MOTOROLA

M68KVSGM/D1

VME/10 Microcomputer System Command and Graphics Primitives Reference Manual

MICROSYSTEMS

QUALITY • PEOPLE • PERFORMANCE

M68KVSGM/D1

NOVEMBER 1983

VME/10

MICROCOMPUTER SYSTEM

COMMAND AND GRAPHICS PRIMITIVES

REFERENCE MANUAL

First Edition

PREFACE

This reference manual (M68KVSGM/D1) and source code on the VME/10 Winchester drive is supplied for the sole purpose of familiarizing you with the graphics capabilities of the VME/10 System. The source code was developed by Motorola Inc. for internal use, such as product testing and writing of demonstration programs for trade shows and other customer presentations. This package is being given to you, the VME/10 Microcomputer System user, for your use and is not to be considered a product from Motorola Inc. and, as such, will not be supported.

The proposed graphics software standards are currently being evaluated by Motorola Inc. The user is cautioned not to expect compatibility of the enclosed source program with future graphics packages or operating systems offered as products by Motorola Inc.

Suggestions and recommendations regarding the formation of graphics essentials will be appreciated and should be addressed to:

Motorola Microsystems
2900 S. Diablo Way
Tempe, AZ 85282
Attn: Gary Hughes

Included on the Winchester drive under user 1, Catalog GRAPHICS, are approximately 50 files. A description of these files follows:

SERVER.LO - This is a graphics server which operates in the background mode under VERSAdos. It is invoked by typing @SERVER on the command line. Several programs demonstrating graphics on the VME/10 may now be run. This server may be terminated by logging off, by depressing the BREAK key, or by giving the VERSAdos command TERM SERV. The graphics server must be invoked in the background mode before executing DEMO, EXAMPLE1, EXAMPLE2, or EXAMPLE3.

DEMO.LO - This is a program, complete with a menu, which allows the user to select various graphics demonstration programs. The graphics server must be running as a background task before invoking this program. The user must also be logged in under user 1, with the default catalog GRAPHICS, for DEMO to work properly.

EXAMPLE1.SA
EXAMPLE1.LO - This is a very short and simple example of how to write a program to run with the graphics server supplied. It draws and fills a small circle in the center of the display.

EXAMPLE2.SA
EXAMPLE2.LO - This is another example of a program using the graphics server. It uses scaling and rotation.

EXAMPLE3.SA
EXAMPLE3.LO - This program draws a filled triangle, rotates it, and uses the exclusive-OR attribute. An assembly language subroutine is called from the graphics primitive level.

SERVEQU.SA
SERVER1.SA
SERVER2.SA
SERVER3.SA
SERVER4.SA
SERVER5.SA - These files comprise the whole source to the graphics server. They are assembled and linked by the use of a chain file SERVER.CF.

GOFF.SA
GOFF.LO - This is a utility used to turn off the high-resolution graphics mode when using the server and an application program. It is useful in restoring the display to normal character mode without the necessity of rebooting the VME/10.

GRFEQU.SA - is an equate file useful in assembling application programs using the graphics server.

SO suffix files are programs called by the DEMO.LO program. Most are used in function key F1 demo.

PO suffix files are programs called by the DEMO.LO program.

PX suffix files are bitmap data files used in the DEMO.LO program for the two "pictures" and the 68000 logo (1 pixel per nibble).

TABLE OF CONTENTS

		Page
CHAPTER 1	GRAPHICS DRIVER DESCRIPTION	
1.1	INTRODUCTION	1-1
1.2	PHILOSOPHY	1-1
1.3	OVERVIEW	1-1
1.3.1	Primitives	1-1
1.3.2	Display Data Formats	1-2
1.3.3	Scaling	1-3
1.4	STRUCTURE	1-4
1.4.1	Master Control Area	1-4
1.4.1.1	Error Address	1-4
1.4.1.2	Display Status	1-4
1.4.2	Command Channel	1-5
1.4.2.1	Command Word	1-5
1.4.2.2	Command Status	1-5
1.4.2.3	Command Program Area	1-5
1.4.3	Display Segments	1-5
1.4.3.1	CMD/DSPLY and Status Words	1-7
1.4.3.2	Display Attributes	1-7
1.4.3.2.1	Scale Factor	1-7
1.4.3.2.2	X and Y Center	1-7
1.4.3.2.3	Color	1-8
1.4.3.2.4	Color Fill	1-8
1.4.3.2.5	Character Size	1-8
1.4.3.2.6	Mask	1-8
1.4.3.2.7	CESP Pointer	1-8
1.4.3.2.8	Count Word	1-8
1.4.4	Option Area	1-9
1.4.4.1	Symbol Table	1-9
1.4.4.2	Bit-Map Area	1-9
1.4.4.3	Common Subroutine Area	1-9
1.5	OPERATION	1-9
1.5.1	Graphics Server	1-9
1.5.2	Commands	1-11
1.5.3	Graphics	1-12
1.6	COMMAND CHANNEL PROTOCOL	1-13
1.6.1	Access by the Application Program	1-13
1.6.2	Access by the Graphics Driver	1-13
1.7	MEDIA CONTENTS	1-14
CHAPTER 2	USER SOFTWARE IMPLEMENTATION	
2.1	INTRODUCTION	2-1
2.2	DECLARE AND EQUATE STATEMENTS	2-2
2.3	CREATE DISPLAY SEGMENT CONTROL PACKETS	2-3
2.4	CLOSED PRIMITIVE SEGMENTS	2-4
2.4.1	Segment Creation	2-4
2.4.2	Single Segment Execution	2-5
2.4.3	All Segment Execution	2-5
2.5	CREATE BLINK SEGMENTS	2-6
2.6	CONTINUOUS EXECUTION SEGMENTS	2-7
2.6.1	Open Continuous Execution Segments	2-7
2.6.2	Execute Continuous Execution Segments	2-8

TABLE OF CONTENTS (cont'd)

		Page
APPENDIX A	COMMAND PRIMITIVES	A-1
APPENDIX B	GRAPHIC PRIMITIVES	B-1

LIST OF ILLUSTRATIONS

FIGURE 1-1.	Communications Interface Shared RAM Memory Map	1-4
1-2.	Display Segment Memory Map	1-6

CHAPTER 1

GRAPHICS DRIVER DESCRIPTION

1.1 INTRODUCTION

The graphics driver interprets commands and displays data placed in the shared RAM area by the application program to produce full color/monochrome presentations on the screen of a color/monochrome CRT monitor. This chapter provides a thorough description of the driver. Throughout this discussion, all references to addresses are in hexadecimal ($) and are relative to the base address of the shared RAM communications interface.

1.2 PHILOSOPHY

Most graphic CRT displays are stand-alone systems using a serial communication link to the application program. Because of this, most features of the display are designed-in and cannot be readily altered by the user. The display update rate is limited by the speed of the serial communications channel. This approach offers a certain ease-of-use in some applications, many require a significant software effort to implement, while others cannot be realized at all.

In order to provide the greatest application flexibility with the least amount of required hardware and/or software effort, a user-oriented display system should combine all of the alphanumeric and graphic control hardware found in most graphic displays with a highly flexible software package capable of direct intervention by the user via the application program. A display system with this degree of flexibility would impose strict disciplines on user-prepared software. To reduce this burden while still providing powerful software features, the control software must have a regular structure with a well defined user interface that is easy to use, extend, and debug.

Many common graphic figures (such as lines, circles, arcs, and rectangles) should be readily available for the user program to position and display at any location on the screen. Figures should be capable of being outlined or shaded in any color or size by simply identifying an attribute for each.

1.3 OVERVIEW

The graphics driver incorporates a straightforward means of providing the application program with overriding control of graphics operation. The driver will accept commands and graphic controls from the application program via the communications interface.

1.3.1 Primitives

Primitives are machine code routines used to perform specific control and display tasks. Residing within the graphics driver are predefined primitives that permit the application program to specify by primitive number each task to be performed.

Two classifications of primitives are used -- command and graphic. Generally, command primitives are used to control the operation of the display system (i.e., system configuration, operational status, define display segments, etc.), while graphic primitives describe the character or graphic figure for display (i.e., alphanumerics, rectangles, figure placement, etc.).

Command primitives are specified to the driver by the application program via the command channel portion of the communications interface.

Graphic primitives are specified to the driver by a number stored in the display segment. This primitive number is interpreted by a graphic interpreter to produce the character or figure on the display screen. A control packet is also used in conjunction with each display segment to determine the attributes of the display produced by that segment (i.e., scale, color, size, etc.).

1.3.2 Display Data Formats

Either of two data formats can be used to store display data in memory -- bit-map and coordinate. The bit-map format permits the user to specify the on/off state of each individual pixel on the screen, and is particularly useful in producing special, fine-grain displays. The coordinate format permits the user to express any location on the screen in terms of its X (horizontal) and Y (vertical) coordinate. Coordinate values may be either positive or negative integer numbers expressed in absolute or relative terms. Absolute coordinates specify an actual pixel location using X and Y values, whereas relative coordinates specify the X and Y displacement between the current coordinate and the next location. Positive X values indicate all locations to the right of the current position, while positive Y values indicate all locations above. Negative X and Y values indicate all locations left and below the current position, respectively.

When the driver is initialized, the point at which the X and Y axis intersect (0,0) is located in the lower left-hand side of the display screen. If this point of intersection is not changed, then any X and Y position can be expressed in absolute terms using only positive integer numbers. However, if the point of intersection (0,0) is changed (i.e., moved to the center of the screen), then both positive and negative integer numbers will be needed to specify every screen location. Positive and negative values are also used when specifying coordinates in relative terms.

Due to the large number of pixels on the display screen (800 x 600 maximum), two 16-bit binary numbers are required to define a primitive requiring X and Y coordinates. These coordinates are each represented by a single word (16-bit) two's complement number. Since the maximum range capable of being defined by 16-bits (0 to 65,535) exceeds the maximum range of the display, the responsibility for limiting the range lies with the user (no hardware exists to limit this parameter). If the value is exceeded, two faults can occur:

a. An excessive X value will wrap-around, causing a change in the Y value.

b. An excessive Y value or an excessive X + Y value can cause an address to be generated that will be outside of the display memory range, resulting in generation of a bus-trap error.

1.3.3 Scaling

Scaling is the process used to control the size of the displayed figure. When the driver is initialized, the 0,0 coordinate is placed at the lower left-hand corner of the screen, and no enlargement or reduction in size will be performed. This size is referred to as Scale 0 and permits the X and Y coordinates of any location on the screen to be expressed with two 16-bit 2's complement words, using only the ten least significant bits and the sign. Although Scale 0 offers the shortest processing time, both the X and Y coordinate values may be exceeded by the programmer. When specifying coordinates in absolute values, the user can easily prevent the limits from being exceeded. When relative values are used, preventing the display limits from being exceeded becomes more difficult, since the point being related to moves about the screen.

Scale 1 is software selectable by the user and provides X and Y coordinates having 15 significant bits instead of the ten bits of Scale 0. This permits the application program to operate with an external peripheral device (such as a plotter) capable of using full 15-bit X and Y coordinate resolution, while still maintaining the 10-bit resolution requirements of the CRT display. Instead of using the ten least significant bits of the word (as done in Scale 0), Scale 1 uses the ten most significant bits of the word (excluding sign bit, which occupies the most significant bit position) for CRT display operations. This permits the lower six bits to be used for increased X and Y resolution. Before the driver processes coordinates with Scale 1 for CRT display, the lower six bits are removed and the upper ten bits are shifted right six places for standard coordinate processing (as performed for Scale 0). The original user-specified coordinates are not changed, thus permitting those coordinates to be accessed by a user-prepared peripheral driver routine (such as a plotter). Because of these shift operations, Scale 1 specified coordinates require a longer time to process than coordinates using Scale 0. Note that the screen limits may still be exceeded using Scale 1.

The driver can also provide Full Scaling of the display (enlarge and reduce) by using the multiply instruction of the resident MC68000 processor to perform a full 2's complement multiply of the coordinate values by a user-specified scale factor (scale factor must be a positive number). This results in a 32-bit sum in which the lower ten bits of the upper word represent the scaled coordinates. The lower word is discarded. This scaled coordinate is then used in place of the coordinate originally specified in the display segment, thus permitting scaling up or down in size. Once again, it is possible to exceed the screen limits if the Large scale factor is used. The following formula provides the means of calculating Full Scale coordinate positions.

$$\frac{\text{Coordinate Number X Scale Factor}}{65{,}536} = \text{Scaled Coordinate}$$

NOTE

Scaling only applies to relative values.
Absolute values will not be scaled.

1.4 STRUCTURE

The communications interface is divided into four main areas, each providing a specific system function (refer to Figure 1-1).

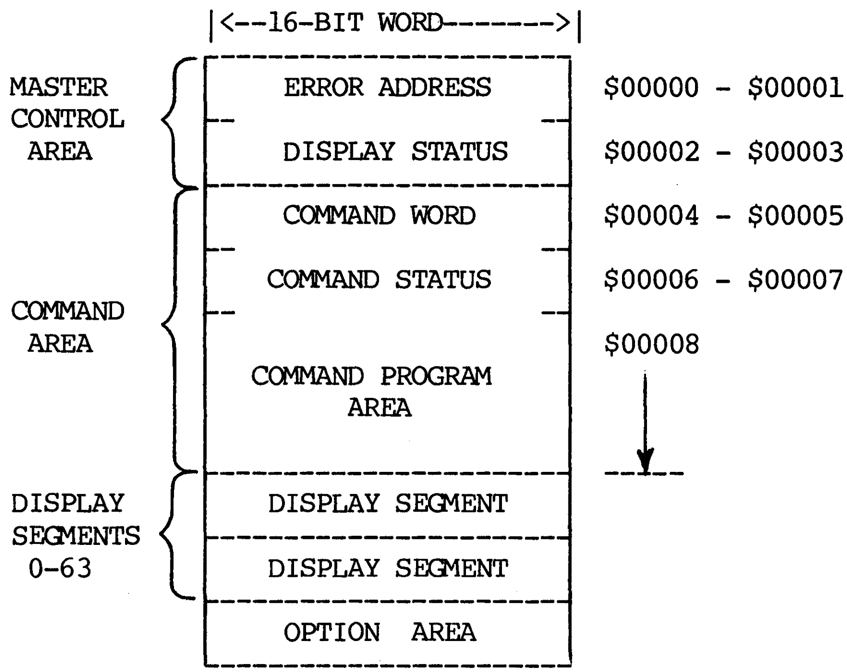


FIGURE 1-1. Communications Interface Shared RAM Memory Map

1.4.1 Master Control Area

The master control area provides the means of overriding driver operation. It contains two words, which are described in the following paragraphs.

1.4.1.1 Error Address. This word will contain the address (relative to the start of the shared segment) of any error detected during command or graphics primitives execution.

1.4.1.2 Display Status. If an error occurs during execution of any display segment, a non-zero value will be stored in the display status word (00002). After the error in the display segment has been found and corrected, the display status word MUST be reset to a logic 0 value to permit further execution of display segments.

1.4.2 Command Area

All commands issued by the application program to the driver are received through the command area and processed by the command processor. The command area consists of a command word, a command status, and a command program area. The following paragraphs describe each of these sections.

1.4.2.1 Command Word. During polling, the command processor checks the command word (00004 and 00005) for a non-zero value. When a non-zero is found, execution control is passed to the command interpreter to initiate interpretation and execution of the commands in the command program area.

1.4.2.2 Command Status. The command status word (00006 and 00007) indicates whether or not command primitives in the command program area are being executed. If the command status word contains a logic 1 value, then command primitives are NOT being executed and the command program area can be accessed by the application program. If the command status word is cleared to a logic 0 value, then previous command primitives are still being executed and the command program area is not yet available. After execution of all command primitives, a logic 1 value is set into the command status word. If an error occurred during primitive execution, the most significant bit of the command status word will be set to a logic 1 (negative value).

1.4.2.3 Command Program Area. The application program enters command primitives in the command program area (00008 and Up) for interpretation and execution by the command interpreter. These command primitives control the sequence of execution, the configuration of the shared RAM communications interface, and the creation and execution of display segments. Unlike other command words, the size of the command program area is determined by the number of command primitives used by the application program. A thorough description of each predefined command primitive within the graphics driver can be found in Appendix A.

1.4.3 Display Segments

Display segments are the control structure used by the application program to display graphic figures and alphanumeric characters on the CRT screen. Up to 64 display segments can be specified by the application program, with each having its own control packet and program area containing graphic primitives (refer to Figure 1-2).

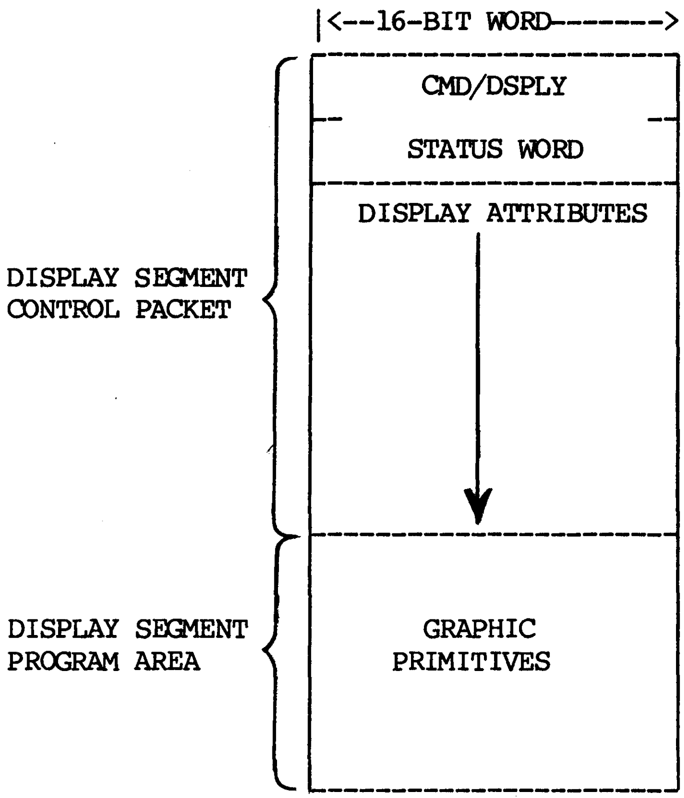


FIGURE 1-2. Display Segment Memory Map

Display segment control packets consist of 32 words (16 bits each) containing segment control flags, segment type, and segment attributes. A display segment is identified to the driver by the application program's use of command primitives issued through the command area. Four different types of display segments can be specified:

a. Visible
b. Non-visible
c. Blink
d. Continuous execution (as new primitives are added)

The following paragraphs describe each of the control words and their offset from the starting address of the display segment used by the application program. Additional allocations have been reserved within the control packet for parameters supplied by the driver. These allocations are not identified in this manual.

1.4.3.1 CMD/DSPLY and Status Words. The CMD/DSPLY word (display segment address + 0) and the status word (display segment address + 2) perform two different functions, depending on the type of display segment defined.

When a continuous execution segment has been specified (indicating all graphic primitives within this display segment are to be continuously executed), the graphic processor interprets the CMD/DSPLY word and the status word as handshake controls similar to those provided by the command word and command status word in the command channel. In this case, a zero value in the status word indicates to the application program that graphic primitives within the display segment program area are being executed and that additional primitives cannot be accepted. Execution begins with the graphic primitive whose starting address is stored in the CESP pointer (part of the display attributes). After all current graphic primitives have been executed, the graphic processor stores a non-zero value in the status word to indicate to the application program that additional graphic primitives will be accepted. The application program may then store these additional primitives into the display segment program area. After completing this task, the application program must then set the CMD/DSPLY Word to any non-zero value to indicate to the graphic processor that new graphic primitives may be executed.

When continuous execution is not specified, the CMD/DSPLY word is interpreted as being a visible/non-visible flag (non-zero = visible/0 = non-visible). If non-visible, the graphic interpreter does not execute the graphic primitives in the display segment program area. If visible, the interpreter will change the status word to a zero value (indicating the display segment program area is in use); will execute the graphic primitives within this display segment; and will then set the status word back to a non-zero value to indicate completion. The graphic processor DOES NOT change the CMD/DSPLY word in the non-continuous execution mode.

1.4.3.2 Display Attributes. Words within the display attributes area are used to provide additional parameters for display primitives. Each of the attributes that can be specified by the application program are described in the following paragraphs in the order in which they must be specified.

1.4.3.2.1 Scale Factor - The scale factor word (display segment address + 4) provides the means by which the application program signifies to the driver whether the original display segment coordinate values will be used or whether the original values will be increased or reduced in size. When the scale factor word contains a zero value, Scale 0 is selected and no scaling is performed. When the scale factor contains a 1, Scale 1 is selected. When the scale factor is any positive number greater than 1, full scaling operations are performed. The scale factor may also be specified using the SCALEF graphic primitive. Once the scale factor has been specified, it will not change until another SCALEF primitive is encountered, or the scale factor word within the display segment control packet is changed.

NOTE

Scaling only applies to relative values.

1.4.3.2.2 X and Y Center - The X and Y Center words (display segment address + 6 and +8, respectively) permit the application program to change the location of the 0,0 coordinate to the coordinate specified by the values contained in the words.

1.4.3.2.3 Color - The color word (display segment address + 10) permits the application program to specify one of eight display colors used to draw graphics figures. Color may also be specified using the COLOR graphic primitive. Once the color has been specified, it will not change until another COLOR primitive is encountered, or the color word within the display segment control packet is changed.

1.4.3.2.4 Color Fill - The color fill word (display segment address + 12) permits the application program to specify the color to be used to fill closed figures. Color fill may also be specified using the FILL graphic primitive. Once the color of the fill has been specified, it will not change until another FILL primitive is encountered, or the color fill word within the display segment control packet is changed.

1.4.3.2.5 Character Size - The character size word (display segment address + 14) allows alphanumeric display characters to be enlarged. Eight enlargement ratios are available, from 1:1 through 1:8. The application program specifies the size desired by storing a value (0 through 7, respectively) into the character size word. Character size may also be specified using the CHSIZE graphic primitive. Once the character size has been specified, it will not change until another CHSIZE primitive is encountered, or the character size word within the display segment control packet is changed.

1.4.3.2.6 Mask - The mask word (display segment address + 16) specifies the bit-plane(s) to be written. The bit-plane(s) may also be specified using the MASK graphic primitive. Once the bit-plane(s) has been specified, it will not change until another MASK primitive is encountered, or the mask word within the display segment control packet is changed.

000 = no bit-plane
001 = red bit-plane
010 = blue bit-plane
011 = red and blue bit-planes
100 = green bit-plane
101 = green and red bit-planes
110 = green and blue bit-planes
111 = all bit planes

1.4.3.2.7 CESP Pointer - The CESP pointer (display segment address + 18) is used whenever the display segment is to be executed in the continuous execution mode. When the CMD/DSPLY word is set to any non-zero value, the CESP pointer must contain the starting address of the first graphic primitive to be executed within a group of graphic primitives in the display segment program area. This pointer (a 32-bit long word) can be changed by the application program to identify any graphic primitive as the first to be executed.

1.4.3.2.8 Count Word - The count word (display segment address + 22) is used to specify the blink rate for a blink display segment. To determine the proper value, divide the desired blink rate by one-half second. For example, a count word with value 1 yields a blink rate of one-half second, while a count word of value 10 yields a blink rate of five seconds.

1.4.4 Option Area

The following areas within the communications interface are optionally used, as determined by the graphic display software package in the application program.

1.4.4.1 Symbol Table. The Symbol Table area is available to the user for building special display symbol sets for use with the SYM graphic primitive.

1.4.4.2 Bit-Map Area. The bit-map Area permits the user to directly control the color of any or all pixels within the display screen area. This is particularly useful when specifying a special, high-resolution symbol (i.e, logo, special title, etc.) for display.

1.4.4.3 Common Subroutine Area. The common subroutine area permits space for user-prepared programs using common subroutines of primitives normally used to define common display subpictures (pictures using less than the full display area). If these subpictures are prepared using relative X and Y coordinate values, this subroutine of primitives can be used to display the subpicture in multiple areas of the screen. The GJSR graphic primitive is used to call subroutines in this area, while the GRTS graphic primitive is used to terminate all subroutines in this area.

1.5 OPERATION

The following paragraphs provide an operational description of the graphics driver.

1.5.1 Graphics Server

The VME/10 graphics server is structured as a VERSAdos server task. The server is run as a background task and is invoked under VERSAdos by placing the commercial "at" sign (@) before the server's name.

Example:

```
=@server
=
```

Note that the VERSAdos prompt returns with nothing apparent happening. Also note that it is assumed (and vitally important) that no other user tasks are active when the server is loaded. Communication between the application program and the server is via TRAP #8 server calls and a shared segment. The type of server call is placed into register D0.

The server may be terminated (that is, the task may be terminated) by =TERM SERV or by the "break" key. It is terminated also by logging off.

Since the server task is loaded, there may not be enough memory remaining in the system for other large programs to be loaded at the same time. Otherwise the server may be left running in the background during a session.

The following are the calls which may be made to the graphics server:

```
Open server        D0 = 0
                   A0 = size of shared segment
                   return D0 = 0  normal
                          D0 = 1  already open
                          D0 = 2  video RAM not available

Close server       D0 = 1
                   return D0 = 0  normal
                          D0 = 3  invalid request

Execute graphics   D0 = 2
                   return D0 = 0  normal
                          D0 = 4  error in command execution

Request 100%       D0 = 3
duty cycle         return no parameters

Request 50%        D0 = 4
duty cycle         return no parameters
```

Note that the open server call will cause the VME/10 display to go into high-resolution mode, and will enable the display of graphics. A subsequent close server call will disable the display of graphics, and will return the VME/10 display to normal. Therefore, it is highly recommended that after displaying graphics, but before terminating the server, that a close server call be made. Otherwise, the user may be left with the VME/10 display in high-resolution mode and with graphics enabled.

The following is an example of the 68000 code required to open, execute, and close the graphics server.

```
        CLR.L     D0          Open graphics directive
        MOVE.L    #$1000,A0   Size of shared segment required for this
                              application
        TRAP      #8
        BNE.S     ERROR
*
        LEA       PARBK,A0    Attach the segment for graphics command
        MOVE.L    #4,D0
        TRAP      #1
        BNE.S     ERROR
*
        LEA       CMDS,A1     Move the commands to the segment
        LEA       CMDSEND,A2
LP1     MOVE.W    (A1)+,(A0)+
        CMP.L     A1,A2
        BNE.S     LP1
*
        MOVE.L    #2,D0       Execute the commands & primitives
        TRAP      #8
*
        MOVE.L    #1,D0       Close the graphics server
        TRAP      #8
*
```

```
EXIT     MOVE.L   #15,D0     Stop
         TRAP     #1
*
ERROR    MOVE.L   #14,D0     Error, abort this program
         TRAP     #1
*
*        Graphics command & primitives
*
CMDS     DC.W     0,0,0,0    MASTER ETC.
         DC.W     OPENS,1    Open segment one
         DC.L     DSPS1-CMDS segment address
         DC.W     CLOSES,1   Close segment one
         DC.W     EXECS,1    Execute segment
         DC.W     CEND       End
*
*        DISPLAY SEGMENT ONE
*
DSPS1    DC.W     1          COMMAND/DISPLAY
         DC.W     0          STATUS
         DC.W     0          SCALE FACTOR
         DC.W     0,0        X,Y CENTER
         DC.W     1          COLOR
         DC.W     2          FILL
         DC.W     0          CHARACTER SIZE
         DC.W     7          MASK
         DC.L     0          CESP
         DC.W     0          COUNT WORD
         DC.W     0,0,0,0    RESERVED
         DC.L     0,0,0,0,0,0,0,0
*
*        ----- DISPLAY SEGMENT -----
*
         . . . . . . . . . .
         Graphics Primitives
         . . . . . . . . . .

         DC.W     PEND
*
CMDSEND  EQU      *
*
PARBK    DC.L 0,0,$20002000,'&VDM',0,0
```

1.5.2 Commands

When enabled by a TRAP #8 call with D0 = 2 (execute), the command processor polls the command channel Command Word to determine whether or not new commands have been issued by the application program. If new commands are found (Command Word = non-zero value), the command processor transfers execution control to the command interpreter to interpret and execute the new command primitives. If the command channel is not active, the command processor will then check for any additional active display segments to be executed. If no active display segments are found and none are currently being executed, the command processor will repeat the polling process.

The driver maintains a bit representation of all segments currently ready to be executed called an active list. The command processor checks this list and if a segment is ready to be executed, execution control is transferred to the graphic processor. If continuous execution is not specified, the corresponding bit in the active list is cleared to logic 0. After all display segments have been checked, the command processor will repeat the polling process.

Display segments have their own primitive interpreter with its own table of primitives, called graphic primitives. These types of primitives are used to describe a graphic figure to be drawn on the display screen. Each primitive word consists of two parts: a code (lower byte) indicating the primitive type and an attribute (upper byte).

ATTRIBUTE	CODE
15 8	7 0

The code indicates the type of graphic operation -- DOT, MOVETO, DRAWTO, CIRCLE, etc. The attribute describes additional modifiers to the primitive -- shading, XOR figure to screen, 90 degree character rotation, absolute or relative of X,Y coordinates, line pattern, etc. These modifiers are bit represented in the upper byte as follows:

```
Bit 0-2 - Line Pattern (0-7, 0 = solid line)
Bit 3   - 0
Bit 4   - Absolute = 0, Relative = 1
Bit 5   - Character Rotation
          (0 deg. = 0, 90 deg. = 1)
Bit 6   - XOR = 1
Bit 7   - FILL = 1
```

How these bits are interpreted depends on the type of primitive. Thus, the FILL bit is not valid in a MOVETO primitive because there is no figure to fill. Only the absolute/relative bit is valid for the MOVETO primitive. Appendix B provides a complete description of all predefined graphic primitives available within the graphics driver.

When a display segment is executed by the graphic processor, all graphic primitives within the program area are interpreted, with the corresponding graphic routines used to produce the display data stored in the display memory bit-planes. Since the bit-plane data is continuosly being sent to the CRT monitor for display, the data in the bit-planes produces figures on the screen.

1.5.3 Graphics

Graphic operations are controlled by the graphic processor checking each display segment for graphic primitives to execute. If the command word in the display segment contains a non-zero value, the graphic interpreter executes the graphic primitives within the display segment.

1.6 COMMAND CHANNEL PROTOCOL

The following protocols are used by the driver and application program to avoid contention during accesses of the command area (required for use with continuous execution and blink segments).

1.6.1 Access by the Application Program

The following steps indicate the sequence used by the application program to access the command segment.

a. Checks if the command program area in the command channel is currently being used. If the command status word contains a logic 1 value, then the command program area is NOT being used. If a zero value, it is in use and is not yet available to the application program. If the command status word is negative (most significant bit set to logic 1), an error has occurred during processing of the previous command primitive.

b. Updates or changes command primitives in the command program area.

c. Clears the command status word in the command channel to a zero value to indicate the channel is being used.

d. Sets the command word to any non-zero value. This causes the graphics driver to begin interpreting the commands in the command program area.

1.6.2 Access by the Graphics Driver

The following steps indicate the sequence used by the driver to access the command channel.

a. Checks the state of the command word before executing commands in the Command Program Area. If the command word value is non-zero, execution proceeds to step b. If zero, execution control is returned to the command processor.

b. Clears the command word to a zero value and proceeds to interpret and execute command primitives in the command program area.

c. Upon completing a single execution of all command primitives in the command program area, the command status word is set to a logic 1 value if no errors were encountered. If an error occurred, an error code will be stored ONLY in the command status word with the most significant bit set (negative sign).

d. Execution control is returned to the command processor to execute any active display segments.

1.7 MEDIA CONTENTS

The software for the VME/10 graphics server is supplied in source form to permit tailoring to specific applications. It may be supplied as part of system release on the Winchester disk in catalog "GRAPHICS", or on a separate floppy diskette.

The catalog contains the source code for the VME/10 graphics server, plus an application program using the server.

a. SERVER - this program provides a graphics segment and primitives handler.

   The modules that make up the server are:

   1. SERVEQU.SA - an equate file used in the assembly of the other modules.

   2. SERVER1.SA - the main program. It allocates memory and accepts messages from application programs, directing their requests to the actual graphics handler.

   3. SERVER2.SA, SERVER3.SA, SERVER4.SA SERVER5.SA - these modules contain the code to process the segment and graphics primitives.

   4. SERVER.CF - a chain file that assembles and links the server.

   The server may be assembled and linked with the following:

   ```
   =SERVER.CF
   ```

b. DEMO.LO - This is a program, complete with a menu, which allows the user to select various graphics demonstration programs. The graphics server must be running as a background task before invoking this program. The user must also be logged in under user 1, with the default catalog GRAPHICS, for DEMO to work properly.

c. EXAMPLE1.SA - this program is a very simple application that uses several different graphics primitives. It opens a single segment and executes it. This program is assembled and linked with the following:

   ```
   =ASM EXAMPLE1
   =LINK EXAMPLE1
   ```

   EXAMPLE1 is then executed with the following:

   ```
   =@SERVER
   =EXAMPLE1
   ```

   Note that executing EXAMPLE1 will leave the VME/10 display in high-resolution mode and with graphics display erased. Use GOFF to return the display to normal.

d. EXAMPLE2.SA, EXAMPLE3.SA - these are two other example programs which use the graphics server. They are assembled and linked the same as EXAMPLE1.

e. GOFF.SA - This program is an application that simply closes the server, thus disabling graphics display and returning the VME/10 display to normal. This program is assembled and linked with the following:

```
=ASM GOFF
=LINK GOFF
```

GOFF is then executed with the following:

```
=GOFF
```

Note that the execution of GOFF does not terminate the server. It only closes the server, thus returning the display to normal.

f. GRFEQU.SA - A file of equates that may be included into graphics applications programs. The equates are those for the graphics and command primitives and their attributes.

CHAPTER 2

USER SOFTWARE IMPLEMENTATION

2.1 INTRODUCTION

This chapter provides examples of typical software modules that must be prepared by the user to create and execute various types of command and display segments as part of his graphic application package. Throughout these paragraphs, the processor registers indicated herein are provided as examples of the type of information needed to be specified prior to beginning execution:

A5 = Display Segment Starting Address
A6 = Starting Address of the Shared Segment
D0 = Display Segment Number

2.2 DECLARE AND EQUATE STATEMENTS

Software programs contain a group of common names and descriptions usually identified at the beginning of each program or module and referred to as declaration and equate statements. These statements permit the programmer to allocate memory for program variables and to identify label names commonly used throughout the program. The following examples provide three types of tables used to allocate memory for display segment tables, command area offsets, and display segment offsets. These tables are referenced throughout the examples presented in this chapter and may also be incorporated in the user's program.

```
* DISPLAY SEGMENT TABLE
SEGTABLE EQU      *
         DC.W     0                     COMMAND WORD
         DC.W     1                     STATUS WORD
         DC.W     0                     SCALE FACTOR
         DC.W     0                     X CENTER
         DC.W     0                     Y CENTER
         DC.W     7                     COLOR=WHITE
         DC.W     7                     FILL COLOR
         DC.W     0                     1:1 CHARACTER SIZE
         DC.W     7                     MASK-ALL COLORS ENABLED
         DC.L     SEGTABLE+$40          CONTINUOUS EXECUTION POINTER
         DC.W     2                     BLINK RATE (1 SEC)

* COMMAND AREA OFFSET EQUATE TABLE
MASTER   EQU      0                     MASTER WORD
DSPSTAT  EQU      +2                    DISPLAY STATUS
CMDWD    EQU      +4                    COMMAND WORD
CMDSTAT  EQU      +6                    COMMAND STATUS
CMDPROG  EQU      +8                    COMMAND PROGRAM AREA

* DISPLAY SEGMENT OFFSET EQUATE TABLE
PCMD     EQU      0                     COMMAND WORD
PSTAT    EQU      2                     STATUS WORD
SCALEF   EQU      4                     SCALE FACTOR
XCENTER  EQU      6                     X CENTER
YCENTER  EQU      8                     Y CENTER
COLOR    EQU      10                    COLOR
COLORFIL EQU      12                    COLOR FILL(RATIO/SEC/PRI)
CHSIZE   EQU      14                    CHARACTER SIZE
MASK     EQU      16                    BIT-PLANE MASK
CESP     EQU      18                    CONTINUOUS EXECUT PTR
COUNT    EQU      22                    BLINK RATE COUNT
```

2.3 CREATE DISPLAY SEGMENT CONTROL PACKETS

Prior to opening a display segment through the command area, the application program must store a segment control packet beginning at the display segment's starting address as specified in processor address register A5. Upon opening the display segment, additional information is stored in the packet by the graphics driver. If the first word in the packet is a zero value (non-visible OR not active-continuous execution), the display segment will not be executed if closed, blinked, or established as a continuous execution segment. The following example illustrates the manner in which a display segment control packet should be created. Upon completing execution of the program in this example, the following processor registers will contain the data indicated:

A4 = Starting Address of Display Segment Program Area
A5 = Starting Address of Display Segment
A6 = Starting Address of the shared segment
D0 = Display Segment Number
D1 = Offset to Display Segment Address

```
SEGCPKT  EQU     *
         MOVE.L  A5,-(A7)          SAVE SEGMENT START ADDR
         LEA.L   SEGTABLE(PC),A4   SEGMENT TABLE ADDRESS
         MOVE.W  #11,D7            NUMBER OF WORDS - 1
SEGCPKT1 MOVE.W  (A4)+,(A5)+       MOVE TO COMM INTERFACE
         DBF     D7,SEGCPKT1       LOOP COUNT
         MOVE.L  (A7)+,A5          RESTORE STARTING ADDRESS
         LEA.L   64(A5),A4         GET PROG AREA START ADDR
         MOVE.L  A4,D1
         SUB.L   A6,D1             CALC. SHARED RAM OFFSET
         MOVE.L  D1,CESP(A5)       IF CONT. EXEC. SEG.
         SUB.L   #64,D1            OFFSET STARTING ADDRESS
         RTS
```

2.4 CLOSED PRIMITIVE SEGMENTS

The following paragraphs describe the method of creating primitives within a closed display segment that can then be executed singly by the EXECS command primitive or sequentially by the EXECAS command primitive.

2.4.1 Segment Creation

The example in this paragraph illustrates a method of programming a closed command primitive.

```
* COMMAND PRIMITIVES EQUATES
OPENS    EQU      2
CLOSES   EQU      3
CEND     EQU      1

         BSR      SEGCPKT              PUT CNTRL PCKT IN SHRD RAM
L1       TST.W    CMDSTAT(A6)          CK CMD STATUS
         BMI      CMDERR               IF NEG.-CMD ERROR
         BNE      L1                   NOT YET AVAIL.
         LEA.L    CMDPROG(A6),A3       GET PROG AREA STRT ADDR
         MOVE.W   #OPENS,(A3)+         OPEN CMD PRIMITIVE
         MOVE.W   D0,(A3)+             GET SEGMENT #
         MOVE.L   D1,(A3)+             GET SEG. OFFSET
         MOVE.W   #CEND,(A3)           END OF CMD PRIMITIVES
         CLR.W    CMDSTAT(A6)          CLOSE CMD CHANNEL
         MOVE.W   #1,CMDWD(A6)         ACTIVATE CMD CHANNEL
         .
         .
Store graphic primitive(s) in display segment program area
         .
         .
         MOVE.W   #1,(A5)              MAKE SEGMENT VISIBLE
L2       TST.W    CMDSTAT(A6)          CK CMD STATUS
         BMI      CMDERR
         BNE      L2                   NOT YET AVAILABLE
         LEA.L    CMDPROG(A6),A3       GET PROG. AREA ADDR
         MOVE.W   #CLOSES,(A3)+        CLOSE THE SEGMENT
         MOVE.W   D0,(A3)+             SEGMENT #
         MOVE.W   #CEND,(A3)           END OF CMD PRIMITIVES
         CLR.W    CMDSTAT(A6)          CLOSE CMD CHANNEL
         MOVE.W   #1,CMDWD(A6)         ACTIVATE CMD CHANNEL
```

2.4.2 Single Segment Execution

The example in this paragraph illustrates the method of executing a single closed display segment.

```
* COMMAND PRIMITIVE EQUATES
EXECS    EQU      5
CEND     EQU      1

L1       TST.W    CMDSTAT(A6)          TEST CMD STATUS
         BMI      CMDERR
         BNE      L1                   NOT YET AVAIL.
         LEA.L    CMDPROG(A6),A3       GET START OF CMD PRIMS
         MOVE.W   #EXECS,(A3)+         GET PRIMITIVE
         MOVE.W   #SEGNMBR,(A3)+       GET SEGMENT #
* EXECS CAN BE USED MULTIPLE TIMES TO EXECUTE OTHER SEGMENTS
         MOVE.W   #CEND,(A3)           END OF CMD PRIMITIVES
         CLR.W    CMDSTAT(A6)          CLOSE CMD CHANNEL
         MOVE.W   #1,CMDWD(A6)         ACTIVATE CMD CHANNEL
```

2.4.3 All Segment Execution

The example in this paragraph illustrates the method of executing all closed display segments.

```
* COMMAND PRIMITIVE EQUATES
EXECAS   EQU      5
CEND     EQU      1

L1       TST.W    CMDSTAT(A6)          TEST CMD STATUS
         BMI      CMDERR
         BNE      L1                   NOT YET AVAIL.
         LEA.L    CMDPROG(A6),A3       GET START OF CMD PRIMS
         MOVE.W   #EXECAS,(A3)+        EXECUTE ALL SEGMENTS
         MOVE.W   #CEND,(A3)           END OF CMD PRIMITIVES
         CLR.W    CMDSTAT(A6)          CLOSE CMD CHANNEL
         MOVE.W   #1,CMDWD(A6)         ACTIVATE CMD CHANNEL
```

2.5 CREATE BLINK SEGMENTS

The example provided in this paragraph demonstrates the method used to create a blink segment.

```
* COMMAND PRIMITIVE EQUATES
OPENS    EQU      2
BLKS     EQU      7
CEND     EQU      1

         BSR      SEGCPKT             PUT CNTRL PCKT IN SHRD RAM
         MOVE.W   #2,BLKRATE(A5)      1 SEC BLINK RATE
         .
         .
```

Place graphic primitives in display segment. For blinking, the XOR display attribute should be used.

```
         .
         .
L1       TST.W    CMDSTAT(A6)         TEST CMD STATUS
         BMI      CMDERR
         BNE      L1                  NOT YET AVAIL.
         LEA.L    CMDPROG(A6),A3      GET PROG AREA START
         MOVE.W   #OPENS,(A3)+        OPEN SEGMENT
         MOVE.W   D0,(A3)+            GET SEGMENT #
         MOVE.L   D1,(A3)+            GET SEGMENT START OFFSET
         MOVE.W   #BLKS,(A3)+         BLINK SEGMENT
         MOVE.W   D0,(A3)+            GET SEGMENT #
         MOVE.W   #CEND,(A3)          END OF CMD PRIMITIVES
         CLR.W    CMDSTAT(A6)         CLOSE CMD CHANNEL
         MOVE.W   #1,CMDWD(A6)        ACTIVATE CMD CHANNEL
         .
         .
```

Place a 1 in the CMD/DSPLY word of the display segment to make it visible when needed and a 0 to turn the blink segment off.

```
         .
         .
         MOVE.W   #1,(A5)             TURN ON VISIBLE SEGMENT
```

2.6 CONTINUOUS EXECUTION SEGMENTS

The following paragraphs describe opening and executing continuous execution segments.

2.6.1 Open Continuous Execution Segments

The continuous execution segment is continuously checked and activated by the CMD/DSPLY and status words. When the segment is activated, the graphics driver uses the Continuous Execution Segment Pointer (CESP) to obtain the starting execution address. The example in this paragraph describes the method used to open a continuous execution segment.

```
* COMMAND PRIMITIVE EQUATES
OPENCES  EQU     $FF02              FF ATTRIBUTE TO OPENS
CEND     EQU     1

         BSR     SEGCPKT            STORE CONTROL PACKET
L1       TST.W   CMDSTAT(A6)        CK COMD STATUS
         BMI     CMDERR
         BNE     L1                 NOT YET AVAIL.
         LEA.L   CMDPROG(A6),A3     GET START OF CMD PRG AREA
         MOVE.W  #OPENCES,(A3)+     OPEN CONT. EXEC. SEG.
         MOVE.W  D0,(A3)+           GET SEGMENT #
         MOVE.L  D1,(A3)+           GET SEG. OFFSET
         MOVE.W  #CEND,(A3)         END OF CMD PRIMITIVES
         CLR.W   CMDSTAT(A6)        CLOSE CMD CHANNEL
         MOVE.W  #1,CMDWD(A6)       ACTIVATE CMD CHANNEL
```

2.6.2 Execute Continuous Execution Segments

This paragraph provides an example of the method used to execute a continuous execution segment.

```
* GRAPHIC PRIMITIVE EQUATES
MOVETO   EQU      3
DRAWTO   EQU      4
RECT     EQU      $1007
COLOR    EQU      12
CIR      EQU      8
GEND     EQU      1

L1       TST.W    PSTAT(A5)
         BEQ      L1                       NOT YET AVAIL.
         MOVE.B   #1,(A4)+                 ATTRIBUTE - RED = 1
         MOVE.B   #COLOR,(A4)+             PRIMITIVE
         MOVE.W   #MOVETO,(A4)+            PRIMITIVIE
         MOVE.W   #30,(A4)+                ABSOLUTE X ADDRESS
         MOVE.W   #60,(A4)+                ABSOLUTE Y ADDRESS
         MOVE.W   #RECT,(A4)+              RECT. PRIMITIVE
         MOVE.W   #40,(A4)+                LENGTH IN X
         MOVE.W   #40,(A4)+                HEIGHT IN Y
         MOVE.W   #CEND,(A3)               END OF CMD PRIMITIVES
         CLR.W    CMDSTAT(A6)              CLOSE CMD CHANNEL
         MOVE.W   #1,CMDWD(A6)             ACTIVATE CMD CHANNEL
* THE CGP MODULE USES THE CURRENT CESP POINTER TO START
* EXECUTION OF PRIMITIVES
L2       TST.W    PSTAT(A5)
         BMI      SEGERR
         BEQ      L2                       NOT YET AVAIL.
         MOVE.L   A4,D0                    LAST PRIMITIVE ADDR
         SUB.L    A6,D0                    GET OFFSET
         MOVE.L   D0,CESP(A5)              NXT EXECUTION ADDR
         MOVE.W   #CIR,(A4)+               CIRCLE PRIMITIVE
         MOVE.W   #40,(A4)+                RADIUS OF CIRCLE
         MOVE.W   #GEND,(A4)               END OF GRAPHIC PRIMITIVES
         CLR.W    PSTAT(A5)                CLOSE DISPLAY SEGMENT
         MOVE.W   #1,PCMD(A5)              ACTIVATE DISPLAY SEGMENT
         MOVE.L   #2,D0
         TRAP     #8                       EXECUTE GRAPHICS
L3       TST.W    DSPSTAT(A6)              CHECK FOR ERROR
         BNE      ERROR
         TST.W    PSTAT(A5)                WAIT UNTIL DONE
         BEQ      L3
```

APPENDIX A

COMMAND PRIMITIVES

This appendix provides a description of each predefined command primitive within the graphics driver. Refer to Chapter 1 for a description of how these primitives are used.

The command primitives are organized in this appendix as follows:

COMMAND PRIMITIVE	CODE	PAGE NUMBER
BITMPTR	13 ($0D)	A-2
BLKS	7 ($07)	A-3
CCUR	26 ($1A)	A-4
CEND	1 ($01)	A-5
CLOSES	3 ($03	A-6
CSETD	18 ($12)	A-7
CUROFF	25 ($19)	A-8
CURON	24 ($18)	A-9
DELAY	20 ($14)	A-10
DELS	4 ($04)	A-11
EXECAS	6 ($06)	A-12
EXECS	5 ($05)	A-13
NCOP	0 ($00)	A-14
OPENS	2 ($02)	A-15
RBLKS	8 ($08)	A-16
SCFN	21 ($15)	A-17
SYMPTR	12 ($0C)	A-18

BITMPTR - BITMAP POINTER

CODE: 13 ($0D)

ATTRIBUTES: None

OPERANDS: Long Address (relative to beginning of shared memory segment)

DESCRIPTION: Establishes the starting address of a bit-mapped display to the graphics driver. Since each pixel is defined in 4 bits, one word will define 4 pixels.

BLKS - BLINK SEGMENT

CODE: 7 ($07)

ATTRIBUTES: None

OPERANDS: Display Segment Number

DESCRIPTION: Adds the previously opened display segment to the blink list. The internal timer service routine will decrement the counter in the display segment control packet and, if zero, will add the segment to the active list and move the count to the counter. If the display segment is visible, it will be executed in the display page. If not visible, the display segment will NOT be executed.

DISPLAY SEGMENT CONTROL PACKET:

- Command word used for visibility (0 = non-visible, non-zero = visible).

- Count must be set to desired blink rate (1 count = 1/2 sec., total on/off time).

CCUR - MOVE COMMAND CURSOR TO X,Y

CODE: 26 ($1A)

ATTRIBUTES: ABSOLUTE/RELATIVE - Absolute or relative

OPERANDS: X-; Y-Coordinates

DESCRIPTION: Move the command (vertical and horizontal) cursors to X-,Y-coordinates. The coordinates may be specified absolutely or as relative to the current screen pointer. Cursor must be ON to be visible (activate/deactivate cursor using command primitives CURON/CUROFF).

CEND - END OF COMMAND PRIMITIVE LIST

CODE: 1 ($01)

ATTRIBUTES: None

OPERANDS: None

DESCRIPTION: This command must be the last command primitive within a group of command primitives within the command program area. It causes the command interpreter to halt execution and to return control to the command processor. A positive value is placed in the command status word to indicate a normal (no error) completion. A negative value indicates that an error has occurred. Error data can then be obtained from the error data table. Errors normally occur only during display system debug.

CLOSES - CLOSE A DISPLAY SEGMENT

CODE: 3 ($03)

ATTRIBUTES: None

OPERANDS: Display Segment Number

DESCRIPTION: Adds a display segment to the valid segment list and allows it to be made active through the EXECS or EXECAS primitives. Before the segment can be closed, it must be opened to obtain its address. The display segment command word is used to indicate the segments visability (0 = non-visible, non-zero = visible).

CSETD - SET COMMAND DISPLAY PAGE

CODE: 18 ($12)

ATTRIBUTES: COLOR - Specified in Upper Byte

OPERANDS: None

DESCRIPTION: Set the entire display page to the specified color. Note that a CSETD with color = 0 (command word = $0012) will clear the entire display.

COLORS:

Black	= 0
Red	= 1
Blue	= 2
Magenta	= 3
Green	= 4
Yellow	= 5
Cyan	= 6
White	= 7

CUROFF - TURN OFF CURSORS

CODE: 25 ($19)

ATTRIBUTES: None

OPERANDS: None

DESCRIPTION: Disable the horizontal and vertical cursors.

CURON - TURN CURSOR ON

CODE: 24 ($18)

ATTRIBUTES: None.

OPERANDS: None

DESCRIPTION: Enable the horizontal and vertical cursors.

DELAY - DELAY PROCESSING

CODE: 20 ($14)

ATTRIBUTES: None

OPERANDS: Delay Time - Word

DESCRIPTION: Delay a specified period of time before proceeding to process the next command primitive.

DELAY TIME VALUE:

```
 1 = 1/10 Sec.
10 = 1 Sec.
```

EXAMPLE:

```
DC.W   DELAY
DC.W   10       DELAY 1 SEC.
```

DELS - DELETE A SEGMENT

CODE: 4 ($04)

ATTRIBUTES: None

OPERANDS: Display Segment Number

DESCRIPTION: Completely remove a display segment. The graphics server will no longer recognize this segment.

EXECAS - EXECUTE ALL DISPLAY SEGMENTS

CODE: 6 ($06)

ATTRIBUTES: None

OPERANDS: None

DESCRIPTION: Places all display segments within the valid segment list into the active list. Once each display segment is executed, it is removed from the active list. The command word is valid for visibility (0 = non-visible, non-zero = visible).

EXECS - EXECUTE ONE DISPLAY SEGMENT

CODE: 5 ($05)

ATTRIBUTES: None

OPERANDS: Display Segment Number

DESCRIPTION: Place the display segment into the active list. Once executed, the segment is removed from the list. The command primitive interpreter will check the command word for visibility (0 = non-visible, non-zero = visible).

NCOP - NO COMMAND OPERATION

CODE: 0 ($00)

ATTRIBUTES: None

OPERANDS: None

DESCRIPTION: No operation occurs. The command primitive interpreter moves to the next command primitive word.

OPENS - OPEN A DISPLAY SEGMENT

CODE: 2 ($02)

ATTRIBUTES: OPEN ONLY - $00 Upper Byte

CONTINUOUS EXECUTION - $80 Upper Byte

OPERANDS: Display Segment Number and Address (long word address of display segment, relative to beginning of shared memory segment).

DESCRIPTION: This command places the display segment address in the internal segment table for later reference to that address by its segment number. Before execution of this primitive, the application program must place the display segment control packet into the shared RAM area. If the segment is of the continuous execution type, the segment is made active to the graphics driver and will be constantly polled. The segment command word and status are used by the application program to control actual execution of the graphic primitives. When the command word is non-zero, the graphics driver will use the Continuous Execution Segment Pointer to start graphic primitive interpretation.

DISPLAY SEGMENT CONTROL PACKET:

The application program must set the following attributes/flags:

- Command Word
- Status
- Scale Factor
- X Center
- Y Center
- Color
- Fill Color
- Character Size
- Mask
- Continuous Execution Segment Pointer (if continuous)
- Blink rate (if blink segment)

RBLKS - REMOVE/DELETE BLINK SEGMENT

CODE: 8 ($08)

ATTRIBUTES: None

OPERANDS: Display Segment Number

DESCRIPTION: Completely removes a previously defined blink display segment.

SCFN - SET DISPLAY SEGMENT SCALE FACTOR NUMBER

CODE: 21 ($15)

ATTRIBUTES: NONE

OPERANDS: Segment Number; Scale Factor Number

DESCRITPION: Set the scale factor of a segment after opening or while in use. See section 1.3.3 for a discussion of scale factors.

SYMPTR - SYMBOL TABLE POINTER

CODE: 12 ($0C)

ATTRIBUTES: None

OPERANDS: X-Size; Y-Size; and Address

DESCRIPTION: Identifies a symbol table stored in shared RAM for use by the symbol (SYM) primitive.

WHERE:

X-Size = Word defining number of horizontal pixels/4. (A symbol is modulo 4).

Y-Size = Word defining number of vertical pixels.

Address = Long word pointing to symbol table start (relative to the start of the shared memory segment).

EXAMPLE:

```
DC.W   SYMPTR
DC.W   52          52 x 4 = 208 horizontal pixels per
                   symbol
DC.W   192         192 vertical pixels per symbol
DC.L   $18000      Address of the symbol table
```

APPENDIX B

GRAPHIC PRIMITIVES

This appendix provides a description of each predefined graphic primitive within the graphics driver. Refer to Chapter 1 for a description of how these primitives are used.

The graphic primitives are organized in this appendix as follows:

GRAPHIC PRIMITIVE	CODE	PAGE NUMBER
ACTSN	30 ($1E)	B-2
ARC	9 ($09)	B-3
BITMAP	11 ($0B)	B-4
CALLASM	32 ($20)	B-5
CHARS	18 ($12)	B-6
CHMARK	17 ($11)	B-7
CHMARKS	29 ($1D)	B-8
CHSIZE	14 ($0E)	B-9
CIRCLE	8 ($08)	B-10
COLOR	12 ($0C)	B-11
CURSPTR	15 ($0F)	B-12
DOT	2 ($02)	B-13
DRAWTO	4 ($04)	B-14
FILL	13 ($0D)	B-15
GCUR	19 ($13)	B-16
GENAB	20 ($14)	B-17
GEND	1 ($01)	B-18
GJMP	21 ($15)	B-19
GJSR	22 ($16)	B-20
GRTS	23 ($17)	B-21
GSETD	28 ($1C)	B-22
LINES	5 ($05)	B-23
MASK	27 ($1B)	B-24
MOVETO	3 ($03)	B-25
NGOP	0 ($00)	B-26
PIE	26 ($1A)	B-27
POLYG	6 ($06)	B-28
RECT	7 ($07)	B-29
ROTATE	31 ($1F)	B-30
SCALEF	16 ($10)	B-31
SYM	10 ($0A)	B-32
SYMARK	24 ($18)	B-33
SYMARKS	25 ($19)	B-34

ACTSN - ACTIVATE DISPLAY SEGMENT NUMBER

CODE: 30 ($1E)

ATTRIBUTES: None

OPERANDS: Display Segment Number

DESCRIPTION: Activate a closed segment.

ARC - DRAW AN ARC

CODE: 9 ($09)

ATTRIBUTES: FILL - Shade inside of figure with color specified in the Display Segment Control Packet

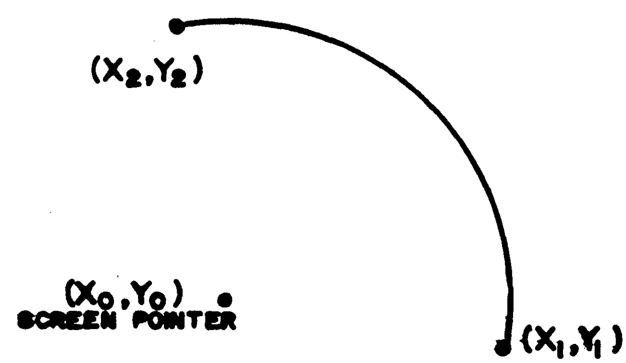
XOR - Exlusive-OR figure to display screen (with color, if filled)

ABSOLUTE/RELATIVE - Absolute or relative

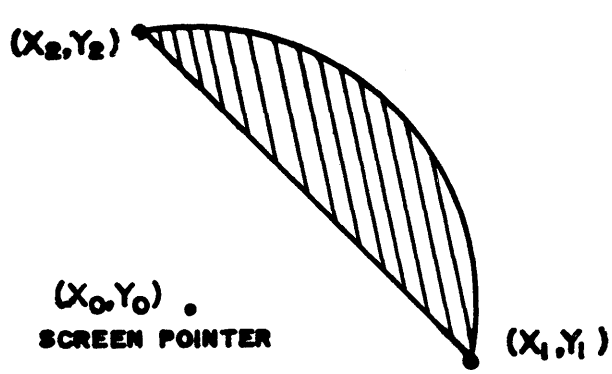
OPERANDS: X_1, Y_1, X_2, Y_2

DESCRIPTION: Draw an arc from X_1/Y_1 to X_2/Y_2 having a center point at the current screen pointer position. Both X_1/Y_1 and X_2/Y_2 must be equidistant from the current screen pointer position (refer to Example 1). The XOR attribute causes the arc to be Exclusive-ORed with the current screen display and the FILL attribute will shade the enclosed arc (Example 2). Arcs are always drawn in a counterclockwise direction from X_1/Y_1 to X_2/Y_2. The screen pointer is left unchanged.

EXAMPLE 1:



EXAMPLE 2:



BITMAP - BIT MAP PICTURE TRANSFERRED

CODE: 11 ($0B)

ATTRIBUTES: XOR - Exclusive-OR bit mapped picture to display screen

OPERANDS: X-Dimension; Y-Dimension

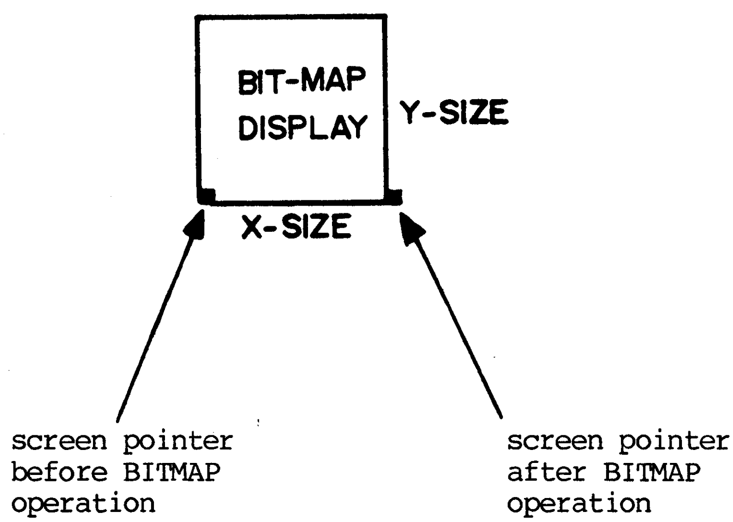
DESCRIPTION: Transfer a bit-mapped picture to the display screen at the current screen pointer position.

WHERE:

X-Dimension = X pixels/4 (modulo 4)
Y-Dimension = Y pixels

The starting address of the bit-mapped picture in shared RAM is established by the BITMPTR command primitive via the command channel. Since each pixel is represented by four bits, each word contains four pixels. The smallest X-Dimension (horizontal) represents four pixels (modulo 4). The screen pointer is left pointing to the lower right-hand corner of the bit-mapped picture.

EXAMPLE:



CALLASM - CALL ASSEMBLY LANGUAGE SUBROUTINE

CODE: 32 ($20)

ATTRIBUTES: None

OPERAND: Long Address of Subroutine (relative to the beginning of the shared memory segment).

DESCRIPTION: A JSR is made to a user-defined assembly language subroutine. The subroutine should return with an RTS instruction. Upon entry, register A5 points to the graphics segment, and A6 points to the next primitive. Unless the user is familiar with the graphics server code, this directive should not be used.

CHARS - CHARACTERS

CODE: 18 ($12)

ATTRIBUTES: XOR - Exclusive-OR characters to the display screen

90 DEG. - Rotate character string 90 degrees

COLOR - Defined in Display Segment Control Packet

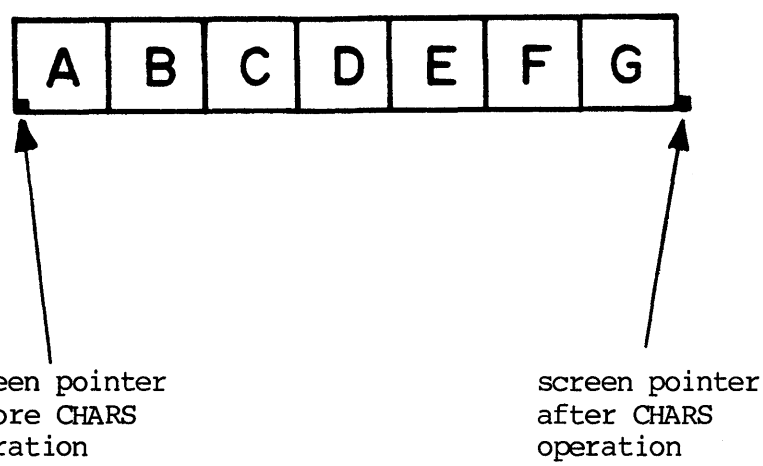
CHSIZE - Defined in Display Segment Control Packet

OPERANDS: Number of Characters followed by packeted ASCII characters.

DESCRIPTION: Present multiple characters on the display screen at the current screen pointer position. The screen pointer is changed to point to the next character position.

EXAMPLE:

```
DC.W  CHARS
DC.W  7
DC.B  'ABCDEFG '   MUST BE EVEN #
```



CHMARK - CHARACTER MARKER

CODE: 17 ($11)

ATTRIBUTES: XOR - Exclusive-OR marker to display screen

90 DEG. - Rotate marker 90 degrees

Color - Defined in Display Segment Control Packet

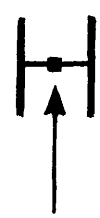
CHSIZE - Defined in Display Segment Control Packet

OPERANDS: Marker character (in high byte of word)

DESCRIPTION: A single character is displayed on the screen, centered on the current screen pointer. The screen pointer is left unchanged.

EXAMPLE:

```
DC.W  CHMARK,'H'
```



current screen pointer position

CHMARKS - MULTIPLE CHARACTER MARKERS

CODE: 29 ($1D)

ATTRIBUTES: XOR - Exclusive-OR markers to display screen

90 DEG. - Rotate markers 90 degrees

ABSOLUTE/RELATIVE - Absolute or relative

Color - Defined Display Segment Control Packet

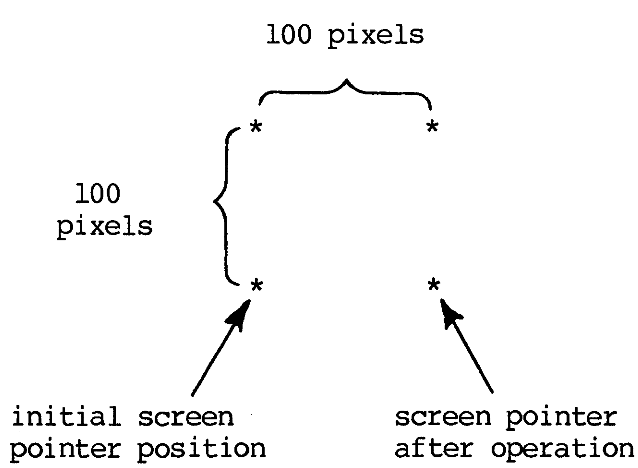
CHSIZE - Defined in Display Segment Control Packet

OPERANDS: Marker Character (in high byte of word); Number of Markers; Starting X,Y Coordinates

DESCRIPTION: Display multiple character markers centered at the specified coordinates. The specified character is displayed on the screen at one or more locations. The locations (coordinates) specified are those of the center of the charater. The coordinates may be specified absolutely or relatively, depending on the attribute bit. In the case of relative coordinates, the first set of coordinates is taken relative to the current screen pointer position, while each of the remaining sets of coordinates is taken relative to the immediately preceding set of coordinates. The screen pointer is left pointing to the center of the last character marker displayed.

EXAMPLE:

```
DC.W  CHMARKS+$1000              Use relative mode
DC.W  '*',4                      Character, number of markers
DC.W  0,0,0,100,100,0,0,-100     Coordinates
```



CHSIZE - SET DEFAULT CHARACTER SIZE

CODE: 14 ($0E)

ATTRIBUTES: SIZE - One of eight charater sizes entered in the Display Segment Control Packet

OPERANDS: None

DESCRIPTION: Establish the default character size for CHARS, CMARK, or CMARKS primitives. CHSIZE is entered in the Display Segment Control Packet.

SIZE	*	CHARACTER (PIXELS)	MATRIX (PIXELS)
0	1	5 x 7	8 x 8
1	2	10 x 14	16 x 16
2	3	15 x 21	24 x 24
3	4	20 x 28	32 x 32
4	5	25 x 35	40 x 40
5	6	30 x 42	48 x 48
6	7	35 x 49	56 x 56
7	8	40 x 56	64 x 64

CIRCLE - DRAW A CIRCLE

CODE:	8 ($08)
ATTRIBUTES:	FILL - Shade inside of figure with color specified in the Display Segment Control Packet XOR - Exlusive-OR figure to display screen (with color, if filled) ABSOLUTE/RELATIVE - Relative Only
OPERANDS:	Radius (in pixels)
DESCRIPTION:	Present a circle on the display screen centered on the position of the current screen pointer and with the radius specified in the operand. (NOTE: The Relative Attribute bit MUST be set to a logic 1 to interpret the operand as the radius.) The screen pointer is left unchanged.

COLOR - DEFINE COLOR ATTRIBUTE

CODE:	12 ($0C)
ATTRIBUTES:	COLOR - One of 8 colors
OPERANDS:	None
DESCRIPTION:	Changes the Color attribute in the Display Segment Control Packet.

COLORS:

Black	= 0
Red	= 1
Blue	= 2
Magenta	= 3
Green	= 4
Yellow	= 5
Cyan	= 6
White	= 7

CURSPTR - MOVE SCREEN POINTER TO CURSOR POSITION

CODE:	15 ($0F)
ATTRIBUTES:	None
OPERANDS:	None
DESCRIPTION:	Obtain the current X and Y (vertical and horizontal) cursor coordinates and move the coordinates to the screen pointer (i.e., change the screen pointer to the current cursor coordinates).

DOT - PLACE A DOT ON THE SCREEN

CODE: 2 ($02)

ATTRIBUTES: XOR - Exclusive - OR dot to display screen

ABSOLUTE/RELATIVE - Absolute or relative

OPERANDS: X-; Y-Coordinates

DESCRIPTION: Display a dot on the display screen at the specified coordinates (X,Y). The color of the dot is provided by the Color attribute in the Display Screen Control Packet. The X,Y coordinate is either absolute or relative, depending on the attribute bit, with scaling indicated by the Scale Factor Word also in the Display Screen Control Packet.

DRAWTO - DRAW LINE TO X,Y

CODE: 4 ($04)

ATTRIBUTES: XOR - Exclusive-OR line to display screen

ABSOLUTE/RELATIVE - Absolute or relative

LINE PATTERN - 0 (solid) - 7

OPERANDS: X-; Y-Coordinate

DESCRIPTION: Draw a line from the current screen pointer to the specified coordinates (X,Y). The coordinate to be drawn to is absolute or relative, depending upon the setting of the attribute bit. Color is determined by the Display Screen Control Packet. The line pattern is determined by the three least significant bits of the attribute. The line may be Exclusive-ORed against any figure currently existing on the display screen.

FILL - SPECIFY FILL COLOR

CODE: 13 ($0D)

ATTRIBUTES: None

OPERANDS: Color

DESCRIPTION: Designates the shade (color) to be used in filling a closed figure by establishing the shade in the Display Screen Control Packet.

COLORS:

Black	= 0
Red	= 1
Blue	= 2
Magenta	= 3
Green	= 4
Yellow	= 5
Cyan	= 6
White	= 7

GCUR - MOVE GRAPHIC CURSOR TO X,Y

CODE: 19 ($13)

ATTRIBUTES: ABSOLUTE/RELATIVE - Absolute or relative

OPERANDS: X-; Y-Coordinates

DESCRIPTION: Move the graphic cursor (vertical and horizontal cursors) to X-,Y-coordinates. Cursor must be ON to be visisble (activate/deactivate cursor using command primitives CURON/CUROFF). The coordinates may be specified absolutely, or as relative to the current screen pointer position (as indicated by the attribute bit). Either way, the screen pointer is left unchanged.

GENAB - GRAPHICS ENABLE

CODE: 20 ($14)

ATTRIBUTES: Pixel Memory Display Enable

OPERANDS: None

DESCRIPTION: Bits 8-10 determine which of the three pixel memory planes are enabled for display. Bit set (1) = enable. Note that this does not preclude read/write operations -- just display.

ENABLE:
- 0 = None, no graphics displayed
- 1 = Red Only
- 2 = Blue Only
- 3 = Red/Blue
- 4 = Green Only
- 5 = Red/Green
- 6 = Green/Blue
- 7 = Red/Green/Blue

GEND - END OF GRAPHIC PRIMITIVE LIST

CODE: 1 ($01)

ATTRIBUTES: None

OPERANDS: None

DESCRIPTION: Entered at the end of a list of graphic primitives in the display program area to terminate execution of the segment.

GJMP - JUMP TO GRAPHICS PRIMITIVE

CODE:	21 ($15)
ATTRIBUTES:	None
OPERANDS:	Long Address (relative to the beginning of the shared memory segment)
DESCRIPTION:	Jump to specified shared RAM long address and continue interpreting graphic primitives.

GJSR - JUMP TO GRAPHICS PRIMITIVE SUBROUTINE

CODE: 22 ($16)

ATTRIBUTES: None

OPERANDS: Long Address (relative to the beginning of the shared memory segment)

DESCRIPTION: Jump to a specified graphic primitive subroutine and continue interpreting graphic primitives. Use GRTS primitive to return.

GRTS - RETURN FROM GRAPHICS PRIMITIVE SUBROUTINE

CODE: 23 ($17)

ATTRIBUTES: None

OPERANDS: None

DESCRIPTION: Provides return from a graphic primitive subroutine. MUST be used if the subroutine was accessed via the GJSR primitive.

GSETD - SET GRAPHIC DISPLAY PAGE

CODE: 28 ($1C)

ATTRIBUTES: COLOR - Specified in Upper Byte

OPERANDS: None

DESCRIPTION: Set entire display page to the specified color. Note that a color of 0 (command word = $001C) will clear the entire display page.

COLORS:

Black	= 0
Red	= 1
Blue	= 2
Magenta	= 3
Green	= 4
Yellow	= 5
Cyan	= 6
White	= 7

LINES - DRAW CONNECTING LINES

CODE:	5 ($05)
ATTRIBUTES:	XOR - Exclusive-OR lines to display screen
	ABSOLUTE/RELATIVE - Absolute or relative
	LINE PATTERN - 0 (solid) - 7
OPERANDS:	Number of lines; X- and Y-Coordinates of Each End Point
DESCRIPTION:	Draw successive connecting lines starting from the current position of the screen pointer. The first Operand must be the number of lines to be drawn, followed by the X- and Y-coordinate word pairs defining the end of each line (and the beginning of the next). The screen pointer is left pointing at the end of the last line drawn.

MASK - SET MASK ATTRIBUTE

CODE: 27 ($1B)

ATTRIBUTES: MASK PLANES - Specify bit-planes

OPERANDS: None

DESCRIPTION: The mask specifies which of the three bit-planes will be written. One or more bit-planes, in any combination, can be enabled. This primitive sets the Display Segment Control Packet attribute mask.

MASK:
- 0 = None - Cannot write to bit-planes
- 1 = Red Only
- 2 = Blue Only
- 3 = Red/Blue
- 4 = Green Only
- 5 = Red/Green
- 6 = Green/Blue
- 7 = Red/Green/Blue

MOVETO - MOVE TO X,Y

CODE: 3 ($03)

ATTRIBUTES: ABSOLUTE/RELATIVE - Absolute or relative

OPERANDS: X-; Y-Coordinates

DESCRIPTION: Move the screen pointer, without drawing, to the specified X-,Y-coordinates. These coordinates are absolute or relative positions, depending upon the attribute bit and scaled to the scale factor defined in the Display Segment Control Packet.

NGOP - NO GRAPHIC OPERATION

CODE:	0 ($00)
ATTRIBUTES:	None
OPERANDS:	None
DESCRIPTION:	No operation occurs. The graphic primitive interpreter moves to the next graphic primitive word.

PIE - DRAW PIE-SHAPED FIGURE

CODE: 26 ($1A)

ATTRIBUTES: FILL - Shade inside of figure with color specified in the Display Segment Control Packet

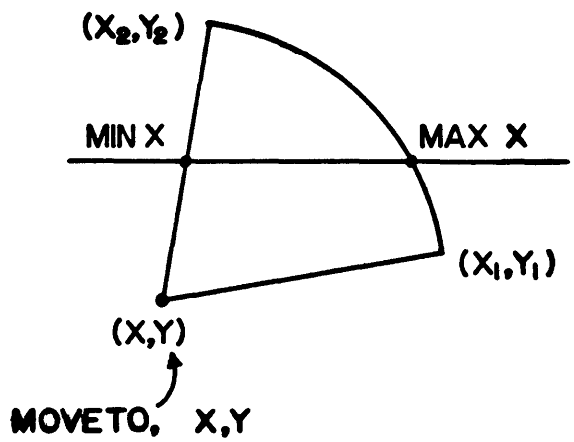
XOR - Exlusive-OR figure to display screen (with color, if filled)

ABSOLUTE/RELATIVE - Absolute or relative

OPERANDS: X_1, Y_1, X_2, Y_2

DESCRIPTION: Draw a pie-shaped segment of a circle having its center point at the current position of the screen pointer. If the figure is to be color filled, only one minimum X and one maximum X may exist for each horizontally intersecting line. The arc of the pie-shaped segment is always drawn counterclockwise from X_1/Y_1 to X_2/Y_2. The screen pointer is left unchanged.

EXAMPLE:



POLYG - DRAW A POLYGON FIGURE

CODE: 6 ($06)

ATTRIBUTES: FILL - Shade inside of figure with color specified in the Display Segment Control Packet.

LINE PATTERN - 0 (solid) - 7 (defines the outline of the polygon)

XOR - Exclusive-OR figure to display screen (either FILL or PATTERN)

ABSOLUTE/RELATIVE - Absolute or relative

OPERANDS: Number of Lines; X- and Y-Coordinates of Each End Point

Draw a polygon (programmer must close on starting point). The number of sides (lines) must be indicated in the Operand field followed by the X- and Y-coordinates of each end point (and starting point of the next if not last). The last coordinate MUST be the same as the starting point to close the figure. The screen pointer is left pointing to the last coordinate specified.

RECT - DRAW A RECTANGLE

CODE: 7 ($07)

ATTRIBUTES: FILL - Shade inside of figure with color specified in the Display Segment Control Packet.

XOR - Exclusive-OR figure to display screen (either FILL or PATTERN)

ABSOLUTE/RELATIVE - Relative Only

OPERANDS: X length; Y length

DESCRIPTION: Draw a rectangle starting from the current position of the screen pointer to the right (X-coordinate) and then up (Y-coordinate). The Relative attribute bit MUST be set. The screen pointer is left unchanged.

ROTATE - ESTABLISH A ROTATION ANGLE

CODE: 31 ($1F)

ATTRIBUTES: ABSOLUTE - Absolute rotation angle

RELATIVE - Angle relative to the previous angle

OPERANDS: Rotation angle in degrees. A positive value is a counter-clockwise rotation. A negative value is a clockwise rotation.

DESCRIPTION: Rotate all subsequent X,Y points relative to the starting point. Only relative points will be rotated. The rotation affects only the (X,Y) points and not subsequent patterns such as the individual pixels of a text character. Note that rectangles may not be rotated, while polygons may.

SCALEF - SET THE SCALE FACTOR

CODE: 16 ($10)

ATTRIBUTES: None

OPERANDS: Scale Factor Number

DESCRIPTION: Set the scale factor in the Display Segment Control Packet. See section 1.3.3 for a discussion of scale factors.

SYM - DISPLAY SYMBOLS

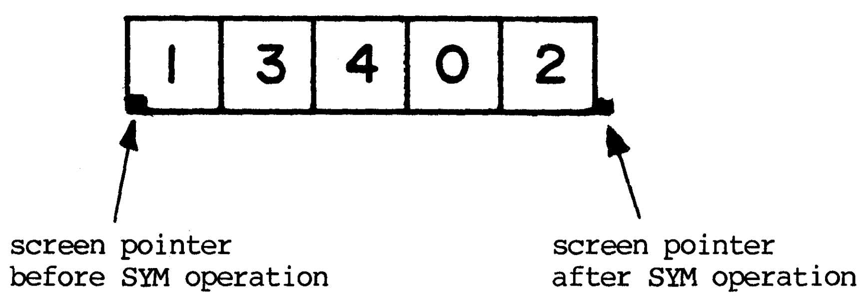
CODE: 10 ($0A)

ATTRIBUTES: XOR - Exclusive-OR symbol to display screen

OPERANDS: Number of Symbols to Follow; Packeted Symbol Offsets Into the Table (one offset per byte)

DESCRIPTION: Display symbols from the symbol table, starting at the current screen pointer position. Symbol numbers (offsets) start at 0 (i.e., 0 = first symbol in symbol table, 1 = second symbol, etc.). The symbol offsets are specified one per byte, with enough bytes to make an integral number of words. The screen pointer is left pointing at the lower right-hand corner of the last symbol displayed.

```
EXAMPLE:  DC.W   SYM
          DC.W   $05           Number of symbols
          DC.B   1,3,4,0,2,0   Symbol offset (one per byte)
```



WHERE SYMBOL OFFSET:

1 Displayed First
3 Displayed Second
4 Displayed Third
0 Displayed Fourth
2 Displayed Fifth
0 Last Zero Is Ignored

The size of the symbol and the location of the symbol table MUST have been previously declared using the SYMPTR command primitive.

SYMARK - SYMBOL MARKER

CODE: 24 ($18)

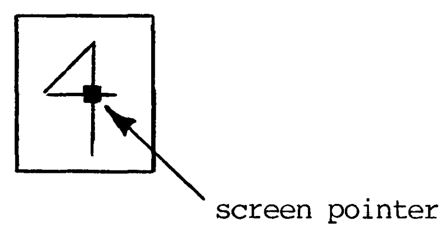
ATTRIBUTES: None

OPERANDS: Symbol Number

DESCRIPTION: Draw a symbol centered on X- and Y-coordinates. The symbol indicated by the Operand is drawn, centered at the current screen pointer position. The screen pointer is left unchanged.

EXAMPLE:

```
DC.W  SYMARK,4
```



SYMARKS - MULTIPLE SYMBOL MARKERS

CODE: 25 ($19)

ATTRIBUTES: XOR - Exclusive-OR marker to display screen

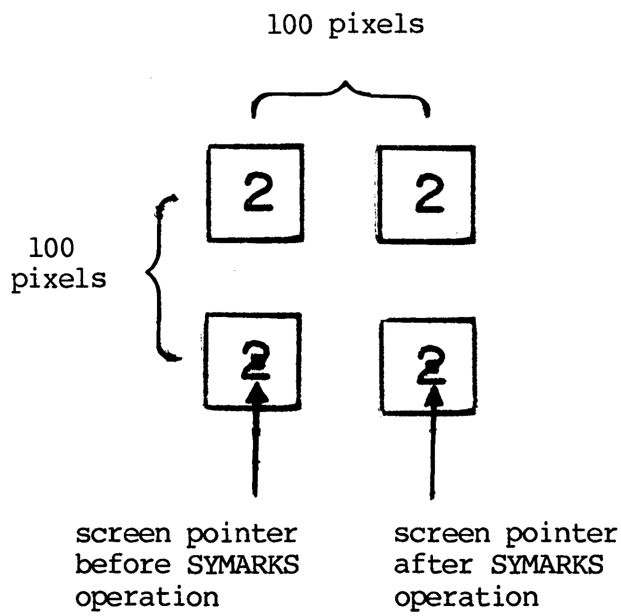
ABSOLUTE/RELATIVE - Absolute or relative

OPERANDS: Symbol Number; Number of Symbols; X- and Y-Coordinates of Each

DESCRIPTION: Draw multiple symbols centered at each of the specified X- and Y-coordinate addresses. The specified symbol is displayed on the screen at one or more locations. The locations (coordinates) specified are those of the center of the symbol. The coordinates may be specified absolutely or relatively, depending on the attribute bit. In the case of relative coordinates, the first set of coordinates is taken relative to the current screen pointer position, while each of the remaining sets of coordinates is taken relative to the immediately preceding set of coordinates. The screen pointer is left pointing to the center of the last symbol marker displayed.

EXAMPLE:

```
DC.W  SYMARKS+$1000           Use relative mode
DC.W  2,4                     Symbol #2, 4 symbols
DC.W  0,0,0,100,100,0,0,-100  Coordinates
```



SUGGESTION/PROBLEM REPORT

MICROSYSTEMS
QUALITY • PEOPLE • PERFORMANCE

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microsystems
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: ______________________ Manual: ______________________

COMMENTS: ____

____

____

____

____

____

____

____

____

____

____

____

Please Print

Name ______________________ Title ______________________

Company ______________________ Division ______________________

Street ______________________ Mail Drop ____________ Phone ____________

City ______________________ State ______________ Zip ____________

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Four Phase/Motorola Customer Support, Tempe Operations
(800) 528-1908
(602) 438-3100

MOTOROLA

MOTOROLA ***Semiconductor Products Inc.***

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

16863-1 PRINTED IN USA (3/84) MPS 3M