in medium resolution (16 colors) would look as follows:

;clmpda                 db      0ffh    ; no red or green data
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                                       ;grey data, no blue data
;                       db      0ffh    ;black
;                       db      00fh    ;white
;                       db      01fh    ;light grey
;                       db      02fh    ;v
;                       db      03fh    ;v
;                       db      04fh    ;v
;                       db      05fh    ;v
;                       db      06fh    ;v
;                       db      07fh    ;medium gray
;                       db      08fh    ;v
;                       db      09fh    ;v
;                       db      0afh    ;v
;                       db      0bfh    ;v
;                       db      0cfh    ;v
;                       db      0dfh    ;v
;                       db      0efh    ;dark grey
;
;on a 100a, only the lower two bits of the monochrome nibble are
;significant, giving only four shades of grey,as opposed to 16 shade on
;the 100b. a sample map for the 100a, monochrome only system, medium
;or high resolution, would look as follows:
;
;clmpda                 db      0ffh    ;no red or green info
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                                       ;grey info, no blue
;                       db      0ffh    ;black
;                       db      0cfh    ;white
;                       db      0dfh    ;light grey
;                       db      0efh    ;dark grey
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;
;a hires color map for a 100b would consist of 4 colors defined that
;utilize all 4 bits of the grey nibble and  would look like this:

;clmpda                 db      0ffh    ;no red or green data
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                       db      0ffh
;                                       ;grey info, no blue info
;                       db      0ffh    ;black
;                       db      00fh    ;white
;                       db      06fh    ;light grey
;                       db      0afh    ;dark grey
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;                       db      0ffh    ;black
;
;in a dual monitor configuration, medium resolution mode, on a 100b,
;there are 4 bits each of red,green,blue and grey. an example colormap would
;be as follows:

;clmpda                 db      0ffh    ;black  -red data,green data
;                       db      000h    ;white
;                       db      0f0h    ;cyan
;                       db      00fh    ;magenta
;                       db      000h    ;
;                       db      00fh    ;red
;                       db      0ffh    ;blue
;                       db      0f0h    ;green
;                       db      0aah    ;dk grey
;                       db      0f8h    ;dk cyan
;                       db      08fh    ;dk magenta
;                       db      088h    ;
;                       db      08fh    ;red
;                       db      0ffh    ;blue
;                       db      0f8h    ;green
;                       db      077h    ;dk grey
;
;                       db      0ffh    ;black,black -grey data,blue data
;                       db      000h    ;white,white
;                       db      010h    ;lightgrey,cyan
;                       db      020h    ;v      ,magenta
;                       db      03fh    ;v
;                       db      04fh    ;v      ,red
;                       db      050h    ;v      ,blue
;                       db      06fh    ;v      ,green
;                       db      07ah    ;medgrey,dk grey
;                       db      0f8h    ;v      ,dk cyan
;                       db      098h    ;       ,dk magenta
;                       db      0afh    ;v
;                       db      0bfh    ;       ,dk red
;                       db      0c8h    ;       ,dk blue
;                       db      0dfh    ;v      ,dk green
;                       db      0e7h    ;dkgrey ,grey
;
;on a 100a, dual monitor configuration, in medium resolution mode, there
;are 4 bits each of red, green, and blue data, all 16 colors, but only 2
;bits of grey data, allowing for only 4 shades grey.

;on a 100a, in high resolution, dual monitor configuration, there are 4
;displayable colors and 2 levels of grey.

;on a 100b, in high resolution, dual monitor configuration, there are 4
;displayable colors and 4 levels of grey.

;in the case of a color monitor only system, the green data must be mapped
;to the monochrome output. for a single color monitor system, medium resolution,
;on a 100b, a sample color map would be as follows:

clmpda                  db      0ffh    ;black  -red data,green mapped to grey
                        db      00fh    ;white
                        db      0ffh    ;cyan
                        db      00fh    ;magenta
                        db      00fh    ;
                        db      00fh    ;red
                        db      0ffh    ;blue
                        db      0ffh    ;green
                        db      0afh    ;gray
                        db      0ffh    ;dk cyan
                        db      08fh    ;dk magenta
                        db      08fh    ;
                        db      08fh    ;dk red
                        db      0ffh    ;dk blue
                        db      0ffh    ;dl green
                        db      07fh    ;gray
;
                        db      0ffh    ;black -green data,blue data
                        db      000h    ;white
                        db      000h    ;cyan
                        db      0f0h    ;magenta
                        db      00fh    ;
                        db      0ffh    ;red
                        db      0f0h    ;blue
                        db      00fh    ;green
                        db      0aah    ;gray
                        db      088h    ;dk cyan
                        db      0f8h    ;dk magenta
                        db      08fh    ;
                        db      0ffh    ;dk red
                        db      0f8h    ;dk blue
                        db      08fh    ;dk green
                        db      077h    ;gray

;as with the previous examples, the same differences apply to high
;resolution (only four colors are displayable) and on the 100a, only
;the lower two bits on the grey nibble are significant (giving only
;four shades of green, since the green data must be output through the
;monochrome line, in either high or medium resolution.

dseg    ends
        end
```

## 5.5.2 Color Map Data

Information in the Color Map is stored as 16 bytes of red and green data followed by 16 bytes of monochrome and blue data. For each color entry, a 0 specifies full intensity and 0fh specifies zero intensity. A sample set of color map entries for a Model 100-B system with a monochrome monitor in medium resolution (16 shades) would look as follows in the shadow area labelled CLMPDA:

```
clmpda                                  ; no red or green data
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
;                                        ;monochrome data, no blue data
                        db    0ffh       ;black
                        db    00fh       ;white
                        db    01fh       ;      .
                        db    02fh       ;      .
                        db    03fh       ;light monochrome
                        db    04fh       ;      .
                        db    05fh       ;      .
                        db    06fh       ;      .
                        db    07fh       ;medium monochrome
                        db    08fh       ;      .
                        db    09fh       ;      .
                        db    0afh       ;      .
                        db    0bfh       ;dark monochrome
                        db    0cfh       ;      .
                        db    0dfh       ;      .
                        db    0efh       ;      .
```

On a Model 100−A system, only the lower two bits of the monochrome nibble are significant. This allows only four monochrome shades as opposed to 16 shades on the Model 100−B system in medium resolution mode. The following sample set of data applies to both the Model 100−A monochrome−only system in either medium or high resolution mode, as well as the Model 100−B monochrome−only system in high resolution mode.

```
clmpda                                   ;no red or green data
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
                        db    0ffh
;                                       ;monochrome data, no blue data
                        db    0ffh      ;black
                        db    00fh      ;white
                        db    05fh      ;light monochrome
                        db    0afh      ;dark monochrome
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
                        db    0ffh      ;black
```

In a dual monitor configuration, with a Model 100-B system in medium resolution mode, all four components of each color entry are present: red, green, blue and monochrome. A sample set of color data would be as follows:

```
clmpda                                  ;red and green data
                        db    offh      ;black
                        db    000h      ;white
                        db    0f0h      ;cyan
                        db    00fh      ;magenta
                        db    000h      ;yellow
                        db    00fh      ;red
                        db    0ffh      ;blue
                        db    0f0h      ;green
                        db    0aah      ;dk gray
                        db    0f8h      ;dk cyan
                        db    08fh      ;dk magenta
                        db    088h      ;dk yellow
                        db    08fh      ;dk red
                        db    0ffh      ;dk blue
                        db    0f8h      ;dk green
                        db    077h      ;gray
;                                       ;monochrome and blue data
                        db    0ffh      ;black        black
                        db    000h      ;white        white
                        db    010h      ;   .         cyan
```

```
                       db    020h      ;   .         magenta
                       db    03fh      ;light mono.  yellow
                       db    04fh      ;   .         red
                       db    050h      ;   .         blue
                       db    06fh      ;   .         green
                       db    07ah      ;med. mono.   dk gray
                       db    0f8h      ;   .         dk cyan
                       db    098h      ;   .         dk magenta
                       db    0afh      ;   .         dk yellow
                       db    0bfh      ;dark mono.   dk red
                       db    0c8h      ;   .         dk blue
                       db    0dfh      ;   .         dk green
                       db    0e7h      ;   .         gray
```

On a Model 100-A dual monitor configuration, in medium resolution mode, all 16 color entries are displayable. However, only two bits of monochrome data are available allowing for only 4 monochrome shades.

On a Model 100-A dual monitor configuration, in high resolution mode, there are four displayable colors and again, four monochrome shades.

On a Model 100-B dual monitor configuration, in high resolution mode, there also are four displayable colors and four monochrome shades.

In a color monitor only system, the green data must be mapped to the monochrome output. For a Model 100-B single color monitor system, in medium resolution mode, a sample color map would be as follows:

```
clmpda                                 ;red data, green data mapped to mono.
                       db    0ffh      ;black
                       db    00fh      ;white
                       db    0ffh      ;cyan
                       db    00fh      ;magenta
                       db    00fh      ;yellow
                       db    00fh      ;red
                       db    0ffh      ;blue
                       db    0ffh      ;green
                       db    0afh      ;dk gray
                       db    0ffh      ;dk cyan
                       db    08fh      ;dk magenta
                       db    08fh      ;dk yellow
                       db    08fh      ;dk red
                       db    0ffh      ;dk blue
                       db    0ffh      ;dk green
                       db    07fh      ;gray
;                                      ;green data, blue data
                       db    0ffh      ;black
                       db    000h      ;white
                       db    000h      ;cyan
```

```
                        db    0f0h      ;magenta
                        db    00fh      ;yellow
                        db    0ffh      ;red
                        db    0f0h      ;blue
                        db    00fh      ;green
                        db    0aah      ;dk gray
                        db    088h      ;dk cyan
                        df    0f8h      ;dk magenta
                        db    08fh      ;dk yellow
                        db    0ffh      ;dk red
                        db    0f8h      ;dk blue
                        db    08fh      ;dk green
                        db    077h      ;gray
```

For a Model 100−A single color monitor system, in either high or medium resolution mode, only the lower two bits of the monochrome output are significant. Therefore, you can only display four intensities of green since the green data must be output through the monochrome line. The same applies to a Model 100−B single color monitor system in high resolution mode.

# CHAPTER 6

# BITMAP WRITE SETUP (GENERAL)

## 6.1 LOADING THE ALU/PS REGISTER

The ALU/PS Register data determines which bitmap planes will be written to during a Read/Modify/Write (RMW) cycle and also sets the operation of the logic unit to one of three write modes.

Assemble a byte where bits 0 through 3 enable or disable the appropriate planes and bits 4 and 5 set the writing mode to REPLACE, COMPLEMENT, or OVERLAY. Bits 6 and 7 are not used. Bit definitions for the ALU/PS Register can be found in Part III of this manual.

Write an EFh to port 53h to select the ALU/PS Register and write the data to port 51h.

### 6.1.1 Example Of Loading The ALU/PS Register

```
;*****************************************************************************
;                                                                            *
;        p r o c e d u r e    a l u p s                                      *
;                                                                            *
;        purpose:        Set the ALU / Plane Select Register                 *
;                                                                            *
;        entry:          bl = value to load into ALU/PS Register             *
;                                                                            *
;                                                                            *
;*****************************************************************************
;
cseg     segment byte    public  'codesg'
         extrn   fifo_empty:near
         public  alups
         assume  cs:cseg,ds:nothing,es:nothing,ss:nothing
alups    proc    near
```

```
         call    fifo_empty
         mov     al,0efh         ;select the ALU/PS Register
         out     53h,al
         mov     al,bl           ;move ALU/PS value to al
         out     51h,al          ;load value into ALU/PS Register
         ret
alups    endp
cseg     ends
         end
```

## 6.2 LOADING THE FOREGROUND/BACKGROUND REGISTER

The data byte in the Foreground/Background Register determines whether bits are set or cleared in each of the bitmap planes during a bitmap write (RMW) operation. Bit definitions for the Foreground/Background Register can be found in Part III of this manual.

Write an F7h to port 53h to select the Foreground/Background Register and write the data byte to port 51h.

### 6.2.1 Example Of Loading The Foreground/Background Register

```
;******************************************************************************
;                                                                             *
;        p r o c e d u r e    f g b g                                         *
;                                                                             *
;        purpose:        Load the Foreground/Background Register              *
;                                                                             *
;        entry:          bl = value to load into the FgBg register            *
;                                                                             *
;                                                                             *
;******************************************************************************
cseg     segment byte    public  'codesg'
         extrn   fifo_empty:near
         public  fgbg
         assume  cs:cseg,ds:nothing,es:nothing,ss:nothing
fgbg     proc    near
         call    fifo_empty
         mov     al,0f7h         ;select the Foreground/Background Register
         out     53h,al
         mov     al,bl
         out     51h,al          ;load the Foreground/Background Register
         ret
```

```
fgbg     endp
cseg     ends
         end
```

# CHAPTER 7

# AREA WRITE OPERATIONS

This chapter contains examples that illustrate displaying a 64K chunk of memory, and clearing a rectangular area of the screen to a given color.

## 7.1 DISPLAY DATA FROM MEMORY

In the following example, video data in a 64K byte area of memory is loaded into the bitmap in order to display it on the monitor. The last byte of the memory area specifies the resolution to be used. A value of zero means use medium resolution mode. A value other than zero means use high resolution mode. In medium resolution mode, the 64K bytes are written to four planes in the bitmap; in high resolution mode, the 64K bytes are written to two planes.

### 7.1.1 Example Of Displaying Data From Memory

```
        title   write entire video screen

        subttl  ritvid.asm
        page 60,132

;********************************************************************************
;                                                                               *
;                                                                               *
;               p r o c e e d u r e     r i t v i d                             *
;                                                                               *
;                                                                               *
;this proceedure will take the contents of the 64k buffer vidsg and insert      *
;that data into the graphics option.                                            *
;                                                                               *
;                                                                               *
;                                                                               *
;                                                                               *
;********************************************************************************
;


extrn   vidseg:near     ;dummy declaration- vidsg is undefined!!!

extrn   nmritl:word,gbmod:byte,gtemp:word,num__planes:byte,curl0:byte
extrn   ginit:near,ifgbg:near,gdc__not__busy:near,ialups:near

        dseg    segment byte    public 'datasg'

;       define the graphics commands
;
curs    equ     49h     ;cursor display position specify command
figs    equ     4ch
gmask   equ     4ah     ;sets which of the 16 bits/word affected
wdat    equ     20h     ;read modify write operation replacing screen data
s__off   equ     0ch     ;blank the display command
s__on    equ     0dh     ;turn display on command
;
;       define the graphics board port addresses
;
graf    equ     50h     ;graphics board base address port 0
gindo   equ     51h     ;graphics board indirect port enable out address
chram   equ     52h     ;character ram
gindl   equ     53h     ;graphics board indirect port in load address
cmaskh  equ     55h     ;character mask high
cmaskl  equ     54h     ;character mask low
gstat   equ     56h     ;gdc status reg (read only)
gpar    equ     56h     ;gdc command parameters (write only)
gread   equ     57h     ;gdc data read from vid mem (read only)
gcmd    equ     57h     ;gdc command port (write only)

;define the indirect register select enables

clrcnt  equ     0feh    ;clear character ram counter
patmlt  equ     0fdh    ;pattern multiplier register
patreg  equ     0fbh    ;pattern data register
fgbg    equ     0f7h    ;foreground/background enable
alups   equ     0efh    ;alu function plane select register
colmap  equ     0dfh    ;color map
modreg  equ     0bfh    ;mode register
scrlmp  equ     07fh    ;scroll map register

dseg ends

        assume  cs:cseg,ds:dseg,es:dseg,es:nothing

        cseg    segment byte    public 'codesg'
```

```
public  ritvid

ritvid  proc    near

;the video data is in vidseg. the last byte in vidseg is the resolution flag.
;if flag is=0 then mid res else is high res. init the option to that resolution.

        mov     ax,vidseg       ;setup es to point at the video buffer.
        mov     es,ax
        mov     si,0ffffh ;fetch the hires/lowres flag from vidbuf last byte.
        mov     al,es:[si]
        test    al,0ffh         ;high res?
        jnz     rt1             ;jump if yes.
        mov     dx,1
        jmp     rt2
rt1:    mov     dx,2
rt2:    call    ginit           ;assert the new resolution.

;init leaves us in text mode with a fg=f0 and a alups=0.

        mov     bl,0fh          ;put all ones into the bg, all 0's into the
        call    ifgbg           ;fg because the char ram inverts incoming data.
        mov     word ptr nmritl,07      ;do eight writes per access.
        test    byte ptr gbmod,1        ;high res?
        jnz     rt3                     ;jump if yes.
        mov     word ptr gtemp,2047     ;8 words writes/plane mid res.
        jmp     rt4
rt3:    mov     word ptr gtemp,4096     ;8 word writes/plane high res.

rt4:    mov     cl,byte ptr num__planes  ;fetch number of planes to be written.
        xor     ch,ch

;enable a plane to be written.

rt5:    push    cx                      ;save plane writing counter.
        mov     bl,byte ptr num__planes  ;select a plane to write enable.
        sub     bl,cl                   ;this is the plane to write enable.
        mov     cl,bl
        mov     bl,0feh                 ;put a 0 in that planes select position.
        ror     bl,cl
        and     bl,0fh                  ;keep in replace mode.
        call    ialups                  ;assert the new alups.

;fill that plane with data 8 words at a time from vidseg.

        mov     word ptr curl0,0        ;start the write at top left corner.
        mov     si,0                    ;start at the beginning of the vidbuf.
        mov     cx,word ptr gtemp       ;number of 8 word writes to fill plane.
rt6:    push    cx                      ;save 8 word write count.
```

```
        call    gdc__not__busy    ;wait until gdc has finished previous write.

        mov     cx,16           ;fetch 16 bytes.
rt7:    mov     al,es:[si]      ;fill ptable with data to be written.
        inc     si
        out     52h,al
        loop    rt7

        mov     al,curs         ;assert the position to start the write.
        out     57h,al
        mov     ax,word ptr curl0
        out     56h,al
        mov     al,ah
        out     56h,al
        mov     al,figs         ;init left gdc mask as ffffh and gbmask as 0.
        out     57h,al          ;all we need is to start the write.
        mov     al,2
        out     56h,al
        mov     al,7
        out     56h,al
        xor     al,al
        out     56h,al
        mov     al,22h
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al
        add     word ptr curl0,08       ;next location to be written.
        pop     cx
        loop    rt6             ;keep looping until this plane all written.

        pop     cx              ;keep looping until all planes written.
        loop    rt5
        ret

ritvid  endp

cseg ends

end
```

## 7.2  SET A RECTANGULAR AREA TO A COLOR

The example that follows illustrates how to set a rectangular area of the screen to some specified color. Input data consists of the coordinates of the upper left and lower right corners of the area (in pixels) plus the color specification (a 4-bit index value). The special case of setting the entire screen to a specified color is included in the example as a subroutine that calls the general routine.

### 7.2.1 Example Of Setting A Rectangular Area To A Color

```
;*************************************************************************
;*                                                                       *
;*      p r o c e d u r e s   t o   w r i t e   a   c o l o r            *
;*                                                                       *
;*      t o   a   r e c t a n g l e   o n   t h e   s c r e e n          *
;*                                                                       *
;*                                                                       *
;*                                                                       *
;*                                                                       *
;*                                                                       *
;*                                                                       *
;*************************************************************************
        public  set_all_screen,set_rectangle
extrn   curl0:word,gbmod:byte,alups:near,xmax:word,ymax:word
extrn   fgbg:near,fifo_empty:near
dseg    segment byte    public  'datasg'
;
;       define the GDC commands
;
curs    equ     49h     ;cursor address specify command
figs    equ     4ch     ;figure specify command.
s_on    equ     0dh     ;bctrl command for screen on.
;
;       define the graphics board port addresses
;
cmaskl  equ     54h     ;write mask low byte
cmaskh  equ     55h     ;write mask high byte
gstat   equ     56h     ;GDC status reg (read only)
gcmd    equ     57h     ;GDC command port (write only)
xstart  dw      0
ystart  dw      0
xstop   dw      0
ystop   dw      0
nmritl   dw      0
dseg            ends
cseg    segment byte    public  'codesg'
        assume  cs:cseg,ds:dseg,es:nothing,ss:nothing
        subttl set all screen

;*************************************************************************
;*                                                                       *
;*              p r o c e d u r e   s e t   a l l   s c r e e n          *
;*                                                                       *
;*  purpose:    set all of the screen to a user defined color.          *
;*  entry:      di is the color to clear the screen to.                 *
;*  exit:                                                               *
;*  registers:                                                          *
;*  stack usage:                                                        *
;*                                                                      *
;*                                                                      *
;*                                                                      *
;************************************************************************
set_all_screen  proc    near
;
;load ax and bx with 0. ax and bx will be used as the upper left corner
; of the rectangle to be written. load cx and dx with the maximum x and
;y of the screen. cx and dx are used to define the bottom right corner
;of the screen.
;
        mov     ax,0                    ;start at the top left corner.
        mov     bx,0
        mov     cx,word ptr xmax        ;fetch the bottom right corner
        mov     dx,word ptr ymax        ;coordinates.
        jmp     set_rectangle           ;lower right max setup by init.
set_all_screen  endp


        subttl set a rectangle to one color


;************************************************************************
;*                                                                      *
;*              p r o c e d u r e   s e t   r e c t a n g l e           *
;*                                                                      *
;*  purpose:    set a user defined screen rectangle to a user           *
;*              defined color.                                          *
;*  entry:      ax has the start x in pixels                            *
;*              bx has the start y in scan lines                        *
;*              cx has the stop x in pixels                             *
;*              dx has the stop y in scan lines                         *
;*              di is the color to clear the screen to.                 *
;*  exit:                                                               *
;*  registers:                                                          *
;*  stack usage:                                                        *
;*                                                                      *
;*                                                                      *
;*                                                                      *
;************************************************************************
set_rectangle           proc    near
;
;save the start/stop coordinates; then, check to see if the option is
;currently occupied before making any changes to its current state.
;this example is not checking for valid entry values.  ax must be less
;than cx.  bx must be less than dx.
```

```
;
        mov     word ptr xstart,ax
        mov     word ptr ystart,bx
        mov     word ptr xstop,cx
        mov     word ptr ystop,dx
        call    fifo_empty    ;wait for an unoccupied graphics option.
;
;assert the new screen color to both sides of the foreground/background
;register.  put the option into replace mode with all planes enabled.
;put the option into write-enabled word mode.
;
        mov     bx,di           ;di passes the color. only lowest nibble valid.
        mov     bh,bl           ;combine the color number into both fg and bg.
        mov     cl,4            ;shift the color up to the upper nibble.
        shl     bh,cl
        or      bl,bh           ;combine the upper nibble with old lower.
        call    fgbg            ;issue to fgbg register.
        xor     bl,bl                   ;assert replace mode, all planes.
        call    alups
        and     byte ptr gbmod,0fdh     ;put into word mode.
        or      byte ptr gbmod,10h      ;put into write-enable mode.
        mov     al,0bfh
        out     53h,al
        mov     al,byte ptr gbmod
        out     51h,al
;
;do the rectangle write.
;
;write one column at a time.  since the GDC is a word device, we have to
;take into account that our write window may start on an odd pixel not
;necessarily on a word boundary.  the graphics options's write mask must be
;set accordingly.
;
;do a write buffer write to the entire rectangle defined by the start/stop
;values.  calculate the first curl0.  calculate the number of scans per
;column to be written.
;
        mov     ax,word ptr xstart      ;turn pixel address into word address.
        mov     cl,4
        shr     ax,cl
        mov     dx,word ptr ystart      ;turn scan start into words per line*y.
        test    byte ptr gbmod,1        ;high resolution?
        jnz     set1                    ;jump if yes.
        mov     cl,5                    ;medium resolution = 32 words per line.
        jmp     set2
set1:   mov     cl,6                    ;high resolution = 64 words per line.
set2:   shl     dx,cl
        add     dx,ax                   ;combine x and y word addresses.
        mov     word ptr curl0,dx       ;first curl0.
        mov     ax,word ptr ystop       ;sub start from stop.
        sub     ax,word ptr ystart
        mov     word ptr nmritl,ax
;
;program the text mask.
;
;there are four possible write conditions:
;
;a)partially write disabled to theleft
;b)completely write enabled
;c)partially write disabled to the right
;d)partially write disabled to both left and right
;
;the portion to be write disabled to the left will be the current xstart
;pixel information.  as we write a column, we update the current xstart
;location.  only the first xstart will have a left hand portion write
;disabled.  only the last will have a right hand portion disabled.  if the
;first is also the last, a portion of both sides will be disabled.
;
cls1:   mov     bx,0ffffh       ;calculate the current write mask.
        mov     cx,word ptr xstart
        and     cx,0fh          ;eliminate all but pixel information.
        shr     bx,cl           ;shift in a 0 for each left  pixel to disable.
;
;write buffer write is done by columns.  take the current xstart and use it
;as the column to be written to.  when the word address of xstart is greater
;than the word address xstop, we are finished.  there is a case where the
;current word address of xstop is equal to the current word address of xstart.
;in that case, we have to be concerned about write disabling the bits to the
;right.  when xstop becomes less than xstart, we are done.
;
        mov     ax,word ptr xstart      ;test to see if word xstop is equal
        and     ax,0fff0h               ;to word xstart.
        mov     cx,word ptr xstop
        and     cx,0fff0h
        cmp     ax,cx                   ;below?
        jb      cls3                    ;jump if yes.
        je      cls2                    ;jump if equal. do last write.
        jmp     exit                    ;all done. exit.
;
;we need to set up the right hand write disable. this is also the last write.
;bx has the left hand write enable mask in it. preserve and combine with the
;right hand mask which will be (f-stop pixel address) bits on the right.
;
cls2:   mov     cx,word ptr xstop       ;strip pixel info out of xstop.
        and     cx,0fh
        inc     cx                      ;make endpoint inclusive of write.
        mov     ax,0ffffh               ;shift the disable mask.
        shr     ax,cl                   ;wherever there is a one, we want to
        xor     ax,0ffffh               ;enable writes.
        and     bx,ax                   ;combine right and left masks.
;
;bx currently has the mask bytes in it. where we have a one we want to make a
```

```
;zero so that that particular bit will be write enabled.
;
cls3:   xor     bx,0ffffh       ;invert so where there is a 1 we write disable.
;
;assert the new text mask. make sure that the GDC is not busy before we change
;the mask.
;
cls4:   call    fifo_empty              ;make sure that the GDC isn't busy.
        mov     al,bh                   ;assert the upper write mask.
        out     cmaskh,al
        mov     al,bl                   ;assert the lower write mask.
        out     cmaskl,al
;
;position the GDC at the top of the column to be written.  this address was
;calculated earlier and the word need only be fetched and applied. the number
;of scans to be written has already been calculated.
;
        mov     al,curs                 ;assert the GDC cursor address.
        out     57h,al
        mov     ax,word ptr curl0       ;assert the word address low byte.
        out     56h,al
        mov     al,dh                   ;assert the word address high byte.
        out     56h,al
;
;start the write operation. write mask, alups, gbmod and fgbg are set up.
;GDC is positioned.
;
        mov     al,figs         ;assert figs to GDC.
        out     57h,al
        xor     al,al           ;direction is down.
        out     56h,al
        mov     ax,word ptr nmritl
        out     56h,al          ;assert number of write operations to perform.
        mov     al,ah
        out     56h,al
        mov     al,22h          ;assert write data command.
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al
;
;update the xstart coordinate for the start of the next column write.
;strip off the pixel information and then add 16 pixels to it to get the next
;word address.
;
        and     word ptr xstart,0fff0h  ;strip off pixel info.
        add     word ptr xstart,16      ;address the next word.
        inc     word ptr curl0
        jmp     cls1                    ;check for another column to clear.
exit:   ret
set_rectangle   endp
```

```
cseg            ends
                end
```

# CHAPTER 8

# VECTOR WRITE OPERATIONS

The examples in this chapter illustrate some basic vector write operations. They cover setting up the Pattern Generator and drawing a single pixel, a line, and a circle.

## 8.1 SETTING UP THE PATTERN GENERATOR

When operating in vector mode, all incoming data originates from the Pattern Generator. The Pattern Generator is composed of a Pattern Register and a Pattern Multiplier. The Pattern Register supplies the basic bit pattern to be written. The Pattern Multiplier determines how many times each basic bit is sent to the bitmap write circuitry before being recirculated.

NOTE

The Pattern Multiplier must be loaded before loading the Pattern Register.

### 8.1.1 Example Of Loading The Pattern Register

The Pattern Register is an 8-bit register that is loaded with a basic bit pattern. This basic bit pattern, modified by a repeat factor stored in the Pattern Multiplier, is the data sent to the bitmap write circuitry when the option is in vector mode.

```
;*****************************************************************************
;                                                                           *
;       p r o c e d u r e   p a t t e r n _ r e g i s t e r                 *
;                                                                           *
;       purpose:        Load the Pattern Register                           *
;                                                                             *
;        entry:          bl = basic bit pattern data                          *
;                                                                             *
;        caution:        You must load the Pattern Multiplier before          *
;                        loading the Pattern Register                         *
;                                                                             *
;******************************************************************************
;
;
;The following are some register values and the corresponding output patterns
;when the repeat factor is a one:
;
;          Value          Pattern
;          -----          -------
;           0FFh          11111111
;           0AAh          10101010
;           0F0h          11110000
;           0CDh          11001101
;
;The following are the same register values and the corresponding output
;patterns when the repeat factor is a three:
;
;          Value          Pattern
;          -----          -------
;           0FFh          111111111111111111111111
;           0AAh          111000111000111000111000
;           0F0h          111111111111000000000000
;           0CDh          111111000000111111000111
;
cseg     segment byte    public  'codesg'
         extrn   fifo_empty:near
         public  pattern_register
         assume  cs:cseg,ds:nothing,es:nothing,ss:nothing
pattern_register proc    near
         call    fifo_empty
         mov     al,0fbh         ;select the Pattern Register
         out     53h,al
         mov     al,bl           ;set up the pattern data
         out     51h,al          ;load the Pattern Register
         ret
pattern_register endp
cseg     ends
         end
```

## 8.1.2 Example Of Loading The Pattern Multiplier

The Graphics Option expects to find a value in the Pattern Multiplier such that sixteen minus that value is the number of times each basic bit in the Pattern Register is repeated. In the following example, you supply the actual repeat factor and the coding converts it to the correct value for the Graphics Option.

```
;*****************************************************************************
;                                                                            *
;        p r o c e d u r e    p a t t e r n _ m u l t                       *
;                                                                            *
;        purpose:        Load the Pattern Multiplier                         *
;                                                                            *
;        entry:          bl = basic bit pattern repeat factor (1 - 16)       *
;                                                                            *
;        caution:        You must load the Pattern Multiplier before         *
;                        loading the Pattern Register                        *
;                                                                            *
;*****************************************************************************
;
;
cseg     segment byte    public  'codesg'
         extrn   fifo_empty:near
         public  pattern_mult
         assume  cs:cseg,ds:nothing,es:nothing,ss:nothing
pattern_mult     proc    near
         call    fifo_empty
         dec     bl              ;adjust bl to be zero-relative
         not     bl              ;invert it (remember Pattern Register is
                                 ;multiplied by 16 minus multiplier value)
         mov     al,0fdh         ;select the Pattern Multiplier
         out     53h,al
         mov     al,bl           ;load the Pattern Multiplier
         out     51h,al
         ret
pattern_mult     endp
cseg     ends
         end
```

## 8.2 DRAW A PIXEL

The following example draws a single pixel at a location specified by a given set of x and y coordinates. Coordinate position 0,0 is in the upper left corner of the screen. The x and y values are in pixels and are positive and zero-based. Valid values are:

```
x = 0 - 799 for high resolution
    0 - 383 for medium resolution

y = 0 - 239 for high or medium resolution
```

Also, in the following example, it is assumed that the Mode, ALU/PS, and Foreground/Background registers have already been set up for a vector write operation.

## 8.2.1 Example Of Drawing A Single Pixel

```
;*****************************************************************************
;                                                                            *
;       p r o c e d u r e    p i x e l                                       *
;                                                                            *
;       purpose:        Draw a pixel                                         *
;                                                                            *
;       entry:          xinit = x location                                   *
;                       yinit = y location                                   *
;                       valid x values  = 0-799 high resolution              *
;                                       = 0-383 medium resolution            *
;                       valid y values  = 0-239 medium or high resolution    *
;                                                                            *
;*****************************************************************************
;
;Do a vector draw of one pixel at location xinit,yinit.  Assume that the
;Graphics Option is already set up in terms of Mode Register, FG/BG, ALU/PS.
;
dseg    segment byte    public  'datasg'
extrn   gbmod:byte,curl0:byte,curl1:byte,curl2:byte,xinit:word,yinit:word
dseg    ends
cseg    segment byte    public  'codesg'
        public  pixel
        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
pixel   proc    near
;
;Convert the starting x,y coordinate pair into a cursor position word value.
;
        mov     al,gbmod        ;are we in medium resolution mode?
        test    al,01
        jz      pv1             ;jump if yes
        mov     cl,06           ;use 64 words per line as a divisor
        jmp     pv2
pv1:    mov     cl,05           ;use 32 words per line as a divisor
pv2:    xor     dx,dx           ;set up for 32bit/16bit math by clearing
        mov     ax,yinit        ;upper 16 bits
        shl     ax,cl
        mov     bx,ax           ;save lines*words per line
        mov     ax,xinit        ;compute the number of extra words on last line
        mov     cx,16           ;16 bits per word
        div     cx              ;ax now has number of extra words to add in
        add     ax,bx           ;dx has the less than 16 dot address left over
        mov     curl0,al        ;this results in the new cursor memory address
        mov     curl1,ah
        mov     cl,04           ;dot address is high nibble of byte
        shl     dl,cl
        mov     curl2,dl
;
;Position the cursor.
;
        mov     al,49h                  ;send out the cursor command byte.
        out     57h,al
        mov     ax,word ptr curl0       ;assert cursor location low byte.
        out     56h,al
        mov     al,ah                   ;assert cursor location high byte.
        out     56h,al
        mov     al,byte ptr curl2       ;assert cursor pixel location.
        out     56h,al
;
;Assert the figs command to draw one pixel's worth of vector.
;
        mov     al,4ch          ;assert the FIGS command
        out     57h,al
        mov     al,02h          ;line drawn to the right.
        out     56h,al
        mov     al,6ch          ;tell the GDC to draw the pixel when ready.
        out     57h,al
        ret
pixel   endp
cseg    ends
        end
```

## 8.3 DRAW A VECTOR

The example in this section will draw a line between two points specified by x and y coordinates given in pixels. The valid ranges for these coordinates are the same as specified for the previous example. Again it is assumed that the Mode, ALU/PS, and Foreground/Background registers have already been set up for a vector write operation. In addition, the Pattern Generator has been set up for the type of line to be drawn between the two points.

### 8.3.1 Example Of Drawing A Vector

```
;******************************************************************************
;                                                                             *
;        p r o c e d u r e    v e c t o r                                      *
;                                                                              *
;        purpose:        Draw a vector                                         *
;                                                                              *
;        entry:          xinit = starting x location                           *
;                        yinit = starting y location                           *
;                        xfinal= ending x location                             *
;                        yfinal= ending y location                             *
;                        valid x values = 0 - 799 high resolution              *
;                                         0 - 383 medium resolution            *
;                        valid y values = 0 - 239 high or medium resolution   *
;        exit:                                                                 *
;                                                                              *
;*******************************************************************************
;
;Assume start and stop co-ordinates to be in registers and
;all other incidental requirements already taken care of.  This code positions
;the cursor, computes the FIGS parameters DIR, DC, D, D2, and D1, and then
;implements the FIGS and FIGD commands.
;What is not shown here, is that the Mode Register is set up for vector
;operations, the write mode and planes select is set up in the ALU/PS Register,
;the FGBG Register is set up with foreground and background colors, and the
;Pattern Multiplier/Register are loaded. In vector mode all incoming data
;is from the Pattern Register. We have to make sure that the GDC's pram 8 and
;9 are all ones so that it will try to write all ones to the bitmap. The
;external hardware will get in there and put the Pattern Register's data
;into the bitmap.
;
;This same basic setup can be used for area fills, arcs and such.
;
extrn    fifo_empty:near,gbmod:byte,p1:byte
cseg     segment byte    public  'codesg'
         public  vector
         assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
vector   proc    near
         call    fifo_empty
         mov     al,78h
         out     57h,al          ;set pram bytes 8 and 9
         mov     al,0ffh
         out     56h,al
         out     56h,al
;
;Convert the starting x,y coordinate pair into a cursor position word value.
;
         mov     al,gbmod        ;are we in low resolution mode?
         test    al,01
         jz      v11             ;jump if yes
         mov     cl,06           ;use 64 words per line as a divisor
         jmp     v2
v11:     mov     cl,05           ;use 32 words per line as a divisor
v2:      xor     dx,dx           ;set up for 32bit/16bit math by clearing
```

```
         mov     ax,yinit        ;upper 16 bits
         shl     ax,cl
         mov     bx,ax           ;save lines*words per line
         mov     ax,xinit        ;compute the no. of extra words on last line
         mov     cx,16           ;16 bits per word
         div     cx              ;ax now has number of extra words to add in
         add     ax,bx           ;dx has the less than 16 dot address left over
         mov     curl0,al        ;this results in the new cursor memory address
         mov     curl1,ah
         mov     cl,04           ;dot address is high nibble of byte
         shl     dl,cl           ;
         mov     curl2,dl
         mov     al,49h          ;set cursor location to that in curl0,1,2
         out     57h,al          ;issue the GDC cursor location command
         mov     al,curl0        ;fetch word low address
         out     56h,al
         mov     al,curl1        ;word middle address
         out     56h,al
         mov     al,curl2        ;dot address (top 4 bits) and high word addr
         out     56h,al
;
;Draw a vector.
;
        mov     ax,word ptr xinit       ;is this a single point draw?
        cmp     word ptr xfinal,ax      ;if yes then start=stop coordinates.
        jnz     v1                      ;jump if definitely not.
        mov     ax,word ptr yinit       ;maybe. check y coordinates.
        cmp     word ptr yfinal,ax
        jnz     v1                      ;jump if definitely not.
        mov     al,04ch                 ;program a single pixel write
        out     57h,al                  ;operation
        mov     al,2                    ;direction is to the right..
        out     56h,al
        mov     al,06ch
        out     57h,al
        ret
v1:     mov     bx,yfinal       ;compute delta y
        sub     bx,yinit        ;delta y negative now?
        jns     quad34          ;jump if not (must be either quad 3 or 4)
quad12: neg     bx              ;delta y is negative, make absolute
        mov     ax,xfinal       ;compute delta x
        sub     ax,xinit        ;delta x negative?
        js      quad2           ;jump if yes
quad1:  cmp     ax,bx           ;octant 2?
        jbe     oct3            ;jump if not
oct2:   mov     p1,02           ;direction of write
        jmp     vxind   ;abs(deltax)>abs(deltay), independent axis=x-axis
oct3:   mov     p1,03           ;direction of write
        jmp     vyind   ;abs(deltax)=<abs(deltay), independent axis=y-axis
quad2:  neg     ax              ;delta x is negative, make absolute
        cmp     ax,bx           ;octant 4?
        jae     oct5            ;jump if not
oct4:   mov     p1,04           ;direction of write
        jmp     vyind   ;abs(deltax)=<abs(deltay), independent axis=y-axis
oct5:   mov     p1,05           ;direction of write
        jmp     vxind   ;abs(deltax)>abs(deltay), independent axis=x-axis
quad34: mov     ax,xfinal       ;compute delta x
        sub     ax,xinit
        jns     quad4           ;jump if delta x is positive
quad3:  neg     ax              ;make delta x absolute instead of negative
        cmp     ax,bx           ;octant 6?
        jbe     oct7            ;jump if not
oct6:   mov     p1,06           ;direction of write
        jmp     vxind   ;abs(deltax)>abs(deltay), independent axis=x-axis
oct7:   mov     p1,07           ;direction of write
        jmp     vyind   ;abs(deltax)<=abs(deltay), independent axis=y-axis
quad4:  cmp     ax,bx           ;octant 0?
        jae     oct1            ;jump if not
oct0:   mov     p1,0            ;direction of write
        jmp     vyind   ;abs(deltax)<abs(deltay), independent axis=y-axis
oct1:   mov     p1,01           ;direction of write
        jmp     vxind   ;abs(deltax)=>(deltay), independent axis=x-axis
;
vyind:  xchg    ax,bx           ;put independent axis in ax, dependent in bx
vxind:  and     ax,03fffh       ;limit to 14 bits
        mov     dc,ax           ;DC=abs(delta x)-1
        push    bx              ;save abs(delta y)
        shl     bx,01           ;multiply delta y by two
        sub     bx,ax
        and     bx,03fffh       ;limit to 14 bits
        mov     d,bx            ;D=2*abs(delta y)-abs(delta x)
        pop     bx              ;restore (abs(delta y)
        push    bx              ;save abs(delta y)
        sub     bx,ax
        shl     bx,1
        and     bx,03fffh       ;limit to 14 bits
        mov     d2,bx           ;D2=2*(abs(delta y)-abs(delta x))
        pop     bx
        shl     bx,1
        dec     bx
        and     bx,03fffh       ;limit to 14 bits
        mov     d1,bx           ;D1=2*abs(delta y)-1
vdo:    mov     al,04ch         ;issue the FIGS command
        out     57h,al
        mov     al,08           ;construct P1 of FIGS command
        or      al,byte ptr p1
        out     56h,al          ;issue a parameter byte
        mov     si,offset dc
        mov     cx,08           ;issue the 8 bytes of DC,D,D2,D1
vdo1:   mov     al,[si]         ;fetch byte
        out     56h,al          ;issue to the GDC
        inc     si              ;point to next in list
        loop    vdo1            ;loop until all 8 done
        mov     al,06ch         ;start the drawing process in motion
        out     57h,al          ;by issuing FIGD
        ret
vector  endp
cseg    ends
dseg    segment byte    public  'datasg'
        public  curl0,curl1,curl2,dc,d,d2,d1,dm,dir,xinit,yinit
        public  xfinal,yfinal
curl0   db      0
curl1   db      0
curl2   db      0
dc      dw      0
d       dw      0
d2      dw      0
d1      dw      0
dm      dw      0
dir     dw      0
xinit   dw      0
yinit   dw      0
xfinal  dw      0
yfinal  dw      0
dseg    ends
        end
```

## 8.4 DRAW A CIRCLE

The example in this section will draw a circle, given the radius and the coordinates of the center in pixels. The code is valid only if the option is in medium resolution mode. If this code is executed in high resolution mode, the aspect ratio would cause the output to be generated as an ellipse. As in the previous examples, the option is assumed to have been set up for a vector write operation with the appropriate type of line programmed into the Pattern Generator.

### 8.4.1 Example Of Drawing A Circle

```
;******************************************************************************
;                                                                             *
;       p r o c e d u r e   c i r c l e                                       *
;                                                                             *
;       purpose:        Draw a circle in medium resolution mode               *
;                                                                             *
;       entry:          xinit = circle center x coordinate (0-799)            *
;                       yinit = circle center y coordinate (0-239)          *
;                       radius = radius of the circle in pixels             *
;                                                                           *
;       caution:        This routine will only work in medium resolution    *
;                       mode. Due to the aspect ratio of high resolution    *
;                       mode, circles appear as ellipses.                   *
;                                                                           *
;****************************************************************************
;
;Draw an circle.
;
;This code positions the cursor and computes the FIGS parameters DIR, DC,
;D, D2, and D1. It then implements the actual FIGS and FIGD commands.
;What you don't see here is that the Mode Register is set up for vector
;operations, the write mode and planes select are set up in the ALU/PS,
;the FGBG Register is loaded with foreground and background colors and the
;Pattern Multiplier/Register are loaded. In vector mode, all incoming data
;is from the Pattern Register. We have to make sure that the GDC's pram 8 and
;9 are all ones so that it will try to write all ones to the bitmap. The
;external hardware will get in there and put the Pattern Register's data
;into the bitmap.
;
extrn   gbmod:byte,curl0:byte,curl1:byte,curl2:byte,xinit:word,yinit:word
extrn   fifo_empty:near,dc:word,d:word,d2:word,d1:word,dm:word,dir:word
dseg    segment byte    public  'datasg'
        public  radius,xad,yad
xad     dw      0
yad     dw      0
radius  dw      0
dseg    ends
cseg    segment byte    public  'codesg'
        public  circle
        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
circle  proc    near
        call    fifo_empty
        mov     al,78h
        out     57h,al                  ;set pram bytes 8 and 9
        mov     al,0ffh
        out     56h,al
        out     56h,al
        mov     word ptr d1,-1          ;set FIGS D1 parameter
        mov     word ptr dm,0           ;set FIGS D2 parameter
        mov     bx,word ptr radius      ;get radius
        mov     ax,0b505h               ;get 1/1.41
        inc     bx
        mul     bx
        mov     word ptr dc,dx          ;set FIGS DC parameter
        dec     bx
        mov     word ptr d,bx           ;set FIGS D parameter
        shl     bx,1
        mov     word ptr d2,bx          ;set FIGS D2 parameter
        mov     ax,word ptr xinit       ;get center x
        mov     word ptr xad,ax         ;save it
        mov     ax,word ptr yinit       ;get center y
        sub     ax,word ptr radius      ;subtract radius
        mov     word ptr yad,ax         ;save it
        call    acvt                    ;position cursor
        mov     byte ptr dir,01h        ;arc 1
        call    avdo                    ;draw it
        call    acvt                    ;position cursor
        mov     byte ptr dir,06h        ;arc 6
        call    avdo                    ;draw it
        mov     ax,word ptr xinit       ;get center x
        mov     word ptr xad,ax         ;save it
        mov     ax,word ptr yinit       ;get center y
        add     ax,word ptr radius      ;add in radius
        mov     word ptr yad,ax         ;save it
        call    acvt                    ;position cursor
        mov     byte ptr dir,02h        ;arc 2
        call    avdo                    ;draw it
        call    acvt                    ;position cursor
        mov     byte ptr dir,05h        ;arc 5
        call    avdo                    ;draw it
        mov     ax,word ptr xinit       ;get center x
        sub     ax,word ptr radius      ;subtract radius
        mov     word ptr xad,ax         ;save it
        mov     ax,word ptr yinit       ;get center y
        mov     word ptr yad,ax         ;save it
        call    acvt                    ;position cursor
        mov     byte ptr dir,03h        ;arc 3
        call    avdo                    ;draw it
        call    acvt                    ;position cursor
        mov     byte ptr dir,00h        ;arc 0
        call    avdo                    ;draw it
        mov     ax,word ptr xinit       ;get center x
        add     ax,word ptr radius      ;add in the radius
        mov     word ptr xad,ax         ;save it
        mov     ax,word ptr yinit       ;get center y
        mov     word ptr yad, ax        ;save it
        call    acvt                    ;position cursor
        mov     byte ptr dir,07h        ;arc 7
        call    avdo                    ;draw it
        call    acvt                    ;position cursor
        mov     byte ptr dir,04h        ;arc 4
        call    avdo                    ;draw it
        ret
;
;Convert the starting x,y coordinate pair into a cursor position word value.
;
acvt:
        mov     al,gbmod        ;are we in low resolution mode?
        test    al,01
        jz      av1             ;jump if yes
        mov     cl,06           ;use 64 words per line as a divisor
        jmp     av2
av1:    mov     cl,05           ;use 32 words per line as a divisor
av2:    xor     dx,dx           ;set up for 32bit/16bit math by
        mov     ax,word ptr yad    ;clearing upper 16 bits
        shl     ax,cl
        mov     bx,ax           ;save lines*words per line
        mov     ax,word ptr xad    ;compute no. of extra words on last line
        mov     cx,16           ;16 bits per word
        div     cx              ;ax now has number of extra words to add in
        add     ax,bx           ;dx has the less than 16 dot address left over
        mov     curl0,al        ;this results in the new cursor memory address
        mov     curl1,ah
        mov     cl,04           ;dot address is high nibble of byte
        shl     dl,cl           ;
        mov     curl2,dl
        mov     al,49h          ;set cursor location to that in curl0,1,2
        out     57h,al          ;issue the GDC cursor location command
        mov     al,curl0        ;fetch word low address
        out     56h,al
        mov     al,curl1        ;word middle address
        out     56h,al
        mov     al,curl2        ;dot address (top 4 bits) and high word addr
        out     56h,al
        ret
avdo:   call    fifo_empty
        mov     al,4ch          ;issue the FIGS command
        out     57h,al
        mov     al,020h         ;construct P1 of FIGS command
        or      al,byte ptr dir
        out     56h,al          ;issue a parameter byte
        mov     si,offset dc
        mov     cx,10           ;issue the 10 bytes of DC,D,D2,D1
avdo1:  mov     al,[si]         ;fetch byte
        out     56h,al          ;issue to the GDC
        inc     si              ;point to next in list
        loop    avdo1           ;loop until all 10 done
        mov     al,6ch          ;start the drawing process in motion
        out     57h,al          ;by issuing FIGD command
        ret
circle  endp
cseg    ends
        end
```

# CHAPTER 9

# TEXT WRITE OPERATIONS

In this chapter the examples illustrate coding for writing byte-aligned 8 X 10 characters, determining type and position of the cursor, and writing bit-aligned vector (stroked) characters.

## 9.1 WRITE A BYTE-ALIGNED CHARACTER

This example uses a character matrix that is eight pixels wide and ten scan lines high. The characters are written in high resolution mode and are aligned on byte boundaries. The inputs are the column and row numbers that locate the character, the code for the character, and the color attribute.

### 9.1.1 Example Of Writing A Byte-Aligned Character

```
;this is an example of a program to impliment character writing on the
;rainbow graphics option. this particular example show how to write
;with each character being eight pixels wide and ten scan lines high in
;high res mode.

;this module assumes that the graphics option is in high res, that the
;coordinates for the write are to passed as a character coordinate, not
;a pixel coordinate, the x,y coordinate is 0 relative (not starting at
;1,1), the character to be written is in dl and the color to write it
;is in dh.

        title graphics text writing routines
        page    60,132

;*************************************************************************
```

```
;*                                                                    *
;*              p r o c e d u r e   g t e x t                         *
;*                                                                    *
;*  purpose:    write graphics text                                   *
;*  entry:      ax,bx is the location of the character in column, row *
;*              dl is the character, dh is the fgbg                   *
;*  exit:                                                             *
;*  registers:                                                        *
;*  stack usage:                                                      *
;*                                              rick haggard 01/30/84 *
;*                                                                    *
;*********************************************************************

extrn   curl0:byte,curl2:byte,fg:byte*gbmskl:byte,gbmod:byte,
extrn   ifgbg:near,imode:near,stgbm:near

        public  $gtext

dseg    segment byte    public  'datasg'

;       define the gdc commands

curs    equ     49h     ;cursor display position specify command
figs    equ     4ch
gmask   equ     4ah     ;sets which of the 16 bits/word affected
rdat    equ     0a0h    ;read command to gdc.
s__on    equ     0fh     ;turn display on command
;
;       define the graphics board port addresses
;
graf    equ     50h     ;graphics board base address port 0
gindo   equ     51h     ;graphics board indirect port enable out address
chram   equ     52h     ;character ram
gindl   equ     53h     ;graphics board indirect port in load address
cmaskh  equ     55h     ;character mask high
cmaskl  equ     54h     ;character mask low
gstat   equ     56h     ;gdc status reg (read only)
gpar    equ     56h     ;gdc command parameters (write only)
gread   equ     57h     ;gdc data read from vid mem (read only)
gcmd    equ     57h     ;gdc command port (write only)
;
;define the indirect register select enables
;
clrcnt  equ     0feh    ;clear character ram counter
patmlt  equ     0fdh    ;pattern multiplier register
patreg  equ     0fbh    ;pattern data register
fgbg    equ     0f7h    ;foreground/background enable
alups   equ     0efh    ;alu function plane select register
colmap  equ     0dfh    ;color map
modreg  equ     0bfh    ;mode register
scrlmp  equ     07fh    ;scroll map register
```

```
;this table has the addresses of the individual text font characters.
;a particular texttab addresses are found by taking the offset of the textab,
;adding in the ascii offset of the character to be printed and loading the
;resulting word. this word is the address of the start of the character's
;text font.

textab  dw      0
        dw      10
        dw      20
        dw      30
        dw      40
        dw      50
        dw      60
        dw      70
        dw      80
        dw      90
        dw      100
        dw      110
        dw      120
        dw      130
        dw      140
        dw      150
        dw      160
        dw      170
        dw      180
        dw      190
        dw      200
        dw      210
        dw      220
        dw      230
        dw      240
        dw      250
        dw      260
        dw      270
        dw      280
        dw      290
        dw      300
        dw      310
        dw      320
        dw      330
        dw      340
        dw      350
        dw      360
        dw      370
        dw      380
        dw      390
        dw      400
        dw      410
        dw      420
        dw      430
        dw      440
        dw      450
        dw      460
        dw      470
        dw      480
        dw      490
        dw      500
        dw      510
        dw      520
        dw      530
        dw      540
        dw      550
        dw      560
        dw      570
        dw      580
        dw      590
        dw      600
        dw      610
        dw      620
        dw      630
        dw      640
        dw      650
        dw      660
        dw      670
        dw      680
        dw      690
        dw      700
        dw      710
        dw      720
        dw      730
        dw      740
        dw      750
        dw      760
        dw      770
        dw      780
        dw      790
        dw      800
        dw      810
        dw      820
        dw      830
        dw      840
        dw      850
        dw      860
        dw      870
        dw      880
        dw      890
        dw      900
        dw      910
        dw      920
        dw      930
        dw      940
```

```
;text font

space   db      11111111b
        db      0ffh
        db      0ffh
        db      0ffh
        db      0ffh
        db      0ffh
        db      0ffh
        db      0ffh
        db      0ffh
        db      11111111b

exclam  db      11111111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11111111b
        db      11100111b
        db      11111111b
        db      11111111b

quote   db      11111111b
        db      0d7h
        db      0d7h
        db      0d7h
        db      0ffh
        db      0ffh
        db      0ffh
        db      0ffh
        db      0ffh
        db      11111111b

num     db      11111111b
        db      11010111b
        db      11010111b
        db      00000001b
        db      11010111b
        db      00000001b
        db      11010111b
        db      11010111b
        db      11111111b
        db      11111111b

dollar  db      11111111b
        db      11101111b
        db      10000001b
        db      01101111b
        db      10000011b
        db      11101101b
        db      00000011b
        db      11101111b
        db      11111111b
        db      11111111b

percent db      11111111b
        db      00111101b
        db      00111011b
        db      11110111b
        db      11101111b
        db      11011111b
        db      10111001b
        db      01111001b
        db      11111111b
        db      11111111b

amp     db      11111111b
        db      10000111b
        db      01111011b
        db      10110111b
        db      11001111b
        db      10110101b
        db      01111011b
        db      10000100b
        db      11111111b
        db      11111111b

apos    db      11111111b
        db      11100111b
        db      11101111b
        db      11011111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b

lefpar  db      11111111b
        db      11110011b
        db      11100111b
        db      11001111b
        db      11001111b
        db      11001111b
        db      11100111b
        db      11110011b
        db      11111111b
        db      11111111b

ritpar  db      11111111b
```

```
        db      11001111b
        db      11100111b
        db      11110011b
        db      11110011b
        db      11110011b
        db      11100111b
        db      11001111b
        db      11111111b
        db      11111111b

aster   db      11111111b
        db      11111111b
        db      10111011b
        db      11010111b
        db      00000001b
        db      11010111b
        db      10111011b
        db      11111111b
        db      11111111b
        db      11111111b

plus    db      11111111b
        db      11111111b
        db      11101111b
        db      11101111b
        db      00000001b
        db      11101111b
        db      11101111b
        db      11111111b
        db      11111111b
        db      11111111b

comma   db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11100111b
        db      11100111b
        db      11001111b
        db      11111111b

minus   db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      00000001b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b

period  db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11100111b
        db      11100111b
        db      11111111b
        db      11111111b

slash   db      11111111b
        db      11111101b
        db      11111001b
        db      11110011b
        db      11100111b
        db      11001111b
        db      10011111b
        db      00111111b
        db      11111111b
        db      11111111b

zero    db      11111111b
        db      11000101b
        db      10010001b
        db      10010001b
        db      10001001b
        db      10001001b
        db      10011001b
        db      10100011b
        db      11111111b
        db      11111111b

one     db      11111111b
        db      11100111b
        db      11000111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      10000001b
        db      11111111b
        db      11111111b

two     db      11111111b
        db      11000011b
        db      10011001b
        db      11111001b
        db      11100011b
        db      11001111b
        db      10011111b
        db      10000001b
        db      11111111b
        db      11111111b

three   db      11111111b
        db      10000001b
        db      11110011b
        db      11100111b
        db      11000011b
        db      11111001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

four    db      11111111b
        db      11110001b
        db      11100001b
        db      11001001b
        db      10011001b
        db      10000001b
        db      11111001b
        db      11111001b
        db      11111111b
        db      11111111b

five    db      11111111b
        db      10000001b
        db      10011111b
        db      10000011b
        db      11111001b
        db      11111001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

six     db      11111111b
        db      11000011b
        db      10011001b
        db      10011111b
        db      10000011b
        db      10001001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b
```

```
seven   db      11111111b
        db      10000001b
        db      11111001b
        db      11110011b
        db      11100111b
        db      11001111b
        db      10011111b
        db      10011111b
        db      11111111b
        db      11111111b

eight   db      11111111b
        db      11000011b
        db      10011001b
        db      10011001b
        db      11000011b
        db      10011001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

nine    db      11111111b
        db      11000011b
        db      10011001b
        db      10010001b
        db      11000001b
        db      11111001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

colon   db      11111111b
        db      11111111b
        db      11111111b
        db      11100111b
        db      11100111b
        db      11111111b
        db      11100111b
        db      11100111b
        db      11111111b
        db      11111111b

scolon  db      11111111b
        db      11111111b
        db      11111111b
        db      11100111b
        db      11100111b
        db      11111111b
        db      11100111b
        db      11100111b
        db      11001111b
        db      11111111b

lesst   db      11111111b
        db      11111001b
        db      11110011b
        db      11001111b
        db      10011111b
        db      11001111b
        db      11110011b
        db      11111001b
        db      11111111b
        db      11111111b

equal   db      11111111b
        db      11111111b
        db      11111111b
        db      10000001b
        db      11111111b
        db      10000001b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b

greatr  db      11111111b
        db      10011111b
        db      11001111b
        db      11110011b
        db      11111001b
        db      11110011b
        db      11001111b
        db      10011111b
        db      11111111b
        db      11111111b


ques    db      11111111b
        db      11000011b
        db      10011001b
        db      11111001b
        db      11110011b
        db      11100111b
        db      11111111b
        db      11100111b
        db      11111111b
        db      11111111b

at      db      11111111b
        db      11000011b
        db      10011001b
        db      10011001b
        db      10010001b
        db      10010011b
        db      10011111b
        db      11000001b
        db      11111111b
        db      11111111b

capa    db      11111111b
        db      11100111b
        db      11000011b
        db      10011001b
        db      10011001b
        db      10000001b
        db      10011001b
        db      10011001b
        db      11111111b
        db      11111111b

capb    db      11111111b
        db      10000011b
        db      10011001b
        db      10011001b
        db      10000011b
        db      10011001b
        db      10011001b
        db      10000011b
        db      11111111b
        db      11111111b

capc    db      11111111b
        db      11000011b
        db      10011001b
        db      10011111b
        db      10011111b
        db      10011111b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

capd    db      11111111b
        db      10000011b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10000011b
        db      11111111b
        db      11111111b

cape    db      11111111b
        db      10000001b
        db      10011111b
        db      10011111b
        db      10000011b
        db      10011111b
        db      10011111b
        db      10000001b
        db      11111111b
        db      11111111b

capf    db      11111111b
        db      10000001b
        db      10011101b
        db      10011111b
        db      10000111b
        db      10011111b
        db      10011111b
        db      10011111b
        db      11111111b
        db      11111111b

capg    db      11111111b
        db      11000011b
        db      10011001b
        db      10011001b
        db      10011111b
        db      10010001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

caph    db      11111111b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10000001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11111111b
        db      11111111b

capi    db      11111111b
        db      11000011b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11000011b
        db      11111111b
        db      11111111b

capj    db      11111111b
        db      11100001b
        db      11110011b
        db      11110011b
        db      11110011b
        db      11110011b
        db      10010011b
        db      11000111b
        db      11111111b
        db      11111111b

capk    db      11111111b
        db      10011001b
        db      10010011b
        db      10000111b
        db      10001111b
        db      10000111b
        db      10010011b
        db      10011001b
        db      11111111b
        db      11111111b

capl    db      11111111b
        db      10000111b
        db      11001111b
        db      11001111b
        db      11001111b
        db      11001111b
        db      11001101b
        db      10000001b
        db      11111111b
        db      11111111b

capm    db      11111111b
        db      00111001b
        db      00010001b
        db      00101001b
        db      00101001b
        db      00111001b
        db      00111001b
        db      00111001b
        db      11111111b
        db      11111111b

capn    db      11111111b
```

```
        db      10011001b
        db      10001001b
        db      10001001b
        db      10000001b
        db      10010001b
        db      10010001b
        db      10011001b
        db      11111111b
        db      11111111b

capo    db      11111111b
        db      11000011b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

capp    db      11111111b
        db      10000011b
        db      10011001b
        db      10011001b
        db      10000011b
        db      10011111b
        db      10011111b
        db      10011111b
        db      11111111b
        db      11111111b

capq    db      11111111b
        db      11000011b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10010001b
        db      10011001b
        db      11000001b
        db      11111100b
        db      11111111b

capr    db      11111111b
        db      10000011b
        db      10011001b
        db      10011001b
        db      10000011b
        db      10000111b
        db      10010011b
        db      10011001b
        db      11111111b
        db      11111111b

caps    db      11111111b
        db      11000011b
        db      10011001b
        db      10011111b
        db      11000111b
        db      11110001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

capt    db      11111111b
        db      10000001b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11111111b
        db      11111111b

capu    db      11111111b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

capv    db      11111111b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11000011b
        db      11100111b
        db      11111111b
        db      11111111b

capw    db      11111111b
        db      00111001b
        db      00111001b
        db      00111001b
        db      00111001b
        db      00101001b
        db      00000001b
        db      00111001b
        db      11111111b
        db      11111111b

capx    db      11111111b
        db      10011001b
        db      10011001b
        db      11000011b
        db      11100111b
        db      11000011b
        db      10011001b
        db      10011001b
        db      11111111b
        db      11111111b

capy    db      11111111b
        db      10011001b
        db      10011001b
        db      11000011b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11000011b
        db      11111111b
        db      11111111b

capz    db      11111111b
        db      10000001b
        db      11111001b
        db      11110011b
        db      11100111b
        db      11001111b
        db      10011101b
        db      10000001b
        db      11111111b
        db      11111111b

lbrak   db      11111111b
        db      10000011b
        db      10011111b
        db      10011111b
        db      10011111b
        db      10011111b
        db      10011111b
        db      10000011b
        db      11111111b
        db      11111111b
```

```
bslash  db      11111111b
        db      10111111b
        db      10011111b
        db      11001111b
        db      11100111b
        db      11110011b
        db      11111001b
        db      11111101b
        db      11111111b
        db      11111111b

rbrak   db      11111111b
        db      10000011b
        db      11110011b
        db      11110011b
        db      11110011b
        db      11110011b
        db      11110011b
        db      10000011b
        db      11111111b
        db      11111111b

carrot  db      11111111b
        db      11101111b
        db      11010111b
        db      10111011b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b

underl  db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      00000000b

lsquot  db      11111111b
        db      11100111b
        db      11100111b
        db      11110111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b

lita    db      11111111b
        db      11111111b
        db      11111111b
        db      10000011b
        db      11111001b
        db      11000001b
        db      10011001b
        db      11000001b
        db      11111111b
        db      11111111b

litb    db      11111111b
        db      10011111b
        db      10011111b
        db      10000011b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10000011b
        db      11111111b
        db      11111111b

litc    db      11111111b
        db      11111111b
        db      11111111b
        db      11000011b
        db      10011001b
        db      10011111b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

litd    db      11111111b
        db      11111001b
        db      11111001b
        db      11000001b
        db      10010001b
        db      10011001b
        db      10010001b
        db      11000001b
        db      11111111b
        db      11111111b

lite    db      11111111b
        db      11111111b
        db      11111111b
        db      11000011b
        db      10011001b
        db      10000011b
        db      10011111b
        db      11000011b
        db      11111111b
        db      11111111b

litf    db      11111111b
        db      11100011b
        db      11001001b
        db      11001111b
        db      10000011b
        db      11001111b
        db      11001111b
        db      11001111b
        db      11111111b
        db      11111111b

litg    db      11111111b
        db      11111111b
        db      11111001b
        db      11000001b
        db      10010011b
        db      10010011b
        db      11000011b
        db      11110011b
        db      10010011b
        db      11000111b

lith    db      11111111b
        db      10011111b
        db      10011111b
        db      10000011b
        db      10001001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11111111b
        db      11111111b

liti    db      11111111b
        db      11111111b
        db      11100111b
        db      11111111b
        db      11000111b
        db      11100111b
        db      11100111b
        db      10000001b
        db      11111111b
        db      11111111b
```

```
litj    db      11111111b
        db      11111111b
        db      11110011b
        db      11111111b
        db      11110011b
        db      11110011b
        db      11110011b
        db      11110011b
        db      10010011b
        db      11000111b

litk    db      11111111b
        db      10011111b
        db      10011111b
        db      10010011b
        db      10000111b
        db      10000111b
        db      10010011b
        db      10011001b
        db      11111111b
        db      11111111b

litl    db      11111111b
        db      11000111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11000011b
        db      11111111b
        db      11111111b

litm    db      11111111b
        db      11111111b
        db      11111111b
        db      10010011b
        db      00101001b
        db      00101001b
        db      00101001b
        db      00111001b
        db      11111111b
        db      11111111b

litn    db      11111111b
        db      11111111b
        db      11111111b
        db      10100011b
        db      10001001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11111111b
        db      11111111b

lito    db      11111111b
        db      11111111b
        db      11111111b
        db      11000011b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

litp    db      11111111b
        db      11111111b
        db      11111111b
        db      10100011b
        db      10001001b
        db      10011001b
        db      10001001b
        db      10000011b
        db      10011111b
        db      10011111b

litq    db      11111111b
        db      11111111b
        db      11111111b
        db      11000101b
        db      10010001b
        db      10011001b
        db      10010001b
        db      11000001b
        db      11111001b
        db      11111001b

litr    db      11111111b
        db      11111111b
        db      11111111b
        db      10100011b
        db      10011001b
        db      10011111b
        db      10011111b
        db      10011111b
        db      11111111b
        db      11111111b

lits    db      11111111b
        db      11111111b
        db      11111111b
        db      11000001b
        db      10011111b
        db      11000011b
        db      11111001b
        db      10000011b
        db      11111111b
        db      11111111b

litt    db      11111111b
        db      11111111b
        db      11001111b
        db      10000011b
        db      11001111b
        db      11001111b
        db      11001001b
        db      11100011b
        db      11111111b
        db      11111111b

litu    db      11111111b
        db      11111111b
        db      11111111b
        db      10011001b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11000011b
        db      11111111b
        db      11111111b

litv    db      11111111b
        db      11111111b
        db      11111111b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11011011b
        db      11100111b
        db      11111111b
        db      11111111b

litw    db      11111111b
        db      11111111b
        db      11111111b
        db      00111001b
        db      00111001b
        db      00101001b
        db      10101011b
        db      10010011b
        db      11111111b
        db      11111111b
```

```
litx    db      11111111b
        db      11111111b
        db      11111111b
        db      10011001b
        db      11000011b
        db      11100111b
        db      11000011b
        db      10011001b
        db      11111111b
        db      11111111b

lity    db      11111111b
        db      11111111b
        db      11111111b
        db      10011001b
        db      10011001b
        db      10011001b
        db      11100001b
        db      11111001b
        db      10011001b
        db      11000011b

litz    db      11111111b
        db      11111111b
        db      11111111b
        db      10000001b
        db      11110011b
        db      11100111b
        db      11001111b
        db      10000001b
        db      11111111b
        db      11111111b

lsbrak  db      11111111b
        db      11110001b
        db      11100111b
        db      11001111b
        db      10011111b
        db      11001111b
        db      11001111b
        db      11100011b
        db      11111111b
        db      11111111b

vertl   db      11111111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11100111b
        db      11111111b

rsbrak  db      11111111b
        db      10001111b
        db      11100111b
        db      11110011b
        db      11111001b
        db      11110011b
        db      11100111b
        db      10001111b
        db      11111111b
        db      11111111b

tilde   db      11111111b
        db      10011111b
        db      01100101b
        db      11110011b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b
        db      11111111b


dseg            ends

cseg    segment byte    public  'codesg'

        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing

$gtext          proc    near

;assume that ax,bx has the column and row number of the character's location.
;assume dh has the attribute, dl has the character.

;assert dh into the fgbg. convert dl into a word address, fetch the address
;from that location and fill the char ram with it. write the character.

;we are going to assume that the character is byte alligned. anything else
;will be ignored with the char being written out to the integer of the byte
;address.

;set the text mask to enable either the first or second byte to be written
;depending on if the x coordinate is an even or odd byte.

;assume that the calling routines will have the ds setup.

;special conditions: if dl=ffh then don't print anything.
```

```
;order of events:
;
;1)make sure that the graphics option doesn't have any pending operations to
;be completed. we don't want to change colors on a line that's in the process
;of being drawn or anything like that.
;
;2)turn the x,y coordinates passed in ax,bx into a cursor word address to be
;saved and then asserted to the gdc.
;
;3)if the current foreground/background colors do not reflect the desired
;fgbg then assert the desired colors to the fgbg register.
;
;4)determine which half of the word to be written the character is to go on
;and then enable that portion of the write.
;
;5)check to see if the character we are being requested to print is legal.
;anything under 20h is consedered to be unprintable and so we just exit. we
;also consider ffh to be unprintable since the rainbow uses this code as
;a delete marker.
;
;6)turn the charter's code into a word offset. use this offset to find an
;address in a table. this table is a table of near addresses that define the
;starting address of the ten bytes that is that particular character's font.
;fetch the first two bytes and assert to the screen. we have to assert char
;ram counter reset because we are only using two of the words in the char ram,
;not all 8. each byte is loaded into both the left and right byte of a char
;ram word. the gdc is programmed to perform the two scan line write and we
;wait for the write to finish. the next 8 scan lines of the character font are
;loaded into both the left and right bytes of the char ram and these eight lines
;are then written to the screen. there is no need to wait for the 8 scans to
;finish before leaving the routine so we simply leave after setting up the gdc.

;before we do anything at all to the various registers of the graphics option
;we have to make sure that the graphics option isn't still in the preocess
;of performing a previous write operation. we can assure ourselves of a free
;gdc by loading it with a harmless command. when this command is read out of the
;command fifo then any commands previous to that must be completed. we can then
;proceed with the knowledge that the graphics option does not have any
;operations pending.

        push    ax              ;save the x coordinate.

        mov     ax,422h         ;make sure that the gdc isn't drawing.
        out     57h,al          ;write a wdat to the gdc.
here:   in      al,56h          ;read the status register.
        test    ah,al           ;did the wdat get performed by the gdc yet?
        jz      here            ;jump if not.

        pop     ax              ;restore the x coordinate.

;ax= the column number of the character. bx is the row number.
```

```
;in hires each bx is = 640 words worth. in midres each row is 320 words.

;cursor position=(ax/2)+10*(bx*scan line width in words).

        mov     di,ax           ;save the x so that we can check it later.
        shr     ax,1            ;turn column position into a word address.
        mov     cx,6            ;hi res is 64 words per line.
        shl     bx,cl                   ;bx*scan line length.
        mov     si,bx                   ;save a copy of scan times count.
        mov     cl,3                    ;to get bx*10 first mult bx by 8
        shl     bx,cl                   ;then
        add     bx,si                   ;add in the 2*bx*scan line length.
        add     bx,si                   ;this gives 10*bx*scan line length.
        add     bx,ax                   ;combine x and y into a word address.
        mov     word ptr curl0,bx       ;position to write the word at.

;assert the colors attributes of the char to fgbg. dh has the foreground and
;background attributes in it. before asserting to fgbg register check to see
;if the new colors need to be asserted or if the fgbg is already set that
;way. we do this because more often than not the desired colors will already
;be asserted and it takes less time to compare the actual with the desired
;than it does to assert the colors everytime regardless of whether or not we
;need to.

        cmp     dh,byte ptr fg          ;is the fgbg already the color we want?
        jz      cont                    ;jump if yes
        mov     bl,dh
        call    ifgbg                   ;update the foreground/background reg.

;assert the graphics board’s text mask. the gdc does 16 bit writes in text mode
;but our characters are only 8 bits wide. we must enable half of the write and
;disable the other half. if the x was odd then enable the right half. if the
;x was even then enable the left half.

cont:   test    di,1            ;is this a first byte?
        jnz     odd             ;jump if not.
        mov     word ptr gbmskl,00ffh
        jmp     com
odd:    mov     word ptr gbmskl,0ff00h
com:    call    stgbm           ;assert the graf board mask

;only the characters below 127 are defined- the others are legal but not in the
;font table....after checking for legal character fetch the address entry
;(character number-20h) in the table. this is the address of the first byte of
;the character’s font.

        cmp     dl,1fh                  ;unprintable character?
        ja      cont0                   ;jump if not.
        jmp     exit                    ;don’t try to print the illegal char.
cont0:  cmp     dl,0ffh                 ;is this a delete marker?
        jnz     cont1                   ;jump if not.
        jmp     exit                    ;jump if yes. just exit.
cont1:  sub     dl,20h                  ;table starts with a space at 0.
        xor     dh,dh
        mov     bx,dx                   ;access a table and index off bx.
        shl     bx,1                    ;turn byte into a word address offset.
        mov     si,textab[bx]           ;fetch relative tab begin char begin.

;textab has the relative offsets of each character in it so that we don’t have
;to go through a bunch of calculations to get the right address of the start of
;a particular character’s font. all we have to do is add the start of the
;font table to the relative offset of the particular character.

        add     si,offset space         ;combine table offset with char offset.

;transfer the font from the font table into char ram. write the first two scans
;then do the last 8.

        cld                     ;make sure lodsb incs si.

        mov     al,clrcnt       ;reset the char ram counter.
        out     53h,al
        out     51h,al
        lodsw           ;fetch both bytes.
        out     chram,al        ;put the byte into both 1 and 2 char ram bytes.
        out     chram,al
        mov     al,ah
        out     chram,al        ;put the byte into both 1 and 2 char ram bytes.
        out     chram,al
        mov     al,clrcnt       ;reset the char ram counter.
        out     53h,al
        out     51h,al

;check to see if already in in text mode.

        test    gbmod,2
        jz      textm           ;jump if already in text mode else
        and     gbmod,0fdh      ;assert text mode.
        call    imode
textm:  mov     al,curs         ;assert the cursor command.
        out     57h,al
        mov     ax,word ptr curl0
        out     56h,al
        mov     al,ah
        out     56h,al
        mov     al,gmask        ;assert the mask command.
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al
        mov     al,figs         ;assert the figs command.
        out     57h,al
        xor     al,al           ;assert the down directinon to write.
        out     56h,al
        mov     al,1            ;do it 2 write cycles.
        out     56h,al
        xor     al,al
        out     56h,al
        mov     al,22h          ;assert the wdat command.
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al

;wait for the first two scans to be written.

        mov     ax,422h         ;make sure that the gdc isn't drawing.
        out     57h,al          ;write a wdat to the gdc.
here1:  in      al,56h          ;read the status register.
        test    ah,al           ;did the wdat get performed by the gdc yet?
        jz      here1           ;jump if not.

;si is still pointing to the next scan line to be fetched. get the next two
;scan lines and then tell the gdc to write them. no new cursor,gdc mask, graf
;mask or mode commands need to be issued.

        mov     cx,8            ;eight scan lines.
ldcr:   lodsb           ;fetch the byte.
        out     chram,al        ;put the byte into both 1 and 2 char ram bytes.
        out     chram,al
        loop    ldcr

        mov     al,figs         ;assert the figs command.
        out     57h,al
        xor     al,al           ;assert the down directinon to write.
        out     56h,al
        mov     ax,7            ;do 8 write cycles.
        out     56h,al
        mov     al,ah
        out     56h,al
        mov     al,22h          ;assert the wdat command.
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al

exit:   ret

$gtext          endp

cseg            ends
```

```
        end
```

## 9.2  DEFINE AND POSITION THE CURSOR

There are two routines in the following example.  One sets the cursor type to no cursor, block, underscore, or block and underscore.  It then sets up the current cursor location and calls the second routine.  The second routine accepts new coordinates for the cursor and moves the cursor to the new location.

### 9.2.1  Example Of Defining And Positioning The Cursor

```
        title   8x10 cursor routines
        page    80,132

;*************************************************************************
;*                                                                      *
;*                                                                      *
;*                                                                      *
;*                                                                      *
;*              c u r s o r   r o u t i n e s                           *
;*                                                                      *
;*                                                                      *
;*                                                                      *
;*  purpose:    assert and display cursors                              *
;*                                                                      *
;*                                                                      *
;*                                                                      *
;*                                                                      *
;*                                                                      *
;*************************************************************************

dseg    segment byte    public  'datasg'

;       port equates

cmaskl  equ     54h     ;graphics text mask right byte.
cmaskh  equ     55h     ;graphics text mask left byte.
gstat   equ     56h     ;gdc status register.

;       define the gdc commands

curs    equ     49h     ;cursor display charcteristics specify command
```

```
figs    equ     4ch     ;figure specify command.

block   db      0,0,0,0,0,0,0,0,0,0
cdis    db      0
lastcl  dw      0               ;last location the cursor was displayed at.
        dw      0
ocurs   db      0               ;last cursor type displayed.
newcl   dw      0
        dw      0
ncurs   db      0
unders  db      0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0,0ffh
userd   db      0,0,0,0,0,0,0,0,0,0

dseg            ends

        title  gcurs.asm
        subttl  gsettyp.asm
        page    60,132

;*************************************************************************
;*                                                                       *
;*              p r o c e e d u r e    g s e t t y p                     *
;*                                                                       *
;*  purpose:    assert new cursor type                                   *
;*  entry:      dl bits set to determine cursor style                    *
;*                      no bits set-no cursor                            *
;*                      d0=block                                         *
;*                      d1=undefined                                     *
;*                      d2=undefined                                     *
;*                      d3=underscore                                    *
;*  exit:                                                                *
;*  registers:                                                           *
;*  stack usage:                                                         *
;*                                                                       *
;*                                                                       *
;*                                                                       *
;*************************************************************************


extrn   alu:byte,curl0:byte,curl2:byte,fg:byte,gbmod:byte

extrn   ifgbg:near,imode:near

;impliments the new cursor type to be displayed. the current cursor type and
;location must become the old type and location. the new type becomes whatever
;is in dl. this routine will fetch the previous cursor type out of ncurs and
;put it into ocurs and put the new cursor type into ncurs. the previous cursor
;locaion is fetched and put into ax,bx. gsetpos is then jumped to in order that
;the old cursor can be erased and the new displayed.

;type bits are not exclusive of each other. a cursor can be both an underscore
```

```
;and a block.

;dl=    0=turn the cursor display off
;       1=display the insert cursor (full block)
;       8=display the overwrite cursor (underscore)
;       9=display a simultaneous underscore and block cursor.

;after the new type has been applied the new cursor need to be displayed.
;put the current cursor type into the previous cursor type storage register.
;update the current cursor type register to the new desired cursor type. move
;the current cursor's location into the proper registers so that after the
;previous cursor is erased the new cursor will be displayed at the same
;location.

cseg    segment byte    public  'codesg'

        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing

        public  gsettyp

gsettyp         proc    near

        mov     al,byte ptr ncurs       ;current cursor is about to become
        mov     byte ptr ocurs,al       ;old cursor type. this is old to erase.
        mov     byte ptr ncurs,dl       ;this is the new to assert.

        mov     ax,word ptr newcl       ;pick up the current x and y so that
        mov     bx,word ptr newcl+2     ;we can display new cur at old loc.
        jmp     pos                     ;assert new cursor type in old position.

gsettyp endp

        subttl  gsetpos.asm
        page    60,132

;*************************************************************************
;*                                                                       *
;*              p r o c e e d u r e     g s e t p o s                    *
;*                                                                       *
;*  purpose:    assert new cursor position                               *
;*  entry:      ax=x location                                            *
;*              bx=y location                                            *
;*  exit:                                                                *
;*  registers:                                                           *
;*  stack usage:                                                         *
;*                                                                       *
;*                                                                       *
;*                                                                       *
;*************************************************************************

                public  gsetpos
```

```
gsetpos proc    near

;display the cursor. cursor type was defined by gsettyp. the cursor type
;is stored in ncurs. fetch the type and address of the previous cursor and
;put into ocurs and lastcl,lastcl+2. if there is a previous cursor displayed
;then erase the old cursor. if there is a new cursor to display then write
;it (or them) to the screen. a cursor may be a blcok or an underscore or
;both.

;the x,y location of the cursor is converted into an address that the
;gdc can use. either the left or the right half of the text mask is enabled
;depending on if the x is even or odd. the write operation itself takes places
;in compliment mode so that no information on the screen is lost or obscured,
;only inverted in value. in order to insure that all planes are inverted a f0
;is loaded into the fgbg register and all planes are write enabled. the cursor
;is written to the screen in two separate writes because the character ram is
;eight, not ten, words long. after the cursor is written to the screen the
;previous graphics states are restored.

;move current cursor type and location to previous type and location.

        mov     cl,byte ptr ncurs       ;turn old curs type into old curs type.
        mov     byte ptr ocurs,cl

pos:    cld
        mov     cx,word ptr newcl       ;turn current location into previous
        mov     word ptr lastcl,cx      ;location.
        mov     cx,word ptr newcl+2
        mov     word ptr lastcl+2,cx

        mov     word ptr newcl,ax       ;save the new cursor location x,y
        mov     word ptr newcl+2,bx     ;coordinates.

;before we do anything to the graphics option we need to make sure that the
;option isn’t already in use. assert a harmless command into the fifo and then
;wait for the gdc to eat it.

        call    not__busy

;setup of the graphics option. put graphics option into compliment, text mode.
;assert fgbg and text mask. calculate the address at which to do the write and
;store in curl0,1.

;assert compliment all planes. the normal ialups routine saves the alups byte in
;register byte alu. this byte will be left undisturbed and will be used later to
;restore the alups to its former state.

        mov     ax,10efh                ;address the alups.
        out     53h,al
        mov     al,ah                   ;issue the compliment mode, all planes
        out     51h,al                  ;enabled byte.
```

```
;assert text mode with read disabled.

        mov     al,byte ptr gbmod       ;fetch the graphics mode byte.
        and     al,0fdh                 ;make sure in text mode.
        or      al,10h                  ;make sure in write enabled mode.
        cmp     al,byte ptr gbmod       ;is the mode already asserted this way?
        jz      gspos0                  ;jump if yes.
        mov     byte ptr gbmod,al       ;update the mode register and assert it.
        call    imode

;assert fgbg of f0.

gspos0: mov     bl,0f0h                 ;is fgbg already f0?
        cmp     byte ptr fg,bl          ;jump if yes else assert the
        jz      gsp01                   ;compliment all colors cursor.
        call    ifgbg

;is there a cursor currently being displayed? if cdis<>0 then yes. any
;current cursor will have to be erased before we display the new one.

gsp01:  test    byte ptr cdis,1
        jz      gspos2          ;no old cursor to erase. just display old.

;this part will erase the old cursor.

        mov     byte ptr cdis,0         ;set no cursor currently on screen.
        mov     dh,byte ptr lastcl      ;fetch x and y. put into dx and call
        mov     dl,byte ptr lastcl+2    ;dx2curl.
        call    asmask                  ;assert the mask registers.
        call    dx2curl                 ;turn dx into a gdc cursor loc address.

        test    byte ptr ocurs,8        ;underline?
        jz      gspos1                  ;jump if not.
        mov     si,offset unders        ;erase the underline.
        call    discurs                 ;write it.
gspos1: test    byte ptr ocurs,1        ;block?
        jz      gspos2                  ;jump if not.

        call    not__busy        ;wait till done if erasing underscore.

        mov     si,offset block         ;erase the block.
        call    discurs                 ;do the write.

;write the new cursor out to the screen.

gspos2: cmp     byte ptr ncurs,0        ;are we going to write a new cursor?
        jz      gspos5                  ;jump if not.

        mov     dh,byte ptr newcl       ;fetch coordinates to write new cursor.
        mov     dl,byte ptr newcl+2
        call    not__busy        ;wait for erase to finish.

        call    asmask                  ;assert the mask registers.
        call    dx2curl
        test    byte ptr ncurs,8        ;underscore?
        jz      gspos3                  ;jump if not.
        mov     si,offset unders        ;write the underline cursor.
        call    discurs                 ;write it.
gspos3: test    byte ptr ncurs,1        ;block cursor?
        jz      gspos4                  ;jump if not.

        call    not__busy        ;wait for block write to finish.

        mov     si,offset block         ;write the block cursor.
        call    discurs                 ;do the write.
gspos4: or      byte ptr cdis,1         ;set cursor displayed flag.

gspos5: call    not__busy

        mov     al,0efh                 ;recover previous alups byte and then
        out     53h,al                  ;apply it to the alups register.
        mov     al,byte ptr alu
        out     51h,al
        ret

;enable one byte of the text mask.

asmask: mov     ax,00ffh        ;setup the text mask.
        test    dh,1            ;write to the right byte?
        jz      ritc4           ;jump if yes
        mov     ax,0ff00h
ritc4:  out     cmaskl,al               ;issue the low byte of mask.
        mov     al,ah
        out     cmaskh,al               ;issue the high byte of the text mask.
        ret

;display the cursor. assume that the graphics option is already setup and
;that the option is in text mode, compliment write and that the appropritate
;textmask is already set. si is loaded with the address to fetch the cursor
;pattern from.

discurs:
        mov     al,0feh         ;clear the char ram counter.
        out     53h,al
        out     51h,al          ;fetch first two lines of the cursor.
        lodsb
        out     52h,al          ;feed the same byte to both halves of
        out     52h,al          ;the word to be written.
        lodsb
        out     52h,al          ;feed the same byte to both left and right
        out     52h,al          ;bytes to be written.
        mov     al,0feh         ;clear the char ram counter.
        out     53h,al
        out     51h,al

        mov     al,curs         ;assert the position to write.
        out     57h,al
        mov     ax,word ptr curl0
        out     56h,al
        mov     al,ah
        out     56h,al

        mov     al,4ah          ;issure the gdc mask command,
        out     57h,al          ;set all gdc mask bits.
        mov     al,0ffh
        out     56h,al
        out     56h,al

        mov     al,figs         ;program a write of ten scans. do 2 then 8.
        out     57h,al
        xor     al,al
        out     56h,al
        mov     al,1
        out     56h,al
        xor     al,al
        out     56h,al
        mov     al,22h          ;start the write.
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al

        call    not__busy        ;wait for first 2 lines to finish.

        mov     cx,8            ;move and then write the next 8 scans.
ritc6:  lodsb                   ;fetch the cursor shape.
        out     52h,al          ;feed the same byte to both left and right sides
        out     52h,al          ;of the word.
        loop    ritc6

        mov     al,figs         ;program a write of 8 scans.
        out     57h,al
        xor     al,al
        out     56h,al
        mov     al,7
        out     56h,al
        xor     al,al
        out     56h,al
        mov     al,22h          ;start the write.
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al

        ret

;turn dh,dl into a word address. dl is the line, dh is the column. store
;result in word ptr curl0.

;start with turning dl (row) into a word address.
;word address=row*number of words per line*10
;turn column into a word address.
;word address=column/2
;combine the two. this gives the curl0 address to be asserted to the gdc.

dx2curl:
        mov     al,dh           ;put the column count safely away.
        mov     cl,5            ;lowres is 32 words per line
        test    byte ptr gbmod,1        ;high res?
        jz      ritc5           ;jump if not.
        inc     cl              ;high res is 64 words per line.
ritc5:  xor     dh,dh
        shl     dx,cl
        mov     bx,dx           ;multiply dx times ten.
        mov     cl,3
        shl     bx,1
        shl     dx,cl
        add     dx,bx           ;this is the row address.
        shr     al,1            ;this is the column number.
        xor     ah,ah
        add     dx,ax           ;this is the combined row and column address.
        mov     word ptr curl0,dx
        ret

;this is a quicker version of gdc__not__busy. we don’t waste time on some of the
;normal checks and things that gdc__not__busy does due to the need to move as
;quickly as possible on the cursor erase/write routines. this routine does the
;same sort of things. a harmless command is issued to the gdc. if the gdc is
;in the process of performing some other command then the wdat we just issued
;will stay in the gdc’s command fifo untill such time as the gdc can get to it.
;if the fifo empty bit is set then the gdc ate the wdat command and must be
;finished with any previous operations programmed into it.

not__busy:
        mov     ax,422h         ;assert a wdat and then wait for fifo to empty.
        out     57h,al
busy:   in      al,gstat        ;wait for fifo empty bit to be asserted.
        test    ah,al
        jz      busy
        ret

gsetpos endp
```

```
cseg            ends

end
```

## 9.3 WRITE A TEXT STRING

The example in this section writes a string of ASCII text starting at a specified location and using a specified scale factor. It uses the vector write routine from Chapter 8 to form each character.

### 9.3.1 Example Of Writing A Text String

```
;*********************************************************************
;
;       p r o c e d u r e   v e c t o r _ t e x t
;
;
;       entry:          cx = string length
;                       text = externally defined array of ascii
;                               characters.
;                       scale = character scale
;                       xinit = starting  x location
;                       yinit = starting  y location
;*********************************************************************
cseg    segment byte    public  'codesg'
         extrn   imode:near,pattern_mult:near,pattern_register:near
         extrn   vector:near
        public  vector_text
        assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
vector_text     proc    near
        or      byte ptr gbmod,082h
         call    imode
        mov     al,4ah
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al          ;enable gdc mask data write
        xor     al,al           ;enable all gb mask writes.
        out     55h,al
        out     54h,al
        mov     bl,1
        call    pattern_mult    ;set pattern multiplier
        mov     bl,0ffh          ;(see example 20)
        call    pattern_register ;set pattern register
```

```
                                  ;(see example 19)
        mov     ax,word ptr xinit       ;get initial x
        mov     word ptr xad,ax         ;save it
        mov     ax,word ptr yinit       ;get initial y
        mov     word ptr yad,ax         ;save it
        mov     si,offset text
do_string:
        lodsb                   ;get character
        push    si
        push    cx
        call    display_character       ;display it
        mov     ax,8
        mov     cl,byte ptr scale       ;move over by cell value
        mul     cx
        add     word ptr xad,ax
        pop     cx
        pop     si
        loop    do_string               ;loop until done
        ret
display_character:
        cmp     al,07fh         ;make sure we’re in table
        jbe     char_cont_1     ;continue if we are
        ret
char_cont_1:
        cmp     al,20h          ;make sure we can print character
        ja      char_cont       ;continue if we can
        ret
char_cont:
        xor     ah,ah           ;clear high byte
        shl     ax,1            ;make it a word pointer
        mov     si,ax
        mov     si,font_table[si]       ;point si to font info
get_next_stroke:
        mov     ax,word ptr xad
        mov     word ptr xinit,ax
        mov     ax,word ptr yad
        mov     word ptr yinit,ax
        lodsb                           ;get stroke info
        cmp     al,endc                 ;end of character ?
        jnz     cont_1                  ;continue if not
        ret
cont_1: mov     bx,ax
        and     ax,0fh                  ;mask to y value
        test    al,08h                  ;negative ?
        jz      ct
        or      ax,0fff0h               ;sign extend
ct:     mov     cl,byte ptr scale
        xor     ch,ch
        push    cx
        imul    cx                      ;multiply by scale value
        sub     word ptr yinit,ax       ;subtract to y offset
        and     bx,0f0h                 ;mask to x value
        shr     bx,1                    ;shift to 4 lsb
        shr     bx,1
        shr     bx,1
        shr     bx,1
        test    bl,08h                  ;negative ?
        jz      ct1
        or      bx,0fff0h               ;sign extend
ct1:    mov     ax,bx
        pop     cx                      ;recover scale
        imul    cx                      ;multiply by scale value
        add     word ptr xinit,ax       ;add to x offset
next_stroke:
        mov     ax,word ptr xad         ;set up xy offsets
        mov     word ptr xfinal,ax
        mov     ax,word ptr yad
        mov     word ptr yfinal,ax
        lodsb                           ;get stroke byte
        cmp     al,endc                 ;end of character ?
        jz      display_char_exit       ;yes then leave
        cmp     al,endv                 ;dark vector ?
        jz      get_next_stroke         ;yes, begin again
        mov     bx,ax
        and     ax,0fh                  ;mask to y value
        test    al,08h                  ;negative
        jz      ct2
        or      ax,0fff0h               ;sign extend
ct2:    mov     cl,byte ptr scale       ;get scale info
        xor     ch,ch
        push    cx
        imul    cx                      ;multiply by scale
        sub     word ptr yfinal,ax      ;subtract to y offset
        and     bx,0f0h                 ;mask to x value
        shr     bx,1                    ;shift to 4 lsb
        shr     bx,1
        shr     bx,1
        shr     bx,1
        test    bl,08h                  ;negative ?
        jz      ct3
        or      bx,0fff0h               ;sign extend
ct3:    mov     ax,bx
        pop     cx                      ;recover scale
        imul    cx                      ;multiply by scale
        add     word ptr xfinal,ax      ;add to x offset
        push    si                      ;save index to font info
        call    vector                  ;draw stroke
        pop     si                      ;recover font index
        mov     ax,word ptr xfinal      ;end of stroke becomes
        mov     word ptr xinit,ax       ;beginning of next stroke
        mov     ax,word ptr yfinal
        mov     word ptr yinit,ax
        jmp     next_stroke
display_char_exit:
        ret
vector_text     endp
cseg    ends
dseg    segment byte    public  'datasg'
extrn   gbmod:byte,xinit:word,yinit:word,xfinal:word,yfinal:word
extrn   xad:word,yad:word,text:byte
public  scale
;*******************************************************************
;*                                                                 *
;*                      stroke font character set                  *
;*                                                                 *
;*******************************************************************
;
;the following tables are vertice information for a stroked character
;set the x,y coordinate information is represented by 4 bit 2's
;complement numbers in the range of +-7 x, +-7 y. end of character
;is represented by -8 x, -8 y and dark vector is represented by -8 x,
; 0 y.
;
;       bit     7 6 5 4 3 2 1 0
;               |     | |     |
;                \   /   \   /
;                  x       y
;
;ascii characters are currently mapped into the positive quadrant,
;with the origin at the lower left corner of an upper case character.
;
endc            equ     10001000b           ;end of character
endv            equ     10000000b           ;last vector of polyline
;
font_table      dw      offset  font_00
                dw      offset  font_01
                dw      offset  font_02
                dw      offset  font_03
                dw      offset  font_04
                dw      offset  font_05
                dw      offset  font_06
                dw      offset  font_07
                dw      offset  font_08
                dw      offset  font_09
                dw      offset  font_0a
                dw      offset  font_0b
                dw      offset  font_0c
                dw      offset  font_0d
                dw      offset  font_0e
                dw      offset  font_0f
                dw      offset  font_10
                dw      offset  font_11
                dw      offset  font_12
                dw      offset  font_13
                dw      offset  font_14
                dw      offset  font_15
                dw      offset  font_16
                dw      offset  font_17
                dw      offset  font_18
                dw      offset  font_19
                dw      offset  font_1a
                dw      offset  font_1b
                dw      offset  font_1c
                dw      offset  font_1d
                dw      offset  font_1e
                dw      offset  font_1f
                dw      offset  font_20                 ;space
                dw      offset  font_21                 ;!
                dw      offset  font_22
                dw      offset  font_23
                dw      offset  font_24
                dw      offset  font_25
                dw      offset  font_26
                dw      offset  font_27
                dw      offset  font_28
                dw      offset  font_29
                dw      offset  font_2a
                dw      offset  font_2b
                dw      offset  font_2c
                dw      offset  font_2d
                dw      offset  font_2e
                dw      offset  font_2f
                dw      offset  font_30
                dw      offset  font_31
                dw      offset  font_32
                dw      offset  font_33
                dw      offset  font_34
                dw      offset  font_35
                dw      offset  font_36
                dw      offset  font_37
                dw      offset  font_38
                dw      offset  font_39
                dw      offset  font_3a
                dw      offset  font_3b
                dw      offset  font_3c
                dw      offset  font_3d
                dw      offset  font_3e
                dw      offset  font_3f
                dw      offset  font_40
                dw      offset  font_41
                dw      offset  font_42
                dw      offset  font_43
                dw      offset  font_44
                dw      offset  font_45
```

```
                dw      offset  font_46
                dw      offset  font_47
                dw      offset  font_48
                dw      offset  font_49
                dw      offset  font_4a
                dw      offset  font_4b
                dw      offset  font_4c
                dw      offset  font_4d
                dw      offset  font_4e
                dw      offset  font_4f
                dw      offset  font_50
                dw      offset  font_51
                dw      offset  font_52
                dw      offset  font_53
                dw      offset  font_54
                dw      offset  font_55
                dw      offset  font_56
                dw      offset  font_57
                dw      offset  font_58
                dw      offset  font_59
                dw      offset  font_5a
                dw      offset  font_5b
                dw      offset  font_5c
                dw      offset  font_5d
                dw      offset  font_5e
                dw      offset  font_5f
                dw      offset  font_60
                dw      offset  font_61
                dw      offset  font_62
                dw      offset  font_63
                dw      offset  font_64
                dw      offset  font_65
                dw      offset  font_66
                dw      offset  font_67
                dw      offset  font_68
                dw      offset  font_69
                dw      offset  font_6a
                dw      offset  font_6b
                dw      offset  font_6c
                dw      offset  font_6d
                dw      offset  font_6e
                dw      offset  font_6f
                dw      offset  font_70
                dw      offset  font_71
                dw      offset  font_72
                dw      offset  font_73
                dw      offset  font_74
                dw      offset  font_75
                dw      offset  font_76
                dw      offset  font_77
                dw      offset  font_78
                dw      offset  font_79
                dw      offset  font_7a
                dw      offset  font_7b
                dw      offset  font_7c
                dw      offset  font_7d
                dw      offset  font_7e
                dw      offset  font_7f
;
font_00         db      endc
font_01         db      endc
font_02         db      endc
font_03         db      endc
font_04         db      endc
font_05         db      endc
font_06         db      endc
font_07         db      endc
font_08         db      endc
font_09         db      endc
font_0a         db      endc
font_0b         db      endc
font_0c         db      endc
font_0d         db      endc
font_0e         db      endc
font_0f         db      endc
font_10         db      endc
font_11         db      endc
font_12         db      endc
font_13         db      endc
font_14         db      endc
font_15         db      endc
font_16         db      endc
font_17         db      endc
font_18         db      endc
font_19         db      endc
font_1a         db      endc
font_1b         db      endc
font_1c         db      endc
font_1d         db      endc
font_1e         db      endc
font_1f         db      endc
font_20         db      endc                    ;space
font_21         db 20h,21h,endv,23h,26h,endc
font_22         db 24h,26h,endv,54h,56h,endc
font_23         db 20h,26h,endv,40h,46h,endv,04h,64h,endv,02h,62h
                 db endc
font_24         db 2fh,27h,endv,01h,10h,30h,41h,42h,33h,13h,04h,05h
                 db 16h,36h,045h,endc
font_25         db 11h,55h,endv,14h,15h,25h,24h,14h,endv,41h,51h,52h
                 db 42h,41h,endc
font_26         db 50h,14h,15h,26h,36h,45h,44h,11h,10h,30h,52h,endc
font_27         db 34h,36h,endc
```

```
font_28         db 4eh,11h,14h,47h,endc
font_29         db 0eh,31h,34h,07h,endc
font_2a         db 30h,36h,endv,11h,55h,endv,15h,51h,endv,03h,63h
                 db endc
font_2b         db 30h,36h,endv,03h,63h,endc
font_2c         db 11h,20h,2fh,0dh,endc
font_2d         db 03h,63h,endc
font_2e         db 00h,01h,11h,10h,00h,endc
font_2f         db 00h,01h,45h,46h,endc
font_30         db 01h,05h,16h,36h,45h,41h,30h,10h,01h,endc
font_31         db 04h,26h,20h,endv,00h,040h,endc
font_32         db 05h,16h,36h,45h,44h,00h,40h,041h,endc
font_33         db 05h,16h,36h,45h,44h,33h,42h,41h,30h,10h,01h,endv
                 db 13h,033h,endc
font_34         db 06h,03h,043h,endv,20h,026h,endc
font_35         db 01h,10h,30h,41h,42h,33h,03h,06h,046h,endc
font_36         db 02h,13h,33h,42h,41h,30h,10h,01h,05h,16h,36h,045h
                 db endc
font_37         db 06h,46h,44h,00h,endc
font_38         db 01h,02h,13h,04h,05h,16h,36h,45h,44h,33h,42h,41h
                 db 30h,10h,01h,endv,13h,023h,endc
font_39         db 01h,10h,30h,41h,45h,36h,16h,05h,04h,13h,33h,044h
                 db endc
font_3a         db 15h,25h,24h,14h,15h,endv,12h,22h,21h,11h,12h
                 db endc
font_3b         db 15h,25h,24h,14h,15h,endv,21h,11h,12h,22h,20h,1fh
                 db endc
font_3c         db 30h,03h,036h,endc
font_3d         db 02h,042h,endv,04h,044h,endc
font_3e         db 10h,43h,16h,endc
font_3f         db 06h,17h,37h,46h,45h,34h,24h,022h,endv,21h,020h
                 db endc
font_40         db 50h,10h,01h,06h,17h,57h,66h,63h,52h,32h,23h,24h
                 db 35h,55h,064h,endc
font_41         db 00h,04h,26h,44h,040h,endv,03h,043h,endc
font_42         db 00h,06h,36h,45h,44h,33h,42h,41h,30h,00h,endv
                 db 03h,033h,endc
font_43         db 45h,36h,16h,05h,01h,10h,30h,041h,endc
font_44         db 00h,06h,36h,45h,41h,30h,00h,endc
font_45         db 40h,00h,06h,046h,endv,03h,023h,endc
font_46         db 00h,06h,046h,endv,03h,023h,endc
font_47         db 45h,36h,16h,05h,01h,10h,30h,41h,43h,023h,endc
font_48         db 00h,06h,endv,03h,043h,endv,40h,046h,endc
font_49         db 10h,030h,endv,20h,026h,endv,16h,036h,endc
font_4a         db 01h,10h,30h,41h,046h,endc
font_4b         db 00h,06h,endv,02h,046h,endv,13h,040h,endc
font_4c         db 40h,00h,06h,endc
font_4d         db 00h,06h,24h,46h,040h,endc
font_4e         db 00h,06h,endv,05h,041h,endv,40h,046h,endc
font_4f         db 01h,05h,16h,36h,45h,41h,30h,10h,01h,endc
font_50         db 00h,06h,36h,45h,44h,33h,03h,endc
```

```
font_51         db 12h,30h,10h,01h,05h,16h,36h,45h,41h,30h,endc
font_52         db 00h,06h,36h,45h,44h,33h,03h,endv,13h,040h,endc
font_53         db 01h,10h,30h,41h,42h,33h,13h,04h,05h,16h,36h
                 db 045h,endc
font_54         db 06h,046h,endv,20h,026h,endc
font_55         db 06h,01h,10h,30h,41h,046h,endc
font_56         db 06h,02h,20h,42h,046h,endc
font_57         db 06h,00h,22h,40h,046h,endc
font_58         db 00h,01h,45h,046h,endv,40h,41h,05h,06h,endc
font_59         db 06h,24h,020h,endv,24h,46h,endc
font_5a         db 06h,46h,45h,01h,00h,40h,endc
font_5b         db 37h,17h,1fh,3fh,endc
font_5c         db 06h,05h,41h,40h,endc
font_5d         db 17h,37h,3fh,2fh,endc
font_5e         db 04h,26h,044h,endc
font_5f         db 0fh,07fh,endc
font_60         db 54h,36h,endc
font_61         db 40h,43h,34h,14h,03h,01h,10h,30h,041h,endc
font_62         db 06h,01h,10h,30h,41h,43h,34h,14h,03h,endc
font_63         db 41h,30h,10h,01h,03h,14h,34h,043h,endc
font_64         db 46h,41h,30h,10h,01h,03h,14h,34h,43h,endc
font_65         db 41h,30h,10h,01h,03h,14h,34h,43h,42h,02h,endc
font_66         db 20h,25h,36h,46h,55h,endv,03h,43h,endc
font_67         db 41h,30h,10h,01h,03h,14h,34h,43h,4fh,3eh,1eh
                 db 0fh,endc
font_68         db 00h,06h,endv,03h,14h,34h,43h,40h,endc
font_69         db 20h,23h,endv,25h,26h,endc
font_6a         db 46h,45h,endv,43h,4fh,3eh,1eh,0fh,endc
font_6b         db 00h,06h,endv,01h,34h,endv,12h,30h,endc
font_6c         db 20h,26h,endc
font_6d         db 00h,04h,endv,03h,14h,23h,34h,43h,40h,endc
font_6e         db 00h,04h,endv,03h,14h,34h,43h,40h,endc
font_6f         db 01h,03h,14h,34h,43h,41h,30h,10h,01h,endc
font_70         db 04h,0eh,endv,01h,10h,30h,41h,43h,34h,14h
                 db 03h,endc
font_71         db 41h,30h,10h,01h,03h,14h,34h,43h,endv,44h
                 db 4eh,endc
font_72         db 00h,04h,endv,03h,14h,34h,endc
font_73         db 01h,10h,30h,41h,32h,12h,03h,14h,34h
                 db 43h,endc
font_74         db 04h,44h,endv,26h,21h,30h,40h,51h,endc
font_75         db 04h,01h,10h,30h,41h,endv,44h,40h,endc
font_76         db 04h,02h,20h,42h,44h,endc
font_77         db 04h,00h,22h,40h,44h,endc
font_78         db 00h,44h,endv,04h,40h,endc
font_79         db 04h,01h,10h,30h,41h,endv,44h,4fh,3eh,1eh
                 db 0fh,endc
font_7a         db 04h,44h,00h,40h,endc
font_7b         db 40h,11h,32h,03h,34h,15h,46h,endc
font_7c         db 20h,23h,endv,25h,27h,endc
font_7d         db 00h,31h,12h,43h,14h,35h,06h,endc
```

```
font_7e         db 06h,27h,46h,67h,endc
font_7f         db 07,77,endc

scale   db      0
dseg    ends
        end
```

# CHAPTER 10

# READ OPERATIONS

## 10.1 THE READ PROCESS

Programming a read operation is simpler than programming a write operation. From the Graphics Option's point of view, only the Mode and ALUPS registers need to be programmed. There is no need to involve the Foreground/Background Register, Text Mask, Write Buffer, or the Pattern Generator. From the GDC's point of view, reading is programmed much like a text write except for the action command which in this case is RDAT. When reading data from the bitmap, only one plane can be active at any one time. Therefore, it can take four times as long to read back data as it did to write it in the first place.

## 10.2 READ A PARTIAL BITMAP

The following is an annotated step-by-step procedure for reading the first ten lines of plane 1 in high resolution mode.

### 10.2.1 Load The Mode Register

This readback operation assumes high resolution, text mode, readback enabled for plane 1, scroll map load disabled, interrupt disabled, and monitor on. Accordingly, select the Mode Register with a BFh to port 53h and load the register with an A5h to port 51h.

### 10.2.2 Load The ALUPS Register

Whenever the GDC accesses the bitmap, it goes through the entire Read/Modify/Write (RMW) cycle. Therefore, writes must be disabled by setting the low-order nibble of the ALUPS Register to all ones; the contents of the high-order nibble are immaterial. Select the ALUPS

Register with an EFh to port 53h and load the register with a 0Fh to port 51h.

NOTE

> This completes the setup of the external hardware. The GDC can now be conditioned to perform the actual read. GDC commands are written to port 57h; GDC parameters are written to port 56h.

### 10.2.3 Set The GDC Start Location

The Cursor command (49h) tells the GDC where to start reading. For a read operation it takes two parameter bytes: the low−order and high−order bytes of the first word address to be read from. Write 49h to port 57h and two bytes of zeros to port 56h.

### 10.2.4 Set The GDC Mask

The GDC Mask is a 16−bit recirculating buffer. The GDC rotates the mask with each write operation. When a one bit rotates out of the mask, the GDC increments the word address. This operation requires that the GDC increment the word address after each write so the mask is loaded with all ones. Write 4Ah to port 57h and two bytes of FFh to port 56h.

### 10.2.5 Program The GDC To Read

The FIGS command (4Ch) provides the GDC with the direction of the read operation and the number of RMW cycles to take. The direction is incrementing through memory, down the video scan line to the right (code 2). Ten lines at high resolution add up to 640 words (10 X 64 words/line) or 280h. Write 4Ch to port 57h and the three bytes 02h, 80h, and 02h to port 56h.

While the number of writes is always one more than the number programmed, the number of read operations is always the exact number entered. In high resolution mode, there are 4000h word addresses in a plane. However, there are only 14 bits in the parameter bytes defining the number of words to be read. If a read of the entire plane is required, two read operations must be performed. The maximum number of words that can be read at any one time is 3FFFh or one less than 16K words.

The RDAT command (A0h) initiates the read operation and sets the read mode to word transfer, first low byte then high byte. RDAT does not take parameters.

As data from the bitmap becomes available in the GDC's FIFO buffer, bit 0 (DATA READY) in the GDC status register will be set. The CPU can interrogate this bit and read any available data out of the FIFO. If the FIFO becomes full before the GDC has completed the specified number of reads, the read cycles are suspended until the CPU has made more room by reading some data out.

## 10.3 READ THE ENTIRE BITMAP

In the following example, the entire bitmap, one plane at a time, is read and written into an arbitrary 64K byte buffer in memory. This example compliments the example of displaying data from memory found in Chapter 7.

### 10.3.1 Example Of Reading The Entire Bitmap

```
        title   read entire video screen
        subttl  redvid
        page 60,132

;*****************************************************************************
;                                                                            *
;                                                                            *
;                                                                            *
;               p r o c e s s          r e d v i d                           *
;                                                                            *
;                                                                            *
;                                                                            *
;this routine will read out all of video memory one plane at a time and then  *
;store that data in a 64k buffer in motherboard memory.                      *
;                                                                            *
;                                                                            *
;                                                                            *
;*****************************************************************************

dseg    segment byte    public 'datasg'

;       define the graphics commands
;
cchar   equ     4bh     ;cursor/character characteristics command
```

```
curd    equ     0e0h    ;display the cursor at a specified location command
curs    equ     49h     ;cursor display charcteristics specify command
figd    equ     6ch
figs    equ     4ch
gchrd   equ     68h
lprd    equ     0a0h
gmask   equ     4ah     ;sets which of the 16 bits/word affected
pitch   equ     47h
pram    equ     70h     ;write to param ram pointed to by pram com low nibble
rdat    equ     60h     ;read command.
reset   equ     00      ;reset command
rmwr    equ     20h     ;read modify write operation replacing screen data
s__off   equ     0ch     ;blank the display command
s__on    equ     0dh     ;turn display on command
start   equ     6bh     ;starts gdc video processes
sync    equ     0fh     ;always enabling screen
vsync   equ     6fh     ;gdc vsync input/output pin set to output
zoom    equ     46h     ;gdc zoom command
;
;       define the graphics board port addresses
;
graf    equ     50h     ;graphics board base address port 0
gindo   equ     51h     ;graphics board indirect port enable out address
chram   equ     52h     ;character ram
gindl   equ     53h     ;graphics board indirect port in load address
cmaskh  equ     55h     ;character mask high
cmaskl  equ     54h     ;character mask low
gstat   equ     56h     ;gdc status reg (read only)
gpar    equ     56h     ;gdc command parameters (write only)
gread   equ     57h     ;gdc data read from vid mem (read only)
gcmd    equ     57h     ;gdc command port (write only)
;
;define the indirect register select enables
;
clrcnt  equ     0feh    ;clear character ram counter
patmlt  equ     0fdh    ;pattern multiplier register
patreg  equ     0fbh    ;pattern data register
fgbg    equ     0f7h    ;foreground/background enable
alups   equ     0efh    ;alu function plane select register
colmap  equ     0dfh    ;color map
modreg  equ     0bfh    ;mode register
scrlmp  equ     07fh    ;scroll map register

dseg ends

cseg segment byte  public 'codesg'

extrn   num__planes:byte,gbmod:byte,nmredl:word,nmritl:word,gtemp:word,curl0:wor
d

extrn   gdc__not__busy:near,ialups:near,ifgbg:near,ginit:near
```

```
assume cs:cseg,ds:dseg,es:dseg,ss:nothing

public  redvid

redvid  proc    near

;redvid moves the information in the bitmap to a 64k chunk of memory in the
;motherboard's addressing space. this routine doesn't a a real legally defined
;64k area to store the data in. i just made up a fake segment i'm calling vidseg
;to use for storage.

;1)setup to enable reads. the graphics option has to disable writes in the
;alups, enable a plane to be read in the mode register and program the gdc to
;perform one plane's worth of reads. gdc programming consists of issuing a
;cursor command of 0, a mask of ffffh, a figs with a direction to the right
;and of an entire plane's worth of read operations and then finally the rdat
;command to start the read in motion. note that the gdc can't read in all
;8000h words of a high res plane but it doesn't matter because not all 8000h
;words of a high res plane has usefull information in it anyway.

        cld                     ;make the coming stosb instruction increment si.
        mov     bl,0fh          ;disable all writes.
        call    ialups          ;issue the new alups byte.
        mov     word ptr curl0,0        ;start at the top.

        mov     ax,7fffh        ;assume hires read.
        test    byte ptr gbmod,01       ;actually hires?
        jnz     rd1             ;jump if yes.
        mov     ax,4000h        ;lowres number of reads.
rd1:    mov     word ptr nmredl,ax

;blank the screen. this will let the gdc have 100% use of time to read the
;screen in.

        mov     al,s__off        ;blank command.
        out     57h,al

;setup to transfer data as it is being read from the screen into the fake
;vidsg data segement. vidseg is undefined as far as this example is concerned.
;you are going to have to set it up before this routine will work.

        mov     ax,vidsg        ;setup the es register to point to vidbuf.
        mov     es,ax
        xor     si,si           ;start at the beginning of the buffer.
        mov     cl,byte ptr num__planes ;init routine set this byte.
        xor     ch,ch           ;num__planes=2 or 4.

;top of the read a plane loops.

rd2:    push    cx                      ;save plane count.
        mov     al,modreg               ;address the mode register.
        out     53h,al
        mov     al,byte ptr num__planes  ;figure out which plane to read enable.
        sub     al,cl
        shl     al,1                    ;shift plane to enable bits over 2.
        shl     al,1
        mov     ah,byte ptr gbmod       ;fetch current mode byte. eliminate
        and     ah,0a1h                 ;graphics, plane to read, write enable.
        or      al,ah                   ;combine new mode with plane to read.
        out     51h,al                  ;assert new mode.

        mov     al,curs         ;position the gdc cursor to top left.
        out     57h,al
        xor     al,al
        out     56h,al
        out     56h,al
        mov     al,gmask        ;set all bits in gdc mask.
        out     57h,al
        mov     al,0ffh
        out     56h,al
        out     56h,al
        mov     al,figs         ;assert the figs command.
        out     57h,al
        mov     al,2            ;direction is to the right.
        out     56h,al
        mov     ax,word ptr nmredl      ;number of reads to do.
        out     56h,al
        mov     al,ah
        out     56h,al
        mov     al,rdat         ;start the read operation now.
        out     57h,al

        mov     cx,word ptr nmredl      ;read in the bytes as they are ready.
        shl     cx,1                    ;bytes=2*words read.
rd4:    in      al,gstat        ;byte ready to be read?
        test    al,1
        jz      rd5             ;jump if not.
        in      al,gread        ;read the byte.
        stosb                   ;stos is es:si auto inc.
        loop    rd4

;we’ve finnished reading all of the information we’re going to get out of that
;plane. if high res then inc si by a word because we were one word short of
;the entire 32k high res plane. recover the plane to read count and loop if not
;done.

        test    byte ptr gbmod,1        ;high res?\
        jz      rd5                     ;jump if not.
        stosw           ;dummy stos just to keep num reads=words per plane.
rd5:    pop     cx              ;transfer all of the planes.
        loop    rd2             ;loop if more planes to be read.
```

```
;we're done with the read. restore video refresh and set the high/mid res
;flag byte at the end of vidsg so that when it is written back
;into the video we do it in the proper resolution. i just arbitrarily decided to
;use the last byte in the vidsg buffer because it won't have any useful data
;there anyway. if i'd wanted to i could have found room for the colormap as
;well but since i always use the same colormap in a resolution anyway i didn't
;see much use for going to the extra trouble.

        mov     al,s__on         ;unblank the screen.
        out     57h,al
        test    byte ptr gbmod,1        ;high res?
        jnz     rd6                     ;jump if yes.
        xor     al,al           ;set last byte in vidsg=0 to indicate midres.
        jmp     rd7
rd6:    mov     al,0ffh         ;set last byte in vidsg=ff to indicate high res.
rd7:    mov     si,0ffffh       ;setup the resolution flag.
        stosb
        ret

redvid  endp

cseg ends

end
```

## 10.4  PIXEL WRITE AFTER A READ OPERATION

After a read operation has completed, the graphics option is temporarily unable to do a pixel write. (Word writes are not affected by preceding read operations.) However, the execution of a word write operation restores the option's ability to do pixel writes. Therefore, whenever you intend to do a pixel write after a read operation, you must first execute a word write. This will ensure that subsequent vectors, arcs, and pixels will be enabled.

The following code sequence will execute a word write operation that will not write anything into the bitmap. The code assumes that the GDC is not busy since it just completed a read operation and that this code is entered after reading all the bytes that were required.

```
        mov     al,s_on    ;Sometimes the GDC will not accept the
        out     57h,al     ;first command after a read. This command
                           ;can safely be missed and serves to make sure
                           ;that the command FIFO is cleared and pointing
                           ;in the right direction.
        xor     bl,bl       ;Restore write enable replace mode to all
        call    ialups      ;planes in the ALU/PS Register.

        mov     al,0ffh     ;Disable writes to all bits at the
        out     55h,al      ;option’s Mask Registers.
        out     54h,al

        or      byte ptr gbmod,10h   ;Enable writes at the Mode Register;
        call    imode                ;it is already in word mode.

        mov     al,figs     ;Not necessary to assert cursor or mask. It does
        out     57h,al      ;matter where you write since the write is going
        xor     al,al       ;to be completely disabled anyway. Just going
        out     56h,al      ;through the word write operation will enable
        out     56h,al      ;subsequent pixel writes.
        out     56h,al
        mov     al,22h
        out     57h,al      ;Execute the write operation

        ret                 ;exit at this point back to calling routine.....
```

# CHAPTER 11

# SCROLL OPERATIONS

## 11.1 VERTICAL SCROLLING

The Scroll map controls the location of 64-word chunks of video memory on the video monitor. In medium resolution mode, this is two scan lines. In high resolution mode, this is one scan line. By redefining scan line locations in the Scroll Map, you effectively move 64 words of data into new screen locations.

All Scroll Map operations by the CPU start at location zero and increment by one with each succeeding CPU access. The CPU has no direct control over which Scroll Map location it is reading or writing. All input addresses are generated by an eight-bit index counter which is cleared to zero when the CPU first accesses the Scroll Map through the Indirect Register. There is no random access of a Scroll Map address.

Programming the Scroll Map involves a number of steps. First ensure that the GDC is not currently accessing the Scroll Map and that it won't be for some time (the beginning of a vertical retrace for example). Clearing bit 5 of the Mode Register to zero enables the Scroll Map for writing. Clearing bit 7 of the Indirect Register to zero selects the Scroll Map and clears the Scroll Map Counter to zero. Data can then be entered into the Scroll Map by writing to port 51h. When the programming operation is complete or just before the end of the vertical retrace period (whichever comes first) control of the Scroll Map addressing is returned to the GDC by setting bit 5 of the Mode Register to one.

If, for some reason, programming the Scroll Map requires more than one vertical retrace period, there is a way to break the operation up into two segments. A read of the Scroll Map increments the Scroll Map Index Counter just as though it were a write. You can therefore program the first half, wait for the next vertical retrace, read the first half and then finish the write of the last half.

### 11.1.1 Example Of Vertical Scrolling One Scan Line

```
        title scroll.asm
        subttl  vscroll.asm
        page 132,60

;*******************************************************************************
;                                                                              *
;                                                                              *
;                                                                              *
;               p r o c e e d u r e   v s c r o l l                            *
;                                                                              *
;       move the current entire screen up a scan line.                         *
;                                                                              *
;                                                                              *
;                                                                              *
;*******************************************************************************

extrn   scrltb:byte,gtemp1:byte,startl:byte,gbmod:byte

extrn   ascrol:near

dseg    segment byte public 'datasg'

pram    equ     70h     ;gdc parameter command.

dseg    ends

cseg    segment byte public 'codesg'

assume  cs:cseg,ds:dseg,es:dseg,ss:nothing

public  vscroll

vscroll proc    near

;basic scrollmap principal- the scrollmap controls which 64 word video memory
;segment will be displayed on the video screen itself. scrollmap location 0
;will display on the top high resolution scan whatever 64 word segment has
;been loaded into it. if that data is a 0 then the first 64 words are accessed.
;if that data is a 10 then the 11th 64 word segment is displayed. by simply
;rewriting the order of 64 word segments in the scrollmap the order in which
;they are displayed is correspondingly altered. if the entire screen is to be
;scrolled up one line then the entire scrollmap's contents are moved up one
;location. address one is moved into address zero, two goes into one and so on.
;a split screen scroll could be accomplished by keeping the stationary part of
;the screen unchanged in the scrollmap while the moving window gets loaded with
;the appropriate information. if more than one scrollmap location is loaded
```

```
;with the same data then the corresponding scan will be displayed multiple times
;on the screen.

;note that the information in the bitmap hasn’t been changed. only the location

;of where the information is displayed on the video monitor has been changed.
;when the bottom lines that used to be off the bottom of the screen scroll up
;and become visible they will have in them what ever was written there before.
;if a guaranteed clear scan line is desirable then before the scroll takes place
;the off the screen lines should be cleared with a write.

;the scrollmap also applies to gdc write operations. if the gdc is programmed
;to perform a write but the scrollmap is altered before the write takes place
;then the write will happen in the new area, not to the memory that was swapped
;to a new location.

;in mid res only the first 128 scrollmap entries have meaning because each mid
;res scan is 32 words long but each scrollmap entry controls the location on
;the screen of a 64 word long line. in mid res this is the same as two entire
;scans. the scrollmap acts as if the msb of the scrollmap entries was always a
;0. loading an 80h into a location is the same as loading a 0. loading an 81h
;is the equivilent to writing a 1. the below example assumes a high res 256
;location scrollmap. had it been mid res then only the first 128 scans would
;have been moved. the other 128 scrollmap locations still exist but are of no
;practical use to the programmer. what this means to the applications
;programmer is that in mid res after the scrollmap has been initialized the
;first 128 entries are treated as if they were the only scrollmap locations in
;the table instead of the 256 that high res has.

;assume that es and ds are already setup to point to the data seg where the
;graphics varibles and data are stored.

;save the contents of the first section of the scrolltable to be
;overwritten, fetch the data from however many scans away we want to scroll by
;and then move in a circular fashion the contents of the table. the last entry
;to be written is the scan we first saved. after the shadow scrolltable has
;been updated it will then be asserted by the call to initterm’s ascrol
;routine.

        mov     si,offset scrltb        ;setup the source of the data.
        mov     di,si                   ;setup the destination of the data.
        lodsb           ;fetch and save the first scan from being overwritten.
        mov     byte ptr gtemp1,al
        mov     cx,255                  ;move the other 255 scroll table bytes.
        rep     movsw
        mov     al,byte ptr gtemp1      ;recover what used to be the first scan.
        stosb                           ;put into scan 256 location.
        call    ascrol                  ;assert new scrolltable to scrollmap.
        ret

vscroll endp
```

```
cseg    ends

end
```

## 11.2 HORIZONTAL SCROLLING

Not only can the video display be scrolled up and down but it can also be scrolled from side to side as well. The GDC can be programmed to start video action at an address other than location 0000. Using the PRAM command to specify the starting address of the display partition as 0002 will effectively shift the screen two words to the left. Since the screen display width is not the same as the number of words displayed on the line there is a section of memory that is unrefreshed. The data that scrolls off the screen leaves the refresh area and it will also be unrefreshed. To have the data rotate or wrap around the screen and be saved requires that data be read from the side about to go off the screen and be written to the side coming on to the screen. If the application is not rotating but simply moving old data out to make room for new information, the old image can be allowed to disappear into the unrefreshed area.

Although the specifications for the dynamic RAMs only guarantee a data persistence of 2 milliseconds, most of the chips will hold data much longer. Therefore, it is possible to completely rotate video memory off one side and back onto the other. However, applications considering using this characteristic should be aware of the time dependency and plan accordingly.

### 11.2.1 Example Of Horizontal Scrolling One Word

```
        title scroll.asm

extrn   scrltb:byte,gtemp1:byte,startl:byte,gbmod:byte

dseg    segment byte public ’datasg’

pram    equ     70h     ;gdc parameter command.

dseg    ends

cseg    segment byte public ’codesg’

assume  cs:cseg,ds:dseg,es:dseg,ss:nothing
```

```
        subttl  hscroll.asm
        page

;******************************************************************************
;                                                                             *
;                                                                             *
;                                                                             *
;               p r o c e d u r e   h s c r o l l                             *
;                                                                             *
;       move the current entire screen to right or left a word address.       *
;                                                                             *
;       entry:  if al=0 then move screen left.                                *
;               if al<>0 then move screen right.                              *
;                                                                             *
;******************************************************************************

;the gdc is programmable on a word boundary as to where it starts displaying
;the screen. by incing or decing that starting address word we can redefine
;the starting address of each scan line and thereby give the appearence of
;horizontal scrolling. assume that this start window display address is stored
;in initterm's varible startl and starth. let's further assume that we want
;to limit scrolling to one scan line's worth so in high res we can never
;issue a starting address higher than 63 and in mid res higher than 31.

public  hscroll

hscroll proc    near

        or      al,al   ;move screen to left?
        jz      hs1     ;jump if not.
        dec     byte ptr startl ;move screen to right.
        jmp     hs2
hs1:    inc     byte ptr startl ;move screen to left.
hs2:    test    byte ptr gbmod,1        ;high res?
        jnz     hs3                     ;jump if yes.
        and     byte ptr startl,31      ;limit rotate to first mid res scan.
        jmp     hs4
hs3:    and     byte ptr startl,63      ;limit rotate to first high res scan.

;assert the new startl, starth to the gdc. assume that starth is always going to
;be 0 although this is not a necessity. issue the pram command and rewrite the
;starting address of the gdc display window 0 to whatever startl,starth now is.

hs4:    mov     al,pram         ;issue the gdc parameter command.
        out     57h,al
        mov     al,byte ptr startl      ;fetch low byte of the starting address.
        out     56h,al
        xor     al,al                   ;assume that high byte is always 0.
        out     56h,al
        ret
```

```
hscroll endp
cseg    ends

end
```

# CHAPTER 12

# PROGRAMMING NOTES

## 12.1 SHADOW AREAS

Most of the registers in the Graphics Option control more than one function. In addition, the registers are write-only areas. In order to change selected bits in a register while retaining the settings of the rest, shadow images of these registers should be kept in main storage. The current contents of the registers can be determined from the shadow area, selected bits can be set or reset by ORing or ANDing into the shadow area, and the result can be written over the existing register.

Modifying the Color Map and the Scroll Map is also made easier using a shadow area in main storage. These are relatively large areas and must be loaded during the time that the screen is inactive. It is more efficient to modify a shadow area in main storage and then use a fast move routine to load the shadow area into the Map during some period of screen inactivity such as a vertical retrace.

## 12.2 BITMAP REFRESH

The Graphics Option uses the same memory accesses that fill the screen with data to also refresh the memory. This means that if the screen display stops, the dynamic video memory will lose all the data that was being displayed within two milliseconds. In high resolution, it takes two scan lines to refresh the memory (approximately 125 microseconds). In medium resolution, it takes four scan lines to refresh the memory (approximately 250 microseconds). During vertical retrace (1.6 milliseconds) and horizontal retrace (10 microseconds) there is no refreshing of the memory. Under a worst case condition, you can stop the display for no more than two milliseconds minus four medium resolution scans minus vertical retrace or just about 150 microseconds. This is particularly important when programming the Scroll Map.

All write and read operations should take place during retrace time. Failure to limit reads and writes to retrace time will result in interference with the systematic refreshing of the dynamic RAMs as well as not displaying bitmap data during the read and write time. However, the GDC can be programmed to limit its bitmap accesses to retrace time as part of the initialization process.

## 12.3 SOFTWARE RESET

Whenever you reset the GDC by issuing the RESET command (a write of zero to port 57h), the Graphics Option must also be reset (a write of any data to port 50h). This is to synchronize the memory operations of the Graphics Option with the read/modify/write operations generated by the GDC. A reset of the Graphics Option by itself does not reset the GDC; they are separate reset operations.

## 12.4 SETTING UP CLOCK INTERRUPTS

With the Graphics Option installed on a Rainbow system, there are two 60 hz clocks available to the programmer - one from the motherboard and one from the Graphics Option. The motherboard clock is primarily used for a number of system purposes. However, you can intercept it providing that any routine that is inserted be kept short and compatible with the interrupt handler.

The following routine inserts a new interrupt vector:

```
        mov     ax,0            ;set ES to point to segment 0.
        mov     es,ax
        mov     si,80h          ;interrupt offset stored at 80h.
        mov     ax,es:[si]      ;fetch vector offset.
        mov     intoff,ax       ;store vector offset.
        mov     ax,newint       ;insert new vector offset.
        cli                     ;disable the interrupts temporarily.
        mov     es:[si],ax
        inc     si              ;vector segment address is at 82h.
        inc     si
        mov     ax,es:[si]      ;fetch it.
        mov     intoff+2,ax     ;store it.
        mov     ax,cs           ;move code segment into int. vector.
        mov     es:[si],ax      ;insert new int. segment into vector.
        sti                     ;re-enable interrupts.
```

The new interrupt handler will look something like this:

```
        intcode:code
                .
                .
                .
                more code
                .
                .
                db      0EAh            ;hex code for far jump.
        intoff  dw                      ;offset address.
                dw                      ;segment address.
```

The new interrupt handler intercepts each 60 hz motherboard interrupt, performs its function, and jumps far to the previous interrupt address. It is suggested that the program exit routine automatically restore the previous interrupt vectors when leaving the program.

Programming an interrupt using the Graphics Option's clock is less complicated since there is no system dependency on it. The offset address is at location 88h and the segment address is at location 8Ah. Load the address and segment of the routine, enable the option interrupts using bit 6 of the Mode Register, and let the interrupt terminate with an IRET.

It is important to keep all interrupt handlers short! Failure to do so can cause a system reset when the motherboard's MHFU line goes active. New interrupt handlers should restore any registers that are altered by the routine.

## 12.5 OPERATIONAL REQUIREMENTS

All data modifications to the bitmap are performed by hardware that is external to the GDC. In this environment, it is a requirement that the GDC be kept in graphics mode and be programmed to write in Replace mode. Also, the internal write data patterns of the GDC must be kept as all ones for the external hardware to function correctly. The external hardware isolates the GDC from the data in the bitmap such that the GDC is not aware of multiple planes or incoming data patterns.

Although it is possible to use the GDC's internal parameter RAM for soft character fonts and graphics characters, it is faster to use the option's Write Buffer. However, to operate in the GDC's native mode, the Write Buffer and Pattern Generator should be loaded with all ones, the Mode Register should be set to graphics mode, and the Foreground/Background Register should be loaded with F0h.

When the Graphics Option is in Word Mode, the GDC's mask register should be filled with all ones. This causes the GDC to go on to the next word after each pixel operation is done. The external hardware in the meantime, has taken care of all sixteen bits on all four planes while the GDC was taking care of only one pixel.

When the option is in Vector Mode, the GDC is also in graphics mode. The GDC's mask register is now set by the third byte of the cursor positioning command (CURS). The GDC will be able to tell the option which pixel to perform the write on but the option sets the mode, data and planes.

## 12.6 SET-UP MODE

When you press the SET-UP key on the keyboard, the system is placed in set-up mode. This, in turn, suspends any non-interrupt driven software and brings up a set-up screen if the monitor is displaying VT102 video output. If, however, the system is displaying graphics output, the fact that the system is in set-up mode will not be apparent to a user except for the lack of any further interaction with the graphics application that has been suspended. The set-up screen will not be displayed.

Users of applications that involve graphics output should be warned of this condition and cautioned not to press the SET-UP key when in graphics output mode. Note also that pressing the SET-UP key a second time will resume the execution of the suspended graphics software.

In either case, whether the set-up screen is displayed or not, set-up mode accepts any and all keyboard data until the SET-UP key is again pressed.

# PART III

# REFERENCE MATERIAL

# CHAPTER 13

# OPTION REGISTERS, BUFFERS, AND MAPS

The Graphics Option uses a number of registers, buffers, and maps to generate graphic images and control the display of these images on a monochrome or color monitor. Detailed discussions of these areas may be found in Chapter 3 of this manual.

## 13.1 I/O PORTS

The CPUs on the Rainbow system's motherboard use the following I/O ports to communicate with the Graphics Option:

| Port | Function |
|---|---|
| 50h | Graphics option software reset and resynchronization. |
| 51h | Data input to area selected through port 53h. |
| 52h | Data input to the Write Buffer. |
| 53h | Area select input to Indirect Register. |
| 54h | Input to low-order byte of Write Mask. |
| 55h | Input to high-order byte of Write Mask. |
| 56h | Parameter input to GDC - Status output from GDC. |
| 57h | Command input to GDC - Data output from GDC. |

## 13.2 INDIRECT REGISTER

The Indirect Register is used to select one of eight areas to be written into.

Load Data: Write data byte to port 53h.

```
                Indirect Register
  +-------------------------------+
  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
  +-------------------------------+
```

where:

| Data Byte | Active Bit | Function |
|---|---|---|
| FEh | 0 | selects the Write Buffer |
| FDh | 1 | selects the Pattern Multiplier. (Pattern Multiplier must always be loaded before the Pattern Register) |
| FBh | 2 | selects the Pattern Register. |
| F7h | 3 | selects the Foreground/Background Register. |
| EFh | 4 | selects the ALU/PS Register. |
| DFh | 5 | selects the Color Map and resets the Color Map Address Counter to zero. |
| BFh | 6 | selects the Graphics Option Mode Register. |
| 7Fh | 7 | selects the Scroll Map and resets the Scroll Map Address Counter to zero. |

NOTE

If more than one bit is set to zero, more than one area will be selected and the results of subsequent write operations will be unpredictable.

## 13.3 WRITE BUFFER

The Write Buffer is the incoming data source when the Graphics Option is in Word Mode.

Select Area: write FEh to port 53h

Clear Counter: write any value to port 51h

Load Data: write up to 16 bytes to port 52h

```
                 As the CPU sees it                As the GDC sees it

               (16 X 8-bit Ring Buffer)            (8 X 16-bit Words)

      byte   7         0   7         0   word   15                     0
            +------------+ +------------+       +---------------------------+
      0,1   |            | |            |   0   |                           |
            +------------+ +------------+       +---------------------------+
      2,3   |            | |            |   1   |                           |
            +------------+ +------------+       +---------------------------+
      4,5   |            | |            |   2   |                           |
            +------------+ +------------+       +---------------------------+
      6,7   |            | |            |   3   |                           |
            +------------+ +------------+       +---------------------------+
      8,9   |            | |            |   4   |                           |
            +------------+ +------------+       +---------------------------+
     10,11  |            | |            |   5   |                           |
            +------------+ +------------+       +---------------------------+
     12,13  |            | |            |   6   |                           |
            +------------+ +------------+       +---------------------------+
     14,15  |            | |            |   7   |                           |
            +------------+ +------------+       +---------------------------+
```

## 13.4 WRITE MASK REGISTERS

The Write Mask Registers control the writing of individual bits in a bitmap word.

Select Area: no selection required

Load Data: write low-order data byte to port 54h
write high-order data byte to port 55h

```
                        As seen by
                         the CPU
              Port 55h             Port 54h
                 |                    |
                 V                    V
          7-----------------0 7-----------------0

          +--------------------+-------------------+
          | Write Mask (high) | Write Mask (low)  |
          +-----------------------------------------+

          15--------------------------------------0

                    Word As Seen By GDC
```

where:

| | |
|---|---|
| bit = 0 | enables a write in the corresponding bit position of the word being displayed. |
| bit = 1 | disables a write in the corresponding bit position of the word being displayed. |

## 13.5 PATTERN REGISTER

The Pattern Register provides the incoming data when the Graphics Option is in Vector Mode.

Select Area: write FBh to port 53h

Load Data: write data byte to port 51h

```
               7                             0
              +-------------------------------+    Bitmap
           :->|   P   a   t   t   e   r   n   |--> Write
           :  +----------------------------v--+    Circuitry
           :                               :
           :...............................:
```

where:

Pattern    is the basic pixel configuration to be drawn by the option when in Vector Mode.

## 13.6  PATTERN MULTIPLIER

The Pattern Multiplier controls the recirculating frequency of the bits in the Pattern Register.

Select Area:  write FDh to port 53h

Load Data:  write data byte to port 51h

```
  7           4   3           0
+--------------------------------+
|   unused      |     value      |
+--------------------------------+
```

where:

| | |
|---|---|
| value | is a number in the range of 0 through 15 such that 16 minus this value is the factor that determines when the Pattern Register is shifted. |

## 13.7 FOREGROUND/BACKGROUND REGISTER

The Foreground/Background Register controls the bit/plane input to the bitmap.

Select Area: write F7h to port 53h

Load Data: write data byte to port 51h

```
  7         data byte         0
+-------------------------------+
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
+-------------------------------+
   Foreground   |  Background
    Register    |   Register
```

where:

| Bits | |
|---|---|
| 0-3 | are the bits written to bitmap planes 0-3 respectively when the option is in REPLACE mode and the incoming data bit is a zero.<br><br>If the option is in OVERLAY or COMPLEMENT mode and the incoming data bit is a zero, there is no change to the bitmap value. |
| 4-7 | are the bits written to bitmap planes 4-7 respectively when the option is in REPLACE or OVERLAY mode and the incoming data bit is a one.<br><br>If the option is in COMPLEMENT mode and the incoming data bit is a one, the Foreground bit determines the action. If it is a one, the bitmap value is inverted; if it is a zero, the bitmap value is unchanged. |

## 13.8 ALU/PS REGISTER

The ALU/PS Register controls the logic used in writing to the bitmap and the inhibiting of writing to specified planes.

Select Area: write EFh to port 53h

Load Data: write data byte to port 51h

```
  7         data byte         0
+-------------------------------+
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
+-------------------------------+
|unused |  ALU  | Plane Select  |
```

where:

| Bit | Value | Function |
|---|---|---|
| 0 | 0 | enable writes to plane 0 |
| | 1 | inhibit writes to plane 0 |
| 1 | 0 | enable writes to plane 1 |
| | 1 | inhibit writes to plane 1 |
| 2 | 0 | enable writes to plane 2 |
| | 1 | inhibit writes to plane 2 |
| 3 | 0 | enable writes to plane 3 |
| | 1 | inhibit writes to plane 3 |
| 5,4 | 00 | place option in REPLACE mode |
| | 01 | place option in COMPLEMENT mode |
| | 10 | place option in OVERLAY mode |
| | 11 | Unused |
| 7,6 | | Unused |

## 13.9 COLOR MAP

The Color Map translates bitmap data into the monochrome and color intensities that are applied to the video monitors.

Select Area:  write DFh to port 53h

Coordinate:  wait for vertical sync interrupt

Load Data:  write 32 bytes to port 51h

```
  2nd 16 bytes  | 1st 16 bytes
   as seen by   |  as seen by
     the CPU    |    the CPU
+--------------------------------+
| mono. |  blue |  red  |  green|
| data  |  data |  data |  data |
|-------------------------------|
|    byte 17    |    byte 1     |
|-------------------------------|
|    byte 18    |    byte 2     |
|-------------------------------|
|    byte 19    |    byte 3     |
|-------------------------------|
|    byte 20    |    byte 4     |
|-------------------------------|
|    byte 21    |    byte 5     |
|-------------------------------|
|    byte 22    |    byte 6     |
|-------------------------------|
|    byte 23    |    byte 7     |
|-------------------------------|
|                               |
/                               /
/                               /
|                               |
|-------------------------------|
|    byte 32    |    byte 16    |
+--------------------------------+
```

## 13.10 MODE REGISTER

The Mode Register controls a number of the Graphics Option's operating characteristics.

Select Area: write BFh to port 53h

Load Data: write data byte to port 51h

```
+-------------------------------+
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
+-------------------------------+
```

where:

| Bit | Value | Function |
|---|---|---|
| 0 | 0 | place option in medium resolution mode |
| | 1 | place option in high resolution mode |
| 1 | 0 | place option into word mode |
| | 1 | place option into vector mode |
| 3,2 | 00 | select plane 0 for readback operation |
| | 01 | select plane 1 for readback operation |
| | 10 | select plane 2 for readback operation |
| | 11 | select plane 3 for readback operation |
| 4 | 0 | enable readback operation |
| | 1 | enable write operation |
| 5 | 0 | enable writing to the Scroll Map |
| | 1 | disable writing to the Scroll Map |
| 6 | 0 | disable vertical sync interrupts to CPU |
| | 1 | enable vertical sync interrupts to CPU |
| 7 | 0 | disable video output from Graphics Option |
| | 1 | enable video output from Graphics Option |

NOTE

The Mode Register must be reloaded following any write to port 50h (software reset).

## 13.11 SCROLL MAP

The Scroll Map controls the location of each line displayed on the monitor screen.

Preliminary: enable Scroll Map writing (Mode Register bit 5 = 0)

Select Area: write 7Fh to port 53h

Coordinate: wait for vertical sync interrupt

Load Data: write 256 bytes to port 51h

Final: disable Scroll Map writing (Mode Register bit 5 = 1)

```
                             256 X 8
                          Recirculating
         GDC            7  Ring Buffer  0
          |            +------------------+
          |           0|                  |
          |            |                  |
      GDC Line         |                  |
       Address         |                  |
     (Bits 6-13)       |                  |
          |            |                  |
          |            |                  |
          +----------> | x x x x x x x x  |
                       |         .        |
                       |         .        |
                       |         .        |
                       |         .        |
                       |         .        |
                       |         .        |
                    255|         .        |  Bitmap Line
                       +------------------+    Address
                                 |           (Bits 6-13)
                                 +------------------------> Bitmap
```

where:

| | |
|---|---|
| GDC Line Address | is the line address as generated by the GDC and used as an index into the Scroll Map. |
| Bitmap Line Address | is the offset line address found by indexing into the Scroll Map. It becomes the new line address of data going into the bitmap. |

# CHAPTER 14

# GDC REGISTERS AND BUFFERS

The GDC has an 8-bit Status Register and a 16 X 9-bit first-in, first-out (FIFO) Buffer that provide the interface to the Graphics Option. The Status Register supplies information on the current activity of the GDC and the status of the FIFO Buffer. The FIFO Buffer contains GDC commands and parameters when the GDC is in write mode. It contains bitmap data when the GDC is in read mode.

## 14.1 STATUS REGISTER

The GDC's internal status can be interrogated by doing a read from port 56h. The Status Register contents are as follows:

```
+-------------------------------+
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
+-------------------------------+
```

where:

| Bit | Status | Explanation |
|---|---|---|
| 0 | DATA READY | When set, data is ready to be read from the FIFO. |
| 1 | FIFO FULL | When set, the command/parameter FIFO is full. |
| 2 | FIFO EMPTY | When set, the command/parameter FIFO is completely empty. |
| 3 | DRAWING IN PROGRESS | When set, the GDC is performing a drawing function. Note, however, that this bit can be cleared before the DRAW command is fully |

| | | |
|---|---|---|
| | | completed. The GDC does not draw continuously and this bit is reset during interrupts to the write operation. |
| 4 | DMA EXECUTE | Not used. |
| 5 | VERTICAL SYNC ACTIVE | When set, the GDC is doing a vertical sync. |
| 6 | HORIZONTAL SYNC ACTIVE | When set, the GDC is doing a horizontal sync. |
| 7 | LIGHT PEN DETECTED | Not used. |

## 14.2 FIFO BUFFER

You can both read from and write to the FIFO Buffer. The direction that the data takes through the buffer is controlled by the Rainbow system using GDC commands. GDC commands and their associated parameters are written to ports 57h and 56h respectively. The GDC stores both in the FIFO Buffer where they are picked up by the GDC command processor. The GDC uses the ninth bit in the FIFO Buffer as a flag bit to allow the command processor to distinguish between commands and parameters. (See Figure 13.) Contents of the bitmap are read from the FIFO using reads from port 57h.

```
                        flg      data byte
                         8   7                 0
                       +----+--------------------+
  Commands and       0  |   :                    |
  parameters            |------------------------|
  from the CPU ====> 1  |   :                    |  ====> Commands and
                        |------------------------|        parameters to
                     2  |   :                    |        the command
                        |------------------------|        processor
                     3  |   :                    |
                        |------------------------|        Data from
  Bitmap data <====     /                        /  <==== the bitmap
  to the CPU            /                        /
                        |------------------------|
                    14  |   :                    |
                        |------------------------|
                    15  |   :                    |
                            +-------------------------+
```

where:

|  |  |
|---|---|
| flg | is a flag bit to be interpreted as:<br><br>0 - data byte is a parameter<br>1 - data byte is a command |
| data byte | is a GDC command or parameter |

Figure 13. FIFO Buffer

When you reverse the direction of flow in the FIFO Buffer, any pending data in the FIFO is lost. If a read operation is in progress and a command is written to port 56h, the unread data still in the FIFO is lost. If a write operation is in progress and a read command is processed, any unprocessed commands and parameters in the FIFO Buffer are lost.

# CHAPTER 15

# GDC COMMANDS

## 15.1 INTRODUCTION

This chapter contains detailed reference information on the GDC commands and parameters supported by the Graphics Option. The commands are listed in alphabetical order within functional category as follows:

- Video Control Commands

| | |
|---|---|
| CCHAR | Specifies the cursor and character row heights |
| RESET | Resets the GDC to its idle state |
| SYNC | Specifies the video display format |
| VSYNC | Selects Master/Slave video synchronization mode |

- Display Control Commands

| | |
|---|---|
| BCTRL | Controls the blanking/unblanking of the display |
| CURS | Sets the position of the cursor in display memory |
| PITCH | Specifies the width of display memory |
| PRAM | Defines the display area parameters |
| START | Ends idle mode and unblanks the display |
| ZOOM | Specifies zoom factor for the graphics display |

- Drawing Control Commands

| | |
|---|---|
| FIGD | Draws the figure as specified by FIGS command |
| FIGS | Specifies the drawing controller parameters |
| GCHRD | Draws the graphics character into display memory |
| MASK | Sets the mask register contents |
| WDAT | Writes data words or bytes into display memory |

- Data Read Commands

| | |
|---|---|
| RDAT | Reads data words or bytes from display memory |

## 15.2 VIDEO CONTROL COMMANDS

### 15.2.1 CCHAR - Specify Cursor And Character Characteristics

Use the CCHAR command to specify the cursor and character row heights and characteristics.

Command Byte

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
+---+---+---+---+---+---+---+---+
```

Parameter Bytes

```
           7   6   5   4   3   2   1   0
         +---+---+---+---+---+---+---+---+
    P1   |DC | 0   0 |        LR         |
         +---+---+---+---+---+---+---+---+
    P2   | BR(lo)|SC |      CTOP         |
         +---+---+---+---+---+---+---+---+
    P3   |       CBOT        |  BR(hi)   |
         +---+---+---+---+---+---+---+---+
```

where:

| | |
|---|---|
| DC | controls the display of the cursor<br>0 - do not display cursor<br>1 - display the cursor |
| LR | is the number of lines per character row, minus 1 |
| BR | is the blink rate (5 bits) |
| SC | controls the action of the cursor<br>0 - blinking cursor<br>1 - steady cursor |
| CTOP | is the cursor's top line number in the row |
| CBOT | is the cursor's bottom line number in the row (CBOT must be less than LR) |

## 15.2.2 RESET - Reset The GDC

Use the RESET command to reset the GDC. This command blanks the display, places the GDC in idle mode, and initializes the FIFO buffer, command processor, and the internal counters. If parameter bytes are present, they are loaded into the sync generator.

Command Byte

```
       7   6   5   4   3   2   1   0
     +---+---+---+---+---+---+---+---+
     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
     +---+---+---+---+---+---+---+---+
```

Parameter Bytes

```
       7   6   5   4   3   2   1   0
     +---+---+---+---+---+---+---+---+
P1   | 0 | 0 | C | F | I | D | G | S |
     +---+---+---+---+---+---+---+---+
P2   |              AW               |
     +---+---+---+---+---+---+---+---+
P3   |  VS(lo)   |        HS         |
     +---+---+---+---+---+---+---+---+
P4   |          HFP          |VS(hi) |
     +---+---+---+---+---+---+---+---+
P5   | 0 | 0 |          HBP          |
     +---+---+---+---+---+---+---+---+
P6   | 0 | 0 |          VFP          |
     +---+---+---+---+---+---+---+---+
P7   |            AL(lo)             |
     +---+---+---+---+---+---+---+---+
P8   |          VBP          |AL(hi) |
     +---+---+---+---+---+---+---+---+
```

where:

| | |
|---|---|
| CG | sets the display mode for the GDC |
| | 00 - mixed graphics and character mode<br>01 - graphics mode only<br>10 - character mode only<br>11 - invalid |
| IS | controls the video framing for the GDC |
| | 00 - noninterlaced<br>01 - invalid |

|  |  |
|---|---|
|  | 10 − interlaced repeat field for character displays<br>11 − interlaced |
| D | controls the RAM refresh cycles<br><br>0 − no refresh − static RAM<br>1 − refresh − dynamic RAM |
| F | controls the drawing time window<br><br>0 − drawing during active display time and retrace blanking<br>1 − drawing only during retrace blanking |
| AW | active display words per line minus 2; must be an even number |
| HS | horizontal sync width minus 1 |
| VS | vertical sync width |
| HFP | horizontal front porch width minus 1 |
| HBP | horizontal back porch width minus 1 |
| VFP | vertical front porch width |
| AL | active display lines per video field |
| VBP | vertical back porch width |

## 15.2.3  SYNC - Sync Format Specify

Use the SYNC command to load parameters into the sync generator. The GDC is neither reset nor placed in idle mode.

Command Byte

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |DE |
+---+---+---+---+---+---+---+---+
```

where:

```
DE     controls the display

       0 - disables (blanks) the display
       1 - enables the display
```

Parameter Bytes

```
        7   6   5   4   3   2   1   0
      +---+---+---+---+---+---+---+---+
P1    | 0 | 0 | C | F | I | D | G | S |
      +---+---+---+---+---+---+---+---+
P2    |              AW               |
      +---+---+---+---+---+---+---+---+
P3    |  VS(lo)   |        HS         |
      +---+---+---+---+---+---+---+---+
P4    |          HFP          |VS(hi) |
      +---+---+---+---+---+---+---+---+
P5    | 0 | 0 |         HBP           |
      +---+---+---+---+---+---+---+---+
P6    | 0 | 0 |         VFP           |
      +---+---+---+---+---+---+---+---+
P7    |             AL(lo)            |
      +---+---+---+---+---+---+---+---+
P8    |          VBP          |AL(hi) |
      +---+---+---+---+---+---+---+---+
```

where:

```
CG     sets the display mode for the GDC

       00 - mixed graphics and character mode
       01 - graphics mode only
       10 - character mode only
       11 - invalid
```

| | |
|---|---|
| IS | controls the video framing for the GDC<br><br>00 − noninterlaced<br>01 − invalid<br>10 − interlaced repeat field for character displays<br>11 − interlaced |
| D | controls the RAM refresh cycles<br><br>0 − no refresh − static RAM<br>1 − refresh − dynamic RAM |
| F | controls the drawing time window<br><br>0 − drawing during active display time and retrace blanking<br>1 − drawing only during retrace blanking |
| AW | active display words per line minus 2; must be an even number |
| HS | horizontal sync width minus 1 |
| VS | vertical sync width |
| HFP | horizontal front porch width minus 1 |
| HBP | horizontal back porch width minus 1 |
| VFP | vertical front porch width |
| AL | active display lines per video field |
| VBP | vertical back porch width |

## 15.2.4 VSYNC − Vertical Sync Mode

Use the VSYNC command to control the slave/master relationship whenever multiple GDC’s are used to contribute to a single image.

Command Byte

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | M |
+---+---+---+---+---+---+---+---+
```

where:

```
       M      sets the synchronization status of the GDC

              0 − slave mode (accept external vertical sync pulses)
              1 − master mode (generate and output vertical sync pulses)
```

## 15.3 DISPLAY CONTROL COMMANDS

### 15.3.1 BCTRL - Control Display Blanking

Use the BCTRL command to specify whether the display is blanked or enabled.

Command Byte

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 | 1 | 1 | 0 |DE |
+---+---+---+---+---+---+---+---+
```

where:

DE     controls the display

0 - disables (blanks) the display
1 - enables the display

## 15.3.2 CURS - Specify Cursor Position

Use the CURS command to set the position of the cursor in display memory. In character mode the cursor is displayed for the length of the word. In graphics mode the word address specifies the word that contains the starting pixel of the drawing; the dot address specifies the pixel within that word.

Command Byte

```
                7   6   5   4   3   2   1   0
              +---+---+---+---+---+---+---+---+
              | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
              +---+---+---+---+---+---+---+---+
```

Parameter Bytes

```
                7   6   5   4   3   2   1   0
              +---+---+---+---+---+---+---+---+
         P1   |            EAD(lo)            |
              +---+---+---+---+---+---+---+---+
         P2   |            EAD(mid)           |
              +---+---+---+---+---+---+---+---+

              +---+---+---+---+---+---+---+---+
         P3   |      dAD      | 0   0 |EAD(hi)| <-- Graphics Mode Only
              +---+---+---+---+---+---+---+---+
```

where:

| | |
|---|---|
| EAD | is the execute word address (18 bits) |
| dAD | is the dot address within the word |

## 15.3.3  PITCH - Specify Horizontal Pitch

Use the PITCH command to set the width of the display memory. The drawing processor uses this value to locate the word directly above or below the current word. It is also used during display to find the start of the next line.

Command Byte

```
                 7   6   5   4   3   2   1   0
               +---+---+---+---+---+---+---+---+
               | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
               +---+---+---+---+---+---+---+---+
```

Parameter Bytes

```
                 7   6   5   4   3   2   1   0
               +---+---+---+---+---+---+---+---+
          P1   |               P               |
               +---+---+---+---+---+---+---+---+
```

where:

| | |
|---|---|
| P | is the number of word addresses in display memory in the horizontal direction |

### 15.3.4 PRAM - Load The Parameter RAM

Use the PRAM command to load up to 16 bytes of information into the parameter RAM at specified adjacent locations. There is no count of the number of parameter bytes to be loaded; the sensing of the next command byte stops the load operation. Because the Graphics Option requires that the GDC be kept in graphics mode, only parameter bytes one through four, nine, and ten are used.

Command Byte

```
                7   6   5   4   3   2   1   0
              +---+---+---+---+---+---+---+---+
              | 0 | 1 | 1 | 1 |      SA       |
              +---+---+---+---+---+---+---+---+
```

where:

```
        SA     is the start address for the load operation (Pn - 1)
```

Parameter Bytes

```
                7   6   5   4   3   2   1   0
              +---+---+---+---+---+---+---+---+
         P1   |            SAD(lo)            |
              +---+---+---+---+---+---+---+---+
         P2   |            SAD(mid)           |
              +---+---+---+---+---+---+---+---+
         P3   |    LEN(lo)    | 0   0 |SAD(hi)|
              +---+---+---+---+---+---+---+---+
         P4   |WD |IM |       LEN(hi)         |
              +---+---+---+---+---+---+---+---+

              +---+---+---+---+---+---+---+---+
         P5   |   | u | n | u | s | e | d |   |
          .   +---+---+---+---+---+---+---+---+
          .
          .   +---+---+---+---+---+---+---+---+
         P8   |   | u | n | u | s | e | d |   |
              +---+---+---+---+---+---+---+---+

              +---+---+---+---+---+---+---+---+
         P9   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
              +---+---+---+---+---+---+---+---+
         P10  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
              +---+---+---+---+---+---+---+---+

              +---+---+---+---+---+---+---+---+
         P11  |   | u | n | u | s | e | d |   |
          .   +---+---+---+---+---+---+---+---+
          .
          .   +---+---+---+---+---+---+---+---+
         P16  |   | u | n | u | s | e | d |   |
              +---+---+---+---+---+---+---+---+
```

where:

| | |
|---|---|
| SAD | is the start address of the display area (18 bits) |
| LEN | is the number of lines in the display area (10 bits) |
| WD | sets the display width<br><br>0 − one word per memory cycle (16 bits)<br>1 − two words per memory cycle (8 bits) |
| IM | sets the current type of display when the GDC is in mixed graphics and character mode<br><br>0 − character area<br>1 − image or graphics area |

NOTE

When the GDC is in graphics mode, the IM bit must be a zero.

## 15.3.5 START − Start Display And End Idle Mode

Use the START command to end idle mode and enable the video display.

Command Byte

```
   7   6   5   4   3   2   1   0
 +---+---+---+---+---+---+---+---+
 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
 +---+---+---+---+---+---+---+---+
```

## 15.3.6 ZOOM - Specify The Zoom Factor

Use the ZOOM command to set up a magnification factor of 1 through 16 (using codes 0 through 15) for the display and for graphics character writing.

Command Byte

```
                7   6   5   4   3   2   1   0
              +---+---+---+---+---+---+---+---+
              | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
              +---+---+---+---+---+---+---+---+
```

Parameter Bytes

```
                7   6   5   4   3   2   1   0
              +---+---+---+---+---+---+---+---+
         P1   |     DISP      |     GCHR      |
              +---+---+---+---+---+---+---+---+
```

where:

| | |
|---|---|
| DISP | is the zoom factor (minus one) for the display |
| GCHR | is the zoom factor (minus one) for graphics character writing and area fills |

## 15.4 DRAWING CONTROL COMMANDS

### 15.4.1 FIGD - Start Figure Drawing

Use the FIGD command to start drawing the figure specified with the FIGS command. This command causes the GDC to:

- o load the parameters from the parameter RAM into the drawing controller, and
- o start the drawing process at the pixel pointed to by the cursor: Execute Word Address (EAD) and Dot Address within the word (dAD)

Command Byte

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
+---+---+---+---+---+---+---+---+
```

## 15.4.2  FIGS - Specify Figure Drawing Parameters

Use the FIGS command to supply the drawing controller with the necessary figure type, direction, and drawing parameters needed to draw figures into display memory.

Command Byte

```
                7   6   5   4   3   2   1   0
              +---+---+---+---+---+---+---+---+
              | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
              +---+---+---+---+---+---+---+---+
```

Parameter Bytes

```
                7   6   5   4   3   2   1   0
              +---+---+---+---+---+---+---+---+
         P1   |SL | R | A |GC | L |    DIR    |
              +---+---+---+---+---+---+---+---+
         P2   |            DC(lo)             |
              +---+---+---+---+---+---+---+---+
         P3   | 0 |GD |        DC(hi)         |
              +---+---+---+---+---+---+---+---+
         P4   |            D(lo)              |
              +---+---+---+---+---+---+---+---+
         P5   | 0   0 |        D(hi)          |
              +---+---+---+---+---+---+---+---+
         P6   |            D2(lo)             |
              +---+---+---+---+---+---+---+---+
         P7   | 0   0 |        D2(hi)         |
              +---+---+---+---+---+---+---+---+
         P8   |            D1(lo)             |
              +---+---+---+---+---+---+---+---+
         P9   | 0   0 |        D1(hi)         |
              +---+---+---+---+---+---+---+---+
         P10  |            DM(lo)             |
              +---+---+---+---+---+---+---+---+
         P11  | 0   0 |        DM(hi)         |
              +---+---+---+---+---+---+---+---+
```

where:

```
        SL     Slanted Graphics Character  \
                                            |
        R      Rectangle                    |  Figure Type Select Bits
                                            |
        A      Arc/Circle                    >  (see valid
                                            |    combinations
        GC     Graphics Character           |    below)
                                            |
```

| | |
|---|---|
| L | Line (Vector) / |
| DIR | is the drawing direction base (see definitions below) |
| DC | is the DC drawing parameter (14 bits) |
| GD | is the graphic drawing flag used in mixed graphics and character mode |
| D | is the D drawing parameter (14 bits) |
| D2 | is the D2 drawing parameter (14 bits) |
| D1 | is the D1 drawing parameter (14 bits) |
| DM | is the DM drawing parameter (14 bits) |

Figure Type Select Bits (valid combinations)

| SL R A GC L | Operation |
|---|---|
| 0 0 0 0 0 | Character Display Mode Drawing, Individual Dot Drawing, WDAT, and RDAT |
| 0 0 0 0 1 | Straight Line Drawing |
| 0 0 0 1 0 | Graphics Character Drawing and Area Fill with graphics character pattern |
| 0 0 1 0 0 | Arc and Circle Drawing |
| 0 1 0 0 0 | Rectangle Drawing |
| 1 0 0 1 0 | Slanted Graphics Character Drawing and Slanted Area Fill |

Drawing Direction Base (DIR)

```
          [101]        [100]        [011]
                +        +        +
                  +      +      +
                    +    +    +
                      +  +  +
          [110]+ + + +[start]+ + + +[010]
                      +  +  +
                  +     +     +
                +       +       +
              +         +         +
          [111]       [000]       [001]
```

## 15.4.3 GCHRD - Start Graphics Character Draw And Area Fill

Use the GCHRD command to initiate the drawing of the graphics character or area fill pattern that is stored in the Parameter RAM. The drawing is further controlled by the parameters loaded by the FIGS command. Drawing begins at the address in display memory pointed to by the Execute Address (EAD) and Dot Address (dAD) values.

Command Byte

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+
```

## 15.4.4 MASK - Load The Mask Register

Use the MASK command to set the value of the 16-bit Mask Register that controls which bits of a word can be modified during a Read/Modify/Write (RMW) cycle.

Command Byte

```
       7   6   5   4   3   2   1   0
     +---+---+---+---+---+---+---+---+
     | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
     +---+---+---+---+---+---+---+---+
```

Parameter Bytes

```
       7   6   5   4   3   2   1   0
     +---+---+---+---+---+---+---+---+
P1   |             M(lo)             |
     +---+---+---+---+---+---+---+---+
P2   |             M(hi)             |
     +---+---+---+---+---+---+---+---+
```

where:

M is the bit configuration to be loaded into the Mask Register (16 bits). Each bit in the Mask Register controls the writing of the corresponding bit in the word being processed as follows:

0 - disable writing
1 - enable writing

## 15.4.5 WDAT - Write Data Into Display Memory

Use the WDAT command to perform RMW cycles into video memory starting at the location pointed to by the cursor Execute Word Address (EAD). Precede this command with a FIGS command to supply the writing direction (DIR) and the number of transfers (DC).

Command Byte

```
                 7   6   5   4   3   2   1   0
               +---+---+---+---+---+---+---+---+
               | 0 | 0 | 1 | TYPE  | 0 |  MOD  |
               +---+---+---+---+---+---+---+---+
```

where:

```
        TYPE   is the type of transfer

               00 - word transfer (first low then high byte)
               01 - invalid
               10 - byte transfer (low byte of the word only)
               11 - byte transfer (high byte of the word only)

        MOD    is the RMW memory logical operation

               00 - REPLACE with Pattern
               01 - COMPLEMENT
               10 - RESET to Zero
               11 - SET to One
```

Parameter Bytes

```
                 7   6   5   4   3   2   1   0
               +---+---+---+---+---+---+---+---+
          P1   |       WORD(lo) or BYTE        |
               +---+---+---+---+---+---+---+---+

               +---+---+---+---+---+---+---+---+
          P2   |            WORD(hi)           |
          .    +---+---+---+---+---+---+---+---+
          .
          .
```

where:

```
        WORD   is a 16-bit data value

        BYTE   is an 8-bit data value
```

## 15.4.6 RDAT - Read Data From Display Memory

Use the RDAT command to read data from display memory and pass it through the FIFO buffer and microprocessor interface to the host system. Use the CURS command to set the starting address and the FIGS command to supply the direction (DIR) and the number of transfers(DC). The type of transfer is coded in the command itself.

Command Byte

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| 1 | 0 | 1 | TYPE  | 0 |  MOD  |
+---+---+---+---+---+---+---+---+
```

where:

```
TYPE   is the type of transfer

       00 - word transfer (first low then high byte)
       01 - invalid
       10 - byte transfer (low byte of the word only)
       11 - byte transfer (high byte of the word only)

MOD    is the RMW memory logical operation

       00 - REPLACE with Pattern
       01 - COMPLEMENT
       10 - RESET to Zero
       11 - SET to One
```

NOTE

The MOD field should be set to 00 if no modification to the video buffer is desired.

## PART IV

## APPENDIXES

# APPENDIX A

# OPTION SPECIFICATION SUMMARY

## A.1 PHYSICAL SPECIFICATIONS

The Graphics Option Video Subsystem is a 5.7" X 10.0", high density, four−layer PCB with one 40−pin female connector located on side 1. This connector plugs into a shrouded male connector located on the system module. The option module is also supported by two standoffs.

## A.2 ENVIRONMENTAL SPECIFICATIONS

### A.2.1 Temperature

- o Operating ambient temperature range is 10 to 40 degrees C.
- o Storage temperature is −40 to 70 degrees C.

### A.2.2 Humidity

- o 10% to 90% non−condensing
- o Maximum wet bulb, 28 degrees C.
- o Minimum dew point, 2 degrees C.

### A.2.3 Altitude

- o Derate maximum operating temperature 1 degree per 1,000 feet elevation
- o Operating limit: 22.2 in. Hg. (8,000 ft.)
- o Storage limit: 8.9 in Hg. (30,000 ft.)

## A.3 POWER REQUIREMENTS

| | Calculated Typical | Calculated Maximum |
|---|---|---|
| +5V DC (+/−5%) | 3.05 amps | 3.36 amps |
| +12V DC (+/−10%) | 180 mA | 220 mA |

## A.4 CALCULATED RELIABILITY

The module has a calculated MTBF (Mean Time Between Failures) of 32000 hours minimum when calculated according to MILSTD 217D.

## A.5 STANDARDS AND REGULATIONS

The Graphics Option module complies with the following standards and recommendations:

- o DEC Standard 119 − Digital Product Safety (covers UL 478, UL 114, CSA 22.2 No. 154, VDE 0806, and IEC 380)
- o IEC 485 − Safety of Data Processing Equipment
- o EIA RS170 − Electrical Performance Standards − Monochrome Television Studio Facilities
- o CCITT Recommendation V.24 − List of Definitions for Interchange Circuit Between Data Terminal Equipment and Data Circuit Terminating Equipment

- CCITT Recommendation V.28 − Electrical Characteristics for Unbalanced Double−Current Interchange Circuits

## A.6 PART AND KIT NUMBERS

| | |
|---|---|
| Graphics Option | PC1XX−BA |
| Hardware: | |
| Printed Circuit Board | 54−15688 |
| Color RGB Cable | BCC17−06 |
| Software and Documentation: | |
| Installation Guide | EK−PCCOL−IN−001 |
| Programmer’s Guide | AA−AE36A−TV |
| GSX−86 Programmer’s Reference Manual | AA−V526A−TV |
| GSX−86 Getting Started | AA−W964A−TV |
| Diagnostic/GSX−86 Diskette | BL−W965A−RV |
| Rainbow 100 Technical Documentation Set | QV043−GZ |

# APPENDIX B

# RAINBOW GRAPHICS OPTION -- BLOCK DIAGRAM

NOTE

This will be a fold-out sheet 11" by approx. 23". The left 8.5" by 11" to be left blank so that the diagram, on the right-hand 11" by 14" or so, can be visible all the while the manual is being used.

```
Fri 20-Apr-1984 11:09 EDT
```