8320

OmniVU Industrial Operator Interface

Printed in the United States of America
Part Number 94536-001

XYCOM
750 North Maple Road
Saline, Michigan 48176
(313) 429-4971

XYCOM REVISION RECORD

Revision	*Description*	*Date*
A	Manual Released	11/89

Trademark Information

- *IBM PC, AT, XT, and VGA are registered trademarks of the International Business Machines Corporation*
- *MS-DOS is a registered trademark of the Microsoft Corporation*

Copyright Information

The information contained within this document is subject to change without notice. Xycom does not guarantee the accuracy of the information and makes no commitment toward keeping it up to date.

Address comments concerning this manual to:

xycom
Technical Publications Dept.
750 North Maple Road
Saline, Michigan 48176

Part Number: ***94536-001 A***

TABLE OF CONTENTS

TABLE OF CONTENTS cont.

TABLE OF CONTENTS cont.

TABLE OF CONTENTS cont.

LIST OF FIGURES

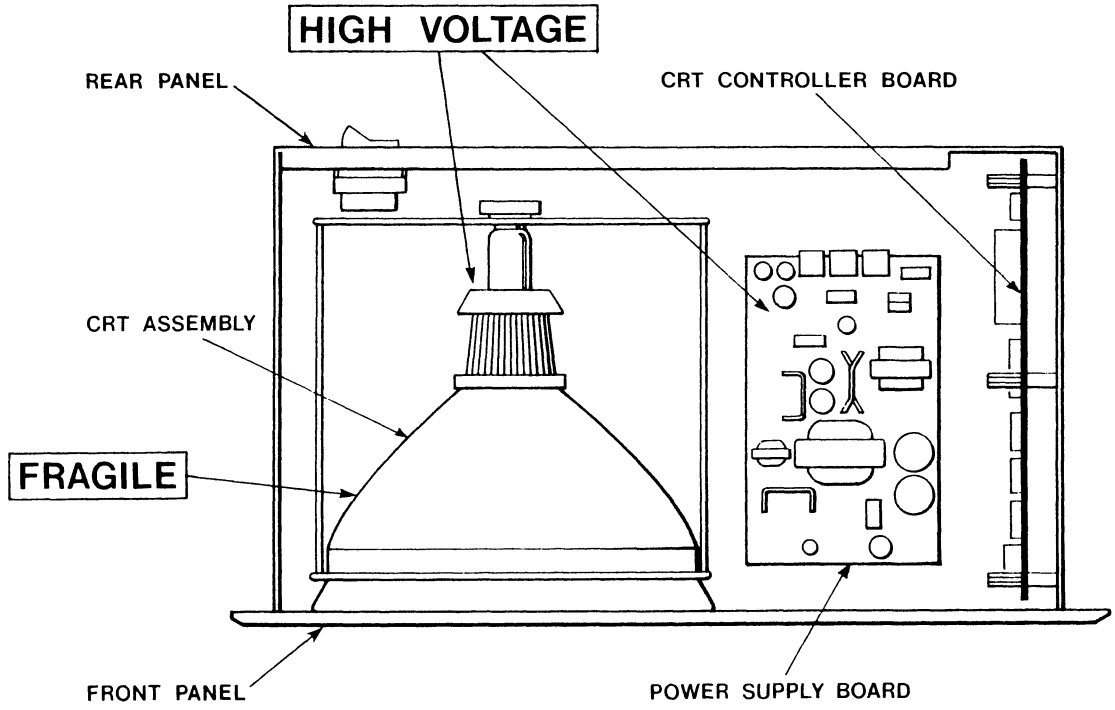
LIST OF TABLES

WARNING

Dangerous voltages are present within the OmniVU 8320 Terminal. These voltages will remain after all electrical power is turned off. Use caution whenever the front panel is opened. Avoid touching high-voltage areas within the OmniVU. Do not work alone.

WARNING

The FRAGILE Cathode Ray Tube (CRT) is exposed when the front panel is opened. Wear safety glasses in case of accidental breakage. Internal coating of CRT is extremely TOXIC. If exposed RINSE IMMEDIATELY and consult a physician.



Chapter 1

INTRODUCTION

1.1 INTRODUCTION

The Xycom OmniVU® 8320 is a color, CRT-based display terminal specifically designed for reliability under the extreme conditions of shock, vibration, temperature, and humidity found on a plant floor. It combines many standard desktop terminal features, comprehensive graphics, and a flexible configuration.

The ruggedness and flexibility make the OmniVU an ideal industrial interface in such applications as machine and process control, on-site data entry, and system diagnosis. It will provide an interactive window into any industrial system.

The OmniVU 8320 seals to both NEMA 4 and NEMA 12 specifications. It can be mounted in a standard 19" rack or in an equipment enclosure panel. A 13"-diagonal CRT (with impact-resistant shield), built-in sealed-membrane keypads, and external keyboard port make the 8320 a complete, self-contained unit.

The OmniVU terminal can exchange data over serial communication lines with virtually any computer: mini, mainframe, board-level micro, PLC BASIC module, or any host with a standard serial port. With the OIL-2 option card installed, the OmniVU can connect directly to many PLCs and their networks.

OmniVU comes in a standard configuration that will emulate the DEC VT100/220 terminal family. With the unique "soft-architecture" of OmniVU, the user can configure the terminal to emulate other standard terminals.

The built-in memory allows storage of text messages and graphics images. A fully integrated development environment includes the drawing of symbols, diagrams, and other graphics to enhance performance and productivity on the plant floor. They can be invoked either from the keypads or remotely.

The rear panel design of the OmniVU allows connection with a multitude of devices. Other computers, PLCs, an external monitor, and an auxiliary keyboard are but a few. A relay output permits control of warning sirens, buzzers, etc. User installed expansion cards can provide additional serial ports.

1.2 MAJOR FEATURES

The OmniVU 8320 has is an open, versatile system. Its features make the system compatible with an expansive array of both industrial and non-industrial interfaces. Built-in versatility includes:

- Two optically isolated serial ports
- Printer port
- Time-of-Day Clock/Calendar (battery-backed)
- Battery-backed memory
- Mouse interface
- Various text sizes
- Various graphic shapes and colors
- Self-diagnostics
- Character Editor
- Graphics function library and Picture Editor
- Font Editor
- Programmable keypad keys
- Password protection
- Acceptance of up to 2 option cards
- Support of Xycom OIL-2, providing communication with Allen-Bradley Data Highway, MODICON MODBUS, Texas Instruments, Square D SY/MAX, Westinghouse, GE, Siemens, and OPTO 22 OPTOMUX PLCs.
- Front and rear-mounted keyboard ports

1.3 MANUAL STRUCTURE

The first chapter provides an overview that outlines the general specifications and functional capabilities of the OmniVU 8320 Industrial Operator Interface. The remaining chapters will discuss the product in more detail as follows:

Chapter One - **Introduction**: general descriptions of OmniVU 8320, including specifications, major features, and sample applications

Chapter Two - **Installation**: information about configuration, setup of hardware, and start-up

Chapter Three - **Menu Overview**: descriptions of the various libraries within the operator interface including command definitions, key definitions, pictures, fonts, and text messages. Also discussion on resource ownership and file types

Chapter Four - **Remote Commands**: a summary of the default and user-defined commands

Chapter Five - **Operator Keys**: information required for familiarity with both the default and user-defined functions of the operator keys

Chapter Six - **Picture Editor**: general descriptions of the features found within the graphics editor, including use of the mouse and keyboard

Chapter Seven - **Font Editor**: descriptions of the components of the font editor and its uses

Chapter Eight - **Terminal Primitives**: descriptions of the terminal primitives, those commands that perform actions within the terminal. These can be altered to suit any function.

Appendix A - **Mounting Dimensions**: diagram depicting the stud locations for rack-mounting the OmniVU 8320

Appendix B - **Quick Reference Guide**: a quick reference list of the connector pinouts, remote commands, terminal primitives, and more.

Appendix C - **ASCII Codes**: a quick reference table depicting the ASCII codes used

1.4 SPECIFICATIONS

Table 1-1 contains specifications for the OmniVU 8320 Operator Interface Terminal.

Table 1-1 8320 Specifications

CHARACTERISTIC	SPECIFICATION
Mechanical	
Dimensions	
Height	15.7" (399 mm)
Width	19" (483 mm)
Depth	17.5" (445 mm)
Weight	40 lbs. (18.14 kg)
Mounting	EIA standard 19" rack or panel
Keypads	sealed membrane type 20 Function Keys 35 Numeric/Hex Keys

Table 1-1 cont.

CHARACTERISTIC	SPECIFICATION
Monitor	VGA 640 x 480 pixels 256 simultaneous colors
Serial Ports	2 RS-232C or RS-485 independently selectable optically isolated
Parallel Port	Centronics-compatible parallel printer port
Relay contacts	NO and NC
Electrical	
Power Supply	115 VAC @ 60 Hz 230 VAC @ 50 Hz switch-selected
Environmental	
Temperature	
Operating	0 °C to 50 °C
Non-operating	-40 °C to 65 °C
Humidity	10 to 80% relative, non-condensing
Shock	
Operating	15 g peak acceleration (11 msec duration)
Non-operating	30 g peak acceleration (11 msec duration)
Vibration	
Operating	.006" peak-to-peak 1.0 g max
Non-operating	.015" peak-to-peak 2.5g max

Chapter 2

INSTALLATION

2.1 INTRODUCTION

This chapter describes the various components of the Xycom 8320 OmniVU Industrial Operator Interface, and includes the necessary information for configuring the hardware before power-up.

Many optional configurations are available for the OmniVU. Expansion modules and direct-connect handlers are among the possibilities. This manual will discuss only those features which are included within the base unit. Optional equipment from Xycom will include all additional documentation related to the performance of the 8320.

2.2 SYSTEM REQUIREMENTS

The 8320 OmniVU is a versatile industrial operator interface. To operate, the various hardware components must be properly installed and configured. Minimum recommended hardware requirements for operation of the 8320 are:

- The base terminal
- An external PC/AT keyboard
- A serial mouse

The external keyboard is used for programming. The mouse is primarily used within the editors in the OmniVU (e.g., Font Editor).

The following sections discuss and illustrate the installation of components for the OmniVU. The user should become familiar with all of the pertinent components before beginning with any installation or operation.

2.3 Front Panel

The OmniVU is equipped with a NEMA 4/NEMA 12 sealed front panel. This panel protects the interior of the system from harsh environmental conditions whenever the system is properly panel-mounted. Features visible on the front panel include:

Monitor — Protected from breakage by a high-impact shield, the 13" EGA monitor displays graphics in high resolution color.

Function Keys — These twenty fully-operable sealed keys are located directly below the monitor. They can be configured to perform a variety of tasks by the user.

PC/AT Keyboard Port — This enclosed port allows a PC/AT compatible keyboard to be attached to the front of the system.

Data Entry Pad — This sealed 3x11 keypad includes: 0 through 9, DEL, ESC, *, #, +, -, cursor arrows, and Enter.

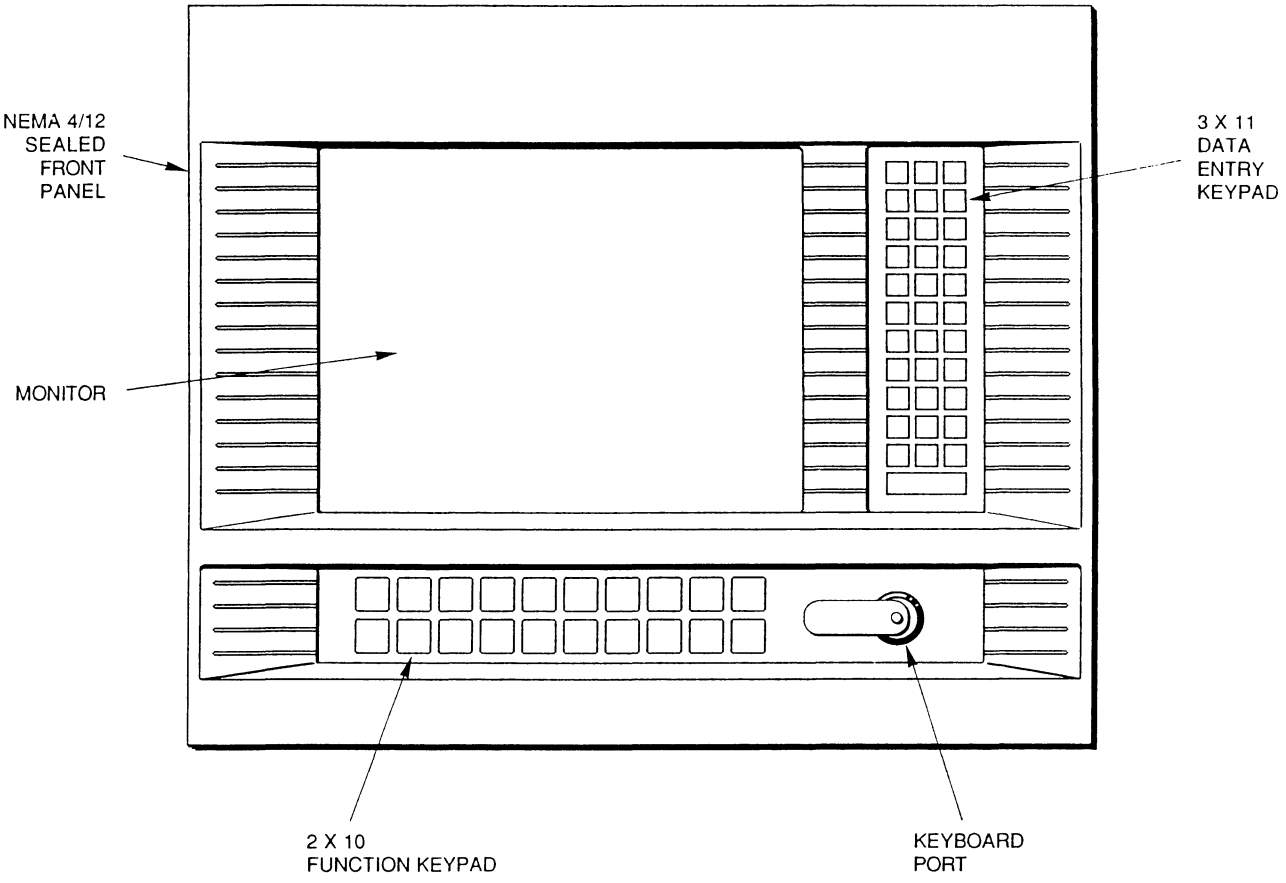


Figure 2-1 OmniVU Front Panel

2.4 Rear Panel

OmniVU's rear panel offers access to the various ports, the power assembly, contrast and brightness controls, and optional expansion modules. Figure 2-2 shows the features visible on the back panel:

Power Switch ——— This switch controls the power flow to the terminal.

Power Receptacle ——— Located under the Power Switch, it is the attachment point for the power cord.

Fuse Receptacle ——— The fuse for the terminal is held behind the black plastic access door.

Adjustment Knobs/Holes ——— Used to control the brightness and contrast of the color monitor. Text knob places the terminal in text mode, and controls the intensity of the text.

Expansion Sites ——— There are two expansion sites next to the OmniVU controller card slot for the installation of additional boards, such as memory or OIL-2 cards.

Voltage Select ——— Selects either 115VAC or 230VAC input.

Battery ——— Keyed cover for RAM backup battery.

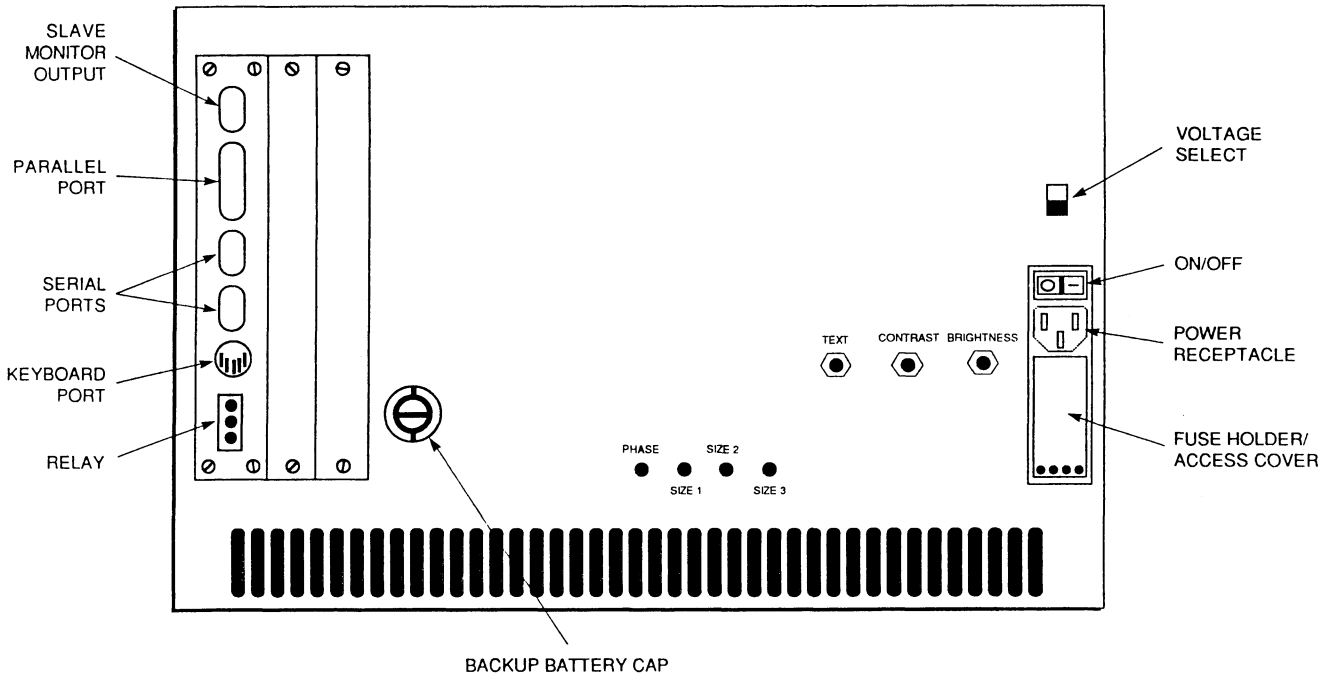


Figure 2-2 OmniVU Back Panel

2.5 INSTALLATION

> **WARNING**
> Before connecting electrical power to the terminal, ensure that the Power Switch is set to **OFF**.
>
> The power cable must be connected to a properly grounded outlet. **DO NOT** use an adapter plug that prevents the terminal from being properly grounded through the power cable.

1. Install any expansion modules or other circuit boards in the 8320. Instructions are provided with all Xycom optional equipment.
2. Secure the 8320 in a suitable mounting location (refer to Appendix A).
3. Connect the female end of the power cable to the terminal electrical power receptacle. Attach the other end to an appropriate power source.
4. Connect any host devices or peripherals to the appropriate ports.
5. Set power switch to **ON**.
6. Adjust the contrast and brightness knobs.

> **CAUTION**
> The four holes on the back of the terminal (labelled Phase, Size 1, Size 2, and Size 3), are for testing purposes **only. DO NOT** insert anything into them. They are to be used by trained Xycom personnel only.

Once the power-up diagnostics are complete, a blue screen will appear. Type the following to continue:

<CTRL><BREAK>

The 8320 OmniVU terminal is ready for use. Chapter 3 will discuss the operational aspects. The remainder of this chapter is devoted to mouse installation, basic user maintenence procedures, wiring descriptions of the various ports and connectors found on the rear panel, and memory units.

2.5.1 **Mouse Installation**

> NOTE
> Installation includes configuration, which in turn will depend upon the mouse type. The Microsoft Corp. mouse will be set to the **Microsoft** setting. The PC Mouse by Mouse Systems, Inc. and the mouse by Logitech, Inc. will be set to **PC Mouse.**

1. With the power set to **OFF**, locate the two serial ports at the rear of the system (from the top, counting downward, they are the second and third connections).

2. Attach the mouse plug to the desired port. (The upper is Serial Port 1, the lower Serial Port 2). Selected port must be configured to the RS-232 standard.

3. Turn the power to **ON**.

4. At the cursor, type **<CTRL><BREAK>**. The Main Menu should appear.

5. Type "**O**ptions **B**ase **M**ouse" from the keyboard.

6. Use the left and right cursor keys to select the Port.

7. Use the down cursor to move to mouse type, and use the left and right keys to select either Microsoft or PC Mouse. Press the **<ESC>** key on either the keypad or the keyboard.

The mouse should then operate. If it does not, return to the Mouse Configuration Menu and redefine either the port number or mouse type.

> NOTE
> The OmniVU will accommodate either a two or three button mouse. On the three button mouse, the center button is not used.

2.5.2 Connecting the Optional Keyboard

There are keyboard connectors on both the front and back of the terminal. Unscrew the protective cap to connect a keyboard to the sealed front connector. The rear connector is not sealed; simply plug the keyboard into it. Both connectors are keyed to prevent incorrect alignment. For both applications, set the switch on bottom of keyboard to the "AT" position.

CAUTION
The cap should be installed on the front panel when the optional keyboard is not attached.

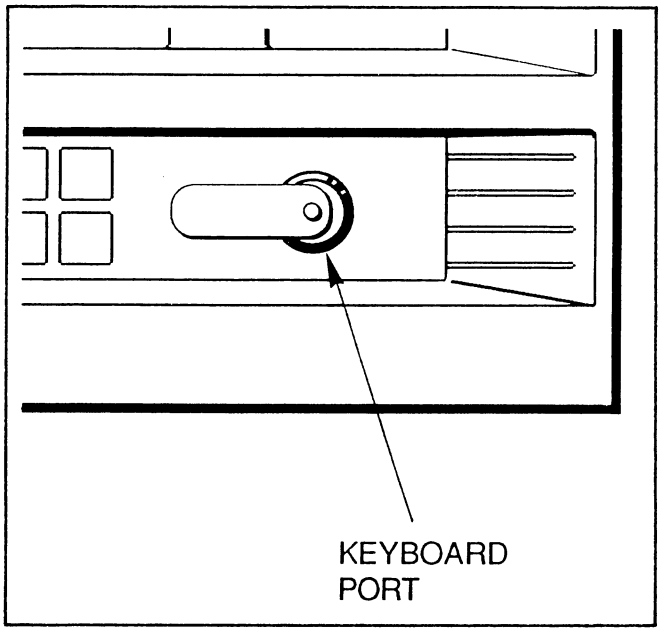


Figure 2-3 Keyboard Connection

2.5.3 **Replacing the Fuse**

The fuse holder/access door is located under the power receptacle. To remove, simply pry it open. Check the fuse for serviceability and correct rating (1.5 amp).

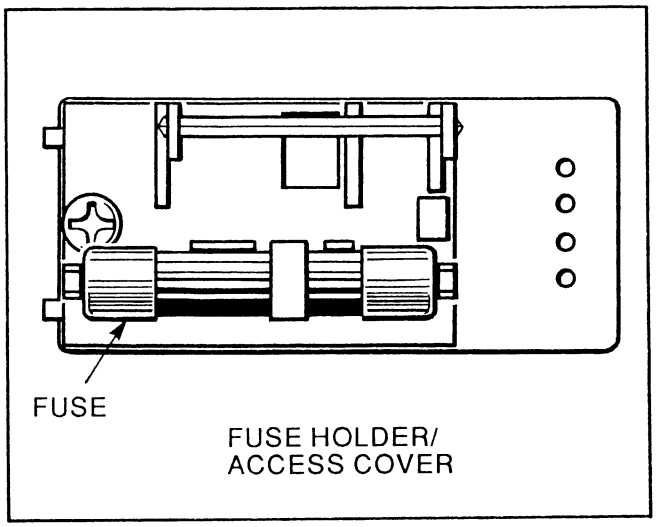


Figure 2-4 Fuse Holder/Access Door

2.5.4 **Replacing the Backup Battery**

Insert a screwdriver into the slotted cap and turn so that it is vertical. Pry cap off and remove battery.

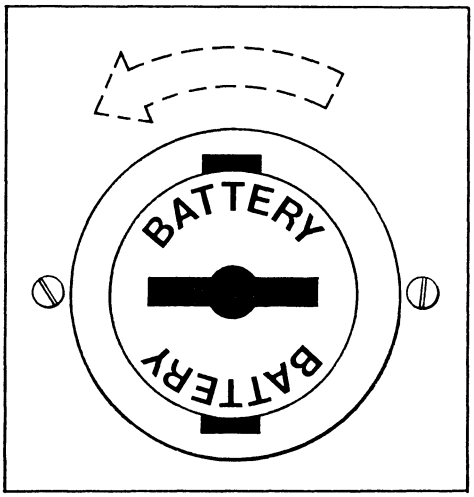


Figure 2-5 Battery Replacement

2.6 PORTS

2.6.1 Printer Port

This is the uppermost connector on the ORB. It is a 25-pin D-type female connector.

Table 2-1 Printer Port

PIN	SIGNAL	PIN	SIGNAL
1	STROBE	10	ACK
2	DATA0	11	BUSY
3	DATA1	12	PE
4	DATA2	13	SELECT
5	DATA3	14	AUTOFEED
6	DATA4	15	ERROR
7	DATA5	16	INIT
8	DATA6	17	SELECT-IN
9	DATA7	18-25	GND

2.6.2 Keyboard Ports

Both are standard, 5-pin DIN keyboard connectors, keyed to permit only one connection alignment.

Table 2-2 Keyboard Connector

PIN	SIGNAL
1	CLOCK
2	DATA
3	N/C
4	GND (SG)
5	+5 VDC
Shell	GND (FG)

2.6.3 Relay Output Port

This connector consists of three small screws to attach up to three wires, surrounded by a black plastic casing.

Table 2-3 Relay

PIN	SIGNAL
1	Normally Closed
2	Common
3	Normally Open

2.6.4 Video Connector

This is a 15-pin D-type female connector, with the standard VGA signals.

Table 2-4 Video Connector

PIN	SIGNAL	PIN	SIGNAL
1	RED	9	N/C
2	GREEN	10	GND
3	BLUE	11	N/C
4	N/C	12	N/C
5	GND	13	HSYNC
6	RGND	14	VSYNC
7	GGND	15	N/C
8	GBLUE		

If an external monitor is connected to the OmniVU, SW1 on the controller board (located near the top) must be reconfigured. When only the internal monitor is connected, all four switches on SW1 are positioned with the arrow on the switch (factory setting). When a slave monitor is connected, all four switches must be positioned against the arrow.

> NOTE
> Only VGA color monitors will work as slave monitors.

2.6.5 Serial Ports

Serial Ports 1 and 2 can be configured to either RS-232 or RS-485 (they are shipped from Xycom as RS-232). They are located below the printer port, and are both 9 pin DIN male connectors. Configuring the ports to RS-485 and/or back to RS-232 requires certain jumpers to be altered (refer to Section 2.6.6). Both pinouts are listed below.

Table 2-5 RS-232 Port

PIN	SIGNAL	PIN	SIGNAL
1	DCD	6	N/C
2	RXD	7	RTS
3	TXD	8	CTS
4	DTR	9	N/C
5	GND		

Table 2-6 RS-485 Port

PIN	SIGNAL	PIN	SIGNAL
1	TXD-	6	RXD-
2	TXD+	7	RXD+
3	RTS-	8	CTS+
4	RTS+	9	CTS-
5	GND		

2.6.6 Serial Port Jumpers

The jumpers in Table 2-7 must be in the correct positions to match the desired configuration for the Serial Ports. The locations of these jumpers is shown in Figure 2-8. The factory settings are RS-232.

Table 2-7 Serial Port Jumpers

JUMPER	COM1 RS-232	RS-485	JUMPER	COM2 RS-232	RS-485
J10	OUT	IN	J48	OUT	IN
J11	B	A	J32	B	A
J18	B	A	J34	B	A
J19	B	A	J35	B	A
J20	B	A	J36	B	A
J21	B	A	J37	B	A
J23	B	A	J39	B	A
J24	B	A	J40	B	A
J26	B	A	J43	B	A

When configured for RS-485, the inputs CTS and RXD may be terminated. Each signal for each port is independently enabled by a pair of jumpers. Removing the jumpers as indicated below will terminate a specific signal.

COM 1:	RXD: J14 and J15 CTS: J12 and J13	COM 2:	RXD: J54 and J55 CTS: J52 and J53

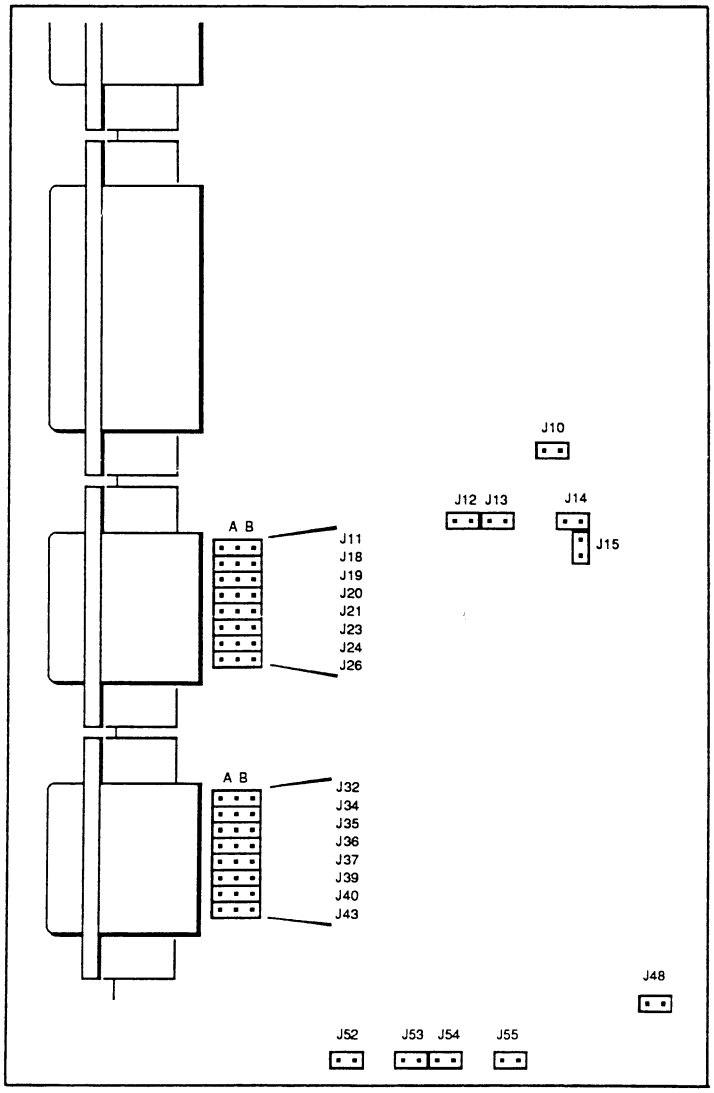


Figure 2-6 Jumper Locations

2.7 MEMORY UNITS

The OmniVU supports three memory units: A, B, and C. Each unit operates independently from one another, each with a separate directory. These letter designations refer to the three slots in the OmniVU terminal. Unit A refers to the OmniVU base, or controller, board. Units B and C refer to the memory space on the option card or cards.

Whether an option board is designated B or C depends on the option card jumper configuration. Table 2-8 shows the configuration.

NOTE
This jumper listing is for the 8110 OIL-2 Option Card **only**. It does not apply to the 8320 Controller Card.

CAUTION
DO NOT configure two option cards as the same memory unit.

Table 2-8 Option Card Memory Configuration

Jumper	Unit B	Unit C
J43	A	B
J47	B	A

The memory on a memory unit is sized and initialized by OmniVU at power-up when it detects that uninitialized memory exists on a board. If the amount of memory on a unit is modified, the memory must be cleared for OmniVU to resize and initialize. Otherwise the change will not be recognized.

Memory sizes are in blocks of 64K. The maximum size on the controller board is 256K. The chips must be installed in a particular order for the controller to recognize them. The first 128K (first and second 64K blocks) are included on every controller board. Correct chip installation is key to the operation of the controller board. Refer to Table 2-9 for a guide:

Table 2-9 Memory Chip Order

Chip Slot	Memory Level
U113, U114	64K
U103, U104	128K
U87, U86	192K
U78, U79	256K

Use only 32K x 8 byte wide, 120 ns, static RAM chips (low-power, battery-backed). Currently, only 32K byte CMOS RAM chips are supported, so there is always an even number of CMOS RAM chips.

2.7.1 File Types

Each of the memory types contains a directory, and each directory a series of files. The number of directory entries is limited to 4095. File names can be up to 20 characters long. The following is a list of the types of files:

File Type	Default Memory Unit	Status
Fonts	A	Configurable
Pictures	A	Configurable
Keypad Definitions	A	Configurable
Base Configuration	A	**Not Configurable**
Remote Commands	A	Configurable
Terminal Configuration	A	**Not Configurable**
Special Terminal Def.s	A	Configurable
Terminal Image file	A	Configurable
OIL-2 Programs	B	Configurable
OIL-2 Registers	B	**Not Configurable**
OIL-2 Autoexecute	B	**Not Configurable**
OIL-2 Configuration	B	**Not Configurable**

Files listed as Configurable can be saved and restored to any memory unit. Files marked **Not Configurable** must never be moved or restored to any other than the default memory unit. To do so would possibly cause the unit to operate erratically.

Chapter 3

OMNIVU OVERVIEW

3.1 INTRODUCTION

The following is an overview of the features of the OmniVU 8320 Operator Interface menu. It will show the user how to access the various functions of the OmniVU Operator Interface. All of the OmniVU functions are accessed through a series of user friendly menus. These guide the user through the appropriate selections, and provide a help option at key places throughout.

There are different methods of accessing the menu selections. These are discussed in the following section.

> NOTE
> When the manual refers to typing a title to gain access to a menu or function, it refers to **only** typing the highlighted letters of the expression within the quotation marks.

3.2 SELECTING MENUS

To view menu selections, type **<CTRL><BREAK>** from the keyboard or **<+><down arrow>** from the keypad (disabled in "**O**ptions **B**ase **K**eypad" menu).

The selections to follow outline the various menu choices. They are intended to provide quick access to the general contents of the menus, although do provide detail for some of the menu features. The first few menus are self-explanatory. Greater detail of the higher and important functions to which the menus lead will be given in subsequent chapters.

All choices can be accessed in the following manner:

1. **Via Keyboard**:

   a. Simplified Method: Type the **highlighted** letter of the selection (i.e., "**F**" for Files, "**B**" for Backup/Restore, "**C**" for Comm Ports, etc.);

   b. Cursor Method: Select the menu item, using the cursor keys, as indicated in the box at the bottom of the screen. When the menu item is highlighted, press **<ENTER>**.

<ESC> (the "Escape" key) will return the screen to the prior menu.

2. Via Mouse:

> **NOTE**
> The mouse cannot be used until it is configured correctly.

a. Move the mouse left-to-right or up-to-down to select a menu item. Once the desired menu item is highlighted, press the left button on the mouse to make the selection.

The right button on the mouse returns the screen to the prior menu.

3.3 OMNIVU MAIN MENU

Menu selections visible on the Main Menu are:

Files
Backup/Restore
Comm Ports
Diagnostics
Options
System

These selections access all of the features of the OmniVU. The following sections explain how to get to the various features via the First-Letter method.

> **NOTE**
> Any feature or selection with an asterisk (*) next to it is an item that is optional with the Xycom OIL-2 Option Card. Without the OIL-2 Option installed, the item will either not appear on the appropriate menu or the selection of that item will not be permitted.

3.4 THE FILES MENU

The Files Menu allows access to the following global submenus:

Edit
Files List
Delete
Rename
Copy
Print
Unit List

The features accessed by each of these submenus are:

Edit ——— Accessing the Edit item will allow the user to gain entry into some of the most often used and important features. They are:

Font Editor - By typing "**F**iles **E**dit **F**ont" from the Main Menu, the Font Editor is called up. From there the user can view and alter all font characters stored in memory in any one of four sizes (small, regular, double, and quad). Refer to Chapter 7.

Picture Editor - Type "**F**iles **E**dit **P**icture" from the Main Menu to access the Picture Editor. Any picture in memory can be altered by simply entering its file name. Refer to Chapter 6.

Keypad Definitions - By typing "**F**iles **E**dit **K**eypad" from the Main Menu, the user is allowed access to the file where all of the keys are defined and edited. Refer to Chapter 5.

Remote Commands - By typing "**F**iles **E**dit **R**emote Commands" from the Main Menu, the user is allowed access to the file where all of the remote commands are defined and edited. Refer to Chapter 4.

OIL-2 Programs - By typing "**F**iles **E**dit **O**IL-2 Programs" from the Main Menu, the user is allowed access to the file where all of the OIL-2 programs are accessed. This is an important feature with the Xycom 8110 Option Card installed.

File List —— Type "**F**iles **F**ile List" at the Main Menu. This submenu will allow the user to view all of the various types of files in memory (font, picture, keypad definitions, remote commands, and OIL-2 programs).

Delete ——— Type "**F**iles **D**elete" at the Main Menu. This submenu will allow the user to delete any of the various types of files in memory (font, picture, keypad definitions, remote commands, and OIL-2 programs).

Rename —— Type "**F**iles **R**ename" at the Main Menu. This submenu will allow the user to rename any of the various types of files in memory (font, picture, keypad definitions, remote commands, and OIL-2 programs).

Copy ——— Type "**F**iles **C**opy" at the Main Menu. This submenu will allow the user to copy any of the various types of files in memory (font, picture, keypad definitions, remote commands, and OIL-2 programs).

***Print** ——— Type "**F**iles **P**rint" at the Main Menu. This submenu will allow the user to print any OIL-2 programs or registers.

Unit List —— Type "**F**iles **U**nit List" at the Main Menu. This submenu will allow the user to access any of the memory units (A, B, or C).

3.5 THE BACKUP/RESTORE MENU

The Backup/Restore Menu allows access to any of the following global submenus:

Port Configuration
Backup
Restore

The features accessed by each of these submenus are:

Port Config. — Type "**B**ackup/Restore **P**ort Configuration" at the Main Menu. This submenu provides a means to configure the ports and the baud rate. Fixed values include: XMODEM protocol, 8 bits per character, no parity, and one stop bit.

Backup — Type "**B**ackup/Restore **B**ackup" at the Main Menu. This submenu facilitates to backing up of files on the installed memory units. Backups may include any or all of the various file types. The user is given a listing of file types to choose from, including an "all" entry.

Restore — Type "**B**ackup/Restore **R**estore" at the Main Menu. This submenu restores files previously backed up by the backup submenu to external memory sites. The user cannot selectively restore files, since the previous back-up listing will be used as the list of files to be restored.

CAUTION
Certain files will not be recognized when restored on a different memory unit than the default configuration. Refer to the section on Default Memory Units (3.9) for a listing of configurable and non-configurable file memory locations.

3.6 THE COMM PORTS MENU

The Comm Ports Menu allows access to any of the following global submenus:

Port **1**
Port **2**
Port **3**
Port **4**

The features accessed by each of these submenus are:

Ports 1, 2 — Type "Comm Ports Port **1** or **2**" from the Main Menu. This will provide access to the various submenus which set up physical and logical configurations for the two serial ports on the controller card.

***Ports 3, 4** — Type "Comm Ports Port **3** or **4**" from the Main Menu. This will provide access to the various submenus which set up physical and logical configurations for the two serial ports on the OIL-2 Option Card.

The comm ports on the OmniVU controller card are referred to as Serial Ports 1 and 2. These are used to communicate with a variety of peripheral devices. The user has an option of configuring the ports for any use. One is a dedicated RS-232 and the other can be configured to either RS-232 or RS-485, depending on need.

These configurations apply only during run-time. The comm ports and configuration used for Backup/Restore and the mouse are set up in separate menus and their configuration is used when in the menus.

Physical Configuration - the user is given the following aspects and options:

Baud Rate	**Bits/ Character**	**Stop Bits**	**Parity**	**Flow Control**	**Signal Control**	**Owner**
110	7	1	None	None	RS-422	None
150	8	2	Even	RTS/CTS	Standard	OIL
300			Odd	XON/XOFF	Half-Duplex	Terminal
600			Zero			
1200			One			
2400						
4800						
9600						
19200						

Logical Configuration - the user is given the following aspects and options:

Auto Linefeed	Auto Wrap	Scroll	Cursor	Local Echo	Display Control Characters
ON	ON	ON	ON	ON	ON
OFF	OFF	OFF	OFF	OFF	OFF

Window X Location	Window Y Location	Window X Size	Window Y Size
0-79	0-24	1-80	1-25

3.7 THE DIAGNOSTICS MENU

The Diagnostics Menu allows access to any of the following global submenus:

Individual Tests
Continuous Test

The features accessed by each of these submenus are:

Individual Tests — Type "Diagnostics Individual Tests" from the Main Menu. It provides access to any of twelve system diagnostics (processor, ROM, RAM, clock, comm ports, peripheral chips, printer port, speaker, relay, keypad keys, keyboard port, and video) one at a time.

Continuous Test — Type "Diagnostic Continuous Test" from the Main Menu. It provides access to the OmniVU continuous testing feature (tests processor, ROM, RAM, clock, comm ports, and peripheral chips). Press **<ESC>** twice to exit.

Total Test ——— Type "Diagnostic Total Test" from the Main Menu. It provides access all of the OmniVU continuous testing features (tests processor, ROM, RAM, clock, comm ports, and peripheral chips), one by one. Press **<ESC>** twice to exit.

During any of the tests, a prompt will appear indicating the test being performed (white letters on a blue background). The prompt is cleared if the test passes. If it fails, then another prompt (white letters on a red background) is displayed. The failure prompt will usually give some indication on the type of failure.

If a failure occurs, pressing the space bar will continue the test. Press **<ESC>** to terminate the test. The following are brief descriptions of each test. Names in **bold** indicate tests performed on power-up.

Processor: — a test of selected instructions and registers.

ROM: ——— tests BIOS, Extended BIOS, Data ROM, Terminal, Kernel, and OIL-2 ROMs, adding up certain bytes to get a checksum. In each case, a failure will indicate what the checksum is and should be.

RAM: ——— tests DRAM, VRAM, and CMOS RAM. A failure in each case will give the location of the error, the data expected, and the data found at that location.

Clock: ——— performs two tests on the real time clock: a running test and a RAM test. The error message will indicate which failed.

Comm Ports: - test involves serial loop back wiring to be run (see below) on all comm ports. OIL-2 ports are also tested if installed. A failure will indicate port and type of failure, indicating which pin is causing the error.

> NOTE
> All ports tested must be set for RS-232 operation.

CONNECTIONS:

DCD(1) - DTR(4) RXD(2) - TXD(3) RTS(7) - CTS(8)

Peripheral Chips: — tests the UIC, peripheral chips, DMA, UIC, PIT, and PIO and indicates any failures.

Printer Port: ——— test writes data to the printer. Will indicate either an "out of paper" or an "off line" condition. A passed test will print three lines of data.

Speaker: —— test pulses speaker, 0.5 sec. duration. Press a key to terminate.

Relay: ——— test pulses relay, 0.5 sec. duration. Press a key to terminate.

Keypad Keys: —— this test draws a picture of the keypads in white on a black screen. When a key is pressed it goes red, and when released goes blue. Press **<CTRL><BREAK>** to exit.

Keyboard Port — confirms the operation of the keyboard connectors. The user is prompted to unplug the keyboard from the front panel and plug it inot the rear port. Between each step a keyt is pressed to continue.

Video: ——— prompts the user with a menu that allows various patterns and colors on the screen to be displayed. The user must hit a key between each pattern/color or hit **<ESC>** to end the test.

3.8 THE OPTIONS MENU

The Options Menu allows access to any of the following global submenus:

Base
Terminal
Oil

The features accessed by each of these submenus are:

Base ——— Type "**O**ptions **B**ase" from the Main Menu. The configuration of the screen, mouse, printer, and keypad can be set from this menu.

Screen - the user has the option of selecting the text or graphics mode from this submenu. Press **<ESC>** to exit.

Mouse - the user can configure the port and mouse type from this submenu. The choices are Port 1 or Port 2 and PC Mouse or Microsoft.

Printer - the user is allowed to choose the selections for three options. They are printer type (either Proprinter or Epson RX-80), color representation (the colors black, blue, green, cyan, red, magenta, brown, and white can all be turned ON or OFF individually), and options (the print screen auto formfeed can be set to ON or OFF)

Keypad - the user can turn the keypad menu entry and keypad beep functions either to ON or OFF independently.

Terminal —— Type "**O**ptions **T**erminal" from the Main Menu. The user has access to the terminal options in this menu, including terminal ON/OFF, start-up definitions, and image files ON/OFF.

Terminal - the user can effectively turn the terminal functions to OFF, allowing the OIL-2 to run unimpeded.

Startup Definition - allows the user to designate any definition number as the start-up definition. The start-up definition is explained in detail in Chapter 8, under programming remote commands.

Image File - the user can turn the image file function either to ON or OFF in this submenu.

***OIL** ——— Type "**O**ptions **O**il" from the Main Menu. The OIL-2 options include setting a program block to autoexecute at start-up and setting the number of OIL-2 registers.

Autoexecute Program - the user can select any program to run at start-up. This gives relief from manually starting programs used at the beginning of each session.

OIL-2 Registers - the user can set the number of OIL-2 registers that will be used in any given operating session. The range is from 50 to 16300.

3.9 THE SYSTEM MENU

The Systems Menu allows access to any of the following global submenus:

Info
Password
Set Clock
Default Memory Units

The features accessed by each of these submenus are:

Info ——— Type "Systems **I**nfo" from the Main Menu. The system identification information will appear on the screen.

> **CAUTION**
> DO NOT FORGET THE SET PASSWORD! Bypassing the password involves a total RAM memory loss.

Password —— Type "Systems **P**assword" from the Main Menu. The user can set or delete a password in this menu. Setting a password will require it to be re-entered before gaining access to the menus.

Set Clock—— Type "Systems **S**et Clock" from the Main Menu. Six clock features can be set: year, month, day, hour, minute, second.

Default Memory Units —— Type "Systems **D**efault Memory Units" from the Main Menu. This provides the user access to the submenus which define the default memory units for the various user-configurable file types (font, picture, keypad definitions, remote commands, and OIL-2 programs). The file types listed under this menu are user configurable to any of the three memory units.

The memory units refer to the memory space on each installed circuit card. This means that memory unit A applies to the memory space on the OmniVU controller card. Memory unit B refers to the memory space available on an OIL-2 option card. Memory unit C refers to an additional memory expansion card, or some other expansion card containing memory.

CAUTION
Certain file types must always reside on their set memory units, or they will not be recognized by OmniVU. Table 3-1 lists the different file types and their default memory units.

Table 3-1 provides a brief listing of the default memory units that certain general file types are listed under. The file types each contain within themselves different files. The file types labelled **Not Configurable** must reside on the given default memory units. The OmniVU will only look for these files on those given units. This is a particular concern when restoring saved files. Should these non-configurable files be saved and then restored on a different memory unit, then the system would fail to operate properly.

All of the other file types can be moved. However, when OmniVU looks for a particular file type, it always begins on the default memory unit. If it is not on that unit it will go to the next one in the sequence (A, B, C), until all of the units are searched or the file is found.

When a memory unit is specified with a file name in a search, only that particular unit is examined.

Table 3-1 Memory Units

File Type	Default Memory Unit	Status
Fonts	A	Configurable
Pictures	A	Configurable
Keypad Definitions	A	Configurable
Base Configuration	A	**Not Configurable**
Remote Commands	A	Configurable
Terminal Configuration	A	**Not Configurable**
Special Terminal Def.s	A	Configurable
Terminal Image file	A	Configurable
OIL-2 Programs	B	Configurable
OIL-2 Registers	B	**Not Configurable**
OIL-2 Autoexecute	B	**Not Configurable**
OIL-2 Configuration	B	**Not Configurable**

It may be helpful for the user to remember that all base and terminal files default to memory unit A, and all OIL-2 files default to memory unit B. To access memory unit C, an additional card must be installed.

3.10 RESOURCE OWNERSHIP

Since OmniVU supports multi-tasking, resource ownership is an important point to be considered. Resources are hardware or software devices that can be used by code running on a system. Each of these tasks may want to use one or more of the resources available. Some resources can be easily shared (e.g., bell or screen). Other resources can be assigned to a task (e.g., comm port).

A few of these resources must be owned by a task at a specific time and specific duration. These are:

- keyboard
- screen cursor

These two resources are owned by the last task that tries to use them. Initially, the terminal owns both of them. It is always waiting for a key to be pressed. But when an OIL-2 program wants to read from the keyboard or move the cursor, then it takes over ownership until the desired task is complete. The resources then return to the previous owner.

If multiple OIL-2 tasks are all trying to use the same resource, the last task that asks for ownership gets it. When it is done, it gives the resource back to the task that it came from.

Chapter 4

REMOTE COMMANDS

4.1 INTRODUCTION

Remote commands are instructions issued from one system (which acts as a host or master) to another (acting as a recipient or slave). These commands allow OmniVU to be controlled by various external devices. The remote commands are also programmable. Commands can be added, deleted, or modified (refer to Chapter 8 for complete details).

OmniVU supports two emulations, ANSI and "other" (where "other" can be Hazeltine or any other non-ANSI system). The default system emulation is ANSI. This can be changed in the "Edit Files List Remote Commands" Menu.

The ANSI-type commands consist of a lead-in, parameters, and a command identifier. The parameters for some commands are optional. The usual format of an ANSI command is:

<ESC> [<pp> ; <pp> c

where:

<ESC> [represents the lead-in. **<ESC>** is the ASCII code for 1BH and **[** is the ASCII character for code 5BH. Other lead-ins include **<ESC> [=** and **<ESC> [?**.

The parameters are represented by **<pp>**. The number of parameters for a particular command depend on the function of the command. The ASCII character ; (code 3BH) is used to separate the parameters when more than one parameter is required.

The command identifier in the example is represented by **c**. It terminates the command and identifies a remote command. The identifier is usually an ASCII letter A-Z or a-z, although other characters are used.

For non-ANSI or other type emulation, the commands consist of a single lead-in character, followed by a command code and then zero or more parameters.

Both emulations also support commands that consist of a single ASCII control code.

In the case of either emulation, characters that are received that are not defined as a remote command (i.e., no lead-in character) are displayed on the screen. However, when a remote command lead-in is received, then the subsequent characters are interpreted as part of a remote command and compared to a listing of defined remote commands. If they all match, then that particular action is carried out. If all of the characters do not match a defined command, then the whole sequence is lost.

4.2 ANSI PARAMETERS

In the following ANSI commands, the <pp> represents an ANSI parameter. Parameters are one or more ASCII characters representing the digits in a decimal number. For example, a parameter value of 10 would be represented by two ASCII characters whose codes are 31H and 30H respectively. Parameters can be preceded by spaces but no other characters other than the characters in the range of 0 through 9 are allowed within a parameter.

The largest allowable parameter value is 65535.

Appendix C provides a table for converting decimal, hexadecimal, and ASCII.

4.3 DEFAULT ANSI REMOTE COMMANDS

The default ANSI remote commands for the ANSI emulation mode emulate a DEC VT 220. They appear in the following order:

4.3.1 ENQUIRE

Syntax: **<ENQ>**

This command sends the string "OMNIVU<CR>" out the assigned serial port (<CR> is a carriage return).

Example:

<ENQ>

4.3.2 BELL

Syntax: **<BEL>**

This command is used to sound the bell on the system to which it is sent.

Example:

<BEL>

4.3.3 ENTER APPLICATION MODE

Syntax: **<ESC> =**

This function will allow the user to enter the application mode. The application mode allows the user to redefine the codes sent out when certain keyboard keys are pressed.

Example:

<ESC> =

4.3.4 EXIT APPLICATION MODE

Syntax: **<ESC> >**

This function allows the user to exit the application mode.

Example:

<ESC> >

4.3.5 CURSOR COMMANDS

In some cursor movement commands, a parameter is requested. In this event, the operation will be performed the number of times specified by that parameter, or will occur at the given location.

Parameters are indicated by **pp**. Command syntax is listed to the left, and a description is listed to the right.

<BS>	Backs the cursor one space
<ESC> [<pp> A	Cursor Up
<ESC> [<pp> B	Cursor Down
<ESC> [<pp> C	Cursor Right
<ESC> [<pp> D	Cursor Left
<HT>	Moves cursor to the next tab stop
<LF>	Moves cursor to the next line (ignored if auto-linefeed is enabled)
<VT>	Same as <LF>
<FF>	Same as <LF>
<CR>	Carriage Return (If "auto-linefeed" is enabled, then the cursor moves to the beginning of the next line. Otherwise the cursor is moved to the beginning of the current line)
<ESC> D	Cursor Down w/Scrolling (if the cursor is on the bottom line and scrolling is enabled, then the screen is scrolled)
<ESC> M	Cursor Up w/Scrolling (if the cursor is on the top line and scrolling is enabled, then the screen is scrolled)
<ESC> E	Carriage Return w/Scrolling (if the cursor is on the bottom line and scrolling is enabled, then the screen is scrolled)
<ESC> [<pp> H	Move cursor to ....
	<ESC> [H — Cursor Home (1,1)
	<ESC> [<yy> ; <xx> H — Move cursor to coordinates <yy> (1-25) and <xx> (1-80)
<ESC> [<pp> f	Move cursor to ....
	<ESC> [f — Cursor Home (1,1)
	<ESC> [<yy> ; <xx> f — Move cursor to coordinates <yy> (1-25) and <xx> (1-80)
<ESC> 7	Saves cursor and attributes.
<ESC> 8	Restore saved cursor and attributes

Example:

<ESC> [5 B

This will move the cursor down five lines.

4.3.6 TAB OPERATIONS

Syntax: **<ESC> H**
<ESC> [<pp> g

where: **<pp>** = a numeric parameter specifying where the tab function is to be performed as follows:
0 = tab at cursor
3 = all tabs
H = set tab
g = clear tab

Set Tab

The following command will set the tab at the current cursor position:

<ESC> H

Clear Tab

The following commands will clear the tab as indicated:

<ESC> [g	Clear Tab at Cursor
<ESC> [0 g	Clear Tab at Cursor
<ESC> [3 g	Clear All Tabs

Example:

<ESC> [3 g

Sending this command will clear all of the set tabs on the current line.

4.3.7 ERASING OPERATIONS

Syntax: **<ESC> [<pp> "alpha"**
<ESC> [? <pp> "alpha"

where **<pp>** = a numeric parameter specifying the erase function to perform as follows:
0 = to end of line
1 = to beginning of line
2 = entire line
"alpha" = an alphanumeric parameter ending the command as follows:
X = erase characters on line
K = erase line
J = erase screen

The following commands will erase all characters in the direction specified

<ESC> [<pp> X	Erase Characters On Line
<ESC> [K	Erase To End Of Line
<ESC> [0 K	Erase To End Of Line
<ESC> [1 K	Erase To Beginning Of Line
<ESC> [2 K	Erase Entire Line
<ESC> [? K	Erase To End Of Line
<ESC> [? 0 K	Erase To End Of Line
<ESC> [? 1 K	Erase To Beginning Of Line
<ESC> [? 2 K	Erase Entire Line

The following commands will erase all elements on a screen in the direction specified

<ESC> [J	Erase To End Of Screen
<ESC> [0 J	Erase To End Of Screen
<ESC> [1 J	Erase To Beginning Of Screen
<ESC> [2 J	Erase Entire Screen
<ESC> [? J	Erase To End Of Screen
<ESC> [? 0 J	Erase To End Of Screen
<ESC> [? 1 J	Erase To Beginning Of Screen
<ESC> [? 2 J	Erase Entire Screen

Example:

<ESC> [? 2 J

Sending this will erase everything currently on the screen.

4.3.8 INSERT AND DELETE

Syntax: **<ESC> [<pp> "alpha"**

where **<pp>** = a numeric parameter specifying the number of lines or spaces to be inserted or deleted
"alpha" = an alphanumeric parameter ending the command as follows:
L = insert lines
M = delete lines
@ = insert spaces into a line
p = delete characters on a line

The commands are:

<ESC> [<pp> L	Insert Lines
<ESC> [<pp> M	Delete Lines
<ESC> [<pp> @	Insert Spaces Into Line
<ESC> [<pp> P	Delete Characters In Line

Example:

<ESC> [4 L

Sending this command will insert four lines at the line on which the cursor sits.

4.3.9 CONFIGURATION SETTINGS

Syntax: **<ESC> [<pp> "alpha"**
<ESC> [= <pp> "alpha"
<ESC> [? <pp> "alpha"

where **<pp>** = a numeric parameter specifying the function to be set or reset as follows:
1 = Cursor
2 = Keyboard
7 = Auto-wrap
20 = Auto-linefeed
25 = Cursor
"alpha" = an alphanumeric parameter which ends the command as follows:
h = Set configuration
l = Reset configuration

Set Configuration

The following commands are used to set configurations.

<ESC> [<pp> h Keyboard/Auto-linefeed: The following commands will set the keyboard and auto-linefeed features as indicated.

Lock Keyboard: **<ESC> [2 h**

Enable Auto-linefeed: **<ESC> [20 h**

<ESC> [= <pp> h Cursor/Scrolling: The following commands will set the cursor and scrolling features as indicated

Cursor ON **<ESC> [= 1 h**

Scrolling ON **<ESC> [= 2 h**

<ESC> [? <pp> h Cursor/Auto-wrap: The following commands will set the cursor and auto-wrap features as indicated.

Cursor ON **<ESC> [? 25 h**

Auto-wrap Enable **<ESC> [? 7 h**

Default set up in Comm Ports Logical Configuration Menu.

Reset Configuration

The following commands are used to reset configurations.

<ESC> [<pp> l Keyboard/Auto-Linefeed: The following commands will reset the keyboard and auto-linefeed features as indicated.

Unlock Keyboard **<ESC> [2 l**

Disable Auto-Linefeed **<ESC> [20 l**

<ESC> [= <pp> l Cursor/Scrolling: The following commands will reset the cursor and scrolling features as indicated

Cursor OFF **<ESC> [= 1 l**

Scrolling OFF **<ESC> [= 2 l**

<ESC> [? <pp> l Cursor/Auto-wrap: The following commands will reset the cursor and auto-wrap features as indicated

Cursor OFF **<ESC> [? 25 l**

Auto-wrap Disable **<ESC> [? 7 l**

Default set up in Comm Ports Logical Configuration Menu.

Example:

<ESC> [= 2 h

Sending this will turn the scrolling function to ON.

4.3.10 CHARACTER ATTRIBUTES

> **NOTE**
> • All of these commands work from the same base command, differing only in the parameter.
>
> • Some of the character attributes commands will only work in either the text or graphics mode. This is set in the "**O**ptions **B**ase **S**creen" menu.

The base command for Character Attributes is:

<ESC> [<pp> m

All of the attribute commands follow this format, with a particular parameter inserted. The following list includes only the parameters. The base command must be used with the parameter. Sending no parameter will turn the attributes to **OFF**.

Attributes (all)

0	Attributes OFF (highlight, underline, and blink all set to **OFF**. Sending no parameter will have the same effect) **(default)**

Highlight ON/OFF:

1	Highlight ON
22	Highlight OFF

Underline ON/OFF:

4	Underline ON (graphics modes only)
24	Underline OFF

Blinking ON/OFF:

5	Blink ON (text mode only)
25	Blink OFF

Reverse Video ON:

7	Reverse Video (swap foreground and background colors)
27	Reverse Video (swap foreground and background colors)

Strikeout ON/OFF:

8	Strikeout ON (graphics mode only)
28	Strikeout OFF

The fore and background color selections are made by invoking a palette register. There are 16 registers and 64 colors. Each palette register contains a code for a particular color. Registers 0-7 are fixed to certain colors, and cannot be changed. Registers 8-15 are configurable, and can be changed from their default settings. Selecting a color will use that register. To alter a color associated with a register, the value associated with that register must be changed.

Note that when a register is changed, all pictures containing that particular register will change color also. The colors are listed in the help menu.

Foreground Color Selection I:

30	Palette Reg. 0 Foreground (fixed Black)
31	Palette Reg. 1 Foreground (fixed Blue)
32	Palette Reg. 2 Foreground (fixed Green)
33	Palette Reg. 3 Foreground (fixed Cyan)
34	Palette Reg. 4 Foreground (fixed Red)
35	Palette Reg. 5 Foreground (fixed Magenta)
36	Palette Reg. 6 Foreground (fixed Yellow)
37	Palette Reg. 7 Foreground (fixed White) **(default)**

Background Color Selection I:

40	Palette Reg. 0 Background (fixed Black)
41	Palette Reg. 1 Background (fixed Blue) **(default)**
42	Palette Reg. 2 Background (fixed Green)
43	Palette Reg. 3 Background (fixed Cyan)
44	Palette Reg. 4 Background (fixed Red)
45	Palette Reg. 5 Background (fixed Magenta)
46	Palette Reg. 6 Background (fixed Yellow)
47	Palette Reg. 7 Background (fixed White)

Character Size Selection:

50	Regular Characters **(default)**	
51	Double Size Characters	
52	Quad Size Characters	(graphics mode only)
53	Half-high Characters	

Foreground Color Selection II:

60	Palette Reg. 8 Foreground (IBlack default)	
61	Palette Reg. 9 Foreground (IBlue default)	
62	Palette Reg. 10 Foreground (IGreen default)	
63	Palette Reg. 11 Foreground (ICyan default)	(graphics mode only)
64	Palette Reg. 12 Foreground (IRed default)	
65	Palette Reg. 13 Foreground (IMagenta default)	
66	Palette Reg. 14 Foreground (IYellow default)	
67	Palette Reg. 15 Foreground (IWhite default)	

Background Color Selection II:

70	Palette Reg. 8 Background (IBlack default)	
71	Palette Reg. 9 Background (IBlue default)	
72	Palette Reg. 10 Background (IGreen default)	
73	Palette Reg. 11 Background (ICyan default)	(graphics mode only)
74	Palette Reg. 12 Background (IRed default)	
75	Palette Reg. 13 Background (IMagenta default)	
76	Palette Reg. 14 Background (IYellow default)	
77	Palette Reg. 15 Background (IWhite default)	

Text Write Mode Selection:

80	"Replace" text write mode **(default)**	
81	"Overlay" text write mode	
82	"Invert" text write mode	
83	"Erase" text write mode	(graphics mode only)
84	"~Replace" text write mode	
85	"~Overlay" text write mode	
86	"~Invert" text write mode	
87	"~Erase" text write mode	

Example:

<ESC> [1 m

Sending this command will turn the highlighting function to ON.

4.3.11 REPORT FUNCTIONS

Syntax: **<ESC> [<pp> n**
<ESC> [<pp> c

where **<pp>** = a numeric parameter
n = an alphanumeric parameter which ends the Report Status statement
c = an alphanumeric parameter which ends the Report Options statement

<ESC> [<pp> n	Report Status	
	<ESC> [5 n	Returns <ESC> [0 n
	<ESC> [6 n	Return Cursor <ESC> [<yy> ; <xx> R
<ESC> [<pp> c	Report Options	
	<ESC> [c	Returns <ESC> [? 1 ; 0 c
	<ESC> [0 c	Returns <ESC> [? 1 ; 0 c

Example:

<ESC> [5 n

Sending this will cause the terminal to send out <ESC> [0 n.

4.3.12 SET DRAWING PARAMETER VALUE

Syntax: **<ESC> [1 ; <dpn> ; <dpv> p**

where **dpv** = drawing parameter value
dpn = drawing parameter number (1-6)
1 = pen size (pixels), dpv = (1,3,5, . . .)
2 = pen and fill pattern, dpv = (0-31)
3 = pen dash, dpv = (0-7)
0 = solid line
1 = -------- -------- --------
2 = ---- ---- ---- ---- ----
3 = -- -- -- -- -- -- --
4 = - - - - - - - - - - - - - -
5 = ---- -- -- ---- -- -- ----
6 = ---- - - ---- ---- - - ----
7 = ---- - - ---- - - ----
4 = pen write mode, dpv = (0-7) - defines how data is written on screen
0 = replace
1 = overlay
2 = invert
3 = erase
4 = ~replace
5 = ~overlay
6 = ~invert
7 = ~erase
5 = pen cap, dpv = (0-2) - defines how ends of lines are drawn
0 = flat (end of line cut at endpoints)
1 = round (end of line rounded past endpoint)
2 = square (end of line squared past endpoint)
6 = pen joint, dpv = (0-1) - defines the corners of rectangles
0 = rounded
1 = squared

The drawing parameters are used by the following commands: Set Pixel, Draw Line, Draw Rectangle, Draw Filled Rectangle, Draw Oval, Draw Filled Oval, Draw Arc, and Draw Filled Arc. The color is selected with the Set Attributes command.

This command has no effect if the screen is in text mode.

All parameters default to the lowest number values.

Example:

<ESC> [1 ; 5 ; 1 p

Sending this command will set the pen cap function (how the ends of a line appear) to rounded.

4.3.13 SET PIXEL

Syntax: **<ESC> [2 ; <y> ; <x> p**

where: x = x-axis pixel coordinate (0-639)
y = y-axis pixel coordinate (0-349)

This command will set the indicated pixel to the current foreground color. Also, this command has no effect if the screen is in text mode.

Example:

<ESC> [2 ; 270 ; 500 p

This command will turn a pixel on at coordinates 500,270 in the current foreground color.

4.3.14 DRAW LINE

Syntax: **<ESC> [3 ; <y1> ; <x1> ; <y2> ; <x2> p**

where **y1** = is a numeric pixel parameter (0 - 349) specifying the location of the starting y-axis coordinate
x1 = is a numeric pixel parameter (0 - 639) specifying the location of the starting x-axis pixel coordinate
y2 = is a numeric pixel parameter (0 - 349) specifying the location of the ending y-axis pixel coordinate
x2 = is a numeric pixel parameter (0 - 639) specifying the location of the ending x-axis pixel coordinate

This command will draw a line between the starting and ending coordinates. The x and y coordinates refer to pixels. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

This command has no effect if the screen is in text mode.

Example:

<ESC> [3 ; 0 ; 0 ; 50 ; 50 p

Sending this command will draw a line from the upper left corner of the screen (coordinates 0,0) to coordinates 50,50.

4.3.15 DRAW RECTANGLE

Syntax: **<ESC> [4 ; <y1> ; <x1> ; <y2> ; <x2> p**

where **y1** = is a numeric pixel parameter (0 - 349) specifying the location of the starting y-axis coordinate
x1 = is a numeric pixel parameter (0 - 639) specifying the location of the starting x-axis pixel coordinate
y2 = is a numeric pixel parameter (0 - 349) specifying the location of the ending y-axis pixel coordinate
x2 = is a numeric pixel parameter (0 - 639) specifying the location of the ending x-axis pixel coordinate

This command will draw a rectangle, using the starting and ending coordinates as the upper left and lower right corners respectively. The x and y coordinates refer to pixels. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

This command has no effect if the screen is in text mode.

Example:

<ESC> [4 ; 0 ; 0 ; 50 ; 50 p

Sending this command will draw a box with the upper left corner of the screen (coordinates 0,0) and coordinates 50,50 as the upper left and lower right corners, respectively.

4.3.16 DRAW FILLED RECTANGLE

Syntax: **<ESC> [5 ; <y1> ; <x1> ; <y2> ; <x2> p**

where **y1** = is a numeric pixel parameter (0 - 349) specifying the location of the starting y-axis coordinate
x1 = is a numeric pixel parameter (0 - 639) specifying the location of the starting x-axis pixel coordinate
y2 = is a numeric pixel parameter (0 - 349) specifying the location of the ending y-axis pixel coordinate
x2 = is a numeric pixel parameter (0 - 639) specifying the location of the ending x-axis pixel coordinate

This command will draw a filled rectangle, using the starting and ending coordinates as the upper left and lower right corners respectively. The x and y coordinates refer to pixels. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

This command has no effect if the screen is in text mode.

Example:

<ESC> [5 ; 0 ; 0 ; 50 ; 50 p

Sending this command will draw the same box as in the previous example, only filled with the current foreground color.

4.3.17 DRAW OVAL

Syntax: **<ESC> [6 ; <y> ; <x> ; <yr> ; <xr> p**

where **x** = x-axis coordinate of center (0-639)
y = y-axis coordinate of center (0-349)
xr = radius along the x-axis
yr = radius along the y-axis

This command will draw an oval in the dimensions specified by the chosen coordinate and radius variables. The x and y coordinates refer to pixels. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

This command has no effect if the screen is in text mode.

Example:

<ESC> [6 ; 175 ; 320 ; 50 ; 50 p

Sending this command will draw a circle with a center at coordinates 320,175 and with a radius of 50 pixels.

4.3.18 DRAW FILLED OVAL

Syntax: **<ESC> [7 ; <y> ; <x> ; <yr> ; <xr> p**

where **x** = x-axis coordinate of center (0-639)
y = y-axis coordinate of center (0-349)
xr = radius along the x-axis
yr = radius along the y-axis

This command will draw a filled oval in the dimensions specified by the chosen coordinate and radius variables. The x and y coordinates refer to pixels. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

This command has no effect if the screen is in text mode.

Example:

<ESC> [7 ; 175 ; 320 ; 50 ; 50 p

Sending this command will draw the same circle as in the previous example, only filled with the current foreground color.

4.3.19 DRAW ARC

Syntax: **<ESC> [8 ; <y> ; <x> ; <yr> ; <xr> ; <ba> ; <ea> p**

where **x** = x-axis coordinate of center (0-639)
y = y-axis coordinate of center (0-349)
xr = radius along the x-axis
yr = radius along the y-axis
ba = beginning angle (0-360)
ea = ending angle (0-360)

This command will draw an arc in the dimensions specified by the chosen coordinate, radius and angle variables. The x and y coordinates refer to pixels. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

0°is considered along the right x-axis (3 o'clock). The sweep is counter-clockwise.

This command has no effect if the screen is in text mode.

Example:

<ESC> [8 ; 175 ; 320 ; 50 ; 50 ; 0 ; 180 p

Sending this command will draw a 180°arc with a radial center at 320,175 with a radius of 50 pixels (the upper half of the circle drawn in DRAW CIRCLE).

4.3.20 DRAW FILLED ARC

Syntax: **<ESC> [9 ; <y> ; <x> ; <yr> ; <xr> ; <ba> ; <ea> p**

where **x** = x-axis coordinate of center (0-639)
y = y-axis coordinate of center (0-349)
xr = radius along the x-axis
yr = radius along the y-axis
ba = beginning angle (0-360)
ea = ending angle (0-360)

This command will draw a filled arc in the dimensions specified by the chosen coordinate, radius and angle variables. The x and y coordinates refer to pixels. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

0 °is considered along the right x-axis (3 o'clock). The sweep is counter-clockwise.

This command has no effect if the screen is in text mode.

Example:

<ESC> [9 ; 175 ; 320 ; 50 ; 50 ; 0 ; 180 p

Sending this command will draw the same 180 °arc as in the previous example, only filled with the current foreground color.

4.3.21 DRAW PICTURE

Syntax: **<ESC> [10 ; <un> ; <nm> ; <y> ; <x> ; <sc> ; p**

where **un** = memory unit to be accessed (0-2)
nm = file number within given memory unit (1-nnn)
x = x-axis coordinate of anchor point (0-639)
y = y-axis coordinate of anchor point (0-349)
sc = scale factor

This command will draw a picture on the screen, positioning it at the x and y coordinates in the specified scale. The scale can be specified in 16ths by using whole numbers from 1 to 16 (8/16, for instance would be half-sized; 32/16 double).

This command has no effect if the screen is in text mode.

The file number can be found by using the "Files Unit List" menu.

Example:

<ESC> [10 ; 1 ; 20 ; 175 ; 320 ; 8 p

Sending this command would retrieve the graphic in file 20, memory unit B and place it on the screen. Placement would be determined by the anchor point on the graphic, as it would be over the coordinate 320,175. The graphic would also be scaled to 50 percent.

4.3.22 WRITE TO OIL REGISTER

Syntax: **<ESC> [11 ; <rg> ; <lsw> ; <msw> p**

where **rg** = OIL register number (50-16300)
lsw = least significant word
msw = most significant word

This command will write 32 bits of information to an OIL register.

This command is valid only if the OIL-2 Option Card is installed in OmniVU.

Example:

<ESC> [11 ; 50 ; 12 ; 34 p

Sending this will write the two 16 bit words listed above to register 50.

4.3.23 READ OIL REGISTER

Syntax: **<ESC> [12 ; <rg> p**

where **rg** = OIL register number (50-16300)

> **NOTE**
> The "Read OIL Register" command causes the following to be sent from the comm port:
> **<ESC> [<lsw> ; <msw> R**
> where the value of the register is represented by <msw><lsw>.

This command is valid only if the OIL-2 Option Card is installed in OmniVU.

Example:

<ESC> [12 ; 80 p

Sending this command will read the contents of register 80 and subsequently send it out the comm port in two 16 bit words. If register 80 contained 2228236, then <ESC> [12 ; 34 R would be sent.

4.3.24 WRITE TO COMM BUFFER

Syntax: **<ESC> [13 ; <bt> p**

where **bt** = decimal value of data byte

This command will write a byte of information to the communications buffer.

This command is valid only if the OIL-2 Option Card is installed in OmniVU.

Example:

<ESC> [13 ; 65 p

Sending this command will write the byte 65 (41 Hex or ASCII "A") to the comm buffer.

4.3.25 WRITE TIME ON SCREEN

Syntax: **<ESC> [14 ; <tf> ; <yt> ; <xt> p**

where **tf** = time display format
0 = hh:mm:ss
yt = y-axis text coordinate (1-25)
xt = x-axis text coordinate (1-80)

This command will display the current time in one format as shown above. The time will be displayed at the coordinate chosen by the selected parameters.

Example:

<ESC> [14 ; 0 ; 15 ; 50 p

Sending this command will write the current hour, minute, and second on the screen at text coordinates 50,15.

4.3.26 WRITE DATE ON SCREEN

Syntax: **<ESC> [15 ; <df> ; <yt> ; <xt> p**

where **df** = date display format
0 = mmm dd, 19yy
1 = mm/dd/yy
yt = y-axis text coordinate (1-25)
xt = x-axis text coordinate (1-80)

This command will display the current date in one format as shown above. The date will be displayed at the coordinate chosen by the selected parameters.

Example:

<ESC> [15 ; 0 ; 15 ; 50 p

Sending this command will write the current month, day, and year on the screen at text coordinates 50,15.

4.3.27 CREATE WINDOW

Syntax: **<ESC> [16 ; <w> ; <yo> ; <xo> ; <ys> ; <xs> p**

where **w** = window number (1-3)
xs = window length on the x-axis in text coordinates
ys = window length on the y-axis in text coordinates
xo = x-axis coordinates of upper left corner in text coordinates
yo = y-axis coordinates of upper left corner in text coordinates

This command will create a window of the size specified in the parameters. Three windows are supported by remote commands. They are numbered from 1 to 3.

Window 1 is the default which is active when OmniVU is initially placed into the operating mode. The initial size and location of Window 1 are specified in the "Terminal Options" menu.

If a window has already been created for a specific number, that number cannot be used again in a window creation until the window assigned to the number has been deleted. Initially, Window-1 is created, and Windows 2 and 3 are in the deleted state.

To change the size and/or location of a window, the window must first be deleted then recreated with a new size and/or location. However, an active window cannot be deleted. It must first be deactivated (by activating another window) then recreated with a new size and/or location.

Also, each window can be treated as a separate screen. All coordinates are relative to the physical location of the window on the screen.

Example:

<ESC> [16 ; 2 ; 0 ; 0 ; 10 ; 10 p

Sending this command will create window 2 whose upper left corner is at the upper left corner of the screen (coordinates 0,0) and whose length and height are both 10 text units.

4.3.28 DELETE WINDOW

Syntax: **<ESC> [17 ; <w> p**

where **w** = window number (1-3)

This command is used to delete a window that has been created. If a window is currently active, this command will not delete it. To delete a currently active window, a second window must be activated. This will make the first window inactive, therefore able to be deleted.

Example:

<ESC> [17 ; 2 p

Sending this command will destroy window 2.

4.3.29 ACTIVATE WINDOW

Syntax: **<ESC> [18 ; <w> p**

where **w** = window number (1-3)

This command is used to activate a previously created window. A deleted window cannot be activated. Once a window is activated, the previous window becomes inactive.

Example:

<ESC> [18 ; 2 p

Sending this command will activate window 2.

4.4 DEC VT220 COMMANDS NOT SUPPORTED

Because of hardware and software constraints, not all of the DEC VT220 commands are supported by the current version of the OmniVU Terminal.

The following are the commands:

<SO>	Invokes G1 character set into GL
<SI>	Invokes G0 character set into GL
<ESC> ~	Invokes G1 character set into GR
<ESC> n	Invokes G2 character set into GL
<ESC> {	Invokes G2 character set into GR
<ESC> o	Invokes G3 character set into GL
<ESC> |	Invokes G3 character set into GR
<ESC> N	Temporarily invokes G2 character set
<ESC> O	Temporarily invokes G3 character set
<ESC> (<f>	Designates <f> as G0 character set
<ESC>) <f>	Designates <f> as G1 character set
<ESC> * <f>	Designates <f> as G2 character set
<ESC> + <f>	Designates <f> as G3 character set
<ESC> [? 42 h	National character set
<ESC> [? 42 l	Multinational character set
<ESC> P	Device control string
**<ESC> **	String terminator
<ESC> [61 " p	Set terminal for compatibility level 1
<ESC> [62 " p	Set terminal for compatibility level 2
<ESC> [62 ; 0 " p	Set terminal for compatibility level 2
<ESC> [62 ; 2 " p	Set terminal for compatibility level 2
<ESC> [62 ; 1 " p	Set terminal for compatibility level 2
<ESC> sp F	Select 7-bit control transmission
<ESC> sp G	Select 8-bit control transmission
<ESC> [4 h	Go into insert mode
<ESC> [4 l	Go into replace mode
<ESC> [12 h	Local echo ON
<ESC> [12 l	Local echo OFF
<ESC> [? 1 h	Application mode for cursor keys
<ESC> [? 1 l	ANSI mode for cursor keys
<ESC> [? 2 l	Go into VT52 mode
<ESC> [? 3 h	Go into 132 column mode
<ESC> [? 3 l	Go into 80 column mode
<ESC> [? 4 h	Smooth scrolling
<ESC> [? 4 l	Jump scrolling
<ESC> [? 5 h	Reverse video screen mode
<ESC> [? 5 l	Normal video screen mode

VT220 Commands not supported cont.

<ESC> [? 6 h	Cursor relative to scrolling region
<ESC> [? 6 l	Cursor relative to screen
<ESC> [? 8 h	Auto repeat ON
<ESC> [? 8 l	Auto repeat OFF
<ESC> [? 18 h	Print Form Feed ON
<ESC> [? 18 l	Print Form Feed OFF
<ESC> [? 19 h	Print extent - full screen
<ESC> [? 19 l	Print extent - scrolling region
<ESC> [5 i	Auto Print Mode ON
<ESC> [4 i	Auto Print Mode OFF
<ESC> [? 1 i	Print line containing cursor
<ESC> [i	Print screen
<ESC> [? 15 n	Return Printer Status
<ESC> [0 " q	Designates characters as erasable
<ESC> [1 " q	Designates characters as not erasable
<ESC> [2 " q	Designates characters as erasable
<ESC> # 3	Make line top half of double high characters
<ESC> # 4	Make line bottom half of double high characters
<ESC> # 5	Make characters for line single-width
<ESC> # 6	Make characters for line double-width
<ESC> [> c	Secondary report options
<ESC> [? 25 n	Return user defined key status
<ESC> [? 26 n	Return keyboard language
<ESC> Z	Return identification
<ESC> [! p	Soft reset
<ESC> c	Hard reset
<ESC> [4 ; <tt> y	Invoke tests
<ESC> # 8	Display alignment pattern

4.5 NON-ANSI PARAMETERS

In the following non-ANSI commands, the parameters are represented by **<pp>**. These parameters consist of a single byte or character of data. The value is represented by a hexadecimal number (e.g., a value of 10 would be represented by the data byte 0AH).

Appendix C provides a table for converting decimal, hexadecimal, and ASCII.

4.6 DEFAULT NON-ANSI REMOTE COMMANDS

4.6.1 BELL

Syntax: **<BEL>**

This command is used to sound the bell.

4.6.2 CURSOR MOVEMENTS

In some cursor movement commands, a parameter is requested. In this case, the operation will be performed the number of times specified by that parameter.

<BS>	Backs the cursor one space
<LF>	Moves cursor to the next line
<TAB>	Moves cursor to the next tab stop
<CR>	Carriage Return (If "auto linefeed" is enabled, then the cursor moves to the beginning of the next line. Otherwise the cursor is moved to the beginning of the current line)
~ <DC2>	Moves cursor to the home position (0,0)
~ <DC1> <pp>	Move cursor to coordinates <yy> (0-24) and <xx> (0-79)

4.7 PROGRAMMABLE REMOTE COMMANDS

Each remote command has a definition associated with it. These definitions contain one or more primitives that determine the action taken when that remote command is invoked. The remote command structure in the 8320 is such that the remote commands can be programmed to perform any function. Commands can be added, deleted, or altered from the default settings.

All of the instructions on how to program the remote commands can be found in Chapter 8. It is devoted entirely to programming for special functions. The user should gain a complete understanding of the terminal and its functions before attempting to reprogram any of the remote commands. Once reprogrammed, the default functions are ignored.

CAUTION
Do not attempt to alter any of the default definitions before understanding completely the primitives of the 8320.

Chapter 5

OPERATOR KEYS

5.1 INTRODUCTION

There are both function and regular keypads located on the face of the OmniVU. The terminal associates a definition with each of these keys. This definition determines the action taken when a key is pressed. The default definitions for each of these keys is the emulation of their counterparts on a regular keyboard, but they can be programmed to suit alternative functions (refer to Chapter 8).

The OmniVU can also use an external PC/AT type keyboard.

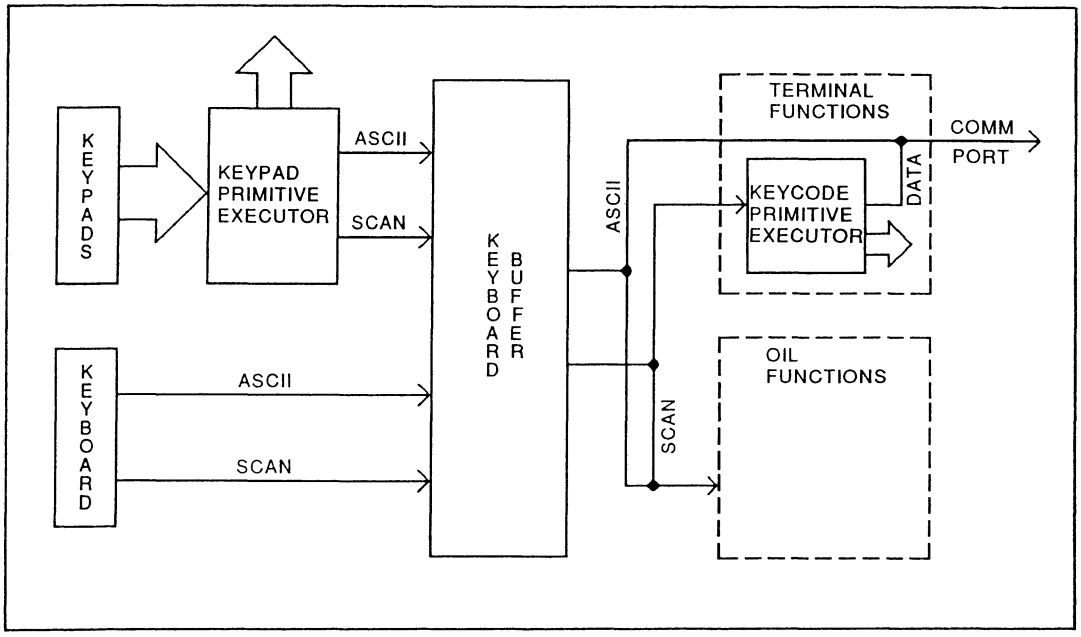


Figure 5-1 Keyboard Data Flow Diagram

When a key on one of the keypads or the external keyboard is pressed, that signal generated is placed into the keyboard buffer. The terminal differentiates these signals into two general groups: regular and special keys. All alpha-numeric keys are classified as regular keys and take the form of the appropriate ASCII code. All of the function and the keys that control cursor movement (including Home, End, PgUp, and PgDn) are called special keys and take the form of scan codes.

The Keyboard Task (refer to Section 8.4.4) reads and processes the information from the keyboard buffer. When it reads an ASCII code from the buffer it passes that code on and out the serial port. When it reads a scan code, it checks to see if it is one of the open codes associated with the special keys. If it is, then it executes that definition.

These definitions can be programmed to suit a particular emulation or perform any function of which the terminal is capable. The default definitions provide for a code sequence to be sent out the serial port.

There are two sets of default definitions: ANSI and non-ANSI. The ANSI set is used when the terminal is set up to emulate ANSI remote commands, and the non-ANSI when it is set up for non-ANSI.

All definitions consist of one or more "primitives." These primitives are described in detail in Chapter 8. Each of the definitions can be edited. The Primitive Editor allows the user to add or delete primitives, as well as change the parameter(s) for each primitive.

5.2 DEFAULT KEYPAD DEFINITIONS

There are 53 total keys between the two keypads on the OmniVU front panel. Table 5-1 shows the codes sent to the keyboard buffer by the default definitions for each of the keys.

Table 5-1 Default Definitions

Key	Code Sent To Buffer	Key	Code Sent To Buffer
ESC	<ESC>	Left Arrow	Scan Code 75
+	"+"	Down Arrow	Scan Code 80
DEL	<DEL>	Right Arrow	Scan Code 77
*	"*"	ENTER	<CR>
-	"-"	F1	Scan Code 59
#	"#"	F2	Scan Code 60
Space	" "	F3	Scan Code 61
BS	8	F4	Scan Code 62
/	"/"	F5	Scan Code 63
=	"="	F6	Scan Code 64
TAB	9	F7	Scan Code 65
INS	Scan Code 82	F8	Scan Code 66
7	"7"	F9	Scan Code 67
8	"8"	F10	Scan Code 68
9	"9"	F11	"Q"
4	"4"	F12	"R"
5	"5"	F13	"S"
6	"6"	F14	"T"
1	"1"	F15	"U"
2	"2"	F16	"V"
3	"3"	F17	"W"
0	"0"	F18	"X"
Up Arrow	Scan Code 72	F19	"Y"
.	"."	F20	"Z"
HOME	Scan Code 71		
PAUSE	"P"		
PGUP	Scan Code 73		
PGDN	Scan Code 81		
END	Scan Code 79		

5.3 ANSI MODE DEFAULT SPECIAL KEY DEFINITIONS

The default definitions for each of the special keys cause one or more ASCII codes to be sent out of the serial port. The special keys, the scan code that identifies them, and the sequence sent out the serial port are listed below.

Table 5-2 ASCII Scan Codes

Scan Code	Key	ASCII Code (sent out of comm port)
59	F1	"G"
60	F2	"H"
61	F3	"I"
62	F4	"J"
63	F5	"K"
64	F6	"L"
65	F7	"M"
66	F8	"N"
67	F9	"O"
68	F10	"P"
71	Home	<ESC> [H
72	Up Arrow	<ESC> [A
73	Page Up	<ESC> [25 A
74	-	"-"
75	Left Arrow	<ESC> [D
76	5	"5"
77	Right Arrow	<ESC> [C
78	+	"+"
79	End	<ESC> [25 ; 1H
80	Down Arrow	<ESC> [B
81	Page Down	<ESC> [25 B
82	Insert	<ESC> [1 @
83	Delete	<ESC> [1 P

5.4 NON-ANSI MODE DEFAULT SPECIAL KEY DEFINITIONS

The default definitions for each of the special keys cause one or more ASCII codes to be sent out of the serial port. The special keys, the scan code that identifies them, and the sequence sent out the serial port are listed below.

Table 5-3 Non-ASCII Scan Codes

Scan Code	Key	ASCII Code (sent out of comm port)
59	F1	nothing
60	F2	
61	F3	
62	F4	
63	F5	
64	F6	
65	F7	
66	F8	
67	F9	
68	F10	
71	Home	
72	Up Arrow	
73	Page Up	
74	-	
75	Left Arrow	
76	5	
77	Right Arrow	
78	+	
79	End	
80	Down Arrow	
81	Page Down	
82	Insert	
83	Delete	nothing

5.5 PROGRAMMABLE KEYPAD AND SPECIAL KEY DEFINITIONS

All of the instructions on how to program the keypad and special keys can be found in Chapter 8. It is devoted entirely to programming for special functions. The user should gain a complete understanding of the terminal and its functions before attempting to reprogram any of the keys. Once reprogrammed, the default functions are lost.

> **CAUTION**
> Do not attempt to alter any of the default definitions before completely understanding the primitives of the 8320.

When the definitions for the keypad and/or special keys are edited, files are created on memory unit A to store the new definitions. To return to the default settings, the files can be renamed or deleted. They can also be backed up and restored to the default or another memory unit for later use.

All of the created definitions will go into one of three files, depending on what keys were edited. The filenames are assigned as follows:

rmakeys - definitions for the ANSI mode special keys
rmpkeys - definitions for the non-ANSI mode special keys
keydefs - definitions for the keypad keys

These files will appear on the system only if the default definitions are edited.

Chapter 6

PICTURE EDITOR

6.1 INTRODUCTION

The OmniVU 8320 Picture Editor is a versatile screen editor that allows graphics and text to be created and altered for use in on-screen or printed displays. The primary features of the editor are:

- 5 open shape templates
- 4 filled shape templates
- A dashed-line template
- Text manipulation
- 4 line-width selections
- 16 on-screen color choices (8 foreground & 8 background)
- 18 edit options (such as "Add Picture," "Set Background," etc.)
- 32 pattern fill selections
- A large, framed, picture editing field
- 2 scrolling bars (for preparation of full-screen displays)

6.2 USING THE MOUSE

The Picture Editor requires the installation of a serial mouse which conforms to either the PC Mouse or Microsoft Mouse standard. Refer to Section 2.5.1 to install the serial mouse.

The mouse is needed with the OmniVU Picture Editor for two primary reasons:

- Making selections from the icon menu

  and

- Creating and manipulating patterns within the Picture Editor editing field

> **NOTE**
> The left button on the mouse will operate only within the editing field. Conversely, the right button will operate only on the icon menus (which fills the left, right and bottom of the screen).

Selections are made by moving the on-screen arrow to the desired selection in one of the icon menus and pressing the right button on the mouse to activate the selection. A thin white line will encircle the selected option.

Any corrections or edits on the editing field itself will operate in much the same manner, only the left button on the mouse is used to activate a feature or control an action.

6.3 USING THE PICTURE EDITOR

Enter the Picture Editor by selecting "**F**iles **E**dit **P**ictures" from the Main Menu. Once the selection is made, the following prompt will appear:

Edit Picture:

This is a prompt for a filename. If a file exists, type the file name and press **<Enter>**. If the file is to be created, type the new file name and **<Enter>**. The Picture Editor will then appear on the screen. Figure 6-1 shows what the Picture Editor screen will look like.

> **NOTE**
> Within the editing field, the mouse cursor will be a crosshatch (+). When the mouse cursor leaves the editing field and enters the surrounding icon menus, it will become an arrow.

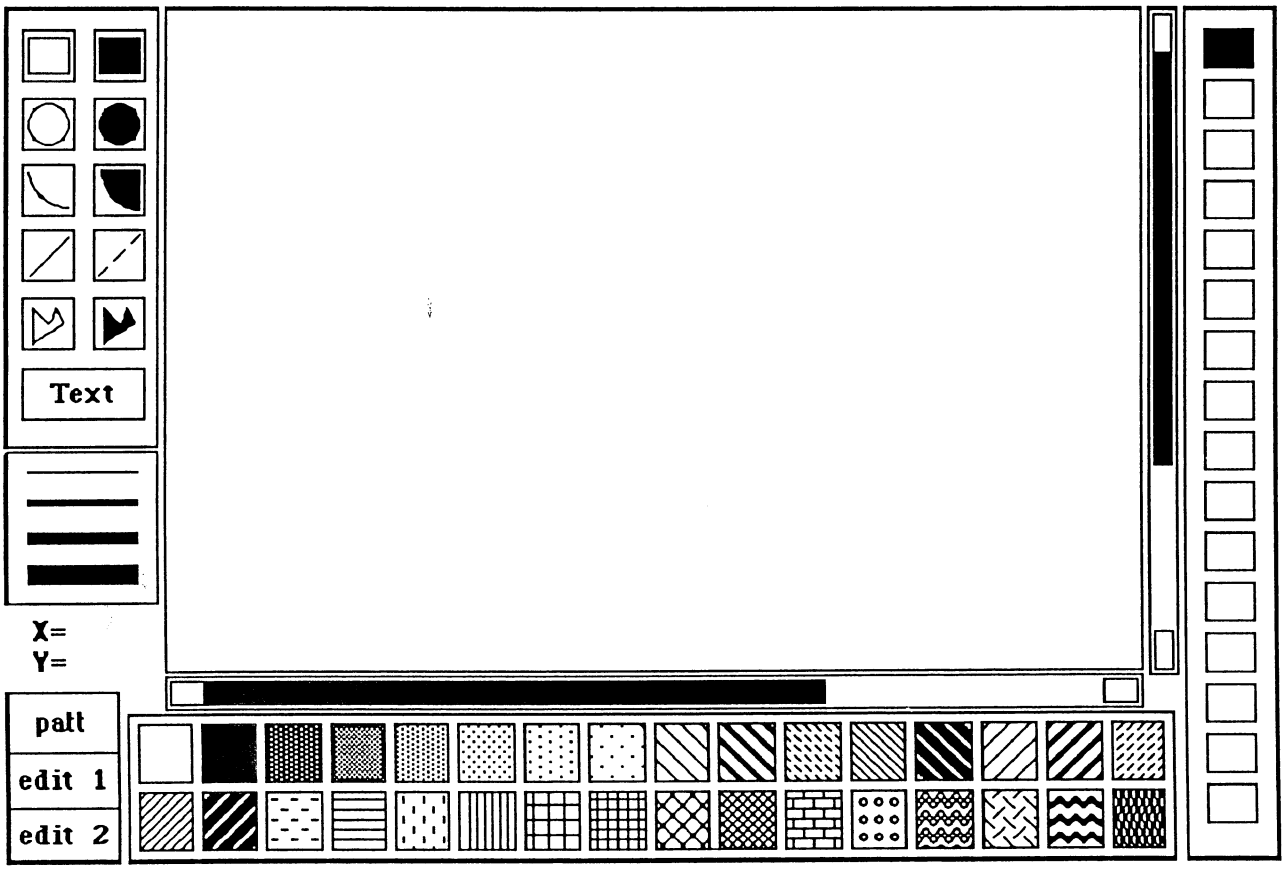


Figure 6-1 Picture Editor Screen

Figure 6-1 shows what the Picture Editor screen should look like. To construct or edit a picture, the user must be familiar with the basic features displayed around the editing field. These are various functions that allow the user to manipulate pictures, or elements within a picture.

6.4 DRAWING WITHIN THE PICTURE EDITOR

All shapes drawn in the Picture Editor Editing Field are created with a graphic "rubberband."

To use the draw features (except polygons) on the picture editor:

1. Select the desired shape and colors (for background and foreground).
2. Move the select arrow into the editing field. The arrow will become a crosshatch.
3. Move the crosshatch to the spot in the field where the shape is to be drawn.
4. Press the left button on the mouse and move the mouse in any diagonal direction, away from the starting point. (The evolving shape should be evident.)
5. When the shape is as desired, release the mouse button.

When drawing a polygon, follow steps 1 thru 4, then:

1. Move the mouse to the first angle, then release and depress the button (repeat this step for each new angle until up to 20 sides are drawn).
2. After the desired shape is made, press and release the button without moving the mouse (a single mouse click terminates the action).

6.5 PICTURE EDITOR FEATURES

There are many features available to the user once in the Picture Editor Mode. Most are methods of creating specific graphic images to suit the user's needs. The main features available in this edition of the OmniVU picture editor are:

Draw Rectangle
Draw Circle
Draw Arc
Draw Line
Draw Polygon
Write Text
Scroll Bars
Line Width Selection
Color Selection
Pattern Selection

There are also two Edit Features in the Picture Editor. These contain special settings and functions that make using the Picture Editor easier. They are:

Edit 1 **Edit 2**

Both contain their own set of functions, described in Sections 6.6 and 6.7.

USER NOTE: —— The best way to learn to use the OmniVU graphics features is by experimentation. Enter the Picture Editor and draw various shapes. Change colors and sizes, and attempt to use all of the features described in this chapter. When finished, hit **<ESC>** to exit without saving any changes.

Certain aspects of the Picture Editor artwork in this manual have been exaggerated. For example, in the case where a shape selection is made, the frame around it is larger than would be seen normally. Also, in many of the illustrations there is more than one cursor shown (one for an open object and one for its filled counterpart, or four in the text example). There will never be more than one cursor on the screen at any time.

6.5.1 **Draw Rectangle**

There are two draw selections for rectangles on the left-side graphics menu in the Picture Editor: the open and the filled rectangle.

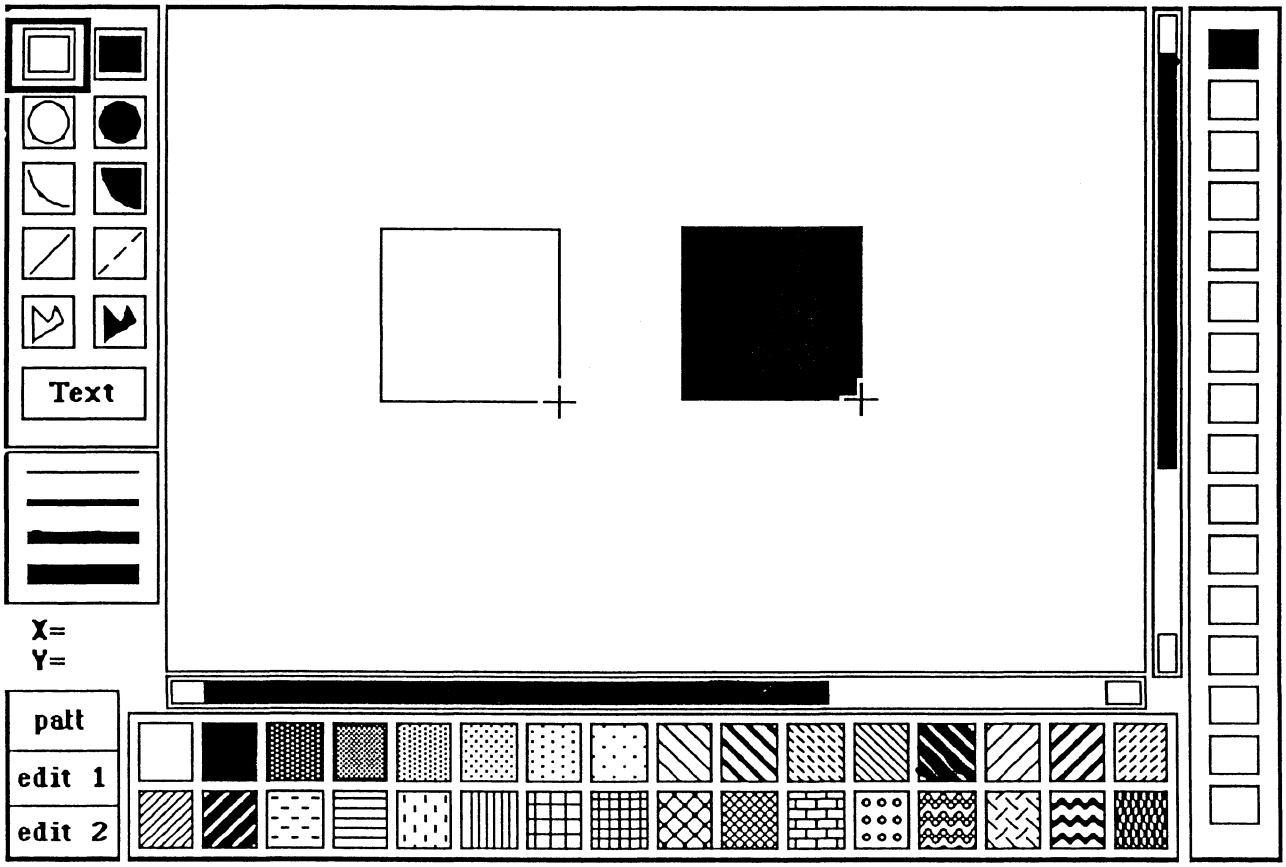


Figure 6-2 Rectangles

Figure 6-2 shows two examples of drawn rectangles. If the open rectangle is chosen, a rectangle in the chosen pattern (including the current line width and line color) will be drawn. If the filled rectangle is chosen, a rectangle in the chosen pattern (including current line width, fill pattern and color) will be drawn. The current line width attribute is only used in filled shapes if the outline function is also selected.

The open rectangle is shown as currently selected on the left edge. The "outline" edit option does not affect open shapes.

6.5.2 **Draw Circle**

There are two draw selections for circles on the left-side graphic menu in the Picture Editor: the open circle and the filled circle.

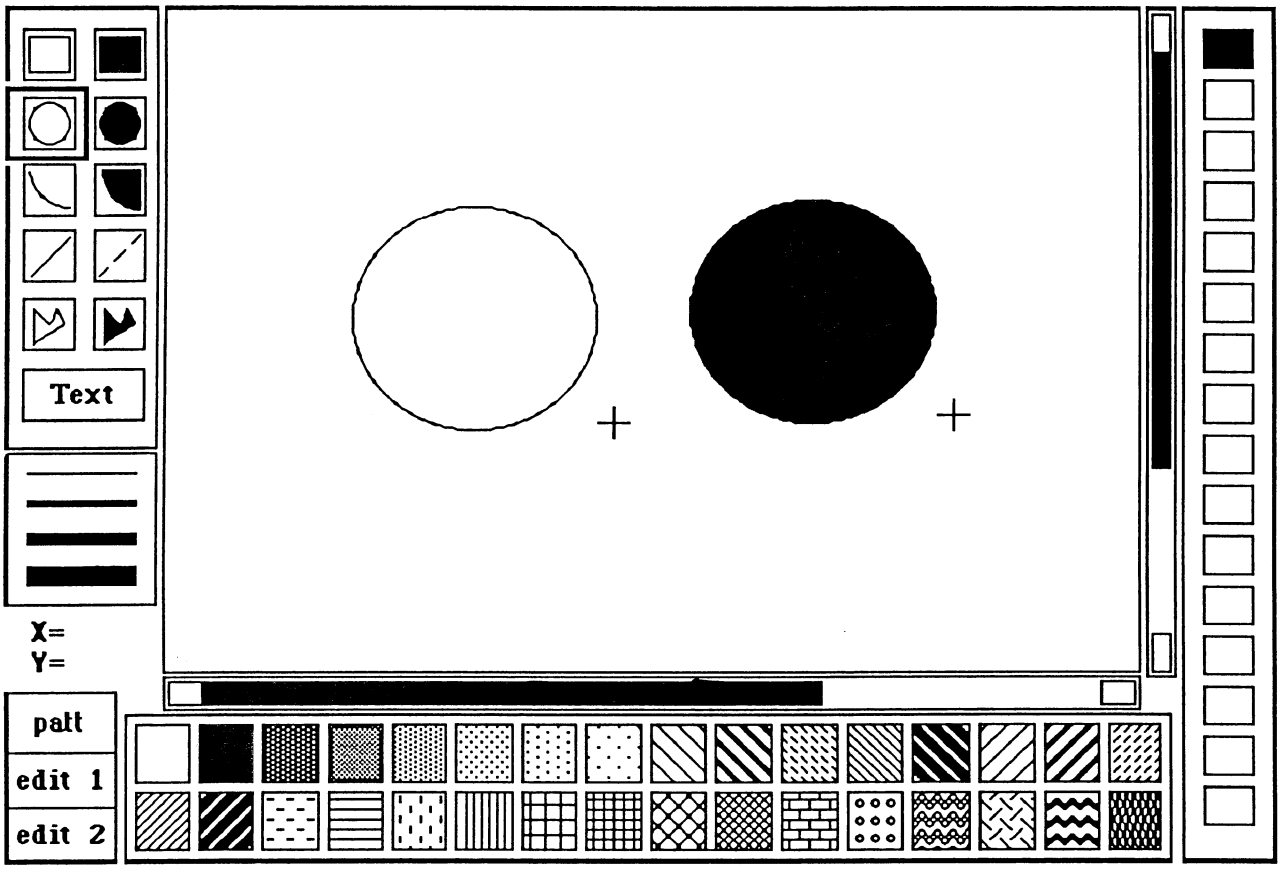


Figure 6-3 Circles

Figure 6-3 shows two examples of drawn circles. If the open circle is chosen, a circle in the chosen pattern (including the current line width and line color) will be drawn. If the filled circle is chosen, a circle in the chosen pattern (including current line width, fill pattern and color) will be drawn. The current line width attribute is only used in filled shapes if the outline function is also selected.

The open circle is shown as currently selected on the left edge. The "outline" edit option does not affect open shapes.

6.5.3 **Draw Arc**

There are two draw selections for arcs on the left-side graphic menu in the Picture Editor: the open arc and the filled arc.

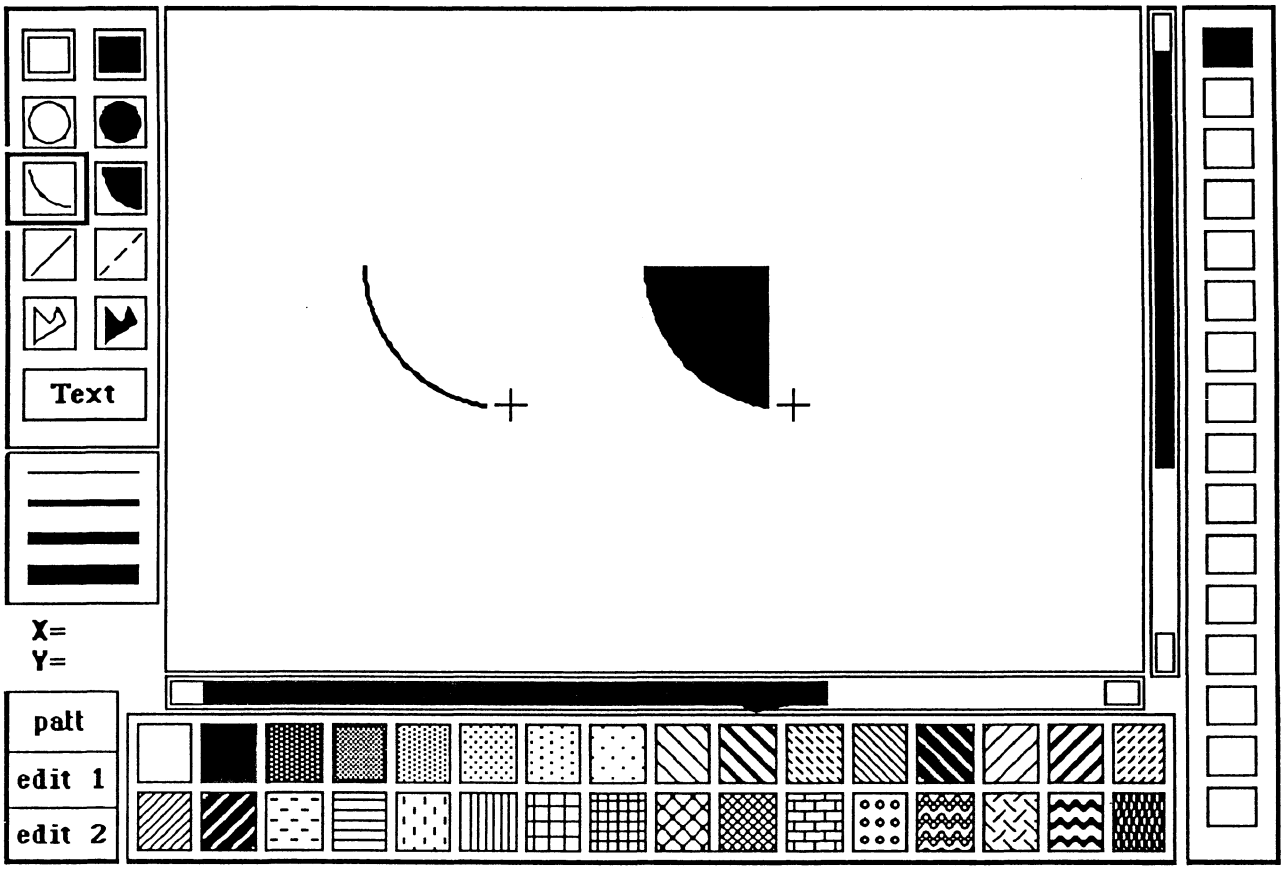


Figure 6-4 Arcs

Figure 6-4 shows two examples of drawn arcs. If the open arc is chosen, an arc in the chosen pattern (including the current line width and line color) will be drawn. If the filled arc is chosen, an arc in the chosen pattern (including current line width, fill pattern and color) will be drawn. The current line width attribute is only used in filled shapes if the outline function is also selected.

The open arc is shown as currently selected on the left edge. The "outline" edit option does not affect open shapes.

6.5.4 **Draw Line**

There are two draw selections for lines on the left-side graphic menu in the Picture Editor: the regular line and the dashed line.

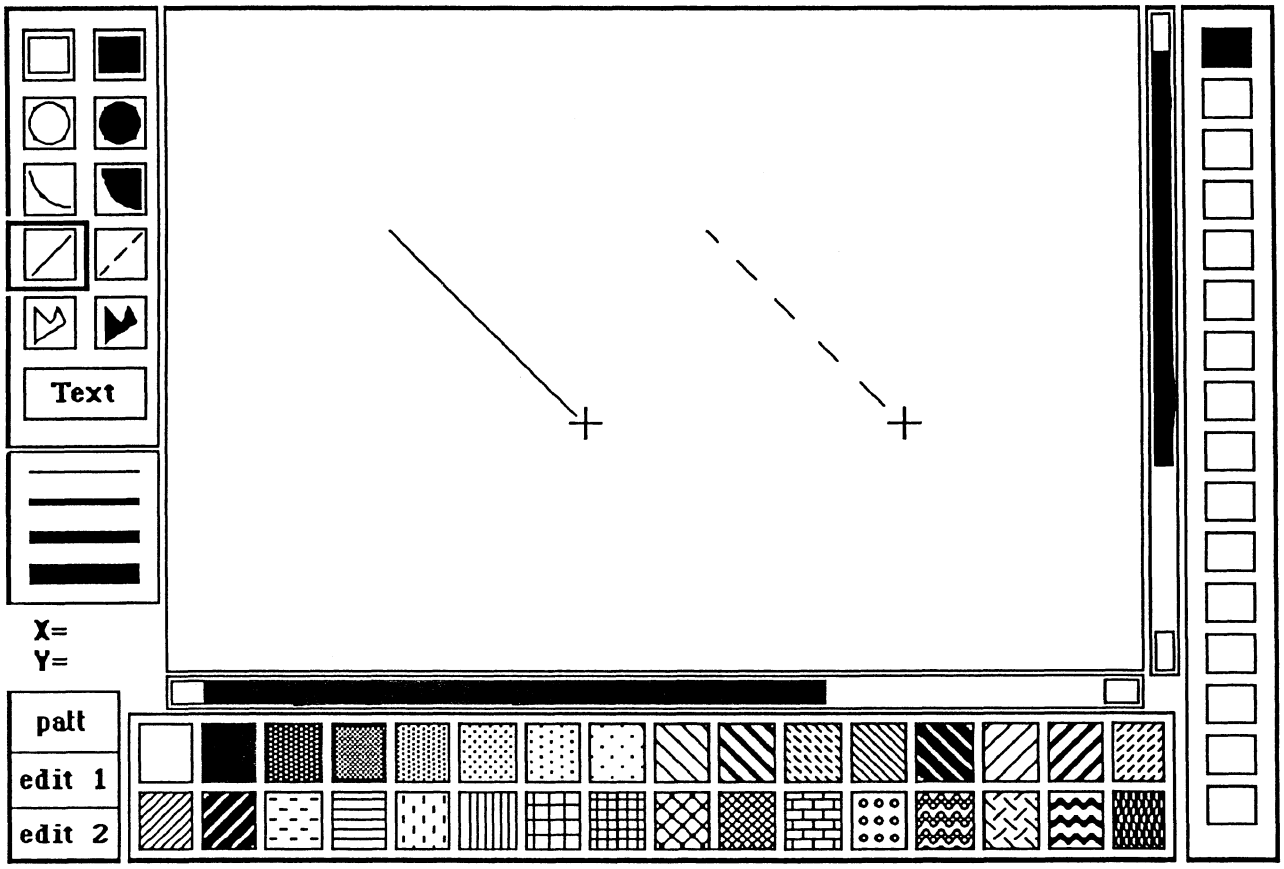


Figure 6-5 Lines

Figure 6-5 shows two examples of drawn lines. If the solid line is chosen, a line in the chosen width, pattern, and color will be drawn. If the dashed line is chosen, a dashed line in the current line width, pattern, and color will be drawn.

The solid line is shown as currently selected on the left edge. The outline option does not affect either the solid or dashed line.

6.5.5 **Draw Polygon**

There are two draw selections for polygons (up to 20 sides) on the left-side graphic menu in the Picture Editor: the open polygon and the filled polygon.

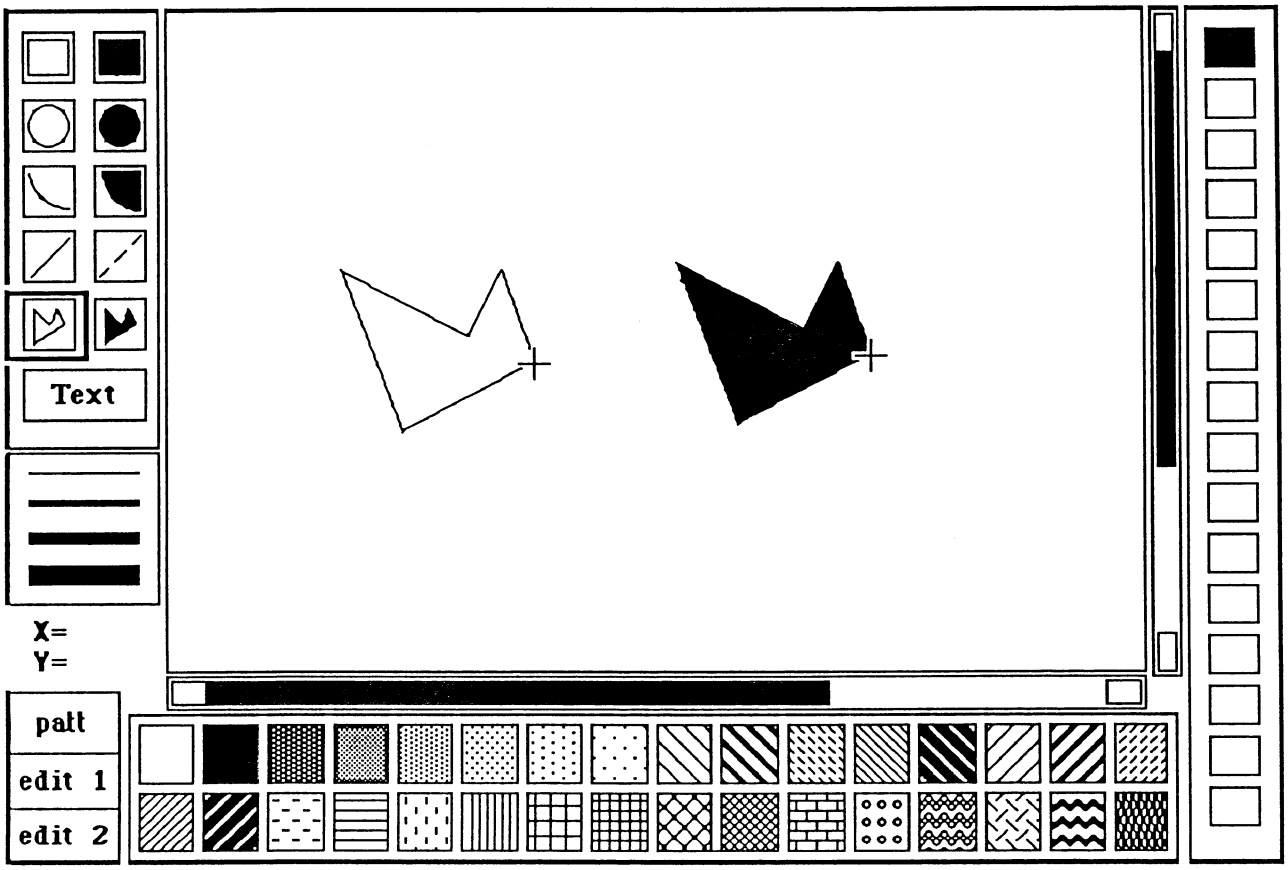


Figure 6-6 Polygons

Figure 6-6 shows two examples of drawn polygons. If the open polygon is chosen, a polygon in the chosen pattern (including the current line width and line color) will be drawn. If the filled polygon is chosen, a polygon in the chosen pattern (including current line width, fill pattern and color) will be drawn. The current line width attribute is only used in filled shapes if the outline function is also selected.

The open polygon is shown as currently selected on the left edge. The "outline" edit option does not affect open shapes.

6.5.6 Write Text

The Picture Editor allows text to be mixed with graphics by selecting the "Text" box on the left-side graphic menu. Once text is selected, the "Edit 1" menu allows the selection of one of four font sizes (Small, Regular, Double, and Quad Size).

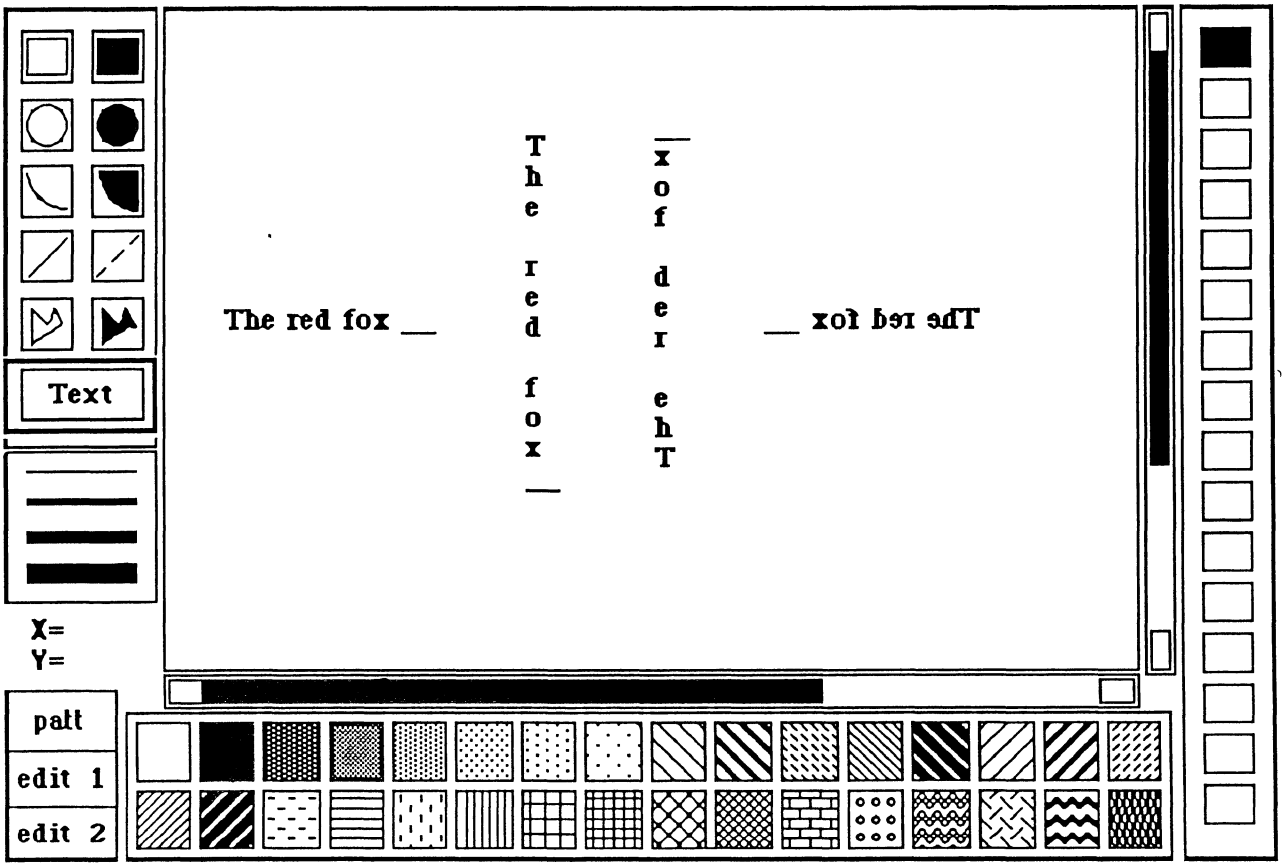


Figure 6-7 Text

Figure 6-7 shows examples of text written in all four directions.

To write text, position the crosshatch into the editing field where the text is to begin. Draw a line in the editing field in one of four directions to be written (up, down, left, or right). Type in the text.

> **NOTE**
> The **<Enter>** key will move the cursor to the next line. Pressing the left mouse button will deactivate the Write Text option in the editing field.

6.5.7 Scroll Bars

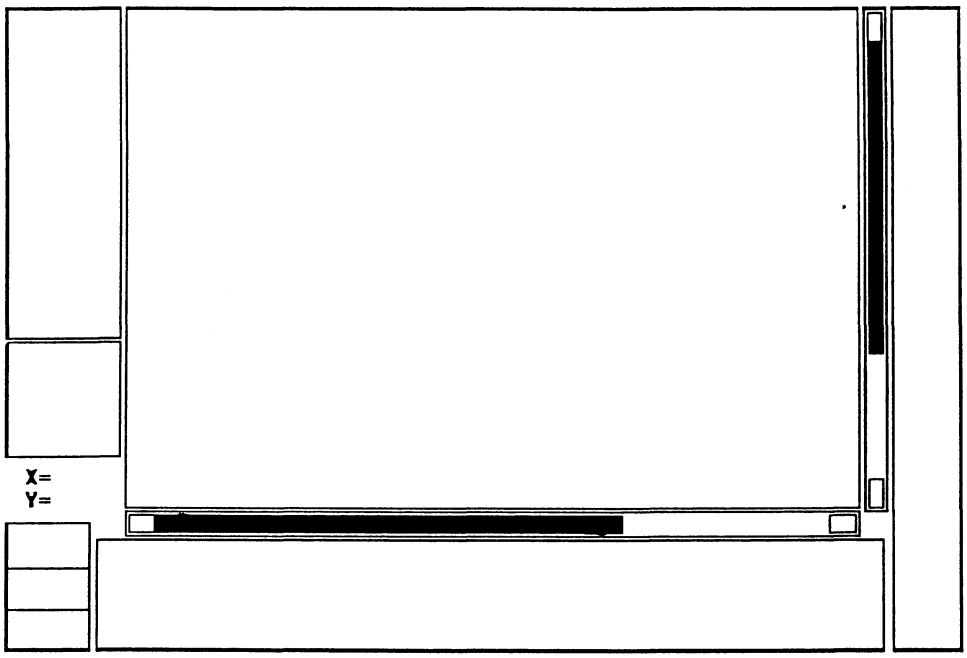


Figure 6-8 Scroll Bars

Figure 6-8 shows the scroll bars that run along the right and bottom edges of the editing window. The editing field of the Picture Editor allows approximately 56% of the edit area to be viewed at any time. Thus, some of the drawing may be off of the current editing screen. These bars allow the user to access any part of the screen.

To move, place the pointer over the arrow indicating the desired direction and press the right button on the mouse. The arrows pointing horizontally move the page left-to-right (left arrow shifts the page to the right; right arrow shifts the page to the left). Those arrows pointing vertically move the page top-to-bottom (upper arrow the page downward; lower arrow shifts the page upward). The dark areas will indicate the relative position of the page.

Example: If the bars are nearest the upper and left arrows (the default setting), the area of the editing field being viewed is nearest the upper-left portion of the page.

6.5.8 Line Width Selection

Four different line widths may be chosen for drawing rectangles, arcs, etc. The selection is made under the shape selections on the left edge of the screen:

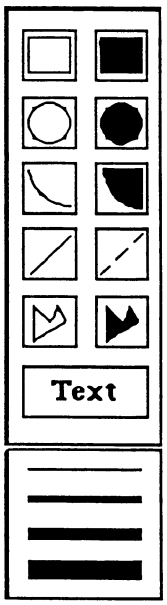


Figure 6-9 Line Widths

Figure 6-9 shows the line width selection window. The three point line width is currently shown selected. The four widths represent:

- 1-point (1/72" - default setting)
- 3-point (1/24")
- 5-point (5/72")
- 7-point (7/72")

> **NOTE**
> The above values are only approximations.

The selected line width will appear as the perimeter in all non-filled shapes, and as the border in all shapes where an outline is specified.

6.5.9 **Color Selection**

Up to 64 different colors can be selected, although only sixteen can be used within the Editor. (The upper eight are fixed and the others user configurable, although not from within the Picture Editor. Refer to the Chapters on Remote Commands and Terminal Primitives for the palette color modification process.)

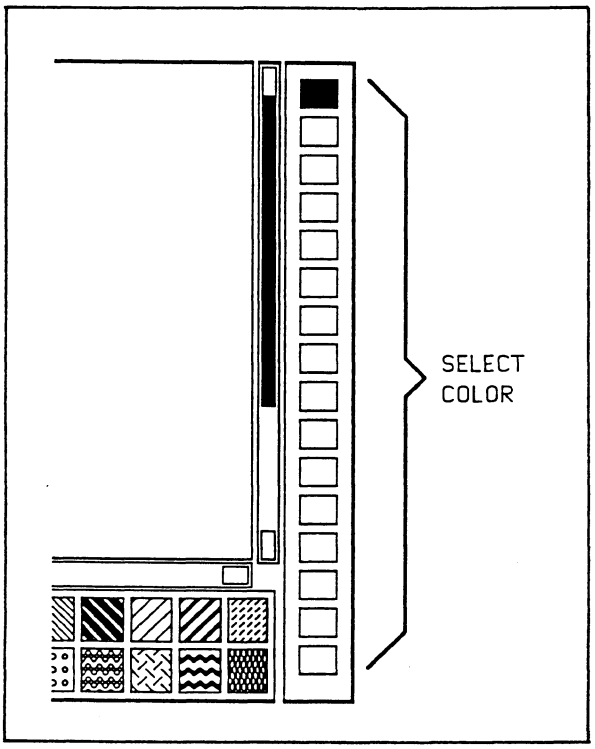


Figure 6-10 Color Selection

Figure 6-10 shows the color icon menu on the right edge of the screen. By positioning the pointer over the selection and pressing the right button, the color selection is made. The sixteen squares represent the current colors. A box will surround the color being used, and the pattern selection along the bottom of the screen should reflect the color in which the user is working.

The default foreground color is black, the background white.

> NOTE
> Colors 8-15 can be changed at run-time from the default values to any other VGA color. Thus, pictures at run-time may differ from those on the Editor. See the help menu for a complete listing of the colors available.

6.5.10 **Pattern Selection**

The OmniVU has 32 fill patterns to choose from, including two solid (the set background and foreground).

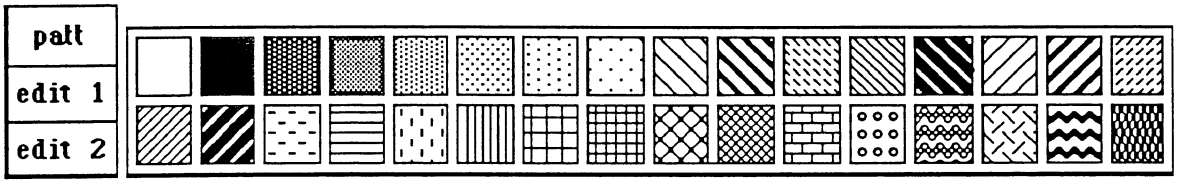


Figure 6-11 Pattern Selection and Edit Menus

Figure 6-11 illustrates the pattern selection menu. These are the patterns that are used when a filled shape is drawn. As a rule, the pattern selected will automatically appear within the perimeter of the shape that has been drawn.

The patterns are selected by moving the pointer over the selection and pressing the right mouse button. A thin white line will encircle the current selection.

The default pattern is the solid foreground color.

6.6 EDIT 1 FEATURES

Nine edit options can be accessed by first selecting "edit 1" with the select arrow, then choosing the appropriate edit option. The nine options available under the "edit 1" title are:

Display Reference Points
Full Screen Display
Outline Items
Display Grid
Snap to Grid
Set Background
Text Fonts
Change Attributes
Add Picture

> **NOTE**
> Press <ESC> to prematurely exit any of these options without saving the changes made.

Figure 6-12 illustrates the menu.

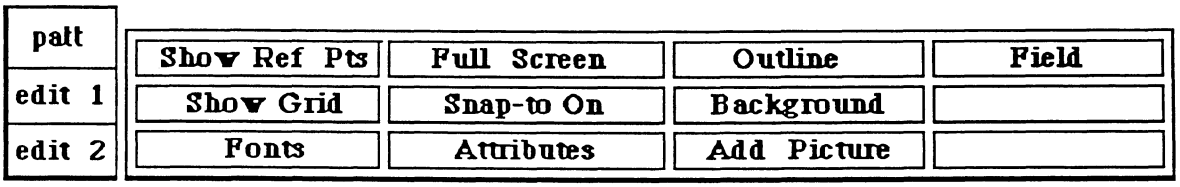


Figure 6-12 Edit 1

The currently selected option will appear with an outline around it as it is chosen.

The user can select "patt" to jump back to the fill pattern menu, or select "edit 2" to jump to the other edit menu.

6.6.1 Display Reference Points

This option acts as a toggle switch, allowing or disallowing reference points to be displayed within the edit field. If this option has been selected, the menu slot will read:

Hide Ref Pts

If the option has not been selected, the menu slot will read:

Display Ref Pts

> **NOTE**
> The "Hide Ref Pts" selection differs from the "Delete Ref Pts" choice. "Hide..." only removes the point from screen view, whereas "Delete..." removes them from the screen and from the picture file.

6.6.2 Full Screen Display

This option, when selected, removes the edit options from the screen and displays a full-screen version of the editing field. No editing, however, can take place in this mode. Pressing any key will return the user to the Editor and the edit mode.

6.6.3 Outline Items

This option adds a solid perimeter to all shapes subsequently created within the editing field. When selected, the following prompt will appear:

Select Outline Color

Select the color, as explained in Section 6.5.9. Each newly drawn item within the editing field will be outlined with that color.

To remove the outline feature, hold the right button and move the cursor between colors until the message appears:

No Outline

Then, release the button.

6.6.4 Display Grid

When this option is selected, a multi-colored grid pattern appears within the Picture Editor edit field. This option can be used to align picture elements as they are being drawn. If the option is selected, the menu will read:

Hide Grid

If the option has not been selected, the menu title will read:

Display Grid

6.6.5 Snap to Grid

When this option is selected, the crosshatch within the editing field will move from one grid square at a time (instead of moving smoothly) as the mouse is moved. If this option is selected, the following title will appear on the menu:

Turn Off Snap

If the option has not been selected, the title will read:

Snap to Grid

> **NOTE**
> If this function is enabled and the Display Grid is not, then the grid will automatically be displayed. Use Hide Grid to remove it.

6.6.6 Set Background

When this option is chosen, the background color used (to fill patterns and write text) changes from the default (white) to the new color selected.

When the prompt appears ("Select Background Color"), move the select arrow to the desired color and press the right mouse button. The background color will be shown as a rectangle surrounding the current foreground color.

6.6.7 Text Fonts

This option selects one of four font sizes for text that is created within the editing field. Those choices are:

- Regular Size (default)
- Small Size
- Double Size
- Quad Size

When Text Fonts is selected, move the mouse left and right to choose the desired font size. Press the right button to enter the selection.

The following choice appears within the menu choices in case a font change is not really desired:

Quit (changes not saved)

The **<ESC>** key will also leave the menu without saving changes.

6.6.8 Change Attributes

This selection allows four characteristics of a picture to be altered. The left mouse button is used to select which picture element is to be changed. Release the button to enter selection. The mouse can be moved left and right to select any of the following options:

change color
change line width
change background color
change pattern
quit (changes not saved [same as **<ESC>**])
quit (changes saved)

Press the right mouse button to select the desired option.

The changes are selected as any other choices are selected. When each choice is made, the user is then given the opportunity to select a new attribute, as if they were being set for the first time. After all of the changes have been made, the exit options can be selected.

6.6.9 **Add Picture**

This option is used to import one picture file into another. The option allows the imported image to be scaled to size, but does not allow the rotation or scaling of text. When this function is selected, the following prompt will appear:

Select Picture Name:

When the prompt appears, enter the filename of the picture file to be imported. Then move the crosshatch to the location (reference point, etc) where the imported image will be placed.

To insert the image, press the left button on the mouse and move diagonally. A rectangle will form that will approximate the size of the placed drawing. When the desired size and location are found, release the button to pull the file in.

If the file does not exist a message will be displayed and a keystroke will be needed to exit.

6.7 EDIT 2 FEATURES

Nine edit options can be accessed by first selecting "edit 2" with the select arrow, then choosing the appropriate edit option. The nine options available under the "edit 2" title are:

Anchor Point
Reference Points
Delete Reference Points
Delete Item
Undelete Item
Copy Items
Move Items
Scale Items
Erase Picture

Figure 6-13 illustrates the menu.

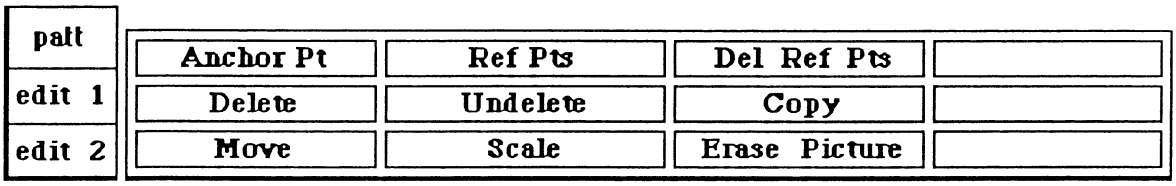


Figure 6-13 Edit 2

The currently selected option will appear with an outline around it as it is chosen.

The user can select "patt" to jump back to the fill pattern menu, or select "edit 1" to jump to the other edit menu.

6.7.1 Anchor Point

This option is used to determine where the outer boundaries of a drawn set of images will be located.

> **OIL-2 ONLY**: When a picture is displayed at run time, the anchor point of the retrieved file can be matched to a reference point in the file already present in the field.

> **NOTE**
> When a picture is drawn at run-time, the anchor point is located at the specified x-y coordinate.

Use the left button to position the anchor point on the screen. Release the button to enter the new location. The anchor point will be displayed on the screen in the following format:

xA

where: **x** = position of anchor point
A = "anchor" label

6.7.2 Reference Points

This option is used primarily when the OIL-2 Expansion Module is installed, but can be used without. It allows images from other files (or elsewhere on the editing field) to be added to the on-screen picture. Up to six reference points may be stored within the picture file at any given time. If the OIL-2 module is not installed, the reference point is an inactive point on the screen.

Move the crosshatch to the point on the edit field where the reference point is to appear. Press and release the left button on the mouse (the reference point will appear as "xA" where "x" is the reference mark and "A" is the "Anchor" point label).

When pictures are drawn at run-time, the physical location of a reference point can be read and used for placement of additional pictures. (See the "DP" command in the OIL-2 manual).

6.7.3 Delete Reference Points

When this option is selected, the left mouse button is used to select the reference point to be deleted. When a small square appears around the x-marker the button can be released, deleting the reference point from the picture file.

6.7.4 **Delete Item**

This option is used to remove an object from the editing field. When selected, the arrow cursor can be used to select the object for deletion.

Locate the item to be deleted and position the crosshatch over it. Press the left mouse button to select the item to be deleted. A frame will highlight the item. Then the button is released, deleting the reference item from the screen and the picture file.

The last deleted item is saved in a buffer until another item is placed into the buffer, or until the Picture Editor has been exited. This buffer is used in the Undelete function.

6.7.5 **Undelete Item/Picture**

"Undelete" is a command function that recalls the last deleted image. When this option is selected, the last deleted image reappears on-screen (at the same coordinates as when deleted, BUT onto the foreground of the screen). Pictures can also be restored if they are the last object to be placed into the delete buffer.

6.7.6 **Copy Items**

This option duplicates the "Move Items" option, except that the blocked items are copied as opposed to being moved. Reference points are not copied.

6.7.7 **Move Items**

This feature allows one or more images within the editing field to be moved to any location in the edit field. This feature can also be used to move smaller objects from behind larger ones (when undelete or another feature has covered them). This is accomplished in two steps.

Step 1:
Move the crosshatch to the appropriate location in the editing field. Press the left button of the mouse and move the cursor (diagonally) until the images to be moved are completely surrounded. When the button is released, small tabs will appear in the corners of the rectangle surrounding the image.

Step 2:
Move the crosshatch to one of the tabs and press the left mouse button again. While holding the button down, move the image to the desired location. Release the button and the image will become seated at the new location.

6.7.8 **Scale Items**

This option works exactly the same way as "Move Items," with one exception. When the tabs are moved, the image is expanded or contracted. The feature allows the blocked images to be altered in size and shape. Text and reference points are not scaled.

6.7.9 **Erase Picture**

When this option is selected, the entire Picture Editor field is cleared. Items deleted using this method can only be recalled if further activity within the editing field does not occur before the planned recall. Pattern, color (foreground & background) and line sizes, however, are reset to start-up values.

Chapter 7

FONT EDITOR

7.1 INTRODUCTION

The OmniVU Font Editor is a menu-based interface that allows any of the characters in the four graphics mode fonts to be edited on screen, pixel by pixel. The editor requires the use of a serial mouse. See Chapter 2 for installation instructions for the mouse.

> **NOTE**
> Only the fonts in the graphics mode can be edited, the text mode fonts are fixed.

7.2 USING THE FONT EDITOR

The Font Editor is accessed by typing "**F**iles **E**dit **F**ont" from the Main Menu. At that point, the on-screen menu will offer the following selections:

Small Size
Regular Size
Double Size
Quad Size

Each of the sizes corresponds to a particular letter size as follows:

Small Size	(8 pixels wide; 7 pixels high)
Regular Size	(8 pixels wide; 14 pixels high)
Double Size	(16 pixels wide; 28 pixels high)
Quad Size	(32 pixels wide; 56 pixels high)

One pixel is equal to approximately 1/72".

All selections can be made by either typing the highlighted letter in front of the choice, or by depressing the left mouse button when the option is highlighted.

7.2.1 Editing Fonts

Once a selection is made, the screen will display the Font Editor screen (see Figure 7-1). The character to be edited will appear in a large grid on the left side of the screen and in a box labelled "Actual Size" on the lower right of the screen.

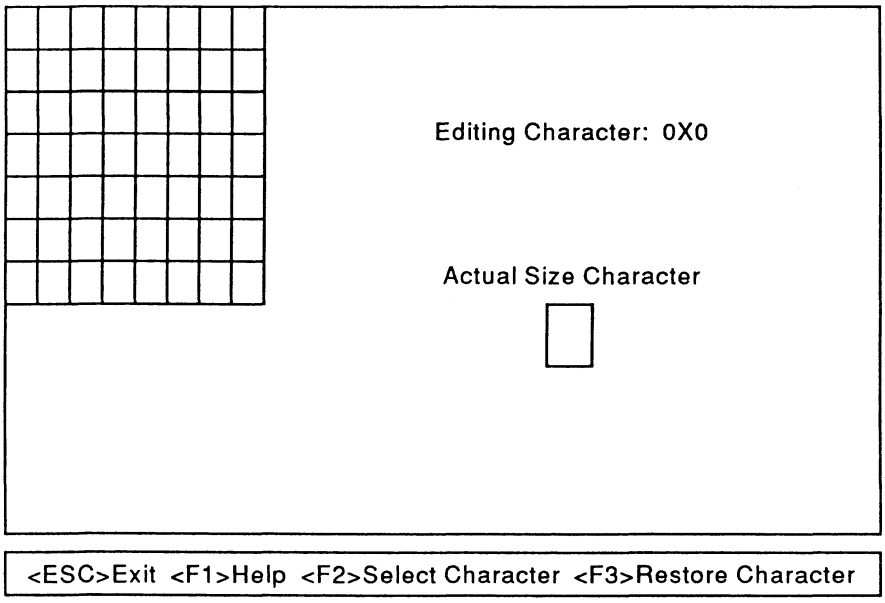


Figure 7-1 Font Editor Screen

The section at the bottom of the screen offers the following four choices:

<ESC> ——— Escape: to save changes or return to the size-selection menu without saving changes

<F1> ——— Help: on-screen assistance regarding the use of the Font Editor

<F2> ——— Select Character: places a prompt in the line near the top of the screen for the two-digit hex code of the ASCII character to be edited

<F3> ——— Restore Character: returns the altered character to its original value

Editing a character:

1. Enter the Font Editor and select the size of the font to be edited.

2. Type **<F2>** and the two-digit hexadecimal code for the ASCII character to select the character to be edited.

3. Move the arrow in the editing field to the pixel square that needs to be altered. Depressing the left button on the mouse will turn a pixel ON if it is OFF, or OFF if it is ON.

4. View the changes in the "Actual Size" box near the lower right corner of the screen.

5. When all changes are made, save the edited character.

Saving a character:

1. Press the **<ESC>** key

2. Press the **<F1>** key to save the changes and exit to the previous menu

7.3 VERIFYING SAVED FONT FILES

To verify that the changes to the font have been made, type "**F**iles **F**iles List **F**ont" from the Main Menu. The name of the last file listed should correspond to the two-digit hexadecimal code that was edited.

When the definitions for the fonts are edited, files are created on memory unit A to store the new definitions. To return to the default settings, the files can be renamed or deleted. They can also be backed up and restored to the default or another memory unit.

All of the created definitions will go into one of four files, depending on which fonts were edited. The filenames are assigned as follows:

smlfnt - definitions for the small size font
regfnt - definitions for the regular size font
dblfnt - definitions for the double size font
quadfnt - definitions for the quad size font

These files will only appear on the system the default definitions are edited.

Chapter 8

TERMINAL PRIMITIVES

8.1 INTRODUCTION

Each of the keypad keys, remote commands, and special keys on an external keyboard have a definition associated with it. These definitions contain one or more primitives that determine the action taken when one is invoked. The primitives contained in each definition can be added and/or deleted entirely, or the parameters contained within each modified, giving the user the ability to reprogram any of the keypad keys, remote commands, or special keys.

The definition, made up of sets of primitives, defines exactly what happens when a key is pressed or a remote command received. This set in each can be one or many primitives. Some primitives have parameters, and some do not.

Figure 8-1 illustrates just how this process works.

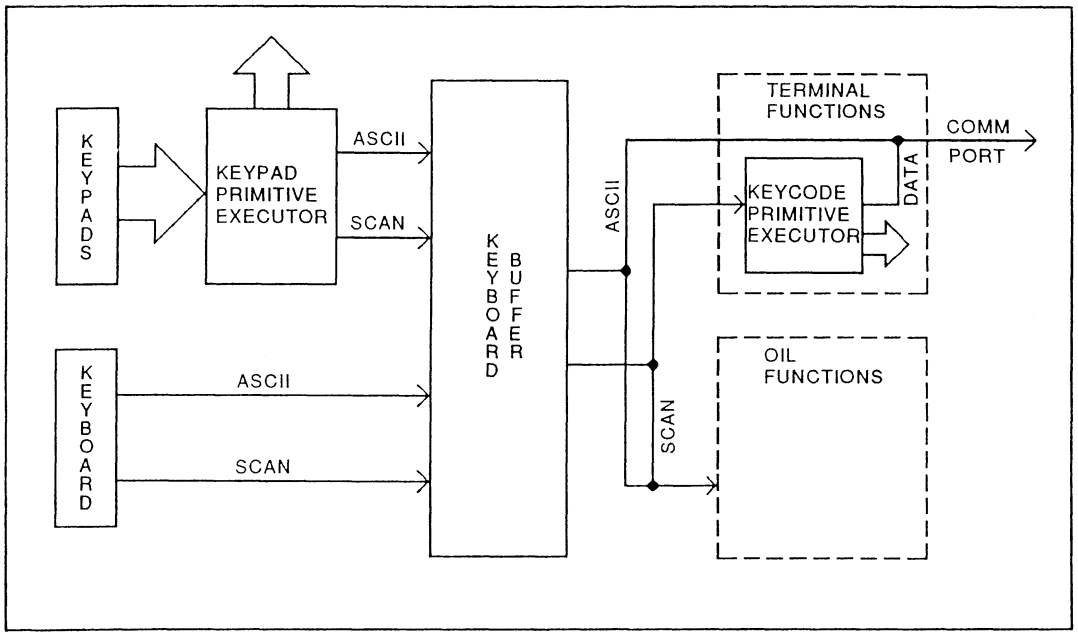


Figure 8-1 Keyboard Data Flow Diagram

When a key on the keypad or an external keyboard is pressed, a signal is sent to the keyboard buffer. This signal can be an ASCII or scan code (refer to Chapter 5 for details). The signal sent from the terminal keypads is generated by the Keypad Primitive Executor (see Figure 8-2).

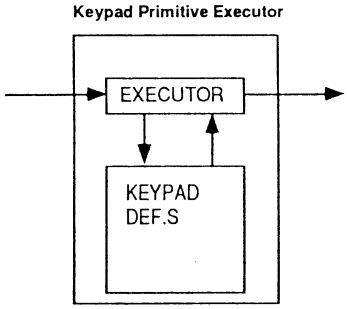


Figure 8-2 Keypad Primitive Executor

The Keypad Primitive Executor holds all of the definitions for the keypad keys. The default definitions for all of the keypad keys are programmed to emulate their counterparts on a regular keyboard. Thus, either the appropriate ASCII or scan code is sent to the keyboard buffer. This code can be altered, using the Primitive Editor, to initiate virtually any action of which the terminal is capable. It does not necessarily have to send a code to the keyboard buffer.

An external keyboard does not route its signals through the Keypad Primitive Executor. It places either an ASCII or scan code directly into the keyboard buffer.

Once stored in the keyboard buffer, these codes can be read by either The Keyboard Task in the Terminal Function Section, or the OIL Function Section (if installed). The OIL function section does not differentiate between ASCII and scan codes as The Keyboard Task does. It will process all of the signals in the same manner.

When the Terminal Function Section reads the signals, it processes them separately (refer to Chapter 5 for a complete explanation). ASCII codes are sent out the serial port. Scan codes invoke a definition in the Keycode Primitive Executor (see Figure 8-3). As in the Keypad Primitive Executor, this will also search a listing of the currently programmed definitions. These definitions can also be edited.

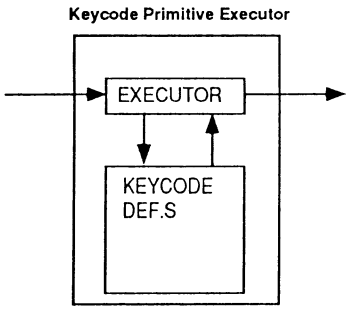


Figure 8-3 Keycode Primitive Executor

The output from the both the Keypad and Keyboard Primitive Executors is simply data, i.e., it does not have to be either an ASCII or scan code. The action specified by a remote command can be any task which the terminal is capable (signified by the large arrow from each Executor in Figure 8-1). The data is either passed on to the buffer (as in the case of the Keypad Executor) or out the serial port (as in the case of the Keycode Executor) or used internally to perform some function.

The Terminal Functions Section also handles the remote commands that come in to the comm port (see Figure 8-4).

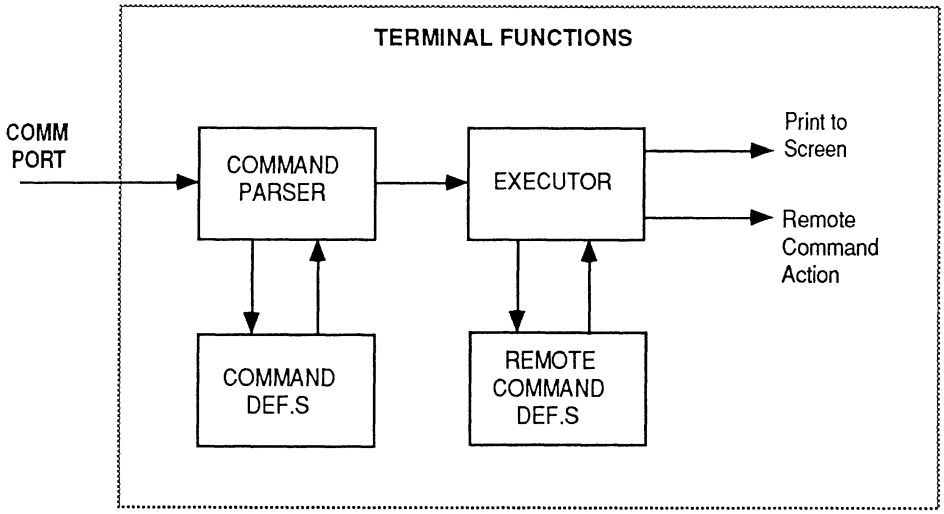


Figure 8-4 Remote Command Processing

When a remote command enters the terminal, the command parser compares it, character by character, against a listing of programmed commands (i.e., a defined lead-in, optional parameter set, and command identifier). If it does not match a specifically defined command, then the default definition number is referred to the Executor. If it does match, then that particular definition number is sent as reference.

The Executor will then use that reference number in another list of remote command definitions to determine the command sequence associated with the particular number. It will then take the appropriate action. The default definition is initially set to simply write those character to the screen, but can be programmed along with the rest of the definitions.

NOTE
For the definitions of the default codes for the remote commands, refer to Chapter 4. For the keypad definitions, refer to Chapter 5.

8.2 PRIMITIVE EDITOR

The OmniVU has a Primitive Editor. It functions outside of run-time. This Editor is used to alter the primitives, or definition, of each key or remote command. The Editor allows the user to add, delete, or change primitives for any of the remote commands or scan codes.

8.3 PROGRAMMING KEYPAD DEFINITIONS

The keypad definition primitives can be altered to perform most any function. To do this, the user must enter the Primitive Editor. This is done by selecting "**F**iles **E**dit **K**eypad Definitions" from the Main Menu.

A listing of all the front panel keys will appear on the screen. The user must select the key that will be programmed by highlighting the desired key and pressing **<Enter>**. When one is selected, the Primitive Editor will appear.

8.4 PROGRAMMING REMOTE COMMANDS

The remote command definition primitives, along with the definitions executed when a command is received and the action taken for special keys, can all be altered, too. To enter this Primitive Editor, the user selects "**F**iles **E**dit **R**emote Commands" from the Main Menu.

A submenu will appear. Here the user has the option of finding out more information about the remote commands and keys before proceeding. The menu looks like:

Type
Commands
Definitions
Keycodes

The user can select any of the four options. The Primitive Editor can be accessed only from the Definitions and Keycodes sections.

Type ——— Provides the user with the mode in which the Primitive Executor interprets each command. The choice is given between "ANSI" and "other." This will also determine the format that must be followed when entering or altering a new command.

Commands — The user can configure command definitions in this menu. A command can be added (the terminal prompts the user for all needed information), deleted, the definition number can be changed (where the user selects a command and assigned definition number and then is prompted for a new number), or the commands can be listed (along with assigned definition numbers).

> **NOTE**
> Each set of primitive instructions is assigned a definition number, which in turn is assigned to each remote commands. This makes it easy for the user to switch the remote command definitions by simply changing the definition number.

Definitions — Selected if the user already knows which definition number will be modified or created. Enter a definition number at the prompt and the Primitive Editor is accessed, or enter nothing and a numerical list is provided, and the Editor is accessed from there.

Keycodes —— When this is selected a listing of all the keys and their definition numbers assignments appear. A key can be selected and the Primitive Editor is accessed.

8.4.1 Terminal Function

The terminal starts two tasks when going into the operating mode: The Keyboard Task and The Comm Port Task. Both of these tasks use the same window. The default size and location of the window are set in the Logical Port Configuration Menu.

Each of the tasks has its own set of 30 registers (0-29) for use in the execution of definitions. These registers are initialized to zeros when entering Operating Mode. Any values placed into these registers remain valid as long as the terminal stays in Operating Mode.

A single INFO Array is used by both of the tasks. This array contains system information and can be used to communicate between the two tasks. The array has ten entries (0-9), currently assigned the following functions:

INFO 0 - comm port in use (1-4)
INFO 1 - window ID of window in use
INFO 2 - echo flag (0 = OFF, 1 = ON)
when ON, keyboard data is echoed to the comm port input buffer as if it were received over the comm port.
INFO 3 - keyboard lock (0 = OFF, 1 = ON)
when ON, keys typed on the keyboard are ignored. Set/cleared with remote commands.
INFO 4 - application mode (0 = OFF, 1 = ON)
when ON, certain keys on the keyboard generate application mode codes instead of regular codes. Set/cleared with remote commands.
INFO 5 - unused
INFO 6 - unused
INFO 7 - unused
INFO 8 - unused
INFO 9 - unused

8.4.2 The Default Definition

The Default Definition is a special definition, invoked when a character is received by the terminal that is not the prefix character of any of the defined remote commands. If any characters are received that fit this criteria, the default definition in invoked. This definition is defaulted to simply write all characters to the screen..

8.4.3 The Start-up Definition

The Start-up Definition is executed every time the terminal goes into Operating Mode. (It is defined in the "Options" menu.) This special definition is used to set up information to be used by the other definitions in the Operating Mode.

The default Start-up Definition sets up some registers for use in the windowing commands. Registers 0-7 are used for the cursor and attribute save and restore commands. Registers 10, 11, and 12 contain window identifiers for windows 1, 2, and 3, respectively. The remaining registers (13-29) are either not in use or are used for temporary storage and calculations.

Both The Default Definition and The Start-up Definition are programmable.

8.4.4 The Keyboard Task

This task reads data from the keyboard buffer. ASCII characters are sent out the comm port. If a scan code is read from the buffer, it is checked to see if it is one of the codes associated with one of the special keyboard keys. If so, that definition is executed. If not, it is ignored.

When the definition is executed, PAR 1 of the parameter array contains the scan code of the special key.

Currently none of the 30 registers are used in this task.

8.4.5 The Comm Port Task

This task reads data from the comm port and interprets it as either data or commands. For data, the Default Definition is executed. For remote commands, the assigned definition is executed.

When The Default Definition is called, the character received is passed to the parameter array entry PAR 1. When the definitions associated with the remote commands are called, PAR 0 contains the number of parameters received with the command and PAR 1 through PAR 9 contain the parameter values.

8.5 USING THE PRIMITIVE EDITOR

The directions for use of the Primitive Editor are positioned at the bottom of the screen. They will look like:

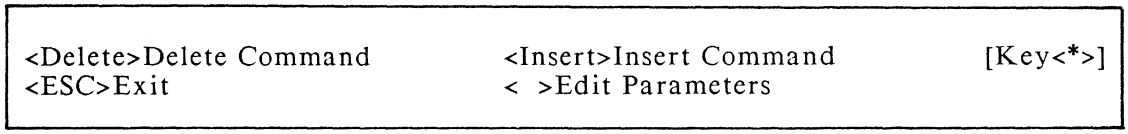


Figure 8-5 Primitive Editor Key

NOTE
* represents the key or definition number currently being modified in the Primitive Editor.

If the command set is correct, but the parameters need to be changed, use the right arrow key to select the parameter. If a deletion is needed, simply highlight the specific primitive and press **<Del>**. Press **<Ins>** if a new primitive is needed. Specifics are discussed below.

8.5.1 Editing Parameters

Select the parameter to be edited using the arrow keys. As soon as the screen highlight goes from the entire line to a specific parameter, the instruction set at the bottom of the screen will change. It will look like:

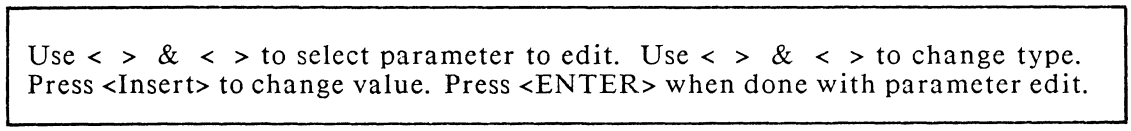


Figure 8-6 Parameter Editor Key

Use the arrow keys to select the parameter and type. Press **<Ins>** and type in a new value over the desired parameter. **<Enter>** returns the user to the previous instruction set window, highlighting the entire line again.

8.5.2 The <Del> Key

To delete a command primitive, highlight the specific primitive and press **<Del>**. The primitive will be deleted.

8.5.3 The <Ins> Key

To insert a command primitive, highlight the primitive that will follow the new command. New commands are always inserted ahead of the highlighted primitive. Press **<Ins>** and the Editor screen will change.

The highlighted command will have moved down one line and a new command will have appeared in the highlighted band. Also, a new set of instructions will appear at the bottom of the screen. They look like:

```
View available commands with < >, < >, <PageUp>, or <PageDown>.
<ENTER>Select < >Select & Edit Parameters <ESC> or <Delete>Delete
```

Figure 8-7 Primitive Insertion Key

The up and down arrows scroll the primitives within the highlighted band. The page up key will shift to the end of the previous "grouping" of primitives. The page down key will shift to the beginning of the next "grouping" of primitives. (Sections 8.6 - 8.9 contain the primitives, listed in "groups.") **<Enter>** saves the primitive in the command set, sending the user back to the previous instruction set window. **<Esc>** or **<Del>** cancels the insert operation.

The right arrow key selects the parameter to be edited. Press **<Enter>** to highlight the entire line again.

8.5.4 Rules For Defining New Commands

As a command is received, the terminal checks the characters. If a correct command identifier is not received first, the command is ignored. If the identifier is correct, the terminal looks at each characters and compares them to a command list, narrowing down the possibilities until the correct command is found. If, at any point, the command deviates from a known command then the command is ignored.

More than one command can use the same definition. It is also possible for one definition to call up another, thus creating subroutines.

Should any characters be received within a parameter that are not numbers or a semi-colon, then the command is ignored.

No command should be a substring of another command. If one command is a substring of another, the first will be executed and the second ignored. In some cases illegal commands could be specified by the user and not detected by the terminal command interpreter. It is the responsibility of the user to define deterministic rather than ambiguous commands. The commands must be defined both clearly and completely.

Illegal and Legal Examples:

Command	Explanation
Non-ANSI mode:	
~ A	Legal Command
~ A B	Command Ignored: previously defined command is a substring of this one
~ x	Command Ignored: previously defined command is a substring or no way to determine the value of "x"
~ B x y	Legal Command
ANSI mode:	
<ESC> [<pp> q	Legal Command
<ESC> [= <pp> q	Legal Command (different lead-in)
<ESC> [<pp> p	Legal Command (different identifier)
<ESC> [q	Illegal Command: cannot be determined if first command is being received without any parameters
<ESC> [q <pp> p	Command Ignored: previously defined command is a substring of this one
<ESC> [1 q	Illegal Command: cannot be differentiated between this one and the first being received with a parameter of "1"

It is easier to tell an illegal non-ANSI command, and often difficult to spot an illegal ANSI command.

In ANSI mode, commands should not be defined containing number symbols because the command interpreter could mistake them for parameters. Commands that contain parameters do not necessarily need to receive parameters to be interpreted.

8.5.5 Exiting the Editor

The user must exit the Editor from the first window. This is done by pressing the **<Esc>** key. A new screen will appear prompting the user to select one of three options. They look like:

```
Press <ESC> to exit without saving changes
Press <F1> to exit and save
Press <ENTER> to return to the editor
```

Figure 8-8 Exit Options

The user must select one of the options. Selecting **<ESC>** or **<F1>** will return the user to the menu from which the Primitive Editor was accessed. Selecting **<ENTER>** will return the user to the Editor.

8.5 PRIMITIVES PARAMETERS

The terminal primitives determine what action is taken when a definition is invoked. As stated earlier, some primitives have parameters, and some do not. It depends on the function of the primitive.

There are eight types of parameters that can be passed to the primitives. They are:

- constant hex values
- constant decimal values
- constant character values
- constant strings
- values stored in registers
- values stored in an information array
- values stored in a parameter array
- variable strings

The following illustrates some of the different types of command parameters.

Command	Param 1	Param 2	Param 3	Param 4
Mov	(PAR 1 ,	REG 1)		
And	(0x7ff ,	REG 1)		
Out COMM Port	(INFO 0,	0	, CSTRING)	
Show Decimal	(REG 1)			
Out COMM Port	(INFO 0,	0	, VSTRING)	
Out COMM Port	(INFO 0,	0	, CHR ';')	

The parameter type varies greatly in this example. In the first command, the values stored in Parameter Array 1 and Register 1 are being called upon. In the second, hex value is used. An information array appears in three places within the example. In the third and fifth, different string values are being used. And in the final line, a constant character value is invoked.

All of these parameter formats are described in detail on the following pages.

Constant hex values: 0xXXXX

where: 0x = specifies the value as a hex constant
XXXX = followed by a 1 to 4 digit hex value.

The hex constant is selected with the up or down arrow key until the digit 0 is displayed in the parameter field. The hex values are set by typing "0X" followed by the appropriate digits. Hex values can range from 0 to 0XFFFF. These values are constant, and are stored as part of the definition being edited.

Constant decimal values: XXXXX

where XXXXX = is any 1 to 5 digit decimal number.

The decimal constant is selected with the up or down arrow key until the digit 0 is displayed in the parameter field. The decimal values are set by typing the appropriate 1 to 5 digits representing the decimal value. Decimal values can range from 0 to 65535. These values are constant, and are stored as part of the definition being edited.

Constant character values: CHR x

where x = is the character.

The character can be any ASCII character. The value is constant, and is stored as part of the definition being edited.

Constant string values: CSTRING

The contents of the string can be viewed by editing the parameter. Non-displayable characters are shown as <xxx>, where xxx is the ASCII name for that value (e.g., NUL, SOH, etc.). Non-displayable characters are entered using the same format. The string is constant, and stored as part of the definition being edited.

Variable string: VSTRING

The variable string is used to build up strings during the execution of a definition. Primitives are used to store data into a variable string. The contents of the variable string is valid only within a single definition after a primitive has written data into it.

Register values: **REG xx**

where **xx** = is the register number.

These are values stored in memory locations used by the definitions for storage of run-time or temporary data. Currently, the definitions for the keypad and function keys do not contain any allotted register space, where as the definitions for the remote commands and standard keyboard keys do. The values in these registers is not constant. The values stored in registers remain valid during the execution of a definition, but might not be preserved from execution to execution.

Information array: **INFO x**

where **x** = is a one digit decimal number indicating a specific information item.

This contains run-time information for use by the primitives. Currently, the definitions for the keypad and function keys do not have an information array, where as the definitions for the remote commands and standard keyboard keys do.

Parameter array **PAR x**

where **x** = is the parameter number.

This array is used to pass information to the definition. For the keypad and function key definitions and terminal special key definitions, **PAR 1** contains the scan code of the key pressed. For the terminal remote command definitions, **PAR 0** contains the number of parameters received for the command and **PAR 1 - Par n** are the values for those parameters.

8.6 FLOW CONTROL PRIMITIVES

The flow control primitives are used to control the flow of execution within a definition. They appear as follows:

8.6.1 END

Syntax: **End**

Signals the end of a definition. All definitions must end with this primitive. It cannot be deleted from the definition.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
End					

This primitive will end a definition.

8.6.2 EXIT

Syntax: **Exit**

This will terminate the definition execution early. When encountered, the execution of the definition is ended.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Exit					

This primitive will end the execution at the point at which it was inserted.

8.6.3 ENDIF

Syntax: **Endif**

This forms the end of an "if block." Every "Ifxx" primitive must have a corresponding "Endif." The two primitives bracket a number of primitives to be executed when the "Ifxx" condition is true.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Endif					

This will be the terminating end of an "if block". The group of primitives surrounded by this and an Ifxx statement will execute only if the specified "if condition" is true.

8.6.4 JUMP TO LABEL

Syntax: **Jump to Label(a)**

where **a** = label number

This causes control to go to the primitive following the "Label" primitive whose number is the same as the number specified.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Jump to Label	(7)			

This primitive will cause the execution to go to the primitive following a Label primitive whose number is 7 (i.e., Label (7)).

8.6.5 CALL DEFINITION

Syntax: **Call Def.(a)**

where **a** = definition being called

This primitive is used to call a definition. When the definition is finished, control is returned to the next primitive.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Call Def.	(41)				

This primitive will invoke definition 41 and returns to the current definition when the execution of definition 41 is complete.

8.6.7 JUMP TO DEFINITION

Syntax: **Jump to Def.(a)**

where **a** = the destination of the jump

This command will jump to the definition indicated.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Jump to Def. (	41)				

This primitive will invoke definition 41, but will not return to the current definition when execution of definition 41 is complete.

8.6.8 LABEL

Syntax: **Label(a)**

where **a** = the label number.

This is the label primitive used by the "Jump to Label" primitive.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Label	(13)				

This primitive defines a label where the execution of a definition will continue after a "Jump to Label" primitive "Jump to Label (13)".

8.6.9 RETURN

Syntax: **Return**

This primitive returns the control to the calling definition. This is used in definitions called by other definitions.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Return					

This would return the execution to the previous definition. It is used as a complement to "Call def.(a)."

8.6.10 JUMP

Syntax: **IJump to Def.(a)**

where **a** = destination

This will cause a jump to the definition specified in the definition table. The definition table is built with a series of "Def." primitives.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
IJump to Def.	(11)			

This primitive will cause the execution to continue in the definition listing in the 11th entry in the table of definition primitives.

8.6.11 DEFINITION

Syntax: **Def.(a)**

where **a** = the definition to be set

Used to build up a definition table.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
Def.	(11)				

This primitive sets an entry in the definition table to definition 11.

If a table had the following three entries:

Def. (5)
Def. (21)
Def. (19)

then the commands listed below would have the effect listed to the right of each:

Command	Effect
IJump to Def.(1)	control continues at definition 5
IJump to Def.(2)	control continues at definition 19
IJump to Def.(3)	control continues at definition 21

8.6.12 SPACE

Syntax: **space**

A blank space. This primitive is used to add white space within a definition.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
space					

This will cause a space to be added within the current definition.

8.6.13 COMMENT

Syntax: **Comment(a)**

where **a** = "CSTRING" parameter containing the comment.

This is the comment primitive. It is used to add comments to a definition.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Comment	(CSTRING)				

This will cause whatever is written in the constant string to be retained as part of the current definition. It can be read by editing the primitive.

8.7 CONDITIONAL PRIMITIVES

The conditional primitives test for some condition. If the condition is true, execution continues with the next primitive. If the condition is false, execution continues with the primitive following the "Endif" primitive associated with the conditional primitive. Nesting of "Ifxx - EndIf" blocks is allowed.

They appear as follows:

8.7.1 EQUAL

Syntax: **IfEQ(a,b)**

where **a** = value
b = value

This is the A = B primitive. If the two variables equal each other, then execution continues on the next primitive. If they are not equal, then execution jumps ahead to the primitive following the "EndIf" primitive. Nested blocks are handled.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
IfEQ	(PAR 1 ,	2)			

In this primitive, the value of Parameter Array 1 is being compared to the decimal value of 2. If it does equal 2, then the next primitive will be executed. If they do not equal, then the execution will jump ahead to the corresponding EndIf primitive.

8.7.2 NOT EQUAL

Syntax: **IfNE(a,b)**

where **a** = value
b = value

This is the A ≠ B primitive. If the two variables do not equal each other, then execution continues on the next primitive. If they are not equal, then execution jumps ahead to the primitive following the "EndIf" primitive. Nested blocks are handled.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
IfNE	(REG 1,	2)			

In this primitive, the value of Register 1 is being compared to the decimal value of 2. If it does not equal 2, then the next primitive will be executed. If they do equal, then the execution will jump ahead to the corresponding EndIf primitive.

8.7.3 GREATER THAN

Syntax: **IfGT(a,b)**

where **a** = value
b = value

This is the A > B primitive. If A is greater than B, then execution continues on the next primitive. If not, then execution jumps ahead to the primitive following the "EndIf" primitive. Nested blocks are handled.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
IfGT	(	INFO 3,	OXFFF0)		

In this primitive, the value of Information Array 3 is being compared to the hexadecimal value of OXFFF0. If it is greater than OXFFF0, then the next primitive will be executed. If it is equal to or less than OXFFF0, then the execution will jump ahead to the corresponding EndIf primitive.

8.7.4 GREATER OR EQUAL

Syntax: **IfGE(a,b)**

where **a** = value
b = value

This is the A >= B primitive. If A is greater than or equal to B, then execution continues on the next primitive. If not, then execution jumps ahead to the primitive following the "EndIf" primitive. Nested blocks are handled.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
IfGE	(	REG 4,	9)		

In this primitive, the value of Register 4 is being compared to the decimal value of 9. If it is greater than or equal to 9, then the next primitive will be executed. If it is less than 9, then the execution will jump ahead to the corresponding EndIf primitive.

8.7.5 LESS THAN

Syntax: **IfLT(a,b)**

where **a** = value
b = value

This is the A < B primitive. If A is less than B, then execution continues on the next primitive. If not, then execution jumps ahead to the primitive following the "EndIf" primitive. Nested blocks are handled.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
IfLT	(	PAR 5 ,	7)		

In this primitive, the value of Parameter Array 5 is being compared to the decimal value of 7. If it is less than 7, then the next primitive will be executed. If it is equal to or greater than 7, then the execution will jump ahead to the corresponding EndIf primitive.

8.7.6 LESS OR EQUAL

Syntax: **IfLE(a,b)**

where **a** = value
b = value

This is the A <= B primitive. If A is less than or equal to B, then execution continues on the next primitive. If not, then execution jumps ahead to the primitive following the "EndIf" primitive. Nested blocks are handled.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
IfLE	(	REG 5,	OX000A)		

In this primitive, the value of Register 5 is being compared to the hexadecimal value of OX000A. If it is less than or equal to OX000A, then the next primitive will be executed. If it is greater than OX000A, then the execution will jump ahead to the corresponding EndIf primitive.

8.7.7 IN RANGE

Syntax: **IfIR(a,b,c)**

where **a** = value
b = value
c = value

This is the "in-range" primitive. If B is less than or equal to A, and A is less than or equal to C (B <= A <=C), then execution continues on the next primitive. If not, then execution jumps ahead to the primitive following the "EndIf" primitive. Nested blocks are handled.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
IfIR	(	PAR 5 ,	PAR 7 ,	REG 8)	

In this primitive, the value of Parameter Array 5 is being compared to the values in Parameter Array 7 and Register 8. If it is greater than or equal to Parameter Array 7 and less than or equal to Register 8, then the next primitive will be executed. If this condition does not exist for either relationship, or for only one, then the execution will jump ahead to the corresponding EndIf primitive.

8.8 OPERATIONAL PRIMITIVES

The operational primitives perform some manipulation of data.

They appear as follows:

8.8.1 MOVE

Syntax: **Mov(a,b)**

where **a** = value to be transferred
b = transfer destination

Moves the value of one register into another.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Mov	(	PAR 3 ,	REG 3)		

This primitive will move the value from Parameter Array 3 into Register 3.

8.8.2 ADD

Syntax: **Add(a,b)**

where **a** = value to be added
b = other value to be added and destination of sum

Add the values of two registers and store the result in **b**.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Add	(	OX3CCC,	INFO 4)		

This primitive will add the hexadecimal value OX3CCC to the value in Information Array 4, and store the sum in Information Array 4.

8.8.3 SUBTRACT

Syntax: **Sub(a,b)**

where **a** = value to be subtracted
b = other value to be subtracted and destination of result

Subtract the value of **a** from **b** and store the result in **b**.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Sub	(346	, PAR 7)			

This primitive will subtract 346 from the value in Parameter Array 7, and store the result in Parameter Array 7.

8.8.4 INCREMENT

Syntax: **Inc(a)**

where **a** = the value to be incremented by one.

This function will increment the value indicated.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Inc	(	REG 5)			

This parameter will increment the value in Register 5 by one each time it is executed.

8.8.5 DECREMENT

Syntax: **Dec(a)**

where **a** = the value to be decremented by one.

This function will decrement the value indicated.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Dec	(	REG 5)			

This parameter will decrement the value in Register 5 by one each time it is executed.

8.8.6 MULTIPLY

Syntax: **Mul(a,b)**

where **a** = value to be multiplied
b = other value to be multiplied and destination of sum

Multiply the values in two registers and store the result in **b**.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Mul	(34	, REG 4)			

This primitive will multiply the value in Register 4 by 34 and store the result in Register 4.

8.8.7 DIVIDE

Syntax: **Div(a,b)**

where **a** = value to be the divisor
b = value to be divided and destination of result

Divide the value of **b** by **a** and store the result in **b.**

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Div	(	OX3456,	PAR 9)		

This primitive will divide the value stored in Parameter Array 9 by the hexadecimal value OX3456, and store the result in Parameter Array 9.

8.8.8 MODULUS

Syntax: **Mod(a,b)**

where **a** = value to be the divisor
b = value to be divided and destination of remainder

Divide the value of one register by another and store the remainder in one of them.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Mod	(	INFO 6,	PAR 9)		

This primitive will divide the value stored in Parameter Array 9 by the value stored in Information Array 6 and store the remainder in Parameter Array 9.

8.8.9 OR

Syntax: **Or(a,b)**

where: **a** = one register
b = the other register and the destination for the result

Does a bit-wise OR of two registers and stores the result in one of them.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Or	(	PAR 4,	PAR 9)		

This primitive will do a bit-wise or of the values in Parameter Arrays 4 and 9, and store the result in Parameter Array 9.

8.8.10 AND

Syntax: **And(a,b)**

where: **a** = one register
b = the other register and the destination for the result

Does a bit-wise AND of two registers and stores the result in one of them.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
And	(	PAR 4,	PAR 9)		

This primitive will do a bit-wise and of the values in Parameter Arrays 4 and 9, and store the result in Parameter Array 9.

8.8.11 SHIFT LEFT

Syntax: **Shift Left(a,b)**

where: **a** = number of bits to shift by
b = value to shift and destination

Shift the data in a register left the number of bits indicated and store the result back into that register.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Shift Left	(3	, REG 6)			

This primitive will shift all of the bits in the value stored in Register 6 to the left by three, and store the resulting value in Register 6.

8.8.12 SHIFT RIGHT

Syntax: **Shift Right(a,b)**

where: **a** = number of bits to shift by
b = value to shift and destination

Shift the data in a register right the number of bits indicated and store the result back into that register.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Shift Right	(2	, REG 6)			

This primitive will shift all of the bits in the value stored in Register 6 to the right by two, and store the resulting value in Register 6.

8.8.13 NOT

Syntax: **Not(a)**

where **a** = a register to perform the operation on and the destination for the result

Perform a one's-complement on a register and store the result in **a**.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Not	(	INFO 8)			

This primitive will perform a one's complement on the value stored in Information Array 8.

8.8.14 READ REGISTER

Syntax: **Read Register(a,b)**

where **a** = register number to be read
b = destination

Read the register number indicated and store the contents in the other. Basically a data transfer. Can also be used for an indirect data transfer.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Read Register	(	REG 5,	PAR 4)	

If Register 5 contained an 8, this primitive would move the contents of Register 8 into Parameter Array 4.

8.8.15 READ PARAMETER

Syntax: **Read Parameter(a,b)**

where **a** = parameter number to be read
b = destination

Read the parameter number indicated and store the contents in the indicated register. Basically a data transfer. Can also be used for an indirect data transfer.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Read Parameter	(	PAR 4,	REG 5)	

If Parameter Array 4 contained a 1, this primitive would move the contents of Parameter Array 1 into Register 5.

8.8.16 WRITE REGISTER

Syntax: **Write Register(a,b)**

where **a** = value to write
b = destination

Write the value indicated to the register indicated. Basically a data transfer. Can also be used for an indirect data transfer.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Write Register	(63345	, REG 3)		

If Register 3 contained a 5, this primitive would write the value 63345 into Register 5.

8.9 COMMAND PRIMITIVES

The command primitives perform some operation on the OmniVU. The primitives that write data to the screen actually write data to the screen through a window on the screen.

There are separate windows used for keypad definitions, terminal remote commands and keycodes, and OIL-2 Tasks. Each window has its own size and location on the display screen. Windows can overlap, but data in one window overwritten by another window is not preserved. Each window has its own state information and attributes. The size and location of a window is specified in text coordinates. All coordinates within a window are relative to the upper left corner of the window.

All of the commands that deal with text and cursor use text coordinates. In text coordinates the x-axis value, referred to as the column, can range from 0 to 79. The y-axis value, referred to as the row, can range from 0 to 24. The maximum values will be dependent on the window size.

Conversely, all of the commands that deal with graphics use pixel coordinates. The x-axis value can range from 0 to 639, and the y-axis from 0 to 349. Again, the maximum values will depend on the set size of the window.

To convert text size to pixel size on the x-axis, multiply by a factor of 8. To convert from pixel size to text size, divide by a factor of 8. For the y-axis, use factors of 14.

The display screen will operate in one of two modes: Text or Graphics (set in "Options Base Screen" menu or in the Primitive Editor). They each have their own advantages and disadvantages. Table 8-1 provides a brief comparison:

Table 8-1 Text vs. Graphics Mode

TEXT	GRAPHICS
• Faster of the two modes	• Slower than text mode
• Supports only regular size characters	• Multiple character sets supported
• No drawing commands supported	• All drawing commands supported

The "Set Attributes" command sets the colors used for the display of text and drawing. It is also used to specify the attributes to use when writing text.

The "Set Draw Pars" command sets the drawing parameters used for the drawing commands.

They appear as follows:

8.9.1 TONE

Syntax: **Speaker(a,b)**

where **a** = the duration of the tone in ms
b = the frequency in Hz

Forces a tone on the speaker. Useful as an alert to perform some action.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Speaker	(10000	, 12000)			

This primitive will sound a 12KHz tone on the speaker for 10 seconds.

8.9.2 DATA TO KEYBOARD

Syntax: **To KYBD(a)**

where **a** = data to be placed in keyboard buffer

Puts data into the keyboard buffer. Data is either an ASCII code or a scan code whose values are from 0-255. To store a value as an ASCII code, **a** should be less than or equal to 255. To store a value as a scan code, multiply the scan code value by 256. Within the keyboard buffer, ASCII code are represented by values from 0-255; scan codes from 256-65535, in multiples of 256.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
To KYBD	(155)				

This primitive will place the ASCII value 155 into the keyboard buffer.

8.9.3 DATA TO SCREEN

Syntax: **To Screen(a)**

where **a** = data to be written

Writes data to the screen at the current cursor location, using the current attributes and mode. Also, the cursor position is updated. If the cursor is moved beyond the end of the line and auto-wrap is enabled, the cursor is moved to the start of the next line. Otherwise, the cursor remains on the current line. The same logic applies to the cursor when at the bottom of a screen with regard to the scrolling function.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
To Screen	(	CHR ';')			

This primitive will write the Character ';' on the screen at the current cursor location.

8.9.4 DATA TO COMM BUFFER

Syntax: **To COMM Buff(a,b)**

where **a** = comm buffer number (1-4)
b = data byte to be written

This primitive writes a data byte into a specific comm buffer. The comm buffer is read by OIL-2 programs when a transfer is performed from a comm port that is not owned by OIL-2.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
To COMM Buff	(3	,	CHR 'A')	

This primitive will write the character 'A' to comm buffer number three.

8.9.5 DATA FROM COMM PORT

Syntax: **Out COMM Port(a,b,c)**

where **a** = comm port (1-4)
b = echo flag value
c = data byte to be written

Write a data byte out the specific comm port indicated. If the echo flag is non-zero, the data byte is also put into the comm input buffer.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Out COMM Port	(1	, 0	, CHR 'B')	

This primitive will write the character 'B' out comm port 1. It will not write the byte to the comm input buffer, though, because the echo flag is zero.

8.9.6 WRITE COMM ECHO

Syntax: **To COMM Echo(a,b)**

where **a** = comm input buffer (1-4)
b = data byte to be written

Puts the specific data byte into the comm input buffer indicated.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
To COMM Echo	(2	,	CHR 'C')	

This primitive will write the character 'C' to comm input buffer 2.

8.9.7 WRITE TO PRINTER

Syntax: **To Printer(a,b)**

where **a** = timeout value in ms
b = data byte to be written

Writes a specific data byte to the printer port. If the timeout value is zero, the timeout time is infinite. The printer times out if it is off-line, out of paper, currently doing a print screen, or busy with the last character sent to it.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
To Printer	(0	, OX0C)			

This primitive will write the byte OX0C to the printer. This primitive will not complete until the data gets written out to the printer since the timeout value is infinite.

8.9.8 ERASE DATA

Syntax: **Erase(a,b)**

where **a** = the data type
0 = characters on a line
1 = from cursor to end of line
2 = from cursor to beginning of line
3 = entire line
4 = from cursor to end of window
5 = from cursor to beginning of window
6 = entire window
b = the count for the erase.

Erases specific data in a window.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
Erase	(3	, 0)			

This primitive will erase the entire line that the cursor is on.

8.9.9 CURSOR UP

Syntax: **Cursor Up(a,b)**

where **a** = the number of lines
b = one of the following options:
0 = no scroll or wrap
1 = scroll data in window down if the cursor is at the top edge
2 = wrap cursor to bottom of window in the same column
3 = wrap cursor to bottom of window in the previous column

Moves cursor up a predetermined number of lines.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Cursor Up	(2	, 0)			

This primitive will move the cursor up two lines, with no scroll or wrap. If the cursor is at the top line, it will not move. If it is on line one, it will only move up one.

8.9.10 CURSOR DOWN

Syntax: **Cursor Down(a,b)**

where **a** = the number of lines
b = one of the following options:
0 = no scroll or wrap
1 = scroll data in window up if the cursor is at the bottom edge
2 = wrap cursor to top of window in the same column
3 = wrap cursor to top of window in the previous column

Moves cursor down a predetermined number of lines

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Cursor Down (	6	, 2)			

This primitive will move the cursor down six lines. If it is within six lines of the bottom of the column, it will wrap the cursor to the top of the same column.

8.9.11 CURSOR LEFT

Syntax: **Cursor Left(a,b)**

where **a** = the number of columns
b = one of the following options:
0 = no scroll or wrap
1 = scroll data in window right if the cursor is at the left edge
2 = wrap cursor to right edge of window in the same row
3 = wrap cursor to right edge of window in the previous row

Moves cursor left a predetermined number of columns.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
Cursor Left	(15	, 0)			

This primitive will move the cursor left 15 spaces, with no scroll or wrap. It will only move as far as the left margin.

8.9.12 CURSOR RIGHT

Syntax: **Cursor Right(a,b)**

where **a** = the number of columns
b = one of the following options:
0 = no scroll or wrap
1 = scroll data in window left if the cursor is at the right edge
2 = wrap cursor to left edge of window in the same row
3 = wrap cursor to left edge of window in the previous row

Moves cursor right a predetermined number of columns.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Cursor Right (	1	, 1)			

This primitive will move the cursor right one space, scrolling data in the window to the left if the cursor is at the right edge of the screen.

8.9.13 SET ATTRIBUTES

Syntax: **Set Attribute(a)**

where **a** = can be set to any of the following definitions:

0	Attributes OFF (highlight, underline, and blink all set to **OFF**. Sending no parameter will have the same effect). **(default)**

Highlight ON/OFF:

1	Highlight ON
22	Highlight OFF

Underline ON/OFF:

4	Underline ON (graphics modes only)
24	Underline OFF

Blinking ON/OFF:

5	Blink ON (text mode only)
25	Blink OFF

Reverse Video ON:

7	Reverse Video (swap foreground and background colors)
27	Reverse Video (swap foreground and background colors)

Strikeout ON/OFF:

8	Strikeout ON (graphics mode only)
28	Strikeout OFF

The fore and background color selections are made by setting a palette register. There are 16 registers and 64 colors. Each palette register contains a code for a particular color. Registers 0-7 are fixed to certain colors, and cannot be changed. Registers 8-15 are configurable, and can be changed from their default settings. Selecting a color will use that register. To alter a color associated with a register, the value associated with that register must be changed.

Note that when a register is changed, all pictures containing that particular register will change color also. The color choices are listed in the help menu.

Foreground Color Selection I:

30	Palette Reg. 0 Foreground (fixed Black)
31	Palette Reg. 1 Foreground (fixed Blue)
32	Palette Reg. 2 Foreground (fixed Green)
33	Palette Reg. 3 Foreground (fixed Cyan)
34	Palette Reg. 4 Foreground (fixed Red)
35	Palette Reg. 5 Foreground (fixed Magenta)
36	Palette Reg. 6 Foreground (fixed Yellow)
37	Palette Reg. 7 Foreground (fixed White) **(default)**

Background Color Selection I:

40	Palette Reg. 0 Background (fixed Black)
41	Palette Reg. 1 Background (fixed Blue) **(default)**
42	Palette Reg. 2 Background (fixed Green)
43	Palette Reg. 3 Background (fixed Cyan)
44	Palette Reg. 4 Background (fixed Red)
45	Palette Reg. 5 Background (fixed Magenta)
46	Palette Reg. 6 Background (fixed Yellow)
47	Palette Reg. 7 Background (fixed White)

Character Size Selection:

50	Regular Characters **(default)**	
51	Double Size Characters	
52	Quad Size Characters	(graphics mode only)
53	Half-high Characters	

Foreground Color Selection II:

60	Palette Reg. 8 Foreground (IBlack default)	
61	Palette Reg. 9 Foreground (IBlue default)	
62	Palette Reg. 10 Foreground (IGreen default)	
63	Palette Reg. 11 Foreground (ICyan default)	(graphics mode only)
64	Palette Reg. 12 Foreground (IRed default)	
65	Palette Reg. 13 Foreground (IMagenta default)	
66	Palette Reg. 14 Foreground (IYellow default)	
67	Palette Reg. 15 Foreground (IWhite default)	

Background Color Selection II:

70	Palette Reg. 8 Background (IBlack default)	
71	Palette Reg. 9 Background (IBlue default)	
72	Palette Reg. 10 Background (IGreen default)	
73	Palette Reg. 11 Background (ICyan default)	(graphics mode only)
74	Palette Reg. 12 Background (IRed default)	
75	Palette Reg. 13 Background (IMagenta default)	
76	Palette Reg. 14 Background (IYellow default)	
77	Palette Reg. 15 Background (IWhite default)	

Text Write Mode Selection:

80	"Replace" text write mode **(default)**	
81	"Overlay" text write mode	
82	"Invert" text write mode	
83	"Erase" text write mode	(graphics mode only)
84	"~Replace" text write mode	
85	"~Overlay" text write mode	
86	"~Invert" text write mode	
87	"~Erase" text write mode	

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
Set Attribute (	43)				

This primitive will set the background color palette to Cyan.

8.9.14 TAB

Syntax: **Tab Operation(a)**

where **a** = any of the following:
0 = Set tab at cursor
1 = Clear tab at cursor
2 = Clear all tabs
3 = Go to tab

Performs all tab operations.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Tab Operation	(3)			

This primitive will send the cursor to the next set tab.

8.9.15 GOTO

Syntax: **Goto XY(a,b)**

where **a** = the column (0-79)
b = the row (0-24)

Sends cursor to the prescribed coordinates.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
GOTO	(0	, 0)			

This primitive will send the cursor to the upper left corner of the screen.

8.9.16 INSERT

Syntax: **Insert (a,b)**

where **a** = the type of insertion
0 = spaces
1 = lines
b = the number of insertions.

Inserts spaces or lines.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Insert	(0	, 6)			

This primitive will insert six spaces at the cursor.

8.9.17 NEW LINE

Syntax: **New Line**

This performs a carriage return. If "auto-linefeed" is ON, the cursor will go to the beginning of the next line, If it is OFF, then the cursor will go to the beginning of the current line. If the cursor is at the bottom of the window and must go to the beginning of the next line, then the data in the window is scrolled up if the "scrolling" function is enabled. Otherwise, the cursor goes to the top of the current window.

> **NOTE**
> The number of actual text lines is determined by the font size.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
New Line					

This primitive will perform a carriage return.

8.9.18 LINE FEED

Syntax: **Line Feed**

This will move the cursor down a number of lines, the exact number being dependent on the character size. If the cursor is at the bottom of the window and the "scrolling "function is enabled, then data in the window will be scrolled. Otherwise, the cursor will move to the top of the window.

This command is ignored if the "auto-linefeed" function is enabled.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
Line Feed					

This will move the cursor down one character space.

8.9.19 BACK SPACE

Syntax: **Back Space**

This will move the cursor back a number of columns, the exact number being dependent on the character size. If the cursor is at the beginning of a line, then it will go to the end of the previous line. If it is at the upper left corner of the window, it will not move.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Back Space					

This primitive will move the cursor back one character space.

8.9.20 SET MODE

Syntax: **Set Mode(a)**

where **a** = be any one of the following:
1 = Auto Linefeed
2 = Cursor
3 = Scrolling
4 = Auto-wrap
5 = Own Cursor

Set the mode of operation for the indicated variable to ON.

> **NOTE**
> Only one window can own the cursor (i.e., the last window to ask for ownership).

All of the Modes are defaulted to ON, except Cursor and Own Cursor.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Set Mode	(3)				

This primitive will turn the scrolling function to ON.

8.9.21 CLEAR MODE

Syntax: **Clear Mode(a)**

where **a** = be any one of the following:
1 = Auto Linefeed
2 = Cursor
3 = Scrolling
4 = Auto-wrap
5 = Own Cursor

Clears the mode set in the previous command. The variable must be set for that particular mode of operation.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Clear Mode	(4)				

This primitive will turn the Auto-wrap function to OFF.

8.9.22 DRAW RECTANGLE

Syntax: **Draw Rectangle(a,b,c,d)**

where **a** = x-axis coordinate (0-639) of upper left corner
b = y-axis coordinate (0-349) of upper left corner
c = x-axis coordinate (0-639) of lower right corner
d = y-axis coordinate (0-349) of lower right corner

This function draws a rectangle. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Draw Rectangle	(150	, 100	, 300	, 320)

This primitive will draw a box with the upper left and lower right corners represented by pixel coordinates 150,100 and 300,320.

8.9.23 FILLED RECTANGLE

Syntax: **Fill Rectangle(a, b, c, d)**

where **a** = x-axis coordinate (0-639) of upper left corner
b = y-axis coordinate (0-349) of upper left corner
c = x-axis coordinate (0-639) of lower right corner
d = y-axis coordinate (0-349) of lower right corner

This function draws a filled rectangle. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Fill Rectangle	(150	, 100	, 300	, 400)

This primitive will draw a box with the upper left and lower right corners represented by pixel coordinates 150,100 and 300,400, and filled with the current foreground color.

8.9.24 DRAW OVAL

Syntax: **Draw Oval(a, b, c, d)**

where **a** = x-axis coordinate (0-639) of center
b = y-axis coordinate (0-349) of center
c = x-axis radius of circle
d = y-axis radius of circle

This function will draw an oval, aligned with either the x- or y-axis. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Draw Oval	(250	, 100	, 50	, 10)	

This primitive will draw an oval with a center of pixel coordinates 250,100 and radii of 50 and 10 along the x- and y-axes, respectively.

8.9.25 FILLED OVAL

Syntax: **Fill Oval(a, b, c, d)**

where **a** = x-axis coordinate (0-639) of center
b = y-axis coordinate (0-349) of center
c = x-axis radius of circle
d = y-axis radius of circle

This function will draw a filled oval, aligned with either the x- or y-axis. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
Fill Oval	(250	, 100	, 50	, 10)	

This primitive will draw an oval with a center of pixel coordinates 250,100 and radii of 50 and 10 along the x- and y-axes, respectively, and filled with the current foreground color.

8.9.26 DRAW ARC

Syntax: **Draw Arc(a, b, c, d, e, f)**

where **a** = x-axis coordinate (0-639) of center
b = y-axis coordinate (0-349) of center
c = x-axis radius
d = y-axis radius
e = starting angle in degrees
f = ending angle in degrees

This function will draw an arc. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

0° is considered along the right x-axis (3 o'clock). The sweep is counter-clockwise.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**	**Param 6**
Draw Arc	(250	, 100	, 50	, 10	, 90	, 270)

This primitive will draw an arc from an oval with a center of pixel coordinates 250,100 and radii of 50 and 10 along the x- and y-axes, respectively, cut between 90° and 270°. This would appear to be the left half of the oval drawn in the DRAW OVAL primitive.

8.9.27 FILLED ARC

Syntax: **Fill Arc(a, b, c, d, e, f)**

where **a** = x-axis coordinate (0-639) of center
b = y-axis coordinate (0-349) of center
c = x-axis radius
d = y-axis radius
e = starting angle in degrees
f = ending angle in degrees

This function will draw a filled arc. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

0 °is considered along the right x-axis (3 o'clock). The sweep is counter-clockwise.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5	Param 6
Fill Arc	(250	, 100	, 50	, 10	, 90	, 270)

This primitive will draw an arc from an oval with a center of pixel coordinates 250,100 and radii of 50 and 10 along the x- and y-axes, respectively, cut between 90 °and 270 °and filled with the current foreground color. This would appear to be the left half of the oval drawn in the FILL OVAL primitive.

8.9.28 DRAW LINE

Syntax: **Draw Line(a, b, c, d)**

where **a** = x-axis coordinate (0-639) of start
b = y-axis coordinate (0-349) of start
c = x-axis coordinate (0-639) of end
d = y-axis coordinate (0-349) of end

This function will draw a line between any two points. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Draw Line	(100	, 100	, 250	, 300)	

This primitive will draw a line from the coordinates 100,100 to coordinate point 250,300.

8.9.29 DRAW PICTURE

Syntax: **Draw Picture(a, b, c, d, e)**

where **a** = unit on which the picture is defined (0-2)
b = file number of picture file on unit
c = x-axis coordinate (0-639) anchor
d = y-axis coordinate (0-349) anchor
e = scale factor

This function will place a picture at the desired coordinates and scaled to the specified factor. The anchor point for each picture can be set in the picture editor. It defaults to the upper left corner. All drawing commands use the colors set with the Character Attribute commands and parameters set with the Set Drawing Parameters command.

The scale can be specified in 16ths by using whole numbers from 1 to 16 (8/16, for instance would be half-sized; 32/16 double).

The "Files Unit List" menu can be used to find the file number of a particular picture file.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Draw Picture (	0 ,	33 ,	50 ,	50 ,	16)

This primitive will read the graphic in file 33, memory unit A and place it on the screen. The location will be determined by placing the anchor point of the graphic on coordinate point 50,50. The scale factor will be 100 percent.

8.9.30 SET PALETTE

Syntax: **Set Palette(a,b)**

where **a** = palette register number (8-15)
b = new color (0-63)

> NOTE
> Refer to the help menu for the color choices.

This is where the user can define the color used in a palette register. Where most commands effect only the current window, this command will effect all windows.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
Set Palette	(11	, 40)			

This primitive will set palette register 11 to a deep purple color.

8.9.31 PAUSE

Syntax: **Pause(a)**

where **a** = wait unit in 1/10 second increments

The user can set a pause before executing a subsequent primitive.

Example:

Command	Param 1	Param 2	Param 3	Param 4	Param 5
Pause	(200)				

This primitive will delay the execution of the definition for 20 seconds.

8.9.32 DRAW PIXEL

Syntax: **Draw Pixel(a,b)**

where **a** = x-axis coordinate (0-639) of destination
b = y-axis coordinate (0-349) of destination

The user can draw a pixel at any location.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Draw Pixel	(250	, 300)			

This primitive will draw a pixel at coordinates 250,300.

8.9.33 SET DRAW PARAMETERS

Syntax: **Set Draw Pars(a,b)**

where **a** = parameter to change, can be any of the following:
- 1 = pen size (pixels), dpv = (1,3,5, . . .)
- 2 = pen and fill pattern, dpv = (0-31)
- 3 = pen dash, dpv = (0-7)
  - 0 = solid line
  - 1 = -------- -------- --------
  - 2 = ---- ---- ---- ---- ----
  - 3 = -- -- -- -- -- -- --
  - 4 = - - - - - - - - - - - - - -
  - 5 = ---- -- -- ---- -- -- ----
  - 6 = ---- - - ---- ---- - - ----
  - 7 = ---- - - ---- - - ----
- 4 = pen write mode, dpv = (0-7) - defines how data is written on screen
  - 0 = replace
  - 1 = overlay
  - 2 = invert
  - 3 = erase
  - 4 = ~replace
  - 5 = ~overlay
  - 6 = ~invert
  - 7 = ~erase
- 5 = pen cap, dpv = (0-2) - defines how ends of lines are drawn
  - 0 = flat (ends at endpoints)
  - 1 = round (end of line rounded past endpoint)
  - 2 = square (end of line squared past endpoint)
- 6 = pen join, dpv = (0-1) - defines the corners of rectangles
  - 0 = rounded
  - 1 = squared

b = new value

The user can adjust the drawing parameters within a program. All parameters default to the lowest number values.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Set Draw Pars(	2	, 20)			

This primitive will set the pen and fill pattern to number 20.

8.9.34 PRINT SCREEN

Syntax: **Print Screen(a,b)**

where **a** = wait flag, can be either of the following:
0 = do not wait for completion
1 = wait for completion
b = window to print

The wait flag determines whether or not to wait for the print to complete before executing the next primitive.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Print Screen	(1	, INFO 1)			

This primitive will print the window whose ID is in Information Array 1, and will wait until completion of the printing before moving on to the next primitive.

8.9.35 PRINT STATUS

Syntax: **Print Status(a)**

where **a** = destination of status
0 = ready and waiting
1 = off line
2 = out of paper
3 = busy
4 = printing screen

The print status is read and stored into a specific location.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Print Status	(	REG 20)			

This primitive will read the printer status and store it in Register 20 as one of the codes listed.

8.9.36 READ SYSTEM INFORMATION

Syntax: **Read Sys Info(a,b)**

where **a** = parameter to read, can be any of the following:
- 0 = current text attributes
  - bit 0 = highlight state
  - bit 1 = underline state
  - bit 2 = blink state
  - bit 4 = strikeout state
- 1 = foreground color
- 2 = background color
- 3 = x-axis cursor coordinate
- 4 = y-axis cursor coordinate
- 5 = mode information
  - bit 0 = auto linefeed
  - bit 1 = cursor ON/OFF
  - bit 2 = scrolling
  - bit 3 = auto-wrap
- 6 = character size
- 7 = window ID
- 8 = text write mode
- 9 = pen size
- 10 = pen pattern
- 11 = pen dash mode
- 12 = pen write mode
- 13 = pen cap mode
- 14 = pen joint mode

b = destination of value read

This function allows access to system information.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Read Sys Info(	8	, REG 5)			

This primitive will store the text write mode currently invoked into Register 5.

8.9.37 SET SYSTEM INFORMATION

Syntax: **Set Sys Info(a,b)**

where **a** = parameter to set, can be any of the following:
- 0 = current text attributes
  - bit 0 = highlight state
  - bit 1 = underline state
  - bit 2 = blink state
  - bit 4 = strikeout state
- 1 = foreground color
- 2 = background color
- 3 = x-axis cursor coordinate
- 4 = y-axis cursor coordinate
- 5 = mode information
  - bit 0 = auto linefeed
  - bit 1 = cursor ON/OFF
  - bit 2 = scrolling
  - bit 3 = auto-wrap
- 6 = character size
- 7 = window ID
- 8 = text write mode
- 9 = pen size
- 10 = pen pattern
- 11 = pen dash mode
- 12 = pen write mode
- 13 = pen cap mode
- 14 = pen joint mode
- 15 = set cursor owner

b = new value

This allows the user to change attributes. This list is identical to the previous list, except for the addition of **a**-15.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Set Sys Info	(15	, INFO 1)			

This primitive will set the cursor owner as the window whose ID is on Information Array 1.

8.9.38 OPEN WINDOW

Syntax: **Open Window(a, b, c, d, e)**

where **a** = x-axis coordinate (0-79) of upper left corner
b = y-axis coordinate (0-24) of upper left corner
c = x-axis length (1-80)
d = y-axis length (1-25)
e = location of window ID

This allows the user to create a new window in which to work. All references are in text coordinates. "**E**" is set to FF if unable to open another window.

Opening a window does not cause anything to be drawn on the screen.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Open Window(	0 ,	0 ,	20 ,	10,	REG 3)

This primitive will open a window 20 units wide and 10 high at the upper left corner of the screen, and store the window ID in Register 3.

8.9.39 CLOSE WINDOW

Syntax: **Close Window(a)**

where **a** = window ID number

The OmniVU supports a limited number of windows at one time. This function deletes the window specified, making room available for new windows.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Close Window	(	REG 3)		

This primitive will close window whose ID is in Register 10.

8.9.40 LOCATE WINDOW

Syntax: **Locate Window(a, b, c)**

where **a** = window ID
b = x-axis coordinate (0-79) of upper left corner
c = y-axis coordinate (0-24) of upper left corner

This function allows the movement of an open window to another location. The upper left corner of the window will appear at the given coordinates.

The size of the window is adjusted to fit the screen at the new location. If relocating and resizing a window, it should be relocated and then resized.

The existing data on the screen within an window is NOT moved when the window is relocated.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Locate Window	(REG 3,	10	, 10)	

This primitive will move the window whose ID is in Register 3 to a location set by the upper left corner being at coordinate point 10,10.

8.9.41 RESIZE WINDOW

Syntax: **Resize Window(a, b, c)**

where **a** = window ID
b = new x-axis length (1-80)
c = new y-axis length (1-25)

This changes the size of a window.

Changing the size of a window does not change any of the data currently on the screen.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Resize Window	(REG 3,	5	, 5)	

This primitive will scale the window whose ID is in Register 3 so that it now has both a height and width of 5 units.

8.9.42 SCREEN MODES

Syntax: **Screen Mode(a)**

where **a** = mode code, can be either:
0 = graphics mode
1 = text mode

This allows the user to switch modes. The current state of all windows is lost when switching.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Screen Mode (	0)				

This primitive will set the screen mode to graphics mode.

8.9.43 WRITE OIL REGISTER

Syntax: **Write OIL Reg(a, b, c)**

where **a** = OIL register number (20-16300)
b = lsw of value
c = msw of value

This allows a user to write a value to an OIL-2 register.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Write OIL Reg	(25	, 5,	1)	

This primitive will write the value 65541 (5 + (1)65536) to OIL register 25.

8.9.44 READ OIL REGISTER

Syntax: **Read OIL Reg(a, b, c)**

where **a** = register number (20-16300)
b = destination for lsw of value
c = destination for msw of value

This allows a user to read a value from an OIL-2 register.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Read OIL Reg	(26	,	REG 1,	REG 2)

This primitive will read the contents of OIL register 26 and store the first half in Register 2 and the second in Register 1.

8.9.45 WRITE TO A TERMINAL IMAGE FILE

Syntax: **Write Term Img(a, b, c, d)**

where **a** = x-axis coordinate (0-79) of write destination
b = y-axis coordinate (0-24) of write destination
c = which byte at location is written to (0-1)
d = byte value

The user can write a byte value to a specific point in an image file. The image file is used for some terminal emulations. The image file contains an image of the characters and attributes written to the screen.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Write Term Img	(20	, 33	, 1	, PAR 1)

This primitive will write the character in Parameter Array 1 to the second half of the data image file at 20,33.

8.9.46 READ A TERMINAL IMAGE FILE

Syntax: **Read Term Img(a, b, c, d)**

where **a** = x-axis coordinate (0-79) of read source
b = y-axis coordinate (0-24) of read source
c = which byte at location is read (0-1)
d = destination

The user can read a byte value from a specific point in an image file, and store that info in any location.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Read Term Img	(10	, 33	, 1	, INFO 4)

This primitive will read the second byte at 10,33 and copy that value to Information Array 4.

8.9.47 READ COMM PORT

Syntax: **Read Comm Port(a, b, c)**

where **a** = comm port (1-4)
b = timeout time in ms (0 = no timeout)
c = data destination

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Read Comm Port	(2	, 0	, REG 1)	

This primitive will read comm port 2 and, with no timeout, write the data to Register 1.

8.9.48 READ CLOCK

Syntax: **Read Clock(a, b)**

where **a** = clock register to be read, can be any one of the following:
- 0 = seconds (0-59)
- 2 = minutes (0-59)
- 4 = hours (0-23)
- 6 = day of week (1-7, where 1 = Sunday)
- 7 = day of month (1-31)
- 8 = month (1-12)
- 9 = year (0-99)

b = value destination

The user can read the current clock value for any of a number of time/date combinations and use that information.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Read Clock	(9	, PAR 2)			

This primitive will read the current year set and write that information to Parameter Array 2.

8.9.49 SHOW DECIMAL

Syntax: **Show Decimal(a)**

where **a** = selected decimal value

This function will create an ASCII string representing the value at a. The ASCII string is stored in VSTRING.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**
Show Decimal	(25)			

This will cause VSTRING to contain the ASCII string "25".

8.9.50 SHOW TIME

Syntax: **Show Time(a, b, c, d)**

where **a** = format of the ASCII string (0 = hh:mm:ss)
b = hour
c = minutes
d = seconds

This creates an ASCII string representing the time. The ASCII string is stored in VSTRING.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Show Time	(0	, PAR 1 ,	PAR 2 ,	PAR 3)	

This primitive will get the hour, minutes, and seconds from Parameter Arrays 1, 2, and 3, respectively, and store them as an ASCII string in VSTRING.

8.9.51 SHOW DATE

Syntax: **Show Date(a, b, c, d, e)**

where **a** = format of the ASCII string, can be either:
0 = mmm dd, 19yy
1 = mm/dd/yy
b = year
c = month
d = day
e = day of the week

This creates an ASCII string representing the date. The ASCII string is stored in VSTRING.

Example:

Command	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Show Date	(0	, PAR 1 ,	PAR 2 ,	PAR 3 ,	PAR 4)

This primitive will get the year, month, date, and day from Parameter Arrays 1, 2, 3, and 4, respectively, and store them in an ASCII string.

Appendix A

MOUNTING DIMENSIONS

A.1 PANEL MOUNTING THE 8320 OmniVU

Figure A-1 shows how a panel should be cut and drilled to panel-mount an 8320 terminal. Note that the .25" holes in the panel will accommodate the 10-32 studs on the back of the terminal front panel.

A.2 RACK-MOUNTING THE 8320 OmniVU

Figure A-1 shows the locations of the studs on the back of the terminal front panel. There are 14 total. Since these studs are threaded, the terminal should be mounted in a rack in which the holes are not threaded. If the rack has threaded holes, they must be drilled out. Using a #3 bit (.213") will provide clearance for the 10-32 studs.

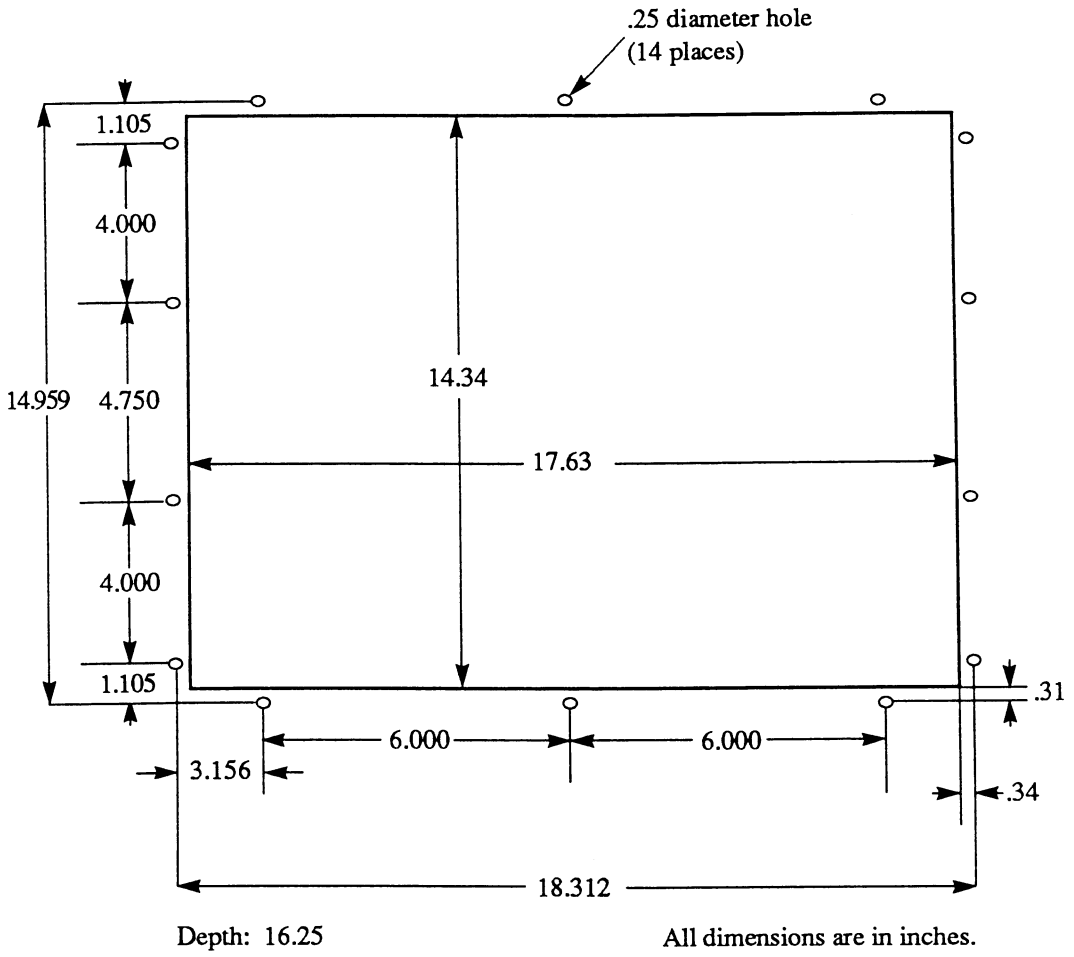


Figure A-1 Mounting Dimensions

Appendix B

QUICK REFERENCE GUIDE

B.1 INTRODUCTION

This appendix repeats some of the lists found elsewhere in the manual.

B.2 PORTS

The following lists are the connector pinouts for the 8320. They are discussed in detail within Chapter 2.

B.2.1 Printer Port

This is the uppermost connector on the ORB. It is a 25-pin D-type female connector.

Table B-1 Printer Port

PIN	SIGNAL	PIN	SIGNAL
1	STROBE	10	ACK
2	DATA0	11	BUSY
3	DATA1	12	PE
4	DATA2	13	SELECT
5	DATA3	14	AUTOFEED
6	DATA4	15	ERROR
7	DATA5	16	INIT
8	DATA6	17	SELECT-IN
9	DATA7	18-25	GND

B.2.2 **Keyboard Port**

This is a standard, 5-pin DIN keyboard connector, keyed to permit only one connection alignment.

Table B-2 Keyboard Connector

PIN	SIGNAL
1	Clock
2	Data
3	N/C
4	GND (SG)
5	+5 VDC
Shell	GND (FG)

B.2.3 **Relay Output Port**

This connector consists of three small screws to attach up to three wires, surrounded by a black plastic casing.

Table B-3 Relay

PIN	SIGNAL
1	Normally Closed
2	Common
3	Normally Open

B.2.4 **Video Connector**

This is a 15-pin D-type female connector, with the standard VGA signals. See the considerations in Chapter 2 before connecting a slave monitor.

Table 2-4 Video Connector

PIN	SIGNAL	PIN	SIGNAL
1	RED	9	N/C
2	GREEN	10	GND
3	BLUE	11	N/C
4	N/C	12	N/C
5	GND	13	HSYNC
6	RGND	14	VSYNC
7	GGND	15	N/C
8	GBLUE		

B.2.5 Serial Ports

Serial Ports 1 and 2 can be configured to either RS-232 or RS-485 (they are shipped from Xycom as RS-232). They are located below the printer port, and are both 9 pin DIN male connectors. Configuring the ports to RS-485 and/or back to RS-232 requires certain jumpers to be altered (refer to Section B.2.6). Both pinouts are listed below.

Table B-5 RS-232 Port

PIN	SIGNAL	PIN	SIGNAL
1	DCD	6	N/C
2	RXD	7	RTS
3	TXD	8	CTS
4	DTR	9	N/C
5	GND		

Table B-6 RS-485 Port

PIN	SIGNAL	PIN	SIGNAL
1	TXD-	6	RXD-
2	TXD+	7	RXD+
3	RTS-	8	CTS+
4	RTS+	9	CTS-
5	GND		

B.2.6 Serial Port Jumpers

The jumpers in Table B-7 must be in the correct positions to match the desired configuration for the Serial Ports. The locations of these jumpers is shown in Figure B-8. The factory settings are RS-232.

Table B-7 Serial Port Jumpers

JUMPER	COM1 RS-232	RS-485	JUMPER	COM2 RS-232	RS-485
J10	OUT	IN	J48	OUT	IN
J11	B	A	J32	B	A
J18	B	A	J34	B	A
J19	B	A	J35	B	A
J20	B	A	J36	B	A
J21	B	A	J37	B	A
J23	B	A	J39	B	A
J24	B	A	J40	B	A
J26	B	A	J43	B	A

When configured for RS-485, the inputs CTS and RXD may be terminated. Each signal for each port is independently enabled by a pair of jumpers. Removing the jumpers as indicated below will terminate a specific signal.

COM 1: ——— RXD: J14 and J15
CTS: J12 and J13

COM 2: ——— RXD: J54 and J55
CTS: J52 and J53

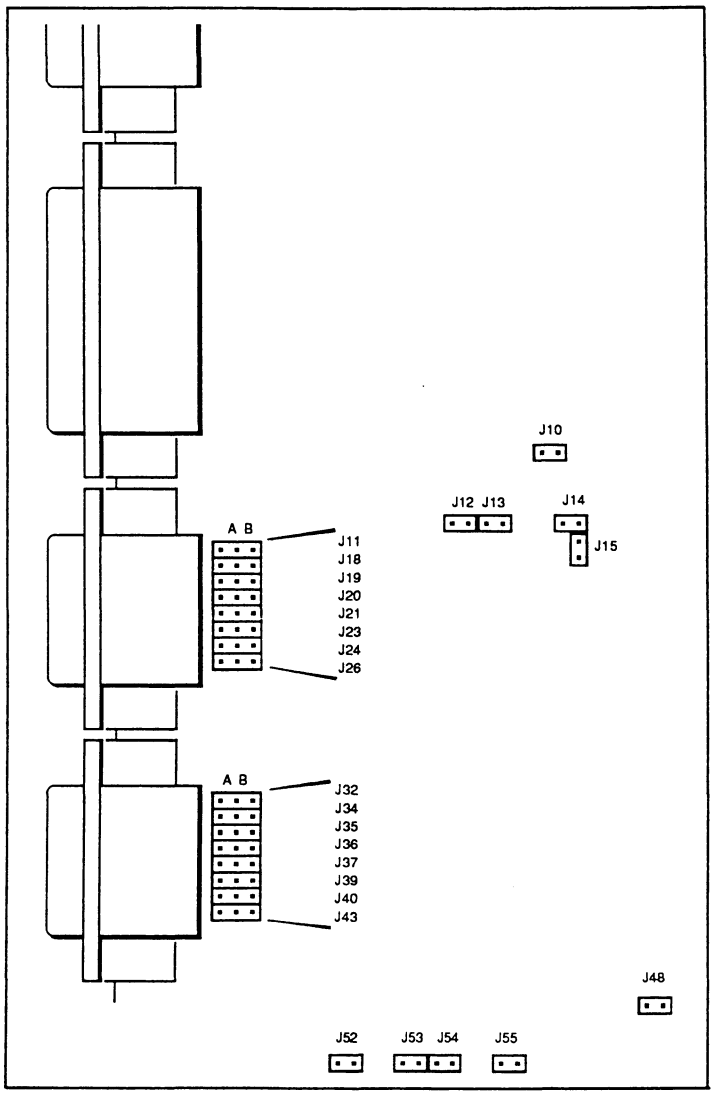


Figure B-1 Jumper Locations

B.3 REMOTE COMMANDS

The following is a listing of all of the remote commands contained in Chapter 4.

B.3.1 Default ANSI Remote Commands

Enquire
Bell
Enter Application mode
Exit Application mode
Cursor Commands
Tab Operations
Erasing Operations
Insert and Delete
Configuration Settings
Character Attributes
Report Functions
Set Drawing Parameter Value
Set Pixel
Draw Line
Draw Rectangle
Draw Filled Rectangle
Draw Oval
Draw Filled Oval
Draw Arc
Draw Filled Arc
Draw Picture
Write to Oil Register
Read Oil Register
Write to Comm Buffer
Write Time on Screen
Write Date on Screen
Create Window
Delete Window
Activate Window

B.3.2 Default Non-ANSI Remote Commands

Bell
Cursor Movements

B.4 TERMINAL PRIMITIVES

The following is a listing of all of the terminal primitives contained in Chapter 8.

B.4.1 Flow Control Primitives

End
Exit
Endif
Jump to Label(a)
Call Def.(a)
Jump to Def.(a)
Label(a)
Return
IJump to Def.(a)
Def.(a)
space
Comment(a)

B.4.2 Conditional Primitives

IfEQ(a,b)
IfNE(a,b)
IfGT(a,b)
IfGE(a,b)
IfLT(a,b)
IfLE(a,b)
IfIR(a,b,c)

B.4.3 Operational Primitives

Mov(a,b)
Add(a,b)
Sub(a,b)
Inc(a)
Dec(a)
Mul(a,b)
Div(a,b)
Mod(a,b)
Or(a,b)
And(a,b)
Shift Left(a,b)
Shift Right(a,b)
Not(a)
Read Register(a,b)
Read Parameter(a,b)
Write Register(a,b)

B.4.4 **Command Primitives**

Speaker(a,b)
To KYBD(a)
To Screen(a)
To COMM Buff(a,b)
Out COMM Port(a,b,c)
To COMM Echo(a,b)
To Printer(a,b)
Erase(a,b)
Cursor Up(a,b)
Cursor Down(a,b)
Cursor Left(a,b)
Cursor Right(a,b)
Set Attribute(a)
Tab Operation(s)
Goto XY(a,b)
Insert(a,b)
New Line
Line Feed
Back Space
Set Mode(a)
Clear Mode(a)
Draw Rectangle(a, b, c, d)
Fill Rectangle(a, b, c, d)
Draw Oval(a, b, c, d)
Fill Oval(a, b, c, d)
Draw Arc(a, b, c, d, e, f)
Fill Arc(a, b, c, d, e, f)
Draw Line(a, b, c, d)
Draw Picture(a, b, c, d, e)
Set Palette(a,b)
Pause(a)
Draw Pixel(a,b)
Set Draw Pars(a,b)
Print Screen(a,b)
Print Status(a)
Read Sys Info(a,b)
Set Sys Info(a,b)
Open Window(a, b, c, d, e)
Close Window(a)
Locate Window(a, b, c)
Resize Window(a, b, c)
Screen Mode(a)
Write OIL Reg(a, b, c)
Read OIL Reg(a, b, c)
Write Term Img(a, b, c, d)

Command Primitives cont.

Read Term Img(a, b, c, d)
Read Comm Port(a, b, c)
Read Clock(a,b)
Show Decimal(a)
Show Time(a, b, c, d)
Show Date(a, b, c, d, e)

Appendix C

ASCII CODES

Table C-1 Conversion Table

3 x 10 Keypad				4 x 7 Keypad			
KEY	ASCII	DEC	HEX	KEY	ASCII	DEC	HEX
F1	G	71	47	A	A	65	41
F2	H	72	48	PAUSE	P	80	50
F3	I	73	49	↑	DC1	17	11
F4	J	74	4A	CLEAR	<None>	132	84
F5	<None>	132	84	B	B	66	42
F6	<None>	133	85	←	DC2	18	12
A	A	65	41	↓	DC4	20	14
B	B	66	42	→	DC3	19	13
C	C	67	43	C	C	67	43
D	D	68	44	7	7	55	37
E	E	69	45	8	8	56	38
F	F	70	46	9	9	57	39
7	7	55	37	D	D	68	44
8	8	56	38	4	4	52	34
9	9	57	39	5	5	53	35
4	4	52	34	6	6	54	36
5	5	53	35	E	E	69	45
6	6	54	36	1	1	49	31
1	1	49	31	2	2	50	32
2	2	50	32	3	3	51	33
3	3	51	33	F	F	70	46
0	0	48	30	0	0	48	30
↑	DC1	17	11	.	.	46	2E
.	.	46	2E	ENTER	<CR>	13	0D
←	DC2	18	12	I	G	71	47
↓	DC4	20	14	II	H	72	48
→	DC3	19	13	III	I	73	49
ENTER	<CR>	13	0D	IV	J	74	4A