RASTER TECHNOLOGIES
MODEL ONE/25
PROGRAMMING GUIDE

Raster Technologies

RASTER TECHNOLOGIES
MODEL ONE/25
PROGRAMMING GUIDE

Revision 4.0 March 31, 1983

Raster Technologies Model One/25 Programming Guide
March 31, 1983

This manual applies to Revision 4.0 (and beyond) of the Model One firmware.

List of Figures

List of Examples

List of Tables

1.0 INTRODUCTION

This manual describes, in detail, the standard commands of the Raster Technologies Model One/25. It is intended for the experienced graphics applications programmer, and assumes that the user has read the Introduction to the Raster Technologies Model One.

Before beginning this manual, the user should verify:

1. that the Model One/25 has been installed and tested on the host computer system, and

2. that graphics commands may be successfully executed locally at the alphanumeric terminal and from the host computer system.

Detailed installation and testing instructions are included in the Installation Guide for the Raster Technologies Model One.

This manual has two parts: the first fifteen sections present the Model One and its commands in conceptual groups, using a tutorial structure; Section 16 provides an alphabetical command reference to all the Model One commands. Section 16 is intended for reference and gives complete details of every command. Finally, Section 17 provides a quick reference to the Model One commands.

Each Section is described below.

1.0 Introduction

2.0 Modes of Operation: this section describes the two operating modes for the Model One: ALPHA mode, in which the programmer communicates in alphanumerics with the host computer, and GRAPHICS mode, in which the Model One's command interpreter is used to decode and execute graphics commands. (Graphics commands may originate from the host or from the local terminal.)

3.0 Coordinates and Image Memory Addressing: this section describes the Model One's two-dimensional coordinate addressing. The coordinate origin, clipping window, screen origin, current point, and coordinate registers are described in detail.

4.0 Pixel Values, Look-Up Tables, and Image Memory: this section explains the use of look-up tables (LUTs), value registers, and write- and read-enable masks. Pixel values and the pixel processor are explained. Blinking colors and the blink tables are described; the CLEAR and FLOOD commands are also explained.

5.0 1024x1024 Addressing Mode: this section explains how to use the Model One/25 in 1K mode.

6.0 Graphics Primitives: this section describes the graphics primitives supported by the Model One: points, lines (vectors), circles, arcs, rectangles, and polygons. The use of text and the text commands is given in detail.

7.0 Dual-Overlay Planes: this section describes the Model One/25 dual-overlay planes and the associated commands. This section should be used only by programmers whose systems include the Option Card.

8.0 Pixel Mover: this section explains the Pixel Mover option, and should be used only by those programmers whose system includes the Option Card.

9.0 Data Read-Back and Image Transmission: this section describes loading image memory from the host computer and reading back information to the host computer.

10.0 Macro Programming: this section explains how to define, write, and execute a Model One macro program. Use of the button table, which may be used to execute macros under user control, is covered.

11.0 Application Development Features: this section describes the Model One local debugger, the command stream translator, and the REPLAY command (which allows the user to play back the last 32 characters sent from the host to the Model One).

12.0 Programming the Z8000: this section describes the commands associated with downloading and debugging Z8000 object code. This section will be needed only by those programmers who are adding commands to the Model One's command set.

13.0 Host FORTRAN Library: this section describes the Model One's host FORTRAN library in detail.

14.0 Host Computer DMA: this section describes the use of the Option Card DMA (Direct Memory Access) port for high-speed transfers between the host computer and the Model One's image memory. This section should be used only by those programmers whose system includes the Option Card.

15.0 Error Conditions: this section lists the possible error messages and their causes. The possible responses are given.

16.0 Alphabetical Command Reference: this section provides a complete reference to every Model One/25 command. Organized alphabetically, it supplies full details of all commands.

17.0 Quick Reference: this section supplies a brief listing of all Model One commands. In addition, it includes a listing of the graphics commands by opcode.

2.0 MODES OF OPERATION

The Model One functions in two modes: GRAPHICS mode, in which the Model One command interpreter decodes and executes graphics commands, and ALPHA mode, in which ASCII characters are passed through the Model One to connect the local alphanumeric terminal and the host computer.

2.1 ALPHA Mode

When the Model One is powered up, or if the RESET button mounted on the rear panel is pressed, a COLDstart command is executed. After COLDstart, the Model One is in ALPHA mode. You can then use the local alphanumeric terminal or keyboard to communicate directly with the host computer.

The Model One's COLDstart command, executed by pressing the RESET button or by entering the COLD command directly, performs a complete COLDstart on the Model One. The COLDstart includes clearing defined macros, coordinate and value registers, resetting the clipping window, and setting the Look-Up-Tables to the default. Section 16 gives complete details of the COLDstart command. You should execute a COLDstart command after each example in this manual to reset the Model One.

The Model One supports three different host transmission formats: 8-bit binary, ASCII hexadecimal, and pure ASCII. The choice between 8-bit binary and ASCII hexadecimal is made at installation, as described in the Model One Installation Guide. ASCII hexadecimal format is used when the host computer cannot be programmed to transmit 8 bits of binary data over each character sent to the terminal.

Pure ASCII format allows the host computer to issue graphics commands in exactly the same format as commands typed at the local terminal. The ASCII flag command is used to set the host interface for pure ASCII format; all subsequent commands must be sent from the host to the Model One exactly as they would be typed in locally.

Pure ASCII format requires many more characters to be sent from the host to execute a series of commands and should be used only when the command stream must be directly interpreted by the programmer or user rather than the Model One.

2.2 GRAPHICS Mode

In GRAPHICS mode, the Model One command interpreter decodes and executes graphics commands coming from the local terminal or from the host computer. To enter GRAPHICS mode, a [CTRL-D] (04H or 84H) is sent from the host computer or from the local terminal. The [CTRL-D] must be sent as a 7-bit ASCII character, independent of whether the the Model One has been set to accept data in 8-bit binary or ASCII hex. The SPCHAR (see section 2.3) command can be used to change the ENTERGRAPHICS control character from a CTRL-D to any desired ASCII code. In fact, the SPCHAR command can be used to change any Model One special control character to any user-defined ASCII code.

Only one I/O port, either the port to the local terminal (ALPHASIO port) or one of the ports to the host computer (HOSTSIO, HOSTGPIB, or HOSTDMA), can be in GRAPHICS mode at any given time. When the ENTERGRAPHICS control character appears at one of these ports, the Model One enters GRAPHICS mode. Once the Model One is in GRAPHICS mode, the Model One expects graphics commands from the host computer or from the local terminal, whichever one initially issued the ENTERGRAPHICS code.

2.2.1 GRAPHICS Mode From the Local Terminal

When the ENTERGRAPHICS control character appears at the Model One's ALPHASIO port, the local terminal is put into GRAPHICS mode; the GRAPHICS prompt character ! is displayed at the terminal. Now, commands that you type are processed directly by the Model One and not sent to the host computer.

The format for graphics commands typed locally is:

```
COMMAND_MNEMONIC  parameter1 ... parameterN
```

For example, the command to draw a circle of a given radius around the current point is:

```
CIRCLE 40
```

where CIRCLE is the command mnemonic, and 40 is the desired radius. Parameter values may be entered as signed, base-10 numbers (such as 117 or 45), or as unsigned hexadecimal numbers when preceded by a # sign (such as #FF, #9E, or #0A0F).

The parameter values must be separated by a comma or space, as desired. This can be used to set off sections of the commands, or to clarify what the command is doing:

```
ARC 10 45,135
```

This command draws an arc of radius 10, with starting angle 45 degrees and ending angle 135. You can also use commas to set off x,y pairs or groups of values, as you will see in the examples in this manual. Some further examples of locally-typed commands are:

```
MOVABS 0,0
DRWREL 30 150
VECPAT #F0F0
```

The Model One command interpreter includes a HELP subsystem. To receive a list of all the Model One commands, type

```
HELP
```

HELP may also be used to obtain parameter information, by typing

```
HELP COMMAND
```

For example, typing

```
HELP MOVABS
```

displays this information:

```
     MOVABS: OPCODE = 001
  1. <16 BIT SIGNED NUMBER>
  2. <16 BIT SIGNED NUMBER>
```

and typing

```
HELP PRMFIL
```

displays

```
     PRMFIL: OPCODE = 031
  1. OFF ON
```

Abbreviations may be used for the command mnemonic; these abbreviations are indicated in the Alphabetical Command Listing (section 16). The shortest abbreviation allowed is indicated by underscores: CIRCLE indicates that C is a valid abbreviation for the CIRCLE command. Additional characters may be used, if desired: C, CI, CIR, CIRC, and CIRCL are all valid abbreviations.

The QUIT command is used to leave GRAPHICS mode and return to ALPHA mode. Type the ENTERGRAPHICS control character [CTRL-D] and then a QUIT command, to verify that the Model One returns to ALPHA mode. In ALPHA mode, you can once again use your terminal to communicate with the host computer.

2.2.2 GRAPHICS Mode From the Host Computer

When the ENTERGRAPHICS character is sent from the host computer to the Model One, the host port which sent the ENTERGRAPHICS character, whether it was the HOSTSIO, HOSTGPIB, or HOSTDMA port, is put into GRAPHICS mode.

A host computer application program, using the Model One's Host FORTRAN library, sends the ENTERGRAPHICS control character by calling the subroutine ENTGRA. All graphics commands from the host computer to the Model One must be sent between a call to the ENTGRA subroutine and a call to the QUIT subroutine, as shown in the example below. (The host FORTRAN library is described in more detail in section 13.)

```
INTEGER I,J,K,L
INTEGER IRAD,IANG,IJANG    Model One in ALPHA Mode
    :                      No graphics commands may be
    :                       issued until a CALL ENTGRA
    :                       command is given.
    :                      Alphanumeric I/O is allowed.
    :
CALL ENTGRA                Enter GRAPHICS Mode.
    :
```

```
     :
CALL MOVABS(I,J)          Graphics subroutine calls are
CALL CIRCLE(M)             allowed: no alphanumeric I/O
CALL DRWABS(K,L)           to Model One.
     :
     :
CALL QUIT                 Quit GRAPHICS Mode; return to
     :                     ALPHA Mode.  Alphanumeric data
     :                     to local terminal allowed.
     :
CALL ENTGRA               Reenter GRAPHICS Mode.
     :
     :
CALL CIRCLE(IRAD)
CALL ARC(IRAD,IANG,JANG)
CALL DRAWBS(J,K)
     :
     :
CALL QUIT                 Exit GRAPHICS Mode.
STOP
END
```

If you are not using the Model One's host FORTRAN library, your program will have to send the ENTERGRAPHICS control code to the Model One in some other way. You should verify that your host programs can send the ENTERGRAPHICS control code and properly issue graphics commands before continuing. Graphics commands from the host computer are composed of a stream of opcodes and parameters. Each opcode is one byte, ranging from 00H to FFH. The opcode for a MOVABS command is 01H. The opcode for a CIRCLE command is 0EH. The MOVABS command has four bytes of parameter data which must immediately follow its opcode. Thus, the byte stream for a complete MOVABS command would be:

```
01H      03H FFH 00H 0FH
opcode   parameters
```

This host command stream is equivalent to typing:

```
MOVABS #03FF,#000F
```

or

```
MOVABS 1023,15
```

at the local alphanumeric keyboard.

The CIRCLE command has one two byte parameter which gives the radius of the circle to be drawn. Thus, the CIRCLE command would be:

```
0EH      01H 03H
opcode   parameters
```

when sent from the host computer. This command is equivalent to typing:

CIRCLE #0103

or

CIRCLE 259

at the local alphanumeric terminal.

The QUIT command, which returns the Model One to ALPHA mode, has an opcode of FFH (with no parameters). Each command stream to the Model One which is to be followed by normal terminal I/O must end with a QUIT command. Thus, a complete command stream to the Model One would be:

04H (or 84H)	01H	03H FFH 00H 00H	0EH	01H 03H	FFH
ENTERGRAPHICS	opcode	parameters	opcode	parameters	QUIT

Note that the ENTERGRAPHICS control code may or may not have its high bit set; it is interpreted as a 7-bit ASCII code.

If the Model One has been configured to accept 8-bit binary host transmission (see the Model One Installation Guide), each byte in the command stream is sent in a single character from the host computer. For 8-bit binary to work properly, the host computer may not use the eighth bit of each character for parity or force it high or low. It also must allow every control code to pass to the terminal. In addition, the host computer must not insert carriage control characters unpredictably into the output stream.

ASCII hex format removes all of these restrictions at the cost of doubling the transmission time. In ASCII hex format, each byte is expanded into two hexadecimal characters, one for the high nibble (four bits) of the byte, followed by a second character for the low nibble of the byte. The byte stream

01H 03H FFH 00H 0FH

requires only five characters when using binary transmission. In ASCII hex, it would require ten characters:

"0" "1" "0" "3" "F" "F" "0" "0" "0" "F"

In ASCII hex format, all carriage control characters are ignored, so that a FORTRAN WRITE statement or a BASIC PRINT statement will work properly.

2.2.3 The Model One's Input and Output Buffers

While the local terminal is in GRAPHICS mode, any characters sent by the host to any of the Model One's host ports (HOSTSIO, HOSTDMA, or HOSTGPIB) are stored in the Model One's input queue until you type QUIT to reenter ALPHA mode. The HOSTSIO input queue defaults to 4096 bytes (characters). The HOSTDMA input buffer is 2 characters.

Similarly, while the host port is in GRAPHICS mode, any characters typed at the local alphanumeric terminal are stored in the ALPHASIO input queue until the host computer issues a QUIT command to exit GRAPHICS mode. The ALPHASIO input queue holds 256 characters.

The ALPHASIO and HOSTSIO input queues can be changed with the CONFIG command; details are given in section 16.0.

Table 2.1 shows the default I/O buffer sizes for the Model One.

Port	Input	Output
HOSTSIO	4096	256
HOSTDMA	2	256
ALPHASIO	256	16
HOSTGPIB	0	0

Table 2.1 Default I/O Buffer Sizes

2.3 The Model One's Special Characters

The Model One responds to a set of special control characters, such as the ENTERGRAPHICS [CTRL-D] described above, to perform certain functions. These are:

Function	Default Special Character
Enter GRAPHICS Mode	[CTRL-D]
Send BREAK to host	[CTRL-P]
Execute WARMstart	[CTRL-[] or [ESC]
Kill current line	@
Backspace	[CTRL-H]
ACKnowledge	[CTRL-F]
Negative ACKnowledge	[CTRL-U]
Enter local debugger	[CTRL-X]
Suspend communications	[CTRL-S]
Restart communications	[CTRL-Q]

Table 2.2 The Model One's Special Characters

Note that [CTRL-S] and [CTRL-Q] are not sent to the host if you are not using the HOSTSIO port. They do, however, stop and start communications to the local terminal.

The SPCHAR char,flag,code command can be used to change the default special characters for the Model One. The most commonly modified default special character is the WARMstart character, which defaults to an ESCAPE. The SPCHAR command to change the WARMstart character to a [CTRL-G] (the bell) is:

```
SPCHAR 2,1,7
```

The command

```
SPCHAR 2,0,0
```

disables the WARMstart character entirely. (While this is dangerous during program development, it may be desirable after debugging has been completed.) The SPCHAR command is described in detail in Section 16.0.

3.0 COORDINATES AND IMAGE MEMORY ADDRESSING

The Model One uses a two-dimensional coordinate system to describe the graphic entities that are drawn into image memory. Each coordinate is stored as an X component and a Y component; these components are stored within the Model One as two's complement 16-bit integers. The Model One's graphics commands use this 16-bit "address space" to specify the position of points, lines, circles, arcs, polygons, rectangles, and so on.

Because the physical image memory of the Model One is not large enough to allow a full 16-bits of addressing in both the X and Y dimensions, the physical image memory covers only a patch of the 16-bit address space, ranging from (-256,-256) to (255,255) in the Model One/25 in 512x512 addressing mode and (-512,-512) to (511,511) in 1024x1024 addressing mode. (The 1024x1024 addressing mode--also called 1K mode--is covered in more detail in section 5.0.) The patch of image memory is called the physical image memory, as it is that section of the virtual 16-bit address space that can be written into when making an image.

3.1 The Coordinate Origin

The CORORG x,y command can be used, immediately after a COLDstart, to reposition the physical image memory in the 16-bit address space. Figure 3.1 shows the default coordinate system for the Model One/25. Figure 3.2 shows the use of the CORORG command to set up the screen so that (0,0) is in the lower-left hand of the screen and all points can be addressed as positive numbers.

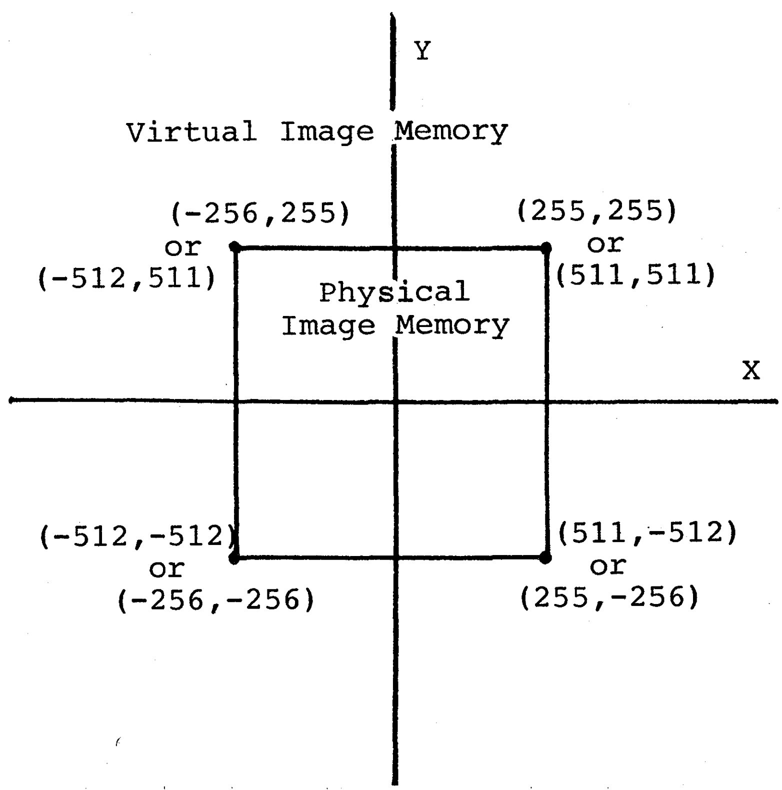

Figure 3.1 The Default Addressing Space

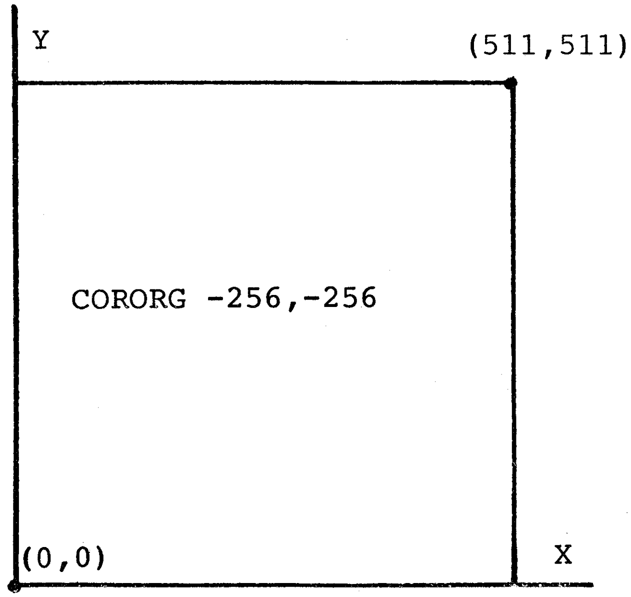

Figure 3.2 Resetting the Coordinate Origin

The CORORG command should be issued only immediately following a COLDstart, because all coordinate registers are modified by the CORORG command.

3.2 The Clipping Window

When a graphic primitive, such as a line or circle, is to be drawn, its coordinates are given as 16-bit addresses. When the primitive is then drawn by the Model One, it is clipped so that it is drawn only into the physical image memory.

The Model One automatically clips all graphics primitives to a preset clipping window. If no clipping window was specified, it draws only that portion of all graphics primitives which lie in physical image memory. To support this clipping, the Model One maintains a clipping window; the clipping window defines a rectangular area of the virtual address space outside of which nothing is drawn. The default clipping window is defined by the physical image memory. In the Model One/25, the default clipping window has the corners (-256,-256), (-256,255), (255,255), (255,-256). In the Model One/25 1K mode, the corners are (-512,-512), (-512,511), (511,511), and (511,-512).

The WINDOW x1,y1 x2,y2 command changes the position of the clipping window by respecifying the lower-left and upper-right corners of the window. For example, the command WINDOW 0,0 255,255 defines the clipping window with a lower-left corner of (0,0) and an upper-right corner of (255,255). Positioning all or part of the clipping window outside of the physical image memory clips the window itself, as clipping occurs automatically beyond image memory bounds.

3.3 The Screen Origin

Ordinarily, the video monitor displays the entire contents of image memory. The displayed image is essentially a window into image memory, however, and it can be modified in both size and position.

The screen origin specifies the physical image memory location that will appear at the center of the screen; it may be placed on 4-pixel boundaries horizontally and 2-pixel boundaries vertically. The default screen origin is (0,0).

The SCRORG x,y command is used to change the position of the screen origin within image memory. For example, SCRORG -40,-40 puts the point (-40,-40) at the center of the screen. If the image is large enough, wrapping around of the image can be seen when the command is executed. The Model One automatically wraps the image around (side-to-side and top-to-bottom panning) when any part of the screen falls off the edge of physical image memory, as the screen refresh addresses are generated modulus 512 for 512 addressing mode or modulus 1024 for 1K addressing mode. This wraparound cannot be disabled.

3.4 Display Scale Factor

The display scale factor determines the number of pixels that are displayed on the screen:

Scale Factor	512 Mode	1K Mode
1	512x512	1024x1024 (averaged to 512x512)
2	256x256	512x512
4	128x128	256x256
8	64x64	128x128

3.5 Zooming the Display

The size of the window of image memory that is displayed is controlled by the ZOOM and ZOOMIN commands, which modify the display scale factor.

The ZOOM factor command allows explicit definition of the display scale; the display scale may be 1, 2, 4, or 8. For example, ZOOM 4 sets the display scale to 4.

The ZOOMIN command zooms in the display by a factor of two; ZOOMIN sets the scale to 4 if the current scale is 2, or 8 if the current scale is 4. Finally, if the current scale is 8, the display is restored to a scale factor of 1 by a ZOOMIN command.

ZOOM and ZOOMIN do not change the current screen origin, clipping window, or coordinate origin.

When you zoom in the display, the number of pixels displayed on the screen in reduced, so that each pixel takes up a larger display area. At a zoom scale factor of eight, for example, you can see each pixel quite clearly.

To zoom in a specific portion of the image, you should move that portion of the image to the center of the screen, using the SCRORG command, then execute the zoom command.

3.6 Inhibiting Screen Refresh

The BLANK flag command totally inhibits screen refresh, leaving image memory available all of the time for updates by the vector generator, pixel processor, or optional host DMA. BLANK 1 inhibits screen refresh: BLANK 0 restores normal screen refresh. When screen refresh is inhibited, the displayed image is forced to black.

The BLANK command can be used to increase the pixel writing rate, since more time is available for vector writing when screen refresh is inhibited.

3.7 The Current Point

All Model One commands which draw graphics primitives use the current point as a reference. For example, the CIRCLE command draws a circle of given radius around the current point.

To draw a line in image memory from a given starting point to a specified ending point, the current point must first be set to the starting point of the line. Then, the line is drawn from the current point to the specified ending point. The line-drawing commands leave the end point of the line as the new current point after the line is drawn.

Five commands move, and therefore modify, the current point: MOVABS, MOVREL, MOV3R, MOV2R, and MOVI.

The MOVABS x,y command specifies a new current point: MOVABS -10,-10 sets the current point to (-10,-10).

The MOVREL dx,dy command moves the current point a relative distance from the previous current point. For example, the command sequence

```
MOVABS 143,271
MOVREL -10,-10
```

would place the current point at (133,261).

The MOV3R dx,dy and MOV2R dx,dy commands are special forms of the MOVREL command for use when the displacement to the new current point is small. The MOV2R command requires only two bytes to be sent from the host computer; the MOV3R command requires three bytes. The MOVABS and MOVREL commands require five bytes when sent from the host. Details of the MOV2R and MOV3R commands are given in Section 16.0.

The MOVI creg command moves the current point to the point specified by the indicated coordinate register. For example, MOVI 2 moves the current point to the point specified by coordinate register 2 (coordinate register 2 gives the location of the cursor on the digitizing tablet). Details of the coordinate registers are given in the next section.

3.8 The Coordinate Registers

The Model One stores 64 coordinate registers internally. The coordinate registers store coordinate values within the Model One: some have a predefined function within the Model One, others are available for programmer use.

Each coordinate register (CREG) stores a 16-bit X coordinate and a 16-bit Y coordinate. Table 3.1 shows the coordinate register assignments for the Model One/25.

Coordinate Register	Function
0	Current point: used as a reference point by graphics commands. The current point is modified by MOVE and DRAW commands.
1	Joystick or trackball location, updated automatically by the Model One every 1/30th second.
2	Digitizing tablet cursor location, updated automatically by the Model One every 1/30th second.
3	Coordinate origin: used to position physical image memory within the virtual address space. The coordinate origin is modified by the CORORG command.
4	Screen origin: specifies the point which appears at the center of the screen. The screen origin is changed by the SCRORG command. CREG 4 is used for horizontal and vertical panning.
5	Crosshair 0 current location: changes made to this register move crosshair 0. The crosshair is enabled/disabled using the XHAIR command (see section 3.9).

Table 3.1 Coordinate Register Assignments
(continued on next page)

Table 3.1 Coordinate Register Assignments (continued)

Coordinate Register	Function
6	Crosshair 1 current location: changes made to this register move crosshair 1. The crosshair is enabled/disabled using the XHAIR command.
7	For Option Card users only: crosshair 2 current location. Crosshair 2 is displayed on overlay plane 0, using the XHAIR command.
8	For Option Card users only: crosshair 3 current location. Crosshair 3 is displayed on overlay plane 1, using the XHAIR command.
9	Clipping window, lower-left corner.
10	Clipping window, upper-right corner.
11,12	For Option Card users only: diagonal corners for source window for PIXMOV command.
13	For Option Card users only: PIXMOV destination window.
14	For Option Card users only: pixel writing direction for PIXMOV destination window.
15	For Option Card users only: screen origin of overlay plane 0.
16	For Option Card users only: screen origin of overlay plane 1.
17-19	Reserved.
20-63	Available for use by applications programmer.

Table 3.1 Coordinate Register Assignments

Four commands load and alter the coordinate registers: CLOAD, CMOVE, CADD, and CSUB. The command READCR reads back or displays the contents of a specified coordinate register.

The CLOAD creg x,y command loads a given 16-bit X coordinate and a 16-bit Y coordinate into the specified coordinate register. For example, CLOAD 25 -75,75 loads the point (-75,75) into coordinate register 25.

The CMOVE cdst, csrc command copies data from one coordinate register into another: CMOVE 0 2 moves the contents of coordinate register 2 (the cursor location) into coordinate register 0 (the current point). The command CMOVE 0 2 thus specifies that the new current point is to be taken from the cursor location on the digitizing tablet.

CADD csum, creg and CSUB cdif, creg add and subtract coordinates between two specified coordinate registers: CADD 0 21 adds the contents of CREG 21 to the contents of CREG 0.

Note that in the coordinate register pairs specified as parameters for CMOVE, CADD, and CSUB, the register which is to be modified is specified first.

Several graphics primitive commands include an indirect addressing form. In this form, coordinates which are needed to execute the command are given by specifying a coordinate register instead of being supplied directly: MOVI moves to the point given by a coordinate register, RECTI uses a coordinate register to specify the diagonal corner of a rectangle, and so on.

Finally, the command READCR creg reads or displays the contents of a specified coordinate register. For example, READCR 0 displays the contents of CREG 0 (the current point).

3.9 The Crosshairs

The XHAIR num,flag command controls the crosshairs. For the Model One/25, four crosshairs are available; two of these are optional and are drawn into the overlay planes. num gives the crosshair number; if flag=1, the crosshair is displayed. If flag=0, the crosshair returns to its default "invisible" state.

Crosshairs, when displayed, take their location from the coordinate registers. Crosshair 0 uses CREG 5. Crosshair 1 uses CREG 6. For the crosshair to track the cursor on the digitizing tablet, it is necessary to write a small macro. (Macro programming is described in detail in section 10.)

```
MACDEF 10
CMOVE 5 2     Load crosshair 0 location with cursor location
MACEND
BUTTBL 0 10   Execute macro 10 every 1/30th second
```

This macro will execute every 1/30th of a second, writing the cursor location into the crosshair location. For the crosshair to be displayed, it is still necessary to execute the command XHAIR 0 1. To turn off the crosshair, use the command XHAIR 0 0.

The crosshair colors are determined by value registers 1 (crosshair 0) and 2 (crosshair 1). The crosshair color is then XORed with the color in image memory to display the crosshair; the default crosshair value is 255,255,255.

4.0 PIXEL VALUES, LOOK-UP TABLES, AND IMAGE MEMORY

The Model One's image memory can be used in several ways. In a fully-configured, 24-bit plane Model One/25, up to 24 bits per picture element (pixel) can be used to store and display a 512x512 image with 8-bits of shading for each primary color. This allows 256 levels for each color: red, green, and blue; full-color imaging is thus provided, with over 16 million possible simultaneous colors.

This full-color imaging configuration, where the red, green, and blue components of the image are stored independently, is most frequently used to display smoothly-shaded three-dimensional objects. In a full-color imaging application, the 24 bits per pixel of image memory are divided into three banks: red, green, and blue.

In a Model One/25 with less than 24 bit planes, the Model One's image memory is used for pseudo-color (or false-color) imaging. In pseudo-color imaging, the Model One's three 512x512x8 image memory banks may not be fully populated, and the Model One's look-up-tables are then used to produce a color display. Also, a fully populated 24 bit per pixel Model One/25 can be used to store three independent 512x512x8 pseudo-color images.

In 1Kx1K addressing mode, the Model One supports up to 6 bits per pixel, allowing up to 64 simultaneous colors (see Section 5).

4.1 Pixel Values

Colors for drawing graphics primitives are selected with the VALUE red,grn,blu command or stored in value registers. Value registers are analogous to coordinate registers and are described in more detail below. Combinations of red, green, and blue are used to create specific shades and varied colors.

Some examples are:

```
PRMFIL ON
VALUE 255,0,0       Selects full-intensity red
CIRCLE 100
VALUE 255,0,255     Selects full-intensity magenta
CIRCLE 80
VALUE 255,255,0     Selects full-intensity yellow
CIRCLE 60
VALUE 255,100,100   Selects a shade of pink
```

You can create fine shades of color by gradually changing the value. For example, the command sequence below will create a finely-shaded cylinder.

```
MACDEF 10       Starts a macro definition
VADD 0 10       Add contents of value register 10 to value
                register 0
CADD 0 21       Add contents of coordinate register 21
                to coordinate register 0
CIRCLE 50       Draw a circle
MACEND          End macro definition
PRMFIL ON       Draw filled graphics primitives
VALUE 0,0,0     Set value to black
MOVABS -256,-256
                Set current location to lower-left
                corner of the screen
VLOAD 10 1,0,0
                Load value register 10 with 1
CLOAD 21 1,1    Load coordinate register 21 with (1,1)
BUTTBL 0 10     Set up button table to execute Macro 10
                every 1/30th of a second
```

Example 4.1 A Crawling Circle Demonstrates Shading

All of the above commands are described in this manual; for the moment, you can enter them by typing an ENTERGRAPHICS character [CTRL-D] at the alphanumeric terminal, then typing the commands exactly as they are written. Execution can be stopped by typing BUTTBL 0 0 at the terminal.

There are three commands available to specify the current pixel value. VALUE, which specifies a 24-bit pixel value, is described above. The other two commands are VAL8 and VAL1K; all three commands modify the current pixel value.

The current pixel value, stored in value register 0, gives the 24-bit value to be used whenever graphics primitives are drawn into image memory. For example, if you issue the drawing commands to draw a line into image memory, the line will be drawn using the current pixel value.

The <u>VALUE r,g,b</u> command specifies 24 bits of data, with 8 bits for each of the red, green, and blue banks. The parameters of the VALUE command are <u>red</u>, <u>green</u>, and <u>blue</u>, in that order. For example, <u>VALUE 0,255,0</u> sets the current pixel value to full-intensity green. The VALUE command is used for 24-bit full-color imaging applications.

The <u>VAL8 val</u> command specifies 8 bits of data, and is usually used for pseudo-color applications, where the Model One is configured with less than 24 bit planes, or when only one bank of image memory is to be written into at one time. VAL8 sets the current pixel value for all three of the red, green, and blue banks to the same 8-bit value. If only one bank is to be written, the WRMASK command, described below, can be used to protect the other banks of image memory while writing into the selected bank. For example, <u>VAL8 100</u> sets the current pixel value to (100,100,100).

In using the Model One for pseudo-color imaging, four special commands are available to optimize the number of bytes which must be transmitted from the host to specify the new pixel values. The four commands are VAL8, PIXEL8, RUNLN8, and LUT8. These commands end in 8, indicating that they are designed specifically for pseudo-color applications and systemss with less than 24 bit planes. Details of each command are given in the appropriate section.

The VAL1K val command specifies the six bits of pixel value data for 1024x1024 addressing mode and is described in more detail in Section 5.0.

4.2 Look-Up Tables

The Model One has three video look-up-tables, each of which is 256x8 bits. Each look-up-table (LUT) is associated with one of the Model One's digital-to-analog converters; the DACs are used to generate the analog video output signal to drive the video monitor (as shown in Figure 4.1).

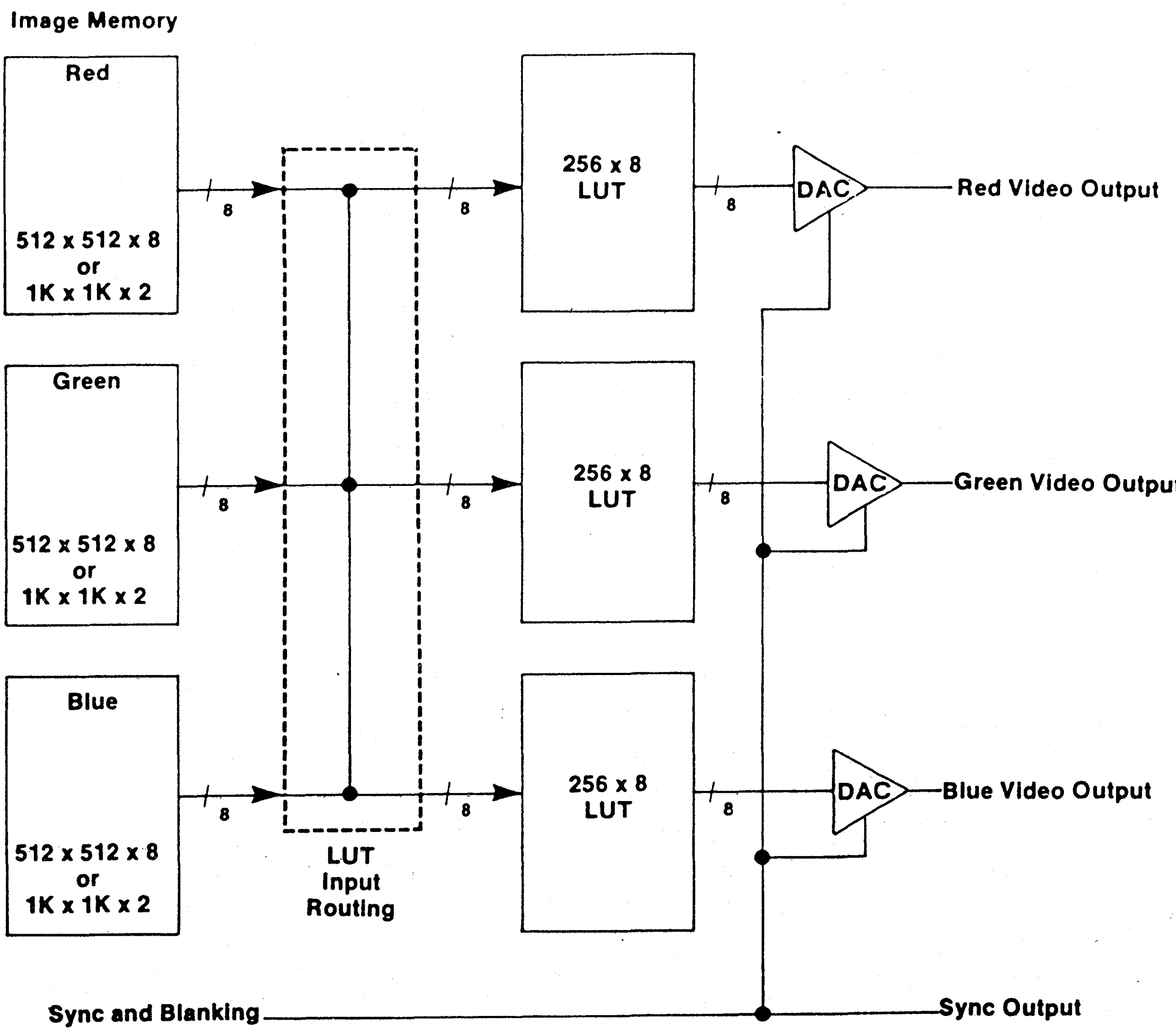

Figure 4.1 The Interrelationship of the Image Memory, Look-Up-Tables, and Digital-to-Analog Converters

Each look-up-table drives one DAC: the red LUT drives the red DAC, the green LUT drives the green DAC, and the blue LUT drives the blue DAC. However, the input to a given look-up-table does not have to come from its respective bank of image memory. Figure 4.1 shows the LUT Input Routing block, which controls the correspondence between the banks of image memory and the red, green, and blue LUTs.

The look-up-table input routing is set with the LUTRTE function (Look-Up-Table RouTE) command. LUTRTE controls which bank of image memory drives which LUT; the most common settings are given in Table 4.1.

Command	Use	Result
LUTRTE 0	Full color	Red bank drives red LUT Green bank drives green LUT Blue bank drives blue LUT
LUTRTE #7E	Pseudo-color	Red bank drives all LUTs
LUTRTE #75	Pseudo-color	Green bank drives all LUTs
LUTRTE #53	Pseudo-color	Blue bank drives all LUTs

Table 4.1 Common Look-Up-Table Routing Settings

The Model One can be programmed to provide double buffering by writing into a single bank of image memory while driving all three look-up-tables from another bank, and switching between banks to change the display rapidly.

4.3 Using the Look-Up Tables

In a 24 bit-per-pixel system, the red, green, and blue banks of image memory drive the red, green, and blue look-up-tables. The LUTs are then used to provide contrast or linearity correction to the displayed image. This direct correspondence (LUTRTE 0) is the default input routing on 24 bit plane Model One systems.

The look-up-tables are set to the system default at COLDstart. The default sets all three LUTs to a linear ramp with an index of 0 as the lowest intensity of the color, and 255 as full intensity. Combinations of red, green, and blue are used to create specific shades and varied colors.

Six Model One commands are available to load the look-up-tables: LUTA, LUTB, LUTG, LUTR, LUT8, and LUTRMP.

LUTA index, entry: The LUTA command sets the same location in all three look-up-tables to a single value. For example, LUTA 255 0 changes location 255 in all three LUTs to zero.

LUTB index, entry, LUTG index, entry, and LUTR index, entry: The LUTB command sets a location in the blue look-up-table; LUTG sets a location in the green look-up-table; LUTR sets a location in the red look-up table. For example, LUTR 100 255 sets location 100 in the red LUT to 255 (full intensity). If you did this after executing the series of commands above, you would see a sudden change to the displayed image as location 100 was changed. (This creates an arc of full intensity red.)

LUT8 index r,g,b: The LUT8 command changes the same location in all three look-up-tables to the three values specified. For example, LUT8 100 50,100,200 changes location 100 in all three LUTs: location 100 of the red LUT is changed to 50, location 100 of the green LUT to 100, and location 100 of the blue LUT to 200.

LUTRMP code sind,eind sent,eent: The LUTRMP command is used to set a linear "ramp" of look-up-table values. The command includes five parameters:

code	indicates the look-up-table to be loaded. code=1 indicates the blue LUT. code=2 indicates the green LUT. code=4 indicates the red LUT. code=7 indicates all LUTs.
sind,eind	these indicate the starting and ending locations within the look-up-table.
sent,eent	these indicate the starting and ending values (entries) to be made into the look-up-table.

For example, LUTRMP 4 0,255 255,0 totally reverses the power-up default entries in the red look-up-table. To go back to the example above (Example 4.1), if you typed in LUTRMP 4 0,255 255,0, you will see the background become red, the areas that were red become black—in general, the intensity components of the image will reverse.

LUTRMP 7 0 255 0 255 restores the power-on default contents of the red, green, and blue LUTs.

4.3.1 4-Bit Plane Model One Systems

In a Model One with four bit planes, the least significant four bit planes of the BLUE image memory bank have been populated. After power-on or COLDstart, the LUTRTE command should be issued:

LUTRTE #53	when typed locally
CALL LUTRTE (83)	when executed by an applications program from the host

This LUTRTE command instructs the Model One to drive the RED, GREEN, and BLUE look-up-tables from the BLUE image memory bank.

Once this is done, the RDMASK command should be used to force the high four bits to all zeros:

```
RDMASK #0F            when typed locally
CALL RDMASK (15)      when executed by an applications
                      program from the host
```

You can now use a series of LUT8 commands to initialize the look-up-tables to map the pixel values in image memory to the desired colors, as shown:

```
LUT8 0 0,0,0          Pixel value of 0 is black
LUT8 1 255,0,0        Pixel value of 1 is red
LUT8 2 0,255,0        Pixel value of 2 is green
LUT8 3 100,100,100    Pixel value of 3 is grey
     :
     :
command is used.  For example:

VAL8 3                Change the current pixel value
                      to 3,3,3
FLOOD                 Flood image memory with the
                      current pixel value
LUT8 3 100,255,100    Pixel value of 3 is light green
VAL8 15               Change current pixel value to
                      15,15,15--the maximum for the 4-bit
                      system
CIRCLE 50             Draw circle of radius 50
```

In summary, for a four-bit plane Model One system, you should use the commands

```
LUTRTE #53
RDMASK #0F
```

to configure the look-up-table routing, and then use the LUT8 command to initialize the look-up-table entries from 0 to 15 to correspond with the sixteen simultaneously displayable colors you wish to use.

4.3.2 8-Bit Plane Model One Systems

In a Model One with eight bit planes, the BLUE image memory bank has been populated. After power-on or COLDstart, the LUTRTE command should be issued:

```
LUTRTE #53            when typed locally
CALL LUTRTE (83)      when executed by an applications
                      program from the host
```

This LUTRTE command instructs the Model One to drive the RED, GREEN, and BLUE look-up-tables from the BLUE image memory bank.

You can now use a series of LUT8 commands to initialize the look-up-tables to map the pixel values in image memory to the desired colors, as shown:

```
LUT8 0 0,0,0           Pixel value of 0 is black
LUT8 1 255,0,0         Pixel value of 1 is red
LUT8 2 0,255,0         Pixel value of 2 is green
LUT8 3 100,100,100     Pixel value of 3 is grey
     :
     :
     :
LUT8 255 255,255,255   Pixel value of 255 is white
```

To change the current pixel value in a 8-bit per pixel system, the VAL8 command is used. For example:

```
VAL8 3                 Change the current pixel value
                       to 3,3,3
FLOOD                  Flood image memory with the
                       current pixel value
LUT8 3 100,255,100     Pixel value of 3 is light green
VAL8 15                Change current pixel value to
                       255,255,255--the maximum for the 8-bit
                       system
CIRCLE 50              Draw circle of radius 50
```

In summary, for a eight-bit plane Model One system, you should use the command

```
LUTRTE #53
```

to configure the look-up-table routing, and then use the LUT8 command to initialize the look-up-table entries from 0 to 255 to correspond with the 256 simultaneously displayable colors you wish to use.

4.3.3 <u>16-Bit Plane Model One Systems</u>

In a Model One with sixteen bit planes, the GREEN and BLUE image memory banks have been populated. After power-on or COLDstart, the LUTRTE command should be issued:

```
LUTRTE #75          when typed locally
```

to select the GREEN image memory bank, or

```
LUTRTE #53          when typed locally
```

to select the BLUE image memory bank.

To select the bank from an applications program:

```
CALL LUTRTE (117)     to select the GREEN bank
```

or

```
CALL LUTRTE (83)      to select the RED bank
```

The LUTRTE #75 command instructs the Model One to drive the RED, GREEN, and BLUE look-up-tables from the GREEN image memory bank, which is necessary to display the contents of the GREEN bank.

The LUTRTE #53 command instructs the Model One to drive the RED, GREEN, and BLUE look-up tables from the BLUE image memory bank, which is necessary to display the contents of the BLUE bank.

You can now use a series of LUT8 commands to initialize the look-up-tables to map the pixel values in image memory to the desired colors, as shown:

```
LUT8 0 0,0,0            Pixel value of 0 is black
LUT8 1 255,0,0          Pixel value of 1 is red
LUT8 2 0,255,0          Pixel value of 2 is green
LUT8 3 100,100,100      Pixel value of 3 is grey
     :
     :
     :
LUT8 255 255,255,255    Pixel value of 255 is white
```

To change the current pixel value in a 16-bit per pixel system, the VAL8 command is used. You should also use the write-protect masks to control whether pixel data is written into the BLUE or GREEN bank. For example:

```
LUTRTE #75             Select the GREEN bank for display
WRMASK 255 2           Write-enable the GREEN bank
VAL8 3                 Change current pixel value to 3,3,3
CIRCLE 25              Draw circle of radius 25
LUT8 3 100,255,100     Pixel value of 3 is light green
LUTRTE #53             Select the BLUE bank for display
WRMASK 255 1           Write-enable the BLUE bank
VAL8 255               Change the current pixel value to
                       255,255,255 (maximum for 16-bit system)
CIRCLE 50              Draw circle of radius 50
LUTRTE #75             Select the GREEN bank for display
```

In summary, for a sixteen-bit plane Model One system, you should use the command

```
LUTRTE #75
```

or

```
LUTRTE #53
```

to configure the look-up-table routing, and then use the LUT8 command to initialize the look-up-table entries from 0 to 255 to correspond with the 256 simultaneously displayable colors you wish to use.

You can then use the WRMASK and VAL8 commands to select the RED or GREEN bank for writing and display.

4.4 The Model One's Value Registers

The Model One uses sixteen value registers, called VREGs, to store 24-bit pixel values internally. Like the coordinate registers described in Section 3.0, some value registers have predefined functions; others are available for use by the programmer. The example above (Example 4.1), used to display fine shading, uses one of the available value registers to store a constant value to be added to the current pixel value.

The current pixel value--the value that is used by all commands that write graphic data to the Model One--is always stored in VREG 0. Table 4.2 shows the value register assignments.

Value Register	Use
VREG 0	The current pixel value; this is the value used by all commands that write graphic data to the Model One.
VREG 1	Crosshair 0 pixel value.
VREG 2	Crosshair 1 pixel value.
VREG 3	Fill mask used for seeded area fills.
VREG 4*	Color assignment for overlay plane 0.
VREG 5*	Color assignment for overlay plane 1.
VREG 6	Reserved.
VREG 7-15	Available for temporary value storage.

*For Option Card Users Only.

Table 4.2 Value Register Assignments

Five Model One commands are available to load and manipulate the contents of the value registers directly: VLOAD, VMOVE, VADD, VSUB, and RDPIXR. VALUE, VAL8, and VAL1K also load and manipulate the value registers, by setting the current pixel value (VREG 0).

<u>VLOAD vreg r,g,b</u> loads a 24-bit pixel value into any one of the sixteen value registers. For example, <u>VLOAD 10 1,0,0,</u> used in Example 4.1, loads value register 10 with the value (1,0,0); this value is added to value register 0

by the VADD command every time the macro is executed.

VMOVE vdst,vsrc moves the contents of one value register into another value register: VMOVE 10 11 copies the contents of value register 11 into value register 10.

VADD vsum,vreg and VSUB vdif,vreg add and subtract values between two value registers. As you saw above, VADD 0 10 adds the contents of value register 10 to the contents of value register 0; in the same way, VSUB 0 10 subtracts the contents of value register 10 from the contents of value register 0.

RDPIXR vreg places the value found in image memory at the current point into the specified value register. RDPIXR 10 determines the value at the current point and then places that value into VREG 10. The RDPIXR command can be used to select from a menu of colors, allowing the user to be given a choice of colors, select a color using the cursor, and make that color the current pixel value: RDPIXR 0 would make the pixel value at the chosen point the current pixel value.

The READVR vreg command does not affect the value within the value register: instead, the value of the specified VREG is displayed at the port that is in GRAPHICS mode (to the host if the host has sent the ENTERGRAPHICS character, to the local terminal if the local terminal has sent the ENTERGRAPHICS character.

4.5 The Pixel Processor

Whenever pixel data is written into image memory, the Model One's pixel processor is used. The pixel processor performs arithmetic and logic functions between incoming pixel data and the pixel values which are already in image memory. The pixel processor supports addition, subtraction, and the logical functions XOR, AND, and OR.

The pixel processor has three independent 8-bit arithmetic logic units (ALUs)--one for each of the red, green, and blue image memory banks. The pixel processor operates on data coming from the Model One's hardware vector generator (which may come from the host serial port or from the optional host DMA (Direct Memory Access) port). All graphics primitives are drawn into image memory by the hardware vector generator and are thus performed by the pixel processor.

Two commands control the pixel processor: PIXCLP and PIXFUN.

The PIXFUN mode command controls the arithmetic or logic function to be performed by the pixel processor. PIXFUN has a single parameter, mode, which sets the pixel processor mode, as shown in Table 4-3.

Mode	Mnemonic	Pixel Function
0		Replace image memory with incoming data
1	INS	Subtract image memory data from incoming data
2	SUBI	Subtract incoming data from image memory
3	SUBN	Add incoming data values to image memory
4	XOR	Exclusive OR incoming data values with image memory
5	OR	OR incoming data values with image memory
6	AND	AND incoming data values with image memory
7	PRESET	PRESET: write all 1's into image memory
8	CONDITIONAL	Conditional: inhibit writing of pixel values of (0,0,0). This mode is not available when performing a PIXMOV command.

Table 4.3 PIXFUN Modes and Functions

PIXCLP flag tells the pixel processor what to do if there is an underflow or overflow from an add or subtract operation on pixel values. PIXCLP 0 instructs the pixel processor ALUs to wrap-around on overflow or underflow; this effectively performs all computation modulus 256. PIXCLP 1 tells the pixel processor ALUs to clip their output to a maximum value of 255 or a minimum value of 0. Clipping may be useful when intensity values from two images are to be added or subtracted, to avoid unexpected results. If PIXCLP 1 were set, the intensity would reach its maximum value without wrapping around to black.

4.6 Write-Enable and Read-Enable Masks

The Model One's image memory planes can be selectively read-enabled and read-disabled, write-enabled and write-protected.

The Model One's video output section includes an eight-bit register called a read-enable mask. The read-enable mask is ANDed with the data from image memory immediately before the data enters the red, green, and blue look-up-tables: the same 8-bit read-mask is used for all three LUTs. If a bit in the read-mask is zero, the corresponding input bit in all three LUTs is forced to zero.

The RDMASK mask command sets the read-mask. For example, RDMASK 0 sets the read-mask to all zeroes and thus completely suppresses display of the image, forcing the input to all three LUTs to 0. (Note that this does not necessarily force the display to black; the display is dependent on the contents of the LUTs at index 0.) RDMASK #AA (hexadecimal) converts to alternating ones and zeroes, suppressing output of every other bit plane.

Whenever you use the RDMASK command, keep in mind that it affects the input to all three LUTs: red, green, and blue.

You should use the RDMASK command with caution because the Read mask register is logically "in front of" the look-up-tables. Thus, it has an effect on the addressing of the look-up-tables. For example, if you have the Read Mask set to #0F and then issue the command LUTA 255 255, you will actually change the look-up-table value at address zero to (255,255,255), because of the masking of the higher-order bits by the Read Mask. NOTE: if you want to be sure you are changing only the correct look-up-table indices, you should first set the Read Mask to #FF, then issue the look-up-table commands, then reset the Read Mask to the desired value.

The WRMASK bitm,bankm command selectively write-protects bit planes in the Model One's image memory. WRMASK uses two parameters: bitm and bankm. bitm is a single-byte mask controlling the write-protect status of each of the eight bit planes of image memory in all three image memory banks: each bit of bitm corresponds to one bit plane in all three banks. Whenever a bit of bitm is set, the corresponding plane in all three banks is writing-enabled. For example, setting bitm to #F0 (hexadecimal) write-enables the four most significant bit planes in the red, green, and blue banks; the four least significant bit planes are write-protected.

bankm is a five bit mask: each bit corresponds to one of the three image memory banks and the two overlay planes. The least significant bit (bit 0) corresponds to the blue bank; bit 1 sets the green bank; bit 2 corresponds to the red bank. If any of these three bits of bankm are set, the corresponding bank of image memory is then write-enabled.

The WRMASK command can also be used to write-protect the optional overlay planes (see section 7.0), as shown in Table 4.4.

bits 7,6,5	must be zero
bit 4	if=1, write-enable overlay plane 0
bit 3	if=1, write-enable overlay plane 1
bit 2	if=1, write-enable red image memory bank
bit 1	if=1, write-enable green image memory bank
bit 0	if=1, write-enable blue image memory bank
(bit 0=LSB, bit 7=MSB)	

Table 4.4 Settings for bankm

Thus, the bitm and bankm parameters of the WRMASK create a write-enable matrix (see Figure 4.2). A specific bit plane will be written only if both bitm and bankm indicate that the plane is write-enabled. WRMASK #F #1 write-enables only the four least significant bit planes of the blue bank; all other image memory bit planes are write-protected.

In 1024x1024 addressing mode, the 2 most significant bits of bitm write-protect/write-enable the two bit planes of image memory in each bank; the least significant 6 bits are ignored. The three bits of bankm are used identically to 512 mode, to selectively write-enable the three banks: red, green, and blue.

Read-masks may be used in conjunction with the write-enable masks in applications which use the 24 bit-plane capacity of the Model One to store multiple frames of a movie-loop animation sequence. The write-enable masks ensure that that only one frame at a time is written by the host; the read-masks and LUTRTE functions simplify the display of one frame at a time.

Section 10 (Macro Programming) gives examples of the use of the read-enable and write-enable masks for this type of animation.

4.7 Blinking Colors and the Blink Table

The Model One uses its look-up-tables and a blink table to provide blinking colors. The blink table lists addresses in the look-up-table; for each index, a pair of LUT entries is stored. The index specifies an address in the look-up-table: the contents of that address is toggled automatically between the specified entries. The blink rate determines the amount of time each entry stays in the look-up-table.

Four commands are used for blinking colors: BLINKC, BLINKD, BLINKE, and BLINKR.

The BLINKC command clears the blink table and stops all look-up-tables from blinking. After clearing, the first value given (entryl of BLINKE) remains.

The BLINKD lut,index command removes a single entry from the blink table and leaves the rest of the blink table intact. For example, BLINKD 7 100 disables blinking of address 100 in all look-up-tables.

The BLINKE lut,index entryl,entry2 command enables blinking of a specified index in the look-up-tables by making an entry in the blink table. For example, BLINKE 7 100 255 125 blinks location 100 in all look-up-tables between values 255 and 125. Up to 32 entries may be made in the blink table.

The BLINKR frames command sets the blink rate. The blink rate can be set to a multiple of the frame time (1/60th of a second). For example, BLINKR 30 sets the blink rate to once per second.

4.8 The CLEAR and FLOOD Commands

The CLEAR and FLOOD commands fill image memory with a uniform pixel value.

The CLEAR command fills the current clipping window as defined by CREGs 9 and 10 (WINDOW command) with the current pixel value. The selected pixel function (PIXFUN command) determines the actual pixel values that left in image memory when the CLEAR is done. For example:

```
VALUE 100,200,50
CLEAR
```

fills image memory with a pixel value of (100,200,50). If this is followed by

```
PIXFUN ADD
VALUE 20,0,200
CLEAR
```

All pixels in image memory would have (20,0,200) ADDed to their current pixel value of (100,200,50), resulting in a value of (120,200,250) in image memory.

The CLEAR command does not affect the current point.

The VECPAT mask command can be used to change the fill pattern for a CLEAR command, by specifying a pattern of pixels to be repeated along every scan line during the execution of the CLEAR command. mask is a 16-bit parameter; for example, VECPAT #AAAA contains alternating ones and zeroes. CLEARing the image memory with VECPAT #AAAA set will create a dotted fill pattern. More information on the VECPAT command is given in section 6.2.

The FLOOD command instantly fills every displayed pixel with the current pixel value. If the screen is zoomed in when the FLOOD command is issued, only the displayed portion will be FLOODed; FLOOD has no effect on undisplayed portions of the screen. The pixel function (PIXFUN command) does not affect the FLOOD command.

5.0 1024x1024 ADDRESSING MODE

The Model One/25 can also use its image memory as a 1024x1024 array, instead of a 512x512 array. With a full memory configuration (24 bit planes), the Model One/25 can store up to 6 bit planes of image data at 1024x1024. Each of the red, green, and blue banks stores 1024x1024x2.

In 1024x1024 addressing mode, the Model One/25's look-up-tables are bypassed. The output of the red, green, and blue banks are used to drive the red, green, and blue DACs (digital-to-analog converters) directly. The MODE1K func command selects the pixel data routing. The command description in Section 16.0 gives full details.

The MODDIS flag command selects between the 512x512 and 1024x1024 addressing modes:

    MODDIS 1

sets the display mode to 1024x1024.

    MODDIS 0

sets the display mode to 512x512 (the power-on default).

In addition, the MODDIS command clears image memory to a pixel value of (0,0,0) whenever the addressing mode is changed. Whenever MODDIS is executed, the look-up-tables and clipping window are reset to their default values. CREG 0, VREG 0, PIXFUN, PIXCLP, PRMFIL, and DEBUG are also reset to their COLDstart default values.

The VAL1K val command specifies the six bits of pixel value data which are needed for 1024x1024 addressing mode. Two bits of data are used for each bank. (The format is packed red-green-blue, in a single byte.) The VAL1K val command is the most efficient way of changing the current pixel value when running in 1024x1024 addressing mode.

When reading back data to the host computer (see section 9.2 for details), the READF command allows you to select the appropriate format for reading back the pixel data. READF func, with func=4, will read back data to the host using the same format as the pixel values in the 1K command. However, you should note that no image transmission commands are available to support this format, should you wish later to reload the Model One with the image. To store a complete image, you should use READF 0 and then restore the image with either RUNLEN or PIXELS.

In 1024x1024 addressing mode, the WRMASK command differs slightly from 512 mode. The two most significant bits of bitm write-protect/write-enable the two bit planes of image memory in each bank; the least significant 6 bits are ignored. The three bits of bankm are used identically to 512 mode, to selectively write-enable the three banks: red, green, and blue. Therefore, you must enable bit-planes in pairs; it is not possible to enable a single plane at a time.

Appendix I provides a full FORTRAN program using Model One/25 1K mode.

6.0 GRAPHICS PRIMITIVES

The Model One graphics primitive commands support local generation of common geometric entities: points, lines, circles, arcs, rectangles, polygons, and text. These entities are called graphics primitives and are used as building blocks for more complex images.

All commands which draw graphics primitives use the Model One's 16-bit virtual address space to define position and shape. The coordinate registers (current point, coordinate origin, and clipping window) control the placement of the graphics primitives within physical image memory, as described in Section 3.0, Coordinates and Image Memory Addressing.

6.1 Points

The simplest graphic entity, the point, is drawn with the POINT command. POINT sets the pixel located at the current point (CREG 0) to the current pixel value (VREG 0); the current pixel value and current point are unchanged.

6.2 Lines

When drawing lines or vectors, the Model One uses the current point as the starting point for the line; a DRAW command then specifies the ending point of the line.

Five DRAW commands are available, which are analogous to the five MOVE commands described in section 3.7. All DRAW commands use the current pixel value to draw the line. The DRAW commands are: DRWABS, DRWREL, DRW2R, DRW3R, and DRWI. The current point is always set to the endpoint of the line after the DRAW command has executed.

The DRWABS x,y command draws a line from the current point to the given ending point. The current point is set to (x,y) after execution. For example, DRWABS 10,10 draws a line to (10,10) from the current point. After execution, the new current point is (10,10).

The DRWREL dx,dy command draws a vector from the current point to the point specified by adding dx to the X coordinate and dy to the Y coordinate. For example, DRWREL -10,-10 would draw a line to (-20,-20) from the current point of (-10,-10). The new current point will again be set to the ending point of the line: (-20,-20).

The DRW2R dx,dy and DRW3R dx,dy commands are special forms of the DRWREL command for use when the displacement to the ending point of the line is small. The DRW2R command requires only two bytes; DRW3R requires only three bytes. Details of DRW2R and DRW3R are given in Section 16.0.

The DRWI creg command draws a line from the current point to the point specified by the given coordinate register. As with the other DRAW commands, the current point is set to the endpoint of the new line after execution of the command. For example, if CREG 23 holds the value (-20,30) and the current point is (10,10), the command DRWI 23 draws a line from (10,10) to (-20,30) and sets the new current point to (-20,30).

Because all DRAW commands set the last pixel of the line to be the current point, drawing a series of lines can produce overwriting of the pixels at the beginning and ending points of the lines. This is particularly obvious if the current pixel function (PIXFUN) is ADD, SUB, or XOR. The FIRSTP flag command can be used to inhibit writing of the first pixel of each line as it is drawn. The command FIRSTP 1 inhibits writing of the first pixel; FIRSTP 0 allows writing of the first pixel. FIRSTP 0 is the default.

Lines drawn by the Model One's DRAW commands may be solid, dotted, dashed, or any repetitive pattern you can specify with a 16-bit parameter. The VECPAT mask command specifies the pattern of pixels to be repeated along every subsequent line that is drawn. VECPAT #AAAA contains alternating ones and zeroes: lines drawn with this mask are drawn with every other pixel omitted and appear dotted. VECPAT #F0F0 draws lines with four pixels drawn followed by four omitted: these lines appear dashed. Other values will create other patterns. Repeating patterns can be generated; every time a pixel is written, the vector pattern is rotated by a single bit, without regard to the starting and ending points of lines.

Note that the VECPAT command also affects filling of graphics primitives (the PRMFIL command) and the fill pattern used in CLEAR.

The default mask for VECPAT is #FFFF, so that every pixel along the line is drawn.

6.3 Circles and Arcs

Three commands can be used to draw a circle: CIRCLE, CIRCXY, and CIRCI.

The CIRCLE radius command draws a circle of the specified radius centered around the current point. For example, to draw a circle with a radius of 30, centered around (50,50), use these commands:

```
MOVABS 50,50
CIRCLE 30
```

The CIRCXY x,y command draws a circle centered around the current point; point (x,y) is on the circumference. (The distance from the current point to point (x,y) determines the radius.) For example, if the current point is (0,0) and the command CIRCXY 10,0 is executed, a circle centered around (0,0), with (10,0) on its circumference, will be drawn.

The CIRCI creg command draws a circle centered around the current point; the point specified by creg lies on the circumference. For example, if the current point is (10,10) and CREG 21 holds point (25,25), the command CIRCI 21 draws a circle whose center is (10,10) and with point (25,25) on the circumference. The CIRCI command can be used to specify arbitrary circles interactively, using the digitizing tablet: the cursor can be used to specify the center point, then used again to specify the radius (by specifying a point on the circumference).

A single Model One command is available for drawing arcs: ARC. The ARC rad,al,a2 command draws an arc around the current point by giving the radius (rad), the starting angle (al), and the ending angle (a2). The starting and ending angles are given in one-degree increments counterclockwise; zero degrees is the positive X-axis; ninety degrees is the positive Y-axis.

For example, ARC 10 0,90, with the current point at (0,0), draws an arc of radius 10 from (0,10) to (10,0).

6.4 Rectangles

Three commands are available to draw rectangles: RECTAN, RECREL, and RECTI.

The RECTAN x,y command draws a rectangle: the current point defines one corner, and point (x,y) defines the diagonal corner. For example, with the current point at (10,10), the command RECTAN 20,20 draws a rectangle with (10,10) as one corner and (20,20) as the diagonally opposite corner.

The RECREL dx,dy command draws a rectangle using the current point as one corner and (dx,dy) as the displacement from the current point to specify the diagonally opposite corner. For example, with the current point at (100,100), the command RECREL 10,10 draws a rectangle with (100,100) as one corner and (110,110) as the diagonal corner.

The RECTI creg command draws a rectangle, using the current point as one corner and the point specified by creg as the diagonal corner. If the current point is (100,100) and CREG 21 holds the point (10,10), the command RECTI 21 draws a rectangle with (100,100) as one corner and (10,10) as the diagonal corner.

The rectangle commands do not change the current point.

6.5 Polygons

The Model One's POLYGN command allows you to draw arbitrary polygons. The POLYGN command has the format

    POLYGN npoly nverts,vert1,vert2...vertn nverts,vert1,vert2...vertn

The command is of varying length, depending on the number of polygons being drawn and the number of vertices in each polygon. The first parameter, npolys, specifies the number of polygons to be drawn, the second parameter, nverts, specifies the number of vertices in the first polygon. Then the vertices, vert1, vert2...vertn, are specified (as relative offsets to the current point). The number of vertices, nverts, in the second polygon may then be specified, and its vertices, vert1, vert2...vertn, and so on until all of the polygons have been drawn.

In using the POLYGN command, all vertices are relative to (or offset from) the current point; the current point is unchanged by the POLYGN command.

Because the POLYGN command allows specification of more than one polygon at a time, the Model One supports arbitrary polygons with interior holes. The polygons which are drawn by the POLYGN command can be filled or unfilled (see section 6.7): with PRMFIL ON, the nested interior polygons (islands) will be left unfilled, while the surrounding polygon is filled, as shown in Figure 6.1.

Some examples will clarify the use of the <u>POLYGN npoly nverts, vert1,vert2...vertn nverts,vert1,vert2...vertn</u> command:

```
MOVABS 0,0
POLYGN 1 3 10,10 10,-10 -10,-10
```

will draw a triangle (a 3-vertex polygon) from (10,10) to (10,-10) to (-10,-10).

Because the POLYGN is drawn offset from the current point, you can change the current point and issue the same POLYGN command:

```
MOVABS 100,100
POLYGN 1 3 10,10 10,-10 -10,-10
```

The above commands draw the same polygon (a triangle) from (110,110) to (110,90) to (90,90).

The command sequence

```
MOVABS 20,0
POLYGN 2 3 10,10 10,-10 -10,-10 4 20,20 20,-20 -20,-20 -20,20
```

would draw a triangle and a rectangle; the triangle would be inside the rectangle. (If PRMFIL ON had been executed first, the rectangle would be filled, the triangle would not be.) Figure 6.1 shows the two polygons.

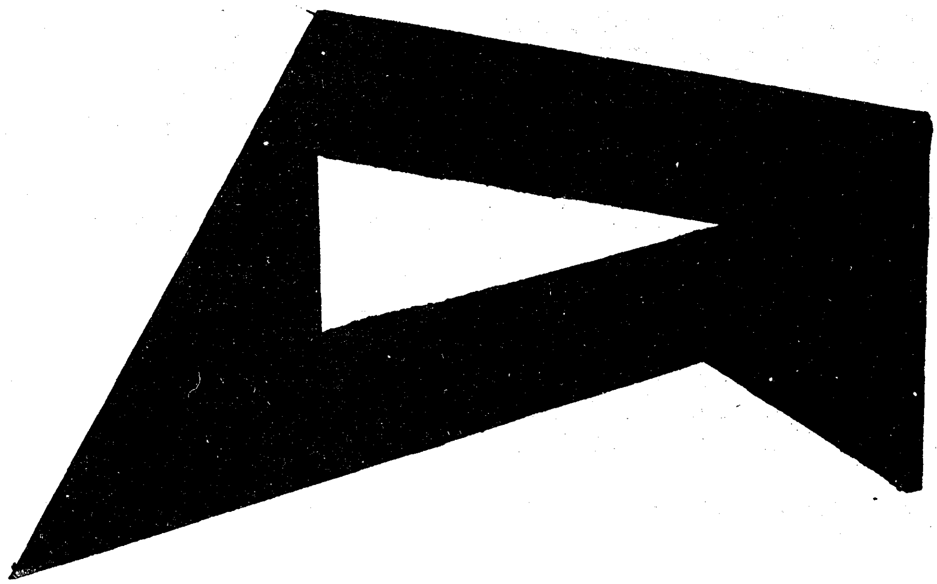

Figure 6.1 A Filled Polygon with a Hole

Whenever you use the POLYGN command, keep in mind that the vertices are relative to the current point. This facilitates dragging of the polygon interactively. For example, by writing Macros (see section 10) which set the pixel function (PIXFUN) to XOR, you could drag the polygon until the user wished to confirm it (without affecting image memory). Then, when the user confirms the location of the polygon, the PIXFUN could be set to change the value of pixel memory appropriately.

6.6 Text

The Model One supports seven text drawing commands: TEXT1, TEXT2, VTEXT1, VTEXT2, TEXTC, TEXTDN, and TEXTRE.

The TEXTC size,angle command sets the size and baseline orientation angle for subsequent text drawing commands. Text may be drawn at any angle, in one-degree increments counterclockwise; the scale factor may range from 0 to 255. A scale factor of 16 is the default; a scale factor of 32 doubles the size of the text. Details of how the angle and scale affect the drawing of text are given below.

The TEXTDN char,veclst command defines a second font (font 2) for text drawing. Font 2 is used by the TEXT2 and VTEXT2 commands. char gives the character to be defined. veclst defines the vectors that will draw the character. Details of the TEXTDN command are given in Section 16.0.

The TEXTRE command erases the definition of font 2 and allows a new font 2 to be defined if desired.

The four commands TEXT1, TEXT2, VTEXT1, and VTEXT2 are all used for drawing text. The commands TEXT1 string and TEXT2 string use font 1 and font 2 to draw text into image memory, at the angle and scale factor specified by the TEXTC command. If a particular character in font 2 has not been defined, the corresponding character in font 1 is used.

TEXT1 and TEXT2 draw horizontal text. Angles are calculated from the positive X axis.

The commands VTEXT1 string and VTEXT2 string also use fonts 1 and 2 to draw text into image memory, at the angle and scale factor given by TEXTC. The text is drawn vertically, starting at the current point and writing down. Angles are calculated from the negative Y axis counterclockwise; an angle of 90 degrees draws side-ways text along the X axis.

Figure 6.2 shows the various text angles and the differences betwen vertical and horizontal text.

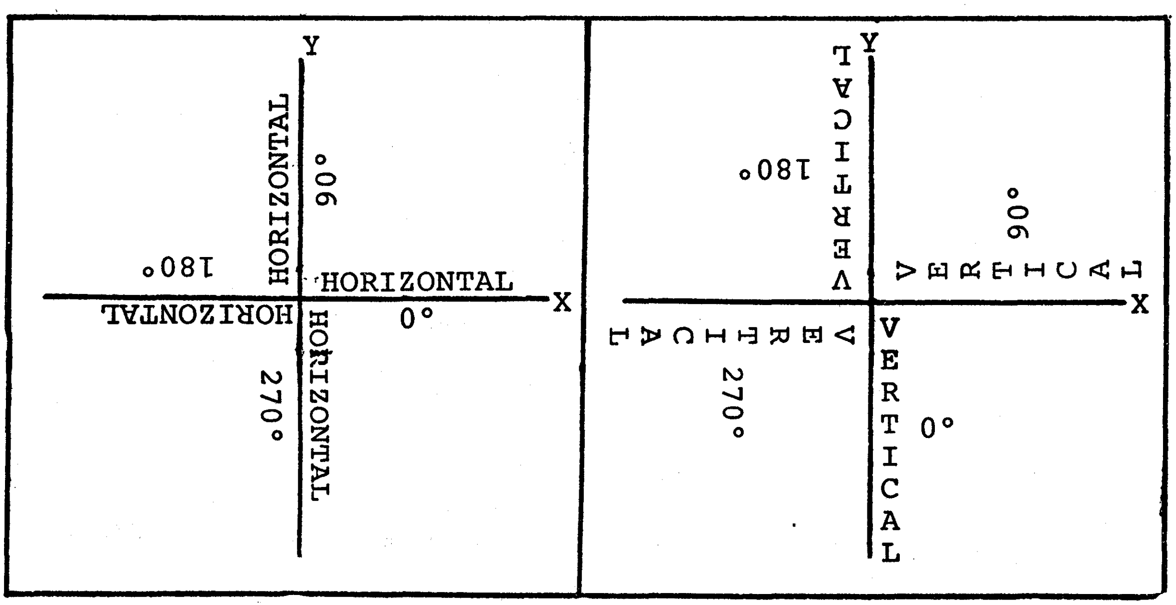

Figure 6.2 Horizontal and Vertical Text

Each text string is drawn with its lower-left corner placed at the current point. If the current scale factor is 16 (the default), each character uses five pixels horizontally and seven pixels vertically (excluding descenders, which take an additional two pixels).

When a TEXT command is sent from the host computer, it includes a one-byte parameter which indicates the number of characters to be drawn, followed by the 7-bit ASCII codes for each of the characters. All TEXT commands may be issued from the local terminal without counting the number of characters to be drawn. Each character after the command and delimiting blank is assumed to be part of the text string. For example:

```
TEXT1 Draw this text
```

will draw the text "Draw this text" horizontally from the current point.

```
VTEXT1 Draw vertical text
```

will draw "Draw vertical text" down from the current point.

6.7 Filled Primitives

Circles, arcs, rectangles, and polygons may be drawn filled or unfilled, using the PRMFIL flag command. When the command PRMFIL 1 or PRMFIL ON is issued, any subsequent graphics commands will draw filled graphics primitives. When a primitive is filled, the current pixel value is used for all interior pixels, as well as for the border. With PRMFIL OFF, the border only is drawn in the current pixel value.

The default setting is PRMFIL OFF.

6.8 Seeded Area Fills

In addition to filling of graphics primitives, the Model One also supports two area fill commands. In a seeded area fill, a seed point and a boundary condition are given. The seed point designates the interior of the area to be filled; the boundary condition tells the Model One the conditions which define the boundary of the area to be filled.

The AREA1 command uses the current point as its seed point. The boundary of the region is defined as any pixel whose value is different from the current pixel value. For example:

```
VALUE 0,0,0         Set current pixel value to black
CLEAR
PRMFIL OFF
VALUE 255,0,0       Red outlined shapes
MOVABS 0,0
CIRCLE 25           Draws a circle of radius 25, center (0,0)
VALUE 0,0,255       Value is now blue
AREA1
```

The commands above will draw a red unfilled circle. When the AREA1 command is executed, the circle is filled with blue, creating a blue disk with a red edge.

The AREA2 vreg command also uses the current point as its seed point. However, AREA2 does not stop filling an area until it encounters a pixel whose value is equal to the value stored in vreg or a pixel whose value is equal to the value stored in vreg 0. For example:

```
CLEAR
VALUE 255,0,0       Set value to red
MOVABS 0,0
PRMFIL OFF
CIRCLE 25
VALUE 0,255,0       Set value to green
RECTAN 25,25        Draw green rectangle from (0,0) to (25,25)
VALUE 0,0,255       Set value to blue
VLOAD 8 0,255,0     Load VREG 8 with green
MOVABS 1,1          Move into rectangle-circle overlap
AREA1               The arc defined by the rectangle
                    will be filled with blue.
AREA2 8             The entire rectangle will be filled
                    with blue.
```

Whenever an area fill is done, using either AREA1 or AREA2, the fill mask, specified by the FILMSK rmsk,gmsk,bmsk command, is ANDed with all pixel values read from image memory. Then the resultant pixel values are tested to see if they meet the prescribed boundary condition. The fill mask allows the boundary pixel values to lie in a different bit plane or image memory bank while performing the area fill in another bit plane or memory bank. The fill mask value is stored in VREG 3; the default value is (255,255,255).

For example:

```
MOVABS 0,0          Move to (0,0)
VALUE 255,255,255   Set value to white
CIRCLE 50           Draw white circle
VALUE 0,0,255       Set value to blue
CIRCLE 40           Draw blue circle
VALUE 0,255,0       Set value to green
CIRCLE 30           Draw green circle
FILMSK 255,0,0      Set fill mask to red only
AREA1               Fill area: green and blue circles are
                    ignored because of ANDing with
                    the red fill mask
```

7.0 DUAL OVERLAY PLANES

This section applies only to Model One/25 systems that include the Option Card.

The Option Card for the Model One/25 supports two 512x512x1 bit-planes. These bit-planes can non-destructively overlay the output from the Model One's image memory. Each overlay plane can be panned independently (each overlay plane has its own screen origin) and can be zoomed to either the same scale factor as image memory or at a scale factor of 1:1.

Each overlay plane may be assigned one of eight possible colors: black, red, green, blue, cyan, magenta, yellow, or white.

Display of the overlay planes has priority over display of image memory; thus, the overlay planes are logically in front of the image memory, as shown in Figure 7.1.

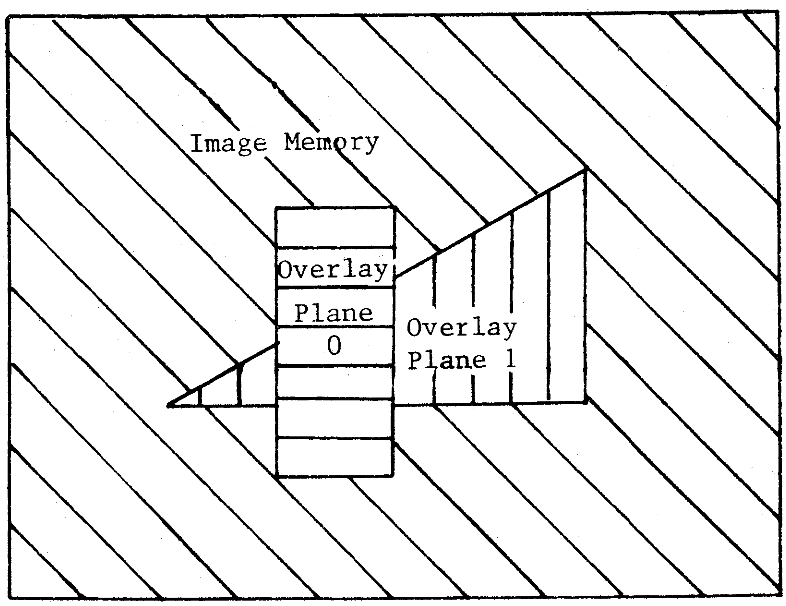

Figure 7.1 Overlay Planes in Front of Image Memory

The overlay planes function as binary bit maps which contain a pattern of 1's and 0's. If both overlay planes contain a zero in a particular screen location, then both planes are transparent, and the contents of the Model One's image memory are displayed at the specified screen location. If either overlay plane contains a one in the screen location, that location takes on the color of that overlay plane. If both overlay planes contain a one in the specified location, the screen location takes on the color of overlay plane 0.

7.1 Overlay Plane Screen Origin

Coordinate registers 15 and 16 specify the screen origin for overlay planes 0 and 1. Changes to the contents of these CREGs initiate automatic panning of the overlay planes.

For example:

SCRORG 100,100	Screen origin is 100,100
CLOAD 15 90,90	Overlay plane 0 screen origin is 90,90
CLOAD 16 80,80	Overlay plane 1 screen origin is 80,80

Macros (see section 10.0) can be used for interactive panning of the overlay planes:

MACDEF 10	Define macro 10
CMOVE 15 2	Put cursor location into CREG 15
MACEND	End macro 10
BUTTBL 0 10	Execute macro 10 every 1/30th second

The commands above define a macro which transfers the current cursor position (CREG 2) into CREG 15 (the screen origin of overlay plane 0). Repeated execution of this macro provides interactive panning of the overlay plane.

7.2 Overlay Plane Color

Value registers 4 and 5 give the color of the overlay planes when a bit in the overlay plane equals 1. VREG 4 determines the color for overlay plane 0. VREG 5 determines the color for overlay plane 1. The most significant bit of each of the red, green, and blue bytes specifies the color for the overlay plane.

For example:

VLOAD 5 #80 #80 #80	Overlay plane 0 is white
VLOAD 5 #80 #00 #00	Overlay plane 1 is red

7.3 Overlay Plane Write-Protect

The WRMASK command also supports write-protecting each of the overlay planes. In using the overlay planes, the bankm bits are used as follows:

bits 7,6,5	must be zero
bit 4	if=1, write-enable overlay plane 0
bit 3	if=1, write-enable overlay plane 1
bit 2	if=1, write-enable red image memory bank
bit 1	if=1, write-enable green image memory bank
bit 0	if=1, write-enable blue image memory bank

(bit 0=LSB, bit 7=MSB)

Software which runs on systems which do not possess overlay planes will run identically, as the overlay plane bits of bankm are ignored on systems without the Option Card. Normally, the overlay planes are write-protected; they must be explicitly write-enabled before any data can be written.

7.4 Overlay Plane Read-Enable

The OVRRD plane,flag command allows you to turn on or off the output of each overlay plane selectively. plane indicates which overlay plane (0 or 1) is being selected; then, if flag=1 the overlay plane is displayed (as described in section 7.0). If flag=0 the overlay plane is not displayed, regardless of its contents.

For example, the command OVVRD 0 1 enables display of overlay plane 0.

7.5 Overlay Plane Zooming

The overlay planes may be zoomed independently for display. Either overlay plane may be displayed at a scale factor of 1:1 or at the same scale factor as current image memory display.

The OVRZM plane,flag command controls the zooming of the overlay planes. As for OVRRD, plane indicates which overlay plane is being set. If flag=0, the overlay plane is always displayed at a scale factor of 1:1. If flag=1, the overlay plane is displayed at the same scale factor as image memory. For example, the command OVRZM 0 1 displays overlay plane 0 at the same scale factor as image memory: if image memory is zoomed, so is overlay plane 0.

This independent zooming allows you to use the overlay planes for information that should not be zoomed, such as descriptive text.

7.6 Overlay Plane Pixel Value

Vectors and graphics primitives drawn by the Model One graphics commands into the overlay planes may write ones or zeroes into the overlay planes, as defined by the OVRVAL plane,flag command. plane, of course, defines which overlay plane is being referenced; if flag=0, zeroes are written into the overlay plane. If flag=1, ones are written into the overlay plane. For example, the command OVRVAL 1 1 causes all subsequent graphics commands to write ones into overlay plane 1.

The more detailed example will demonstrate how this works:

```
OVRVAL 0 1      Write ones into overlay plane 0
OVRVAL 1 1      Write ones into overlay plane 1
WRMASK 0 #18    Write enable only the overlay planes
CIRCLE 50       Draw a circle
CLOAD 15 20,20  Set screen origin of overlay plane 0 to 20,20
```

7.7 Overlay Plane Crosshairs

Crosshairs 2 and 3 are used with the overlay planes. Crosshair 2 is drawn into overlay plane 0; crosshair 3 is drawn into overlay plane 1. The XHAIR command (see section 3.9) enables these crosshairs. The overlay plane crosshairs are displayed in the color chosen for the overlay plane and do not depend on the image memory data or the look-up-tables (as do crosshairs 0 and 1). Crosshairs 2 and 3 damage any data already drawn into the overlay planes, and should be used with caution. Crosshairs 2 and 3 take their positions from CREGs 7 and 8.

8.0 PIXEL MOVER

This section applies only to Model One systems that include the Option Card.

The Option Card pixel mover allows a window of pixel data to be moved between banks of image memory and to a different location within any bank. The pixel mover uses coordinate registers 11, 12, 13, and 14 to specify the location of the pixel data which is to be moved. You must set the CREGs which control the pixel mover before the pixel move is initiated.

8.1 Source and Destination Windows

The pixel mover moves data from a rectangular source window to a destination window of the same size. Coordinate register 11 contains the location of one corner of the source window; CREG 12 contains the diagonal corner of the window.

The pixel mover begins scanning the source window by moving the pixel at the location specified by CREG 11 first. Subsequent pixels in the window are moved in rows, proceeding towards CREG 12. Thus, the selection of the corners of the source window, as placed in CREGs 11 and 12, give you control over the order in which pixels are moved.

The destination window is specified by CREGs 13 and 14. CREG 13 contains the location of one corner of the destination window. CREG 14 controls the direction of scanning of pixels into the destination window. The direction of the displacement of CREG 14 relative to CREG 13 defines the position of the diagonal corner of the destination window, allowing mirroring as shown in Figure 8.1.

After loading CREGs 11 through 14, the PIXMOV command is executed to perform the actual move.

For example:

To move a 20 by 50 pixel array defined by the corner points (-100,100) and (-81,51) to the rectangle defined by the corner points (100,100) and (119,51), this command sequence would be used:

```
CLOAD 11 -100,100    Load source corner
CLOAD 12 -81,51      Load diagonal source corner
CLOAD 13 100,100     Load destination corner
CLOAD 14 119,51      Load destination scanning direction
PIXMOV               Perform the actual pixel move
```

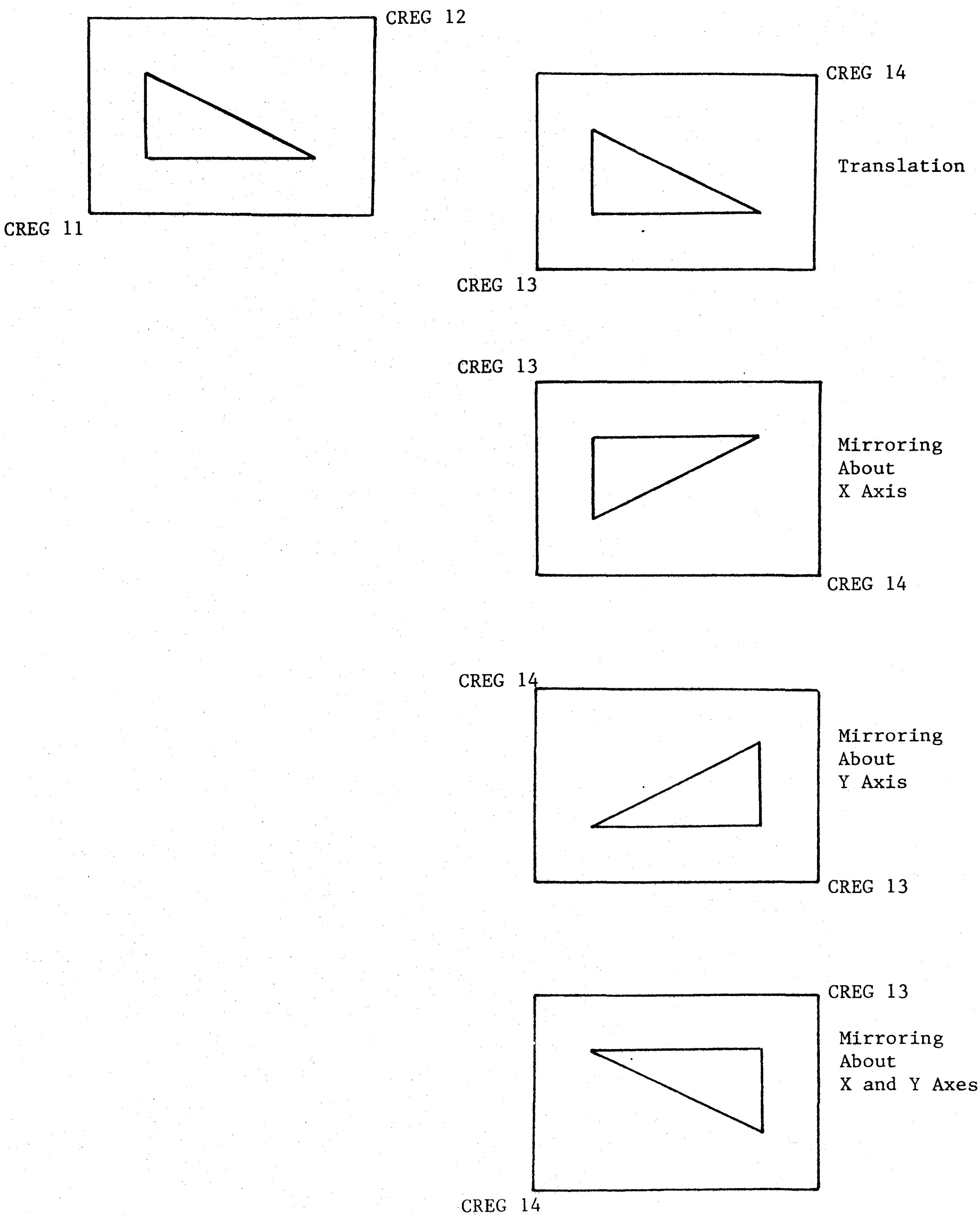

Figure 8.1 Settings of CREGs 11 Through 14 Allow Mirroring

8.2 Data Routing

The pixel mover can also be used to move data between the red, green, and blue banks of image memory. The default is to read pixel data from one color bank into the same color bank. For example, data from the red bank is transferred into the red bank during a pixel move.

The PMCTL 0 0 0 0 redrte,greenrte,bluerte command changes the default pixel mover routing and allows pixel data to be moved between the red, green, and blue banks. The PMCTL command includes seven parameters: the first four must be set to zero for successful command execution.

The redrte parameter controls which bank of image memory is to be moved into the red bank. If redrte equals zero, no data is moved into the red bank. If redrte equals one, data from the red bank is moved into the red bank (the default). If redrte equals two data from the green bank is loaded into the red bank. Finally, if redrte equals three, data from the blue bank is transferred into the red bank by the pixel mover.

The greenrte and bluerte parameters function in the same way, as shown in Table 8.1.

	redrte		greenrte		bluerte
0	nothing into red	0	nothing into green	0	nothing into blue
1	red to red	1	red to green	1	red to blue
2	green to red	2	green to green	2	green to blue
3	blue to red	3	blue to green	3	blue to blue

Table 8.1 PMCTL Parameter Values

For example, the command

```
PMCTL 0 0 0 0 1,2,3
```

is the default setting for pixel mover data routing: red into red, green into green, and blue into blue.

The command

```
PMCTL 0 0 0 0 1,1,1
```

writes all pixel data from the red bank: red into red, red into green, red into blue.

You may also use the WRMASK command to prevent the pixel mover from writing into the red, green, or blue banks. For example, if you are using the pixel mover to transfer data from the red bank into the green bank but not into the blue bank, only the green bank should be write-enabled. This allows maximum flexibility in specifying pixel moves.

The PIXFUN command controls the pixel processor mode--whether new data is inserted, ANDed, ORed, and so on--during a pixel move. The CONDITIONAL mode is not available during a pixel move.

Thus, you can use the pixel mover to pick up a window--perhaps containing a commonly-used area of the image--and transfer it to any desired area of the screen. This image section can then be inserted, ADDed, XORed, and so on. XOR can be used to drag the window until the correct positioning is determined, when the pixel function would be switched to INSERT.

9.0 DATA READ-BACK AND IMAGE TRANSMISSION

This section describes the Model One commands which are used to load image memory with data from the host computer, and also describes the commands which read data from the Model One back to the host computer.

9.1 Reading Back Information to the Host Computer

The Model One supports seven commands for reading back data from the Model One to the host computer: READP, READCR, READVR, READW, READWE, READBU, and READF.

Whenever any of these READ commands is issued and data is sent by the Model One to the host computer, the host must acknowledge receipt of the transmitted data by sending a 7-bit ASCII ACK character (06H or 86H) back to the Model One. Any data sent from the host before the ACK is received is ignored by the Model One.

The host acknowledge character must be sent as a 7-bit ASCII ACK, regardless of whether the host normally sends data to the Model One in 8-bit binary or ASCII hexadecimal format.

This handshaking protocol ensures proper operation of the Model One when communicating with the host over a full-duplex serial communication line. A full duplex host computer will echo back the characters sent by the Model One; ignoring characters sent by the host until the ACK is sent alleviates this problem.

When a READ command is issued, the data requested is sent back to the port that requested the information.

When the READ command is issued from the local alphanumeric terminal, no acknowledgement is expected or required.

The READP command reads the pixel value of the current point and sends that value to whichever port (host or alphanumeric terminal) is in graphics mode. The value is sent as three ASCII decimal numbers, representing the red, green, and blue pixel values. When the pixel value is sent to the host, three three-digit integers (FORTRAN 3I3 format) are sent.

The READCR creg and READVR vreg commands read back the contents of the coordinate registers and value registers. The READCR command returns two 6-digit integers (2I6), representing the X and Y cooordinates; the READVR command returns three 3-digit integers (3I3) representing the red, green, and blue pixel values. For example, if CREG 23 holds the point (10,-10), the command READCR 23 returns 00010 -00010.

The READF func command specifies the format of the pixel data that is sent to the host when a READW or READWE command is executed. (The READW and READWE commands are described in the next few paragraphs.) func=0 tells the Model One to send full 24 bit per pixel values--red, green, and blue--in FORTRAN 3I3 format. func=1, 2, or 3 selects data from a single image memory bank (red=1, green=2, blue=3) and send the data in I3 format. func=4 instructs the Model One to send data in the same packed RGB format used to send data to the Model

One in the VAL1K command; the data is sent in I3 format to the host computer.

The READW nrows,ncols bf command instructs the Model One to read back a rectangular array of pixels. nrows and ncols specify the number of rows and columns to be read. bf, the blocking factor, specifies the number of pixels to be sent back to the host before waiting for the ACKnowledge character to be sent to the Model One from the host. If the end of the window is reached before the block is full, it is padded with zeroes and sent.

If, after the READW command has been issued and before it completes, the host wants the Model One to discontinue sending data blocks, the host can send the NACK (negative acknowledge) character instead of an ACK. The NACK character must be sent as a seven-bit ASCII NACK (15H or 95H); the character may be changed with the SPCHAR command, described in section 2.3. Once the NACK is sent, the command is completely aborted--in fact, the Model One leaves graphics mode.

The READW block factor, bf, prevents host input buffer overflows.

Once the window is defined by nrows and ncols, the READW command sends the pixels in the window from left-to-right and top-to-bottom. The current point is used to specify the upper-left corner of the window. For example, to read back all of image memory, the current point is moved to (-256,255) for the Model One/25 in 512 mode. Then, the command READW 512 512 bf, with the correct blocking factor, is given.

The READW command sends data to the host computer in the same format that the PIXELS and PIXEL8 commands use to send data to the Model One.

The READWE nrows,ncols bf command is a run-length encoded form of the READW command. Like the READW command, the READWE command is aborted if the host computer sends a NACK character. When the READWE command is executed, it sends back data in a form that can later be transmitted to the Model One by the RUNLEN and RUNLN8 commands: it includes a pixel value (r,g,b for RUNLEN and val for RUNLN8) and a count. The count is from 0 to 255, and indicates one less than the actual number of pixels set to that value. (Note: the count is done this way to allow 256 pixels--one half of a scanning line for the Model One/25--to be set at once, if appropriate.) The count is sent in FORTRAN I3 format. As in READW, the blocking factor, bf, specifies the number of pixels to be sent before waiting for an ACK. Again, if the end of window is reached before the block is full, the block is padded with zeroes and sent.

The READBU flag,cflg command allows the host computer to determine which function buttons have been pressed at the user's workstation. Whenever a function button is pressed, the Model One makes an entry in the function button event queue; this entry includes the button number and the digitizer or joystick coordinates. The function button event queue is eight events deep. The READBU command removes one entry from the event queue and sends the data to the host. The event queue is first-in, first-out (FIFO): if more than one button is pressed, the first button is sent to the host.

The flag parameter of the READBU command indicates whether the Model One should respond immediately.

If flag=1, the Model One will wait until there is an entry in the button queue and then send the data; if the button queue is not empty, the Model One sends the data immediately. if flag=0, the Model One responds immediately: if the queue is empty, a button number of zero is sent; if there is data in the queue, the data is sent. The button number is returned as a three-digit integer (I3). The (x,y) coordinates--whether from the digitizer or joystick--are returned as 2I6.

The cflg parameter of the READBU command indicates whether the digitizer coordinate (cflg=0) or the joystick coordinate (cflg=1) should be returned.

If, for example, the user presses button 4 twice, and you execute this command sequence, you will see:

```
READBU 1 0
 004  00010 -00015
READBU 0 0
 004  00010 -00015
READBU 1 0
```

and no value will be returned from the third READBU command until another button is pressed.

9.2 Image Transmission

Four commands are available to load a rectangular array of image memory pixels with arbitrary data specified by the host computer. PIXELS and PIXEL8 send data to the Model One in a pixel-by-pixel form. RUNLEN and RUNLN8 send pixel data in a run-length encoded format, in which each pixel value is followed by a one-byte count specifying the number of pixels (horizontally) to be set to the specified pixel value.

The transmission time for some images is reduced by using the run-length transmission format; other images may require more time to transmit because of the overhead involved in sending the one-byte count along with each pixel value. The easiest way to determine which format is more rapid for a particular type of image is to compare the required transmission time for sample images in each of the two possible formats.

The PIXELS nrows,cols r,g,b ... r,g,b command transmits 24 bits of pixel value data per pixel to fill a specified rectangle. nrows and ncols specify the number of rows--the height-- and the number of columns--the width--of the rectangular array of pixels which is to be filled. For each pixel in the rectangle, one byte each of red, green, and blue pixel value data is sent, starting at the upper-left corner and working left-to-right and top-to-bottom. The upper-left corner of the rectangle begins at the current point (CREG 0).

For example, to fill the entire image memory of the Model One/25 in 512x512 addressing mode, the current point is moved to (-256,255) --the upper-left corner of pixel memory--and the command PIXELS 512 512 given, followed by 3x512x512 bytes of image memory data, used to fill the 512x512 rectangle with red, green, and blue pixel values.

The PIXELS command is analogous to issuing a series of VALUE, POINT, and MOVREL 1,0 commands to fill image memory. (Of course, the analogy breaks down at the end of the scan line, when you would have to move explicitly to the next line.)

The PIXEL8 nrows,ncols val ... val command also defines a rectangular array of pixels in the Model One's image memory. The pixel value data is given as a single one-byte value, as in the VAL8 command (see section 4.0). The PIXEL8 command functions in the same way as the PIXELS command: nrows and ncols define the rectangle; the series of val is used to fill the rectangle with pixel values.

The PIXEL8 command is most often used to transmit 8 bit-per-pixel pseudo-color images; it is analogous to a series of VAL8, POINT, and MOVREL 1,0 commands.

The RUNLEN nrows,ncols r,g,b count command is a run-length encoded form of the PIXELS command. nrows and ncols again define the rectangle for the pixel data. r,g,b and count define the pixel value and the number of horizontal pixels to be set to the given r,g,b pixel value. count gives one less than the number of adjacent pixels to be set: for example, if count=0, one pixel is set; if count=1, two pixels are set. The maximum count of 255 sets 256 pixels to the r,g,b pixel value.

The RUNLN8 nrows,ncols val count command is the pseudo-color form of the RUNLEN command. The pixel value is interpreted as in the VAL8 command. Like the PIXEL8 command, RUNLN8 is used in applications using less than 24 bits per pixel.

10.0 MACRO PROGRAMMING

The Model One's macro programming facility allows you to define a group of graphic commands to be executed together by issuing a single command whenever desired.

Up to 256 macros may be simultaneously defined and stored at any one time; a COLDstart or power-up will erase all macros.

Four commands are available for defining and executing macro commands. MACDEF and MACEND define a macro command; MACRO executes a macro; and MACERA erases a macro.

10.1 Defining a Macro

Macros are referred to by number, from 0 to 255. The command MACDEF num starts a macro definition. The command MACEND ends the macro definition. In between these two commands, you can include any number of graphics commands, including a MACRO command to execute another macro. The only graphics commands which cannot be included in a macro are ASCII and QUIT.

After you issue the MACDEF command from the local terminal, you will receive the special macro definition prompt $. Graphics commands may be entered without being executed; they are included in the macro definition and executed when the macro is executed. Macro definition ends when you type the command MACEND.

For example:

! MACDEF 11	Start definition of macro 11: user command is underlined.
$ CMOVE 4 2	Make digitizer location (CREG 2) the screen origin (CREG 4)
$ MACEND	End macro definition

The commands above define Macro 11. Macro 11 provides for interactive panning of the display. Every time Macro 11 is executed, the screen origin will be placed at the current digitizer location. If you want to constantly update the screen origin to reflect the digitizer origin, you can use the button table (section 10.5, example 10.4).

If you make an error--a typo or an invalid parameter-- while entering a macro definition, you will be given an error message, and the line will be erased. Lines cannot be changed or edited, however. You can then continue entering the macro command.

10.2 Executing a Macro

Once you've defined a macro, you can then execute it. To execute macro 11 once, type

MACRO 11

The more g
number you used in defining the macro command.

Calling a macro that has not been defined does nothing. No error message is given.

10.3 Erasing a Macro

To erase any macro, you can type the command MACERA num. To erase all macros, you can use the COLDstart command. COLDstart, however, does reset a large number of other functions.

For example, you can erase Macro 11 with the command MACERA 11.

To redefine a macro, simply type MACDEF num. This will erase the old definition of the macro and provide you with a clean slate.

10.4 Suggestions for Writing Macros

As a programming practice, you should be careful to restore conditions to what they were before the macro was executed. For example, in defining macros which draw objects using MOVE and DRAW commands, you should restore the current point to its original value before the end of the macro. Relative MOVE and DRAW commands allow you to execute the macro with the current point positioned anywhere in the image memory by assigning the current point appropriately.

The examples below show several macros.

MACDEF 25	Begin Macro 25
ZOOMIN	Zoom in by factor of two
VALUE 0,255,0	Set color to green
MACEND	End Macro 25
VAL8 255	Set color to white (255,255,255)
CIRCLE 50	Draw circle of radius 50 around current point
MACRO 25	Execute Macro 25: zoomin and change the color to green
CIRCLE 40	Draw green circle of radius 40 around current point
ZOOM 1	Restore scale factor

Example 10.1 Defining A Macro

Example 10.1 shows a simple macro; all it does is change the color to green and zoomin by a factor of 2.

```
MACDEF 5              Start definition of Macro 5
DRWREL 10,0           Draw four line segments: these define a box
DRWREL 0,10           with the lower-left corner at the current
DRWREL -10,0          point
DRWREL 0,-10
MACEND                End Macro 5
VALUE 255,255,255     Change color to white
MOVABS 50,50          Move to the point (50,50)
MACRO 5               Draw the box defined by Macro 5
MOVABS 70,70          Move to (70,70)
MACRO 5               Draw the box defined by Macro 5
MACDEF 6              Define Macro 6
MACRO 5               Macro 5 will be executed as part of
                      Macro 6
MOVREL 20,0           Move right 20
MACRO 5               Execute Macro 5 again
MOVREL 0,20           Move up 20
MACRO 5               Execute Macro 5 again
MOVREL -20,0          Move left 20
MACRO 5               Execute Macro 5 one more time
MOVREL 0,-20          Move back to starting point
MACEND                End definition of Macro 6
MOVABS 0,0            Move to (0,0)
MACRO 6               Execute Macro 6
                      --This will draw four boxes.
MOVABS -50,-50        Move to (-50,-50)
MACRO 6               Draw four more boxes.
```

Example 10.2 Lots of Little Boxes

Example 10.2 defines a macro, which in turn is executed by another macro to draw a collection of boxes. You can nest macros up to 16 deep. For example, you could set up macro 6 to be executed by a macro similar to the one in Example 4.1:

```
MACDEF 10              Define macro 10
VADD 0 10              Add contents of VREG 10 to current value
CADD 0 21              Add contents of CREG 21 to current point
MACRO 6                Execute macro 6
MACEND                 End macro 10
VALUE 0,0,0            Set initial current value to black
MOVABS -256,-256       Move current point to lower-left corner
VLOAD 10 1,0,4         Load VREG 10
CLOAD 21 1,1           Load CREG 21
BUTTBL 0 10            Execute macro 10 every 1/30th second
```

Example 10.3 A Fine-Shading Macro

Example 10.3 takes advantage of the button table to execute macro 10 every 1/30th of a second. The net result is a set of finely-shaded square tubes. CADD and VADD are used to change the location and the color gradually. The next section describes the button table in detail. Issue the command <u>BUTTBL 0 0</u> to terminate its execution.

10.5 <u>Using the Button Table</u>

Many interactive applications require the display controller to perform graphics operations in response to user input. The digitizing tablet's cursor supplies a set of function buttons; macro commands can be set to be executed in response to these function buttons.

When a function button is pressed by the user, the button number is sent to the Model One over the TABLETSIO interface. The Model One then uses the button number to specify the macro to be executed in response to the function button. The <u>button table</u> keeps track of which macro should be executed in response to which button. The <u>BUTTBL index,nmac</u> command is used to load the button table. <u>index</u> gives <u>the button number;</u> <u>nmac</u> gives the number of the macro to be executed in response to that button.

The 13 or 16 buttons on the digitizing tablet cursor correspond to entries 1 through 13 (for the 13 button cursor) or entries 1 through 15 (for the 16 button cursor, where button 0 is ignored by the Model One). For example, if button 5 were pressed, entry 5 of the button table would indicate which macro should be executed in response.

Every 1/30th of a second, the Model One checks to see if any function buttons have been pressed. If a function button has been pressed, the button table is used to indicate which macro to execute; if no button has been pressed, the macro indicated by button table location zero is executed.

Button table location zero provides a "background macro"-- the macro specified by location zero is executed every 1/30th of a second when no buttons are pressed. For example, button table location zero can be used to provide interactive cursor tracking, by setting the cursor location equal to the data tablet's current coordinate every 1/30th second.

```
XHAIR 0 1          Turn on crosshair 0
MACDEF 15          Define Macro 15
CMOVE 5 2          Copy current cursor location into current
                   crosshair location
MACEND             End Macro 15
BUTTBL 0 15        Execute Macro 15 if no button is pushed
```

Example 10.4 Interactive Cursor Tracking

The macro running from location 0 cannot be interrupted until the final MACEND is executed. However, the macro at location 0 can interrupt other executing macros.

10.6 Advanced Macro Programming

This section gives several examples of macro programming, such as rubber-banding, panning, double-buffering, and movie-loop animation.

10.6.1 Panning

This macro continuously pans the display in steps set by the contents of CREG 20, one step every 1/30th second:

```
MOVABS 0,0
CIRCLE 50

MACDEF 10
CADD 4 20          Add the contents of CREG 20 to the
                   screen origin (CREG 4)
MACEND
CLOAD 20 10,15
BUTTBL 0 10        Execute Macro 10 every 1/30th second
```

Example 10.5 Display Pans Continuously

In Example 10.5, the panning of the display depends on the contents of CREG 20; to change the step, change CREG 20. To stop the panning entirely, change the button table entry at location 0 (BUTTBL 0 0, for example) or erase macro 10 (MACERA 10).

The digitizing tablet cursor can also be used for panning, as shown in the next example, where the cursor location controls the center of the screen.

```
MACDEF 11
CMOVE 4 2        Make the cursor location the screen origin
MACEND
BUTTBL 0 11      Execute Macro 11 every 1/30th second
```

Example 10.6 Cursor-controlled Panning

You can also define a macro to zoom in on the display whenever a function button is pressed:

```
MACDEF 12
ZOOMIN
FLUSH            Empty the function button event queue
MACEND
BUTTBL 2 12      Execute Macro 12 when button 2 is pressed
```

Example 10.7 Button-controlled Zooming

The FLUSH command in Example 10.7 empties the function button event queue. (The event queue is described in section 9.2, with the READBU command.) The FLUSH command should be used whenever you want to make sure that no extraneous buttons have been pushed. Note that once the function button event queue has been filled, further button depressions are ignored. The function button event queue holds up to eight events.

The READBU flag,cflg command, described in detail in section 9.2, takes an entry from the event queue and sends it to the host; the flag indicates whether or not the Model One should wait for a button to be pressed or send a button number of zero if the event queue is empty. clfg indicates whether the digitizer or joystick coordinates should be sent.

The next set of macros allow you to perform rubber-banding and confirmation of lines. To execute the set of macros, type in the three macros, enter the last three commands, and start entering lines. Button 1 confirms the endpoint of each line (and the start point of the next line).

```
MACDEF 40        Macro to draw and un-draw lines
MOVI 22          CREG 22 gives the starting point of the line
DRWI 21          CREG 21 gives the current endpoint
MOVI 22
CMOVE 21 2       Make cursor location the current endpoint
DRWI 21
MACEND

MACDEF 41        Macro to start rubberbanding of lines
XHAIR 0 0        Disable crosshair 0
PIXFUN 4         XOR lines with image memory
BUTTBL 0 40      Execute Macro 40 every 1/30th second
```

```
FLUSH
MACEND

MACDEF 42       Macro to confirm a given line and
MOVI 22          restart rubberbanding
DRWI 21
PIXFUN 0        Add the line to image memory
MOVI 22
DRWI 21
PIXFUN 4
CMOVE 22 21     Make line endpoint the new starting point
BUTTBL 0 40     Execute Macro 40 every 1/30th second
FLUSH
MACEND

VALUE 0,0,255   Set value to blue
MACRO 41        Execute setup macro
BUTTBL 1 42     Press button 1 to confirm the endpoint
```

Example 10.8 Rubberbanding Lines Create a Pattern

Example 10.8 sets up two states for the Model One: macro 40, which executes every 1/30th second, repeatedly draws a line (using the pixel function XOR to keep from destroying image memory), while macro 42 confirms the line and draws it into image memory. (Macro 41 sets things up.)

This sort of multi-state macro programming can be extended to support movie-loop animation and double-buffering. In movie-loop animation the bit planes of image memory are used to store a series of sequential frames from an animation sequence; the bit planes are then played back.

To perform this sort of animation, the various frames of the sequence are first loaded into the proper bit planes. The WRMASK command helps in making sure the planes are loaded correctly.

Then, to perform the playback of the images, a series of macro commands is used. Each macro displays one frame of the animation, perhaps changing the read-enable masks, look-up-tables, and screen origin. The last command of each macro changes the button table to display the next frame in the sequence. Thus, you can create a linked list of macros to display the desired animation sequence.

11.0 APPLICATION DEVELOPMENT FEATURES

The Model One includes special features to help the applications programmer to identify and correct problems with host-based applications programs and Model One macro commands.

The local debugger allows the user to step through the command sequence being sent by the host, display a listing of currently defined macros, and issue graphics commands to the Model One from the local keyboard.

The command stream translator disassembles commands that are being executed by the Model One and displays them at the local alphanumeric terminal in mnemonic form.

The REPLAY comand allows the user to examine the last 32 characters that were sent from the host.

The ALPHAO strlen, string command allows ASCII data to be sent to the local terminal while the Model One is in GRAPHICS mode. This allows an applications program to issue prompt messages at the local workstation.

11.1 The Local Debugger

To enter the local debugger, type a [CTRL-X] at the terminal. (See section 2.3 for more information about special characters.) If you are in GRAPHICS mode when you type the [CTRL-X], type a carriage return after the [CTRL-X]. Once the current graphics command has been completed, the Model One responds with the debugger prompt DEBUG>. This prompt indicates that the Model One has suspended command execution and is ready for debugger commands. You can now enter any valid graphics command, as if you were in local GRAPHICS mode, or any of the special debugger commands (/S, /L, /M, /C, and /V).

The /S (Step) command allows you to step through the host command stream or a local macro. /S may include a number to indicate the number of commands to execute before returning to the debugger. Once you specify a number, this number becomes the new default: /S 3 changes the default number of commands to three, instead of one. For example:

```
DEBUG> /S          Execute one command
DEBUG> /S 1000     Execute 1000 commands
DEBUG> /S 5        Execute 5 commands
DEBUG> /S          Execute 5 commands
```

The /M (Macro) command lists the numbers of all currently-defined macros. For example:

```
DEBUG> /M
0 25 100 101 102 200          Lists all non-empty macros
DEBUG>
```

The /L nmac (List macro) command lists the macro whose number is given. For example:

```
DEBUG> /M            List numbers of defined macros
0 25 100 101 102
DEBUG> /L 100        List contents of macro number 100
MOVABS 105 230
DRWREL 10 20
CIRCLE 50
DRWREL -10 20
CIRCLE 20
DEBUG>               After listing, returns to debugger
```

The /V flag enables or disables execution of the macro associated with button table entry 0. During normal execution, the macro indicated by button table entry 0 is executed every 1/30th of a second; when in the debugger, execution is automatically suspended. The command /V 1 restarts execution (every 1/30th second) of the macro indicated by button table entry 0. Note that, like all commands, the macro will not be executed unless you type /S. (/V 0 disables execution, if desired, once execution has been restarted.)

The /C command exits the local debugger and returns to normal operation of the Model One. Command execution continues from the point where it was interrupted.

Table 11.1 summarizes the local debugger commands.

Command	Use
/S number	Step through number commands
/M	List all defined macros
/L nmac	List contents of macro number nmac
/V flag	flag=1 enable macro at button table entry 0; flag=0 disable macro at button table entry 0.
/C	Exit debugger and continue normal command execution
All graphics commands	Normal execution of graphics commands directly

Table 11.1 Summary of Local Debugger Commands

11.2 Command Stream Translator

The Model One graphics command set includes the command DEBUG flag, which is used to turn the command stream translator on and off. The command stream translator should not be confused with the local debugger. The command stream translator disassembles the commands which are being executed by the Model One and displays them on the local alphanumeric terminal in mnemonic form.

The command stream translator will work with both DMA and serial interfaces; however, the Model One should be configured with XON/XOFF communications protocol enabled, to avoid host input buffer overflow problems.

Each command opcode is translated into the command name (0E (hex) becomes CIRCLE, for example); the parameters are converted to ASCII decimal.

DEBUG 1 starts the command stream translator; DEBUG 0 disables the command stream translator.

The command stream translator can be used from within an applications program to help diagnose problems. For example, if a particular section of code is causing problems, you can call the DEBUG command before entering that section of code:

```
  :
  :
  :
CALL PROC1          Call user procedure PROC1
CALL DEBUG (1)      Enable the command stream translator
CALL PROC2          Call user procedure PROC2 and
                    use the command stream translator to
                    follow it in detail
CALL DEBUG (0)      Disable the command stream translator
CALL PROC3          Call user procedure PROC3
  :
  :
```

Example 11.1 Using the DEBUG Command

You can also use the command stream translator in conjunction with the local debugger (see section 11.1). For example, you can halt normal command execution with the [CTRL-X] entry into the local debugger, type DEBUG 1 to turn on the command stream translator, and inspect the commands being executed by using the /S command of the local debugger to step through the commands. The next example shows what such a command sequence would look like:

```
CTRL-X              Enter local debugger with [CTRL-X]
DEBUG> DEBUG 1      Enable command stream translator
DEBUG> /S 5         Execute 5 commands
MOVABS 0,0          Display disassembled command stream
DRWABS 50,100
CIRCLE 20
DRWABS 10,10
DRWABS 10,20
DEBUG> DEBUG 0      Disable command stream translator
DEBUG> /C           Exit local debugger
```

Example 11.2 Using the Local Debugger and the DEBUG Command

11.3 Instant Replay

The Model One command REPLAY displays the last 32 characters which were sent by the host to the Model One. The characters are displayed in ASCII hexadecimal form. For example, the command REPLAY executed from the local alphanumeric terminal outputs an array of ASCII hex characters:

```
00 FF FD F0 A0 AA AB BF
FF F0 FC 00 00 00 00 00
F0 FF 3F 3C 80 89 8A 76
7F FF FF FF 30 3F 55 F5
```

The command stream starts at the upper-left corner, goes across the top row, and continues until it reaches the lower-right corner.

The REPLAY command may also be useful in debugging the HOSTSIO interface to identify stray characters from a modem or bad transmission line.

12.0 PROGRAMMING THE Z8000

The Model One supports four commands for downloading and debugging Z8000 object code: DNLOAD, MAP, PEEK, and POKE.

The DNLOAD command allows Z8000 object code to be downloaded and added to the standard command set.

The MAP command displays a Z8000 memory map on the local alphanumeric terminal. The map is used to determine the starting address for downloading code.

The PEEK addr command displays the contents of address addr. For example, PEEK #0FFE displays F0A0 in response.

The POKE addr,data command changes the data at address addr to data. The POKE command allows one word of memory to be changed each time it is issued. Like all the Z8000 programming command, POKE should be used with great caution: POKING AROUND CARELESSLY CAN CRASH THE CENTRAL PROCESSOR. If you do crash the processor with careless POKEs, you can recover by pressing the START button on the back panel (which performs a COLDstart). Poking into PROM memory will have no effect.

13.0 HOST FORTRAN LIBRARY

The Model One host FORTRAN library, called ONELIB, gives the programmer access to all of the Model One commands through subroutine CALLs from the host application program.

To send any command from the host to the Model One, the programmer issues a CALL to a ONELIB subroutine. All ONELIB command subroutines have the same name as the local command mnemonic. For example, these FORTRAN lines:

```
CALL MOVABS (0,0)
CALL CIRCLE (100)
CALL DRWABS (20,50)
```

are identical to typing thesal:

```
MOVABS 0,0
CIRCLE 100
DRWABS 20,50
```

The FORTRAN library contains several levels of subroutines. The subroutines which generate Model One commands are called by the application program; those subroutines in turn call low-level subroutines to perform I/O between the host and the Model One.

Each Model One command has an equivalent FORTRAN subroutine. The command descriptions in section 16.0 contain full descriptions of the FORTRAN call, the parameters, and the variable names for the parameters. For example, the FORTRAN call for the MOVABS command is:

```
CALL MOVABS (IX,IY)
```

IX and IY are INTEGER*2 variables; the order for parameters is the same as for locally-typed commands.

For all FORTRAN subroutines, these conventions are used: parameters are given in the same order as for locally-typed commands; they are always INTEGER*2 (ranging from -32,768 to 32,767); and they are never changed by the FORTRAN call.

13.1 Output to the Model One

ONELIB uses buffered output when sending data to the Model One. The command subroutines do not actually perform output; instead, data is packed into an output buffer which is sent to the Model One when it is full. Two subroutines are used by the ONELIB subroutines to put data into the output buffer: SEND1 and SEND2.

SEND1 puts a single byte of data into the output buffer, passed in the low eight bits of the calling parameter. For example:

```
CALL SEND1 (64)       Put 40H into the output buffer
CALL SEND1 (255)      Put FFH into the output buffer
CALL SEND1 (0)        Put 0 into the output buffer
```

SEND2 puts a 16-bit value into the output buffer, passed in the single calling parameter. For example:

```
CALL SEND2 (256)        Put 0100H into the output buffer
CALL SEND2 (32767)      Put FFFFH (maximum) into the buffer
CALL SEND2 (0000H)      Put 0000H into the buffer
```

Both SEND1 and SEND2 automatically make calls to empty the output buffer if it becomes full as a result of the call to SEND1 or SEND2.

Note that you can use SEND1 and SEND2 to send commands without using the FORTRAN subroutine calls. These subroutines are not normally called directly by the programmer. For example:

```
CALL SEND1 (01)        Put 01H into buffer: opcode for MOVABS
CALL SEND2 (IX)        X coordinate
CALL SEND2 (IY)        Y coordinate
CALL EMPTYB            Send command, padded with nulls
```

Thus, the command sequence above duplicates the direct command MOVABS IX,IY or

The EMPTYB subroutine empties the output buffer. Several ONELIB subroutines use EMPTYB to dump the buffer. The READ commands, which read data from the Model One, dump the output buffer to ensure that the READ command was sent. A READ command which is still in the host's buffer cannot be executed until the buffer is sent; thus, the buffer is dumped, the READ command is sent, and the host awaits input from the Model One.

The QUIT command, which exits the Model One's GRAPHICS mode, uses EMPTYB to force dumping of the output buffer to make sure the QUIT command is received before any other terminal I/O is started.

The user may call EMPTYB from the application program to force emptying of the output buffer.

For example, the program can use SEND1, SEND2, and EMPTYB as follows:

```
CALL SEND1 (1)       Put 01H into the output buffer
CALL SEND2 (64)      Put 0040H into the output buffer
CALL SEND2 (65)      Put 0041H into the output buffer
CALL EMPTYB          Dump the output buffer
```

which will send this to the Model One:

```
01H   00H   40H   00H   41H   (Five bytes)
```

13.2 Entering Graphics Mode

The Model One powers up and COLDstarts into ALPHA mode (see Section 2.1). Before an application program can issue graphics commands to the Model One, the Model One must be placed into GRAPHICS mode, using the ENTGRA subroutine (which is the equivalent of [CTRL-D]), as shown in the next example.

```
        :
        :
     WRITE (IOUT,1000)                      Normal operation
1000 FORMAT ('ENTER X,Y,RAD,A1,A2:')
     READ (IN,1010) IX,IY,IRAD,IA1,IA2
1010 FORMAT(I6)
C
     CALL ENTGRA                            Enter GRAPHICS mode
C
     CALL VAL8 (100)                        Program may now use
     CALL FLOOD                             graphics commands to
     CALL PRMFIL(1)                         the Model One.  No
     CALL MOVABS (IX,IY)                    normal terminal I/O
     CALL VAL8 (255)                        until exit GRAPHICS
     CALL CIRCLE (IRAD)                     mode.
     CALL VAL8 (0)
     CALL ARC (IRAD,IA1,IA2)
C
     CALL QUIT                              Exit GRAPHICS mode.
C
     WRITE (IOUT,1020)                      Normal operation.
1020 FORMAT ('DONE WITH GRAPHICS')
        :
        :
```

Example 13.1 Enter and Exit GRAPHICS Mode

An application program can enter and exit GRAPHICS mode as many times as
n in ALPHA mode, and QUIT only when in GRAPHICS mode. ONELIB
traps violations of this rule and signals an error.

13.3 Initializing I/O to the Model One

Before beginning I/O to the Model One, ONELIB must be initialized, using the subroutine RTINIT. The application program must make a call to RTINIT before any other library calls are made. For example:

```
      CALL RTINIT           Initialize the library
      CALL ENTGRA           Enter GRAPHICS mode
      DO 1000 I=10,200,10
      CALL CIRCLE (I)
1000  CONTINUE
      CALL QUIT             Exit GRAPHICS mode
      STOP
      END
```

Example 13.2 Initializing ONELIB

13.4 ONELIB COMMON Blocks

There are two primary COMMON blocks used by ONELIB subroutines for global variables and parameters. The application program can check the ERRCOD parameter in the /ERROR/ COMMON block to determine whether a ONELIB subroutine has generated an error; ERRWHO contains the subroutine's name.

```
   INTEGER BUFFER(256),BUFSIZ,PTR,BYTCNT
      INTEGER LUNERR,LUNIN,LUNOUT
      INTEGER IBF,IBFMIN,IBFMAX,IFMT
      LOGICAL GRFMOD,HIBYTE,BUFFLG,FILFLG
C
      COMMON /RASTEK/ BUFFER,BUFSIZ,PTR,BYTCNT
      COMMON /RASTEK/ LUNERR,LUNIN,LUNOUT
      COMMON /RASTEK/ IBF,IBFMIN,IBFMAX,IFMT
      COMMON /RASTEK/ GRFMOD,HIBYTE,BUFFLG,FILFLG
C
C ---ERROR---
C
      INTEGER ERRWHO(3),ERRCOD
C
      COMMON /ERROR/ ERRWHO,ERRCOD
```

Table 13.1 Raster COMMON Blocks

13.5 ONELIB Error Reporting

The ONELIB subroutines perform error checking on parameters and report any illegal values; no Model One command will be output if an illegal parameter value is given. Instead, the subroutine IERROR is used to generate an error message, which is sent to the LUNERR logical unit.

For example, if the ZOOM subroutine is given an illegal scale factor, it calls IERROR:

```
ONELIB -- [ZOOM  ]:  ERROR #18
```

You can then call subroutine ERRMSG for more detail. For example, if you call ZOOM with an illegal scale factor, and then call ERRMSG, these messages are displayed on the LUNERR logical unit:

```
ONELIB -- [ZOOM  ]:  ERROR #18
ONELIB -- [ZOOM  ]:  ILLEGAL SCALE (ZOOM) FACTOR
```

If the LUNERR and LUNOUT logical units are the same (the terminal is on the Model One's ALPHASIO port), the error message routines momentarily leave GRAPHICS mode, send the error message, and reenter GRAPHICS mode.

The ONELIB error codes are given in Table 13.2.

Code	Error Message
1	Illegal angle
2	Illegal value register
3	Illegal radius
4	Illegal flag
5	Illegal button table index
6	Illegal macro number
7	Illegal coordinate register
8	Illegal coordinate
9	Illegal displacement
10	Illegal value
11	Illegal look-up-table index
12	Illegal look-up-table entry value
13	Illegal function
14	Illegal mask
15	Illegal string length
16	Illegal scale
17	Illegal crosshair number
18	Illegal scale (zoom) factor
19	Illegal number of polygons
20	Illegal pixel functions
21	Illegal look-up-table routing
22	Illegal look-up-table number
23	Illegal frame rate
24	Illegal light number
25	Illegal font
26	Illegal array dimension
27	Illegal parameter range
28	I/O error
29	Illegal count parameter
30	Run-lengths and window size disagree

Table 13.2 ONELIB Error Codes and Messages

13.6 Input From the Model One

The subroutines which read data back from the Model One are included in ONELIB. These routines--READBU, READCR, READP, READVR, READW, AND READWE--change some of their calling parameters to return the requested data. (READF sets the readback format and does not actually read data back.) The FORTRAN calls for these commands are described in Section 16.0.

13.7 Additional ONELIB Subroutines

In addition to the ONELIB subroutines corresponding to the Model One local commands, there are additional subroutines available to supplement the library. These subroutines add to the standard image transfer commands and provide greater flexibility. They are listed in Table 13.3.

Subroutine	Function
PX8HDR (ROWS,COLS)	Starts an image data transfer by sending the PIXEL8 command, but does not send data.
PX8STR (VAL)	Sends one 8-bit pixel data value
PXSHDR (ROWS,COLS)	Starts an image data transfer by sending the PIXELS command, but does not send data
PXSSTR (RED,GRN,BLU)	Sends one 24-bit pixel data value
RN8HDR (ROWS,COLS)	Starts an image data transfer by sending the RUNLN8 command, but does not send data
RN8STR (VAL,CNT)	Sends one 8-bit pixel data value and a count
RNLHDR (ROWS,COLS)	Starts an image data transfer by sending the RUNLEN command, but does not send data
RNLSTR (RED,GRN,BLU,CNT)	Sends one 24-bit pixel data value and a count

Table 13.3 ONELIB Image Data Transfer Commands

14.0 HOST COMPUTER DMA

The Model One option card includes a DMA (Direct Memory Access) port for high speed transfers between the host computer and the Model One. The format of the data which sent over the DMA interface is the same 8-bit binary transmission format supported over the HOSTSIO RS-232 interface. Note that the selection made when installing the Model One, between ASCII hexadecimal and 8-bit binary, applies only to the HOSTSIO interface. DMA transmission always occurs in 8-bit binary.

The Model One's host FORTRAN library, ONELIB, provides supported DMA drivers for some host computers. Users of other host computers wishing to construct their own DMA drivers can use these as a reference point for their development.

15.0 ERROR CONDITIONS

The Model One command interpreter sends error status information to the local alphanumeric terminal whenever an error occurs. This section lists the error messages.

The number indicates the error message number (which is given to simplify looking up the message in this manual).

0. Illegal call to routine "ERROR": If you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

1. Bad command opcode: the host computer has tried to issue a command with an undefined opcode. (Section 17.0 lists all the command opcodes.)

2. Unrecognized command: the command mnemonic that was given does not exist. The command HELP lists all the available commands.

3. Unimplemented command: the command mnemonic exists but has not been implemented.

4. Number is out of range: the range for the parameter has been exceeded; for example, the range for CREGs is 0 to 63. pp

5. String is not a number: the value given for a parameter was not a valid number. Keep in mind that hexadecimal numbers must be preceded by a #: #FFFF or #A0.

6. Bad hex digit in stream: the Model One received a bad ASCII hex digit from the host computer over its HOSTSIO interface. (To get this message, the HOSTSIO interface must be set up for operation in ASCII hexadecimal mode: see the Installation Guide for details.)

7. Illegal parameter: an illegal parameter was given that was not out of range; for example, this message would appear if you gave the command ZOOM 3.

14. Macro calls nested too deeply: macro calls may not be nested more than 16 levels deep.

15. Call to undefined macro: you attempted to execute a macro that had not been defined.

16. Attempt to erase active macro: you cannot erase a macro that is currently being executed. (Note: if the macro is being executed as a result of button table location zero, you can change the button table with the BUTTBL command.)

17. Not enough space for definition: the available space for macros and downloaded text has been used. Reconfigure user memory with the CONFIG command or erase text and macro definitions.

18. Unknown command in macro definition: the QUIT and ASCII commands may not be used in a macro.

19. Allowed only in macro definitions: the MACEND command cannot be used outside of a macro.

20. Arithmetic overflow in calculation: a calculation performed by the Model One central processor resulted in an overflow. This happens most commonly with graphics commands that "fall off" the edge of the 16-bit virtual address space.

21. ASCII command not allowed in macro definition: the ASCII command may not be used in a macro.

22. LUTRTE command has illegal data--ignored: the function given for the LUTRTE command is illegal. The command is ignored.

23. Insufficient space to complete operation: the POLYGN, AREA1, or AREA2 command ran out of stack space to fill the specified area. Reconfigure user memory with the CONFIG command to allocate more space.

24. Lookup table commands not used in 1K mode: the Model One/25's 1K mode does not allow use of the look-up-table commands.

25. Add and subtract not allowed in 1K mode: the Model One/25's 1K mode does not allow PIXFUN add or subtract functions.

26. Not allowed in ALPHA mode: the DNLOAD command can only be issued by the host computer.

27. Bad name length: this message is generated by the DNLOAD command.

28. Bad record format: this message is generated by the DNLOAD command.

29. No start address: this message is generated by the DNLOAD command.

30. Bad checksum: this message is generated by the DNLOAD command.

31. Name table overflow: this message is generated by the DNLOAD command.

32. Blink table overflow: the blink table (see section 4.7) has been filled. You can clear the blink table with the BLINKC command.

33. Loading into protected area: this message is generated by the DNLOAD command.

34. No graphic input box present

35. Only from parallel port

37. Input queue full on serial port

38. Overrun error on serial port

39. Parity error on serial port

40. Framing error on serial port

41. Break detected on serial port

47. Model One Firmware Failure: If you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

48. IEEE-488 bus error

49. Bad Z80002 vectored interrupt: You may receive this message during a DNLOAD command; otherwise, if you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

50. Illegal Z8002 instruction: You may receive this message during a DNLOAD command; otherwise, if you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

51. Privileged Z8002 instruction: You may receive this message during a DNLOAD command; otherwise, if you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

52. Z8002 segmentation trap: You may receive this message during a DNLOAD command; otherwise, if you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

53. Bad Z8002 non-vectored interrupt: You may receive this message during a DNLOAD command; otherwise, if you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

54. Bad Z8002 non-maskable interrupt: You may receive this message during a DNLOAD command; otherwise, if you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

55. Bad Z8002 system call: You may receive this message during a DNLOAD command; otherwise, if you receive this error message, please write down the circumstances under which it happened and then contact Raster Technologies, Inc. or your local representative.

56. Insufficient memory space for configuration: there is insufficient memory space available for the configuration specified.

SYNTAX

ALPHAO strlen, string

FUNCTION

The ALPHAO (ALPHA Output) command outputs a text string on the local alphanumeric display screen. The text to be output is specified by **string**. If the command is being entered in ASCII mode from the local alphanumeric terminal or keyboard, **string** is the set of ASCII characters remaining on the command line. If the command is not being sent in ASCII mode, then **strlen** must be given. **Strlen** contains the number of characters in the string followed by a **string** with **strlen** bytes.

PARAMETERS

strlen	the number of characters in string; needed only if the command is not sent in ASCII mode.
string	the text to be output.

HOST BINARY COMMAND STREAM

[$B4_H$][strlen]([char1][char2]...[charn])
$B4_H=264_8=180_{10}$

FORTRAN CALL

CALL ALPHAO (STRLEN,STRING)

STRLEN is an integer specifying the number of characters that are to be output. **STRING** is an integer array with two characters packed per 16-bit word, as in FORTRAN A2 format.

EXAMPLE

!ALPHAO ABCDEF 1 2 3	Output text string "ABCDEF 1 2 3"
!ALPHAO WXYZ	Output text string "WXYZ"

SYNTAX

ARC rad,a1,a2

FUNCTION

The ARC command draws a circular arc with its center at the current point (CREG Ø) and with a radius of **rad,** the starting angle **a1** and ending angle **a2.** The angle is specified in integer degrees measured counter-clockwise. An angle of Ø specifies horizontal along the positive X axis. The arc is drawn counter-clockwise from the start angle to the end angle.

PARAMETERS

rad	gives the radius of the arc; range is -32,768 to +32,767.
a1	starting angle; range is -32,768 to +32,767.
a2	ending angle; range is -32,768 to +32,767.

HOST BINARY COMMAND STREAM

[11_H][highrad][lowrad][higha1][lowa1][higha2][lowa2]
11_H=021_8=17_{10} 7 bytes

FORTRAN CALL

CALL ARC (IRAD,IA1,IA2)

EXAMPLE

!MOVABS Ø Ø	Move current point to location 0,0
!ARC 75 45 135	Draw circular arc of radius 75, starting at 45° and ending at 135°
!ARC 100 -30 60	Draw circular arc of radius 100, starting at -30° and ending 60°
!PRMFIL ON	Select filled primitives
!ARC 40 -10 40	Draw filled (pie shape) arc of radius 40, starting at 10°, ending at 40°

SYNTAX

AREA1

FUNCTION

The AREA1 command is used for area fill. AREA1 sets all pixels in a given closed region to the current value (VREG Ø). The region extends from the current point (CREG Ø) outward in all directions until a boundary whose pixel values differ from the pixel value at the current point is reached. The boundary pixel values and the original pixel value are AND'ed with FILMSK (VREG 3) before the comparison is made. The FILMSK is set by the FILMSK command.

HOST BINARY COMMAND STREAM

[13_H] (1 byte)
$13_H=023_8=19_{10}$

FORTRAN CALL

CALL AREA1

EXAMPLE

!VAL8 255	Set current pixel value to 255,255,255
!PRMFIL OFF	Select unfilled primitives
!MOVABS 0 0	Move current point to 0,0
!CIRCLE 30	Draw circle of radius 30
!MOVABS 25 20	Move current point to 25,20
!CIRCLE 35	Draw circle of radius 35
!VALUE 255 0 0	Set current value to 255,0,0
!AREA 1	Begin area fill from 25,20 outward to boundary
!MOVABS 10 -10	Move current point to 10,-10
!VALUE 0 255 0	Set current pixel value to 0,255,0
!AREA 1	Begin area fill from 10,-10 outward to boundary

RELATED COMMAND

FILMSK

SYNTAX

AREA2 vreg

FUNCTION

The AREA2 command performs area filling. AREA2 sets all pixels in a given closed region to the current value (VREG Ø). The region extends from the current point (CREG Ø) outward until a boundary of pixels whose value is specified by value register **vreg** or **vregØ** is reached. The boundary pixel values and the value specified by value register **vreg** are AND'ed with FILMSK (VREG 3) before the comparison is made. The FILMSK is set by the FILMSK command.

The AREA2 command differs from AREA1 in that the pixel value of the boundary must be known and placed in **vreg** before the area fill is begun.

PARAMETERS

vreg gives the value register containing the boundary pixel value. Range is 0-15.

HOST BINARY COMMAND STREAM

$[14_H][vreg]$ (2 bytes)
$14_H=024_8=20_{10}$

FORTRAN CALL

CALL AREA2 (IVREG)

EXAMPLE

!VAL8 255	Set current pixel value to 255,255,255
!PRMFIL OFF	Select unfilled primitives
!MOVABS 0 0	Move to 0,0
!CIRCLE 20	Draw circle of radius 20
!VALUE 0 0 255	Set current pixel value to 0,0,225
!CIRCLE 25	Draw circle of radius 25
!VLOAD 9 0 0 255	Load VREG 9 with 0,0,255
!AREA2 9	Begin area fill. Boundary pixel value is found in VREG 9. (The inner circle is over-written because it is not drawn in boundary pixel value.)

RELATED COMMAND

FILMSK

SYNTAX

ASCII flag

FUNCTION

The ASCII command sets the host input port. If **flag=1** the host port input command stream is interpreted as free format ASCII (as from the local alphanumeric terminal/ keyboard). If **flag=Ø**, the host port input command stream is interpreted as the default 8 bit binary or ASCII hex.

PARAMETERS

flag If **flag=1**, host is free-format ASCII. If **flag=0** host is 8 bit binary or ASCII hex (normal default).

HOST BINARY COMMAND STREAM

[$9B_H$][flag] (2 bytes)
$9B_H=233_8=155_{10}$

FORTRAN CALL

CALL ASCII (IFLAG)

SYNTAX

*

FUNCTION

The * command allows use of program comments in ALPHA mode. Any characters following the asterisk (before the carriage return) are ignored.

EXAMPLE

!*   Just a waste of typing.

SYNTAX

BLANK flag

FUNCTION

The BLANK command changes the blank flag. If **flag=1**, the screen is blanked, and the contents of image memory are no longer displayed. This frees all cycles of image memory for writing and reading operations by the CPU, vector generator, pixel processor and DMA. Vector generator, pixel processor, and DMA performance is therefore substantially improved, since all memory cycles are available.

PARAMETERS

flag   Blank the screen when **flag=1**, if **flag=Ø** unblank screen.

HOST BINARY COMMAND STREAM

$[31_H][flag]$ (2 bytes)
$31_H=061_8=49_{10}$

FORTRAN CALL

CALL BLANK (IFLAG)

EXAMPLE

!PRMFIL ON	Select filled primitives
!CIRCLE 50	Draw circle of radius 50
!BLANK 1	Blank screen
!CIRCLE 100	Draw circle of radius 100
!BLANK Ø	Unblank screen

SYNTAX

BLINKC

FUNCTION

The BLINKC command clears the blink table, and disables blink of all LUT locations. The LUT entries are set to **entryl** of their blink values.

HOST BINARY COMMAND STREAM

[23_H] (1 byte)
$23_H=043_8=35_{10}$

FORTRAN CALL

CALL BLINKC

SYNTAX

BLINKD lut,index

FUNCTION

The BLINKD command disables blinking of the LUT location specified by **lut**. The **index** gives the pixel value in image memory that will address this location in the LUT.

When the blink is disabled, the entry in the LUT will be the same as the first LUT entry in the BLINKE command which enabled it.

PARAMETERS

lut	lut=7, use all look-up-tables
	lut=1, use blue LUT
	lut=2, use green LUT
	lut=4, use red LUT
index	LUT index location; range is 0-255

HOST BINARY COMMAND STREAM

[21_H][lut][index] (3 bytes)
$21_H=041_8=33_{10}$

FORTRAN CALL

CALL BLINKE (LUT,INDEX)

SYNTAX

BLINKE lut, index, entryl, entry2

FUNCTION

The BLINKE command enables blinking of the LUT location specified by **lut.** The **index** specifies the address of the location in the LUT to blink. **entryl** and **entry2** are swapped back and forth at the rate set by the BLINKR command.

PARAMETERS

lut	lut=7, use all look-up-tables
	lut=1, use blue LUT
	lut=2, use green LUT
	lut=4, use red LUT
index	LUT index location; range is 0-255
entryl	and entry2 give LUT values to blink between; range is 0 to 255.

HOST BINARY COMMAND STREAM

[2Ø$_H$][lut][index][entryl][entry2]
$20_H=040_8=32_{10}$

FORTRAN CALL

CALL BLINKE (LUT,INDEX,IENTl,IENT2)

EXAMPLE

!BLINKE 7 100 255,125	Blink location 100 in all LUTs between 255 and 125
!BLINKC	Location 100 has value 255 in all LUTs
!BLINKE 7 100 100,255	Blink between 100 and 255
!BLINKC	Location 100 has value 100 in all LUTs

SYNTAX

BLINKR frames

FUNCTION

The BLINKR command sets the blink rate to **frames** frame times. This rate is specified as the number of frame times between swapping the two LUT entries. One frame time is 1/60 second.

PARAMETERS

frames each frame time is 1/60 second; the range is 0 to 255.

HOST BINARY COMMAND STREAM

$[22_H][\text{frames}]$ (2 bytes)

$22_H = 042_8 = 34_{10}$

FORTRAN CALL

```
CALL BLINKR (FRAMES)
```

SYNTAX

BUTTBL index,macnum

FUNCTION

The BUTTBL command is use to load the Button Table. **Index** gives the button number; **macnum** gives the macro number to execute when button number **index** is pressed.

PARAMETERS

index	button number; range is 0 to 64.
macnum	macro number; range is 0 to 255.

HOST BINARY COMMAND STREAM

$[AA_H]$[index][macnum] (3 bytes)
$AA_H=252_8=170_{10}$

FORTRAN CALL

CALL BUTTBL (INDEX,MACNUM)

EXAMPLE

!MACDEF 37	Start definition of Macro #37
$ZOOMIN	Increase scale factor by 2
$MACEND	End Macro definition
!BUTTBL 13 37	Execute Macro #37 when button #13 is pressed

RELATED COMMANDS

FLUSH

SYNTAX

BUTTON index

FUNCTION

The BUTTON command executes the Macro specified by the Button Table at location **index.** This command performs the same function as pressing function button **index** on the cursor of the digitizing tablet.

PARAMETERS

index   button number; range is 0 to 31.

HOST BINARY COMMAND STREAM

$[AB_H][index]$ (2 bytes)
$AB_H=253_8=171_{10}$

FORTRAN CALL

CALL BUTTON (INDEX)

EXAMPLE

!MACDEF 15	Begin definition of Macro #15
$ZOOMIN	Zoomin by a factor of 2x
$MACEND	End Macro definition
!BUTTBL 21 15	Execute Macro #15 when button #21 is depressed
!BUTTON 21	Simulate having pushed button #21

RELATED COMMANDS

BUTTBL

SYNTAX

CADD csum, creg

FUNCTION

The CADD command adds the contents of coordinate register **creg** to the contents of coordinate register **csum**. The result is put into coordinate register **csum**.

PARAMETERS

csum, creg   coordinate registers for addition; range is 0 to 63.

HOST BINARY COMMAND STREAM

[$A2_H$][csum][creg]
$A2_H = 242_8 = 162_{10}$

FORTRAN CALL

```
CALL CADD (ICSUM,ICREG)
```

EXAMPLE

!CLOAD 25 100 150	Load CREG 25 with 100,150
!CLOAD 26 10 20	Load CREG 26 with 10,20
!CADD 25 26	Add the contents of CREG 26 to CREG 25 and place result in CREG 25
!READCR 25	Read the contents of CREG 25
110 170	(Response from Model One)
!CADD 25 26	Add the contents of CREG 26 to CREG 25 place result in CREG 25
!READCR 25	Read contents of CREG 25
120 190	(Response from Model One)

SYNTAX

CIRCI creg

FUNCTION

The CIRCI command draws a circle with the center point of the circle at the current point (CREG Ø) and the point specified by coordinate register **creg** on the circumference of the circle. The CIRCI command is useful for drawing circles with the radius controlled by an interactive device such as the digitizing tablet.

PARAMETERS

creg   coordinate register for point on circumference; range is 0 to 63.

HOST BINARY COMMAND STREAM

[$1Ø_H$][creg]   (2 bytes)
$10_H=020_8=16_{10}$

FORTRAN CALL

CALL CIRCI (ICREG)

EXAMPLE

!MOVABS 100 100	Move to location 100,100
!CIRCI 4	Draw circle whose center is 100,100 and whose perimeter includes the location given in CREG 4.
!MOVABS 120 150	Move to location 120,150
!CLOAD 27 200 220	Load CREG 27 with 200,220
!CIRCI 27	Draw circle whose center is at 120,150 and whose perimeter includes the location given in CREG 27 (200,220)

SYNTAX

CIRCLE rad

FUNCTION

The CIRCLE command draws a circle of radius **rad**, with the center at the current point (CREG Ø). The circle is drawn in the current pixel value. A circle of radius zero sets the current point to the current pixel value.

PARAMETERS

rad the circle radius; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[ØE_H][highrad][lowrad] (3 bytes)
$0E_H=016_8=14_{10}$

FORTRAN CALL

CALL CIRCLE (IRAD)

EXAMPLE

!MOVABS 100 150	Move current point to 100,150
!CIRCLE 30	Draw circle of radius 30 centered at 100,150
!MOVREL 10 0	Move current point by 10,0 to 110,150
!CIRCLE 20	Draw circle of radius 20 centered at 110,150
!CIRCLE 10	Draw circle of radius 10 centered at 110,150

SYNTAX

CIRCXY x,y

FUNCTION

The CIRCXY command draws a circle with the center of the circle at the current point (CREG Ø) and the point **x,y** on its circumference.

PARAMETERS

x, y   coordinate (x,y) is on circumference.

HOST BINARY COMMAND STREAM

[ØF_H][highx][lowx][highy][lowy]   (5 bytes)
$0F_H=017_8=15_{10}$

FORTRAN CALL

CALL CIRCXY (IX,IY)

EXAMPLE

!MOVABS 0 0	Move current point to 0,0
!CIRCXY 30 30	Draw circle centered at 0,0 with 30,30 on its circumference
!CIRCXY 30 40	Draw circle centered at 0,0 with 30,40 on its circumference
!CIRCXY 50 50	Draw circle centered at 0,0 with 50,50 on its circumference

SYNTAX

CLEAR

FUNCTION

The CLEAR command changes all pixels in the currently defined window to the current pixel value (VREG Ø). Pixels outside the current clipping window are unchanged. The corners of the current window are held in CREGs 9 and 10.

HOST BINARY COMMAND STREAM

[87_H] (1 byte)
$87_H = 207_8 = 135_{10}$

FORTRAN CALL

CALL CLEAR

EXAMPLE

!VALUE 100 100 255	Change current pixel value to 100,100,255
!CLEAR	Clear current window to current pixel value (100,100,225)
!VAL8 0	Change current pixel value to 0,0,0
!CLEAR	Clear current window to current pixel value

RELATED COMMANDS

WINDOW
VALUE

SYNTAX

CLOAD creg,x,y

FUNCTION

The CLOAD command loads coordinate register **creg** with the given (**x,y**) coordinate.

PARAMETERS

creg	coordinate register; range is 0 to 63.
x, y	(x,y) coordinate pair; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[$A\emptyset_H$][creg][highx][lowx][highy][lowy] (6 bytes)
$A\emptyset_H = 240_8 = 160_{10}$

FORTRAN CALL

CALL CLOAD (ICREG,IX,IY)

EXAMPLE

!CLOAD 17 100 150	Load CREG 17 with 100,150
!READCR 17	Read contents of CREG 17
100 150	(Response from Model One)
!CLOAD 17 50 -50	Load CREG 17 with 50,-50
!READCR 17	Read contents of CREG 17
50 -50	(Response from Model One)

SYNTAX

CMOVE   cdst,csrc

FUNCTION

The CMOVE command copies the contents of coordinate register **csrc** to coordinate register **cdst.** Any data in **cdst** is replaced by the new data.

PARAMETERS

cdst, csrc   coordinate register; range is 0 to 63.

HOST BINARY COMMAND STREAM

[Al_H][cdst][csrc]   (3 bytes)
$Al_H=241_8=161_{10}$

FORTRAN CALL

CALL CMOVE (ICDST,ICSRC)

EXAMPLE

!CLOAD 25 100 150	Load CREG 25 with 100,150
!CLOAD 26 20 -50	Load CREG 26 with 20,-50
!READCR 26	Read contents of CREG 26
20 -50	(Response from Model One)
!CMOVE 26 25	Move contents of CREG 25 into CREG 26
!READCR 26	Read contents of CREG 26
100 150	(Response from Model One)

SYNTAX

COLD

FUNCTION

The COLD command performs a coldstart to the Model One, equivalent to pushing the START button on the back panel. COLD executes the Model One's restart sequence and diagnostics. The COLDstart state of the Model One is defined as follows:

MODDIS Ø	Display Mode of 512x512
LUTRTE Ø	Look-up-table routing (full color)
LUTRMP 7 0,255 0,255	Ramp function in all LUTs
ZOOM 1	Scale factor of 1:1
CORORG 0,0	Coordinate Origin 0,0
SCRORG 0,0	Screen Origin 0,0
WINDOW -256,-256 255,255	Clipping window set to image memory bounds
PRMFIL Ø	Unfilled primitives
MACERA 0 through MACERA 255	Erase all macros
TEXTRE	Erase all user defined fonts
VAL 8 0	Set current pixel value to 0,0,0
FLOOD	Flood image memory to 0,0,0
VAL8 255	Set current pixel value to 255,255,255
BUTTBL 0, 0 to BUTTBL 32,0	Set all button table entries to zero
DELAY 0	No delay
VECPAT #FFFF	Draw solid lines
PIXFUN 0	INSert mode
PIXCLP 0	Use wraparound on pixel clipping
WRMASK 255,7	All bit-planes write-enabled
RDMASK 255	No read masking, all planes read-enabled.

The Model One is in ALPHA mode after the COLDstart.

NOTE: If you want to reset the Model One to a state other than the above, you will want to set up a standard command to execute after the COLDstart command. In addition, if the COLD command is sent from the host, you must wait several seconds before sending any more data.

HOST BINARY COMMAND STREAM

$[FD_H]$ (1 bytes)
$FD_H=375_8=253_{10}$

FORTRAN CALL

CALL COLD

EXAMPLE

!COLD	COLDstart the Model One
Model One Firmware Rev. X	Response (after 2-3 seconds or so)

SYNTAX

CONFIG dwnlod, maclst, txtfnt, inpque

FUNCTION

The CONFIG command is used to configure central processor RAM. Each parameter in the CONFIG command specifies the number of words that are to be used for that function.

The CONFIG command should be executed immediately following a COLDstart. All affected memory space is cleared when this command is executed. This clears the Serial I/O queues, user downloaded code, macro definitions, and text font descriptions. The range of all parameters is Ø to 32,767 such that the sum of all parameters does not exceed the available memory space. Use the MAP command to check space available. If an error condition results, no action will be taken. The CONFIG command takes several seconds to execute; if the CONFIG command is sent from the host, you must wait several seconds before sending any more data.

PARAMETERS

dwnlod - specifies number of words of memory to be used for user downloaded commands.

maclst - specifies number of words of memory to be used for macro definitions.

txtfnt - specifies number of words of memory to be used for vector list description of characters when using text font 2.

inpque - specifies number of additional words of memory to be used for the host port serial input queue. The host serial input queue must be configured to a power of 2. (e.g. #100, #400, #800 . . .)

(Continued)

HOST BINARY COMMAND STREAM

$[24_H]$[highdwnlod][lowdwnlod][highmaclst][lowmaclst]
[hightxtfnt][lowtxtfnt][highinpque][lowinpque]
(9 bytes)
$24_H=044_8=36_{10}$

FORTRAN CALL

CALL CONFIG (DWNLOD,MACLST,TXTFNT,INPQUE)

DWNLOD, MACLST, TXTFNT and INPQUE are integers.

RELATED COMMANDS

MAP
COLD

SYNTAX

CORORG x,y

FUNCTION

The CORORG command sets the coordinate origin register (CREG3) to the point specified by **x, y**. The contents of this register are added to all incoming coordinates; all coordinates are relative displacements from the coordinate origin. The CORORG command resets all other coordinate registers and should be used only immediately after a COLDstart.

PARAMETERS

x, y the new coordinate origin; x and y range from -32,768 to 32,767

HOST BINARY COMMAND STREAM

[37_H][highx][lowx][highy][lowy]

$37_H = 067_8 = 55_{10}$

FORTRAN CALL

CALL CORORG (X,Y)

SYNTAX

CSUB cdif, creg

FUNCTION

The CSUB command subtracts the contents of coordinate register **creg** from the contents of coordinate register **cdif** and places the result into coordinate register **cdif.**

PARAMETERS

cdif, creg   coordinate registers; range is from 0 to 63.

HOST BINARY COMMAND STREAM

[$A3_H$][cdif][creg]   (3 bytes)
$A3_H=243_8=163_{10}$

FORTRAN CALL

CALL CSUB (ICDIF,ICREG)

EXAMPLE

!CLOAD 20 100 150	Load CREG 20 with 100,150
!CLOAD 21 25 30	Load CREG 21 with 25,30
!CSUB 20 21	Subtract the contents of CREG 21 from CREG 20 and place result in CREG 20
!READCR 20	Read contents of CREG 20
75 120	(Response from Model One)

SYNTAX

DEBUG flag

FUNCTION

The DEBUG command is used to enter and exit the command stream interpreter. When **flag is 1**, the central processor displays, in mnemonic form, the commands that are being executed by the command interpreter. If the **flag is Ø**, DEBUG is disabled.

PARAMETERS

flag   flag=1, enable Command Stream Interpreter; flag=0, disable Command Stream Translator.

HOST BINARY COMMAND STREAM

[$A8_H$][flag]

$A8_H=250_8=168_{10}$

FORTRAN CALL

CALL DEBUG (FLAG)

SYNTAX

DELAY amount

FUNCTION

The DELAY command inserts a preprogrammed delay between characters sent to the host computer over the HOSTSIO interface. The DELAY command is necessary because many host computers can not process incoming characters as fast as characters can be sent by the Model One. **Amount** specifies the amount of time to insert between characters. The Model One inserts a delay of approximately 750***amount** microseconds. The default is no delay.

PARAMETERS

amount   the amount to delay; range is from 0 to 255.

HOST BINARY COMMAND STREAM

[$B6_H$][amount]
$B6_H=266_8=182_{10}$

FORTRAN CALL

CALL DELAY (IAMT)

D F T C F G

SYNTAX

DFTCFG

FUNCTION

The DFTCFG command restores all ports on the Model One to the default configuration. In addition, it restores all special characters to the default special characters (see SPCHAR). The default configuration is:

Port mnemonic	RTS	CTS	Baud	Parity	XIN	XOUT	CTRL	STOP	NBITS
0 MODEMSIO	off	off	1200	none	on	off	off	1	8
1 KEYBDSIO	off	off	1200	none	on	off	on	1	8
2 TABLETSIO	off	off	1200	none	on	off	off	2	8
3 GRINSIO	off	off	1200	none	off	off	off	2	7
4 HOSTSIO	off	off	9600	none	off	on	off	2	8
5 ALPHASIO	off	off	9600	none	on	off	on	2	8
6 IEEE	-	-	-	-	-	-	-	-	-

The default configuration is modified through the use of the SYSCFG and SAVCFG commands. The SYSCFG command configures the Model One's ports; the SAVCFG command stores those configurations (which are then loaded whenever a COLDstart is performed).

The default configuration is not modified by SYSCFG or SAVCFG; the DFTCFG command should be used only when it is necessary to restore all the Model One's ports to a known state.

In an dire emergency, where all ports have been rendered totally incommunicado through injudicious use of the SYSCFG and SAVCFG commands, the top board includes an Internal Reset Button which restores all configurations to the default configuration above. THIS BUTTON SHOULD BE USED ONLY IN AN EMERGENCY!

To display the current configurations, you can use the DISCFG command.

The DFTCFG command can be executed only from the local alphanumeric terminal, and cannot be included in a macro.

EXAMPLES

SYSCFG SERIAL HOSTSIO RTS OFF CTS OFF XIN ON XOUT ON PARITY N
configures port HOSTSIO as indicated (see SYSCFG for details)

SAVCFG — saves the configuration of port HOSTSIO

DFTCFG — restores the default configuration for all ports (not just port HOSTSIO)

D I S C F G

SYNTAX

DISCFG

FUNCTION

The DISCFG command displays the current configurations, as set with the SYSCFG command. The DISCFG command can be executed from the local terminal only, and may not be included in a macro.

RELATED COMMANDS

SYSCFG
DFTCFG
SAVCFG

SYNTAX

DNLOAD string

FUNCTION

The DNLOAD command is used to download new Z8002 graphics commands to the Model One central processor.

PARAMETERS

string

HOST BINARY COMMAND STREAM

[FB_H][Object code and header]...
$FB_H=373_8=251_{10}$

SYNTAX

DRW2R dx,dy

FUNCTION

The DRW2R command is a two-byte version of the DRWREL command. The DRW2R command draws a vector from the current point (CREG Ø) to the point relative to the current point by **dx** and **dy** (+7 to -8) and changes the current point (CREG Ø) to this new point. This command minimizes the number of bytes which must pass between the host computer and the Model One for drawing short vectors.

PARAMETERS

dx, dy the relative distance; range is -8 to 7.

HOST BINARY COMMAND STREAM

$[84_H][dxdy]$ (2 bytes)
$84_H=204_8=132_{10}$

The most significant nibble (high four-bits) of dxdy specifies dx and the least significant nibble (low four-bits) of dxdy specifies dy.

FORTRAN CALL

CALL DRW2R (IDX,IDY)

SYNTAX

DRW3R dx, dy

FUNCTION

The DRW3R command is a three byte form of the DRWREL command. The DRW3R command draws a vector from the current point (CREG Ø) to the point relative to the current point offset by **dx** and **dy**. The current point is changed to this new point. The range of **dx** and **dy** is -128 to +127. This command reduces the number of bytes which must pass between the Model One and the host computer for vectors whose maximum displacement is within the given range.

PARAMETERS

dx, dy the relative distance; range is -128 to 127.

HOST BINARY COMMAND STREAM

$[83_H][dx][dy]$ (3 bytes)
$83_H=203_8=131_{10}$

FORTRAN CALL

CALL DRW3R (IDX,IDY)

RELATED COMMANDS

All DRW Commands
FIRSTP

SYNTAX

DRWABS x,y

FUNCTION

The DRWABS command draws a vector from the current point (CREG Ø) to the point specified by **x,y** and changes the current point (CREG Ø) to **x,y**. The pixels along the line are drawn in the current pixel value (VREG Ø).

PARAMETERS

x, y the absolute x,y coordinate; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[81_H][highx][lowx][highy][lowy]
$81_H=201_8=129_{10}$

FORTRAN CALL

CALL DRWABS (IX,IY)

EXAMPLE

!MOVABS 50 50	Move current point to 50,50
!DRWABS 60 50	Draw line to 60,50 (horizontal line 11 pixels long - both end points included).
!MOVABS 60 60	Move current point to 60,60
!DRWABS 60 70	Draw line to 60,70 (vertical line 11 pixels long).
!DRWABS 70 70	Draw line to 70,70 (diagonal line connected to previous line).
!DRWABS 80 100	Draw line to 80,100

RELATED COMMANDS

All DRW Commands
FIRSTP

SYNTAX

DRWI creg

FUNCTION

The DRWI command draws a vector from the current point (CREG Ø) to the point given be coordinate register **creg** and changes the current point (CREG Ø) to the new point.

PARAMETERS

creg   coordinate register; range is 0 to 63.

HOST BINARY COMMAND STREAM

[85_H][creg]   (2 bytes)
$85_H=205_8=133_{10}$

FORTRAN CALL

CALL DRWI (ICREG)

EXAMPLE

!MOVABS -100 -50	Move current point to -100,-50
!DRWI 4	Draw vector from -100,-50 to location given in CREG 4
!MOVABS -30 -60	Move current point to -30,-60
!CLOAD 33 100 150	Load CREG 33 with 100,150
!DRWI 33	Draw vector from current point (-30,-60) to location given in CREG 33 (100,150)

RELATED COMMANDS

All DRW Commands
FIRSTP

SYNTAX

FILMSK   rmsk, gmsk, bmsk

FUNCTION

The FILMSK command sets the fill mask (VREG 3) with the value specified by **rmsk, gmsk, bmsk.** The FILMSK is AND'ed with boundary values before the boundary check comparison is made. The FILMSK command is equivalent to VLOAD 3 **rmsk gmsk bmsk**, which loads VREG 3 with the fill mask.

PARAMETERS

rmsk	red mask; range is 0 to 255.
gmsk	green mask; range is 0 to 255.
bmsk	blue mask; range is 0 to 255.

HOST BINARY COMMAND STREAM

$[9F_H]$[rmsk][gmsk][bmsk]   (4 bytes)
$9F_H=237_8=159_{10}$

FORTRAN CALL

CALL FILMSK (IRMSK,IGMSK,IBMSK)

EXAMPLE

!MOVABS 0,0	Move current point to 0,0
!VAL8 255	Set current pixel value to 255,255,255
!CIRCLE 50	Draw circle of radius 50
!VALUE 0 0 255	Set current pixel value to 0,0,255
!CIRCLE 40	Draw circle of radius 40
!VALUE 0 255 0	Set current pixel value to 0,255,0
!CIRCLE 30	Draw circle of radius 30
!FILMSK 255 0 0	Set fill mask to 255,0,0 (ignore green and blue)
!AREA1	Fill area, green and blue circles are ignored because of ANDing with fill mask

RELATED COMMANDS

AREA1
AREA2

SYNTAX

DRWREL dx,dy

FUNCTION

The DRWREL command draws a vector from the current point (CREG Ø) to the point relative to the current point offset by **dx** and **dy.** The current point is set to the sum of the x-component of the old current point plus **dx** and the sum of the y-component of the old current point plus **dy.**

PARAMETERS

dx, dy relative offset for the coordinate; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[82_H][highdx][lowdx][highdy][lowdy] (5 bytes)
$82_H=202_8=130_{10}$

FORTRAN CALL

CALL DRWREL (IDX,IDY)

EXAMPLE

!MOVABS 50 30	Move to location 50,30
!DRWREL 10 20	Draw line from 50,30 to 60,50
!DRWREL 10 0	Draw line from 60,50 to 70,50
!DRWREL 0 10	Draw line from 70,50 to 70,60

RELATED COMMANDS

All DRW Commands
FIRSTP

SYNTAX

FIRSTP flag

FUNCTION

The FIRSTP command inhibits the writing of the first pixel on vectors when the **flag=1.** If **flag=Ø,** all pixels are written. This prevents shared endpoints of concatenated lines from being written twice into image memory.

PARAMETERS

flag   flag=1, inhibit writing of first pixel; flag=0, write all pixels.

HOST BINARY COMMAND STREAM

[$2F_H$][flag]   (2 bytes)
$2F_H$=057_8=47_{10}

FORTRAN CALL

CALL FIRSTP (IFLAG)

RELATED COMMANDS

All DRW Commands

SYNTAX

FLOOD

FUNCTION

The FLOOD command changes all displayed pixels to the current pixel value (VREG Ø) in a single frame time. Pixels that are not being displayed when the FLOOD command is issued are not changed. FLOOD does not affect the overlay planes.

HOST BINARY COMMAND STREAM

[$Ø7_H$] (1 byte)
$07_H=007_8=7_{10}$

FORTRAN CALL

CALL FLOOD

EXAMPLE

!VALUE 100 255 200	Change current pixel value to 100,255,200
!FLOOD	Flood displayed image
!VAL8 0	Change current pixel value to 0,0,0
!FLOOD	Flood displayed image
!VALUE 255 0 0	Change current pixel value to 255,0,0
!FLOOD	Flood displayed image

RELATED COMMANDS

CLEAR
ZOOM
ZOOMIN

SYNTAX

FLUSH

FUNCTION

The FLUSH command empties the function button event queue. The event queue keeps a record of all function buttons which have been pushed at the workstation. Each time the host issues a READBU command, one entry is taken out of the event queue and sent to the host. If the host is not interested in knowing which buttons have been pushed at the workstation, the FLUSH command should be included in macros which use function button. The event queue holds eight entries. Overflow of the event queue results in subsequent function buttons being ignored.

HOST BINARY COMMAND STREAM

[15_H] (1 byte)
$15_H = 025_8 = 21_{10}$

FORTRAN CALL

CALL FLUSH

EXAMPLE

!MACDEF 10	Start definition of Macro #10
$ZOOMIN	Increase scale factor by 2X
$FLUSH	Empty event queue
$MACEND	End macro definition
!BUTTBL 13 10	Execute Macro #10 in response to Button #13

RELATED COMMANDS

All BUTTON Commands

SYNTAX

HOSTO strlen, string

FUNCTION

The HOSTO (HOST Output) command outputs a text string to the host computer over the currently selected host interface. The text is specified by **string.** If the command is being entered in ASCII mode from the local alphanumeric terminal or keyboard, the **string** to be output is the set of ASCII characters remaining on the command line. If the command is not being sent in ASCII mode, then the first byte of **string** contains the number of characters in the string **(strlen)** followed by **strlen** bytes containing the ASCII characters to be drawn.

PARAMETERS

strlen the number of bytes; needed only if command not sent in ASCII mode.
string the text string.

HOST BINARY COMMAND STREAM

[$B5_H$][strlen]([char1][char2]...[charn])
$B5_H=265_8=181_{10}$

FORTRAN CALL

CALL HOSTO (STRLEN,STRING)

STRLEN is an integer specifying the number of characters that are to be output. STRING is an integer array with two characters packed per 16-bit word, as in FORTRAN "A2" format.

EXAMPLE

!HOSTO ABCDEF 1 2 3	Output text string "ABCDEF 1 2 3" to the host
!HOSTO WXYZ	Output text string "WXYZ" to the host

RELATED COMMANDS

ALPHAO

SYNTAX

LIGHTS b1, b2, b3, b4

FUNCTION

The LIGHTS command controls the lights on the Graphic Input Box by setting the 32 bit light mask. The high eight bits of the light mask are specified by **b1**. The low eight bits are specified by **b4**. If a bit in the mask is set, the corresponding button on the Model One's optional Graphic Input Box will be lighted. The range of **b1** through **b4** is Ø to 255.

PARAMETERS

b1, b2, b3, b4	the 32-bit light mask; b1 holds the high 8 bits, b2 the next 8 bits, b3 the next, and b4 the low 8 bits.

HOST BINARY COMMAND STREAM

[AC_H][b1][b2][b3][b4] (5 bytes)
$AC_H=254_8=172_{10}$

FORTRAN CALL

CALL LIGHTS (IB1,IB2,IB3,IB4)

RELATED COMMANDS

All BUTTON Commands

SYNTAX

LUT8 index, rentry, gentry, bentry

FUNCTION

The LUT8 command changes the entries in all three Look-Up-Tables (LUTs) at the location specified by **index** to the new values **rentry, gentry, bentry.** The Red LUT is loaded with **rentry,** the Green with **gentry,** and the Blue with **bentry.** The entries are the values that are stored in the red, green and blue LUTs that will be loaded into their respective digital-to-analog converters (DACs) when a pixel of value entry is encountered when screen refresh is being performed. The LUT8 command is most useful when the LUT input routing (LUTRTE command)is set to other than its default value.

PARAMETERS

index	the LUT location to be set; range is 0 to 255.
rentry	red entry; range is 0 to 255.
gentry	green entry; range is 0 to 255.
bentry	blue entry; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$1C_H$][index][rentry][gentry][bentry]
$1C_H=034_8=28_{10}$

FORTRAN CALL

CALL LUT8 (INDEX,IRENT,IGENT,IBENT)

EXAMPLE

!VAL8 100	Change current pixel value to 100,100,100
!FLOOD	Flood displayed pixels to current pixel value
!LUT8 100 50 100 200	Change location 100 in red LUT to 50 in green LUT to 100, and blue LUT to 200
!LUT8 100 200 70 30	Change location 100 in red LUT to 200, green LUT to 70, blue LUT to 30

SYNTAX

LUTA index, entry

FUNCTION

The LUTA command makes three identical entries in all three Look-Up-Tables. The value stored in the red, green and blue LUTs is passed to each of the digital-to-analog converters (DACs) when a pixel of value **index** is encountered when reading from image memory to refresh the display screen.

PARAMETERS

index   the LUT location; range is 0 to 255.
entry   the entry at the LUT location; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$1B_H$][index][entry]   (3 bytes)
$1B_H = 033_8 = 27_{10}$

FORTRAN CALL

CALL LUTA (INDEX,IENTRY)

EXAMPLE

!VAL8 255	Set current pixel value to 255,255,255
!FLOOD	Flood displayed pixels to current pixel value (screen goes white if LUTA has not been set otherwise)
!LUTA 255 0	Change entry in location 255 of the red, green, and blue LUTS to 0 (screen goes black)
!LUTA 255 100	Change entry in location 255 of the red, green, and blue LUTS to 100 (screen goes to grey)

SYNTAX

LUTB   index, entry

FUNCTION

The LUTB command changes the entry in the Blue Look-Up-Table (LUT) at location **index** to the new value **entry.** The entry stored in the LUT is passed to the blue digital-to-analog converter (DAC) when a pixel of value **index** is encountered when reading from image memory to refresh the display screen.

PARAMETERS

index   the blue LUT location; range is 0 to 255.
entry   the entry at the location; range is 0 to 255.

HOST BINARY COMMAND STREAM

$[1A_H][index][entry]$ (3 bytes)
$1A_H=032_8=26_{10}$

FORTRAN CALL

CALL LUTB (INDEX,ENTRY)

EXAMPLE

!VALUE 0 0 100	Set current pixel value to 0,0,100
!FLOOD	Flood all displayed pixels to current pixel value
!LUTB 100,0	Change entry in blue LUT location 100 to 0 (black)
!LUTB 100 255	Change entry in blue LUT location 100 to 255 (full intensity)
!VAL8 0	Change current pixel value to 0,0,0
!FLOOD	Flood all displayed pixels to current pixel value
!LUTB 0 100	Change entry in blue LUT location 0 to 100

SYNTAX

LUTG index, entry

FUNCTION

The LUTG command changes the entry in the Green Look-Up-Table (LUT) at location **index** to the new value **entry.** The entry stored in the LUT is passed to the green digital-to-analog converter (DAC) when a pixel of value **index** is encountered when reading from image memory to refresh the display screen.

PARAMETERS

index	the green LUT location; range is 0 to 255.
entry	the entry at the location; range is 0 to 255.

HOST BINARY COMMAND STREAM

[19_H][index][entry]
$19_H=031_8=25_{10}$

FORTRAN CALL

CALL LUTG (INDEX,ENTRY)

EXAMPLE

!VALUE 0 100 0	Set current pixel value to 0,100,0
!FLOOD	Flood all displayed pixels to current pixel value
!LUTG 100,0	Change entry in green LUT location 100 to 0 (black)
!LUTG 100 255	Change entry in green LUT location 100 to 255 (full intensity)
!VAL8 0	Change current pixel value to 0,0,0
!FLOOD	Flood all displayed pixels to current pixel value
!LUTG 0 100	Change entry in green LUT location 0 to 100

SYNTAX

LUTR index, entry

FUNCTION

The LUTR command changes the entry in the Red Look-Up-Table (LUT) at location **index** to the new value **entry.** The entry stored in the LUT is passed to the red digital-to-analog converter (DAC) when a pixel of value **index** is encountered when reading from image memory to refresh the display screen.

PARAMETERS

index	the red LUT location; range is 0 to 255.
entry	the entry at the location; range is 0 to 255.

HOST BINARY COMMAND STREAM

$[18_H]$[index][entry]   (3 bytes)
$18_H = 030_8 = 24_{10}$

FORTRAN CALL

```
CALL LUTR (INDEX,IENTRY)
```

EXAMPLE

!VALUE 100 0 0	Set current pixel value to 100,0
!FLOOD	Flood all displayed pixels to current pixel value
!LUTR 100,0	Change entry in red LUT location 100 to 0 (black)
!LUTR 100 255	Change entry in red LUT location 100 to 255 (full intensity)
!VAL8 0	Change current pixel value to 0,0,0
!FLOOD	Flood all displayed pixels to current pixel value
!LUTR 0 100	Change entry in red LUT location 0 to 100

SYNTAX

LUTRMP num, sind, eind, sent, eent

FUNCTION

The LUTRMP command loads the Look-Up-Tables (LUT's) specified by **num** from LUT index **sind** to LUT index **eind** with a ramp function linearly interpolated from the start entry **sent** to the end entry **eent**. The LUTRMP command is useful whenever multiple, successive look-up-table entries are to be set to either a ramp function or to a constant value.

PARAMETERS

num	The LUT(s) to load: num=1, load blue LUT; num=2, load green LUT; num=4, load red LUT; num=7, load all LUTs.
sind	start index; range is 0 to 255.
eind	end index; range is 0 to 255.
sent	start entry; range is 0 to 255.
eent	end entry; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$1D_H$][num][sind][eind][sent][eent] (6 bytes)

$1D_H=035_8=29_{10}$

FORTRAN CALL

```
CALL LUTRMP (NUM,ISIND,IEIND,ISENT,IEENT)
```

(Continued)

EXAMPLE

!PRMFIL ON	Select filled primitives
!VAL8 255	Set current pixel value to 255,255,255
!MOVABS 0 0	Move current point to 0,0
!CIRCLE 110	Draw circle of radius 110
!VAL8 200	Set current pixel value to 200,200,200
!CIRCLE 90	Draw circle of radius 90
!VAL8 150	Set current pixel value to 150,150,150
!CIRCLE 70	Draw circle of radius 70
!VAL8 100	Set current pixel value to 100,100,100
!CIRCLE 50	Draw circle of radius 50
!VAL8 50	Set current pixel value to 50,50,50
!CIRCLE 30	Draw circle of radius 30
!VAL8 0	Set current pixel value to 0,0,0
!CIRCLE 10	Draw circle of radius 10
!LUTRMP 7 0 255 255 0	Load locations 0-255 in all LUTS to value of 255-0 (reverse ramp function)
!LUTRMP 1 0 255 0 255	Load blue LUT locations 0-255 with values of 0-255 (ramp function)
!LUTRMP 2 0 255 0 0	Load green LUT location 0-255 with 0
!LUTRMP 4 0 100 255 255	Load red LUT locations 0-100 with 255
!LUTRMP 7 0 255 0 255	Restore default LUT for 24 bit system

SYNTAX

LUTRTE func

FUNCTION

The LUTRTE command changes the routing of data between the RED, GREEN and BLUE banks of image memory and the red, green and blue look-up-tables within the Model One. The parameter **func** specifies the input routing. The most useful values of **func** are:

COMMAND	PURPOSE	RESULT
!LUTRTE 0	full-color imaging	RED bank drives red LUT GREEN bank drives green LUT BLUE bank drives blue LUT
!LUTRTE #7E	pseudo-color imaging	RED bank drives red, green, and blue LUTs
!LUTRTE #75	pseudo-color imaging	GREEN bank drives red, green, and blue LUTs
!LUTRTE #53	pseudo-color imaging	BLUE bank drives red, green, and blue LUTs

Sections 4.1 and 4.2 give more information about using the LUTRTE commands.

PARAMETERS

func input routing; range is 0 to 127 (00 to 7F hex).

HOST BINARY COMMAND STREAM

[$1E_H$][func] (2 bytes)
$1E_H=036_8=30_{10}$

FORTRAN CALL

CALL LUTRTE (IFUNC)

SYNTAX

MACDEF num

FUNCTION

The MACDEF command defines a new Macro specified by **num.** After entering the MACDEF command, a series of commands is entered. A Macro ends with a MACEND command. A Macro may include any combination of valid command strings (commands and parameters) including nested MACDEF commands and user-defined commands. Macros cannot contain QUIT or ASCII commands. The length of a Macro command is limited only by the available memory space. See Section 10 for extensive examples.

Up to sixteen Macros can be nested.

PARAMETERS

num

HOST BINARY COMMAND STREAM

[$8B_H$][num] (2 bytes)
$8B_H = 213_8 = 139_{10}$

FORTRAN CALL

CALL MACDEF (NUM)

EXAMPLE

!MACDEF 40	Start definition of Macro #40
$CIRCLE 50	Draw circle of radius 50
$CIRCLE 40	Draw circle of radius 40
$MACEND	End Macro definition
!MACRO 40	Execute Macro #40

RELATED COMMANDS

CONFIG
MACRO
MACEND
MACERA

SYNTAX

MACEND

FUNCTION

The MACEND command ends a Macro definition. If no Macro is being defined, an error results. A MACEND command must be used for each MACDEF command.

HOST BINARY COMMAND STREAM

[ØC$_H$] (1 byte)
$0C_H = 014_8 = 12_{10}$

FORTRAN CALL

CALL MACEND

EXAMPLE

!MACDEF 17	Start definition of Macro #17
$MOVABS 50 50	Move current point to 50,50
$DRWABS 100 150	Draw to 100,150
$MACEND	End Macro definition
!MACRO 17	Execute Macro #17

SYNTAX

MACERA num

FUNCTION

The MACERA command clears Macro definition number **num.** Macro number **num** cannot be executed after the MACERA command has been issued.

PARAMETERS

num the Macro number; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$8C_H$][num] (2 bytes)
$8C_H=214_8=140_{10}$

FORTRAN CALL

CALL MACERA (NUM)

EXAMPLE

!MACDEF 23	Define Macro #23
$ZOOMIN	Zoom in by factor of 2
$MACEND	End Macro definition
!MACRO 23	Execute Macro 23 (Zooms in)
!MACERA 23	Erase the definition of Macro #23
!MACRO 23	Execute Macro 23 (has no effect now)

SYNTAX

MACRO num

FUNCTION

The MACRO command executes Macro number **num**. Note that Macros can also be executed by pressing buttons or by using the BUTTON command. Executing a macro that has not been defined has no effect.

PARAMETERS

num the Macro number; range is 0 to 255.

HOST BINARY COMMAND STREAM

[ØBH][num]
$0B_H=013_8=11_{10}$

FORTRAN CALL

CALL MACRO (NUM)

EXAMPLE

!MACDEF 23	Define Macro 23
!ZOOMIN	
$MACEND	
!MACRO 23	Execute Macro 23

RELATED COMMANDS

All Macro commands
BUTTON
BUTTBL

SYNTAX

MAP

FUNCTION

The MAP command displays the ASCII text memory map at the local alphanumeric terminal; the map shows the RAM, ROM, and control register areas used by the firmware.

HOST BINARY COMMAND STREAM

[FC_H] (1 byte)
$FC_H=374_8=252_{10}$

FORTRAN SUBROUTINE CALL

CALL MAP

EXAMPLE

!MAP		Display a memory map
8000	9900	Monitor area
9900	BC00	Temporary data area
BC00	CC00	HOSTSIO input queue
CC00	DC00	User defined fonts area
DC00	FC00	Macro definition area
FC00	0000	System stack

RELATED COMMANDS

DNLOAD
PEEK
POKE
CONFIG

SYNTAX

MODDIS flag

FUNCTION

The MODDIS command changes the display addressing mode. If the **flag=Ø**, the display mode is set to 512x512. If the **flag=1**, the display mode is set to 1Kx1K. The image memory is cleared to a pixel value of 0,0,0 whenever the display mode is changed.

PARAMETERS

flag display mode flag; flag=0, mode is 512 x 512; flag=1, mode is 1K x 1K

HOST BINARY COMMAND STREAM

$[2C_H][flag]$ (2 bytes)
$2C_H=054_8=44_{10}$

FORTRAN SUBROUTINE CALL

CALL MODDIS (IFLAG)

EXAMPLE

!MODDIS Ø	Select 512x512 mode
!CIRCLE 200	Draw circle of radius 200
!MOVABS -256 -256	Move current point to -256,-256
!RECT 255 255	Draw rectangle whose corners are at (-256,-256) (255,-256) (255,255) (-256,255)
!MOVABS 0 0	Move current point to 0,0
!TEXT1 512 MODE	Draw text string
!MODDIS 1	Select 1K mode
!CIRCLE 200	Draw circle of radius 200 (note smaller size)
!CIRCLE 400	Draw circle of 400
!MOVABS -512 -512	Move current point to -512,-512
!RECT 511 511	Draw rectangle whose corners are at (-512,-512) (511,-512) (511,511) (-512,511)
!MOVABS 0 0	Move current point to 0,0
!TEXT1 1K MODE	Draw text string (note its smaller size)
!MODDIS Ø	Restore 512 mode

SYNTAX

MODE1K func

FUNCTION

The data routing of the pixel data in 1K mode is selected by **func.** When 1K mode is selected, the standard red, green and blue look-up-tables are no longer used. To increase the flexibility of 1K mode, several display options are available. If the function, specified by **func**, is Ø, the two bits of red, two bits of green and two bits of blue image memory data is routed directly to the DAC's.

In the other modes, the image memory is organized as two separate 3 bit per pixel images using 1 bit per primary color (red, green, blue). These 3 bits are used to turn on and off each of the three primary colors.

Func=1 will display 1K image Ø and 1K image 1 overlayed on top of one another. If **func=2**, 1K image Ø will be displayed. Writing into this 1K image plane is controlled by bit 7 (MSB) of the write enable mask. If **func is 3**, 1K image 1 will be displayed (controlled by bit 6 of the write enable mask).

PARAMETERS

func the data routing function in 1K mode; range is 0 to 3.

HOST BINARY COMMAND STREAM

$[2D_H][func]$ (2 bytes)
$2D_H=055_8=45_{10}$

FORTRAN SUBROUTINE CALL

CALL MODE1K (IFUNC)

SYNTAX

MOV2R dx,dy

FUNCTION

The MOV2R command changes the current point (CREG Ø) by a relative amount specified by **dx,dy.** The MOV2R command is a two-byte form of the MOVREL command. MOV2R reduces the number of bytes which must be sent from the host computer to the Model One to specify displacements of the current point which are very small (within the range -8 to +7 in both dx and dy).

PARAMATERS

dx,dy   relative offset; range is -8 to 7.

HOST BINARY COMMAND STREAM

[Ø4_H][dxdy]   (2 bytes)
$04_H=004_8=4_{10}$

The most significant nibble (high four-bits) of dxdy specifies dx and the least significant nibble (low four-bits) specifies dy.

FORTRAN SUBROUTINE CALL

CALL MOV2R (IDX,IDY)

SYNTAX

MOV3R dx,dy

FUNCTION

The MOV3R command changes the current point (CREG Ø) by the relative amount specified by **dx,dy**. The MOV3R command is a three-byte form of the MOVREL command. MOV3R reduces the number of bytes which the host must send to the Model One when the displacement of the current point is within the -128 to +127 range in both dx and dy.

PARAMETERS

dx,dy   the relative offset; range is -128 to 127.

HOST BINARY COMMAND STREAM

[Ø3_H][dx][dy]   (3 bytes)

03_H=003_8=3_{10}

FORTRAN SUBROUTINE CALL

CALL MOV3R (IDX,IDY)

SYNTAX

MOVABS x,y

FUNCTION

The MOVABS command changes the current point (CREG Ø) to the point specified by **x,y.** All subsequent graphics primitives (lines, circles, arcs, polygons...) are drawn beginning at the location of the current point.

PARAMETERS

x,y   the x,y coordinate; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[Ø1_H][highx][lowx][highy][lowy]   (5 bytes)
$01_H=001_8=1_{10}$

FORTRAN SUBROUTINE CALL

CALL MOVABS (IX,IY)

EXAMPLE

!MOVABS 50 70	Move the current point to 50,70
!DRWABS 100 -10	Draw line from current point (50,70) to 100,-10.
!CIRCLE 15	Draw a circle of radius 15.
!MOVABS 0 0	Move the current point to 0,0
!CIRCLE 20	Draw a circle of radius 20.

SYNTAX

MOVI creg

FUNCTION

The MOVI command changes the current point (CREG Ø) to the address specified in coordinate register **creg**. This command effectively performs a 'CMOVE Ø creg', which copies a given coordinate register into CREG Ø. The MOVI command is most often used to access the current coordinate from the digitizing tablet which is stored in CREG 2.

PARAMETERS

creg coordinate register; range is 0 to 63.

HOST BINARY COMMAND STREAM

[Ø5_H][creg] (2 bytes)
$05_H=005_8=5_{10}$

FORTRAN SUBROUTINE CALL

CALL MOVI (ICREG)

EXAMPLE

!CLOAD 15 100 150	Load 100,150 into CREG 15
!VALUE 255 255 255	Set current pixel value to 255,255,255
!MOVI 15	Move to location given in CREG 15
!DRWABS 140 100	Draw line from current point (100,150) to 140,100
!MOVI 2	Move to the location given in CREG 2 (the current digitizing tablet location)
!CIRCLE 25	Draw circle of radius 25 at current point

SYNTAX

MOVREL dx,dy

FUNCTION

The MOVREL command changes the current point (CREG Ø) by a relative amount specified by **dx, dy**. The new current point is equal to the sum of the x-component of the old current point plus dx and the sum of the y-component of the old current point plus dy.

PARAMETERS

dx,dy the relative offset; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[Ø2_H][highdx][lowdx][highdy][lowdy] (5 bytes)
02_H=002_8=2_{10}

FORTRAN SUBROUTINE CALL

CALL MOVREL (IDX,IDY)

EXAMPLE

!MOVABS 100 -130	Move the current point to 100,-130
!MOVREL 50 100	Move current point by 50,100 to 150,-30
!CIRCLE 30	Draw circle of radius 30 centered at current point
!MOVREL 20 20	Move current point by 20,20 to 170,10
!CIRCLE 10	Draw circle of radius 10 centered at current point
!MOVREL -20 -20	Move current point by -20,-20 to 150,-30
!CIRCLE 25	Draw circle of radius 25 centered at current point

SYNTAX

NULL

FUNCTION

The NULL command is analogous to a NOP (No OPeration) and has no effect. The NULL command has several opcodes (00_H,$0A_H$,$0D_H$,80_H,$8A_H$,$8D_H$), all of which execute the NULL command. The NULL command can be used to pad a command data buffer so that the Model One can be used on systems capable of transmitting only fixed length blocks.

The opcodes of the NULL command were chosen so that the Model One ignores carriage returns and line-feeds sent between commands for hosts that can not be made to inhibit sending these characters.

HOST BINARY COMMAND STREAM

[00_H or $0A_H$ or $0D_H$ or 80_H or $8A_H$ or $8D_H$]
$00_H=000_8=0_{10}$

FORTRAN SUBROUTINE CALL

CALL NULL

EXAMPLE

!NULL Just a waste of typing

RELATED COMMANDS

*(Asterisk)

SYNTAX

OVRRD* plane, flag

FUNCTION

The OVRRD command sets the display mode of the specified overlay plane. **Plane** is either 1 or Ø to specify overlay plane 1 or Ø.

If **flag is Ø** then the plane specified will not be displayed on the screen. If the **flag is 1** then the plane will be displayed. The default is to display neither plane.

PARAMETERS

plane	plane specifies overlay plane 1 or Ø
flag	display flag: flag=0, do not display; flag=1, display

HOST BINARY COMMAND STREAM

[BA_H][plane][flag] 3 bytes

FORTRAN SUBROUTINE CALL

CALL OVRRD (IPLANE,IFLAG)

EXAMPLE

OVRRD Ø Ø	Overlay plane Ø will not be displayed.
OVRRD 1 1	Overlay plane 1 will be displayed.

Note that display is further controlled by display rules. If both planes are zero, image memory is displayed; if one is zero and the other is one, the plane with the one is displayed; if both are one (and OVRRD does not inhibit display), overlay plane Ø is displayed.

* Option Card users only.

SYNTAX

OVRVAL* plane, flag

FUNCTION

The OVRVAL command sets the value for writes into the specified overlay plane. Plane is either 1 or Ø to specify overlay plane 1 or Ø.

If **flag is** Ø then all vectors drawn into the specified plane will reset the bits in that plane to zero. If **flag is** 1, then vectors will set bits in that plane to one. The default is to write ones into both overlay planes.

PARAMETERS

plane	plane specifies overlay plane 1 or Ø.
flag	write mode flag: flag=0, set bits to Ø; flag=1, set bits to 1.

HOST BINARY COMMAND STREAM

[$B9_H$][plane][flag] 3 bytes

FORTRAN SUBROUTINE CALL

CALL OVRVAL (IPLANE,IFLAG)

EXAMPLE

!OVRVAL Ø 1	Causes writes into overlay plane Ø to set bits in that plane.
!OVRVAL 1 Ø	Causes writes into overlay plane 1 to reset bits in that plane.

* Option Card users only.

SYNTAX

OVRZM* plane, flag

FUNCTION

The OVRZM command sets the zoom factor for the specifed overlay plane. **Plane** is either 1 or Ø to specify overlay plane 1 or Ø.

If **flag is Ø** then the specified plane will be displayed at a scale of 1:1. If **flag is 1,** the plane will be displayed at the same scale as image memory. The default is to display both planes at scale 1:1.

PARAMETERS

plane	plane specifies overlay plane Ø or 1.
flag	zoom scale flag: flag=0, display scale is 1:1; flag=1, display at image memory scale.

HOST BINARY COMMAND STREAM

[$B8_H$][plane][flag] 3 bytes

FORTRAN SUBROUTINE CALL

CALL OVRZM (IPLANE,IFLAG)

EXAMPLE

!OVRZM Ø 1	Causes overlay plane Ø to be displayed at same scale as image memory.
!OVRZM 1 Ø	Causes overlay plane 1 to be displayed at a scale of 1:1.

* Option Card users only.

SYNTAX

PEEK addr

FUNCTION

The PEEK command sends the contents of the central processor memory location **addr** to the requesting device. The address must be an even number (word address), the data is displayed in ASCII hexadecimal.

PARAMETERS

addr   word address

HOST BINARY COMMAND STREAM

[BD_H][highaddr][lowaddr]   (3 bytes)
$BD_H=275_8=189_{10}$

FORTRAN SUBROUTINE CALL

CALL PEEK (IADDR, IWORD)

IADDR and IWORD are integers. IWORD is changed by the call. IWORD is set equal to the contents of the location specified by IADDR.

EXAMPLE

!PEEK Ø	Display contents of location 0
FF00	
!PEEK #0FFE	Display contents of location $0FFE_H$
FØAØ	

RELATED COMMANDS

DNLOAD
MAP
POKE
CONFIG

SYNTAX

PIXCLP flag

FUNCTION

The PIXCLP command sets the pixel clipping status. When **flag=1,** the pixel processor will clip to 255 on overflow conditions and clip to Ø on underflow conditions. If the **flag=Ø,** the Pixel Processor will perform all computations MODULUS 256 for 512 mode.

PARAMETERS

flag clipping flag: flag=1, clip; flag=0, disable clipping.

HOST BINARY COMMAND STREAM

[$3C_H$][flag] (2 bytes)
$3C_H=074_8=60_{10}$

FORTRAN SUBROUTINE CALL

CALL PIXCLP (IFLAG)

SYNTAX

PIXEL8 nrows,ncols, [val1, val2 . . . valn]

FUNCTION

The PIXEL8 command transmits an image to the Model One pixel-by-pixel. The array of data is **nrows** high and **ncols** wide. The upper left corner of the array is defined by the current point (CREG Ø). Pixels in image memory are filled left-to-right, top-to-bottom. Each pixel value is sent as an 8-bit quantity, as in the VAL8 command. The PIXEL8 command is most useful in loading one of the image memory banks with pixel data, as in an 8-bit pseudo-color application.

PARAMETERS

nrows,ncols, [val1, val2, . . . valn]

HOST BINARY COMMAND STREAM

[29_H][highnrows][lownrows][highncols]
[lowncols][val].. (5+nrows*ncols bytes)
$29_H=051_8=41_{10}$

FORTRAN SUBROUTINE CALL

CALL PIXEL8(INROWS,INCOLS,IPIXELS)

NROWS and **NCOLS** are integers which specify the number of rows and columns in the rectangle to be filled.

IPIXELS is an integer array which has the pixel values. The first pixel is in the first element of **IPIXELS**. Each subsequent pixel is stored in successive array elements. **IPIXELS** contains at least **NROWS*NCOLS** elements.

SYNTAX

PIXELS nrows,ncols, [r,g,b]. . .[r,g,b]

FUNCTIONS

The PIXELS command transmits an image to the Model One pixel-by-pixel. The array of data is **nrows** high and **ncols** wide. The upper left corner of the array is given by the current point (CREG Ø). Pixels in image memory are filled left-to-right, top-to-bottom. Each pixel value is sent as a full, 24-bit quantity, one byte each of red, green and blue.

PARAMETERS

nrows,ncols, [r,g,b]. . .[r,g,b]

HOST BINARY COMMAND STREAM

$[28_H]$[highnrows][lownrows][highncols]
[lowncols][r][g][b]. . . (5+3*nrows*ncols bytes)
$28_H=050_8=40_{10}$

FORTRAN SUBROUTINE CALL

CALL PIXELS(NROWS,NCOLS,IRED,IGRN,IBLU)

Where **NROWS** and **NCOLS** are integers which specify the number of rows and columns in the rectangle to be filled.

IRED, IGRN and **IBLU** are integer arrays which contain the pixel values. Each element of the array has a single one-byte pixel value in its least significant eight bits. Each array must be dimensioned to at least NROWS*NCOLS elements. The RUNLEN subroutine encodes the data found in the arrays into run-length form automatically.

SYNTAX

PIXFUN mode

FUNCTION

The PIXFUN command sets the pixel processor mode. All operations which affect the image memory (with the exception of FLOOD) are performed by the pixel processor. These include graphics primitives draw by the vector generator, pixel mover, and DMA write operations.

The operation to be performed by the pixel processor is specified by **mode**. A character string can be substituted for the mode number when entered from the local alphanumeric terminal. Valid character strings are:

Function	Mode	Operation
INS	mode=Ø	- Directly insert new data. (default)
SUBI	mode=1	- Subtract image data from new data.
SUBN	mode=2	- Subtract new data from image data.
ADD	mode=3	- Add new data to image data.
XOR	mode=4	- XOR new data to image data.
OR	mode=5	- OR new data to image data.
AND	mode=6	- AND new data to image data.
PRESET	mode=7	- Write all ones into image memory.
CONDITIONAL	mode=8	- Inhibit writing of all pixels whose value is 0,0,0.

Note that ADD, SUBI, and SUBN are not available in 1K addressing mode. PRESET and CONDITIONAL are not available with DMA and PIXMOV.

PARAMETERS

mode   the mode may be 0 to 8; it can also be given as a character string.

HOST BINARY COMMAND STREAM

$[3B_H][\text{mode}]$ (2 bytes)
$3B_H=073_8=59_{10}$

FORTRAN SUBROUTINE CALL

CALL PIXFUN (MODE)

EXAMPLES

!PIXFUN 4   mode is XOR of new data with image data.
!PIXFUN INS   mode is insertion of new data.

SYNTAX

PIXMOV*

FUNCTION

The PIXMOV command initiates a Pixel Mover transfer. This command moves the block of pixels in the window specified by CREGs 11 and 12 into the window specified by CREGs 13 and 14. The windows are of the same size, and the size is specified by CREGS 11 and 12 (the source). CREG 13 corresponds to CREG 11; the pixel at the location in CREG 11 is transferred to the location in CREG 13. CREG 14 indicates the direction in which the window extends from CREG 13. The PMCTL and PIXFUN commands not including 7, 8, control the data routing and functions applied to the moved data. Note that judicious setting of CREGs 13 and 14 allows mirroring around the X and Y axes.

HOST BINARY COMMAND STREAM

[BB_H] 1 byte

FORTRAN SUBROUTINE CALL

CALL PIXMOV (MODE)

EXAMPLE

!CLOAD 13 -256 255	Sets up destination window.
!CLOAD 14 -156 155	
!CLOAD 11 -50 50	Sets up source window.
!CLOAD 12 50 -50	
!PIXMOV	Moves data in center window up to upper left corner.
!CLOAD 13 -156	Sets up new destination window.
!CLOAD 14 -256 255	
!PIXMOV	Moves data in center window to upper left corner upside down and flipped left to right.

*Option Card users only.

SYNTAX

PMCTL* Ø Ø Ø Ø redrte greenrte bluerte

FUNCTION

The PMCTL command sets up the routing for pixel mover operations.

Redrte is a 2 bit value indicating the data to be written into the red bank:

Ø: load no data into the red.
1: load red data into the red.
2: load green data into the red.
3: load blue data into the red.

Similarly, **greenrte** is a 2 bit value indicating which data will be written into the green bank, the legal values being:

Ø: load no data into the green.
1: load red data into the green.
2: load green data into the green.
3: load blue data into the green.

Also **bluerte** is a 2 bit value indicating which data will be written into the blue bank, the legal values being:

Ø: load no data into the blue.
1: load red data into the blue.
2: load green data into the blue.
3: load blue data into the blue.

The WRMASK command may be used in conjunction with the PMCTL command to control writing into the image planes. The first four parameters (now Øs) are reserved for future use.

(Continued)

*Option Card users only.

Continued from previous page

PMCTL Ø Ø Ø Ø redrte, greenrte, bluerte

PARAMETERS

redrte, greenrte, bluerte

HOST BINARY COMMAND STREAM

[BF_H][Ø][Ø][Ø][Ø][redrte][greenrte][bluerte] (8 bytes)

FORTRAN SUBROUTINE CALL

CALL PMCTL (Ø,Ø,Ø,Ø,IREDRT, IGRNRT, IBLURT)

EXAMPLE

!PMCTL Ø Ø Ø Ø 1 2 3	Data is moved unmodified (default)
!PMCTL Ø Ø Ø Ø 2 2 2	Green data is moved into all three banks.

SYNTAX

POINT

FUNCTION

The POINT command sets the current point (CREG Ø) to the current pixel value (VREG Ø). The current point and the current pixel value remain unchanged.

HOST BINARY COMMAND STREAM

[88_H] (1 byte)

$88_H=210_8=136_{10}$

FORTRAN SUBROUTINE CALL

CALL POINT

EXAMPLE

!VALUE 255 0 255	Change current pixel value to 255,0,255
!MOVABS 100 100	Move current point to location 100,100
!POINT	Set pixel at location 100,100 to 255,0,255
!MOVREL 1 0	Move current point by 1,0 to 101,100
!POINT	Set pixel at location 101,100 to 255,0,255
!VALUE 0 0 255	Change current pixel value to 0,0,255
!MOVREL 1 1	Move current point by 1,1 to 102,101
!POINT	Set pixel at 102,101 to 0,0,255

SYNTAX

POKE addr,data

FUNCTION

The POKE command writes a given word of **data** into a given address, **addr,** in central processor memory. The POKE command is dangerous and should be used with care. POKING around carelessly can crash the central processor. The POKE command can be used in conjunction with the PEEK command to assist in debugging downloaded object code. POKEs into PROM memory are harmless and futile.

HOST BINARY COMMAND STREAM

[BE_H][highaddr][lowaddr][highdata][lowdata] (5 bytes)
$BE_H=276_8=190_{10}$

FORTRAN SUBROUTINE CALL

CALL POKE (IADDR,IDATA)

EXAMPLE

!PEEK #8050	Display contents of location 8050_H
FF00	
!POKE #8050 #0000	Change location 8050_H to 0
!PEEK #8050	Display contents of location 8050_H
0000	

RELATED COMMANDS

DNLOAD
MAP
PEEK
CONFIG

SYNTAX

POLYGN npoly, nvertl,xl,yl...

FUNCTION

The POLYGN command draws polygons in image memory in the current pixel value (VREG Ø). The number of polygons is given by **npoly**. Polygons are specified by giving the number of vertices followed by the list of vertices. The number of vertices is specified by **nvertl** for each polygon. Each vertex is specified by four bytes (two for x and two for y). The X,Y pair describing each vertex is taken to be a displacement from the current point (CREG Ø); polygons are relative to the current point.

PARAMETERS

npoly	the number of polygons to draw; range is 0 to 255.
nvertl	the number of vertices for polygon 1; range is 0 to 32,767.
xl,yl	first vertex; coordinates may range from -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[12_H][npoly][hinvertl][lownvertl][highXl][lowXl]
[highYl][lowYl]. . .
$12_H=022_8=18_{10}$

FORTRAN SUBROUTINE CALL

CALL POLYGN (NPOLY,NVERT,NVERTS)

NPOLY is an integer specifying the number of polygons to be drawn.

NVERT is an integer array containing the number of vertices in each of the NPOLY polygons.

IVERTS is an integer array of vertices. The first element of the array contains the x-component of the first vertex. The second element contains the y-component. The total number of vertices equals the sum of NPOLY elements of the NVERT array, each vertex requiring 2 elements of the IVERTS array.

(Continued)

EXAMPLES

!MOVABS 0 0
    Move current point to 0,0

!POLYGN 1 3 20 20 30 30 30 20
    Draw 1 polygon with 3 vertices located at
    (20,20), (30,30), and (30,20)

!POLYGN 1 4 10 10 50 70 -20 65 30 -10
    Draw 1 polygon with 4 vertices located at
    (10,10), (50,70), (-20,65), and (30,-10)

!PRMFIL ON
    Select filled graphics primitives

!POLYGN 1 3 -50 100 50 75 30 65
    Draw 1 polygon with 3 vertices located at
    (-50,100), (50,75), and (30,65)

!POLYGN 2 3 -50,100 50,75 30,65 3 20,20 30,30 30,20
    Draw 2 polygons, each with 3 vertices

SYNTAX

PRMFIL flag

FUNCTION

The PRMFIL command changes the Primitive Fill flag to indicate the desired Fill. If **flag=1**, filled graphics primitives will be drawn when a graphics primitive command is executed. If **flag=Ø**, the perimeter of the graphics primitive will be drawn. The graphics primitive commands affected by this flag are CIRCLE, CIRCXY, CIRCI, ARC, RECTAN, RECTI, RECREL, and POLYGN.

PARAMETERS

flag   fill flag: flag=0, disable filling; flag=1, enable filling

HOST BINARY COMMAND STREAM

$[1F_H][flag]$ (2 bytes)
$1F_H=037_8=31_{10}$

FORTRAN SUBROUTINE CALL

CALL PRMFIL (IFLAG)

EXAMPLE

!PRMFIL ON	Select filled primitives
!MOVABS 50 50	Move current point to 50,50
!CIRCLE 20	Draw filled circle of radius 20
!MOVABS 100, 100	Move current point to 100,100
!PRMFIL OFF	Select unfilled primitives
!CIRCLE 20	Draw circle of radius 20
!MOVABS 100 50	Move current point to 100,50
!RECTAN 110 60	Draw rectangle from current point to 110,60
!PRMFIL ON	Select filled primitives
!MOVABS 50 100	Move current point to 50,100
!RECTAN 60 110	Draw filled rectangle from current point to 60,110

SYNTAX

QUIT

FUNCTION

The QUIT command exits GRAPHICS mode and returns to ALPHA mode. This command should be used to return to ALPHA mode when the host or local alphanumeric terminal is finished issuing graphics commands.

HOST BINARY COMMAND STREAM

[FF_H] (1 byte)
$FF_H=377_8=255_{10}$

FORTRAN SUBROUTINE CALL

CALL QUIT

SYNTAX

RDMASK mask

FUNCTION

The RDMASK command sets the Read Mask. The Read Mask is an 8-bit mask which is ANDed with the output of the red, green, and blue banks of image memory before the values are fed into the Look-Up-Tables. The same 8-bit **mask** is used on all three (red, green, blue) image banks. If a bit in the **mask** is Ø, the corresponding bit plane in the image memory is forced low. The read masks are used in conjunction with write masks to allow multiple images to be stored in image memory and selected for display without changing Look-Up-Table entries.

The Read Mask should always be set to #FF (255) before the Look-Up-Tables are loaded; any other value may interfere with loading of the LUTs.

PARAMETERS

mask the 8-bit read mask; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$9E_H$][mask] (2 bytes)

$9E_H=236_8=158_{10}$

FORTRAN SUBROUTINE CALL

CALL RDMASK (MASK)

SYNTAX

RDPIXR vreg

FUNCTION

The RDPIXR command reads the pixel value from image memory at the current point (CREG Ø) and places the value into VREG number **vreg**.

PARAMETERS

vreg   value register; range is 0 to 15.

HOST BINARY COMMAND STREAM

$[AF_H][vreg]$ (2 bytes)
$AF_H=257_8=175_{10}$

FORTRAN SUBROUTINE CALL

CALL RDPIXR (IVREG)

EXAMPLE

!VALUE 255 100 105	Change current pixel value to 255,100,105
!POINT	Set current point to current value
!RDPIXR 13	Read current point and place value in VREG 13
!READVR 13	Display contents of VREG 13
255 100 105	

SYNTAX

READBU flag, cflg

FUNCTION

The READBU command sends to the port in GRAPHICS mode, the function button number of a button which was pushed. The READBU command removes one entry from the function button event queue. The function button event queue is eight events deep.

If the event queue is not empty, send the button number of the entry in the event queue. If the queue is empty and **flag**=1, wait for the next button to be pushed; if the queue is empty and **flag**=0, send a button number of zero.

If **cflg=1**, return the location of the digitizing tablet at the time the button was pushed. If **cflg=Ø**, send the location of the joystick or trackball.

The function button number and coordinate are sent in ASCII decimal. The format is FORTRAN I3,2I6 followed by a carriage return.

If the READBU command was sent from the host, the host must send an ACK (06_H or 86_H) to the Model One to resume normal command interpretation. The acknowledge character must be sent from the host as a single 7-bit control character, regardless of whether the host normally sends data to the Model One in 8-bit binary or ASCII hex.

PARAMETERS

flag event queue flag: flag=1, empty queue and wait for next button; flag=0, send next queue entry (or zero). See above for details.

cflg coordinate flag: flag=1, digitizing tablet location; cflg=0, joystick/trackball location.

HOST BINARY COMMAND STREAM

[$9A_H$][flag][cflg] (3 bytes)
$9A_H=232_8=154_{10}$

FORTRAN SUBROUTINE CALL

CALL READBU (IFLAG,ICFLAG,IBUTT,IX,IY)

IFLAG and ICFLAG are integers which specify whether or not to wait for a button and whether the coordinate returned should be from the digitizing tablet of the joystick/trackball.

IBUTT is an integer number returned from the subroutine call containing the number of the function button that was pushed.

IX,IY are integers returned from the call containing the location of the digitizing tablet or joystick/trackball when the button was pushed.

SYNTAX

READCR creg

FUNCTION

The READCR command sends the data in Coordinate Register **creg** to the port in GRAPHICS mode. The address is sent as two ASCII decimal numbers representing the x and y component of the address respectively. The numbers are sent in FORTRAN 2I6 format, followed by a carriage return.

If the command was issued by the host over the HOSTSIO interface, the Model One will wait for an acknowledge character (06_H or 86_H) from the host before command interpretation continues. The acknowledge character must be sent from the host as a single 7-bit control character, regardless of whether the host normally sends data to the Model One in 8-bit binary of ASCII hex.

PARAMETERS

creg coordinate register; range is 0 to 63.

HOST BINARY COMMAND STREAM

[98_H][creg] (2 bytes)
$98_H=230_8=152_{10}$

FORTRAN SUBROUTINE CALL

CALL READCR (ICREG,IX,IY)

IX AND IY are changed by the call and are returned from the subroutine with the X and Y components of the address contained in the coordinate register specified by ICREG.

EXAMPLE

!CLOAD 23 110 200	Load CREG 23 with 110,200
!READCR 23	Read contents of CREG 23
110 200	(Response from Model One)

SYNTAX

READF func

FUNCTION

The READF command controls the format and meaning of the data sent by the Model One as a result of READW or READWE command. The parameter **func** specifies the format. Its range is Ø to 4, interpreted as follows:

Func	Data	Format
Ø*	Full 24 bit data	FORTRAN 3I3
1	Red channel only	FORTRAN I3
2	Green channel only	FORTRAN I3
3	Blue channel only	FORTRAN I3
4	1K mode packed r,g,b	FORTRAN I3

*Default setting after COLDSTART

PARAMETERS

func format function; range is Ø to 4.

HOST BINARY COMMAND STREAM

[27_H][func] (2 bytes)
$27_H = 047_8 = 39_{10}$

FORTRAN SUBROUTINE CALL

CALL READF (IFUNC)

SYNTAX

READP

FUNCTION

The READP command sends the pixel value of the current point (CREG Ø) to the port in graphics mode. The pixel value will be sent as three ASCII decimal numbers representing the red, green, and blue pixel values. The format of the data sent is FORTRAN 3I3 followed by a carriage return.

If the READP command was issued by the host, the Model One will wait for an ACK (06_H or 86_H) character from the host before continuing command interpretation. The acknowledge character must be sent from the host as a single 7-bit control character, regardless of whether the host normally sends data to the Model One in 8-bit binary or ASCII hex.

HOST BINARY COMMAND STREAM

[95_H] (1 byte)
95_H=225_8=149_{10}

FORTRAN SUBROUTINE CALL

CALL READP (IRED,IGRN,IBLU)

IRED, IGRN AND IBLU are integers, which are returned from the subroutine call, containing the red, green and blue values of the pixel found at the current point in image memory.

EXAMPLE

```
!VAL8 0      Set current pixel value to 0,0,0
!FLOOD       Flood displayed image memory
!READP       Read pixel value at current point
 000 000 000 (Response from Model One)
```

RELATED COMMANDS

READF

SYNTAX

READVR vreg

FUNCTION

The READVR command sends the pixel value in Value Register **vreg** to the port in GRAPHICS mode. The data is sent as three ASCII decimal numbers representing the red, green and blue components of the pixel value respectively. The numbers are sent in FORTRAN 3I3 format and are followed by a carriage return.

If the READVR command was issued by the host, the Model One will wait for an ACK (06_H or 86_H) character from the host before continuing command interpretation. The acknowledge character must be sent from the host as a single 7-bit control character, regardless of whether the host normally sends data to the Model One in 8-bit binary or ASCII hex.

PARAMETERS

vreg value register; range is 0 to 15.

HOST BINARY COMMAND STREAM

[99_H][vreg] (2 bytes)
$99_H=231_8=153_{10}$

FORTRAN SUBROUTINE CALL

CALL READVR (IVREG,IRED,IGRN,IBLU)

IRED,IGRN and IBLU are changed by the call and returned from the subroutine with the red, green and blue components of the value contained in the value register specified by IVREG.

EXAMPLE

!VLOAD 3 35 100 255	Load VREG 3 with 35,100,255
!READVR 3	Read contents of VREG 3
35 100 255	(Output from Model One)

SYNTAX

READW nrows,ncols,bf

FUNCTION

The READW command sends the values of the pixels (r,g,b) in a window which is **nrows** high and **ncols** wide to the port in GRAPHICS mode. The current point is used as the upper left corner of the window. The window is scanned left to right and top to bottom.

The pixels are sent to the host as ASCII decimal numbers, in format set by the READF command. The numbers sent from the Model One can range from Ø to 255.

The **bf** parameter (blocking factor) tells the Model One how many pixel values to send before inserting a carriage return into the output stream. If the end of the window is reached before the block is filled, the block is padded with zeroes and sent.

The Model One will then wait for an ACK (06_H or 86_H) character from the host before sending out another block of data. The acknowledge character must be sent from the host as a single 7-bit control character (ASCII 06_H or 86_H), regardless of whether the host normally sends data to the Model One in 8-bit binary of ASCII hex.

PARAMETERS

nrows, ncols,	number of rows and columns.
bf	blocking factor.

HOST BINARY COMMAND STREAM

[96_H][highnrows][lownrows]
[highncols][lowncols][bf] (6 bytes)
$96_H = 226_8 = 150_{10}$

FORTRAN SUBROUTINE CALL

CALL READW (NROWS, NCOLS,IRED,IGRN,IBLU)

NROWS and NCOLS are integers specifying the number of rows and columns in the image to be read.

IRED, IGRN and IBLU are integer arrays which contain the pixel values returned by the subroutine call. The array must be defined in the main program to a dimension at least as large as the number of pixels in the window which is to be read.

RELATED COMMANDS

READF

SYNTAX

READWE nrows,ncols,bf

FUNCTION

The READWE command sends the pixel values (r,g,b) in a window which is **nrows** high and **ncols** wide to the port in GRAPHICS mode. The data is sent in a run-length encoded format.

Each pixel value sent from the Model One has a one byte count parameter indicating the number of pixels in a row which are of this same pixel value.

The current point is used as the upper left-hand corner of the window. The window is scanned left to right and top to bottom. The pixels and "count" are sent to the host as ASCII decimal numbers.

The format of the data sent is set by the READF command. The numbers sent from the Model One range from Ø to 255.

The **bf** parameter (blocking factor) tells the Model One how many pixels and counts to send before inserting a carriage return into the output stream. If the end of the window is reached before the block is filled, the block is padded with zeroes and sent.

The Model One will then wait for an ACK (06_H or 86_H) character from the host before sending out another block of data. The acknowledge character must be sent from the host as a single 7-bit control character (ASCII 06_H or 86_H), independent of host configuration.

PARAMETERS

nrows,ncols	number of rows and columns.
bf	blocking factor.

HOST BINARY COMMAND STREAM

[97_H][highnrows][lownrows]
[highncols][lowncols][bf] (6 bytes)
$97_H=227_8=151_{10}$

FORTRAN SUBROUTINE CALL

CALL READWE (NROWS,NCOLS,IBF,IRED,IGRN,IBLU)

NROWS and NCOLS are integers specifying the number of rows and columns in the image to be read.

IRED, IGRN AND IBLU are integer arrays which contain the pixel values returned by the subroutine call. The array must be defined in the main program to a dimension at least as large as the number of pixels in the window which is to be read.

RELATED COMMANDS

READF

SYNTAX

RECREL dx,dy

FUNCTION

The RECREL command draws a rectangle in image memory with one corner at the current point (CREG Ø) and the diagonally opposite corner displaced from the current point by **dx,dy.** The rectangle is drawn in the current pixel value (VREG Ø). The current point is unchanged.

PARAMETERS

dx,dy the relative opposite corner; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[89_H][highdx][lowdx][highdy][lowdy] (5 bytes)
$89_H=211_8=137_{10}$

FORTRAN SUBROUTINE CALL

CALL RECREL (IDX,IDY)

EXAMPLE

!MOVABS 100 150	Move current point to 100,150
!RECREL 10 10	Draw rectangle with diagonally opposite corner displaced by 10,10 (at 110,160)
!RECREL -20 -30	Draw rectangle with diagonally opposite corner displaced by -20,-30 (at 80,120)

SYNTAX

RECTAN x,y

FUNCTION

The RECTAN command draws a rectangle in image memory with one corner at the current point (CREG Ø) and the diagonally opposite corner at the point specified by **x,y.**

PARAMETERS

x,y the opposite corner of the rectangle; range is from -32,768 to 32,767.

HOST BINARY COMMAND STREAM

$[8E_H]$[highx]lowx][highy][lowy] (5 bytes)
$8E_H=216_8=142_{10}$

FORTRAN SUBROUTINE CALL

CALL RECTAN (IX,IY)

EXAMPLE

!MOVABS 30 50	Move current point to location 30,50
!RECTAN 70 100	Draw rectangle whose corners are located at 30,50 30,100 70,100 70,50
!MOVABS -20 -10	Move current point to -20,-10
!RECTAN -25 15	Draw rectangle whose corners are located at -20,-10 -20,15 -25,15 -25,15

SYNTAX

RECTI creg

FUNCTION

The RECTI command draws a rectangle primitive with one corner at the current point (CREG Ø) and the diagonally opposite corner at the point specified by coordinate register **creg.**

PARAMETERS

creg   coordinate register; range is 0 to 63.

HOST BINARY COMMAND STREAM

[$8F_H$][creg]   (2 bytes)
$8F_H = 217_8 = 143_{10}$

FORTRAN SUBROUTINE CALL

CALL RECTI (ICREG)

EXAMPLE

!MOVABS -20 -100	Move current point to -20,-100
!CLOAD 17 50 70	Load 50,70 into CREG 17
!RECTI 17	Draw rectangle whose corners are 50,70 50,-100 -20,-100 -20,70
!CLOAD 18 40 60	Load 40,60 into CREG 18
!RECTI 18	Draw rectangle whose corners are 40,60 40,-100 -20,-100 -20,60

SYNTAX

REPLAY

FUNCTION

The REPLAY command sends a dump of the last 32 characters sent by the host over the HOSTSIO interface to the local alphanumeric display screen. The last character output is the last character that was sent by the host.

HOST BINARY COMMAND STREAM

[BC_H] (1 byte)
$BC_H=274_8=188_{10}$

FORTRAN SUBROUTINE CALL

CALL REPLAY

EXAMPLE

!REPLAY	Dump last 32 characters from host input queue
ØØ FF FØ FD EØ E2 20 40 30 3F E3 20 21 31 ØØ ØØ 33 53 E5 25 20 32 37 70 7F FF FF FF 30 3F 55 F5	(Response from Model One)

SYNTAX

RUNLEN   nrows,ncols,[r,g,b,cnt]. . .[r,g,b,cnt]

FUNCTION

The RUNLEN command is used to transmit a run-length encoded image to the Model One. The array of pixels is **nrows** high and **ncols** wide. The location of the upper left corner of the array is given by the current point (CREG Ø).

Pixels in image memory are filled left-to-right, top-to-bottom. Each pixel value is sent as a full, 24-bit quantity one byte each of red, green and blue.

Each pixel value is followed by a one byte count, **cnt,** which specifies the number of horizontally contiguous pixels which are to be set to the given **r,g,b** value. If **cnt=Ø**, one pixel is set, if **cnt=1**, two pixels are set; the range is up to **cnt=255**, where 256 pixels are set.

PARAMETERS

nrows, ncols	number of rows and columns.
r,g,b	24-bit pixel value.
cnt	number of horizontal pixels to be set to the r,g,b value; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$2A_H$][highnrows][lownrows][highncols]
[lowncols][r][g][b][cnt]... (5+4*number of runs)
$2A_H$=0528=42_{10}

FORTRAN SUBROUTINE CALL

CALL RUNLEN(NROWS,NCOLS,IRED,IGRN,IBLU)

Where **NROWS** and **NCOLS** are integers which specify the number of rows and columns in the rectangle to be filled.

IRED, IGRN and IBLU are integer arrays which contain the pixel values. Each element of the array has a single one-byte pixel value in its least significant eight bits. Each array must be dimensioned to at least NROWS*NCOLS elements. The RUNLEN subroutine encodes the data found in the arrays into run-length form automatically.

RELATED COMMANDS

READF

SYNTAX

RUNLN8   nrows,ncols, [val][cnt]. . .[val,cnt]

FUNCTION

The RUNLN8 command transmit an image to the Model One in a run-length encoded fashion. The array of data is **nrows** high and **ncols** wide. The location of the upper left corner of the array is given by the current point (CREG Ø).

Pixels in image memory are filled left-to-right, top-to-bottom. Each pixel value is sent as an 8-bit quantity as in the VAL8 command.

Each pixel value is followed by a one byte count parameter, **cnt**, which specifies the number of horizontally contiguous pixels which are to be set to the given value. If **cnt=Ø**, one pixel is set, if **cnt=1**, two pixels are set; the range is up to **cnt=255** where 256 pixels are set.

PARAMETERS

nrows,ncols	the number of rows and columns.
val	8-bit pixel value.
cnt	the number of horizontal pixels to be set to val; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$2B_H$][highnrows][lownrows][highncols]
[lowncols][val][cnt]..   (5+2*number of runs)
$2B_H=053_8=43_{10}$

FORTRAN SUBROUTINE CALL

CALL RUNLN8(NROWS,NCOLS,IPIXELS)

Where **NROWS** and **NCOLS** are integers which specify the number of rows and columns in the rectangle to be filled

IPIXELS is an integer array which has the pixel values. The first pixel is in the first element of **IPIXELS**, each subsequent pixel is stored in successive array elements. The **IPIXELS** array contains at least **NROWS*NCOLS** elements. The RUNLN8 subroutine encodes the pixel data into run-length form automatically.

RELATED COMMANDS

READF

S A V C F G

SYNTAX

SAVCFG

FUNCTION

The SAVCFG command saves the Model One port configurations defined with the SYSCFG command. SAVCFG saves all port configurations; any port configurations that were not changed by SYSCFG, however, are not changed by SAVCFG. In addition, if special characters have been changed with the SPCHAR command, the SAVCFG command also saves the new special characters.

Port configurations in the Model One are stored in NVRAM (non-volatile random-access memory). On COLDstart, the configurations are copied into the NVRAM from PROM (programmable read-only memory). The SYSCFG command modifies the NVRAM; the SAVCFG command copies the NVRAM into PROM. Thus, changed configurations can be modified by a COLDstart until they are copied into PROM by a SAVCFG command. To ensure that configurations are saved, the SAVCFG command must be executed.

The DFTCFG command can be used to restore all ports to the default configuration (see DFTCFG for details). All special characters are also restored to the default.

If a SYSCFG command inadvertantly renders all ports incommunicado, the DFTCFG command description will explain how to reset the system so that SYSCFG and SAVCFG can be used again.

The SAVCFG command can be executed only from the local alphanumeric terminal, and cannot be included in a macro.

EXAMPLES

SYSCFG SERIAL HOSTSIO RTS OFF CTS OFF XIN ON XOUT ON PARITY N

	configures serial port HOSTSIO (see the SYSCFG command for details)
SYSCFG ALPHA ALPHASIO	configures serial port ALPHASIO as the local alphanumeric port
SYSCFG HOST MODEMSIO MODE BINARY	configures serial port MODEMSIO as the host port
SYSCFG ERROR HOSTSIO	configures serial port HOSTSIO as the error port
SAVCFG	stores the configurations defined by the above commands; ports that were not changed are left unchanged

SYNTAX

SCRORG x,y

FUNCTION

The SCRORG command sets the Screen Origin Register (CREG 4) to the point specified by **x,y**. The screen origin specifies the coordinate in image memory that will be displayed at the center of the screen. This command is used to pan the displayed image. The range of **x** and **y** is -32,768 to +32,767.

PARAMETERS

x,y   screen origin coordinate; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

$[36_H]$[highx][lowx][highy][lowy]   (5 bytes)
$36_H=066_8=54_{10}$

FORTRAN SUBROUTINE CALL

CALL SCRORG (IX,IY)

EXAMPLE

!MOVABS 0 0	Move current point to 0,0
!CIRCLE 100	Draw circle of radius 100
!SCRORG 50 50	Set screen origin to 50,50
!SCRORG 60 50	Set screen origin to 60,50
!SCRORG 70 50	Set screen origin to 70,50
!SCRORG 255 0	Set screen origin to 255,0 (note wrap-around)
!SCRORG 0 0	Restore screen origin to 0,0

SYNTAX

SPCHAR char,flag,code

FUNCTION

The SPCHAR command may be used to redefine the special characters used by the Model One, thereby circumventing problems with certain host computers and operating systems. The parameter **char** specifies which of the special characters is to be defined. The eight special characters are defined as follows:

Char	Purpose	Default	ASCII Code
0	Enter GRAPHICS mode	04_H or 84_H	CTRL-D
1	Send BREAK to host	10_H or 90_H	CTRL-P
2	WARMstart Model One	$1B_H$ or $9B_H$	CTRL-[or ESC
3	Kill line	40_H or $B0_H$	@
4	Backspace	08_H or 88_H	CTRL-H
5	ACK (Acknowledge)	06_H or 86_H	CTRL-F
6	NACK (Negative Acknowledge)	15_H or 95_H	CTRL-U
7	Enter Debug	18_H or 98_H	CTRL-X
8	Restart communications	13_H or 93_H	CTRL-Q
9	Suspend communications	11_H or 91_H	CTRL-S

The parameter **flag** may be 1 or 0. If **flag=1** the third parameter, **code**, specifies the new ASCII code of the special character. If **flag=0**, the Model One no longer responds to the special character and **code** is ignored but must be present.

The COLDstart sequence restores the default settings of the special characters.

PARAMETERS

char   special character number.
flag   use flag; flag=1, redefine; flag=0, ignore.
code   hex code for character.

HOST BINARY COMMAND STREAM

[$B2_H$][char][flag][code]   (4 bytes)
$B2_H=262_8=178_{10}$

FORTRAN SUBROUTINE CALL

CALL SPCHAR (ICHAR,IFLAG,ICODE)

EXAMPLE

!SPCHAR Ø 1 #05	Change the ENTER-GRAPHICS mode control code to 05_H or 85_H (CTRL-E)
!SPCHAR Ø 1 #04	Restore default ENTER-GRAPHICS code
!SPCHAR 2 Ø Ø	Disable the WARMstart character

S Y S C F G

SYNTAX

SYSCFG SERIAL [port mnemonic] [RTS on/off] [CTS on/off] [STOP 1/2]
[BITS 7/8] [PARITY e/o/1/h/n] [BAUD rate] [CTRL on/off]
[XIN on/off] [XOUT on/off]

SYSCFG SERIAL TABLETSIO [GTCO old/new] [SUMMA]

SYSCFG IEEE [address] [NORMAL] [TALK] [LISTEN]

SYSCFG ALPHA [port mnemonic]

SYSCFG ERROR [port mnemonic]

SYSCFG HOST [port mnemonic] [ASCII] [BINARY]

FUNCTION

The SYSCFG command is used:

1. To configure the Model One's serial ports, using the SYSCFG SERIAL command and its parameters,

2. To configure the Model One's IEEE port, using the SYSCFG IEEE command and its options, and

3. To configure the Model One's HOST, ALPHAnumeric, and ERROR ports, using the SYSCFG HOST, SYSCFG ALPHA, and SYSCFG ERROR commands with the appropriate port mnemonic.

Once the ports have been configured, the new configurations must be saved with the SAVCFG command. Until the SAVCFG command is executed, the new configurations will be overwritten by a COLDstart; once they have been saved by SAVCFG, however, they will restored by a COLDstart.

If it is necessary to restore the port configurations to a known state, the DFTCFG command is used.

The configurations can be displayed using the DISCFG command.

The SYSCFG command can be executed only from the local alphanumeric terminal, and cannot be included in a macro.

The default configurations are listed below. In using the SYSCFG command, the port assignments given in this list should be followed. The mnemonic for the port (HOST, KEYBD, etc.) should be used in place of the port number; however, the port number can be used if desired.

Port mnemonic	RTS	CTS	Baud	Parity	XIN	XOUT	CTRL	STOP	BITS
0 MODEMSIO	off	off	1200	none	on	off	off	1	8
1 KEYBDSIO	off	off	1200	none	on	off	on	1	8
2 TABLETSIO	off	off	1200	none	on	off	off	2	8
3 GRINSIO	off	off	1200	none	off	off	off	2	7
4 HOSTSIO	off	off	9600	none	off	on	off	2	8
5 ALPHASIO	off	off	9600	none	on	off	on	2	8
6 IEEE	-	-	-	-	-	-	-	-	-

PARAMETERS

SYSCFG SERIAL

The SYSCFG SERIAL command has the following parameters:

PORT mnemonic supplies the mnemonic of the serial port that is to be configured. The port mnemonic (or number) must be given.

RTS specifies that Request-To-Send should be asserted only before transmit (on) or at all times (off).

CTS specifies whether the Clear-To-Send protocol should be observed: with CTS OFF, the Model One may transmit at any time; with CTS ON, the Model One must wait for a Clear-To-Send.

PARITY specifies the parity on input/output for the given serial port. Parity may be Even, Odd, High (parity bit always set), Low, or Not_used.

BAUD specifies the baud rate for the serial port. The baud rate may be: 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 4800, 9600, 19200, or 38400.

If an illegal baud rate is specified, the command will generate an error and then terminate.

XIN indicates whether XON/XOFF is to be accepted at input: XIN ON specifies that output will be enabled or disabled according to the XON/XOFF signals received by the port; XIN OFF indicates that XON/XOFF signals should be ignored. XIN does not apply to TABLETSIO.

XOUT indicates whether XON/XOFF should be sent when the port's queue is near full (XOUT ON) or simply not used (XOUT OFF). XOUT does not apply to TABLETSIO.

CTRL instructs the Model One whether it should accept control characters from the port (CTRL ON) or ignore them (CTRL OFF). Note that this includes [CTRL-S] and [CTRL-Q].

STOP specifies whether one or two stop bits should be used.

STOP specifies whether one or two stop bits should be used.

BITS tells whether seven or eight bits are sent per byte.

SYSCFG TABLETSIO [GTCO old/new] specifies the old (prior to 8/26/82) or new GTCO tablet setup for the TABLETSIO port; SYSCFG TABLETSIO SUMMA specifies the Summagraphics Bit-Pad settings.

SYSCFG IEEE [address] [NORMAL] [TALK] [LISTEN]

The SYSCFG IEEE command is used to configure the Model One's IEEE-488 port. The command uses a single hexadecimal number to specify the address for the IEEE 488 port. [address] gives the address; it may be between 0 and 31 for an existing device. -1 indicates that no IEEE port exists. (The default is -1: no IEEE port.)

[NORMAL], [TALK], and [LISTEN] are mutually exclusive options. In NORMAL mode, the IEEE 488 port operates with a controller; TALK and LISTEN are used for local talk and listen on a bus with no controller.

SYSCFG ALPHA [port mnemonic]
SYSCFG ERROR [port mnemonic]

These commands are used to specify which ports are to be used as the ALPHAnumeric and ERROR ports.

SYSCFG HOST [port mnemonic] [MODE [ASCII] [BINARY]]

This command configures the HOST port. The port mnemonic, if given, indicates a port other than HOSTSIO port. The MODEMSIO port can be designated as the host port by using this command, as can the IEEE port. The port may be configured to expect ASCII hexadecimal characters or 8-bit binary characters from the host. One or the other parameter must be given. For example, the command SYSCFG HOST MODE BINARY is valid; it does not change the default host port.

EXAMPLES

SYSCFG SERIAL HOSTSIO RTS OFF CTS OFF XIN ON XOUT ON PARITY N BAUD 9600

	configures serial port HOSTSIO for 9600 baud, to ignore RTS and CTS, to expect XON/XOFF protocol on both input and output, and to ignore parity bits
SYSCFG ALPHA ALPHASIO	configures serial port ALPHASIO as the local alphanumeric device
SYSCFG HOST MODEMSIO MODE BINARY	configures serial port MODEMSIO as the host port
SYSCFG ERROR HOSTSIO	configures port HOSTSIO as the error port (the port to which error messages are sent)
SAVCFG	saves the configurations for ports HOSTSIO and 0; the configurations for the other ports are left unchanged

COLD	the new configurations are used at COLDstart
DFTCFG	restores all Model One ports to a known default state

RELATED COMMANDS

SAVCFG
DFTCFG
DISCFG

SYNTAX

TEXT1 strlen, string

FUNCTION

The TEXT1 command draws horizontal text into image memory, using font 1. At a size of 16 (set by TEXTC command), text drawn with font 1 will look like 5x7 dot matrix characters. The text to be drawn is specified by **string.**

If the command is being entered in ASCII mode from the local alphanumeric terminal/keyboard, the **string** to be drawn is the set of ASCII characters remaining on the command line. If the command is not being sent in ASCII mode, then the first byte of **string** contains the number of characters in the string (**strlen**) followed by **strlen** bytes containing the ASCII characters to be drawn. The current point (CREG Ø) specifies the starting point for the text string and remains unchanged.

PARAMETERS

strlen	the length of the string; required if string is not sent in ASCII mode.
string	the text string.

HOST BINARY COMMAND STREAM

[9Ø$_H$][strlen]([char1][char2]...[charn])
$90_H=220_8=144_{10}$

FORTRAN SUBROUTINE CALL

CALL TEXT1 (STRLEN,STRING)

STRLEN is an integer specifying the number of characters that are to be drawn. STRING is an integer array with two characters packed per 16-bit word, as in FORTRAN A2 format.

EXAMPLE

!MOVABS 0 0	Move current point to 0,0
!TEXT1 ABCDEF 1 2 3	Draw text string "ABCDEF 1 2 3"
!MOVABS 0 20	Move current point to 0,20
!TEXTC 32 0	Change scale to 32, angle to 0°
!TEXT1 WXYZ	Draw text string "WXYZ"

SYNTAX

TEXT2 strlen, string

FUNCTION

The TEXT2 command draws horizontal text into image memory using font 2. Font 2 is a user-defined character set which is downloaded using the **TEXTDN** command.

At power-on or coldstart, each character in font 2 defaults to the same character in font 1. When font 2 is downloaded, each character replaces the power-on default definition.

The **TEXT2** command is issued in the same manner as **TEXT1**. The current point (CREG Ø) specifies the starting point for the text string and remains unchanged.

Strlen specifies the length of the string when text is not sent in ASCII mode.

PARAMETERS

strlen	the length of the string; required if string is not sent in ASCII mode.
string	the text string.

HOST BINARY COMMAND STREAM

[91_H][strlen]([char1][char2]...[charn])
$91_H=221_8=145_{10}$

FORTRAN SUBROUTINE CALL

CALL TEXT2 (STRLEN,STRING)

STRLEN is an integer specifying the number of characters that are to be drawn. STRING is an integer array with two characters packed per 16-bit word, as in FORTRAN A2 format.

EXAMPLE

!MOVABS -100 -100	Move to -100,-100
!TEXT2 This is font 2	Draw text string in font 2
!MOVABS -100 -50	Move up to -100,-50
!TEXT1 This is font 1	Draw string in font 1 to compare

SYNTAX

TEXTC size,ang

FUNCTION

The TEXTC command specifies the size and angle of text for subsequent TEXT commands. A **size** of 16 will cause the text to be drawn at normal scale. Doubling the size parameter will double the size of the text.

The angle parameter **ang** specifies the angle in degrees at which the text will be drawn. The angle is measured counter-clockwise from the Ø degree direction. An angle of Ø will cause the TEXT1 and TEXT2 commands to draw normally oriented text from left to right, and cause the VTEXT1 and VTEXT2 commands to draw normally oriented text from top to bottom.

PARAMETERS

size	text size; range is 0 to 255.
ang	text angle; range is -32,768 to 32,767.

HOST BINARY COMMAND STREAM

[92_H][size][highang][lowang] (4 bytes)
$92_H=222_8=146_{10}$

FORTRAN SUBROUTINE CALL

CALL TEXTC (ISIZE,IANG)

EXAMPLE

!MOVABS 0,0	Move current point to 0,0
!TEXTC 16 0	Set text size to 16, angle to 0°
!TEXT1 This is a test	Draw text string
!TEXTC 16,0	Set text size to 16, angle to 30°
!TEXT1 of angled text	Draw text string
!TEXTC 32 0	Set text size to 32, angle 0°
!MOVABS 0 50	Move current point to 0,50
!TEXT1 and scaled text	Draw text string

SYNTAX

TEXTDN char,veclst

FUNCTION

The TEXTDN command defines the vectors to be drawn into image memory for the character specified by **char**. Whenever this character is encountered when using the TEXT2 and VTEXT2 commands, the vectors specified by **veclst** will be drawn.

If the scale and orientation are normal (scale=16, angle=Ø), the characters will be drawn exactly as specified. The character is specified as a series of relative moves and draws. Each relative move or draw requires one word (two bytes) to specify. The first two bytes of **veclst** specify the number of points that will be output.

The format for each point is defined as follows:

Bit Ø - 6	:	7 bit two's complement delta Y. (-64<=Y<=63)
Bit 7	:	Don't care. (May be 1 or Ø)
Bit 8 - 14	:	7 bit two's complement delta X. (-64<=X<=63)
Bit 15	:	Set if a draw, cleared if a move.

The range of **char** is Ø to 127. The number of vectors per character is limited only by the available memory space.

PARAMETERS

char   character to be defined.
veclst   vector list for char.

HOST BINARY COMMAND STREAM

[26_H][char][highnumpts][lownumpts]
[highvl][lowvl]....[highvn][lowvn]
(4+2*numpts bytes)
$26_H=046_8=38_{10}$

(Continued)

FORTRAN SUBROUTINE CALL

CALL TEXTDN (ICHAR,NUMPTS,IPOINTS)
ICHAR gives the character number, from 1 to 255.

NUMPTS is an integer specifying the number of points in the user defined character; range is 0 to 32,767.

IPOINTS is an integer array containing the list of moves and draws required to draw the character specified by CHAR. It must be dimensioned to at least NUMPTS, and must be in the above format.

Bit 0 to 6	:	7-bit 2's complement delta Y.
Bit 7	:	1 or Ø.
Bit 8 to 14	:	7-bit 2's complement delta X.
Bit 15	:	1, draw; Ø, move.

SYNTAX

TEXTRE

FUNCTION

The TEXTRE command erases the definition of any user defined characters sent via the TEXTDN command and restores the default character font #2. The TEXTRE command frees all of the space used by previously defined characters.

HOST BINARY COMMAND STREAM

[$B1_H$] (1 byte)
$B1_H=261_8=177_{10}$

FORTRAN SUBROUTINE CALL

CALL TEXTRE

SYNTAX

VADD vsum,vreg

FUNCTION

The VADD command adds the contents of value register **vreg** to the contents of value register **vsum** and places the result into value register **vsum**.

PARAMETERS

vsum,vreg coordinate registers; range is 0 to 15.

HOST BINARY COMMAND STREAM

[$A6_H$][vsum][vreg] (3 bytes)
$A6_H=246_8=166_{10}$

FORTRAN SUBROUTINE CALL

CALL VADD (IVSUM,IVREG)

EXAMPLE

!VLOAD 10 37 103 200	Load VREG 10 with 37,103,200
!VLOAD 11 100 100 50	Load VREG 11 with 100,100,50
!VADD 10 11	Add VREG 10 and VREG 11 place result in VREG 10
!READVR 10	Read contents of VREG
137 203 250	(Response from Model One)

SYNTAX

VAL1K rgbval

FUNCTION

The VAL1K command changes the current pixel value (VREG Ø) to the value **rgbval**. The two least significant two bits of **rgbval** (bits 1 and Ø) specify the two bits which are used in the blue channel of image memory. Bits 3 and 2 of **rgbval** specify the two bits of green; bits 5 and 4 specify the two bits of red. The VAL1K command reduces the number of bytes which must pass between the host and the Model One to change the current pixel value in 1K mode. The two most significant bits (bits 7 and 6) of **rgbval** should be zeros.

PARAMETERS

rgbval the 1K mode value; only the 6 least-significant bits are used.

HOST BINARY COMMAND STREAM

[$BØ_H$][rgbval] (2 bytes)
$BO_H=260_8=176_{10}$

FORTRAN SUBROUTINE CALL

CALL VAL1K (IRGBV)

EXAMPLE

!MODDIS 1	Put the model One in 1K addressing mode
!VAL1K #3F	Set the current pixel value to 192,192,192
!CIRCLE 50	Draw circle of radius 50
!VAL1K #3	Set current pixel value to 0,0,192
!CIRCLE 40	Draw circle of radius 40
!VAL1K #0	Set current pixel value to 0,0,0
!FLOOD	Flood image memory to the current pixel value

SYNTAX

VAL8 val

FUNCTION

The VAL8 command changes the current pixel value (VREG Ø) to the value **val**. Each of the three bytes of VREG Ø (red, green, and blue) are set to the same value. All operations which write into image memory use this value. This command can be used whenever the red, green, and blue components of the current pixel value are to be set to the same value. This command is particularly useful in systems with less than 24 planes of image memory.

PARAMETERS

val   the red, green, and blue pixel value to be used; range is 0 to 255.

HOST BINARY COMMAND STREAM

[86_H][val]   (2 bytes)
$86_H = 206_8 = 134_{10}$

FORTRAN SUBROUTINE CALL

CALL VAL8 (IVAL)

EXAMPLE

!VAL8 255	Set current pixel value to red=255, green=255, blue=255
!MOVABS 10,10	Move current point to 10,10
!DRWABS 25, 35	Draw vector in current pixel value to 25,35
!VAL8 100	Set current pixel value to red=100, green=100, blue=100
!DRWABS 40, 60	Draw vector in current pixel value to 40,60
!VAL8 100	Set current pixel value to red=50 green=50, blue=50
!CIRCLE 75	Draw circle of radius 75

SYNTAX

VALUE red,grn,blu

FUNCTION

The VALUE command changes the current pixel value (VREG Ø) to the value specified by **red,grn,blu.** All operations which write into image memory use VREG Ø, the current pixel value. The VALUE command specifies a full 3 bytes (24 bits) of pixel value data.

PARAMETERS

red	the red component of the current pixel value; range is 0 to 255.
grn	the green component of the current pixel value; range is 0 to 255.
blu	the blue component of the current pixel value; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$Ø6_H$][red][grn][blu] (4 bytes)
$06_H=006_8=6_{10}$

FORTRAN SUBROUTINE CALL

CALL VALUE (IRED,IGRN,IBLU)

EXAMPLE

!VALUE 0 255 0	Set current pixel value to red=Ø, green=255, blue=Ø
!MOVABS -10 25	Move to -10,25
!DRWABS 50 -30	Draw line from current point to 50, -30 in current pixel value
!VALUE 255 100 50	Set current pixel value to red=255, green=100, blue=50
!MOVABS 50 100	Move current point to 50,100
!CIRCLE 50	Draw circle of radius 50 at current point

SYNTAX

VECPAT mask

FUNCTION

The VECPAT command sets the vector generator pattern register to **mask.** The vector generator incorporates a pattern mask which allows patterned lines, such as dotted or dashed, to be generated.

This mask is loaded with a 16 bit value specified by **mask.** Every time a pixel is generated by the vector generator, the mask is rotated by one bit. If the least significant bit of the mask is set to 1, the pixel will be written into image memory. If the bit is reset to Ø, the pixel will be skipped.

The VECPAT command also affects the filling of graphics primitives when PRMFIL is on, as well as the fill pattern used when the CLEAR command is executed.

PARAMETERS

mask pattern register mask; range is 0 to 65,535.

HOST BINARY COMMAND STREAM

[$2E_H$][highmask][lowmask] (3 bytes)
$2E_H=056_8=46_{10}$

FORTRAN SUBROUTINE CALL

CALL VECPAT (MASK)

EXAMPLE

!VECPAT #FOFO	Set vector pattern mask to $FOFO_H$ (# indicates hex literal)
!MOVABS 0 0	Move current point 0,0
!DRWABS 100,200	Draw vector to 100,200
!VECPAT #AAAA	Set vector pattern mask to $AAAA_H$ (alternating 1's and 0's)
!DRWABS 100 -200	Draw vector from 100,200 to 100,-200
!VECPAT #00FF	Set vector pattern mask to $00FF_H$
!DRWABS -100 -200	Draw vector from 10,-200 to -100,-200
!VECPAT #FFFF	Restore vector pattern mask to $FFFF_H$

RELATED COMMANDS

all graphics primitives commands (RECTAN,CIRCLE,etc.)
all line drawing commands
CLEAR
PRMFIL

SYNTAX

VGWAIT frames

FUNCTION

The VGWAIT command inhibits transfer of vectors from the vector queue for a number of frame times, as specified by **frames.** This command may be used to synchronize vector writing with vertical blanking. For example, if a Ø is specified for the **frames** parameter, vector writing will be inhibited until the next vertical blanking interval. The delay in seconds will be **frames** divided by sixty.

PARAMETERS

frames the number of frames to wait; range is 0 to 65,535.

HOST BINARY COMMAND STREAM

[$3Ø_H$][highframes][lowframes] (3 bytes)
30_H=060_8=48_{10}

FORTRAN SUBROUTINE CALL

CALL VGWAIT (IFRAMES)

SYNTAX

VLOAD vreg,red,grn,blu

FUNCTION

The VLOAD command loads value register **vreg** with the pixel value specified by **red, grn, blu.**

PARAMETERS

vreg	value register; range is 0 to 15.
red	the red component of the current pixel value; range is 0 to 255.
grn	the green component of the current pixel value; range is 0 to 255.
blu	the blue component of the current pixel value; range is 0 to 255.

HOST BINARY COMMAND STREAM

[$A4_H$][vreg][red][grn][blu] (5 bytes)
$A4_H=244_8=164_{10}$

FORTRAN SUBROUTINE CALL

CALL VLOAD (IVREG,IRED,IGRN,IBLU)

EXAMPLE

!VLOAD 13 50 25 100	Load VREG 13 with 50,25,100
!READVR 13	Read contents of VREG 13
50 25 100	(response from Model One)

SYNTAX

VMOVE vdst,vsrc

FUNCTION

The VMOVE command copies the value in value register **vsrc** to value register **vdst.**

PARAMETERS

vdst,vsrc   value registers; range is 0 to 15.

HOST BINARY COMMAND STREAM

$[A5_H][vdst][vsrc]$ (3 bytes)
$A5_H=245_8=165_{10}$

FORTRAN SUBROUTINE CALL

CALL VMOVE (IVDST,IVSRC)

EXAMPLE

!VLOAD 10 25 25 50	Load VREG 10 with 25 25 50
!VMOVE 11 10	Move contents of VREG 10 into VREG 11

```
!READVR 11
 25 25 50
```

SYNTAX

VSUB vdif,vreg

FUNCTION

The VSUB command subtracts the contents of value register **vreg** from the contents of value register **vdif** and places the result into the value register specified by **vdif.**

PARAMETERS

vdif, vreg   value registers; range is 0 to 15.

HOST BINARY COMMAND STREAM

$[A7_H][vdif][vreg]$ (3 bytes)
$A7_H = 247_8 = 167_{10}$

FORTRAN SUBROUTINE CALL

CALL VSUB (IVDIF,IVREG)

EXAMPLE

!VLOAD 10 25 10 50	Load VREG 10 with 25,10,50
!VLOAD 11 100 200 140	Load VREG 11 with 100,200,140
!VSUB 11 10	Subtract VREG 10 from VREG 11 place result in VREG 11
!READVR 11	Read contents of VREG 11
75 190 90	

SYNTAX

VTEXT1 strlen, string

FUNCTION

The VTEXT1 command draws vertical text into image memory using font 1. At a size of 16 (set by TEXTC command), text drawn with font 1 will look like 5x7 dot matrix characters. The text to be drawn is specified by **string.**

If the command is being entered in ASCII mode from the local alphanumeric terminal/keyboard, the **string** to be drawn is the set of ASCII characters remaining on the command line.

If the command is not being sent in ASCII mode, then the first byte of **string** contains the number of characters in the string (**strlen**) followed by **strlen** bytes containing the ASCII characters to be drawn. The current point (CREG Ø) specifies the starting point for the text string and remains unchanged.

PARAMETERS

strlen	the length of the string; required if string is not sent in ASCII mode.
string	the text string.

HOST BINARY COMMAND STREAM

[93_H][strlen]([char1][char2]...[charn])
$93_H=223_8=147_{10}$

FORTRAN SUBROUTINE CALL

CALL VTEXT1 (STRLEN,STRING)

STRLEN is an integer specifying the number of characters that are to be drawn. STRING is an integer array with two characters packed per 16-bit word in FORTRAN A2 format.

EXAMPLE

!MOVABS 0,0	Move current point to 0,0
!VTEXT1 ABCDEF 1 2 3	Draw vertical text string
!MOVABS 0 20	Move current point to 0,20
!TEXTC 32 0	Change scale to 32, angle to 0°
!VTEXT1 WXYZ	Draw vertical text string

RELATED COMMANDS

TEXTC

SYNTAX

VTEXT2 strlen, string

FUNCTION

The VTEXT2 command draws vertical text into image memory using font 2. Font 2 is a user-defined character set which is downloaded using the **TEXTDN** command.

At power-on or COLDstart, each character in font 2 defaults to the same character in font 1. When font 2 is downloaded, each character replaces the power-on default definition. The **VTEXT2** command is issued in the same manner as **TEXT2**. The current point (CREG Ø) specifies the starting point for the text string and remains unchanged.

Strlen gives the number of characters in the string if the command is not given in ASCII mode.

PARAMETERS

strlen	the length of the string; required if string is not sent in ASCII mode.
string	the text string.

HOST BINARY COMMAND STREAM

$[94_H]$[strlen]([char1][char2]...[charn])
$94_H=224_8=148_{10}$

FORTRAN SUBROUTINE CALL

CALL VTEXT2 (STRLEN,STRING)

STRLEN is an integer specifying the number of characters that are to be drawn. STRING is an integer array with two characters packed per 16-bit word in FORTRAN A2 format.

EXAMPLE

!MOVABS -100 -100	Move current point to -100,-100
!VTEXT2 This is font 2	Draw vertical text string in font 2
!MOVABS -50 -100	Move current point up to -100,-50
!VTEXT1 This is font 1	Draw vertical text string in font 1

RELATED COMMANDS

TEXTC
VTEXT1

SYNTAX

WAIT frames

FUNCTION

The WAIT command waits for **frames** frame times (each frame time is a sixtieth of a second) before continuing command execution. This command is often used for choreographing graphic displays and synchronizing updates with vertical blanking. The delay in command execution in seconds will be **frames** divided by sixty.

If the **frames** parameter is zero, command execution will stop until the next vertical blanking interval.

PARAMETERS

frames   the number of frames to wait; range is 0 to 65,535.

HOST BINARY COMMAND STREAM

[$3D_H$][highframes][lowframes]   (3 bytes)

$3D_H=075_8=61_{10}$

FORTRAN SUBROUTINE CALL

CALL WAIT (IFRAMES)

SYNTAX

WARM

FUNCTION

The WARM command warm starts the Model One firmware. This clears the serial input and output queues, re-initializes the DIP switch programmable functions, and returns the system to ALPHA mode.

HOST BINARY COMMAND STREAM

[FE_H] (1 byte)
$FE_H=376_8=254_{10}$

FORTRAN SUBROUTINE CALL

CALL WARM

SYNTAX

WINDOW xl,yl,x2,y2

FUNCTION

The WINDOW command sets the current clipping (CREGS 9 and 10) window to the rectangle specified by **xl, yl, x2, y2**. The lower left corner of the window (CREG 9) is specified by **xl** and **yl**. The upper right corner of the window (CREG 10) is specified by **x2** and **y2**. The range of **xl, yl, x2** and **y2** is -32,768 to +32,767 with **xl<=x2** and **yl<=y2**. All vectors and graphics primitives are clipped to the current window.

PARAMETERS

xl,yl	lower left corner of clipping window.
x2,y2	upper right corner of clipping window.

HOST BINARY COMMAND STREAM

[$3A_H$][highxl][lowxl][highyl][lowyl]
[highx2][lowx2][highy2][lowy2]   (9 bytes)
$3A_H=072_8=58_{10}$

FORTRAN SUBROUTINE CALL

CALL WINDOW (IX1,IY1,IX2,IY2)

EXAMPLE

!WINDOW -20 -20 30 30	Set current window to rectangle [-20,-20][-20,30][30,30][30,-20]
!MOVABS 0 0	Move the current point to 0,0
!CIRCLE 25	Draw circle of radius 25
!TEXTC 32 0	Set text scale to 32, angle=0°
!TEXT1 CLIPPED TEXT	Draw text string
!CLEAR	Clear current window to current pixel value
!VAL8 0	Set current pixel value to Ø
!WINDOW -256 -256 255 255	Restore default window for 512 mode
!CLEAR	Clear current window

SYNTAX

WRMASK bitm,bankm

FUNCTION

The WRMASK command sets the write-enable masks. An 8-bit mask specified by **bitm** enables or disables write operations to specific bit planes of image memory. The same 8-bit mask is used for the red, green and blue image memory banks. If a bit in **bitm** is set, the corresponding bit plane in image memory will be write-enabled.

Bankm is used to enable or disable write operations into the red, green, blue image memory banks, and to the two overlay planes.

If bit Ø (the least significant bit) of **bankm** is set, the blue bank is write-enabled. If bit 1 is set, the green bank is write-enabled. Bit 2 of **bankm** write-enables the red bank if set. Similarly, bit 3 of **bankm** write-enables overlay plane 1 if set and bit 4 of **bankm** write-enables overlay plane Ø if set. As bits 3 and 4 are used to write-enable the overlay planes (on the Option Card), they will be ignored if the hardware does not include the Option Card. The range of **bankm** is Ø to 32. The range of **bitm** is Ø to 255.

Bankm is set as follows:

(MSBs)	bits 7,6,5	Currently unused: must be zeros
	bit 4	1: write-enable overlay plane Ø 0: do not write-enable overlay plane Ø
	bit 3	1: write-enable overlay plane 1 0: do not write-enable overlay plane 1
	bit 2	1: write-enable red bank 0: do not write-enable red bank
	bit 1	1: write-enable green bank 0: do not write-enable green bank
(LSBs)	bit 0	1: write-enable blue bank 0: do not write-enable blue bank

(Continued)

PARAMETERS

bitm   8-bit mask; range is 0 to 255.
bankm bank to write enable; range is 0 to 32.

HOST BINARY COMMAND STREAM

[$9D_H$][bitm][bankm]   (3 bytes)
$9D_H=235_8=157_{10}$

FORTRAN SUBROUTINE CALL

```
CALL WRMASK (IBITM,IBANKM)
```

SYNTAX

XHAIR num,flag

FUNCTION

The XHAIR command is used to enable the crosshairs. If the **flag=1**, crosshair **num** is enabled. If the **flag=Ø**, crosshair **num** is disabled.

The choices for **num** are 0, 1, 2 or 3. Crosshairs Ø and 1 are XOR'ed into the image memory and are always available. Crosshair 2 is drawn into overlay plane Ø, and crosshair 3 is drawn into overlay plane 1; crosshairs 2 and 3 are available only to users whose systems include the Option Card.

The color of crosshair Ø is taken from VREG 1. The color of crosshair 1 is taken from VREG 2. The position of crosshair Ø is taken from CREG 5, crosshair 1 from CREG 6, crosshair 2 from CREG 7, and crosshair 3 from CREG 8. Crosshairs 2 and 3 are always displayed in the color of the overlay plane into which they are drawn. Further, crosshairs 2 and 3 are drawn directly into the overlay planes and will damage data already drawn into the overlay planes.

The position of crosshairs is updated automatically every thirtieth of a second.

PARAMETERS

num   crosshair: range is 0 to 3.
flag enable/disable flag: flag=1, enable; flag=0, disable.

HOST BINARY COMMAND STREAM

[$9C_H$][num][flag]   (3 bytes)
$9C_H=234_8=156_{10}$

FORTRAN SUBROUTINE CALL

CALL XHAIR (NUM,IFLAG)

SYNTAX

ZOOM fact

FUNCTION

ZOOM sets the scale factor of the display screen to 1, 2, 4 or 8. In 512 mode, a scale factor of 1 has a 512 pixel-wide window visible. Scale factors of 2, 4 or 8 use pixel replication to display 256 X 256, 128 X 128 or 64 X 64 pixel windows of the 512 X 512 image memory.

In 1K mode, the entire 1K X 1K array is viewed at a scale factor of 1. This is done by performing hardware pixel averaging on 4 pixels in the 1K X 1k array to display each pixel on the screen. At a scale factor of 2, pixel averaging is no longer performed. Instead, a 512 X 512 window into the 1K X 1K array is displayed. At scale factors of 4 and 8, pixel replication is used to display 256 X 256 and 128 X 128 windows respectively.

When the scale factor is changed, the display is zoomed about the center of the screen.

PARAMETERS

fact   scale factor: values may be 1, 2, 4, or 8.

HOST BINARY COMMAND STREAM

[34_H][fact]
$34_H=064_8=52_{10}$

FORTRAN SUBROUTINE CAL (2 bytes)

CALL ZOOM (IFACT)

EXAMPLE

!MOVABS 0 0	Move current point to 0,0
!CIRCLE 30	Draw circle of radius 50
!ZOOM 2	Set scale factor to 2:1
!ZOOM 4	Set scale factor to 4:1
!ZOOM 8	Set scale factor to 8:1
!ZOOM 1	Restore scale factor of 1:1

RELATED COMMANDS

ZOOMIN
FLOOD

SYNTAX

ZOOMIN

FUNCTION

The ZOOMIN command sets the scale factor of the display screen to a factor of two greater than the current scale factor. The maximum scale factor is 8. If the current scale factor is 8, the ZOOMIN command sets the scale factor to 1.

HOST BINARY COMMAND STREAM

[35_H] (1 byte)
$35_H=065_8=53_{10}$

FORTRAN SUBROUTINE CALL

CALL ZOOMIN

EXAMPLE

!MOVABS 0 0	Set current point 0,0
!CIRCLE 30	Draw circle of radius 30
!ZOOMIN	Increase current scale factor by 2x (set to 2:1)
!ZOOMIN	Increase current scale factor by 2x (set to 4:1)
!ZOOMIN	Increase current scale factor by 2x (set to 8:1)
!ZOOMIN	Increase current scale factor by 2x (set to 1:1)

RELATED COMMAND

ZOOM
FLOOD

16.0 ALPHABETICAL COMMAND REFERENCE

This section lists, one command to a page, every Model One command. These subsections are included for each command:

SYNTAX: this section gives the command syntax.

FUNCTION: this section describes the function of the command.

PARAMETERS: this section lists the parameters for the command, gives their ranges, and supplies any other needed information on the command parameters.

HOST BINARY COMMAND STREAM: this section gives the host binary command stream and the hexadecimal, octal, and decimal opcodes for the command.

FORTRAN CALL: this section describes the FORTRAN call for the command, including the variable names.

EXAMPLE: this section gives an example of how the command is used.

RELATED COMMANDS: this section lists any related commands.

The following pages present the Model One commands in alphabetical order.

17.0 QUICK REFERENCE

This section provides you with a brief summary of each command, its parameters, and its hexadecimal opcode. The text is identical to that of the Model One Programming Reference Card.

MODEL ONE
PROGRAMMING REFERENCE CARD

The following is a summary of the graphics commands supported by the standard firmware in the Model One product family. Brackets [] indicate the hexadecimal opcode of each command.

HELP	List all command mnemonics.
HELP mnemonic	Give comand information.

Graphics Primitives

ARC rad,al,a2	Draw arc of radius rad. Starting angle is al; ending angle is a2. [11_H]
AREA1	Area fill. Boundary is any pixel different in value from the current point. The area is filled with current value. [13_H]
AREA2 vreg	Area fill. Boundary pixel value given in vreg. [14_H]
CIRCI creg	Draw circle. Location given by creg lies on the circumference. [10_H]
CIRCLE rad	Draw a circle of radius rad. [OE_H]
CIRCXY x,y	Draw circle. Point x,y lies on the circumference. [OF_H]
CLEAR	Flood current window to current pixel value. [87_H]
DRW2R dx,dy	Draw vector relative by dx,dy. [84_H]
DRW3R dx,dy	Draw vector relative by dx,dy. [83_H]
DRWABS x,y	Draw vector from current point to the point x,y. [81_H]
DRWI creg	Draw vector to location given by creg. [85_H]
DRWREL dx,dy	Draw vector relative by dx,dy. [82_H]
FILMSK rmsk,gmsk,bmsk	Image data is ANDed with masks before checking value in AREA fill commands. [$9F_H$]

FLOOD	Flood displayed image memory to current pixel value. [07]
MOV2R dx,dy	Move relative by dx,dy. [04_H]
MOV3R dx,dy	Move relative by dx,dy. [03_H]
MOVABS x,y	Move absolute location of current point to x,y. [01_H]
MOVI creg	Move to location given by coordinate register creg. [05_H]
MOVREL dx,dy	Move relative by dx,dy. [02_H]
POINT	Set current point to current pixel value. [88_H]
POLYGN npoly, verts	Draw polygons. Npoly gives number of polygons; for each polygon, verts gives number of vertices and the vertices. [12_H]
PRMFIL flag	Primitive fill. Filled primitives are drawn if flag=1. If flag=0, the perimeter of graphics primitives is drawn. [$1F_H$]
RECREL dx,dy	Draw rectangle. Diagonal corner is dx,dy away from current point. [89_H]
RECTAN x,y	Draw rectangle Diagonal corner is point x,y. [$8E_H$]
RECTI creg	Draw rectangle. Location given by creg is diagonal corner. [$8F_H$]
TEXT1 string	Draw text string with font 1. [90_H]
TEXT2 string	Draw text string with font 2. [91_H]
TEXTC size,ang	Specify size of text and draw at angle ang. [92_H]
TEXTDN char,veclst	Define downloaded character in font 2. [26_H]
TEXTRE	Restore default character set. [$B1_H$]
VAL1K val	Set current pixel value (for 1K mode). [$B0_H$]
VAL8 val	Set current pixel value to val,val,val. [86_H]

VALUE r,g,b	Set current pixel value to r,g,b. [06_H]
VTEXT1 string	Vertical text string with font 1. [93_H]
VTEXT2 string	Vertical text string with font 2. [94_H]

Look-Up Table Commands

LUT8 index,r,g,b	Make entry r,g,b at location given by index in Red, Green, and Blue LUTS. [$1C_H$]
LUTA index,entry	Make entry in all LUTs. Place entry at location given in index. [$1B_H$]
LUTB index,entry	Make entry in Blue LUT. Place entry at location given in index. [$1A_H$]
LUTG index,entry	Make entry in Green LUT. Place entry at location given in index. [19_H]
LUTR index,entry	Make entry in Red LUT. Place entry at location given in index. [18_H]
LUTRMP code,sind,eind, sent,ent	Load LUTs with ramp function. [$1D_H$]
LUTRTE+ func	Change LUT routing function specified by func. [$1E_H$]

+Model One/20 and Model One/25 Users Only.

Image Transmissions

PIXEL8 nrows,ncols,val	Pixel by pixel image definition. Pixel values are val,val,val. [29_H]
PIXELS nrows,ncols,r,g,b	Pixel by pixel image definition. Pixel values are r,g,b. [28_H]

RUNLEN nrows, ncols, r,b,b,cnt	Run-length encoded stream. Pixel value is r,g,b. Horizontal count is cnt. [$2A_H$]
RUNLN8 nrows,ncols,val, cnt	Run-length stream. Pixel value is val,val,val. Horizontal count is cnt. [$2B_H$]

<u>**Display Control**</u>

ASCII flag	Sets host port input as free format ASCII if flag=1, if flag=0 binary. [$9B_H$]
BLANK flag	Blanks screen when flag=1, normal video is restored when flag=0. [31_H]
COLD	Coldstart. Reset the Model One. [FD_H]
CORORG x,y	Loads coordinate origin register with x,y. [37_H]
FIRSTP flag	First pixel on vectors is inhibited when flag=1, uninhibited when flag=0. [$2F_H$]
MODDIS flag	Select display mode. 512 mode is selected if flag=0, 1K mode is selected if flag=1. [$2C_H$]
MODE1K func+	Select output data routing in 1K mode. [$2D_H$]
OVRRD*+ plane,flag	Display specified overlay plane when flag=0, inhibit display when flag=1. [BA_H]
OVRVAL*+ plane,flag	Set bits to 1 in specified overlay plane when flag=1, reset bits to 0 when flag=0. [$B9_H$]
OVRZM*+ plane,flag	Display plane at scale factor 1:1 if flag=0, display at same scale fact as image memory if flag=1. [$B8_H$]
PIXCLP flag	Pixel processor clipping status. Clip on over/underflow flag=1. [2]
PIXFUN mode	Set pixel processor mode. All vectors and DMA writes are affected. [2]
PIXMOV*	Initiate pixel mover transfer. Move window specified by CREG 11 and 12 as controlled by CREG 13 and 14. [BB_H]

PMCTL* Ø Ø Ø Ø redrte,greenrte,bluerte	Set pixel mover mode; redrte, greenrte, and bluerte control writing into the red, green, and blue banks. [BF_H]
QUIT	Exit graphics mode. [FF_H]
RDMASK mask	Set Read Mask. All pixel values read from image are ANDed with mask. [$9E_H$]
SCRORG x,y	Set screen origin register to x,y. [36_H]
SPCHAR string	Redefine special characters (ENTERGRAPHICS, etc.) [$B2_H$]
VECPAT mask	Vector generator pattern register is set to mask. [$2E_H$]
VGWAIT frames	Inhibit transfer of vectors from vector queue for frames frame times. [30_H]
WAIT frames	Wait for given number of frame times before continuing command execution. [$3D_H$]
WARM	Warmstart. Reinitialize Model One. [FE_H]
WINDOW x1,y1,x2,y2	Set current window. Defined by diagonal x1,y1 and x2,y2. [$3A_H$]
WRMASK bitm,bankm	Set Write-Enable Mask. Bit planes indicated by bitm and banks indicated by bankm are write-enabled. [$9D_H$]
XHAIR num,flag	Enable/Disable Crosshair number num. If flag=1 enable, if flag=0 disable. [$9C_H$]
ZOOM fact	Zoom by factor of fact=1,2,4 or 8. [34_H]
ZOOMIN	Zoom in by factor of 2. [35_H]

*Option Card Users Only.
+Model One/20 and Model One/25 Users Only.

Special Characters (Default Values)

CTL D	0	ENTERGRAPHICS
CTL P	1	Break

ESC	2	Warmstart
@	3	Line Kill
CTL H	4	Backspace
CTL F	5	ACK
CTL U	6	NACK
CTL X	7	Invoke Debug
CTL S	8	Suspend Communications
CTL Q	9	Resume Communications

FORTRAN Utility Subroutines

Call ENTGRA	Enter Graphics Mode
Call EMPTYB	Empty Buffers
Call SEND1(val)	Send one byte to output buffer.
Call SEND2(val)	Send two bytes (16 bits) to output buffer.

Readback Commands

All Read commands require a 7-bit ASCII ACK.

READBU flg,cflg	Read button number. If flag=1 wait for next button. If flag=0 send number of last button pushed. If cflg=1 send current digitizing tablet coordinate, if clfg=0 send current joystick/trackball coordinate. [$9A_H$]
READCR creg	Read coordinate register. Send x,y to port in graphics mode. [98_H]
READF func	Sets pixel readback format. Func specifies format. [27_H]
READP	Read Pixel. Send value of pixel to port in graphics mode. [95_H]
READVR vreg	Read value register. Send pixel value to port in graphics mode. [99_H]
READW nrows,ncols,bf	Read Window. Send values of pixels in window to port in graphics mode. [96_H]
READWE nrows,ncol	Read Window run-length encoded. Send values of pixels in window in run-length encloded form to port in graphics mode. [97_H]

Register Operations

CADD csum,creg	Place result of csum+creg in csum. [$A2_H$]
CLOAD creg,x,y	Load coordinate register creg with x,y. [$A\emptyset_H$]

CMOVE cdst,csrc	Move contents of csrc into cdst. [$A1_H$]
CSUB cdif,creg	Place result of cdif-creg in cdif. [$A3_H$]
VADD vsum,vreg	Place result of vsum+vreg into vsum. [$A6_H$]
VLOAD vreg,r,g,b	Load contents of value register vreg with r,g,b. [$A4_H$]
VMOVE vdst,csrc	Move contents of vsrc with vdst. [$A5_H$]
VSUB vdif,vreg	Place result of vdif-vreg into vdif. [$A7_H$]

Software Development

*	Program comment
ALPHAO strlen,string	Send text string to local alpha-numeric display. [$B4_H$]
CONFIG dwnlod,maclst, textfrst,inpque	Configure central processor RAM. [24_H]
DEBUG flag	Enter/Exit Command Stream Translator. Exit when flag=0, else enter. [$A8_H$]
DELAY factor	Delay transmission of characters. [$B6_H$]
DNLOAD	Download Z8002 object code. String format is Tektronix Hex. [FB_H]
HOSTO strlen,string	Send a text string to the host. [$B5_H$]
MAP	Display Z8002 memory map on local terminal. [FC_H]
NULL	No operation. [$ØØ_H$]
PEEK addr	Display contents of CPU memory. [BD_H]
POKE addr,data	Change contents of addr in CPU memory. [BE_H]
REPLAY	Dump last 32 characters of HOSTSIO input buffer to ALPHASIO port.

[BC_H]

Macro Programming

MACDEF num	Define Macro number num is terminated by MACEND command. [$8B_H$]
MACEND	End of Macro definition. [$\emptyset C_H$]
MACERA num	Erase Macro num. [$8C_H$]
MACRO num	Execute Macro num. [$\emptyset B_H$]

Interactive Device Support

BLINKC	Clear blink table. [23_H]
BLINKD lut,index	Disable Blink of specified lut,index. [21_H]
BLINKE lut,index entry1,entry2	Enable Blink of specified lut, index. Use entry 1 and entry 2 as alternate values. [$2\emptyset_H$]
BLINKR frames	Blink rate is frame times. [22_H]
BUTTBL index,nmac	Place Macro nmac in Button Table at location index. [AA_H]
BUTTON index	Execute Macro indicated by Button Table at location index. [AB_H]
FLUSH	Empty function button event queue. [15_H]
RDPIXR vreg	Places value of pixel at current point in specified value register vreg. [AF_H]

Coordinate Register Assignment

CREG 0	Current Point. Starting point of graphics primitives. Updated by a MOVE or DRAW command.
CREG 1	Joystick/Trackball Cursor Location. Current coordinate from the joystick or trackball. Updated automatically.
CREG 2	Digitizing Tablet Cursor Location. Current coordinate from the digitizing tablet. Updated automatically.
CREG 3	Coordinate Origin. Coordinate of the

	center of image memory.
CREG 4	Screen Origin. Coordinate of the pixel in the center of the screen.
CREG 5	Crosshair 0 location in Image Memory.
CREG 6	Crosshair 1 Location in Image Memory.
CREG 7*+	Crosshair 2 Location in Image Memory.
CREG 8*+	Crosshair 3 Location in Image Memory.
CREG 9	Clipping Window Origin. Lower left corner of current clipping window. All vectors are clipped to this window.
CREG 10	Clipping Window Origin. Upper right corner of current clipping window. All vectors are clipped to this window.
CREG 11,12*	Diagonal corners for PIXMOV command source window definition.
CREG 13*	Defines start corner for PIXMOV destination window.
CREG 14*	Controls direction of pixel writing for PIXMOV destination window.
CREG 15*+	Screen origin of overlay plane Ø.
CREG 16*+	Screen origin of overlay plane 1.
CREG 17-19	Reserved for future definition.
CREG 20-63	Unassigned. Available for temporary coordinate storage.

*Option Card Users Only.
+Model One/20 and Model One/25 Users Only.

Value Register Assignments

VREG 0 Current Value	The value used in all graphics primitives commands.
VREG 1	Value used for crosshair 0.
VREG 2	Value used for crosshair 1.
VREG 3	Fill Mask used for Area fills.

VREG 4*+	Color assignment for overlay plane Ø.
VREG 5*+	Color assignment for overlay plane 1.
VREG 6	For future definition.
VREG 7-15	Available for temporary value storage.
VREG 16**	Foreground color, alphanumeric window 0.
VREG 17**	Background color, alphanumeric window 0.
VREG 18**	Cursor color, alphanumeric window 0.
VREG 19**	Foreground color, alphanumeric window 1.
VREG 20**	Background color, alphanumeric window 1.
VREG 21**	Cursor color, alphanumeric window 1.
VREG 22**	Foreground color, alphanumeric window 2.
VREG 23**	Background color, alphanumeric window 2.
VREG 24**	Cursor color, alphanumeric window 2.
VREG 25**	Foreground color, alphanumeric window 3.
VREG 26**	Background color, alphanumeric window 3.
VREG 27**	Cursor color, alphanumeric window 3.

*Option Card Users Only.
+Model One/20 and Model One/25 Users Only.
**Advanced Graphics Development Firmware

System Configuration Commands

DFTCFG	Restore all ports to default configurations.
DISCFG	Display current system configurations.
SAVCFG	Save configuration set with SYSCFG.
SYSCFG HOST	[ASCII/BINARY]
SYSCFG IEEE	[address] [NORMAL] [TALK] [LISTEN]
SYSCFG SERIAL	[port-mnemonic] [RTS on/off] [CTS on/off] [STOP 1/2] [BITS 7/8] [PARITY e/o/1/h/n] [BAUD rate] [XIN on/off] [XOUT on/off]

Default Port Configurations

Port Mnemonic	RTS	CTS	Baud	Parity	XIN	XOUT	STOP	NBITS
MODEMSIO	off	off	1200	none	on	off	1	8
KEYBSIO	off	off	1200	none	on	off	1	8
TABLETSIO	off	off	1200	none	on	off	2	8
GRINSIO	off	off	1200	none	off	off	2	8
HOSTSIO	off	off	9600	none	off	on	2	8
ALPHASIO	off	off	9600	none	on	off	2	8

Alphanumeric Terminal Emulation**

ALPHEM** flag	Enables (flag=1 or ON) or disables (flag=0 or OFF) the alphanumeric terminal emulator. Routes text to selected window. [$C2_H$]
BOLD** flag	Enables (flag=1 or ON) or disables (flag=0 or OFF) drawing of bold text. [CC_H]
DEFWIN** window,x1,y1,x2,y2 bitm,bankm	Defines size and position of indicated window number. (x1,y1) defines first corner; (x2, y2) defines diagonal corner. bitm, bankm define write mask for window (see WRMASK command). [$C0_H$]
DIRCUR** x,y	Moves cursor to character position x,y within window. [$C4_H$]
GETCUR**	Returns Model One coordinates of cursor in currently-selected window. [$C9_H$]
GETPOS**	Returns character position of cursor in currently-selected window. [$C5_H$]
GETWIN**	Returns number of active window (-1 for no active window). [CE_H]
HOME**	Moves cursor to character position (0,0), the upper-left corner of the window. [CF_H]
MOVCUR** x,y	Moves cursor to Model One coordinate x,y within window limits. [$C8_H$]
OVRSTK** flag	Enables (flag=1 or ON) or disables (flag=0 or OFF) overstriking of text. [CD_H].
SCROLL** flag	Enables (flag=1 or ON) or disables (flag=0 or OFF) scrolling of text.
SELWIN** window	Select window as defined by DEFWIN. Sets routing for ALPHEM command. [$C1_H$]
SETCUR** flag	Enables (flag=1 or ON) or disables (flag=0 or OFF) cursor. [$C7_H$]

SETSIZ** xscale,yscale	Sets x,y scaling (multiples of 16 pixels). Default is (1,1).[$C6_H$]
WRAP** flag	Enables (flag=1 or ON) or disables (flag=0 or OFF) wraparound of text.[CB_H]

**Advanced Graphics Development Firmware

18.0 INDEX

APPENDIX I A MODEL ONE/25 1K MODE PROGRAM

```
C*
C EXAMPLE PROGRAM.
C This program has been developed to run on a Prlme 50-Series computer. Changes
C to the program will be necessary to adapt the program for other computers.
C Descriptions of the PRIMOS routines used are provided to aid in this process.
C
C
C  PRIMOS operating system routines used throughout this program:
C
C  Name       Function
C  ______     ______________________________________________________________
C
C  TNOU       Outputs a single character string of fixed length directly on
C             the users terminal.  This routine appends a new line character
C             at the end of the user's string.
C
C  TNOUA      Performs the same function as TNOU except that it does NOT append
C             a new line character at the end of the string.  Therefore, this
C             routine can be used to prompt a user and accept data off the same
C             line as the prompt text is on.
C
C  OPEN$A     Opens or closes a given file and file-unit
C             pair on the PRIMOS disk system. Note: this routine uses a physical
C             device unit number which is 12 units less than the actual FORTRAN
C             unit used by standard reads and writes. Also, the unit number
C             number is returned by the routine.
C
C  CLOS$A     This routine will close a given file by its' physical file unit
C             number.  No name is necessary.
C
C  RNAM$A     This routine will prompt the user with a given fixed length string
C             and accept as input another string typed in from the users
C             terminal.  In most cases, the string is assumed to be a PRIMOS
C             tree name reference to a given file in the file system.
```

```
C  Routines defined locally to this program:
C
C  Name       Function
C
C  ------     ------------------------------------------------------------
C
C  LOCVTX     This routine takes as input a pick point defined by IX,IY and
C             searches a list of vertices defined by IVERTS.  ICOUNT is used
C             to define the number of vertices in IVERTS.
C             Upon returning from this routine, 3 variables will be modified;
C             IX,IY will now reflect the actual point within the list that was
C             closest to the original IX,IY and WHERE will be an index to the
C             IVERTS array pointing to the x,y pair of that list closest to
C             the original IX,IY, (this x,y pair will have the same value as the
C             returned IX,IY). (There might be a bug in this routine for
C             version 7.)
C
C  SINGLE     This routine simply defines a set of macros to be run locally
C             on the Model One to rubber-band a single line.  This set of
C             Model One macros is invoked when the user goes into the "CREATE-
C             POLYGON" mode of this program.
C
C  DOUBLE     This routine is similar to SINGLE except that it defines a set
C             of macros to rubber-band two lines with a common vertex locally
C             on the Model One.  These macros are invoked by this program
C             when the user selects the "EDIT" sub-mode of the system.
C
C             (*** Both routines SINGLE and DOUBLE define macros in the Model
C             One that will run asynchronously from the host on the Model One. ***)
C
C  SCOLOR     This routine is a function which takes as input an x,y pair found
C             in a linear array and generates a new color which is a function
C             of the radius described by the pair.  Since the VAL1K ONELIB
C             routine is called after the color is selected, the Model One is
C             ready to generate graphics in the new color upon returning from
C              any call.
C
C  DISPMU     This routine takes as input a reference to an existing file and
C             reads the contents to generate new menu selection within the
C             defined menu area.  The HOME argument to this routine is not
C             used in this  program, but could be used to write part
C             of the new menu in a different color.  Can be used to identify
C             parts of the menu that remain static from menu to menu and make
C             those parts that change stand out.
C
C  FATTXT     This a very simple routine to generate fat text from the standard
C             Model One text font-1.  This assumes that the programmer has
C             already set the text size and angle before making the call.  The
C             text location is specified as part of the call along with the
C             text its self.
C
C  ISNAP      A function to take an arbitrary x or y value and return the
C             closest logical grid point on the screen.
C
C  NXTOBJ     This routine is used to traverse the complete list of objects,
```

```
C               (defined or undefined), and find the next DEFINED object in that
C               list.  The routine takes as input the list to be searched and
C               a start point within that list from which to begin the search.
C               It returns as its' value a pointer index to the next DEFINED
C               object. If the list is empty, it returns an undefined pointer
C               index and generates an error.
C
C  NXTMTY       This routine is much like NXTOBJ except that is searches the
C               list of object looking for the next UNDEFINED slot in the list.
C               Takes as input the list to be searched and returns as its' value
C               a pointer index to the next UNDEFINED object in the list.
C               If the list is full, it returns an undefined pointer and generates
C               an error message.
C
C  PICK         This function takes as input an arbitrary x,y pair defined by
C               IX,IY and searches the complete list of objects to find the
C               object with the vertex closest to the original pick point.  It
C               returns as its value the x,y pair that was closest, (IX,IY) and
C               sets WHERE to point to the object within the object list
C               that was closest.
C               This routine makes multiple calls to LOCVTX to find the closest
C               vertex/object.
C
C  BLDGRD       This routine simply defines a macro to be run locally on the
C               Model One every time the grid needs to be drawn or undrawn. This
C               macro will be invoked by the host program, (this program) and
C               will locally draw the grid in XOR mode.  The grid size corresponds
C               to the grid size defined in the SNAP function, (20x20 pixels).
```

```
C Model One MACROS used in conjunction with this program.
C
C  Macro-#   Function
C
C  -------   ------------------------------------------------------------
C
C        5   General purpose macro used to track the cross hairs.  When
C            working it will be associated with buttbl location 0 and there-
C            fore be updating the cross hair location every 30th of a second.
C
C    40-44   This set of macros runs locally on the Model One to control the
C            function of rubber-banding a single line.  For more details,
C            refer to the actual code found in the subroutine definition
C            of SINGLE.
C
C    50-57   This set of routines handles the rubber-banding of two lines
C            joined at a single vertex run locally on the Model One.  For
C            more details, see DOUBLE.
C
C      101   This macro is used to drag a given object locally on the Model
C            One.  It uses the same basic algorithm as SINGLE and DOUBLE
C            only it calls one of the object macros to draw th figure.
C
C      153   This macro holds all the graphics necessary to display the
C            "POLYGON-DRAG" sub-menu.  It is run locally when called by the
C            host program by a MACRO(153) command.
C
C  156-255   These macros are "object" macros.  They are defined by the host
C            each time a new object is created or edited.  All they contain
C            is a POLYGN command and the vertices necessary to define the
C            object.  These macros are used in order to reduce the amount of
C            information sent between the Host and the Model One every time
C            an object needs to be redrawn.
C            Only objects DEFINED in the host should have active macros in
C            this range of values.
```

```
C For a complete definition of the ONELIB routines used in this program, please
C refer to the Model One Programming Guide.
C

C--- SYSCOM keys are system defined constants and variables for system calls.
      INTEGER*2 MAXOBJ,MXOBJ2,MAXVRT
      PARAMETER (MAXOBJ=50,MXOBJ2=MAXOBJ*2,MAXVRT=200)
$INSERT SYSCOM>A$KEYS.INS.FTN

      INTEGER*2 INVERT(MAXOBJ)
      INTEGER*2 IVERTS(MAXVRT,MAXOBJ)
      INTEGER*2 ICOUNT(MAXOBJ)
      LOGICAL*2 EMPTY(MAXOBJ)
      INTEGER*2 ICOLOR(MAXOBJ)

      IN          INTEGER*2 NAME(40),NAMEL

      INTEGER*2 PICK,SCOLOR

      INTEGER*2 IBUTT,IX,IY,WHERE
      LOGICAL*2 GRIDON,OGRDON, DRAGNG
      INTEGER*2 NXTMTY,NXTOBJ
      LOGICAL*2 FIRST
      INTEGER*2 I,J,K,JUNK(2)
      INTEGER*2 IXDEL,IYDEL
      INTEGER*2 NSTART,NOBJX
      INTEGER*2 IUNIT,FIUNIT, OUNIT,FOUNIT

C* <*>modellconsts.ins.ftn> (Created: Thursday, April 1, 1982) */

      INTEGER*2 WHITE$, BLACK$

      INTEGER*2 CREG0, CREG1, CREG2, CREG3, CREG4, CREG5
      INTEGER*2 CREG6, CREG7, CREG8, CREG9, CREG10,CREG11
      INTEGER*2 CREG12,CREG13,CREG14,CREG15,CREG16,CREG17
      INTEGER*2 CREG18,CREG19,CREG20,CREG21,CREG22,CREG23
      INTEGER*2 CREG24,CREG25,CREG26,CREG27,CREG28,CREG29
      INTEGER*2 CREG30,CREG31,CREG32,CREG33,CREG34,CREG35
      INTEGER*2 CREG36,CREG37,CREG38,CREG39,CREG40,CREG41
      INTEGER*2 CREG42,CREG43,CREG44,CREG45,CREG46,CREG47
      INTEGER*2 CREG48,CREG49,CREG50,CREG51,CREG52,CREG53
      INTEGER*2 CREG54,CREG55,CREG56,CREG57,CREG58,CREG59
      INTEGER*2 CREG60,CREG61,CREG62,CREG63

      INTEGER*2 CURPNT,JOYSTK,DIGTZR,CORRG ,SCRRG ,XHAIR0
      INTEGER*2 XHAIR1,LWNORG,UWNORG,OVRPL0,OVRPL1


      INTEGER*2 VREG0, VREG1, VREG2, VREG3, VREG4, VREG5
      INTEGER*2 VREG6, VREG7, VREG8, VREG9, VREG10,VREG11
      INTEGER*2 VREG12,VREG13,VREG14,VREG15
```

```
INTEGER*2 CURVAL, XR0VAL, XR1VAL, FLMSK


PARAMETER (WHITE$=255, BLACK$=0)

PARAMETER (CREG0 = 0, CREG1 = 1, CREG2 = 2, CREG3 = 3)
PARAMETER (CREG4 = 4, CREG5 = 5, CREG6 = 6, CREG7 =-1)
PARAMETER (CREG8 =-1, CREG9 = 9, CREG10=10, CREG11=11)
PARAMETER (CREG12=12, CREG13=13, CREG14=14, CREG15=15)
PARAMETER (CREG16=16, CREG17=-1, CREG18=-1, CREG19=-1)
PARAMETER (CREG20=20, CREG21=21, CREG22=22, CREG23=23)
PARAMETER (CREG24=24, CREG25=25, CREG26=26, CREG27=27)
PARAMETER (CREG28=28, CREG29=29, CREG30=30, CREG31=31)
PARAMETER (CREG32=32, CREG33=33, CREG34=34, CREG35=35)
PARAMETER (CREG36=36, CREG37=37, CREG38=38, CREG39=39)
PARAMETER (CREG40=40, CREG41=41, CREG42=42, CREG43=43)
PARAMETER (CREG44=44, CREG45=45, CREG46=46, CREG47=47)
PARAMETER (CREG48=48, CREG49=49, CREG50=50, CREG51=51)
PARAMETER (CREG52=52, CREG53=53, CREG54=54, CREG55=55)
PARAMETER (CREG56=56, CREG57=57, CREG58=58, CREG59=59)
PARAMETER (CREG60=60, CREG61=61, CREG62=62, CREG63=63)

PARAMETER (CURPNT=CREG0,  JOYSTK=CREG1,  DIGTZR=CREG2)
PARAMETER (CORRG =CREG3,  SCRRG =CREG4,  XHAIR0=CREG5)
PARAMETER (XHAIR1=CREG6,  LWNORG=CREG9,  UWNORG=CREG10)
PARAMETER (OVRPL0=CREG15, OVRPL1=CREG16)

PARAMETER (VREG0 = 0, VREG1 = 1, VREG2 = 2, VREG3 = 3)
PARAMETER (VREG4 = 4, VREG5 = 5, VREG6 =-1, VREG7 =-1)
PARAMETER (VREG8 = 8, VREG9 = 9, VREG10=10, VREG11=11)
PARAMETER (VREG12=12, VREG13=13, VREG14=14, VREG15=15)

PARAMETER (CURVAL=VREG0,  XR0VAL=VREG1,  XR1VAL=VREG2)
PARAMETER (FLMSK =VREG3)


INTEGER*2 XLL1,YLL1, XUR1,YUR1      /*  SCREEN BOUNDIRES
PARAMETER (XLL1=-960,YLL1=-930, XUR1=958,YUR1=988)

INTEGER*2 XLLB,YLLB, XURB,YURB     /* WORK AREA BOUNDRIES
PARAMETER (XLLB=-478,YLLB=-220, XURB=464,YURB=380)

INTEGER*2 XLLH,YLLH, XURH,YURH     /* HOME MENU BOUNDRIES
PARAMETER (XLLH=-474,YLLH=-478, XURH=-280,YURH=-242)

INTEGER*2 UPXFUN,UBTTB0,UPRMFL,URTNP,GOHOME
PARAMETER (UPXFUN=250,UBTTB0=251,UPRMFL=252,URTNP=253,
+          GOHOME=215)
```

```
C--- Prelude. This section defines the screen format, clipping
C---          window and sets all counters, constants, etc. to
C---          their initial values.

      CALL TNOU('[EditPoly Rev 7.0]',18)

      DO 45 I = 1,MAXOBJ
        ICOUNT(I) = 0
        INVERT(I) = 0
        EMPTY(I)  = .TRUE.
45    CONTINUE

      CALL ENTGRA
      DO 46 I = 1,255
        CALL MACERA(I)
46    CONTINUE

      CALL BUTTBL(0,0)

      CALL MODDIS(1)        /* Mode 1k.  (1024X1024 addressing mode.)
      CALL PRMFIL(1)        /* Fill all polygons from here on.
      CALL FLUSH            /* Flush the Model One button queue.
      CALL SINGLE           /* Load local macros for rubberbanding a
                            /* single line into the Model One Z8002
                            /* memory.
      CALL DOUBLE           /* Load local macros for rubberbanding
                            /* two lines with a common single floating
                            /* vertex into the Z8002 memory.

C--- Write Raster Technologies name and program title into the Model
C--- Ones' frame buffer.

      CALL MACDEF(153)  /* Polygon Drag sub-menu
        CALL DISPMU('SHOWS>.POLY>DRAG.MENU',21,.TRUE.)
      CALL MACEND

      CALL VAL1K(52)
      CALL BUTTBL(1,43)
      CALL WINDOW(-512,-512, 511,511)
      CALL VAL1K(52)
      CALL TEXTC(64,0)
      CALL FATTXT('Raster',6,-450,434)
      CALL FATTXT('Technologies',12,-469,402)
      CALL MOVABS(XURB-340,422)
      CALL TEXT1(14,'Polygon Editor')
      CALL VAL8(WHITE$)
      CALL MOVABS(XLLB-1,YLLB-1)
      CALL PRMFIL(0)
      CALL RECTAN(XURB+1,YURB+1)
      CALL PRMFIL(1)
      CALL WINDOW(XLLB,YLLB, XURB,YURB)

      CALL MACDEF(5)        /* General purpose macro for tracking the cross hairs.
        CALL CMOVE(5,2)
```

```
      CALL MACEND

C--- Initialize constants.

      CALL PIXCLP(1)
      CALL VLOAD(VREG15,100,100,110)
      NOBJ = 0
      OBJNO = 1
      INVERT(OBJNO) = 1
      GRIDON = .TRUE.
      CALL BLDGRD
      IF (GRIDON) CALL MACRO(154)   /* Grid on
      CALL PIXFUN(0)
```

```
C       RETURN HERE FOR MAIN MENU
C
6000    CONTINUE
C
C       DISPLAY TOP MENU
C
        CALL DISPMU('SHOWS>.POLY>MAIN.MENU',21,.TRUE.)
C
6050    CONTINUE  /* GET NEXT BUTTON, MAIN MENU
        NAMEL = 80
        CALL QUIT
        WRITE(1,101)
101     FORMAT(1X,/,'Select a menu option using the cursor.',//)
        CALL ENTGRA
        CALL XHAIR(2,1)
        CALL BUTTBL(0,5)
        CALL READBU(1,1,IBUTT,IX,IY)    /*  WAIT FOR A BUTTON
C
        IF(IBUTT .EQ. 1) GO TO 6100  /*  RESTORE SOME EXISTING DATA
        IF(IBUTT .EQ. 2) GO TO 6200  /*  SAVE THE DATA FROM ON SCREEN
        IF(IBUTT .EQ. 3) GO TO 6300  /*  BEGIN AN EDIT SESSION
        IF(IBUTT .EQ. 4) GO TO 9999  /* HE WANTS TO QUIT OUT
        CALL ALPHAO(21,'***Invalid response.')
        GOTO 6050  /*  LOOP AND WAIT FOR A VALID BUTTON

6100    CONTINUE /* RESTORE OBJECTS.
        CALL QUIT
6150    CONTINUE
        IF (NAMEL .GT. 0) GOTO 6160
        CALL ENTGRA
        GOTO 6000

6160    CONTINUE
        NAMEL = 80
        CALL RNAM$A('Input file',11, A$FUPP,NAME,NAMEL)
        NAMEL = NLEN$A(NAME,NAMEL)
        IF (.NOT. OPEN$A(A$READ+A$GETU,NAME,NAMEL,IUNIT)) GOTO 6150
        FIUNIT = IUNIT + 12
        READ(FIUNIT,6112) NOBJX
        NSTART = NOBJ+1
        NOBJ   = NOBJ + NOBJX
        CALL ENTGRA
        IF (GRIDON) CALL MACRO(154)    /* Grid erase
        CALL PIXFUN(0)
        OBJNO = 0
        DO 6110 I = NSTART,NOBJ
          OBJNO = NXTMTY(EMPTY)
          EMPTY(OBJNO) = .FALSE.
          READ(FIUNIT,6113) ICOLOR(OBJNO),INVERT(OBJNO)
          K = INVERT(OBJNO)*2
          READ(FIUNIT,6114) (IVERTS(J,OBJNO),J=1,K)
          IVERTS(K+1,OBJNO) = IVERTS(1,OBJNO)
          IVERTS(K+2,OBJNO) = IVERTS(2,OBJNO)
```

```
        ICOUNT(OBJNO) = K+1
        CALL VAL1K(ICOLOR(OBJNO))
        CALL MACDEF(OBJNO+155)
          CALL POLYGN(1,INVERT(OBJNO),IVERTS(1,OBJNO))
        CALL MACEND
        CALL MACRO(OBJNO+155)
6110  CONTINUE
      IF (GRIDON) CALL MACRO(154)    /* Grid on
      CALL PIXFUN(0)
      CALL QUIT
      CALL CLOS$A(IUNIT)
      WRITE(1,6111) NOBJX,NAME
6111  FORMAT(1X,I3,' Objects read from: ',40A2,//)
      CALL ENTGRA
      GOTO 6050

6200  CONTINUE /* SAVE OBJECTS.
      CALL QUIT
6250  CONTINUE
      IF (NAMEL .GT. 0) GOTO 6260
      CALL ENTGRA
      GOTO 6000

6260  CONTINUE
      NAMEL = 80
      CALL RNAM$A('Output file',12, A$FUPP,NAME,NAMEL)
      NAMEL = NLEN$A(NAME,NAMEL)
      IF (.NOT. OPEN$A(A$WRIT+A$GETU,NAME,NAMEL,OUNIT)) GOTO 6250
      FOUNIT = OUNIT + 12
      WRITE(FOUNIT,6112) NOBJ
      OBJNO = 0
      DO 6210 I = 1,NOBJ
        OBJNO = NXTOBJ(EMPTY, OBJNO+1)
        WRITE(FOUNIT,6113) ICOLOR(OBJNO),INVERT(OBJNO)
        K = INVERT(OBJNO)*2
        WRITE(FOUNIT,6114) (IVERTS(J,OBJNO),J=1,K)
6210  CONTINUE
      CALL CLOS$A(OUNIT)
      WRITE(1,6211) NOBJ,NAME
6211  FORMAT(1X,I3,' Objects written to: ',40A2,//)
      CALL ENTGRA
      GOTO 6050

6300   CONTINUE /* EDIT OBJECTS.
C
C      DISPLAY MAIN MENU
C
      CALL DISPMU('SHOWS>.POLY>PE2.MENU',20,.TRUE.)
```

```
C--- Main Loop. This section of code interprets the main menu
C---            buttons and calls the appropriate subroutines to handle
C---            each button.

1000  CONTINUE
      FIRST=.TRUE.
      CALL XHAIR(0,1)
      CALL BUTTBL(0,5)
1005  CONTINUE
      CALL READBU(1,1,IBUTT,IX,IY)

C  1.  Define a polygon
C  2.  Edit a polygon
C  3.  Delete an existing polygon
C  4.  Read colors from the grid
C  5.  Drag an existing polygon
C  6.  Grid On/Off
C  7.  (Undefined)
C  8.  (Undefined)
C  9.  Return to previous menu for Restore, Save, and Quit functions
C 10.  (UNdefined)
C 11.  (Undefined)
C 12.  (Undefined)
C 13.  Confirm a polygon being created.
C 14.  (Undefined)
C 15.  (Undefined)

      GOTO (1010,2000,3000,4000,5000,7000,1005,1005,9000), IBUTT
      GOTO 1005

C--- Define Polygon. This section is invoked when button #1 is pressed
C---                 and the user is looking at the main menu.  A check
C---                 is made to see if the max number of objects has been
C---                 exceeded.

1010  CONTINUE
      CALL VAL8(WHITE$)
      IF (NOBJ .LT. MAXOBJ) GOTO 1011
      CALL ALPHAO(26,'Only 100 objects allowed.')
      GOTO 1000

1011  CONTINUE
      OBJNO = NXTMTY(EMPTY)
      ICOUNT(OBJNO) = 1

1012  CONTINUE
      IF(.NOT.FIRST) CALL READBU(1,1,IBUTT,IX,IY)
      IF(IBUTT .EQ. 9 ) GOTO 9999     /* EXIT PROGRAM
      IF(IBUTT .EQ. 13) GOTO 1020     /* CLOSE & FILL THE POLYGON.
      IF(IBUTT .NE. 1 ) GOTO 1012     /* ONLY RECOGNIZE BUTTON 1
      IF (ICOUNT(OBJNO) .LT. MAXVRT-1) GOTO 1014
      CALL ALPHAO(27,'Only 200 vertices allowed.')
      GOTO 1020
```

```
1014  CONTINUE
      IX = ISNAP(IX, GRIDON)
      IY = ISNAP(IY, GRIDON)
      IVERTS(ICOUNT(OBJNO),OBJNO)=IX
      IVERTS(ICOUNT(OBJNO)+1,OBJNO)=IY
      CALL CLOAD(CREG25,IX,IY)
      ICOUNT(OBJNO)=ICOUNT(OBJNO)+2

      IF (FIRST) GOTO 1234
      CALL MACRO(42)  /* NOT THE FIRST POINT, CONFIRM THE CURRENT LINE.
      GO TO 1012  /* KEEP GOING UNITL BUTTON 13 IS PUSHED

1234  CONTINUE   /* FIRST POINT OF THE POLYGON, DEFINE COLOR AND DRAW THE CIRCLE
      ICOLOR(OBJNO) = SCOLOR(IVERTS(1,OBJNO))
      CALL MOVABS(380,-300)
      CALL WINDOW(-512,-512,511,511)
      CALL CIRCLE(40)
      CALL CMOVE(CREG21,CREG25)
      CALL CMOVE(CREG22,CREG25)
      CALL MACRO(41)  /* START RUBBERBANDING
      FIRST = .FALSE.
      GO TO 1012

1020  CONTINUE  /* CONNECT LAST LINE TO THE START VERTEX; REDRAW
                /* THE POLYGON FILLED AND IN THE CORRECT COLOR.
      IF (ICOUNT(OBJNO) .GT. 3) GOTO 1025

C--- POLYGON HAD LESS THAN 3 LINES: ERASE IT AND RETURN TO THE MAIN MENU.

      CALL MOVABS(IVERTS(1,OBJNO),IVERTS(2,OBJNO))
      CALL POINT
      CALL DRWI(CREG21)
      CALL MACRO(44)
      ICOUNT(OBJNO) = 0
      IVERTS(1,OBJNO) = 0
      IVERTS(2,OBJNO) = 0
      EMPTY(OBJNO) = .TRUE.
      GOTO 1000

1025  CONTINUE    /* THIS IS A VALID POLYGON, DRAW IT INTO IMAGE MEMORY.
      CALL BUTTBL(0,0)
      CALL MOVI(CREG22)
      CALL DRWI(CREG21)
      IVERTS(ICOUNT(OBJNO),OBJNO)=IVERTS(1,OBJNO)
      IVERTS(ICOUNT(OBJNO)+1,OBJNO)=IVERTS(2,OBJNO)
      CALL MOVABS(IVERTS(1,OBJNO),IVERTS(2,OBJNO))
      J = ICOUNT(OBJNO) - 2
      DO 1030 I = 3,J,2
        CALL DRWABS(IVERTS(I,OBJNO),IVERTS(I+1,OBJNO))
1030  CONTINUE
      INVERT(OBJNO)=ICOUNT(OBJNO)/2
      CALL PIXFUN(0)    /* INSERT MODE
      CALL CMOVE(30,0)
      IF (GRIDON) CALL MACRO(154)   /* Grid erase
```

```
CALL PIXFUN(0)
CALL MOVABS(0,0)
CALL VAL1K(ICOLOR(OBJNO))
CALL MACDEF(OBJNO+155)
  CALL POLYGN(1,INVERT(OBJNO),IVERTS(1,OBJNO))
CALL MACEND
CALL MACRO(OBJNO+155)    /* DRAW THE POLYGON VIA THE MACRO
CALL MOVI(30)
NOBJ = NOBJ + 1
EMPTY(OBJNO) = .FALSE.   /* INDICATE THIS IS A VALID OBJECT
OBJNO = NXTMTY(EMPTY)    /* GET THE NEXT OBJECT IN THE LIST
CALL MACRO(44)           /* DISPLAY MAIN MENU
IF (GRIDON) CALL MACRO(154)    /* Grid on
CALL PIXFUN(0)
GO TO 1000
```

```
C--- Edit a polygon.  Locate the closest polygon and vertices within that
C---                  polygon.

2000  CONTINUE   /* EDIT POLY
      OGRDON = GRIDON

      CALL BUTTBL(0,0)
      CALL BUTTBL(1,0)
      CALL BUTTBL(2,55)    /* Always turn off buttbl 0, 0.
      CALL MACRO(56)   /* Display this sub-menu

      OBJNO = PICK(IX,IY,WHERE,IVERTS,EMPTY,ICOUNT,NOBJ) /* Locate the closest objec
t
                                                /* and vertex within that object.
      IF (GRIDON) CALL MACRO(154)   /* Grid erase
      CALL PIXFUN(0)
      CALL MOVABS(0,0)
      CALL VAL8(0)
      CALL MACRO(OBJNO+155)   /* DRAW THE POLYGON LOCALLY ON THE Z8002
      IF (GRIDON) CALL MACRO(154)   /* Grid on
      CALL PIXFUN(0)
      CALL MOVABS(0,0)
      J = ICOUNT(OBJNO)
      CALL VAL8(WHITE$)
      DRAGNG = .TRUE.

2001  CONTINUE/* Return here when in edit more than vertex on a given obj.
      CALL MOVABS(IVERTS(1,OBJNO),IVERTS(2,OBJNO))
      CALL PIXFUN(4)              /* XOR MODE

      DO 2003 I = 3,J,2
        CALL DRWABS(IVERTS(I,OBJNO),IVERTS(I+1,OBJNO))
2003  CONTINUE
      I = WHERE
      IF (I .EQ. 1) I = ICOUNT(OBJNO)
      CALL CLOAD(CREG21,IVERTS(I-2,OBJNO),IVERTS(I-1,OBJNO))
      CALL CLOAD(CREG22,IX,IY)
      I = WHERE

      CALL CLOAD(CREG23,IVERTS(I+2,OBJNO),IVERTS(I+3,OBJNO))

      IF (.NOT. FIRST) GOTO 2004
      CALL MACRO(51)   /* Track 2 lines, first time
      FIRST = .NOT. FIRST
      GOTO 2005

2004  CONTINUE
      IF (DRAGNG) GOTO 2005
      CALL MACRO(54)   /* Track cross hairs
      GOTO 2006

2005  CONTINUE
      CALL MACRO(51)   /* Rubberband 2 lines
```

```
2006  CONTINUE
      GRIDON = .FALSE.
      CALL READBU(1,1,IBUTT,IX,IY)
      CALL BUTTBL(0,0)
      IF ((IBUTT .EQ. 13) .AND. DRAGNG) GOTO 2010
      IF ((IBUTT .EQ. 13) .AND..NOT. DRAGNG) GOTO 2017
      IF (IBUTT .NE.  2) GOTO 2005

      IF (DRAGNG) GOTO 2010
C--- We were not dragging. Therefore find the vertex closest
      CALL LOCVTX(IX,IY,WHERE,ICOUNT,IVERTS(1,OBJNO))
      IX = ISNAP(IX, OGRDON)
      IY = ISNAP(IY, OGRDON)
      GOTO 2017

2010  CONTINUE
      IX = ISNAP(IX, OGRDON)
      IY = ISNAP(IY, OGRDON)
      IVERTS(WHERE,OBJNO) = IX
      IVERTS(WHERE+1,OBJNO) = IY
      IF (WHERE .NE. 1) GOTO 2015
      IVERTS(ICOUNT(OBJNO),OBJNO) = IX
      IVERTS(ICOUNT(OBJNO)+1,OBJNO) = IY

2015  CONTINUE
      CALL MACRO(52)
      CALL CLOAD(CREG22,IX,IY)
      CALL MACRO(52)

2017  CONTINUE /* Enter here when we go from tracking xhairs to dragging lines
      CALL MOVABS(IVERTS(1,OBJNO),IVERTS(2,OBJNO))
      J = ICOUNT(OBJNO)
      DO 2020 I = 3,J,2
        CALL DRWABS(IVERTS(I,OBJNO),IVERTS(I+1,OBJNO))
2020  CONTINUE

      DRAGNG = .NOT. DRAGNG
      IF (IBUTT .EQ. 2) GOTO 2001    /* Go back and edit more vertices

2025  CONTINUE  /* All done editing, redraw the new object.
      CALL BUTTBL(1,43)
      GRIDON = OGRDON  /* Restore the use of button 1

      IF (GRIDON) CALL MACRO(154)   /* Grid erase
      CALL PIXFUN(0)
      CALL MOVABS(0,0)
      CALL MACDEF(OBJNO+155)    /* DEFINE OBJECT MACRO
        CALL POLYGN(1,INVERT(OBJNO),IVERTS(1,OBJNO))
      CALL MACEND
      OBJNO = 0
      DO 2099 I = 1,NOBJ
        OBJNO = NXTOBJ(EMPTY,OBJNO+1)
        CALL VAL1K(ICOLOR(OBJNO))
        CALL MACRO(OBJNO+155)   /* DRAW THE POLYGON LOCALY ON THE Z8002
```

```
2099  CONTINUE
      CALL MACRO(57) /* Display main menu
      IF (GRIDON) CALL MACRO(154)    /* Grid on
      CALL PIXFUN(0)
      GOTO 1000
```

```
3000  CONTINUE  /* DELETE POLYGON
      OBJNO = PICK(IX,IY,WHERE,IVERTS,EMPTY,ICOUNT,NOBJ)
      IF (GRIDON) CALL MACRO(154)    /* Grid erase
      CALL PIXFUN(0)
      CALL MOVABS(0,0)
      CALL VAL8(0)
      CALL MACRO(OBJNO+155)   /* DRAW THE POLYGON LOCALLY ON THE Z8002
      CALL MACERA(OBJNO+155) /* ERASE THAT OBJECT, NOT NEED NOW.
      ICOUNT(OBJNO) = 0
      NOBJ = NOBJ - 1
      EMPTY(OBJNO) = .TRUE.
      IF (NOBJ .EQ. 0) GOTO 3008  /* DON'T ATTEMPT TO DRAW POLYGONS THAT DON'T EXIST
      OBJNO = 0
      DO 3007 I = 1,NOBJ
        OBJNO = NXTOBJ(EMPTY,OBJNO+1)
        CALL VAL1K(ICOLOR(OBJNO))
        CALL MACRO(OBJNO+155)    /* DRAW THE POLYGON LOCALLY ON THE Z8002
3007  CONTINUE
3008  IF (GRIDON) CALL MACRO(154)  /* Grid on
      CALL PIXFUN(0)
      GOTO 1005
```

```
4000  CONTINUE    /* READ COLOR
      JUNK(1) = ISNAP(IX, GRIDON)
      JUNK(2) = ISNAP(IY, GRIDON)
      J = SCOLOR(JUNK)
      CALL MOVABS(380,-300)
      CALL WINDOW(-512,-512,511,511)
      CALL CIRCLE(40)
      CALL WINDOW(XLLB,YLLB,XURB,YURB)
      GOTO 1005
```

```
5000  CONTINUE    /*   DRAG POLYGON
C     U TO BE DRAGGED
      CALL BUTTBL(1,0)
      CALL MACRO(153)  /* Display sub-menu
      CALL BUTTBL(0,0)

      OBJNO = PICK(IX,IY,WHERE,IVERTS,EMPTY,ICOUNT,NOBJ)

      IF (GRIDON) CALL MACRO(154)    /* Grid erase
      CALL PIXFUN(0)
      CALL MOVABS(0,0)
      CALL VAL8(0)
      CALL MACRO(OBJNO+155)  /* DRAW THE POLYGON LOCALLY ON THE Z8002
      CALL MOVABS(IVERTS(1,OBJNO),IVERTS(2,OBJNO))
      J = ICOUNT(OBJNO)
      CALL VAL8(WHITE$)
      IF (GRIDON) CALL MACRO(154)    /* Grid on
      CALL PIXFUN(4)               /* XOR MODE
C    SELECTED POLYGON IS DRAWN BY CALL TO MACRO (OBJNO +155)

      CALL VAL8(255)

C    CREG 30 HOLDS COORDS OF POLY IN ITS LAST POSITION
C    CREG 31 HOLDS COORDS OF POLY IN CURRENT POSITION
C    CREG 32 HOLDS COORDS OF THE FIRST POINT ON THE POLYGON

      CALL MACDEF(101) /* MACRO 101 IS EXECUTED OVER AND OVER
        CALL CMOVE(31,2) /* SAVE THE NEW POSITION IN CREG 31
        CALL CSUB(31,32) /* SUBTRACT AWAY THE START POINT
        CALL MOVI(30) /* MOVE BACK TO THE OLD POSITION
        CALL MACRO(OBJNO+155) /* UNDRAW THE POLY FROM THE OLD POSITION
        CALL MOVI(31) /* MOVE TO THE NEW POSITION
        CALL MACRO(OBJNO+155)  /* REDRAW THE POLY IN THE NEW POSITION
        CALL CMOVE(30,31) /* NOW MAKE THE CURRENT POS INTO THE OLD POS
        CALL WAIT(0)
      CALL MACEND  /* THATS IT

C    TO START THE WHOLE THING, CHANGE THE PIXEL FUNCTION AND
C    DRAW THE SHAPE THE FIRST TIME

      CALL CMOVE(30,2)    /* LOAD THE CURRENT POSITION
C   LOAD THE START POINT OF THE POLYGON INTO CREG 32
      CALL CLOAD(32,IVERTS(1,OBJNO),IVERTS(2,OBJNO))
      CALL CSUB(30,32)  /* SUBTRACT AWAY THE START POINT
      CALL MOVI(30)    /* AND MOVE THERE
      CALL PIXFUN(4) /* PIX FUNCTION TO XOR
      CALL PRMFIL(0)  /* SELECT UNFILLED PRIMS
      CALL XHAIR(0,0)  /* SHUT THE CROSSHARS OFF
      CALL MACRO(OBJNO+155) /* DRAW THE POLY
      CALL BUTTBL(0,101) /* AND LET IT RIP

C     WAIT FOR A CONFIRM
5057  CONTINUE
```

```
      CALL READBU(1,1,IBUTT,IX,IY)
      IF(IBUTT .EQ. 5 .OR. IBUTT .EQ. 13)GO TO 5058
      GO TO 5057

5058  CONTINUE
C     GOT THE CONFIRMATION, NOW, ERASE OLD WHITE PERIMETER
      CALL BUTTBL(0,5)

      IF (GRIDON) CALL MACRO(154)  /* Grid erase
      CALL PIXFUN(0)
      CALL VAL8(0) /* CHANGE COLOR TO BLACK
      CALL MOVI(30)
      CALL MACRO(155+OBJNO)

C     NOW REDEFINE THE POLYGON IN THE ARRAY TO REFLECT ITS NEW POSITION

      J = ICOUNT(OBJNO)*2
      IXDEL = ISNAP(IX, GRIDON) - IVERTS(1,OBJNO)
      IYDEL = ISNAP(IY, GRIDON) - IVERTS(2,OBJNO)
      DO 5060 I=1,J,2
        IVERTS(I,OBJNO)=IVERTS(I,OBJNO)+IXDEL
        IVERTS(I+1,OBJNO)=IVERTS(I+1,OBJNO)+IYDEL
5060  CONTINUE

C      REDO THE MACRO THAT REDRAWS THIS POLY

      CALL MACDEF(155+OBJNO)
        CALL POLYGN(1,INVERT(OBJNO),IVERTS(1,OBJNO))
      CALL MACEND

C      NOW REDRAW IT THE OLD FASHIONED WAY
      CALL PRMFIL(1)
      CALL MOVABS(0,0)

      OBJNO=0
      DO 5099 I= 1,NOBJ
        OBJNO = NXTOBJ(EMPTY,OBJNO + 1)
        CALL VAL1K(ICOLOR(OBJNO))
        CALL MACRO(OBJNO+155)
5099  CONTINUE

      IF (GRIDON) CALL MACRO(154)  /* Grid on
      CALL PIXFUN(0)
      CALL MACRO(57)  /* Display main menu
      GO TO 1000
```

```
7000  CONTINUE  /* Grid ON/Off
      CALL MACRO(154)  /* Complement the current grid state (i.e. If on,
                       /* turn if off. If off, turn it on.)
      CALL PIXFUN(0)
      GRIDON = .NOT. GRIDON
      GOTO 1000

9000  CONTINUE   /* PROCESS RETURN
      GO TO 6000   /* RETURN TO TOP MENU

9999  CONTINUE
      CALL QUIT
      CALL EXIT

6112  FORMAT(I3)
6113  FORMAT(I2,I3)
6114  FORMAT(I4)

      END
```

```
      SUBROUTINE SINGLE

C* <*>modellconsts.ins.ftn> (Created: Thursday, April 1, 1982) */

      INTEGER*2 WHITE$, BLACK$

      INTEGER*2 CREG0, CREG1, CREG2, CREG3, CREG4, CREG5
      INTEGER*2 CREG6, CREG7, CREG8, CREG9, CREG10,CREG11
      INTEGER*2 CREG12,CREG13,CREG14,CREG15,CREG16,CREG17
      INTEGER*2 CREG18,CREG19,CREG20,CREG21,CREG22,CREG23
      INTEGER*2 CREG24,CREG25,CREG26,CREG27,CREG28,CREG29
      INTEGER*2 CREG30,CREG31,CREG32,CREG33,CREG34,CREG35
      INTEGER*2 CREG36,CREG37,CREG38,CREG39,CREG40,CREG41
      INTEGER*2 CREG42,CREG43,CREG44,CREG45,CREG46,CREG47
      INTEGER*2 CREG48,CREG49,CREG50,CREG51,CREG52,CREG53
      INTEGER*2 CREG54,CREG55,CREG56,CREG57,CREG58,CREG59
      INTEGER*2 CREG60,CREG61,CREG62,CREG63

      INTEGER*2 CURPNT,JOYSTK,DIGTZR,CORRG ,SCRRG ,XHAIR0
      INTEGER*2 XHAIR1,LWNORG,UWNORG,OVRPL0,OVRPL1


      INTEGER*2 VREG0, VREG1, VREG2, VREG3, VREG4, VREG5
      INTEGER*2 VREG6, VREG7, VREG8, VREG9, VREG10,VREG11
      INTEGER*2 VREG12,VREG13,VREG14,VREG15

      INTEGER*2 CURVAL, XR0VAL, XR1VAL, FLMSK


      PARAMETER (WHITE$=255, BLACK$=0)

      PARAMETER (CREG0 = 0, CREG1 = 1, CREG2 = 2, CREG3 = 3)
      PARAMETER (CREG4 = 4, CREG5 = 5, CREG6 = 6, CREG7 =-1)
      PARAMETER (CREG8 =-1, CREG9 = 9, CREG10=10, CREG11=11)
      PARAMETER (CREG12=12, CREG13=13, CREG14=14, CREG15=15)
      PARAMETER (CREG16=16, CREG17=-1, CREG18=-1, CREG19=-1)
      PARAMETER (CREG20=20, CREG21=21, CREG22=22, CREG23=23)
      PARAMETER (CREG24=24, CREG25=25, CREG26=26, CREG27=27)
      PARAMETER (CREG28=28, CREG29=29, CREG30=30, CREG31=31)
      PARAMETER (CREG32=32, CREG33=33, CREG34=34, CREG35=35)
      PARAMETER (CREG36=36, CREG37=37, CREG38=38, CREG39=39)
      PARAMETER (CREG40=40, CREG41=41, CREG42=42, CREG43=43)
      PARAMETER (CREG44=44, CREG45=45, CREG46=46, CREG47=47)
      PARAMETER (CREG48=48, CREG49=49, CREG50=50, CREG51=51)
      PARAMETER (CREG52=52, CREG53=53, CREG54=54, CREG55=55)
      PARAMETER (CREG56=56, CREG57=57, CREG58=58, CREG59=59)
      PARAMETER (CREG60=60, CREG61=61, CREG62=62, CREG63=63)

      PARAMETER (CURPNT=CREG0,  JOYSTK=CREG1,  DIGTZR=CREG2)
      PARAMETER (CORRG =CREG3,  SCRRG =CREG4,  XHAIR0=CREG5)
      PARAMETER (XHAIR1=CREG6,  LWNORG=CREG9,  UWNORG=CREG10)
      PARAMETER (OVRPL0=CREG15, OVRPL1=CREG16)

      PARAMETER (VREG0 = 0, VREG1 = 1, VREG2 = 2, VREG3 = 3)
```

```
PARAMETER (VREG4 = 4, VREG5 = 5, VREG6 =-1, VREG7 =-1)
PARAMETER (VREG8 = 8, VREG9 = 9, VREG10=10, VREG11=11)
PARAMETER (VREG12=12, VREG13=13, VREG14=14, VREG15=15)

PARAMETER (CURVAL=VREG0,  XR0VAL=VREG1,  XR1VAL=VREG2)
PARAMETER (FLMSK =VREG3)

CALL MACDEF(40)
  CALL MOVI(CREG22)
  CALL DRWI(CREG21)
  CALL WAIT(0)
  CALL MOVI(CREG22)
  CALL CMOVE(CREG21,DIGTZR)
  CALL DRWI(CREG21)
  CALL WAIT(0)
CALL MACEND

CALL MACDEF(41)
  CALL DISPMU('SHOWS>.POLY>CREATE.MENU',23,.TRUE.)
  CALL XHAIR(0,0)
  CALL PIXFUN(4)
  CALL BUTTBL(0,40)           /* DRAG A SINGLE LINE
  CALL FLUSH
CALL MACEND

CALL MACDEF(42)   /* BUTTON 4 PUSHED, CONFIRM POINT.
  CALL MOVI(CREG22)
  CALL DRWI(CREG21)
  CALL MOVI(CREG22)
  CALL CMOVE(CREG21,CREG25)
  CALL DRWI(CREG21)
  CALL CMOVE(CREG22,CREG21)
  CALL BUTTBL(0,40)
CALL MACEND

CALL MACDEF(43)
  CALL BUTTBL(0,0)
CALL MACEND

CALL MACDEF(44)
  CALL DISPMU('SHOWS>.POLY>PE2.MENU',20,.TRUE.)
CALL MACEND

RETURN
END
```

```
      SUBROUTINE LOCVTX(IX,IY,WHERE,ICOUNT,IVERTS)
      INTEGER*2          IX,IY,WHERE,ICOUNT,IVERTS(1)
      INTEGER*2 I,J,K,MIN,DIST(1000)
      REAL*4    DX,DY

      DX = IX-IVERTS(1)
      DY = IY-IVERTS(2)
      DIST(1) = SQRT(DX*DX + DY*DY)
      DO 1000 I = 3,ICOUNT,2
        DX = IX-IVERTS(I)
        DY = IY-IVERTS(I+1)
        DIST(I) = SQRT(DX*DX + DY*DY)
1000  CONTINUE

      WHERE = 1
      IF (ICOUNT .EQ. 1) GOTO 3000
      K = DIST(1)
      DO 2000 I = 3,ICOUNT,2
        IF (K .LE. DIST(I)) GOTO 2000
        K = DIST(I)
        WHERE = I
2000  CONTINUE
3000  CONTINUE
      IX = IVERTS(WHERE)
      IY = IVERTS(WHERE+1)

      RETURN
      END
```

```
      SUBROUTINE DOUBLE

C* <*>modellconsts.ins.ftn> (Created: Thursday, April 1, 1982) */

      INTEGER*2 WHITE$, BLACK$

      INTEGER*2 CREG0, CREG1, CREG2, CREG3, CREG4, CREG5
      INTEGER*2 CREG6, CREG7, CREG8, CREG9, CREG10,CREG11
      INTEGER*2 CREG12,CREG13,CREG14,CREG15,CREG16,CREG17
      INTEGER*2 CREG18,CREG19,CREG20,CREG21,CREG22,CREG23
      INTEGER*2 CREG24,CREG25,CREG26,CREG27,CREG28,CREG29
      INTEGER*2 CREG30,CREG31,CREG32,CREG33,CREG34,CREG35
      INTEGER*2 CREG36,CREG37,CREG38,CREG39,CREG40,CREG41
      INTEGER*2 CREG42,CREG43,CREG44,CREG45,CREG46,CREG47
      INTEGER*2 CREG48,CREG49,CREG50,CREG51,CREG52,CREG53
      INTEGER*2 CREG54,CREG55,CREG56,CREG57,CREG58,CREG59
      INTEGER*2 CREG60,CREG61,CREG62,CREG63

      INTEGER*2 CURPNT,JOYSTK,DIGTZR,CORRG ,SCRRG ,XHAIR0
      INTEGER*2 XHAIR1,LWNORG,UWNORG,OVRPL0,OVRPL1


      INTEGER*2 VREG0, VREG1, VREG2, VREG3, VREG4, VREG5
      INTEGER*2 VREG6, VREG7, VREG8, VREG9, VREG10,VREG11
      INTEGER*2 VREG12,VREG13,VREG14,VREG15

      INTEGER*2 CURVAL, XR0VAL, XR1VAL, FLMSK


      PARAMETER (WHITE$=255, BLACK$=0)

      PARAMETER (CREG0 = 0, CREG1 = 1, CREG2 = 2, CREG3 = 3)
      PARAMETER (CREG4 = 4, CREG5 = 5, CREG6 = 6, CREG7 =-1)
      PARAMETER (CREG8 =-1, CREG9 = 9, CREG10=10, CREG11=11)
      PARAMETER (CREG12=12, CREG13=13, CREG14=14, CREG15=15)
      PARAMETER (CREG16=16, CREG17=-1, CREG18=-1, CREG19=-1)
      PARAMETER (CREG20=20, CREG21=21, CREG22=22, CREG23=23)
      PARAMETER (CREG24=24, CREG25=25, CREG26=26, CREG27=27)
      PARAMETER (CREG28=28, CREG29=29, CREG30=30, CREG31=31)
      PARAMETER (CREG32=32, CREG33=33, CREG34=34, CREG35=35)
      PARAMETER (CREG36=36, CREG37=37, CREG38=38, CREG39=39)
      PARAMETER (CREG40=40, CREG41=41, CREG42=42, CREG43=43)
      PARAMETER (CREG44=44, CREG45=45, CREG46=46, CREG47=47)
      PARAMETER (CREG48=48, CREG49=49, CREG50=50, CREG51=51)
      PARAMETER (CREG52=52, CREG53=53, CREG54=54, CREG55=55)
      PARAMETER (CREG56=56, CREG57=57, CREG58=58, CREG59=59)
      PARAMETER (CREG60=60, CREG61=61, CREG62=62, CREG63=63)

      PARAMETER (CURPNT=CREG0,  JOYSTK=CREG1,  DIGTZR=CREG2)
      PARAMETER (CORRG =CREG3,  SCRRG =CREG4,  XHAIR0=CREG5)
      PARAMETER (XHAIR1=CREG6,  LWNORG=CREG9,  UWNORG=CREG10)
      PARAMETER (OVRPL0=CREG15, OVRPL1=CREG16)

      PARAMETER (VREG0 = 0, VREG1 = 1, VREG2 = 2, VREG3 = 3)
```

```
PARAMETER (VREG4 = 4, VREG5 = 5, VREG6 =-1, VREG7 =-1)
PARAMETER (VREG8 = 8, VREG9 = 9, VREG10=10, VREG11=11)
PARAMETER (VREG12=12, VREG13=13, VREG14=14, VREG15=15)

PARAMETER (CURVAL=VREG0,  XR0VAL=VREG1,  XR1VAL=VREG2)
PARAMETER (FLMSK =VREG3)

CALL MACDEF(50)  /* Runs in buttbl 0 to rubber-band both lines.
  CALL MACRO(52)
  CALL CMOVE(CREG22,DIGTZR)
  CALL MACRO(52)
CALL MACEND

CALL MACDEF(51)  /* Invoked at the start of an edit session.
  CALL XHAIR(0,0)
  CALL PIXFUN(4)
  CALL BUTTBL(13,53)
  CALL BUTTBL(0,50)
  CALL FLUSH
CALL MACEND

CALL MACDEF(52)  /* Performs the actual draw when called from 50.
  CALL MOVI(CREG21)
  CALL DRWI(CREG22)
  CALL DRWI(CREG23)
CALL MACEND

CALL MACDEF(53) /* this is invoked at the end of an edit session.
                /* (I.e., button 13 waspushed to confirm the last edited
                /* vertex.)
  CALL BUTTBL(13,0)
  CALL BUTTBL(1,43)
  CALL BUTTBL(2,0)
CALL MACEND

CALL MACDEF(54)  /* invoked when were dragging a vertex and now want to
                 /* get another.
  CALL BUTTBL(0,5)  /* Restart xhairs tracking
  CALL XHAIR(0,1)
CALL MACEND

CALL MACDEF(55)
  CALL BUTTBL(0,0)
CALL MACEND


CALL MACDEF(56)
  CALL DISPMU('SHOWS>.POLY>EDIT.MENU',21,.TRUE.)
CALL MACEND

CALL MACDEF(57)
  CALL DISPMU('SHOWS>.POLY>PE2.MENU',20,.TRUE.)
CALL MACEND
```

```
RETURN
END
```

```
INTEGER*2 FUNCTION SCOLOR(X)
INTEGER*2  X(2),COLOR
REAL*4 XX,YY

XX = X(1)
YY = X(2)
COLOR = AND(INTS(SQRT(XX*XX/100.+YY*YY/100.)),63)
IF (COLOR .LE. 0) COLOR = LS(1,MOD(COLOR,5))
CALL VAL1K(COLOR)
SCOLOR = COLOR

RETURN
END
```

```
C* <*>dispmu.ftn> (Created: Thursday, June 3, 1982) */
      SUBROUTINE DISPMU(FN,FNL,HOME)
      INTEGER*2  FN(1),FNL
      LOGICAL*2 HOME

C--- SYSCOM keys are system defined constants and variables for system calls.
$INSERT SYSCOM>A$KEYS.INS.FTN
$INSERT SYSCOM>KEYS.INS.FTN

C* <*>modeliconsts.ins.ftn> (Created: Thursday, April 1, 1982) */

      INTEGER*2 WHITE$, BLACK$

      INTEGER*2 CREG0, CREG1, CREG2, CREG3, CREG4, CREG5
      INTEGER*2 CREG6, CREG7, CREG8, CREG9, CREG10,CREG11
      INTEGER*2 CREG12,CREG13,CREG14,CREG15,CREG16,CREG17
      INTEGER*2 CREG18,CREG19,CREG20,CREG21,CREG22,CREG23
      INTEGER*2 CREG24,CREG25,CREG26,CREG27,CREG28,CREG29
      INTEGER*2 CREG30,CREG31,CREG32,CREG33,CREG34,CREG35
      INTEGER*2 CREG36,CREG37,CREG38,CREG39,CREG40,CREG41
      INTEGER*2 CREG42,CREG43,CREG44,CREG45,CREG46,CREG47
      INTEGER*2 CREG48,CREG49,CREG50,CREG51,CREG52,CREG53
      INTEGER*2 CREG54,CREG55,CREG56,CREG57,CREG58,CREG59
      INTEGER*2 CREG60,CREG61,CREG62,CREG63

      INTEGER*2 CURPNT,JOYSTK,DIGTZR,CORRG ,SCRRG ,XHAIR0
      INTEGER*2 XHAIR1,LWNORG,UWNORG,OVRPL0,OVRPL1


      INTEGER*2 VREG0, VREG1, VREG2, VREG3, VREG4, VREG5
      INTEGER*2 VREG6, VREG7, VREG8, VREG9, VREG10,VREG11
      INTEGER*2 VREG12,VREG13,VREG14,VREG15

      INTEGER*2 CURVAL, XR0VAL, XR1VAL, FLMSK


      PARAMETER (WHITE$=255, BLACK$=0)

      PARAMETER (CREG0 = 0, CREG1 = 1, CREG2 = 2, CREG3 = 3)
      PARAMETER (CREG4 = 4, CREG5 = 5, CREG6 = 6, CREG7 =-1)
      PARAMETER (CREG8 =-1, CREG9 = 9, CREG10=10, CREG11=11)
      PARAMETER (CREG12=12, CREG13=13, CREG14=14, CREG15=15)
      PARAMETER (CREG16=16, CREG17=-1, CREG18=-1, CREG19=-1)
      PARAMETER (CREG20=20, CREG21=21, CREG22=22, CREG23=23)
      PARAMETER (CREG24=24, CREG25=25, CREG26=26, CREG27=27)
      PARAMETER (CREG28=28, CREG29=29, CREG30=30, CREG31=31)
      PARAMETER (CREG32=32, CREG33=33, CREG34=34, CREG35=35)
      PARAMETER (CREG36=36, CREG37=37, CREG38=38, CREG39=39)
      PARAMETER (CREG40=40, CREG41=41, CREG42=42, CREG43=43)
      PARAMETER (CREG44=44, CREG45=45, CREG46=46, CREG47=47)
      PARAMETER (CREG48=48, CREG49=49, CREG50=50, CREG51=51)
      PARAMETER (CREG52=52, CREG53=53, CREG54=54, CREG55=55)
      PARAMETER (CREG56=56, CREG57=57, CREG58=58, CREG59=59)
      PARAMETER (CREG60=60, CREG61=61, CREG62=62, CREG63=63)
```

```
      PARAMETER (CURPNT=CREG0,  JOYSTK=CREG1,  DIGTZR=CREG2)
      PARAMETER (CORRG =CREG3,  SCRRG =CREG4,  XHAIR0=CREG5)
      PARAMETER (XHAIR1=CREG6,  LWNORG=CREG9,  UWNORG=CREG10)
      PARAMETER (OVRPL0=CREG15, OVRPL1=CREG16)

      PARAMETER (VREG0 = 0, VREG1 = 1, VREG2 = 2, VREG3 = 3)
      PARAMETER (VREG4 = 4, VREG5 = 5, VREG6 =-1, VREG7 =-1)
      PARAMETER (VREG8 = 8, VREG9 = 9, VREG10=10, VREG11=11)
      PARAMETER (VREG12=12, VREG13=13, VREG14=14, VREG15=15)

      PARAMETER (CURVAL=VREG0,  XR0VAL=VREG1,  XR1VAL=VREG2)
      PARAMETER (FLMSK =VREG3)

      INTEGER*2 XLL1,YLL1, XUR1,YUR1      /*  SCREEN BOUNDIRES
      PARAMETER (XLL1=-960,YLL1=-930, XUR1=958,YUR1=988)

      INTEGER*2 XLLB,YLLB, XURB,YURB     /* WORK AREA BOUNDRIES
      PARAMETER (XLLB=-478,YLLB=-220, XURB=464,YURB=380)

      INTEGER*2 XLLH,YLLH, XURH,YURH     /* HOME MENU BOUNDRIES
      PARAMETER (XLLH=-474,YLLH=-478, XURH=-280,YURH=-242)

      INTEGER*2 UPXFUN,UBTTB0,UPRMFL,URTNP,GOHOME
      PARAMETER (UPXFUN=250,UBTTB0=251,UPRMFL=252,URTNP=253,
     +           GOHOME=215)


      INTEGER*2 I,J,K,UNIT,FUNIT,CODE,L(40)

      CALL OPEN$A(A$READ+A$GETU,FN,FNL,UNIT,CODE)
      IF (CODE .NE. 0) CALL ERRPR$(K$NRTN,CODE,FN,FNL,'OPEN$A',6)
      FUNIT = UNIT + 12

      CALL MOVABS(-474,-478)
      CALL WINDOW(-512,-512, 511,511)
      CALL CLOAD(CREG26,-240,-242)
      CALL TEXTC(32,0)
      CALL PIXFUN(0)
      CALL PRMFIL(1)
      CALL BUTTBL(0,0)
      CALL VAL8(BLACK$)
      CALL RECTI(CREG26)  /* CREG26 HOLDS COORDS OF UPPER RIGHT CORNER OF MENU
      CALL MOVABS(-474,-242)
      CALL VAL8(WHITE$)
      DO 3000 I = 1,20
        CALL MOV3R(0,-20)
        READ(FUNIT,100,END=4000) L
100     FORMAT(40A2)
        K = 40
        DO 1000 J = 1,39
          IF (L(K) .NE. '  ') GOTO 2000
          K = K - 1
1000    CONTINUE
```

```
2000     CONTINUE
         IF ((.NOT. HOME) .AND. (I .EQ. 5)) CALL VAL1K(8)
         CALL TEXT1(K*2,L)
3000  CONTINUE

4000  CONTINUE
      CALL SRCH$$(K$CLOS,0,0,UNIT,0,0)
      CALL WINDOW(XLLB,YLLB, XURB,YURB)

      RETURN
      END
```

```
      SUBROUTINE FATTXT(T,L,X,Y)
      INTEGER*2  T(1),X,Y,L

      CALL MOVABS(X,Y)
      CALL TEXT1(L,T)
      CALL MOVABS(X,Y+1)
      CALL TEXT1(L,T)
      CALL MOVABS(X+1,Y+1)
      CALL TEXT1(L,T)
      CALL MOVABS(X+1,Y)
      CALL TEXT1(L,T)

      RETURN
      END

      INTEGER*2 FUNCTION ISNAP(I, GRIDON)
      INTEGER*2 I
      LOGICAL*2 GRIDON

      ISNAP = I
      IF (.NOT. GRIDON) RETURN     /*  Don't snap unless the grid is on.

      IF (I .LT. 0) GOTO 1000
      ISNAP = (I+10)/20*20
      GOTO 2000

1000  CONTINUE
      ISNAP = (I-10)/20*20

2000  CONTINUE
      RETURN
      END
```

```
C--- NXTOBJ
C SEARCH FROM START TO END FOR A NON-EMPTY ITEM.

      INTEGER*2 FUNCTION NXTOBJ(MTYLST,START)
      INTEGER*2 START,I
      LOGICAL*2 MTYLST(1)

      INTEGER*2 MAXOBJ,MXOBJ2,MAXVRT
      PARAMETER (MAXOBJ=50,MXOBJ2=MAXOBJ*2,MAXVRT=200)

      DO 1000 I = START,MAXOBJ
        IF (.NOT. MTYLST(I)) GOTO 2000
1000  CONTINUE

C--- ERROR
      CALL TNOU('*** Error, too many objects.',28)
      RETURN

2000  CONTINUE  /* FOUND THE NEXT OBJECT.
      NXTOBJ = I
      RETURN
      END

C--- NXTMTY
C SEARCH FROM THE FIRST ELEMENT IN THE LIST UNTIL WE FIND AN EMPTY LIST.

      INTEGER*2 FUNCTION NXTMTY(MTYLST)
      LOGICAL*2 MTYLST(1)
      INTEGER*2 I

      INTEGER*2 MAXOBJ,MXOBJ2,MAXVRT
      PARAMETER (MAXOBJ=50,MXOBJ2=MAXOBJ*2,MAXVRT=200)

      DO 1000 I = 1,MAXOBJ
        IF (MTYLST(I)) GOTO 2000
1000  CONTINUE

C--- ERROR
      CALL TNOU('*** Error, no empty slots.',26)
      RETURN

2000  CONTINUE   /* FOUND AN EMPTY SLOT, RETURN IT.
      NXTMTY = I
      RETURN
      END
```

```
C--- PICK, Pick a point within the closest object.

      INTEGER*2 FUNCTION PICK(IX,IY,WHERE,IVERTS,EMPTY,ICOUNT,NOBJ)

      INTEGER*2 MAXOBJ,MXOBJ2,MAXVRT
      PARAMETER (MAXOBJ=50,MXOBJ2=MAXOBJ*2,MAXVRT=200)

      INTEGER*2 IX,IY,WHERE,IVERTS(MAXVRT,MAXOBJ),ICOUNT(MAXOBJ),NOBJ
      LOGICAL*2 EMPTY(MAXOBJ)
      INTEGER*2 A(MXOBJ2),B(MAXOBJ),OBJNO(MAXOBJ)
      INTEGER*2 I,J

      OBJNO(1) = 1
      DO 2050 I = 1,NOBJ
        J = NXTOBJ(EMPTY,OBJNO(I))
        OBJNO(I) = J
        A(J*2-1) = IX
        A(J*2) = IY
        CALL LOCVTX(A(J*2-1),A(J*2),B(J),
     +              ICOUNT(J),IVERTS(1,J))
        OBJNO(I+1) = J+ 1
2050  CONTINUE
      CALL LOCVTX(IX,IY,WHERE,NOBJ*2-1,A)
      PICK = OBJNO(WHERE/2+1)
      WHERE = B(WHERE/2+1)

      RETURN
      END
```

```
      SUBROUTINE BLDGRD
      INTEGER*2  I

      CALL MACDEF(154)
        CALL PIXFUN(4)
        CALL VAL1K(1)  /* Very dim blue.
        CALL MOVABS(-512,0)
        DO 1000 I = 1,10
          CALL DRWREL(1024,0)
          CALL MOVREL(0,20)
          CALL DRWREL(-1024,0)
          CALL MOVREL(0,20)
1000    CONTINUE
        CALL MOVABS(0,400)
        DO 2000 I = 1,12
          CALL DRWREL(0,-620)
          CALL MOVREL(-20,0)
          CALL DRWREL(0,620)
          CALL MOVREL(-20,0)
2000    CONTINUE
        CALL MOVABS(-512,-20)
        DO 3000 I = 1,5
          CALL DRWREL(1024,0)
          CALL MOVREL(0,-20)
          CALL DRWREL(-1024,0)
          CALL MOVREL(0,-20)
3000    CONTINUE
        CALL MOVABS(20,400)
        DO 4000 I = 1,11
          CALL DRWREL(0,-620)
          CALL MOVREL(20,0)
          CALL DRWREL(0,620)
          CALL MOVREL(20,0)
4000    CONTINUE
        CALL MOVABS(0,0)
      CALL MACEND

      RETURN
      END
```