VSV21 Version 2.0
Programmer's Guide

AA-FV67B-TC

March 1986

This manual describes how the VSV21 graphics peripheral is programmed. It provides an overview of the system software, a description of the programming language and a description of commands for controlling the VSV21.

This guide is part of the VSV21 Version 2.0 document set that supersedes the VSV21 Version 1.0 document set.

Operating Systems: RSX-11M-PLUS Version 3.0
Micro/RSX Version 3.0
MicroVMS Version 4.2

Software: RSX-11M-PLUS VSV21 Version 2.0
Micro/RSX VSV21 Version 2.0
MicroVMS VSV21 Version 2.0

digital equipment corporation • maynard, massachusetts

First Edition, July 1985
Second Edition, March 1986

Using Digital's networked computer systems, this book was produced electronically by the Media, Publishing and Design Services department in Reading, England.

Printed in U.K.

CONTENTS

CHAPTER 6 CONTROL INSTRUCTIONS

CHAPTER 17 VSV11 I/O FUNCTIONS

PART IV CONFIGURING AND CONTROLLING THE VSV21

CHAPTER 18 THE VSV21 CONTROL PROGRAM (VCP)

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES

The *VSV21 Programmer's Guide* explains how to create pictures for the VSV21 graphics system and how to display the pictures using the available Digital software. This manual is designed as a guide for programmers who are developing graphics applications for the VSV21.

INTENDED AUDIENCE

The audience is intended primarily to be application programmers who are developing low-level or high-level programs for a wide range of graphics applications.

This manual describes how to build graphics display lists and write programs to execute them. A display list may be either of the following types:

- VIVID (VSV21 Instructions for Visual Display)
- VSV11 emulation

Programmers should be familiar with programming in a language supported by the host operating system.

STRUCTURE OF THIS DOCUMENT

The *VSV21 Version 2.0 Programmer's Guide* is divided into four parts, each containing one or more chapters.

- Part I (Chapter 1) provides an overview of programming for the VSV21.
- Part II (Chapters 2-14) describes how to build segmented display lists using the VIVID instruction set, and use of the VIVID Subroutine Library functions.
- Part III (Chapters 15-17) describes building and processing display lists in VSV11 emulation, and use of the FORTRAN Draw library.

- Part IV (Chapters 18 and 19) provides an account of the VSV21 Control Program (VCP), used for configuring and controlling the VSV21 to run both VIVID and VSV11 emulation. It includes a description of how to get started with VIVID, and an example of a program.

ASSOCIATED DOCUMENTS

- The *VSV21 Version 2.0 Programmer's Reference Card (AV-FV68B-TC)* is a quick-reference guide to the programming functions covered in this manual.
- The *VSV21 Version 2.0 Software Installation Guide (AA-FV66B-TC)* describes the process of installing the software on the host operating system, and of running the Installation Verification Program (IVP).
- The *VSV21 Version 2.0 User's Guide (AZ-FV70B-TC)* describes all the VSV21 facilities that are provided to the user.
- The *VSV11/VS11 Option Description (YM-C183C-00)* describes the method of creating display lists for the VSV11 and VS11 graphics systems.
- The *VSV11-M/M-PLUS Software Driver Guide (AA-J287D-TK)* describes the contents of the FORTRAN Draw package.
- The *RSX-11M/M-PLUS Executive Reference Manual (AA-L675A-TC)* and the *RSX-11M/M-PLUS and MicroIRSX Executive Reference Manual (AA-Z508A-TC)* describe how to call system services to perform input/output requests.
- The *VAX/VMS System Services Reference Manual (AA-D018C-TE)* describes how to call system services to perform input/output requests under MicroVMS.

CONVENTIONS USED IN THIS DOCUMENT

Convention	Meaning
UPPERCASE	Uppercase letters in a command or call format indicate data that must be entered as shown.
lowercase	Lowercase letters in a command or call format indicate that the data is variable.
[ ]	Brackets in a command or call format indicate that the argument is optional.
< >	Angle brackets in a call format indicate that the parameters are device-specific.

CHAPTER 1
OVERVIEW OF VSV21 SYSTEM SOFTWARE

The VSV21 is a single-board graphics module for use on Q22-bus processors. The VSV21 system software is supported by the following host processors and operating systems:

- MicroPDP-11, running
  - RSX-11M-PLUS Version 3.0 or later
  - Micro/RSX Version 3.0 or later
- MicroVAX, running MicroVMS Version 4.2 or later

The VSV21 Version 2.0 can run any one of three processes:

- VIVID Interpreter

  VIVID (VSV21 Instructions for Visual Display) is the VSV21 instruction set. It allows the VSV21 to run applications written specifically for it.

  Part II of this Guide describes how to develop applications for VIVID.

- VSV11 Emulator

  This provides emulation of a VS11/VSV11 system. VSV11 emulation allows the VSV21 to run applications written for VS11 and VSV11 systems. The VS11 and VSV11 processors can be regarded as identical for emulation purposes. In the rest of this guide VS11/VSV11 emulation is referred to as VSV11 emulation.

  Note that the VIVID and VSV11 instruction sets are not compatible.

  Part III of this Guide describes how to develop applications for the VSV21 in VSV11 emulation.

- VT220 Emulator

  This provides emulation of a subset of VT220 capabilities, giving the user access to the DCL or MCR command language and to standard program development tools. The VSV21 runs full screen VT220 emulation on power-up. Refer to the *VSV21 User's Guide (AZ-FV70B-TC)* for the method of using VT220 emulation.

1.1 GRAPHICS DISPLAY LISTS

A graphics application for the VSV21 consists of one or more programs. These programs can be written in any language supported by the host operating system. System software and layered products are available to help programmers to create pictures and output them to the VSV21 for display. These include the VIVID Subroutine Library (VSL), described in Chapter 5.

The graphics picture is described by a list of instructions and data. The instructions control output to the display and the data describes or refers to screen coordinates, colors, other instructions or peripheral devices. Some examples of what the instructions do are as follows:

- Identify subsequent data as being of a particular type, such as:
  - other instructions
  - font data
  - pixel data
  - keyboard data
  - report data
  - attribute data
- Describe an action, such as:
  - drawing a straight line or curve
  - drawing a character
  - filling an area with color
- Control display magnification, colors and other attributes

- Control input from peripheral devices

The list of instructions and data is known as a display list. The VSV21 supports two different types of display list:

- VIVID Display Lists

  These are defined by the VIVID command set and the VIVID Subroutine Library (VSL). Segmentation is a feature of VIVID display lists. They are described in Chapter 3.

- VS11/VSV11 Display Lists

  These are defined by the VSV11 instruction set and the FORTRAN Draw library. They are described in Chapter 16.

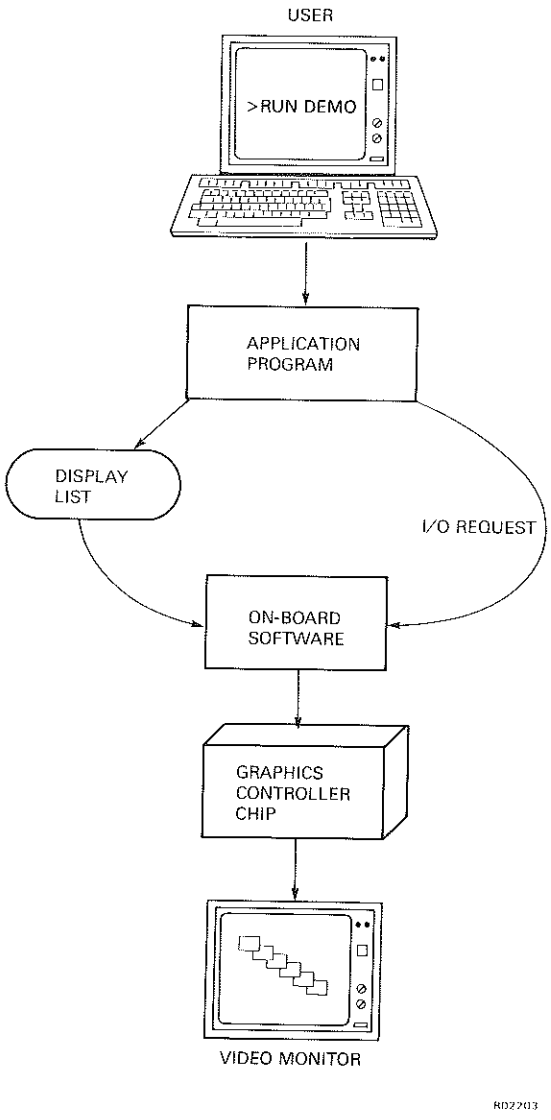


Figure 1-1 How Pictures are Created and Displayed

Display lists may be created and executed by a program running on the host processor. A program can also execute display lists which another program has created. Display lists can be stored in the host memory. VIVID display-list segments can also be stored in the memory on the VSV21 module, for faster access by the program.

The host program sends the display list to the VSV21 module, where it generates a picture in the pixel memory. The program controls the processing of display lists by issuing input/output requests. Figure 1-1 shows how an applications program creates and displays a picture using display lists and input/output requests.

1.2 PROGRAMMING INTERFACES

The system software provides two methods of building display lists and sending them to the VSV21 for display:

- Library Routines

  A high-level programming interface is provided through calls to a library of graphics subroutines. In VSV11 emulation, the FORTRAN Draw library supplied with VS11/VSV11 systems can be used. In VIVID, the VIVID Subroutine Library (VSL) can be used.

  VSL is described in Chapter 5. The method of using the FORTRAN Draw package is described in Chapter 15.

- QIO Calls

  Display lists can also be created by combining the individual components of the picture. The display list is then processed by issuing an output request to the device driver. If display lists are to be processed in VIVID, only VIVID instructions may be used. If the display list is to be processed in VSV11 emulation, only VSV11 primitives may be used.

  This low-level method is recommended for applications in which display speed and program efficiency are critical.

Both the library routines and QIO calls also allow programs to handle input from the pointing devices which are supported by the VSV21. The VSV21 supports the following pointing devices:

- Joystick
- Trackball
- Digitizing tablet

The interfaces to these devices are described in Chapter 2.

The method of creating display lists is described in Chapter 3 (VIVID) and Chapter 16 (VSV11 emulation). The input/output requests used to control display list processing are described in Chapter 4 (VIVID) and Chapter 17 (VSV11 emulation). A guide to getting started with VIVID is given in Chapter 19.

1.3 VS11/VSV11 EMULATION

The VSV21 is provided with VSV11 emulation software which enables it to run applications that have been developed for the VS11 and VSV11 systems. The VSV21 can emulate a minimum-configuration single-channel VSV11 system. The VSV11 emulation software supports the following VS11/VSV11 features:

- QIO format, identical to the VS11
- Main and auxiliary display lists
- The FORTRAN Draw package
- Joystick control

These features allow most VS11/VSV11 applications to run on the VSV21 without modification or recompilation.

The VSV21 does not support the following VS11/VSV11 features:

- Multiple channels
- 8-bit pixel data
- Hardware register programming

1.4 SYSTEM SOFTWARE COMPONENTS

Most of the VSV21 system software is supplied on a distribution kit and must be installed on the host system before any applications can be run. The procedure for performing and verifying the software installation is described in the *VSV21 Software Installation Guide (AA-FV66B-TC).*

Three categories of system software are provided. They are as follows:

- Host Software
  - VSV21 device driver
  - VSV21 Control Program (VCP)
  - Subroutine libraries
  - Diagnostics
- Resident VSV21 Software
  - Initialization and self-test
  - VSV21 system software
  - VT220 emulator
- Downloaded VSV21 Software
  - Kernel
  - Pointing device drivers
  - VIVID interpreter
  - VSV11 emulator
  - VT220 emulation code
  - VIVID default font

The relationship between the software components and the application user is shown in Figure 1-2. The following sections in this chapter describe the major components and their importance to the programmer.

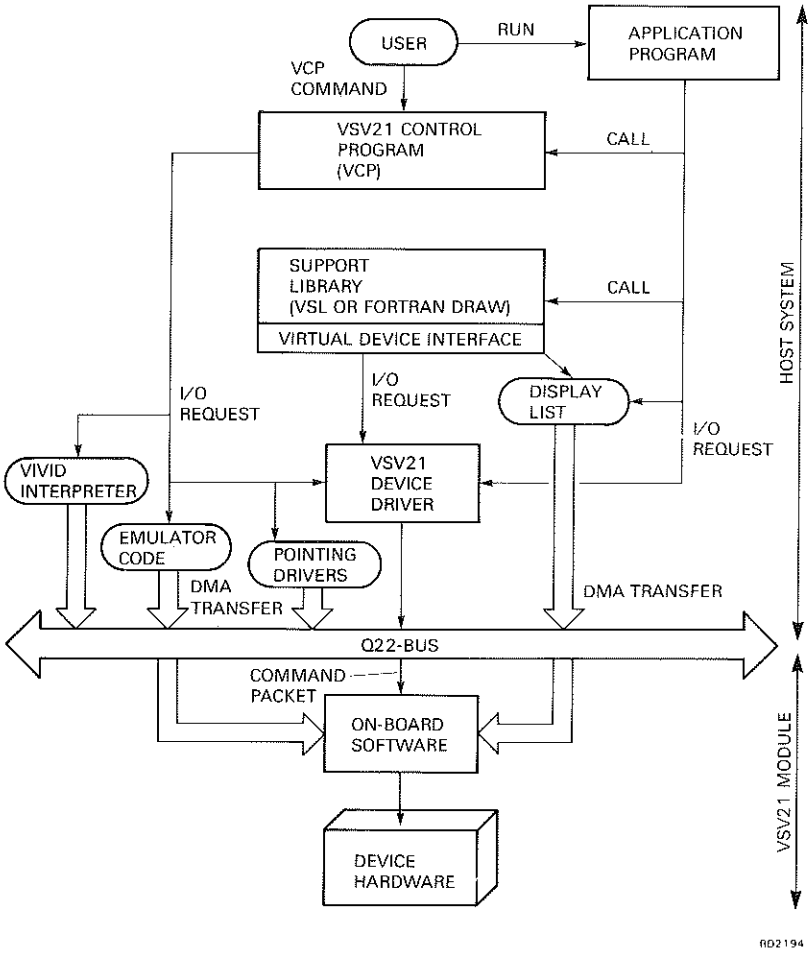


Figure 1-2 VSV21 System Software Block Diagram

1.4.1 Host Software

This consists of the programs which reside and run on the host system. These are the following:

- VSV21 device driver
- VCP (VSV21 Control Program)
- Subroutine libraries
- Diagnostics

1.4.1.1 VSV21 Device Driver – The device driver handles all communications between application programs and the VSV21 device. It receives input/output requests from programs in the form of QIO calls to system service routines in the operating system executive. The driver passes the requests to the VSV21 processor in the form of command packets, using a programmed I/O mechanism and DMA (Direct Memory Access). The QIO mechanism is described in more detail in Appendix A.

The functions provided by the VSV21 device driver are of the following types:

- Configuration
- Initialization
- Diagnostic and self-test
- Device control
- Drawing control

Before an application is run, the VIVID interpreter or VSV11 emulator must be downloaded to the VSV21 module. This process also sets the device driver to accept the VSV11 or VIVID QIO functions. The VSV11 functions are not compatible with the VIVID instruction set.

Two or more tasks can share a device under any of the operating systems in either VSV11 emulation or VIVID. The tasks can issue QIOs to the same device concurrently, and the QIOs are queued to the driver in alternating packets if necessary.

1.4.1.2 VSV21 Control Program (VCP) – The VSV21 Control Program (VCP) is a utility program which enables users, system managers and application programmers to configure and control the VSV21 device. It provides facilities to:

- Select the operating mode (VIVID, VSV11 emulation or VT220 emulation) by loading the appropriate software into the VSV21 module.
- Set the device configuration parameters, for example, to describe the peripheral devices currently attached to the serial ports.
- Show the current settings of the device configuration parameters.
- Show the current status of the device.

The VCP commands enable you to configure the VSV21 system, and to set the device into a specified operating state before running an application. By incorporating these same commands into a graphics program, you can develop self-contained applications. Users of these applications do not have to make sure that the device is set up correctly before running the application. The VCP commands are described in Chapter 18.

1.4.1.3 Subroutine Libraries – In VSV11 emulation, the FORTRAN Draw library supplied with VS11/VSV11 systems can be used. The VIVID Subroutine Library (VSL) provides a high-level interface to VIVID.

1.4.2 Resident On-board Software

The VSV21 is controlled by on-board software. This consists of software permanently stored in ROM and software downloaded from the host.

The following software is stored permanently in ROM:

- Initialization and self-test routines
- On-board driver

  This provides controlled access to the host.
- VT220 emulator

  This provides a subset of the VT220 functionality, allowing the VSV21 to be used as a system console. At system power-up, full-screen VT220 emulation is automatically provided.

1.4.3 Downloaded On-board Software

This consists of the programs and fonts which are stored on the host, but are downloaded to the VSV21 module by the VCP. They are then run by the on-board microprocessor. The following software is downloaded:

- Kernel Routine

  This controls the operation of the VSV21 and provides diagnostic facilities.

- Pointing Device Controllers

  The VSV21 uses the following pointing device controllers:

  - MSI driver, controlling the MSI trackball, joystick and mouse
  - Penny and Giles driver, controlling the Penny and Giles trackball and mouse
  - Digitizing tablet driver, controlling the digitizing tablet

- Transparent Port Driver

  This controls I/O at the fourth VSV21 port.

- VIVID Interpreter

  This enables the VSV21 to interpret VIVID instructions in display lists.

- VSV11 Emulator

  This provides emulation of a minimum-configuration single-channel VSV11 system

- VT220 Emulation Code

  This renews full-screen VT220 emulation if it has been replaced by downloading VIVID or VSV11 emulation.

- VIVID Default Font

  The VIVID default font is the DIGITAL multinational character set. Its cell size is 10 (vertical) × 8 (horizontal). The top row and righthand column are empty.

  The default font is automatically downloaded with the VIVID interpreter. It is stored in VSV21 memory as a segment with a segment ID of 10FF. If it has been deleted from the VSV21 memory, it can be downloaded separately by using either a library routine (Chapter 5) or the VCP (Chapter 18).

The following downloaded routines can be simultaneously available to an application on the VSV21:

- One pointing device driver
- The transparent port driver

- The VIVID interpreter, VSV11 emulator or VT220 emulator. Only one of these processes can run on the VSV21 at any time. The last interpreter or emulator loaded replaces the interpreter or emulator on board.

The VCP can download software with individual commands or a command procedure. For a description of the method of downloading software and fonts, refer to Section 18.3.

CHAPTER 2
OVERVIEW OF VIVID

VIVID is a set of instructions used to develop graphics applications on the VSV21 system.

The VIVID interpreter receives commands and data from application programs which run on the host processor under RSX-11M-PLUS, Micro/RSX or MicroVMS. The programs make calls to stored graphical information (display lists, described in Chapter 3) and library routines (VSL, described in Chapter 5).

VIVID is implemented as a software package running on the VSV21 processor. VIVID communicates with the host processor by means of the Q-bus interface, using VSV21 registers and DMA (Direct Memory Access).

VIVID is particularly suited to applications of the following types:

- Applications requiring high-speed execution
- Applications requiring efficient storage of images
- Implementation of graphics subroutine libraries for specific applications

2.1 USING THE VIVID INSTRUCTION SET

You can use the instruction set to program VIVID for the following range of tasks:

- Control the general operation of the VSV21 system
- Perform drawing and viewing transformations
- Set screen, color and drawing attributes
- Draw straight lines and arcs
- Fill areas
- Select text fonts

- Clear specified areas of screen
- Read and write pixel data
- Scroll, pan and zoom
- Control cursor style and visibility
- Control rubber band
- Enable interaction with keyboard and pointing devices
- Handle report packets

2.2 INSTRUCTION TYPES

Each VIVID instruction is defined in the form of a mnemonic instruction name and accompanying arguments. The number of arguments can be fixed or variable. Some instructions require no arguments. The set is made up of instructions of the following types:

- Control instructions initialize the VIVID interpreter, begin and end display list segments and control the general operation of VIVID (Chapter 6).
- Transformation instructions control the magnification of the display and the position of the window and viewport (Chapter 7).
- Global Attribute instructions set the drawing and screen display characteristics (Chapter 8).
- Drawing instructions generate the individual lines that make up an image (Chapter 9).
- Filled Figure instructions are used to paint or flood specified areas (Chapter 10).
- Text instructions control the selection of character fonts, the magnification of characters and output of text to the screen (Chapter 11).
- Area Operation instructions control such operations as scroll, clear screen, copy and pixel read (Chapter 12).
- Interactive instructions control cursor positioning and keyboard operation (Chapter 13).
- Report Handling instructions place a report packet in the current report segment (Chapter 14).

The chapters referenced in this section describe the use of each instruction and its parameters and provide a MACRO-32 example.

A brief description of each VIVID instruction is given in this chapter (Sections 2.4 through 2.12).

2.3 ACCESS TO VIVID INSTRUCTIONS

An application program can use VIVID instructions in either of the following ways:

- Display lists, called by Queue Input/Output (QIO) instructions.
- VSL functions and subroutines, called directly by the program.

2.3.1 Display Lists

A display list is a list of VIVID instructions and data which defines a picture. A VIVID display list consists of a number of segments.

A segment is a list of VIVID instructions and data which has one of six specific functions, depending on the segment type. The segment type is identified by the first instruction in the segment.

The instructions in a segment are in the form of opcodes. This first instruction in a segment is one of the first six control instructions listed in Section 2.4. It identifies the segment as one of six types:

- Instruction segment, consisting of VIVID instructions, stored as opcodes and parameters.
- Font segment, consisting of a set of character cell definitions.
- Pixel segment, consisting of a pixel data map.
- Keyboard segment, consisting of data input from the keyboard.
- Report segment, consisting of report packets.
- Attribute segment, consisting of global attribute information.

Display lists and segments are described in Chapter 3.

QIO calls are used to control the processing of display lists. The QIO calls perform the following functions:

- Allocate a segment area on the host
- Define a segment in a host-allocated area
- Download a segment to the VSV21 processor
- Delete a segment from the host memory or VSV21 memory
- Start, stop and resume segment execution
- Define report processing requirements

The use of QIOs is described in Chapter 4.

2.3.2 VIVID Subroutine Library (VSL)

VSL is a library of functions and subroutines (see Chapter 5). VSL functions and subroutines control the segments, execute segment or drawing commands and handle replies from VIVID. VSL automatically generates VIVID drawing instructions and parameters from the VIVID instruction set. You can also call VSL functions and subroutines to do the following:

- Initialize display list processing
- Start or end a segment
- Execute a segment
- Save or restore a segment on disk
- Load a segment to VSV21 from a disk or the host memory
- Delete a segment
- Get keyboard input
- Get a report
- End display list processing

2.4 CONTROL INSTRUCTIONS

Control instructions regulate the operation of the VIVID interpreter. The set of control instructions is as follows:

- START_INSTRUCTION_LIST

  Identifies the contents of the segment as display instructions.

- START_FONT

  Identifies the segment contents as a font.

- START_PIXEL_DATA

  Identifies the segment contents as pixel data.

- START_KEYBOARD_DATA

  Identifies the segment contents as keyboard input.

- START_REPORT_DATA

  Identifies the segment contents as reports.

- START_ATTRIBUTES_DATA

  Identifies the segment contents as global attributes data.

- INITIALIZE

  Causes one or more graphics control item to be reset to a default value.

- CALL_SEGMENT

  Transfers execution to the identified segment in host memory or VSV21 memory.

- SAVE_ATTRIBUTES

  Writes the current attributes to a stack in VSV21 memory.

- RESTORE_ATTRIBUTES

  Reads the latest attributes from the stack in the VSV21 memory. SAVE_ATTRIBUTES and RESTORE_ATTRIBUTES allow attributes in a nested segment to be changed and recovered before control is returned to the calling segment.

- DUMP_ATTRIBUTES

  Saves the current set of global attributes in a segment.

- RECOVER_ATTRIBUTES

  Recovers specified attributes from a segment.

- START_ATTRIBUTES_DATA

  Identifies the segment as holding saved attributes.

- DISPLAY_WAIT

  Delays execution of the next display instruction for a specified time.

- NO_OPERATION

  Causes no operation to be performed. This instruction may be used during program testing. Patching a segment may result in gaps that can be filled with NO_OPERATION instructions.

- STOP_DISPLAY

  Stops the processing of segments and returns control to the application program.

- CREATE_SEGMENT

  Creates an empty segment in the VSV21 memory.

- SEGMENT_RETURN

  Marks the end of a segment. Control is returned to the user program or to the invoking segment level.

- JUMP_RELATIVE

  Causes a jump in segment execution by a specified relative offset.

- DISPLAY_REPEAT

  Defines the start of a loop in the segment. Loops can be nested up to 32 levels deep.

- DISPLAY_END_REPEAT

  Defines the end of a loop in the segment.

2.5 THE VIEWING TRANSFORMATION

To display a stored picture, VIVID uses the following areas of screen and memory:

- VIVID Address Space

  VIVID defines picture data in virtual coordinates space called VIVID Address Space (VAS). VAS holds a picture in the form of Cartesian (X,Y) coordinates. The range of both X and Y is ± 32K.

- The Screen Dimensions

  You can define the scale and aspect ratio of the display by defining the screen dimensions. These are the number of VAS units to be displayed in both X and Y directions.

- The VIVID Window

  The VIVID window is the area of VAS which will be mapped to the viewport. You can set the origin (lower left corner) of the window. The extent of the window can be input as a parameter or determined by the screen dimensions.

- The Viewport

  The viewport is an area of the screen into which the window is projected.

The relationships between these areas are summarized in Figure 2-1.

Images can be entered into VAS in either of the following ways:

- Untransformed

  The picture data in the segment is preceded by a DRAWING_VAS instruction. This identifies the data as actual VAS units and disables the current magnification and translation factors.

- Transformed

  The picture data in the segment is preceded by a DRAWING_TRANSFORM instruction. This indicates that the data is in units which require transformation before display, and enables the magnification and translation factors.

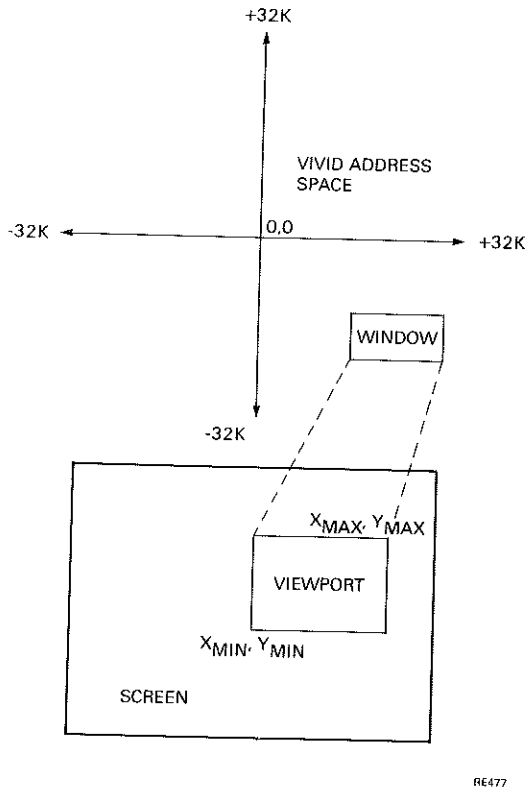


Figure 2-1 Relationships between VIVID Address Space, Window, Viewport and Screen

The VIVID drawing transformation can be regarded as being in two stages:

1. Transforming the units given in the segment to VAS units.
2. Transforming the VAS units to screen display units.

2.5.1 Transforming Input Data to VAS Units

You can transform the input data to VAS units by using the following VIVID instructions:

- DRAWING_MAGNIFICATION

  Defines the magnification of the elements being entered to VAS, in both absolute and relative drawing operations.

- DRAWING_TRANSLATION

  Defines the point which corresponds to (0,0) in subsequent drawing instructions.

- DRAWING_TRANSFORM

  Applies the current magnification and translation to subsequent drawing instructions.

- DRAWING_VAS

  Disables the current magnification and translation. Subsequent drawing instructions have VAS units and origin.

2.5.2 Transforming VAS Units to Screen Display Units

You can project the picture data stored in VAS to the screen by using the following VIVID instructions:

- SCREEN_DIMENSIONS

  Defines the number of logical pixels displayed in each dimension of the screen. This allows you to define the aspect ratio and resolution of the display.

- WINDOW_ORIGIN

  Sets the window origin to a VAS coordinate. This position is the lower left corner of the window.

- ZOOM_FACTOR

  Defines magnification factors for zoom magnification of the window in X and Y directions. This allows you to magnify the picture.

- SET_VIEWPORT

  Defines the area of the screen used to display the image. The window is mapped to the viewport using this instruction, along with either the WINDOW_ORIGIN and ZOOM_FACTOR instructions or the SET_WINDOW instruction.

- SET_WINDOW

  Defines a window in VAS to be projected on to the viewport. This is equivalent to a combination of WINDOW_ORIGIN and ZOOM_FACTOR instructions.

2.6 GLOBAL ATTRIBUTE INSTRUCTIONS

Global attribute instructions describe how objects will be drawn. The commands are as follows:

- SCREEN_BLINK

  Enables or disables blinking.

- BLINK_TIMING

  Sets the blink timing.

- SCREEN_BLANK

  Enables or disables screen blanking. Drawing is faster when the screen is blank.

- FOREGROUND_COLOR

  Sets the foreground color to be used for drawing and text.

- BACKGROUND_COLOR

  Sets the background color to be used for drawing and text.

- NORMAL_COLORS

  Sets up to 16 colors in terms of Color Look-up Table (CLUT) index and relative intensities of red, green and blue. The CLUT is described in Chapter 18.

- BLINK_COLORS

  Defines CLUT indices and alternate colors for the blink colors.

- BLINK_COUNT

  Defines the number of colors that blink when blink is enabled.

- DRAWING_MODE

  Sets the drawing mode as follows:

  - foreground and background, foreground only or background only
  - conditional replacement of display image

- LINE_TEXTURE

  Defines the line texture as a string of foreground and background bits.

- AREA_TEXTURE

  Defines the area texture as a matrix of foreground and background bits.

2.7 DRAWING INSTRUCTIONS

Many drawing instructions operate in two modes:

- Absolute

  This specifies a position as an absolute location in VAS. Absolute instructions have the suffix _ABS.

- Relative

  This specifies a relative position, defined in terms of displacement from the current position. Relative instructions have the suffix _REL.

The set of drawing instructions is as follows:

- MOVE_ABS
  MOVE_REL

  Moves to the specified position. Nothing is drawn.

- MOVE_TO_CURSOR

  Moves the current drawing position to the cursor position.

- LINES_ABS
  LINES_REL

  Draws lines from the current position to specified points.

- POLYMARKS_ABS
  POLYMARKS_REL

  Draws the specified marker character at the specified points.

- ARCS_ABS
  ARCS_REL

  Draws the specified sequence of circular arcs.

- ELLIPSE_ARCS_ABS
  ELLIPSE_ARCS_REL

  Draws the specified sequence of elliptical arcs.

- RECTANGLE_ABS
  RECTANGLE_REL

  Draws a rectangle defined by a vertex at the current position and the specified diagonal vertex.

- ELLIPSE

  Draws an ellipse with a specified aspect ratio and major axis whose center is the current position.

- CIRCLE

  Draws a circle of a specified radius whose center is the current position.

- DOT

  Draws a dot at the current position. The point defined by the terminating position in the instructions in this section is not drawn on the screen. You must draw it explicitly with a DOT instruction.

2.8 FILLED FIGURE INSTRUCTIONS

A filled figure is an area of the screen which is filled by a pattern. The instruction used to fill the area determines the boundary conditions, and may be as follows:

- FILLED_RECT_ABS
  FILLED_RECT_REL

  Draws a filled rectangle from a vertex at the current position to the diagonal vertex you specify. The rectangle is filled with the area texture pattern.

- FLOOD_AREA

  Uses the area texture pattern to fill the area defined by a specific edge color and containing the current position.

- PAINT_AREA

  Uses the area texture pattern to fill an area of specific color containing the current position.

2.9 TEXT INSTRUCTIONS

The VIVID text instructions deal with setting up and using fonts to display alphanumeric characters. A VIVID font is a set of indexed cells which contain pictorial information coded by pixel. The set of text instructions is as follows:

- INITIALIZE_FONT

  Initializes the specified segment as a font.

- SET_FONT

  Sets the current font.

- LOAD_CHAR_CELL

  Loads a numbered character cell into the font using pixel data.

- CELL_OBLIQUE

  Defines whether subsequent cells are to be written normally or in italic (sloped) form.

- CELL_ROTATION

  Defines the angle at which cells are to be written to the display.

- CELL_SIZE

  Defines the display image size and the displacement of the stored font cell within the display cell.

- CELL_MAGNIFICATION

  Defines the factors by which the cell is to be magnified vertically and horizontally.

- CELL_MOVEMENT

  Defines the vertical and horizontal displacement from the end of one character cell to the final position.

- DRAW_CHARS

  Displays the characters specified by the accompanying cell numbers. The cell number is specified by one parameter word. This allows 16-bit addressing, providing address space for a font of up to 64K characters.

- DRAW_PACKED_CHARS

  Displays the characters specified by the accompanying cell numbers. Two cell numbers are specified by a parameter word. This provides 8-bit addressing, so a font of up to 256 cells may be referenced.

2.10 AREA OPERATION INSTRUCTIONS

Area operation instructions perform operations on pixel memory.

- CLEAR_SCREEN

  Clears the displayed image.

- CLEAR_VIEWPORT

  Clears the viewport.

- SCROLL_VIEWPORT

  Moves the data vertically, horizontally or diagonally to the position you define within the viewport. Data moved outside the viewport is lost.

- PIXEL_READBACK

  Reads a display image area to a specified segment. The segment may be used for pixel write operations.

- PIXEL_WRITE

  Writes a specified segment containing pixel data to the display. The image is clipped by the viewport.

- FAST_PIXEL_WRITE

  Writes a specified segment containing pixel data to the display from the host or VSV21 memory. The viewport is ignored.

- FAST_PIXEL_MODIFY

  Performs a specified logical operation between the contents of a specified pixel data segment and the image data. The viewport is ignored.

- SELECTIVE_CLEAR

  Clears a defined area, depending on the outcome of a logical operation between a parameter and defined image data.

- COPY_ABS
  COPY_REL

  Copy a specified area to a different area with a specified vertex and attitude.

2.11 INTERACTIVE INSTRUCTIONS

Interactive instructions determine cursor characteristics, switch interrupt facilities and keyboard input.

The set of interactive instructions is as follows:

- CURSOR_STYLE

  Sets the cursor style to the shape specified by the parameters, or to one of the default cursor styles. The default styles are small cross-hair and full-screen cross-hair. The parameters define pixel data.

- POSITION_CURSOR

  Sets the cursor to the position defined by the parameters.

- CURSOR_VISIBILITY

  Defines whether or not the cursor is visible.

- RUBBER_BAND

  Defines rubber-band characteristics (none, linear or rectangle) and the base point.

- SWITCH_REPORT_ENABLE

  Enables a facility for sending a pointing device report to the host processor.

- SWITCH_REPORT_DISABLE

  Disables switch reports.

- AUTOSWITCH

  Simulates a switch interrupt after a specified time. Display list processing continues.

- WAIT_SWITCH

  Causes the processor to wait for one of a specified range of switch interrupts before executing the next VIVID instruction.

- MATCH_ENABLE

  Enables a report facility. When subsequent drawing meets the cursor position, a report including the ID of the segment and the display list instruction which caused the pixel at the current position to be drawn is sent to the report segment.

- MATCH_DISABLE

  Disables the match report facility.

- ACCEPT_KEYBOARD_INPUT

  Passes input from the keyboard into a specified segment. Input from the keyboard can continue until one of the following occurs:

  - the specified termination character is received
  - the buffer is full
  - a specified number of characters has been read

  Input may be echoed to the screen.

- START_KEYBOARD_INPUT

  Begins keyboard input for asynchronous processing. Input is directed to the AST on RSX-11M-PLUS and Micro/RSX systems, and to the mailbox on MicroVMS systems. The input is echoed to the screen.

- STOP_KEYBOARD_INPUT

  Disables keyboard input for asynchronous processing.

2.12 REPORT HANDLING INSTRUCTION

The application program can request reports on the following:

- Current drawing position in VAS
- Current cursor position in VAS
- Current text parameters
- Current global attribute parameters
- Current transformation parameters

- Screen format
- Space available for downloaded segments
- IDs of segments in the VSV21 or host memory
- VIVID version number
- Nested segment calls to the current segment

VIVID can be programmed to ignore certain classes of report, or to direct them to a mailbox in VMS or to a report segment. This allows you to optimize the performance of the VSV21 and to accept input from peripheral devices.

If a report segment has been defined, the VIVID interpreter will initialize it at the start of segment processing. If a report is generated during the processing of a QIO request, it is written to the current report segment. If no report segment, AST or mailbox is defined, the report is lost. The report segment includes details of events occurring during display segment processing, such as input from peripheral devices.

CHAPTER 3
DISPLAY LISTS

A display list is a list of VIVID instructions and data which defines a picture. A VIVID display list consists of a number of segments.

A segment is a list of VIVID instructions and data which have one of six specific functions, depending on the segment type. The instructions in a segment are in the form of opcodes. The segment type is identified by the first instruction in the segment.

The first word of a segment defines the segment type. A segment can be one of six types:

- Instruction segment, consisting of VIVID instructions, stored as opcodes and parameters
- Font segment, consisting of a set of character cells
- Pixel data segment, consisting of a pixel data map
- Keyboard segment, consisting of data input from the keyboard
- Report segment, consisting of report packets
- Attribute segment, consisting of global attributes data

The segment types are described in Section 3.2.

You can pass segments to the task by one of the following means:

- Using a QIO call to communicate directly with the VIVID interpreter. The VIVID I/O functions are described in Chapter 4.
- Calling functions from the VIVID Subroutine Library (VSL) described in Chapter 5.

Do not mix these methods in an individual application; use only QIO calls or only VSL.

3.1 IDENTIFYING SEGMENTS

When you are building a segment, you give it an ID number. This allows VIVID to access the segment individually. The segment ID is stored in the second word of the segment.

VIVID instructions can identify segments in two ways:

- ID number – segments stored on the host are given an ID number as a parameter within a QIO call.
- Address – each segment downloaded to the VSV21 is given an address as a parameter in the downloading QIO call.

The segment ID is in two parts: class and number within class. This structure allows you to group similar segments in a class to facilitate storing and deleting groups of segments (Section 3.2).

It occupies two bytes, as follows:

BYTE	CONTENTS	RANGE
MSB	Class	Host : 1 to 32 VSV21: 1 to 16
LSB	Number	1 to 255

Two segment class numbers are reserved as follows:

- Class 16 is used for multinational font segments
- Class 32 is used for VSL segments

You should avoid using these class numbers when building your own segments.

You can delete segment ID numbers. Deleting the IDs of host segments makes them inaccessible to the VSV21 but does not otherwise affect the segments.

3.2 STORING AND DELETING SEGMENTS

3.2.1 Storing Segments in the Host Memory

VIVID allows storage of up 512 segments in the host memory. Host-resident segments are stored in a contiguous area of memory, known as the display area. You use a QIO or a VSL function to identify the display area to the host device driver for the duration of the task. When a VSL function transfers segments to the host memory, VSL automatically defines the display area.

It may be convenient to store segments in the host memory if they are not used frequently enough to justify downloading them to the VSV21, or if they would not fit in the VSV21 memory space (Section 3.2.2).

3.2.2 Storing Segments in the VSV21 Memory

You can download segments from the host to the VSV21 by three methods:

- Issuing QIOs (Chapter 4)
- Calling VSL functions (Chapter 5)
- Using VCP (Chapter 18)

A downloaded segment remains accessible to all tasks until you delete it.

The advantages of downloading a segment to the VSV21 are as follows:

- The VSV21 can access the segment more quickly
- The load on the Q-bus is reduced
- Space may be released in the host memory

The space available for storing downloaded segments is part of a linear memory of 64K words. This space is also occupied by the VIVID interpreter, downloaded drivers and their data areas, and saved attributes. The space available for segments is the 29K words remaining when the drivers and attributes have been stored (Figure 3-1). The procedure for downloading the VIVID interpreter and the drivers is given in Section 18.3.

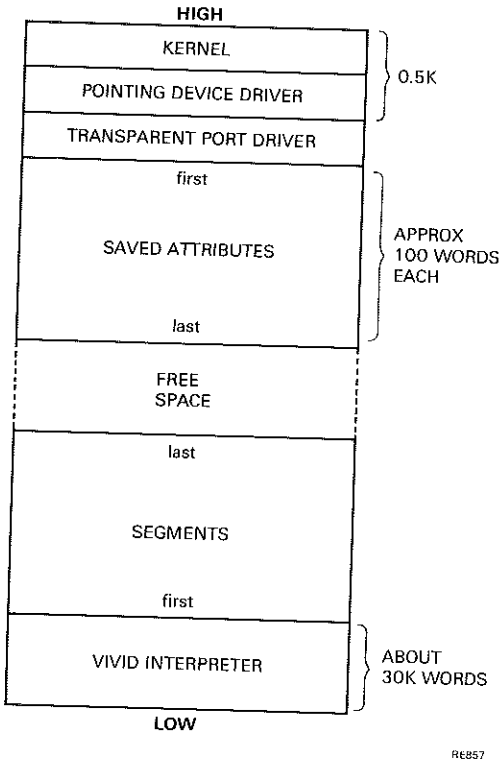


Figure 3-1 VSV21 Memory Space

If you delete a segment, no extra space is effectively created unless the segment was at one end of the memory space. If the space available is not enough for the segment you are downloading, the VSV21 automatically compresses the stored segments to maximize the free space. However, to minimize compression delays, you should download segments of long-term requirement before those of short-term requirement.

About 29K words of memory are available for storing segments and saved attributes. If this is not enough to store all the segments you have defined, it is best to download the segments which the program references most frequently. This maximizes processing speed.

3.2.3 Deleting Segments

You can delete an specified segment from VSV21 memory or host memory by using a QIO call or a VSL subroutine and the segment ID. To delete all segments of a particular class from memory, use a delete command with that class number and a segment number of zero. The use of this command is described in Chapter 4.

Deleting a host segment makes the segment inaccessible to the VSV21, but does not delete the data from the host memory.

Deleting on-board segments allows the VSV21 to recover memory space. A segment can be deleted from the VSV21 memory in two ways:

- Delete the addresses of the segment. This frees the VSV21 memory space.
- Define or download a segment which has the same number as the existing segment. This replaces the existing segment.

3.2.4 The VIVID Default Font

The VIVID default font is downloaded automatically with the VIVID interpreter. It is stored in VSV21 memory as a segment, with segment ID of 10FF. It can be deleted to free VSV21 memory space.

3.3 SEGMENT TYPES

There are six types of segment. The first word of the segment defines the segment type. The second word contains the segment identifier. The third word contains the segment length in bytes (Figure 3-2). The contents of the remaining words depend on the segment type.

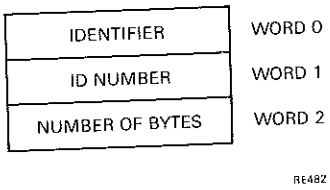


Figure 3-2 Contents of the First Three Words of a Segment

3.3.1 Instruction Segment

Identifier: START_INSTRUCTION_LIST

Each instruction segment holds VIVID instructions and associated data. The first word of a VIVID instruction has the instruction code in the most significant byte and the length (number of words) of the associated parameter list in the least significant byte.

If the number of parameters exceeds 254 or is variable, set the number of parameters to 255 and terminate the parameter list with the END_PARAMETERS delimiter (value hex 8000, decimal 32768).

3.3.2 Font Segment

Identifier: START_FONT

The START_FONT instruction and parameter list are set up automatically when a font is created with the INITIALIZE_FONT instruction. You must include it if you are setting up a font segment for transmission to the VSV21 in any other way.

Specify the number of words used to define each cell in the font with a START_FONT parameter. There may be up to 16 words. Each word represents a row of 16 pixels. Each bit represents one pixel. The pixel state (foreground or background) is indicated by setting the bit to 1 or 0 respectively. If the cell is to be less than 16 pixels wide, the unused pixels are ignored.

Pixel rows in a cell are represented by words in inverse order: the first word in the segment represents the bottom pixel row on the display, and bit 0 represents the leftmost pixel of the cell.

The default font (segment ID 10FF) is stored in VSV21 memory automatically when the VIVID interpreter is downloaded. The segment which defines the default font is listed in Appendix B.

3.3.3 Pixel Data Segment

Identifier: START_PIXEL_DATA

The START_PIXEL_DATA instruction is set up automatically when pixel data is written using the PIXEL_READBACK instruction.

The pixel data map is the form in which a rectangular area of the display image is stored. The rectangle is defined in terms of a vertex origin (X,Y) and the displacement (DX,DY) from that vertex.

DX is the number of segment words where pixels are stored. DY is the number of pixel rows. Pixels are stored four per word. Of the pixels stored in any word, the leftmost pixel on the display image is stored in bits 15 to 12, irrespective of the direction given by DX.

Example: DX and DY positive

If both DX and DY are positive, the defined vertex is at the lower left corner of the rectangle. The pixel data is stored in the segment beginning with the left pixel in the bottom row and continuing left to right and bottom to top. In the following examples, the position of each number corresponds to the pixel position; the number itself refers to a storage position in the data segment. The defined vertex is at the asterisk (*).

```
19  20  21  22  23  24  -->

10  11  12  13  14  15  16  17  18

 1   2   3   4   5   6   7   8   9
*
```

Example: DX negative, DY positive

If DX is negative and DY is positive, the vertex is at the lower right corner of the defined rectangle, and the pixels are stored in the segment as follows:

```
<--  24  23  22  21  20  19

18  17  16  15  14  13  12  11  10

 9   8   7   6   5   4   3   2   1
                                  *
```

3.3.4 Keyboard Input Segment

Identifier: START_KEYBOARD_DATA

The START_KEYBOARD_DATA instruction is initialized when keyboard data input is initiated by an ACCEPT_KEYBOARD_INPUT instruction.

The end pointer and completion flags are parameters of the START_KEYBOARD_INPUT instruction. As the keyboard data is written to the segment, the end pointer is updated. When input is complete, the completion flag is set.

Two keyboard input characters are stored in each word. Characters are represented as they are in VT220 emulation.

The extent of the segment is stored in a START_KEYBOARD_DATA parameter.

3.3.5 Report Segment

Identifier: START_REPORT_DATA

The START_REPORT_DATA instruction is set up automatically by a Define Reporting QIO. It can be set up by the Start Segment Exection QIO if the report segment is already defined. Reports are written to the segment during display list processing.

A segment can be defined as a reporting segment and initialized by any of three methods:

- Define a new segment as a reporting segment as follows:

    1. Define a segment using the Define Segment QIO.
    2. Define the segment as a report segment and set up the report mask using the Define Reporting QIO.
    3. Initialize the segment by issuing a Start Segment Execution call without any reporting parameters.

- Define a new segment as a reporting segment as follows:

    1. Define a segment using the Define Segment QIO.
    2. Define the segment as a reporting segment and initialize it by issuing a Start Segment Execution QIO with the optional reporting parameters.

- Initialize a previously-defined reporting segment as follows:

    If a segment has already been defined as a report segment a Start Segment QIO without the optional reporting parameters is sufficient to initialize it.

3.3.6 Attribute Segment

Identifier: START_ATTRIBUTES_DATA

The START_ATTRIBUTES_DATA instruction is automatically set up in a segment by a DUMP_ATTRIBUTES instruction specifying the segment ID.

If the segment does not already exist, it is created on board the VSV21.

If the segment exists and is big enough to hold the attributes, the START_ ATTRIBUTES_DATA instruction is written to the beginning of the segment.

If the segment exists but is too small to hold the attributes, an error results.

CHAPTER 4
VIVID I/O FUNCTIONS

The application program running on the host processor uses QIO calls to communicate with the VSV21 device driver. The QIO calls described in this chapter may be used to carry out the following operations:

- Attach and detach the VSV21 device
- Allocate and release display areas on the host
- Define, delete and load segments
- Start, stop and resume execution of segments

You can download segments from the host memory to the VSV21 memory by using QIO calls. An individual segment is identified for access by its segment address (Section 3.1).

VIVID allows storage of up to 512 defined segments in the host memory. The number of segments which can be stored in the VSV21 memory is limited by the memory space occupied by downloaded drivers and saved attributes. A total of about 29K words of VSV21 memory is available for segments. The VSV21 memory is described in Section 3.2.

You can delete individual segments on the VSV21 by using QIOs. This process frees the VSV21 memory space if the segments remaining were downloaded before the deleted segments. If there is insufficient space, the download operation performs a compress. To minimize processing time, download the long-term segments first.

An introduction to the QIO call mechanism is given in Appendix A.

You can issue a VIVID QIO call from a program running under RSX-11M-PLUS or Micro/RSX or MicroVMS. The format of a VIVID QIO call depends on:

- The host operating system:
  - RSX-11M-PLUS or Micro/RSX
  - MicroVMS
- The programming language

  This can be MACRO-11 or MACRO-32, or any high-level language for which the host has a compiler.

Each call includes a function and a list of parameters. The error and warning return codes are described in Section 4.3.

4.1 THE QIO FUNCTIONS AND PARAMETERS

Each of the VIVID QIO functions is described in this section. For each function, the version for RSX-11M-PLUS and Micro/RSX is given first, followed by the MicroVMS version, each with its associated parameters. Note that the functions which attach and detach the VSV21 are not used under MicroVMS. The contents of the I/O status block and the error and warning codes are given in Section 4.3.

An example of a VIVID MACRO-32 program which includes QIO calls is given in Chapter 19.

4.1.1 Attach VSV21 Device – IO.ATT

This function attaches a VSV21 unit to the task.

RSX-11M-PLUS and Micro/RSX

Function: IO.ATT

Octal value: 1400

Parameters: None

MicroVMS

This function is not used under MicroVMS.

4.1.2 Detach VSV21 Device – IO.DET

This function detaches the VSV21 unit from the task.

RSX-11M-PLUS and Micro/RSX

Function: IO.DET

Octal value: 2000

Parameters: None

MicroVMS

This function is not used under MicroVMS.

4.1.3 Allocate Display Area – IO.ADA and IO$__ALLOCATE

This allocates a display area for segments in the host memory. Only one display area can be allocated to the device. A later Allocate call releases the already-allocated area and allocates the newly-defined area.

If shared device access is required under MicroVMS, the allocated display is a shared global section. Applications using this allocated display area must map the section in identical virtual address space.

RSX-11M-PLUS and Micro/RSX

Function: IO.ADA

Octal value: 7400

Parameters: p1 = virtual address of area
p2 = number of bytes in display area
p3 = partition name (RAD50) characters 1 to 3
p4 = partition name (RAD50) characters 4 to 6

Two of these parameters, either p1 and p2 or p3 and p4, can be specified. The remaining two must be set to zero.

MicroVMS

Function: IO$_ALLOCATE

Hex Value: 3C

Parameters: p1 = virtual address of area
p2 = number of bytes in area

4.1.4 Release Display Area – IO.RDA and IO$_RELEASE

This releases a display list area which has been allocated using the Allocate function (Section 4.1.2). References to host segments are deleted and all display list processing stops.

RSX-11M-PLUS and Micro/RSX

Function: IO.RDA

Octal value: 10000

Parameters: None

MicroVMS

Function: IO$_RELEASE

Hex value: 3D

Parameters: None

4.1.5 Define Segment – IO.DFS and IO$_DEFSEG

This function defines a segment which is already in the host display area by entering its details on the VSV21 segment map. No download takes place.

The segment ID number must be set up in word 1 of the segment before the QIO is issued. The number of bytes in the segment is stored in word 2 of the segment (Figure 4-1). In general, this should be maintained by the application program. VIVID writes the length in bytes given by this function into word 2 of the segment. If you want to change the segment length, you must redefine the segment.

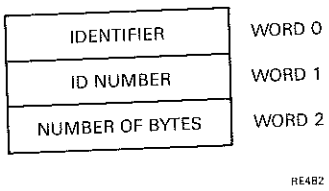


Figure 4-1 Format of the First Three Words of a Defined Segment

RSX-11M-PLUS and Micro/RSX

Function: IO.DFS

Octal value: 5400

Parameters: None

MicroVMS

Function: IO$__DEFSEG

Hex value: 3A

Parameters: p1 = virtual address of segment
p2 = length of segment in bytes

4.1.6 Load Segment – IO.LSG and IO$__LOADSEG

This function downloads a segment from the host to the VSV21 device and enters segment details into the segment map. Any segment with the same ID as the new segment is automatically deleted.

The complete segment must be downloaded without the intervention of any other QIO. The system recognizes the end of the transfer when the number of words transferred is equal to, or greater than, the segment length stored in the third word of the segment. The length stored here determines the total amount of space allocated to the segment.

RSX-11M-PLUS and Micro/RSX

Function: IO.LSG

Octal value: 6000

Parameters: p1 = virtual address of segment
p2 = number of bytes in transfer block
p3 = block sequence number

MicroVMS

Function: IO$_LOADSEG

Hex value: 3B

Parameters: p1 = virtual address of segment
p2 = number of bytes in transfer block
p3 = block sequence number

4.1.7 Delete Segment – IO.DSG and IO$_DELSEG

This deletes a segment from the host memory. If the segment has been downloaded to the VSV21, the space there is freed.

RSX-11M-PLUS and Micro/RSX

Function: IO.DSG

Octal value: 11000

Parameters: p1 = segment ID

MicroVMS

Function: IO$_DELSEG

Hex value: 2F

Parameters: p1 = segment ID

4.1.8 Start Segment Execution – IO.SSE and IO$__STARTSEG

This function starts the execution of a single predefined segment. The time-out parameter is optional; the default value is five seconds.

RSX-11M-PLUS and Micro/RSX

Function: IO.SSE

Octal value: 11400

Parameters: p1 = segment ID
p2 = time-out period in seconds
p3 = report segment ID. This segment must already be defined. See Section 3.3.5 for the definition procedure.
p4 = reporting mask. This is described in Section 4.1.11.

MicroVMS

Function: IO$__STARTSEG

Hex value: 3E

Parameters: p1 = segment ID
p2 = time-out period in seconds
p3 = report segment ID
p4 = reporting mask

4.1.9 Stop Display List Execution – IO.STP and IO$__STOP

This function stops display list execution when the current display list instruction is completed. Processing a display list instruction is not interrupted, except for DISPLAY__WAIT, WAIT__SWITCH and ACCEPT__KEYBOARD__INPUT instructions.

RSX-11M-PLUS and Micro/RSX

Function: IO.STP

Octal value: 3400

Parameters: None

MicroVMS

Function: IO$_STOP

Hex value: 35

Parameters: None

4.1.10 Resume Execution – IO.REX and IO$_CONTINUE

Using this function, display list execution is resumed at the next instruction. The parameter p1 is optional; the default value is five seconds. If display list processing terminates with an error condition, resumption of processing causes an error.

RSX-11M-PLUS and Micro/RSX

Function: IO.REX

Octal value: 12400

Parameters: p1 = time-out period in seconds
p2 = report segment ID
p3 = reporting mask

MicroVMS

Function: IO$_CONTINUE

Hex value: 3F

Parameters: p1 = time out period in seconds (default = 5)
p2 = report segment ID
p3 = reporting mask

4.1.11 Define Reporting – IO.DRP and IO$_DEFREP

This function defines the reporting requirements by the report class and initializes the report segment.

One of the parameters required is a reporting mask. This is a set of bit pairs, as follows:

BIT NUMBERS	CONTENTS
15 to 10	unassigned
9 and 8	timeout/stop
7 and 6	match
5 and 4	switch
3 and 2	errors
1 and 0	warnings

The bit pair values have the following effects:

BIT PAIR VALUES	ACTION REQUIRED
0	as previously
1	to report segment
2	to mailbox
3	ignore

In the case of MicroVMS, the application program must assign a mailbox on the Assign Channel System Service if any reports are to be directed to a mailbox, that is, to be handled asynchronously. The mailbox should be enabled for Write Attention AST. This means that the processing AST routine is activated when the VSV21 driver makes an entry.

The method of defining report segments is given in Section 3.3.5.

RSX-11M-PLUS and Micro/RSX

Function: IO.DRP

Octal value: 10400

Parameters: p1 = reporting segment ID
p2 = reporting mask
p3 = AST address

MicroVMS

Function: IO$__DEFREP

Hex value: 34

Parameters: p1 = reporting mask ID
p2 = reporting mask
p3 = mailbox channel

4.1.12 Read Data – IO.RED and IO$__READDATA

RSX-11M-PLUS and Micro/RSX

Function: IO.RED

Octal value: 6400

Parameters: p1 = buffer address
p2 = buffer length
p3 = table ID

MicroVMS

Function: IO$__READDATA

Hex value: 39

Parameters: p1 = buffer address
p2 = buffer length
p3 = table ID

4.1.13 Write Data – IO.WRT and IO$__WRITEDATA

RSX-11M-PLUS and Micro/RSX

Function: IO.WRT

Octal value: 7000

Parameters: p1 = buffer address
p2 = buffer length
p3 = table ID

MicroVMS

Function: IO$__WRITEDATA

Hex value: 6400

Parameters: p1 = buffer address
p2 = buffer length
p3 = table ID

4.2 QIO STATUS REPLIES

This section describes the information returned from QIOs and the error and warning codes in the report packets.

4.2.1 QIO Replies from RSX-11M-PLUS and Micro/RSX

The contents of the I/O status block are shown in Figure 4-2.

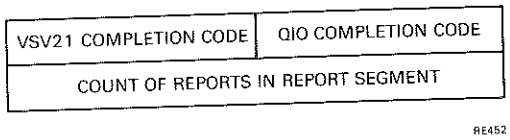


Figure 4-2 Contents of RSX I/O Status Block

The QIO reply byte contains the following octal codes and decimal equivalents:

OCTAL	DECIMAL	REPLY CODE
001	–1	IS.SUC – Success
361	–15	IE.ABO – QIO aborted
372	–6	IE.SPC – Illegal user buffer
376	–2	IE.IFC – Illegal function code
254	–84	IE.ALC – Allocation failure (define segment)
366	–8	IE.DAA – Device already attached
242	–94	IE.PNS – Partition/Region not in system

The VIVID reply byte contains the following decimal codes, where relevant:

0 – Normal completion

128 – Stop acknowledge

129 – Maximum matches reached

130 – VIVID error

The number of reports indicates the total number of reports entered to the report display segment. If no report display segment has been defined, then the number of reports that would have been written to the report display segment is given.

4.2.2 QIO Replies from MicroVMS

The contents of the I/O status block are given in Figure 4-3.

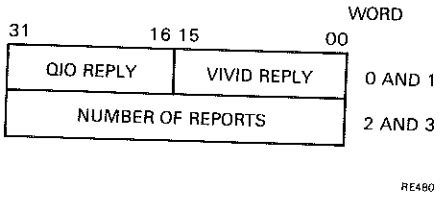


Figure 4-3 Contents of MicroVMS I/O Status Block

The QIO reply word may contain one of the following:

HEX	CODE	MEANING
0001	SS_$_NORMAL	Successful completion
030C	SS_$_BUFBYTALI	Buffer byte aligned
002C	SS_$_ABORT	_QIO aborted
018C	SS_$_LENVIO	Buffer length violation
022C	SS_$_TIMEOUT	QIO time-out
0334	SS_$_DEVREQERR	Device request error
02C4	SS_$_DEVACTIVE	Device active
032C	SS_$_DEVCMDERR	Device command error

The VIVID reply word contains one of the following decimal codes:

0 – Normal completion

128 – Stop acknowledge

129 – Maximum matches reached

130 – VIVID error

The number of reports indicates the total number of reports entered to the report segment. If no report segment has been defined, then the number of reports that would have been written to the report segment is given.

4.2.3 VIVID Error/Warning Codes

The following decimal error/warning codes are used in VIVID__WARNING and VIVID__ERROR report packets:

100 – Memory protection error

101 – Reserved instruction

102 – Invalid segment type

103 – Invalid segment ID

104 – Maximum number of segments reached

105 – Instruction sequence error

106 – Segment not defined

107 – No report segment defined

108 – Segment stack overflow

109 – Attribute stack overflow

110 – No attributes saved

111 – Parameter out of range

112 – Incorrect number of parameters

113 – No space in segment for output

114 – Total magnification exceeds 127.996

115 – START__FONT in instruction list

116 – START__PIXEL__DATA in instruction list

117 – START__KEYBOARD__DATA in instruction list

118 – START__REPORT__DATA in instruction list

119 – Total magnification less than 1/256

120 – Elliptic aspect ratio out of range

121 – FLOOD/PAINT__AREA shape is too complex

122 – Segment too small for desired use

123 – Segment is not a font

124 – Download segment ID mismatch

125 – Download block sequence error

126 – Memory allocation error

127 – No font is currently defined

128 – Specified segment is not pixel data

129 – Pixel data written has been truncated

130 – Switch interrupts not enabled

131 – Segment ID in segment is incorrect

132 – Segment length in header is incorrect.

CHAPTER 5
THE VIVID SUBROUTINE LIBRARY (VSL)

VSL is a library of functions and subroutines which can be called from a high-level language. VSL controls the display list segments, executes display list or drawing commands and handles replies from VIVID. It automatically generates VIVID instructions and parameters.

A class number of 32 is reserved for VSL segments. You should not use this class number when building other segments.

There are five groups of functions and one group of subroutines in VSL. Each function or subroutine has a six-character name of the form VVbaaa where:

- The third letter 'b' of the name identifies the VSL function or subroutine group
- The last three letters denoted by 'aaa' identify the individual function or subroutine.

The functions and subroutines are categorized as follows:

- General functions (VVXaaa), used to:
  - initialize and end display processing
  - execute VIVID instructions defined as parameters
  - set the drawing mode
- Segment Manipulation functions (VVCaaa), used to:
  - initialize segment building
  - save and restore segments to and from disk
  - load segments to the VSV21 from disk and from the host memory
  - delete segments

- Segment Execution subroutines (VVEaaa), used to:
  - start, stop and resume segment execution
- Reporting functions (VVRaaa), used to:
  - control input from pointing devices and keyboard
  - get status and reports
- Segment Building functions (VVBaaa), used to:
  - start and end segment building
  - set the drawing mode
- Instruction Generation functions

  Each instruction generation function corresponds to a VIVID instruction. The functions are categorized in the same way as the VIVID instruction set and have corresponding identifiers as follows:

  VVCaaa – Control

  VVVaaa – Transformation

  VVGaaa – Global Attributes

  VVDaaa – Drawing

  VVFaaa – Filled Figure

  VVTaaa – Text

  VVAaaa – Area Operation

  VVIaaa – Interactive

  VVQaaa – Report Handling

  where "aaa" identifies the individual function.

VSL creates a display area in the host memory with the VVXINI function (Section 5.1.1). The display area contains segments and a segment control table. VSL uses the table to control memory allocation for segments entered to the display area.

The display area is a region in RSX-11M-PLUS and Micro/RSX systems. Access to the region is by means of a window, which you must define when you are building the task. The window must be mapped into the program space at 28K. The window characteristics are as follows:

Window name: VV21RG

Window size: 4K words

The display area is a 2 Mbyte global section in MicroVMS. Access to the section is by array element. Each element is a two-word integer in the array.

The call formats given are for FORTRAN. All parameters are integers unless otherwise stated. All the VSL functions return a status value istat. Reply values are additive; if two different errors are detected, the reply is the sum of the error codes.

Use the /NOOP qualifier when compiling programs which contain calls to VSL.

5.1 GENERAL FUNCTIONS

5.1.1 Initialize Display Processing – VVXINI

Sets up the display area and initializes VIVID processing.

Call format: istat = VVXINI ([dlen [,maxno]])

Parameters: dlen = size of VSL display area in bytes
0 : default of 16K

maxno = maximum number of segments
0 : default of 640

Reply value: 0 = already initialized
1 = completed successfully
4 = could not create area
8 = could not create window
(Micro/RSX and RSX-11M-PLUS only)

5.1.2 End Display Processing – VVXEND

Releases the VSV21 processor and frees the VSV21 buffers. To restart processing, you must call the VVXINI and VVXATT functions.

Call format: istat = VVXEND

Parameters: None

Reply value: 0 = completed successfully
1 = unsuccessful

5.1.3 Assign VSV21 Device – VVXASS

Assigns and attaches a VSV21 device to VSL processing.

Call format: istat = VVXASS (dev, lun [, rslen [, clarr]])

Parameters: dev = VSV21 device name
lun = device logical unit number
rslen = length of report segment. If rslen = 0, a default report buffer size of 1 Kbyte is assumed.
clarr = array containing host classes accessible to unit

Reply value: 0 = not initialized
1 = completed succesfully
3 = invalid segment class
16 = device already assigned
32 = could not assign device
64 = could not attach device
256 = more than 512 segments sent to logical unit
512 = VSL region full
1024 = report segment setup failed
2048 = invalid segment class

Notes: You must attach the VSV21 device before using it. You can attach a maximum of four devices under RSX-11M-PLUS and Micro/RSX and a maximum of eight under MicroVMS. A unique report segment ID for each device is generated in display segments of class 32.

The clarr parameter is an array of host classes, null terminated, that are explicitly available to the unit. If it is omitted, all segments previously set up on the host are available to the unit.

The numbers of devices which may be attached are as follows:

RSX-11M-PLUS : 4 devices

MicroVMS : 8 devices

5.1.4 Release VSV21 Device – VVXREL

Releases the VSV21 device from VSL processing.

Call format: istat = VVXREL (lun)

Parameters: lun = device logical unit number

Reply value: 0 = not initialized
1 = completed successfully
4 = device not assigned

Notes: To detach all the devices when display processing is complete, use VVXEND (Section 5.1.2).

5.1.5 Get VIVID version number – VVXVER

Gets the VIVID version number.

Call format: istat = VVXVER (len, vnarr)

Parameters: len = length of array in bytes (minimum 6)
vnarr = character array for version number

Reply value: 0 = array length too short (<6)
1 = completed successfully

5.2 SEGMENT MANIPULATION FUNCTIONS

These functions provide movement of segments between disk, host and VSV21, and create segments for input from VSV21 devices.

5.2.1 Save Segments on Disk – VVMSAV

Writes up to eight specified segments on host memory to a disk file.

Call format: istat = VVMSAV (filn, idarr, nseg)

Parameters: filn = name of disk file
idarr = array containing segment IDs
nseg = number of segments to be written

Reply value: 0 = not initialized
1 = completed successfully
4 = segment not found

5.2.2 Restore Segments from Disk – VVMGET

Reads segments to the host memory from a specified disk file. The segments must already have been saved using VVMSAV or be one of the following types:

- Instruction
- Font
- Pixel data

Call format: istat = VVMGET (lun, fun, filn)

Parameters: lun = VSV21 logical unit number
−1 : all units for which segment class is valid
fun = FORTRAN unit number for file
filn = name of disk file

Reply value: 0 = not initialized
1 = completed successfully
4 = segment exists on host
16 = device not assigned
32 = define segment failed
256 = over 512 segments for lun
512 = display area full
1024 = too many host segments
2048 = invalid segment class

Notes: If the segment is already in the host memory, no display segments are created. If any segments have been created already, the success bit will also be set.

The segment IDs assigned are those appearing in the segments in the file.

5.2.3 Copy Segment – VVMCPY

Copies a segment to the VSL display area. The segments must have already been saved using VVMSAV or be of one of the following types:

- Instruction
- Font
- Pixel data

Call format: istat = VVMCPY (lun, arr)

Parameters: lun = logical unit number
–1 : all units for which segment class is valid
arr = array containing segment

Reply value: 1 = completed successfully
0 = not initialized
4 = segment exists on host
16 = device not assigned
32 = segment download failed
512 = display area full
1024 = too many host segments
2048 = invalid segment class

Notes: If a segment already exists, no segment is created or updated.

The segment ID and length are derived from the array.

5.2.4 Load Segments from File – VVMDLD

This function reads segments to the VSV21 memory from a specified disk file. If the segment is already in VSV21 memory, no segment is read. The segments must have been already saved using VVMSAV or be of one of the following types:

- Instruction
- Font
- Pixel data

Call format: istat = VVMDLD (lun, fun, filn)

Parameters: filn = name of disk file
fun = FORTRAN unit number
lun = device logical unit number
−1 : all units for which segment class is valid

Reply value: 0 = not initialized
1 = completed successfully
4 = segment exists on host
16 = device not assigned
32 = segment download failed
512 = display area full
1024 = too many host segments
2048 = invalid segment class

Notes: If a segment already exists on the VSV21, it is replaced.

If one or more segments have been successfully downloaded when the error occurs, the success bit is also set in the reply.

The assigned segment IDs are those appearing in the segments in the file.

The transfer uses a 512-byte work buffer in the display area.

5.2.5 Load Segment from Host – VVMMLD

Load to the VSV21 a segment which has been generated by the application program between VVBBGN and VVBEND or read from file using VVMGET.

The segment is downloaded to the indicated logical unit number.

If the segment has been downloaded to all units to which it is defined, it is deleted from the display area.

Call format: istat = VVMMLD (lun, segid)

Parameters: lun = logical unit number
−1 : all units for which segment class is valid

segid = segment ID

Reply value: 1 = completed successfully
0 = not initialized
4 = segment not found
16 = device not assigned
32 = download segment failed
2048 = invalid segment class.

Notes: The segment ID format may be found in Section 3.3.1.

5.2.6 Delete Segment – VVMDEL

Deletes a segment on the host or VSV21 memory. The segment may have been generated by any means.

If the segment is in the display area and has been deleted from all units to which it is defined, it is deleted in the display area segment control and the space becomes free.

Call format: istat = VVMDEL (lun, segid)

Parameters: lun = logical unit number
–1 : all values for which segment class is valid

segid = segment ID

Reply value: 1 = completed successfully
4 = segment not found
16 = device not assigned

Notes: The segment ID format is described in Section 3.3.1.

5.2.7 Create Segment – VVMCRS

Creates a segment in the VSL display area for keyboard or pixel data map data input from the VSV21, or for subsequent use for VIVID instruction generation.

The segment is defined to the indicated logical unit number.

Call format: istat = VVMCRS (lun, segid, len)

Parameters: lun = VSV21 logical unit number
–1 : all units
segid = segment ID
len = length of segment in bytes

Reply value:
0 = not initialized
1 = completed successfully
4 = segment exists already
16 = device not assigned
32 = define segment failed
256 = over 512 segments for lun
512 = display area full
1024 = too many host segments
2048 = invalid segment class

Notes: The segment ID format may be found in Section 3.1.

This facility is also a useful performance tool. The display execution subroutines only initiate output of a segment. For volatile top-level segments, a double buffering system may be implemented.

5.3 SEGMENT EXECUTION SUBROUTINES

These subroutines initiate VSV21 output operations. The operation initiated is completed only when a further Segment Execution call is made or a Reporting Function is accessed for the same logical unit.

If status and reports are required, no segment Execution call should intervene before Reporting calls have been completed.

While the output operation is in progress, the segment is locked and the segment cannot be written to or deleted.

5.3.1 Execute Segment – VVEEXE

Initiates output of the specified segment.

Call format: CALL VVEEXE (lun, segid [,tout [,rsegid]])

Parameters:
lun = logical unit number
segid = segment ID
tout = time out in seconds
rsegid = reporting segment ID

Notes: The segment ID format may be found in Section 3.1.

If the time-out value is zero, or omitted, a default value of 10 seconds is used. If pointing device activity and/or keyboard activity is to occur during execution of the segment, a considerably longer time-out value is required.

If time-out occurs, there will be a VIVID_INTERRUPT packet on the report segment for the logical unit.

The reporting segment is optional if a reporting segment size was given to VVXASS. Otherwise any reports will be lost if no reporting segment is identified.

If the segment is currently being built, VVBEND is actioned automatically first.

5.3.2 Resume Segment Execution – VVERES

Resumes execution of the last segment executed or resumed for the indicated logical unit.

Call format: CALL VVERES (lun [,tout [,rsegid]])

Parameters: lun = logical unit number
tout = time out in seconds
0 : default to 5
resegid = reporting segment ID

Notes: If the time-out value is zero, or omitted, a default of 5 seconds is used. If pointing device activity and/or keyboard activity is to occur during execution of the segment, a considerably longer time out value is required.

If time out occurs, there will be a VIVID_INTERRUPT packet on the report segment for the logical unit.

The reporting segment is optional if a reporting segment size was given to VVXASS. Otherwise any reports will be lost if no reporting segment is identified.

5.3.3 Stop Segment Execution – VVESTP

Stops execution of the last segment executed or resumed for the indicated logical unit.

Call format: CALL VVESTP (lun)

Parameters: lun = logical unit number

Notes: There will be a VIVID_INTERRUPT packet on the report segment for the logical unit.

5.4 REPORTING FUNCTIONS

5.4.1 Get Status – VVRSTA

Provides the status of the preceeding display output for the unit.

Call format: istat = VVRSTA (lun, qiost, nrep)

Parameters: lun = logical unit number
qiost = QIO status reply (output)
nrep = total report count (output)

Reply value: 0 = not initialized
1 = completed successfully
16 = device not assigned

Notes: The QIO status replies are given in Section 4.1.

The report count is the total number of reports issued for a previous call invoking display list processing. See VVEEXE (Section 5.3.1) and VVERES (Section 5.3.2) for further details.

The report formats may be found in Chapter 14.

When status is requested, any initiated output to the logical unit is completed before return from the function.

5.4.2 Get Report – VVRREP

Obtains a report from the report segment for the indicated logical unit number. The report may be of any type or of a specified type.

Call format: istat = VVRREP (lun, rtype, arr, alen [,rsegid])

Parameters: lun = logical unit number
rtype = report type required
–1 : any type
arr = array for report (output)
alen = array length
rsegid = report segment ID

Reply value: 0 = not initialized
1 = completed successfully
4 = segment not found
8 = not report segment
16 = device not assigned
32 = report segment overflow
129 = area too short
–32768 = all reports read

Notes: See Chapter 14 for details of report requests.

If the area is not long enough, transfer of the report data continues until the area is full.

Reports may be requested before the QIO is completed. The buffer is polled, so QIO completion is not forced.

5.4.3 Get Keyboard Input – VVRKBD

Obtains the contents of the keyboard input segment in the specified string.

Call format: istat = VVRKBD (segid, charr,alen [[,dlen], segst])

Parameters: segid = keyboard input segment ID
charr = char array to contain input (output)
alen = size of array in bytes
dlen = length of data in bytes (output)
segst = segment status (output)

Reply value: 1 = completed successfully
0 = not initialized
4 = segment not found
8 = not keyboard input segment
16 = device not assigned
32 = report segment exception
129 = area too short

Notes: The string entered to the array by the function is a standard ASCII string, null terminated (unless overflow occurs).

5.4.4 Get Segment Block – VVRSEG

Get a block of data from a segment in the VSL display area. This is intended specifically for access to pixel data maps, but may be used to access any segment.

Call format: istat = VVRSEG (segid, start, ilen, barr, olen)

Parameters:
- segid = segment ID
- start = segment start byte offset
- ilen = block length in bytes
- barr = array to receive block (output)
- olen = length in bytes transferred (output)

Reply Value:
- 1 = completed successfully
- 0 = not initialized
- 4 = segment not found
- 32 = start byte offset out of range
- 129 = area too short

Notes: The blocked transfer allows processing of large pixel data maps for screen printing.

5.5 SEGMENT BUILDING FUNCTIONS

5.5.1 Start Segment – VVBBGN

Starts a new segment. If the segment currently exists in the VSL display area, initialize it for entry of a new set of VIVID instructions.

The segment header START_INSTRUCTION_LIST (Section 3.3.1) is set up. Subsequent calls to VIVID Instruction Generation Functions cause VIVID instructions to be put in this segment.

Call format: istat = VVBBGN (segid)

Parameters: segid = segment ID

Reply value:
- 1 = success
- 0 = not initialized
- 4 = segment not found

Notes: The segment ID format may be found in Section 3.1.

If a segment is currently being built, a call to VVBEND (Section 5.5.2) is implied for that segment.

5.5.2 End Segment – VVBEND

Ends the segment currently being built. Subsequent calls to VIVID Instruction Generation Functions are ignored until a call to VVBBGN is encountered.

Call format: istat = VVBEND

Parameters: None

Reply Value: 1 = completed successfully
0 = not initialized
8 = no segment build in progress
16 = segment overflow has occurred

Notes: If no display segment is currently being built, no action is performed.

VIVID instructions which overflow an existing display area are lost.

5.5.3 Set Drawing Mode – VVBMOD

Sets the mode required for subsequent VIVID Instruction Generation calls to absolute or relative.

Call format: istat = VVBMOD (dmode)

Parameters: dmode = drawing mode
0 : absolute (this is the default)
1 : relative

Reply Value: 1 = completed successfully
0 = invalid parameter

Notes: If the mode parameter is not zero, relative drawing is assumed.

5.5.4 Set Instruction Parameter Mode – VVBPMD

Sets the parameter mode for subsequent VIVID Instruction Generation functions to "parameter list" or "array list". This only affects certain functions. For further details of the action performed, see Section 5.6.

Call format: istat = VVBPMD (pmode)

Parameters: pmode = parameter mode:
0 : parameter list (this is the default)
1 : array list

Reply Value: 1 = completed successfully
0 = invalid parameter

Notes: If the mode parameter is not 1, parameter list is assumed.

5.6 INSTRUCTION GENERATION FUNCTIONS

Each VSL instruction generation function generates an instruction from the VIVID instruction set. An instruction generation function has the same parameters as its corresponding VIVID instruction.

VSL operates in either of two modes, depending on the most recent call to VVBPMD. The modes are as follows:

- Parameter list mode
- Array list mode

In parameter list mode (Section 5.5.4) the number of parameters declared on the function call is variable. All the parameters and the parameter count are passed in the opcode word to the VIVID instruction. Thus, the list delimiter END_PARAMETERS is not required and must not be used. VSL checks that the number of parameters is within the permitted range for each instruction.

In array list mode (Section 5.5.4) an alternative call format is used for some functions. These functions are marked "*" in Sections 5.6.1 to 5.6.8. In this case, the first parameter is an array containing the actual parameter list. The parameter list may be terminated by END_PARAMETERS, or a second parameter indicating the number of parameters in the list may be provided on the function call. The functions to which this facility applies correspond to the VIVID instructions for which END_PARAMETERS may be used.

Where the VIVID instruction has the forms ABS and REL, the mode used is dependent on the last call to VVBMOD (Section 5.5.3) encountered.

The function call should occur between VVBBGN and VVBEND calls (Section 1.6). The call then causes the appropriate VIVID instruction to be added to the segment being built. Reply values from the function are:

0 = no segment active

1 = completed successfully

2 = segment overflow

If there is an error reply, no VIVID instruction is generated. However the reply information is available when the VVBEND instruction is executed so when the segment is completed, these functions may be accessed as subroutines with general status information.

5.6.1 Control Functions

VVCINI – INITIALIZE

VVCCAL – CALL_SEGMENT

VVCSAV – SAVE_ATTRIBUTES

VVCRES – RESTORE_ATTRIBUTES

VVCDMP – DUMP_ATTRIBUTES

VVCRCV – RECOVER_ATTRIBUTES

VVCDWT – DISPLAY_WAIT

VVCSTP – STOP_DISPLAY

VVCCRS – CREATE_SEGMENT

VVCJMP – JUMP_RELATIVE

VVCREP – DISPLAY_REPEAT

VVCERP – DISPLAY_END_REPEAT

5.6.2 Transformation Functions

VVVDIM – SCREEN_DIMENSIONS

VVVWOR – WINDOW_ORIGIN

VVVZMF – ZOOM_FACTOR

VVVDRM – DRAWING_MAGNIFICATION

VVVDRT – DRAWING_TRANSLATION

VVVTRN – DRAWING_TRANSFORM

VVVVAS – DRAWING_VAS

VVVSVU – SET_VIEWPORT

VVVWND – SET_WINDOW

5.6.3 Global Attribute Functions

VVGBLK – SCREEN_BLINK

VVGBLT – BLINK_TIMING

VVGSCB – SCREEN_BLANK

VVGFCL – FOREGROUND_COLOR

VVGBCL – BACKGROUND_COLOR

VVGNLC – NORMAL_COLORS *

VVGBLC – BLINK_COLORS *

VVGBCT – BLINK_COUNT

VVGMOD – DRAWING_MODE

VVGLTX – LINE_TEXTURE

VVGATX – AREA_TEXTURE

5.6.4 Drawing Functions

VVDMOV – MOVE_ABS, MOVE_REL

VVDMTC – MOVE_TO_CURSOR

VVDLIN – LINES_ABS, LINES_REL *

VVDPMK – POLYMARKS_ABS, POLYMARKS_REL *

VVDARC – ARCS_ABS, ARCS_REL *

VVDEAR – ELLIPSE_ARCS_ABS, ELLIPSE_ARCS_REL *

VVDREC – RECTANGLE_ABS, RECTANGLE_REL

VVDELL – ELLIPSE

VVDCRC – CIRCLE

VVDDOT – DOT

5.6.5 Filled Figure Functions

VVFRCT – FILLED_RECT_ABS, FILLED_RECT_REL

VVFFLD – FLOOD_AREA

VVFPNT – PAINT_AREA

5.6.6 Text Functions

VVTIFT – INITIALIZE_FONT

VVTSFT – SET_FONT

VVTLDC – LOAD_CHAR_CELL *

VVTOBL – CELL_OBLIQUE

VVTROT – CELL_ROTATION

VVTSIZ – CELL_SIZE

VVTMAG – CELL_MAGNIFICATION

VVTMOV – CELL_MOVEMENT

VVTDRC – DRAW_CHARS *

VVTDRP – DRAW_PACKED_CHARS *

5.6.7 Area Operation Functions

VVACLS – CLEAR_SCREEN

VVACLV – CLEAR_VIEWPORT

VVASCV – SCROLL_VIEWPORT

VVAPXR – PIXEL_READBACK

VVAPXW – PIXEL_WRITE

VVAFPR – FAST_PIXEL_WRITE

VVAFPM – FAST_PIXEL_MODIFY

VVASCL – SELECTIVE_CLEAR

VVACPY – COPY_ABS, COPY_REL

5.6.8 Interactive Functions

VVICUS – CURSOR_STYLE *

VVIPCU – POSITION_CURSOR

VVICUV – CURSOR_VISIBILITY

VVIRUB – RUBBER_BAND

VVISWE – SWITCH_REPORT_ENABLE

VVISWD – SWITCH_REPORT_DISABLE

VVIASW – AUTOSWITCH

VVIWSW – WAIT_SWITCH

VVIMTE – MATCH_ENABLE

VVIMTD – MATCH_DISABLE

VVIAKI – ACCEPT_KEYBOARD_INPUT

5.6.9 Report Handling Functions

VVQREP – REQUEST_REPORT

5.7 CALLING VSL

The call formats given in Sections 5.1 through 5.6 are for FORTRAN. The following constraints must be observed when programming in high-level languages other than FORTRAN:

5.7.1 Passing Parameters to VSL

A program call to VSL must pass the parameters to VSL by reference. This means that the program passes the parameter by giving its address, not its value. FORTRAN automatically passes parameters by reference. Calls in other high-level languages must specify the reference method.

EXAMPLE

The FORTRAN call to the Assign function (Section 5.1.3) might be as follows:

```
istat   = VVXASS ( 0, 1, 1024 )
```

Since the C language passes parameters by value, the equivalent call in C is as follows:

```
extern short vvxass ( );

.
.
.

device = 0;
lun    = 1;
size   = 1024;
reply  = vvxass ( &device, &lun, &size );
```

The equivalent PASCAL call is as follows:

```
FUNCTION vvxass ( device, lun, size : INTEGER ) : INTEGER; EXTERN;


BEGIN
    device := 0;
    lun    := 1;
    size   := 1024;
    reply  := vvxass ( %ref ( device ), %ref ( lun ), %ref ( size ))
END;
```

Note the use of the "%ref" to force the passing of an address.

If VAR parameters are declared in the PASCAL function, "%ref" can be omitted:

```
FUNCTION vvxass ( VAR device, lun, size : INTEGER ) : INTEGER;
EXTERN;


BEGIN
      device := 0;
      lun    := 1;
      size   := 1024;
      reply  := vvxass ( device, lun, size)
END;
```

5.7.2 VSL Word Length on MicroVMS Systems

VSL works mainly in units of 16-bit words. As MicroVMS uses 32-bit words, arrays must define word length as 16 bits. Other parameters passed to VSL may be 16-bit or 32-bit. However, 32-bit variables must be within the 16-bit integer range (−32768 to +32767).

A VSL function returns a 32-bit integer to the parameter istat. Attempts to return variables of other types have unpredictable results.

EXAMPLE

The VVICUS function (Section 5.6.8) is used to describe a cursor style. The number of parameters is variable.

```
PROCEDURE cursor;

TYPE
      word   = −32768..32767;
VAR
      arr    : PACKED ARRAY [ x..y ] OF word;
BEGIN
      arr [ 1 ] := a;
      arr [ 2 ] := b;
      .
      .
      arr [ n ] := n;
      reply  := vvicus ( arr )
END;
```

Note the use of PACKED in the definition of this PASCAL example array. If PACKED is not included, the cursor type is stored in a 32-bit word and the VSL routine fails.

CHAPTER 6
CONTROL INSTRUCTIONS

This chapter contains a description of each VIVID control instruction. Opcodes are given in decimal. A MACRO-32 example of each instruction is provided.

6.1 START_INSTRUCTION_LIST

Identifies the segment contents as display instructions.

Opcode: 01 Length: 2

Format: START_INSTRUCTION_LIST segid, slen

Parameters: segid = segment ID
slen = total segment size in bytes

End Position: The current drawing position is not changed.

Errors: Warning if this instruction is encountered after the start of the instruction segment.

Notes: This must be the first instruction in a VIVID instruction segment.

Example:

```
.BYTE 2.,1.             ;length and opcode
.WORD ^X010A            ;segment ID 1, class 10
.WORD 2048.             ;2K bytes of segment area
```

6.2 START_FONT

Identifies the segment contents as a font.

Opcode: 02 Length: 6

Format: START_FONT segid, slen, ncc, xdim, ydim, dflt

Parameters:
- segid = segment ID
- slen = total segment size in bytes
- ncc = total number of character cells in the font
- xdim = cell width in bits (range 1-16)
- ydim = cell height in bits (range 1-16)
- dflt = default row value. Defines whether unspecified rows are drawn in foreground or background color.
  - 0 : background
  - −1 : foreground

End Position: The current position is not changed.

Errors: Warning if this instruction is encountered during an instruction segment.

Notes: This instruction is set up automatically by the INITIALIZE__FONT instruction. When a font is accessed, this must be the first instruction in the font segment.

Example:

```
.BYTE 6.,2.          ;length and opcode
.WORD ^X010A.        ;segment ID 1, class 10
.WORD 66.            ;segment length in bytes
.WORD 26.            ;26-cell font
.WORD 12.            ;12-bit width
.WORD 10.            ;10-bit height
.WORD -1.            ;foreground color
```

6.3 START__PIXEL__DATA

Identifies the segment contents as pixel data.

Opcode: 06 Length: 4

Format: START__PIXEL__DATA segid, slen, xdis, ydis, xrat, yrat

Parameters:
- segid = segment ID
- slen = total segment size in bytes
- xdis = X distance to opposite vertex in words. May be negative.
- ydis = Y distance to opposite vertex in pixels. May be negative.
- xrat = X pixel screen to monitor ratio
- yrat = Y pixel screen to monitor ratio

End Position: The current drawing position is not changed.

Errors: Warning if this instruction is encountered during an instruction segment.

Notes: This instruction is set up automatically by the PIXEL_READBACK instruction. When a pixel data map display segment is accessed, this must be the first instruction in the display segment.

A pixel data word contains four pixels.

The screen to monitor ratio is the ratio of the logical screen dimensions to the physical monitor dimensions. The most significant byte (MSB) holds the integer part and the least significant byte (LSB) the fractional part of a fixed-point number.

Example:

```
.BYTE 4.,3.          ;length and opcode
.WORD ^X080C         ;segment class 8, number 12
.WORD 66.            ;segment length in bytes
.WORD 20.            ;X distance 20 words to right
.WORD 32.            ;Y distance 32 pixels
                     ;(eight words) downwards
;
;Typical ratio for a low
;resolution monitor defined
;as a high-resolution logical
;screen
;
.BYTE 1.,0.          ;logical X dim = monitor X dim
.BYTE 2.,0.          ;logical X dim = 2.0 times
                     ;monitor X dim
```

6.4 START_KEYBOARD_DATA

Identifies the segment contents as keyboard input.

Opcode: 04 Length: 4

Format: START_KEYBOARD_DATA segid, slen, istat, icnt

Parameters:
- segid = segment ID
- slen = total segment size in bytes
- istat = current buffer status
  - 0 : transfer in progress
  - 1 : transfer ended at termination character
  - 2 : transfer completed on maximum length
  - 3 : transfer completed on time out or stop
  - 4 : transfer completed on buffer full
- icnt = count of bytes entered to segment

End Position: The current drawing position is not changed.

Errors: Warning if this command is encountered during an instruction segment.

Notes: This is set up automatically by the ACCEPT_KEYBOARD_INPUT instruction (Section 13.12).

The termination character is not entered to the segment.

Example:

```
.BYTE 4.,4.          ;length and opcode
.WORD ^X0C0E.        ;segment class 12, number 14
.WORD 66.            ;segment length in bytes
.WORD 2.             ;transfer completed on max.
                     ;length
.WORD 45.            ;45 bytes entered to segment
```

6.5 START_REPORT_DATA

Identifies the segment contents as report data.

Opcode: 05 Length: 4

Format: START_REPORT_DATA segid, slen, istat, nextb

Parameters:
- segid = segment ID
- slen = total segment size in bytes
- istat = current buffer status:
  - 0 : active
  - 1 : initialized/complete
  - 2 : segment overflow
- nextb = byte offset of next entry processed. This is a pointer to the next free byte in the segment, counting from the start of the segment. It is always word-aligned.

End Position: The drawing current position is not changed.

Errors: Error if other display segment type expected. Warning if encountered during instruction display segment.

Notes: This is set up automatically by QIOs which execute a segment or resume segment execution.

The parameter istat gives the status of the report segment activity. The application program may poll istat to check if the the segment is being written to by VIVID (active status) or is full (segment overflow).

Example:

```
.BYTE 4.,5.          ;length and opcode
.WORD ^X030A         ;segment class 3, number 10
.WORD 66.            ;segment length in bytes
.WORD 1.             ;buffer initialized
.WORD 10.            ;start of free space
```

6.6 INITIALIZE

Restores VIVID download status to one or more graphics-control facets (addressing, global attributes, text or all).

Opcode: 06 Length: 1

Format: INITIALIZE mask

Parameters: mask = sum of values indicating requirements
−1: all values
See Appendix D for values.

End Position: If transformations are initialized, the current position is set to the origin. Otherwise the current position is not changed.

Errors: None

Notes: Initialization values may be found in Appendix D. The values required should be summed to determine the value of the mask parameter.

Example:

```
.BYTE 1.,6.          ;length and opcode
.WORD 4.             ;initialize drawing colors
```

6.7 CALL_SEGMENT

Executes the identified segment from the host or VSV21 memory.

Opcode: 07 Length: 1

Format: CALL_SEGMENT segid

Parameters: segid = segment ID

End Position: The current position is not changed.

Errors: Error if the segment is not found.

Notes: The segment must already be defined as a host segment or be downloaded.

Example:
```
.BYTE 1.,7.          ;length and opcode
.WORD ^X010A.        ;segment class 1, number 10
```

6.8 SAVE_ATTRIBUTES

The current attributes are added to an attribute stack. This allows you to change attributes in a nested segment and to recover attributes before returning to the calling segment (see also RESTORE_ATTRIBUTES, Section 6.9).

Opcode: 08 Length: 0

Format: SAVE_ATTRIBUTES

Parameters: None

End Position: The current position is not changed.

Errors: Terminal error if stack would overflow.

Notes: The space available for the stack depends on the number of downloaded segments and on the number and order of deletions. No implicit compress is performed.

The attributes stacked are identified in Appendix D. This includes drawing and cursor positions.

Example:
```
.BYTE 0.,8.          ;length and opcode
```

6.9 RESTORE_ATTRIBUTES

The last attributes saved by SAVE_ATTRIBUTES are removed from the stack and set up as the current attributes. The previous attributes are lost.

Opcode: 09 Length: 1

Format: RESTORE_ATTRIBUTES mask

Parameters: mask = mask value indicating requirements: see Appendix D for values.
−1 : all values are restored

End Position: As given by the stacked parameters. If the drawing position is not restored, it is not changed. The cursor position is handled similarly.

Errors: Fatal error if no parameter value is supplied.

Warning if there are no stacked attributes.

Notes: Mask values entered indicate these attributes are to be restored.

The attributes that will be restored from the stack for each mask value are identified in Appendix D.

Example:
```
.BYTE 1.,9.          ;length and opcode
.WORD 100            ;octal mask value to
                     ;restore text attributes
```

6.10 START_ATTRIBUTES_DATA

Identifies the segment contents as attributes data.

Opcode: 127 Length: 2

Format: START_ATTRIBUTES_DATA segid, slen

Parameters: segid = segment ID
slen = total length of segment in bytes

End Position: The current position is not changed.

Errors: A warning is issued and the instruction is ignored if it is encountered in an instruction segment.

Notes: START_ATTRIBUTES_DATA is used only as the first instruction in a segment containing attributes data.

This instruction is generated automatically by the DUMP_ATTRIBUTES instruction.

Example:
```
.BYTE 2.,127.        ;length and opcode
.WORD ^X040A         ;segment class 4, number 10
.WORD 120.           ;segment length
```

6.11 DUMP_ATTRIBUTES

Saves the current set of attributes in a specified segment.

Opcode: 121 Length: 1

Format: DUMP_ATTRIBUTES segid

Parameters: segid = segment ID

End Position: The current position is not changed.

Errors: A warning is issued and the instruction is ignored in the following situations:

- Segment ID outside the valid range
- Segment too small to contain the attributes
- Insufficient on-board space to create the attribute dump segment

Notes: A minimum size of 256 bytes is recommended for the attributes dump segment.

The attributes dump segment must start with a START_ATTRIBUTES_DATA instruction.

If the specified segment does not exist, it is created on the VSV21 module.

The user has no read access to on-board segments.

If the contents of the attributes dump segment are to be saved, the segment must be defined already on the host.

Example:
```
.BYTE 1.,121.          ;length and opcode
.WORD ^X030A           ;segment class 3, number 10
```

6.12 RECOVER_ATTRIBUTES

Reads the specified attributes from the specified segment.

Opcode: 122 Length: 2

Format: RECOVER_ATTRIBUTES segid, mask

Parameters: segid = segment ID
mask = bit-mask value defining attributes to be recovered

End Position: As given by the recovered drawing position if it is specified in the mask. Otherwise unchanged.

Errors: A warning is issued and the instruction is ignored in the following cases:

- Specified segment does not exist
- Specified segment does not start with a START_ATTRIBUTES_DATA instruction

Notes: Details of the mask are given in Appendix D.

Example:
```
.BYTE  2.,121.          ;length and opcode
.WORD  ^X030A           ;segment class 3, number 10
.WORD  10               ;octal mask for drawing
```

6.13 DISPLAY_WAIT

Waits for a specified time before executing the next display instruction.

Opcode: 10 Length: 1

Format: DISPLAY_WAIT nfram

Parameters: nfram = number of video frames delay required.
There are sixty frames per second.

End Position: The drawing position is not changed.

Errors: None

Notes: Execution is interrupted by a Stop Execution QIO, or a QIO time-out. Processing resumes at the next instruction.

Example:
```
.BYTE 1.,10.            ;length and opcode
.WORD 300.              ;=5 seconds delay
```

6.14 NO_OPERATION

No operation is performed and nothing is changed.

Opcode: 11 Length: 0

Format: NO_OPERATION

Parameters: None

End Position: The drawing position is not changed.

Errors: Error if length is not equal to zero.

Example:
```
.BYTE 0.,11.            ;length and opcode
```

6.15 STOP_DISPLAY

Stops display list processing. Control is returned to the application program, with a status value.

Opcode: 12 Length: 0

Format: STOP_DISPLAY

Parameters: None

End Position: The drawing position is not changed.

Errors: None

Notes: Segment processing stops and the invoking QIO is completed.

Example:
```
.BYTE 0.,12.          ;length and opcode
```

6.16 CREATE_SEGMENT

Creates a segment in the VSV21 memory.

Opcode: 13 Length: 2

Format: CREATE_SEGMENT segid, slen

Parameters: segid = segment ID
slen = total segment size in bytes

End Position: The current drawing position is not changed.

Errors: Terminal error if space is insufficient. Warning if segment already exists.

Notes: The segment ID format may be found in Chapter 3.

Example:
```
.BYTE 2.,13.          ;length and opcode
.WORD ^X010A          ;segment class 1, number 10
.WORD 54.             ;54 bytes in segment
```

6.17 SEGMENT_RETURN

Marks the end of an instruction segment.

Opcode: 14 Length: 0

Format: SEGMENT_RETURN

Parameters: None

End Position: The drawing position is not changed.

Errors: None

Notes: Control is returned as follows:

- For a nested display segment, control returns to the instruction following the invoking CALL_SEGMENT instruction.
- For a top-level segment, display list processing stops and the invoking QIO is completed. Control returns to the application program, with a status value.

SEGMENT_RETURN or STOP_DISPLAY must appear as the last instruction in the segment. Otherwise a memory protection violation or other error will occur.

Example:
```
.BYTE 0.,14.          ;length and opcode
```

6.18 JUMP_RELATIVE

Adds the specified number of words to the display list pointer.

Opcode: 120 Length: 1

Format: JUMP_RELATIVE nwords

Parameters: nwords = number of words

End Position: The current drawing position is not changed.

Errors: None

Notes: This instruction is useful for patching display lists. It operates only within the current segment. A parameter value of zero causes a jump to the JUMP_RELATIVE instruction.

Example:
```
.BYTE 1.,120.        ;length and opcode
.WORD 17.            ;jump 17 words
```

6.19 DISPLAY__REPEAT

Marks the start of a loop.

Opcode: 15 Length: 1

Format: DISPLAY__REPEAT nloop

Parameters: nloop = number of times the loop is to be repeated.
0 : loop is repeated infinitely

End Position: The current drawing position is not changed.

Errors: If loops are nested more than 32 levels deep or the parameter nloop is negative, a warning is issued and the instruction is ignored.

Notes: This instruction must have a corresponding DISPLAY__END__ REPEAT instruction in the same segment to terminate the loop.

Example:
```
.BYTE 1.,15.         ;length and opcode
.WORD 3.             ;repeat loop 3 times
```

6.20 DISPLAY__END__REPEAT

Marks the end of a repeatable loop.

Opcode: 16 Length: 0

Format: DISPLAY__END__REPEAT

Parameters: None

End Position: The current drawing position is not changed.

Errors: If no corresponding DISPLAY__REPEAT instruction in the same segment has been executed, a warning is issued and the instruction is ignored.

Example:
```
.BYTE 1.,16.         ;length and opcode
```

CHAPTER 7
TRANSFORMATION INSTRUCTIONS

This chapter describes the instructions used in drawing and viewing transformations of VSV21 data. The transformation process is described in Section 2.5.

Opcodes are given in decimal. A MACRO-32 example of each instruction is provided.

7.1 DRAWING_MAGNIFICATION

This instruction defines the magnification of the drawing elements being entered to VAS. This applies to both absolute and relative drawing operations.

Opcode: 21 Length: 0 or 2

Format: DRAWING_MAGNIFICATION [xmag, ymag]

Parameters: xmag = magnification along X axis
ymag = magnification along Y axis

End Position: The current position in VAS is not changed.

Errors: Fatal error if total magnification is outside the valid range (see ZOOM_FACTOR, Section 7.7). Display processing stops. The start and resume QIOs do not reset the transformation. Drawing can be continued. The magnification is truncated to the nearest valid value.

Notes: The parameters are input as follows:

- Enter zero values or omit the parameters for no magnification.
- Enter positive parameters to multiply the existing magnification by the absolute value of the parameters.
- Enter negative parameters to divide the existing magnification by the absolute value of the parameters.

The parameters are specified in fixed binary point format. The LSB represents the fractional part of the number and the MSB represents the integer part.

Example:
```
.BYTE 2.,21.        ;length and opcode
.WORD ^X800         ;magnify times 8 in horizontal
.WORD ^X400         ;magnify times 4 in vertical
```

7.2 DRAWING_TRANSLATION

This defines the coordinates by which the transformation origin is shifted relative to the previous transformation origin.

Opcode: 22 Length: 2

Format: DRAWING_TRANSLATION x,y

Parameters: x = X coordinate of translation
y = Y coordinate of translation

End Position: The current drawing position is not changed.

Errors: None

Notes: No drawing occurs.

The viewport is not changed.

Example:
```
.BYTE 2.,17.        ;length and opcode
.WORD 50.           ;relative X coordinate of origin
.WORD 200.          ;relative Y coordinate of origin
```

7.3 DRAWING_TRANSFORM

This enables the DRAWING_MAGNIFICATION and DRAWING_TRANSLATION instructions. The instruction can be used with the DRAWING_VAS instruction to turn the transformations on or off as required.

Opcode: 130 Length: 0

Format: DRAWING_TRANSFORM

Parameters: None

End Position: The current drawing position is not changed.

Errors: None

Notes: The transformations are disabled by DRAWING__VAS.

No drawing occurs.

The viewport is not changed.

Example:
```
.BYTE 0.,130.          ;length and opcode
```

7.4 DRAWING__VAS

This disables DRAWING__MAGNIFICATION and DRAWING__TRANSLATION. Subsequent input is in VAS units.

Opcode: 131 Length: 0

Format: DRAWING__VAS

Parameters: None

End Position: The current drawing position is not changed.

Errors: None

Notes: The transformations are reenabled by DRAWING__TRANSFORM.

No drawing occurs. The viewport is not changed.

Example:
```
.BYTE 0.,131.          ;length and opcode
```

7.5 SCREEN__DIMENSIONS

Defines the screen dimensions in logical pixels.

Opcode: 18 Length: 2

Format: SCREEN__DIMENSIONS width, height

Parameters: width = width of display in logical pixels
height = height of display in logical pixels

End Position: The current position in VAS is the window origin. The drawing position in screen terms is at the bottom left-hand corner.

Errors: Error if a parameter is invalid and display processing stops.

Error if total magnification exceeds maximum (see ZOOM__FACTOR, Section 7.7) and display processing stops.

Notes: The valid sets of values for the parameters xd, yd are:

512, 256
640, 240
512, 512
640, 480

If the screen X dimension is changed from 512 to 640 or from 640 to 512, the display image and the screen are cleared to the current background color. The screen contents are not affected otherwise.

The viewport is reset to the full screen image. The magnification factors remain the same. The window origin is unchanged. The window extent is adjusted to reflect the change in the viewport.

The default screen is 640 × 480 logical pixels.

Example:
```
.BYTE 2.,18.        ;length and opcode
.WORD 640.          ;set width to 640
.WORD 240.          ;set height to 240
```

7.6 WINDOW_ORIGIN

Sets the window origin to a VAS position. This defines a window which may be projected into the VSV21 viewport in conjunction with the ZOOM_FACTOR and SET_VIEWPORT instructions (Sections 7.7 and 7.8).

Opcode: 17 Length: 2

Format: WINDOW_ORIGIN x, y

Parameters:
x = X coordinate of the window origin in VAS
y = Y coordinate of the window origin in VAS

End Position: The current drawing position is set to the window origin.

Errors: None

Notes: No drawing occurs. The viewport is not changed.

Example:
```
.BYTE 2.,17.        ;length and opcode
.WORD 50.           ;X coordinate of window origin
.WORD 200.          ;Y coordinate of window origin
```

7.7 ZOOM_FACTOR

This defines the horizontal and vertical magnification factors for the zoom facility. It defines the mapping between the window and the viewport. The window extent is defined by the relationship between the viewport extent and the zoom factors.

Opcode: 19 Length: 2

Format: ZOOM_FACTOR xmag, ymag

Parameters: xmag = X direction magnification factor
ymag = Y direction magnification factor

End Position: The current position in VAS is set to the window origin.

Errors: Error if total magnification exceeds 127. Display processing stops. The start and resume QIOs do not reset the transformations, so subsequent drawing is unpredictable until magnification returns to within the permitted limits.

Notes: No drawing occurs.

The viewport is not changed. The window extent is adjusted accordingly.

Example:

```
.BYTE 2.,19.        ;length and opcode
.WORD 100.          ;magnify times 100 in
                    ;horizontal
.WORD 50.           ;magnify times 50 in vertical
```

7.8 SET_VIEWPORT

This instruction defines a screen area to which drawing is restricted. The area units are as defined by the SCREEN_DIMENSIONS instruction (Section 7.5).

The default viewport is the display image. However, the viewport may be reduced so that the segment contents generate the image only to a reduced area defined by the viewport.

Opcode: 20 Length: 0 or 4

Format: SET_VIEWPORT [xmin, ymin, width, height]

Parameters: xmin = X coordinate of lower left corner
ymin = Y coordinate of lower left corner
width = width of viewport in logical pixels
height = height of viewport in logical pixels

End Position: The current position in VAS becomes the window origin.

Errors: Warning if coordinates are out of range.

Notes: If no parameters are supplied, or all the parameters are zero, the viewport is set to the boundaries of the screen. The viewport is restricted to the screen display area. The viewport origin is mapped to the window origin. The window is unchanged and the zoom factor is adjusted accordingly.

Example:

```
.BYTE 4,.20.        ;opcode and non-default
                    ;length
.WORD 50.           ;lower left X value
.WORD 40.           ;lower left Y value
.WORD 200.          ;viewport width
.WORD 300.          ;viewport height
```

7.9 SET_WINDOW

This instruction defines a window in VIVID Address Space. The window is mapped automatically to the viewport.

Opcode: 23 Length: 4

Format: SET_WINDOW xw, yw, width, height

Parameters:
xw = X coordinate of lower left corner of window in VAS
yw = Y coordinate of lower left corner of window in VAS
width = width of window in VAS
height = height of window in VAS

End Position: The current position in VAS is set to the window origin.

Errors: Warning if co-ordinates are out of range.

Notes: This instruction provides an alternative to ZOOM_FACTORS for mapping the viewport.

No drawing occurs. The viewport is not changed and the zoom factor is adjusted accordingly.

Example:

```
.BYTE 4,.23.        ;length and opcode
.WORD 50.           ;lower left X value
.WORD 40.           ;lower left Y value
.WORD 200.          ;width
.WORD 100.          ;height
```

CHAPTER 8
GLOBAL ATTRIBUTE INSTRUCTIONS

This chapter contains a description of each VIVID global attribute instruction. Opcodes are given in decimal. A MACRO-32 example of each instruction is provided.

8.1 SCREEN_BLINK

This instruction enables or disables screen blinking.

Opcode: 25 Length: 1

Format: SCREEN_BLINK bmod

Parameters: bmod = blink mode (on/off)
0 : screen blink off
non-zero : screen blink on

End Position: The current position is not changed.

Errors: Fatal error if length is incorrect.

Notes: When this command is executed, the colors specified in the NORMAL_COLORS command (Section 8.6) are alternated with those specified in the BLINK_COLORS command (Section 8.7). The interval is specified by the BLINK_TIMING instruction (Section 8.2). Blinking continues for the whole screen until it is disabled.

Example:
```
.BYTE 1.,25          ;length and opcode
.WORD 0.             ;screen blink off
```

8.2 BLINK_TIMING

This sets screen blink timings.

Opcode: 35 Length: 2

Format: BLINK_TIMING norm, blnk

Parameters: norm = number of frames of normal colors (range 8-64)
blink = number of frames of blink colors (range 8-64)

End Position: The current position is not changed.

Errors: Fatal error if length is incorrect.

Warning if either norm or blink is outside the range 8-64. The instruction is ignored.

Notes: No screen blink occurs until it is enabled by the SCREEN_BLINK instruction (Section 8.1). If screen blink is currently enabled, the timing changes apply immediately. Until a BLINK_TIMING instruction is encountered, blink on/off occurs at 0.5 second intervals.

The frame counts are rounded down to multiples of four.

Example:

```
.BYTE 2.,35.        ;length and opcode
.WORD 48.           ;48 frames of normal colors
.WORD 24.           ;24 frames of blink colors
```

8.3 SCREEN_BLANK

This instruction enables or disables screen blanking. Screen blanking gives priority to drawing rather than display. This allows drawing speed to increase by a factor of up to 4.

Opcode: 26 Length: 1

Format: SCREEN_BLANK bmod

Parameters: bmod = screen mode (blank/not blank)
0 : screen not blank. Display has priority.
non-zero : screen blank. Drawing has priority.

End Position: The current position is not changed.

Errors: None

Notes: The screen is blanked and drawing speed is increased by a factor of up to 4 on a high-resolution system. This is useful for drawing a new picture quickly. On low-resolution monitors, the gain in speed is negligible.

The screen remains blank until another SCREEN_BLANK instruction with bmod = 0 is encountered.

Example:
```
.BYTE 1.,26.        ;length and opcode
.WORD 1.            ;screen blank
```

8.4 FOREGROUND_COLOR

This sets the foreground color to be used for subsequent drawing.

Opcode: 27 Length: 1

Format: FOREGROUND_COLOR ind

Parameters: ind = color index number in color look-up table

End Position: The current position is not changed.

Errors: If the index is outside the range 0 to 15, a warning is issued and the foreground color is not changed.

Notes: The index to the Color Look-up Table (CLUT) identifies which of the 16 available colors are used. Color 15 is the default. The CLUT is briefly described in Section 8.6. For a fuller description of the CLUT, see Section 18.5.1.

Example:
```
.BYTE 1.,27.        ;length and opcode
.WORD 12.           ;color 12 from the CLUT
```

8.5 BACKGROUND_COLOR

This instruction sets the background color to be used for subsequent drawing.

Opcode: 28 Length: 1

Format: BACKGROUND_COLOR ind

Parameters: ind = color index number in color look-up table

End Position: The current position is not changed.

Errors: If the index is outside the range 0 to 15, a warning is issued and the foreground color is not changed.

Notes: The index to the color look-up table (CLUT) identifies which of the 16 available colors are used. Color 0 is the default. The CLUT is described briefly in Section 8.6. For a full description of the CLUT, see Section 18.5.1.

Example:
```
.BYTE 1.,28.          ;length and opcode
.WORD 10              ;color 10 from CLUT
```

8.6 NORMAL_COLORS

This sets up to 16 colors (in terms of index and red, green and blue intensities) in the color look-up table (CLUT).

Opcode: 29 Length: 2 × number of colors or 255

Format: NORMAL_COLORS ind_1, int_1 [, ind_2, int_2, ... ind_n, int_n]
where n = 1-16

Parameters: ind_n = color index number in CLUT
int_n = intensities of red, green and blue

End Position: The current position is not changed.

Errors: Fatal error if the index or intensity is out of range.

Notes: The color look-up table (CLUT) contains 16 entries with indices 0 to 15. Each color is stored in terms of red, green, and blue intensities in the range 0 – 15, specified by the parameter int. CLUT entries not referenced in the parameter list are not changed.

If you want to use other colors, it is recommended that you use a VCP command file or a segment to initialize the CLUT, blink table and blink count and to set up your own standard table. The method is described in Section 18.5.1.

The new colors are applied to all previous drawing on the monitor screen. Only one NORMAL_COLORS update occurs per frame, so it is quickest to include the commands for all the required CLUT updates in a single instruction.

Where the maximum command length of 255 is used, the parameter list must be terminated with an END_PARAMETERS delimiter (Section 3.3.1).

The CLUT is described in detail in Section 18.5.1.

Example: The color values used in this example are those used in the CLUT example given in Section 18.5.1.

```
.BYTE 10.,29.        ;length and opcode - 5 colors
.WORD 1.             ;CLUT entry #1
.WORD ^X0F0          ;green = red 0, green maximum,
                     ;blue 0
.WORD 2.             ;CLUT entry #2
.WORD ^X0B8          ;cyan
.WORD 3.
.WORD ^X06C          ;turquoise
.WORD 4.
.WORD ^XF00          ;red
.WORD 5.
.WORD ^XA0A          ;magenta
```

8.7 BLINK_COLORS

This defines CLUT colors (normal colors) and alternate colors (blink colors) for blinking.

When a SCREEN_BLINK command has enabled blinking, the normal colors are alternated with blink colors from the blink color look-up table (BCLUT) described in Section 18.4.2, as defined by BLINK_COLORS and by BLINK_COUNT (Section 8.8).

This command lists the indices of the normal colors which are to be alternated while blinking. For each normal color it provides a blink color, defined in terms of red, green and blue intensities.

The number of colors blinked is determined by BLINK_COUNT. You can change the number of colors blinking by changing the BLINK_COUNT parameter.

Opcode: 30 Length: 3 × number of entries or 255

Format: BLINK_COLORS $bind_1$, ind_1, int_1, $bind_2$, ind_2, int_2 ... $bind_n$, ind_n, int_n
where n = 1-16

Parameters:
$bind_n$ = color index in blink colors look-up table (BCLUT)
ind_n = corresponding CLUT index
int_n = red, green and blue intensity code (range 0-15)

End Position: The current position is not changed.

Errors: Fatal error if parameter out of range.

Notes: The maximum number of entries in the BCLUT is 16.

There are no default BCLUT settings in VIVID. It is recommended that you set these up using a VCP command file or segment.

If a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (Section 3.3.1).

Example: The color values used in this example are those used in the CLUT example given in Section 18.5.1.

```
.BYTE 9.,30.        ;length and opcode - 3 colors
.WORD 1.            ;BCLUT entry #1
.WORD 3.            ;alternating CLUT entry
                    ;(turquoise in example)
.WORD ^X00F         ;BCLUT color blue
                    ;
.WORD 2.            ;BCLUT entry #2
.WORD 5.            ;alternating CLUT entry
                    ;(black)
.WORD ^X880         ;BCLUT color yellow
                    ;
.WORD 3.            ;BCLUT entry #3
.WORD 4.            ;alternating CLUT entry
                    ;(red)
.WORD ^X06C         ;turquoise
```

8.8 BLINK_COUNT

This defines the number of colors that blink when blink is enabled.

Opcode: 31 Length: 1

Format: BLINK_COUNT ncol

Parameters: ncol = number of colors to blink

End Position: The current position is not changed.

Errors: If the count is outside the range 0 to 16, a warning is issued and the command is ignored.

Notes: VIVID has no default blink colors. It uses whatever CLUT and BCLUT colors exist when it is loaded. VCP command file or display segment is recommended for setting up your own defaults (Section 8.6).

Example:

```
.BYTE 1.,31.        ;length and opcode
.WORD 2.            ;two colors can blink
```

8.9 DRAWING_MODE

This instruction sets the drawing mode so that subsequent drawing operations do one of the following:

- Replace the display image unconditionally
- Replace the display image depending on the outcome of a logical operation on the frame buffer contents
- Replace the display image depending on the outcome of a logical or arithmetic comparison between the frame buffer and the drawing or comparison color

The image may be drawn in either the foreground color or the background color, or both.

Opcode: 32 Length: 2 or 3

Format: DRAWING_MODE cmod, pmod [, ccol]

Parameters:
cmod = color mode:
0 : draw foreground and background
1 : draw foreground only; this is the default
2 : draw background only

pmod = operational mode:
0 : replace display image; this is the default
1 : OR to display image
2 : AND to display image
3 : EOR to display image
4 : replace if display color = ccol
5 : replace if display color ≠ ccol
6 : replace if display color < draw color
7 : replace if display color > draw color

ccol = comparison color for pmod values of 4 and 5. A ccol value provided for any other pmod value is ignored.

End Position: The current position is not changed.

Errors: If a parameter is outside the specified range, a warning is issued and the parameter is ignored.

Example:

```
.BYTE 3.,32.        ;length and opcode
.WORD 0.            ;draw foreground and
                    ;background
.WORD 4.            ;replace if
                    ;display color = ccol
.WORD 4.            ;CLUT color number 4
```

8.10 LINE_TEXTURE

This instruction defines the line texture. This is a bit pattern that is repeated in the drawn lines.

Opcode: 33 Length: 2

Format: LINE_TEXTURE nbit, fbcod

Parameters: nbit = number of bits in bit pattern
fbcod = bit pattern for background or foreground colors

End Position: The current position is not changed.

Errors: If nbit is outside the range 1 to 16, a warning is issued and the instruction is ignored.

Notes: The bit pattern represents foreground and background colors to be used in line drawing instructions. The line begins at bit 0 and continues for the number of bits specified, after which bit 0 is used again. A bit set to 1 is drawn in the foreground color and a bit set to zero is drawn in the background color.

Each line drawing instruction continues from the point reached by the previous line drawing instruction. To reset to the beginning of the bit pattern, a further LINE_TEXTURE instruction must be issued. The default line texture is solid foreground color.

The line texture is magnified by the drawing magnification defined at the time of the LINE_TEXTURE instruction. Later changes to the relative magnification must be followed by another LINE_TEXTURE instruction if the new drawing magnification is to be applied.

Example:

```
.BYTE 2.,33.              ;length and opcode
.WORD 12.                 ;12 bits in pattern
.WORD ^B100110010011      ;sets up bit pattern
                          ;line pattern is
                          ;  "o  oo  o  oo"
```

8.11 AREA_TEXTURE

This defines a cell containing the area texture pattern.

Opcode: 34 Length: 1 + number of rows

Format: AREA_TEXTURE nbit, $patt_1$ [, $patt_2$, ... $patt_n$]
where n = 1-16

8.9 DRAWING_MODE

This instruction sets the drawing mode so that subsequent drawing operations do one of the following:

- Replace the display image unconditionally
- Replace the display image depending on the outcome of a logical operation on the frame buffer contents
- Replace the display image depending on the outcome of a logical or arithmetic comparison between the frame buffer and the drawing or comparison color

The image may be drawn in either the foreground color or the background color, or both.

Opcode: 32 Length: 2 or 3

Format: DRAWING_MODE cmod, pmod [, ccol]

Parameters:
cmod = color mode:
0 : draw foreground and background
1 : draw foreground only; this is the default
2 : draw background only
pmod = operational mode:
0 : replace display image; this is the default
1 : OR to display image
2 : AND to display image
3 : EOR to display image
4 : replace if display color = ccol
5 : replace if display color ≠ ccol
6 : replace if display color < draw color
7 : replace if display color > draw color
ccol = comparison color for pmod values of 4 and 5. A ccol value provided for any other pmod value is ignored.

End Position: The current position is not changed.

Errors: If a parameter is outside the specified range, a warning is issued and the parameter is ignored.

Example:

```
.BYTE 3.,32.        ;length and opcode
.WORD 0.            ;draw foreground and
                    ;background
.WORD 4.            ;replace if
                    ;display color = ccol
.WORD 4.            ;CLUT color number 4
```

8.10 LINE_TEXTURE

This instruction defines the line texture. This is a bit pattern that is repeated in the drawn lines.

Opcode: 33 Length: 2

Format: LINE_TEXTURE nbit, fbcod

Parameters: nbit = number of bits in bit pattern
fbcod = bit pattern for background or foreground colors

End Position: The current position is not changed.

Errors: If nbit is outside the range 1 to 16, a warning is issued and the instruction is ignored.

Notes: The bit pattern represents foreground and background colors to be used in line drawing instructions. The line begins at bit 0 and continues for the number of bits specified, after which bit 0 is used again. A bit set to 1 is drawn in the foreground color and a bit set to zero is drawn in the background color.

Each line drawing instruction continues from the point reached by the previous line drawing instruction. To reset to the beginning of the bit pattern, a further LINE_TEXTURE instruction must be issued. The default line texture is solid foreground color.

The line texture is magnified by the drawing magnification defined at the time of the LINE_TEXTURE instruction. Later changes to the relative magnification must be followed by another LINE_TEXTURE instruction if the new drawing magnification is to be applied.

Example:

```
.BYTE 2.,33.            ;length and opcode
.WORD 12.               ;12 bits in pattern
.WORD ^B100110010011    ;sets up bit pattern
                        ;line pattern is
                        ;  "o  oo  o  oo"
```

8.11 AREA_TEXTURE

This defines a cell containing the area texture pattern.

Opcode: 34 Length: 1 + number of rows

Format: AREA_TEXTURE nbit, $patt_1$ [, $patt_2$, ... $patt_n$]
where n = 1-16

Parameters: nbit = number of bits in bit pattern
$patt_n$ = bit pattern of row n
0 : background
1 : foreground

End Position: The current position is not changed.

Errors: If nbit is outside the range 1 to 16, a warning is issued and the instruction is ignored. If the number of parameters is outside the range 2 to 16, there is a fatal error.

Notes: This instruction is similar in function to the LINE_TEXTURE instruction (Section 8.10). Rows in the pattern are ascending; the first row appears at the bottom of the screen. The number of rows is given by the instruction length.

The pattern is replicated evenly throughout the filled area, and the start point in the pattern is reset for each filled area instruction processed. The current drawing magnification and zoom factors are applied. The default area texture is solid foreground color.

The area texture is magnified by the drawing magnification defined at the time of the AREA_TEXTURE instruction. Later changes to the magnification must be followed by another AREA_TEXTURE instruction if the new drawing magnification is to be applied.

Example:

```
.BYTE 4.,34.          ;length and opcode (3 rows)
.WORD 12.             ;12 bits in pattern
                      ;
.WORD ^B10011001001100    ;sets up bit pattern
                      ; line pattern is "o  oo  o  oo  "
                      ;
.WORD ^B01001100100110    ;sets bit pattern
                      ; line pattern is " o  oo  o  oo "
                      ;
.WORD ^B00100110010011    ;set bit pattern
                      ; line pattern is "  o  oo  o  oo"

.WORD ^B10010011001001    ;set bit pattern
                      ; line pattern is "o  o  oo  o  o"
                      ;
                      ;texture is
                      ;  o  oo  o  oo  o  oo  o  oo  o
                      ;   o  oo  o  oo  o  oo  o  oo  o
                      ;    o  oo  o  oo  o  oo  o  oo  o
                      ;  o  o  oo  o  oo  o  oo  o  oo  o
```

CHAPTER 9
DRAWING INSTRUCTIONS

This chapter contains a description of each VIVID drawing instruction. Opcodes are given in decimal. A MACRO-32 example of each instruction is provided.

Many of the instructions described in this chapter can be supplied with 255 in the instruction length byte. This is a code which indicates that the instruction length is undefined and that the parameter list will be terminated by the END_PARAMETERS delimiter (hex 8000).

The line drawing instructions do not draw the last point in the line. This is so that this point can be the first point in the next drawn line. Use the DOT instruction (Section 9.16) to draw the last pixel.

9.1 MOVE_ABS

This moves the current drawing position to the absolute location specified.

Opcode: 38 Length: 2

Format: MOVE_ABS x, y

Parameters: x = X coordinate in VAS
y = Y coordinate in VAS

End Position: As defined by x, y

Errors: None

Notes: The position may be outside the screen image boundaries.

No drawing is performed.

Example:
```
.BYTE 2.,38.        ;length and opcode
.WORD 100.          ;X coordinate
.WORD 200.          ;Y coordinate
```

9.2 MOVE_REL

This instruction moves the current drawing position to the relative position specified.

Opcode: 39 Length: 2

Format: MOVE_REL dx, dy

Parameters: dx = X displacement from the current position
dy = Y displacement from the current position

End Position: As defined by the previous position and the new coordinates.

Errors: None

Notes: The position may be outside the viewport boundaries.

No drawing is performed.

Example:
```
.BYTE 2.,39.        ;length and opcode
.WORD 15.           ;X displacement
.WORD 60.           ;Y displacement
```

9.3 MOVE_TO_CURSOR

This instruction gives a move to the current cursor position.

Opcode: 53 Length: 0

Format: MOVE_TO_CURSOR

Parameters: None

End Position: The current cursor position.

Errors: None

Notes: The position is always within the screen boundaries.

No drawing is performed.

Example:
```
.BYTE 0.,53.        ;length and opcode
```

9.4 LINES_ABS

This draws the specified sequence of lines, starting from the current position.

The first line begins at the current position. Subsequent lines are drawn from the end of the previous line to the next position specified in VAS.

Opcode: 40 Length: 2 × number of lines, or 255

Format: LINES_ABS x_1, y_1[, x_2, y_2, ... x_n, y_n]
where n has no defined limit

Parameters: x_n = X coordinate for the end of the line
y_n = Y coordinate for the end of the line

End Position: As defined by the final coordinates

Errors: None

Notes: The lines are drawn in the current drawing mode with the current line texture. No drawing occurs outside the current viewport, although the drawing position may move outside the viewport.

The last pixel is not drawn.

Where a command length of 255 is used, the parameter list must be terminated by END_PARAMETERS (Section 3.3.1).

Example:

```
.BYTE 4.,40.        ;length and opcode
.WORD 60.           ;X coordinate and
.WORD 20.           ;Y coordinate
                    ;for end of first line
.WORD 45.           ;X coordinate and
.WORD 50.           ;Y coordinate
                    ;for end of second line
```

9.5 LINES_REL

This instruction draws the specified sequence of lines, starting from the current position.

The first line begins at the current position. Later lines are drawn from the end of the previous line to the next position specified as a dx, dy displacement pair of coordinates.

Opcode: 41 Length: 2 × number of lines or 255

Format: LINES_REL dx_1, dy_1 [, dx_2, dy_2 ... dx_n, dy_n]
where n has no defined limit

Parameters: dx_n = X displacement for the next end vector
dy_n = Y displacement for the next end vector

End Position: As defined by the final coordinates.

Errors: None

Notes: The lines are drawn in the current drawing mode with the current line texture. No drawing occurs outside the current viewport, although the drawing position may move outside the viewport.

The last pixel is not drawn.

Where a command length of 255 is used, the parameter list must be terminated by END_PARAMETERS (Section 3.3.1).

Example:
```
.BYTE 6.,41.        ;length and opcode
.WORD 10.           ;X displacement
.WORD 20.           ;Y displacement
.WORD 15.           ;X displacement
.WORD 40.           ;Y displacement
.WORD -35.          ;X displacement
.WORD -5.           ;Y displacement
```

9.6 POLYMARKS_ABS

This draws the specified character from the current font at each point given by a list of X, Y coordinate pairs.

The character cell specified by CELL_SIZE is used and centered at the specified position. The parameters given with the CELL_MOVEMENT command are ignored, but the CELL_OBLIQUE, CELL_ROTATION and CELL_MAGNIFICATION parameters are applied. These commands are described in Chapter 11.

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (Section 3.3.1).

Opcode: 42 Length: 1 + (2 × number of points), or 255

Format: POLYMARKS_ABS ichar, x_1, y_1 [, x_2, y_2,... x_n, y_n]
where n has no defined limit

Parameters: ichar = index of character required
x_n = X coordinate in VAS
y_n = Y coordinate in VAS

End Position: The final position specified. If there is an error, no drawing occurs and the final position is unchanged.

Errors: Error if index of character is out of range.

Notes: No drawing occurs outside the viewport.

The character cell is centered on the specified position. No marker is drawn at the starting position. If this is required, the initial displacement must be (0, 0).

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (Section 3.3.1)

Example:

```
.BYTE 255.,42.      ;length (undefined) and opcode
.WORD 9.            ;index of character
.WORD 100.          ;X coordinate of 1st position
.WORD 200.          ;Y coordinate of 1st position
.WORD 300.          ;X coordinate of 2nd position
.WORD 200.          ;Y coordinate of 2nd position
      .
      .
      .

.WORD 32768.        ;END_PARAMETERS
```

9.7 POLYMARKS_REL

This draws the character specified from the current font at each of the points specified by a list of X, Y displacements.

The character cell specified by CELL_SIZE is used and centered at the specified position. The parameters given with the CELL_MOVEMENT command are ignored, but the CELL_OBLIQUE, CELL_ROTATION and CELL_MAGNIFICATION parameters are applied. These commands are described in Chapter 11.

Opcode: 43 Length: 1 + (2 × number of points), or 255

Format: POLYMARKS_REL ichar, dx_1, dy_1[, dx_2, dy_2 ... dx_n, dy_n]
where n has no defined limit

Parameters: ichar = index number of character required
dx_n = X displacement
dy_n = Y displacement

End Position: The final position specified. If there is an error, the final position is unchanged.

Errors: Error if index of character is out of range.

Notes: No drawing occurs outside the viewport.

No marker is drawn at the starting position. If this is required, the initial displacement must be (0, 0). Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (Section 3.3.1)

Example:
```
.BYTE 4.,43.        ;length and opcode
.WORD 100.          ;X displacement
.WORD 200.          ;Y displacement
.WORD -50.          ;X displacement
.WORD 20.           ;Y displacement
```

9.8 ARCS_ABS

This draws the specified sequence of circular arcs starting from the current position.

Each arc continues from the last. It is defined in terms of its center and end position in X, Y coordinates in VAS. The arcs are drawn in the current drawing mode with the current line texture.

The radius of the arc is the distance between its center and the starting position. The specified end point should be on the circumference of the arc. If it is not, a straight line is drawn from the circumference to the end point.

Opcode: 44 Length: 5 × number of arcs, or 255

Format: ARCS_ABS dir_1, $xcen_1$, $ycen_1$, $xend_2$, $yend_2$
[,dir_2, $xcen_2$, $ycen_2$, $xend_2$, $yend_2$ ...
dir_n, $xcen_n$, $ycen_n$, $xend_n$, $yend_n$]
where n has no defined limit

Parameters:
dir_n = drawing direction
0 : counterclockwise
1 : clockwise
$xcen_n$ = X coordinate of center in VAS
$ycen_n$ = Y coordinate of center in VAS
$xend_n$ = X coordinate of end position in VAS
$yend_n$ = Y coordinate of end position in VAS

End Position: As defined by the final coordinates.

Errors: A fatal error occurs if the transformed (pixel) values of the parameters do not have the following relationship:

$$SQRT\ (XC^2 + YC^2) \leq 4095 / (MAX(A,B))^2$$

where A = transformed relative X length in pixels
B = transformed relative Y length in pixels
XC = transformed value of $xcen_n - xc$
YC = transformed value of $ycen_n - yc$
where (xc,yc) are the coordinates of the current position

These transformed values A and B are calculated as follows:

1. Take the total magnifications in X and Y directions respectively to obtain A and B.

2. Divide both A and B by the largest integral power of two such that A $\geq$1 and B $\geq$1.

A fatal error occurs if the end point of a circular or elliptic arc falls within any of the shaded areas of Figure 9-1. The points A,B,C and D are points at which lines of gradient +1 and −1 are tangential to the arc. Figure 9-1 shows a circle of which the arc is a part. The same principle applies to elliptic arcs.

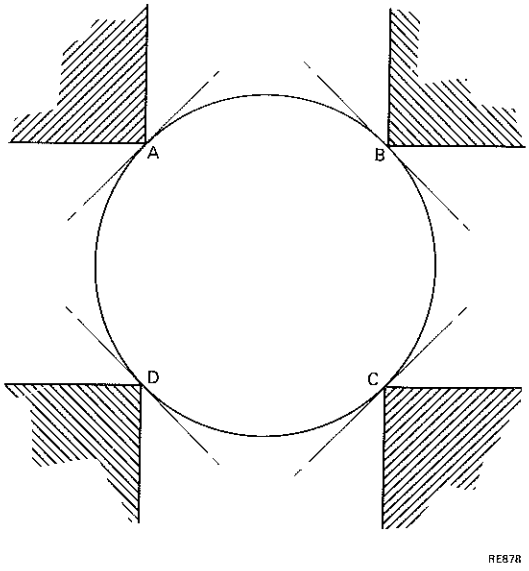


Figure 9-1 Error areas for end points of circular arcs

Notes: No drawing occurs outside the current viewport, though the drawing position may move outside the viewport.

The parameters should be chosen such that they specify a circular arc; the distance from the current position to the center should equal the distance from the end point to the center. The radius should be positive and no greater than 4K VAS units after all transformations have been applied.

The last pixel is not drawn.

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (see Section 3.3.1)

Example:

```
.BYTE 10.,44.        ;length and opcode
                     ;first arc
.WORD 0.             ;direction counterclockwise
.WORD 200.           ;X coordinate of center
.WORD 30.            ;Y coordinate of center
.WORD 70.            ;X coordinate of end position
.WORD 10.            ;Y coordinate of end position
                     ;
                     ;second arc
.WORD 1.             ;direction clockwise
.WORD 10.            ;X coordinate of center
.WORD 300.           ;Y coordinate of center
.WORD 300.           ;X coordinate of end position
.WORD 360.           ;Y coordinate of end position
```

9.9 ARCS_REL

This instruction draws the specified sequence of circular arcs, starting from the current position.

The first arc is defined in terms of its center and the displacement of its end from the current position. Each later arc is defined in terms of its center and of the displacement of its end from the end position of the previous arc. The arcs are drawn in the current drawing mode with the current line texture.

The radius of the arc is the absolute distance between its center and the starting position. If the specified end point is not on the circumference of the arc, a straight line is drawn from the circumference to the end point.

Opcode: 45 Length: 5 × number of arcs, or 255

Format: ARCS_REL dir_1, $xcen_1$, $ycen_1$, $xend_2$, $yend_2$
[,dir_2, $xcen_2$, $ycen_2$, $xend_2$, $yend_2$ ...
dir_n, $xcen_n$, $ycen_n$, $xend_n$, $yend_n$
where n has no defined limit

Parameters: dir_n = drawing direction
0 : counterclockwise
1 : clockwise
$xcen_n$ = X displacement of center
$ycen_n$ = Y displacement of center
$xend_n$ = X displacement of end position
$yend_n$ = Y displacement of end position

End Position: As defined by the final coordinates.

Errors: As for ARCS_ABS (Section 9.8).

Notes: No drawing occurs outside the current viewport, although the drawing position may move outside the viewport. The last pixel is not drawn. If necessary, you can draw it by using the DOT instruction.

The parameters must be chosen so that they define an arc with a positive radius. The radius should not exceed 4K VAS units after all the transformations have been applied.

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (see Section 3.3.1).

Example:

```
.BYTE 10.,45.       ;length and opcode
                    ;first arc
.WORD 0.            ;direction counterclockwise
.WORD 10.           ;X displacement of center
.WORD -30.          ;Y displacement of center
.WORD -20.          ;X displacement of end point
.WORD -20.          ;Y displacement of end point
                    ;second arc
.WORD 0.            ;direction clockwise
.WORD 50.           ;X displacement of center
.WORD 30.           ;Y displacement of center
.WORD 100           ;X displacement of end point
.WORD 0             ;Y displacement of end point
```

9.10 ELLIPSE_ARCS_ABS

This draws the specified sequence of elliptic arcs, starting from the current position.

Each arc is described in terms of an x:y aspect ratio, and of its center and end position in VAS coordinates. The aspect ratio relates to VAS coordinates and defines the width:height relationship. The arcs are drawn in the current drawing mode with the current line texture.

The specified end point should be on the circumference of the arc. If it is not, a straight line is drawn from the circumference to the end point.

Opcode: 46 Length: 7 × number of arcs or 255

Format: ELLIPSE_ARCS_ABS dir_1, ax_1, ay_1, $xcen_1$, $ycen_1$, $xend_1$, $yend_1$
[dir_2, ax_2, ay_2, $xcen_2$, $ycen_2$, $xend_2$, $yend_2$ ...
dir_n,ax_n, ay_n, $xcen_n$, $ycen_n$, $xend_n$, $yend_n$]
where n has no defined limit

Parameters:
dir_n = drawing direction
0 : counterclockwise
1 : clockwise
ax_n = relative X length
bx_n = relative Y length
$xcen_n$ = X coordinate of center in VAS
$ycen_n$ = Y coordinate of center in VAS
$xend_n$ = X coordinate of end position in VAS
$yend_n$ = Y coordinate of end position in VAS

End Position: As defined by the final coordinates.

Errors: A fatal error occurs if the transformed (pixel) values of the parameters do not have the following relationship:

$$SQRT (XC^2 + YC^2) \leq 4095 / (MAX(A,B))^2$$

where A = transformed relative X length in pixels
B = transformed relative Y length in pixels
XC = transformed value of (xcen - cpx)
YC = transformed value of (ycen - cpy)
where (cpx,cpy) are the coordinates of the current position

These transformed values A and B are calculated as follows:

1. Multiply ax and bx by the total magnifications in X and Y directions respectively to obtain A and B.

2. Divide both A and B by the largest integral power of two such that A $\geq$1 and B $\geq$1.

A fatal error occurs if the end point of an elliptic arc falls within any of the shaded areas of Figure 9-2. The points A,B,C and D are points at which lines of gradient +1 and −1 are tangential to the ellipse.

Notes: No drawing occurs outside the current viewport, though the drawing position may move outside the viewport. The last pixel is not drawn (see the DOT instruction).

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (see Section 3.3.1)

Example:

```
.BYTE 14.,46.       ;length and opcode
                    ;first arc
.WORD 0.            ;direction counterclockwise
.WORD 3.            ;relative X length
.WORD 1.            ;relative Y length
.WORD 50.           ;X coordinate of center
.WORD 30.           ;Y coordinate of center
.WORD -10.          ;X coordinate of end position
.WORD 30.           ;Y coordinate of end position
                    ;
                    ;second arc
.WORD 1.            ;direction clockwise
.WORD 4.            ;relative X length
.WORD 1.            ;relative Y length
.WORD -50.          ;X coordinate of center
.WORD 30.           ;Y coordinate of center
.WORD -50.          ;X coordinate of end position
.WORD 20.           ;Y coordinate of end position
```

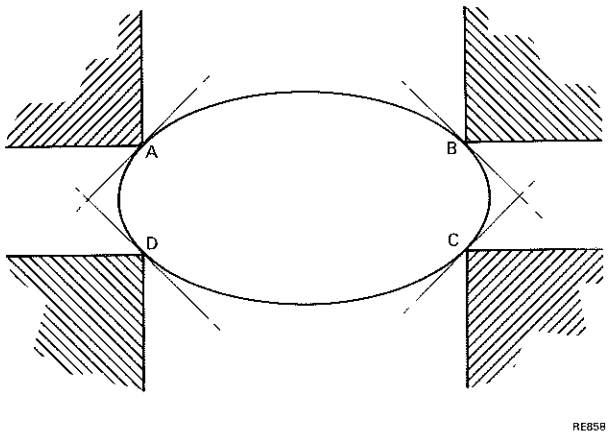


Figure 9-2 Error Areas for End Points of Elliptic Arcs

9.11 ELLIPSE_ARCS_REL

This instruction draws the specified sequence of elliptic arcs, starting from the current position.

Each arc is defined in terms of three parameters, as follows:

- Aspect ratio
- Position of its center
- Displacement of its end from the end of the previous arc

The first arc is drawn from the current position. The aspect ratio relates to VAS coordinates and defines the width:height relationship. The arcs are drawn in the current drawing mode with the current line texture.

The specified end point should be on the circumference of the arc. If it is not, a straight line is drawn from the circumference to the end point.

Opcode: 47 Length: 7 × number of arcs, or 255

Format: ELLIPSE_ARCS_REL dir_1, ax_1, ay_1, dxc_1, dyc_1, dxe_1, dye_1
[dir_2, ax_2, ay_2, dxc_2, dyc_2, dxe_2, dye_2,
...dir_n,ax_n, ay_n, dxc_n, dyc_n, dxe_n, dye_n
where n has no defined limit

Parameters:
dir_n = drawing direction
0 : counterclockwise
1 : clockwise
ax_n = relative X length
bx_n = relative Y length
dxc_n = X displacement of center in VAS
dyc_n = Y displacement of center in VAS
dxe_n = X displacement of end position in VAS
dye_n = Y displacement of end position in VAS

End Position: As defined by the final coordinates.

Errors: As for ELLIPSE_ARCS_ABS (Section 9.10).

Notes: No drawing occurs outside the current viewport, though the drawing position may move outside the viewport. The last pixel is not drawn.

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (see Section 3.3.1)

Example:

```
.BYTE 14.,47.       ;length and opcode
                    ;first arc
.WORD 0.            ;direction counterclockwise
.WORD 7.            ;relative X length
.WORD 1.            ;relative Y length
.WORD 0             ;X displacement of center
.WORD -10.          ;Y displacement of center
.WORD -70.          ;X displacement of end point
.WORD -10.          ;Y displacement of end point
                    ;second arc
.WORD 1.            ;direction clockwise
.WORD 1.            ;relative X length
.WORD 5.            ;relative Y length
.WORD 0             ;X displacement of center
.WORD 50.           ;Y displacement of center
.WORD 10.           ;X displacement of end point
.WORD 50.           ;Y displacement of end point
```

9.12 RECTANGLE_ABS

This instruction draws a rectangle from a vertex at the current position to the diagonal vertex specified.

The rectangle is drawn in the current drawing mode with the current line texture.

Opcode: 48 Length: 2

Format: RECTANGLE_ABS x, y

Parameters:
- x = X coordinate in VAS of opposite vertex
- y = Y coordinate in VAS of opposite vertex

End Position: The current position is not changed.

Errors: None

Notes: No drawing occurs outside the current viewport.

Example:

```
.BYTE 2.,48.        ;length and opcode
.WORD 250.          ;X coordinate
.WORD 150.          ;Y coordinate of opposite
                    ;vertex
```

9.13 RECTANGLE_REL

This draws a rectangle from a vertex at the current position to the diagonal vertex specified.

The rectangle is drawn in the current drawing mode with the current line texture.

Opcode: 49 Length: 2

Format: RECTANGLE_REL dx, dy

Parameters: dx = X displacement of opposite vertex
dy = Y displacement of opposite vertex

End Position: The current position is not changed.

Errors: None

Notes: No drawing occurs outside the current viewport.

Example:
```
.BYTE 2.,49.        ;length and opcode
.WORD 150.          ;X displacement
.WORD -30.          ;Y displacement of opposite
                    ;vertex
```

9.14 ELLIPSE

This instruction draws an ellipse of a specified VAS aspect ratio and major axis, with its center on the current position.

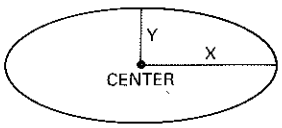


Figure 9-3 Quantities Used to Define an Ellipse

An ellipse is defined by three quantities (Figure 9-3). These are as follows:

- Center, given by the current drawing position
- Aspect ratio. This is the ratio between the lengths of the two axes of the ellipse (X:Y)
- Radius along the X axis

The ellipse is drawn in the current drawing mode with the current line texture.

Opcode: 50 Length: 3

Format: ELLIPSE ax, by, rad

Parameters: ax = relative horizontal length
by = relative vertical length
rad = radius along X axis in VAS

End Position: The current position is not changed.

Errors: A fatal error occurs if the transformed (pixel) values of the parameters do not have the following relationship:

$R \leq 4095 / (MAX(A,B))^2$

where A = X length in pixels
B = Y length in pixels
R = radius along X axis in pixels

These transformed values are calculated as follows:

1. Multiply ax and bx by the total magnifications in X and Y directions respectively to obtain A and B.

2. Divide both A and B by the largest integral power of two such that $A \geq 1$ and $B \geq 1$.

Notes: No drawing occurs outside the current viewport.

Example:

```
.BYTE 3.,50.        ;length and opcode
.WORD 4.            ;relative horizontal
                    ;dimension
.WORD 1.            ;relative vertical dimension
.WORD 50.           ;absolute radius along X axis
```

9.15 CIRCLE

This draws a circle with specified radius, centered on the current position.

The X and Y relative magnifications and zoom factors are applied independently. If the two zoom factors are not equal, the VAS circle appears on the screen as an ellipse.

The circle is drawn in the current drawing mode with the current line texture.

Opcode: 51 Length: 1

Format: CIRCLE rad

Parameters: rad = radius

End Position: The current position is not changed.

Errors: A fatal error occurs if the transformed (pixel) values of the parameters do not have the following relationship:

$R \leq 4095 / (MAX(A,B))^2$

where A = X length in pixels
B = Y length in pixels
R = radius along X axis in pixels

These transformed values are calculated as follows:

1. Take the total magnifications in X and Y directions respectively to obtain A and B.

2. Divide both A and B by the largest integral power of two such that $A \geq 1$ and $B \geq 1$.

Notes: No drawing occurs outside the current viewport.

Example:
```
.BYTE 1.,51.        ;length and opcode
.WORD 50.           ;radius of circle
```

9.16 DOT

A dot is drawn at the current position, in the current drawing mode.

Opcode: 52 Length: 0

Format: DOT

Parameters: None

End Position: The current position is not changed.

Errors: None

Notes: The point designated by the terminating position in the preceding line and arc drawing instructions is not drawn automatically. It must be drawn explicitly with a DOT instruction. This permits the line to be continued with other drawing instructions when in drawing modes such as EOR, where overwriting would cancel the point.

No drawing occurs outside the current viewport.

Example:

```
.BYTE 0.,52.          ;length and opcode
```

CHAPTER 10
FILLED FIGURE INSTRUCTIONS

This chapter contains a description of each VIVID filled figure instruction. Opcodes are given in decimal. A MACRO-32 example of each instruction is provided.

10.1 FILLED_RECT_ABS

A rectangle is drawn from a vertex at the current position to the diagonal vertex specified as an absolute position in VAS. The rectangle is then filled with the area texture pattern.

Opcode: 56 Length: 2

Format: FILLED_RECT_ABS x, y

Parameters: x = X coordinate in VAS of opposite vertex
y = Y coordinate in VAS of opposite vertex

End Position: The current position is not changed.

Errors: None

Notes: No drawing occurs outside the current viewport. After all the transformations have been applied, the extent of the rectangle should not exceed ± 16383 in the X or Y direction.

Example:

```
.BYTE 2.,56.        ;length and opcode
.WORD 100.          ;x=100
.WORD 200.          ;y=200
```

10.2 FILLED_RECT_REL

A rectangle is drawn from a vertex at the current position to the diagonal vertex and filled with the area texture pattern. The diagonal vertex is specified as a displacement from the current position.

Opcode: 57 Length: 2

Format: FILLED_RECT_REL dx, dy

Parameters: dx = horizontal displacement of opposite vertex
dy = vertical displacement of opposite vertex

End Position: The current position is not changed.

Errors: None

Notes: No drawing occurs outside the current viewport. After all the transformations have been applied, the extent of the rectangle should not exceed ± 32767 in the X or Y direction.

Example:
```
.BYTE 2.,57.        ;length and opcode
.WORD 200.          ;xd=200
.WORD 100.          ;yd=100
```

10.3 FLOOD_AREA

This instruction fills the area which includes the current position to the defined edge color, or current foreground color, with the area texture pattern. The area texture pattern is written in Replace mode, irrespective of the current drawing mode.

Opcode: 58 Length: 0 or 1

Format: FLOOD_AREA [ind]

Parameters: ind = CLUT index of edge color to which filling occurs. Range 0 to 15.
−1: defaults to current foreground color

End Position: The current position is not changed.

Errors: A warning is issued if the color parameter is invalid (outside the range −1 to 15), and no flooding occurs.

Notes: The foreground and background colors are also edge colors, and it may be necessary to set them. If the foreground or background color can appear in the area to be filled, it is safer to use the following procedure:

1. Save attributes.
2. Set foreground and background to the same color. This color is otherwise unused.
3. Flood the area to the foreground color.
4. Restore color attributes.
5. Paint the area containing the unused color as selected in step 2.

If the edge color parameter is omitted or is −1, filling occurs to the current foreground color.

No matches are generated by this instruction.

No drawing occurs outside the current viewport.

Example:

```
.BYTE 1.,58.        ;length and opcode
.WORD 10.           ;color 10 from CLUT
```

10.4 PAINT_AREA

This instruction fills the area of the specified color which includes the current position with the area texture pattern.

The current foreground and background colors cannot be used as the specified color. The area texture pattern is written in Replace mode, irrespective of the current drawing mode.

Opcode: 59 Length: 1

Format: PAINT_AREA ind

Parameters: ind = index of color to be replaced

End Position: The current position is not changed.

Errors: Warning if parameter invalid.

Warning if the color to be replaced is the current foreground or background color.

Notes: No matches are generated by this instruction.

No drawing occurs outside the current viewport.

Example:

```
.BYTE 1.,59.        ;length and opcode
.WORD 11.           ;color 11 from CLUT
```

CHAPTER 11
TEXT INSTRUCTIONS

This chapter contains a description of each VIVID text instruction. Opcodes are given in decimal. A MACRO-32 example of each instruction is provided.

The instructions DRAW_CHARS and DRAW_PACKED_CHARS (Sections 11.9 and 11.10) are used to draw characters. The attributes of these characters are defined by the other instructions in this chapter.

The CELL_MAGNIFICATION instruction specifies one of the following character modes:

- Pixel mode

  In this mode, only the cell magnification factors are applied. All dimensions and movements are defined in terms of pixels on the display surface. Consequently, the aspect ratio of the characters will vary according to the resolution of the monitor, as follows:

  - High resolution monitor gives a pixel aspect ratio of 1:1
  - Low resolution monitor gives a pixel aspect ratio of 1:2

  This variation can be corrected by using a Y magnification factor which is twice that of the X factor when a high-resolution monitor is used. The default values used implement this principle. See CELL_MAGNIFICATION (Section 11.7).

  When cells are drawn at angles of 45, 135, 225 or 315 degrees, they appear larger by a factor of 1.414 than those drawn in a horizontal plane. In pixel mode, the current point is maintained true in terms of pixels on the display surface, but not in VAS units.

- Relative mode

The current point is maintained true in terms of VAS units. This is because the parameters entered with the SET_WINDOW, SET_VIEWPORT and ZOOM_FACTOR instructions (Chapter 7) are taken into account. Cells are always drawn to the size nearest the ideal, so the characters drawn at angles which are multiples of 45 degrees will be nominally the same size as those drawn horizontally. This may result in cells overlapping, but this effect can be corrected with the CELL_MOVEMENT instruction. Within the limitations imposed by the monitors, characters are displayed at the same size on monitors of both types.

11.1 INITIALIZE_FONT

This instruction initializes the specified segment as a font, irrespective of the segment contents.

Opcode: 63 Length: 4 or 5

Format: INITIALIZE_FONT segid, width, height, ncell [,init]

Parameters:
segid = segment ID
width = cell width in pixels (1 to 16)
height = cell height in pixels (1 to 16)
ncell = number of cells in the font (>0)
init = initialization style for cells:
0 : blank; this is the default
−1 : solid

End Position: The current position is not changed.

Errors: An error occurs in the following cases:

- Parameter is out of range
- Segment is not large enough
- Segment has not been defined

If there is an error, the segment is not initialized as a font. It retains its original identity.

Example:

```
.BYTE 5.,63.          ;length and opcode
.WORD ^X0F01          ;segment class 15, number 1
.WORD 10.             ;cell width 10 pixels
.WORD 12.             ;cell height 12 pixels
.WORD 36.             ;36 cells in font
.WORD -1.             ;foreground initialization
                      ;style
```

11.2 SET_FONT

This sets the current font to the identified font segment. This font is used for subsequent VIVID instructions which access fonts.

Opcode: 64 Length: 1

Format: SET_FONT segid

Parameters: segid = font segment ID

End Position: The current position is not changed.

Errors: An error occurs in the following cases:

- Segment is not found
- Segment is not a font segment

If there is an error, the current font remains unchanged.

Notes: If no SET_FONT has been encountered, the following rules apply:

- The supplied multinational font is used if it has not been deleted. It has a segment ID of hex 10FF, decimal 4351. Fonts may be downloaded by a VCP command (Section 18.3).
- If the supplied font has been deleted, the current font is undefined. A font reference other than SET_FONT causes an error.

Example:

```
.BYTE 1.,64.          ;length and opcode
.WORD 3500.           ;font number 3500
```

11.3 LOAD_CHAR_CELL

A character cell is loaded to the current font from the pixel data given as parameters.

Opcode: 65 Length: 255 or number of rows + 1

Format: LOAD_CHAR_CELL ind, $irow_1$ [, $irow_2$, ... $irow_n$]
where n = 1-16

Parameters: ind = cell index
$irow_n$ = image value for a pixel row

End Position: The current position is not changed.

Errors: A warning is issued if there are too many rows; excess rows are discarded.

Notes: Rows are in sequence, the first being the bottom row of the cell. Details of the format within a row may be found in Section 2.2. Any rows at the top of the cell not provided are filled solid or blank, depending on font initialization (Section 11.1).

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (Section 3.3.1).

Example:

```
.BYTE 11.,65.          ;length and opcode
.WORD 82.              ;cell index 82
.WORD ^B00000000       ;bottom row of cell
.WORD ^B00000000
                       ;letter "R"
.WORD ^B10000010
.WORD ^B01000010
.WORD ^B00100010
.WORD ^B01111110
.WORD ^B10000010
.WORD ^B10000010
.WORD ^B01111110
.WORD ^B00000000       ;top row of cell
```

11.4 CELL_OBLIQUE

This defines whether subsequent cells are to be drawn rectangularly, or in italic (45-degree slope) form.

Opcode: 66 Length: 1

Format: CELL_OBLIQUE ital

Parameters: ital = parameter for rectangular or italic character
0 : rectangular character
1 : italic character

End Position: The current position is not changed.

Errors: If the parameter is invalid, a warning is generated and oblique is assumed.

Notes: No drawing is performed.

Example:

```
.BYTE 1.,66.           ;length and opcode
.WORD 1.               ;italic character
```

11.5 CELL_ROTATION

This instruction defines the angle at which cells are written to the display image. The angle is defined in 45-degree counterclockwise units.

Opcode: 67 Length: 1

Format: CELL_ROTATION ndeg

Parameters: ndeg = number of 45-degree units of rotation
0 : horizontal
1 : 45 degrees
2 : 90 degrees
3 : 135 degrees
4 : 180 degrees
5 : 225 degrees
6 : 270 degrees
7 : 315 degrees

End Position: The current position is not changed.

Errors: A warning is issued if the parameter is out of range, and zero rotation is assumed.

Notes: Cells rotated at 45, 135, 225 and 315 degrees are distorted on presentation.

No drawing is performed.

Example:
```
.BYTE 1.,67.        ;length and opcode
.WORD 6.            ;rotate cell 270 degrees
                    ;counterclockwise
```

11.6 CELL_SIZE

This defines the length and width of the display image cell and the displacement of the stored font cell within the display image cell.

Opcode: 68 Length: 4

Format: CELL_SIZE width, height, xdis, ydis

Parameters: width = width of display cell in pixels (range 1-16)
height = height of display cell in pixels (range 1-16)
xdis = horizontal displacement of font cell (range 0-15)
ydis = vertical displacement of font cell (range 0-15)

End Position: The current position is not changed.

Errors: Error if parameter out of range. The currently-defined value remains unchanged.

Notes: Any part of the font cell whose dimensions or displacement would place it outside the display cell is truncated. Any part of the display cell not covered by the font cell is set to the font default cell value; that is, all foreground or all background.

If no CELL_SIZE instruction has been encountered, the display cell for any font corresponds to the font dimensions.

Units are applied to the dimensions by CELL_MAGNIFICATION (Section 11.7).

No drawing is performed.

Example:

```
.BYTE 4.,68.        ;length and opcode
.WORD 10.           ;width 10 pixels
.WORD 12.           ;height 12 pixels
.WORD 1.            ;horizontal displacement
.WORD 2.            ;vertical displacement
```

11.7 CELL_MAGNIFICATION

This instruction defines the horizontal and vertical cell magnification, in terms of pixels or relative magnification.

Opcode: 69 Length: 3

Format: CELL_MAGNIFICATION utyp, xmag, ymag

Parameters: utyp = code for magnification unit type
0 : pixels
1 : relative (this is the default)

xmag = magnification in the cell X direction (range 1-16). The default is 1.

ymag = magnification in the cell Y direction (range 1-16) The default is 2.

End Position: The current position is not changed.

Errors: A warning is issued if the maximum magnification of 16 is exceeded. A default magnification of 16 is then used.

Notes: Cell magnification operates in addition to other magnification factors. The maximum total magnification on either axis is 16.

For pixel magnification, the magnification indicates the number of replications of each pixel on subsequent text-outputting instructions.

For relative magnification, the units correspond to the units used in relative drawing (Chapter 9) and viewport/window mapping and zoom factors are multiplied by the cell magnification factors to give the total magnification. Cells are displayed at the nearest size available to the ideal size calculated. If the calculated size is less than 0.5 in either the X or Y direction, the value is rounded to zero and there is no visual output. However, the current point is moved by the appropriate amount.

This can be used to reveal information as the screen is zoomed; text which under certain conditions would not appear may be revealed as the overall magnification factor is increased.

No drawing is performed.

Example:

```
.BYTE 3.,69.      ;length and opcode
.WORD 1.          ;relative magnification
.WORD 4.          ;magnification in cell X direction
.WORD 6.          ;magnification in cell Y direction
```

11.8 CELL_MOVEMENT

This defines a relative movement from the end of one character cell to a final current position.

The relative movement rotates and the distances are altered, as in CELL_ROTATION (Section 11.5).

Opcode: 70 Length: 2

Format: CELL_MOVEMENT xd, yd

Parameters: xd = horizontal displacment
yd = vertical displacement

End Position: The current position is not changed.

Errors: None

Notes: Until a CELL_MOVEMENT instruction is encountered, the default movement sets the drawing position to the start point for a following cell with no gap, irrespective of the rotation.

The distances xd, yd have units as defined by CELL_MAGNIFICATION (Section 11.7).

No drawing is performed.

Example:
```
.BYTE 2.,70.          ;length and opcode
.WORD 1.              ;X movement
.WORD 2.              ;Y movement
```

11.9 DRAW_CHARS

This displays the characters specified by each index in the parameter list. There is one index per word.

Opcode: 71 Length: number of characters, or 255

Format: DRAW_CHARS ind_1[, ind_2, ... ind_n]
where n has no defined limit

Parameters: ind_n = index to cell in font

End Position: Defined by the CELL_MOVEMENT instruction (Section 11.8). If no CELL_MOVEMENT instruction has been encountered, the instruction "CELL_MOVEMENT 0, 0" is implied. The final end position follows the last valid index for which a cell has been written to the display image.

Errors: Error if index is out of range or if font is not currently defined.

Warning if total cell magnification exceeds 16.

Notes: The output uses the current foreground and background colors and the current drawing mode.

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (Section 3.3.1).

No drawing occurs outside the viewport. Match may be detected.

Example:
```
.BYTE 3.,71.          ;length and opcode
.WORD 1.              ;draw character 1
.WORD 20.             ;draw character 20
.WORD 16.             ;draw character 16
```

11.10 DRAW_PACKED_CHARS

This instruction displays the characters specified by each index in the parameter list. Indices are packed two per parameter word.

Opcode: 72 Length: (number of chars+1)/2 or 255

Format: DRAW_PACKED_CHARS i_1 j_1 [,i_2 j_2, ...i_n j_n]
where n has no defined limit

Parameters: i_n j_n = any two characters from byte string

Errors: Error if index is out of range or if font is not currently defined.

Warning if total cell magnification exceeds 16.

Notes: Except for the parameter format, processing is as for DRAW_CHARS (Section 11.9). If the number of characters to be output is odd, use a final END_PARAMETERS index of 255. This will be ignored.

The character defined by the low byte is drawn first.

Example:

```
.BYTE 2.,72.          ;length and opcode
.BYTE 11.,5.          ;character 11,5
.BYTE 6.,12.          ;characters 6,12
```

CHAPTER 12
AREA OPERATION INSTRUCTIONS

This chapter contains a description of each VIVID area operation instruction. Opcodes are given in decimal. A MACRO-32 example of each instruction is provided.

12.1 CLEAR_SCREEN

This instruction clears the display.

Opcode: 76 Length: 0 or 1

Format: CLEAR_SCREEN [patt]

Parameters: patt = list of color indices for screen

End Position: The current position is unchanged.

Errors: None

Notes: If no parameter is supplied, the screen is cleared to the current background color.

The color indices define a repeating four-pixel pattern from left to right along each screen raster. In display order, the indices are in bits 0-3, 4-7, 8-11 and 12-15. See the NORMAL_COLORS command (Section 8.6) for a description of these indices.

The viewport is ignored.

Example:

```
.BYTE 1.,76.        ;length and opcode
.WORD ^X1111        ;clear to color 1
```

12.2 CLEAR_VIEWPORT

This clears the viewport to the current background color.

Opcode: 77 Length: 0

Format: CLEAR_VIEWPORT

Parameters: None

End Position: The current position is set to the window origin.

Errors: None

Example:
```
.BYTE 1.,77.      ;length and opcode
```

12.3 SCROLL_VIEWPORT

This moves the data within the viewport. The data is moved by the indicated displacement. Data falling outside the viewport is lost.

Opcode: 79 Length: 2

Format: SCROLL_VIEWPORT dx, dy

Parameters:
- dx = horizontal displacement of data. Positive values indicate displacement to right
- dy = vertical displacement of data. Positive values indicate upward displacement

End Position: The current position is set to the window origin.

Errors: None

Notes: The display data bounded by the viewport is moved by the displacement specified. The new data replaces the previous display data. The viewport itself is not moved.

The area of the viewport not overlaid by the move is cleared to the current background color.

Example:
```
.BYTE 2.,79.      ;length and opcode
.WORD 50.         ;displace 50 VAS units to right
.WORD -100.       ;and 100 downwards
```

12.4 PIXEL_READBACK

This reads a display image area to a specified segment. This segment is normally in host memory. The segment may be used for subsequent pixel write operations.

Each row of pixel data is an integer number of frame buffer words. A frame buffer word contains four pixels. The start position of the transfer is the frame buffer (display image) word containing the current position.

Opcode: 80 Length: 3

Format: PIXEL_READBACK segid, dxw, dyp

Parameters:
- segid = pixel data map segment ID
- dxw = area width in words (of 4 pixels each). Positive values indicate displacement to right
- dyp = area height in pixels. Positive values indicate upward displacement

End Position: The current position is not changed.

Errors: Error if segment is not found or is too small.

Notes: The segment is initialized as a pixel data segment, irrespective of its former identity.

The area read back to the segment is rectangular.

The pixel data is organized in the segment as specified in Section 3.3.3.

Example:

```
.BYTE 3.,80.      ;length and opcode
.WORD ^XD01       ;segment class 13, number 1
.WORD 25.         ;25 words wide = 100 pixels
.WORD 50.         ;50 pixels high
```

12.5 PIXEL_WRITE

This instruction writes a specified segment containing pixel data to the display image at the current drawing position.

The pixel data is organized as specified in Section 3.3.3. The data runs a number of words (of four pixels each) right or left and a number of pixels upward or downward, depending on the sign. These size parameters are held in the segment header. The data is displayed starting at the current position.

Current magnification factors are applied to the output.

The actual display image output is restricted to the viewport. This feature may be used to remove unwanted pixels in the pixel data map segment.

Opcode: 81 Length: 1

Format: PIXEL_WRITE segid, mag

Parameters: segid = pixel data map segment ID

End Position: The current position is not changed.

Errors: Error if one or more of the following is true:

- Segment is not found
- Segment is of the wrong type
- Segment is too small for its defined contents

Warning if output would exceed display image bounds.

Notes: Data written to the display with this instruction is subject to window and viewport mapping or to zooming. Consequently, the output is independent of the physical dimensions of the monitor; it is controlled by the sprx and spry values entered with the START_PIXEL_DATA instruction (Section 6.3).

Matches may be detected.

Example:
```
.BYTE 2.,81.     ;length and opcode
.WORD ^XD02.     ;segment class 13, number 2
```

12.6 FAST_PIXEL_WRITE

This instruction writes a specified segment containing pixel data from the host or VSV21 memory to the display image, starting at the word (a unit of four pixels) containing the current position.

If the display bounds in the Y axis are exceeded, output is truncated to the screen bounds.

The pixel data is organized as specified in Section 3.3.3 and runs a number of words (of four pixels each) right or left and a number of pixels upward or downward depending on the sign. Positive values denote movement upward or to the right. These size parameters are held in the segment header.

No matches are detected. The viewport is ignored.

Opcode: 82 Length: 1

Format: FAST__PIXEL__WRITE segid

Parameters: segid = pixel data map segment ID

End Position: The current position is not changed.

Errors: Error if one or more of the following is true:

- Segment is not found
- Segment is the wrong type
- Segment is too small for its defined contents

Warning if output would logically exceed display image bounds.

Notes: The area written is logically rectangular. However, if the display image X range is exceeded, the display is wrapped around.

The current position must be within the screen bounds.

Example:
```
.BYTE 1.,82.        ;length and opcode
.WORD ^XD01         ;segment class 13, number 1
```

12.7 FAST__PIXEL__MODIFY

This writes a specified segment which contains pixel data from the host or VSV21 memory to the display image by performing a specified logical operation. It is done starting at the word (a unit of four pixels) containing the current position.

If the display bounds in the X axis are exceeded, wraparound occurs. If the display bounds in the Y axis are exceeded, output is truncated to the screen bounds.

The pixel data is organized as specified in Section 3.3.3 and runs a number of words (of four pixels each) right or left and a number of pixels upward or downward depending on the sign. Positive values denote movement upward or to the right. These size parameters are held in the segment header.

Opcode: 83 Length: 3

Format: FAST_PIXEL_MODIFY segid, mode, mask

Parameters: segid = pixel data map segment ID
mode = operational mode:
0 : replace display image
1 : OR with display image
2 : AND with display image
3 : EOR with display image
mask = word bit mask

End Position: The current position is not changed.

Errors: Error if one of the following is true:

- Segment is not found
- Segment is of the wrong type
- Segment is too small for its defined contents

Warning if output would exceed the address range of the display image frame buffer.

Notes: The parameter mask selects the bits in each word for use in the operation (1 = on, 0 = off). This enables overlays to be written, for example.

The area written is logically rectangular. However, if the display image X range is exceeded, the display is wrapped around.

The current position must be within the screen bounds.

No matches are detected. The viewport is ignored.

Example:

```
.BYTE 3.,83.      ;length and opcode
.WORD ^XD01       ;segment class 13, number 1
.WORD 1.          ;OR with display image
.WORD ^XD4A7      ;mask 1101 0100 1010 0111
```

12.8 SELECTIVE_CLEAR

The specified logical operation is performed on the rectangular area whose opposite vertices are defined by the current position and the specified displacement.

If the display bounds in the Y axis are exceeded, the selective clear is truncated to the screen bounds.

No matches are detected. The viewport is ignored.

Opcode: 84 Length: 4 or 5

Format: SELECTIVE_CLEAR mode, mask, [patt,] dxw, dyp

Parameters:
mode = operational mode:
0 : replace display image
1 : OR withdisplay image
2 : AND with display image
3 : EOR with display image
mask = word bit mask
patt = color bit pattern for 4 pixels
dxw = signed area width in words (of 4 pixels each)
dyp = signed area height in pixels

End Position: The current position is not changed.

Errors: Warning if the selective clear is truncated to the screen bounds.

Notes: Positive values of dxw and dyp indicate movement to the right and upward from the current position.

This instruction performs word operations on the image. The logical operation specified by the mode parameter is performed between the image data and the parameter pattern. Section 3.3.3 describes how the image is stored.

The mask parameter selects the bits in each word to be used in the operation (1 = used, 0 = not used). For example, this enables overlays to be maintained while the rest of the data is cleared, or the reverse.

If the pattern parameter is omitted, the current background color is assumed for each of the four pixels making up the word.

The area cleared is logically rectangular. However, if the display image X range is exceeded, the display wraps around on the screen.

The current position must be within the screen bounds.

No matches are detected. The viewport is ignored.

Example:

```
.BYTE 5.,84.      ;length and opcode
.WORD 1.          ;OR to display image
.WORD ^XD4A7      ;mask 1101 0100 1010 0111
.WORD ^X4A6D      ;colors 4, 10, 6 and 13 from CLUT
.WORD 20.         ;width 20 words = 80 pixels
.WORD 100.        ;height 100 pixels
```

12.9 COPY_ABS

The specified source area is copied to an area with a vertex at the current position and a defined orientation. The origin of the source area is expressed as an absolute position in VAS.

Opcode: 85 Length: 5

Format: COPY_ABS amod, xs, ys, xdim, ydim

Parameters:

amod	= attitude mode
xs	= X position of the source area origin in VAS
ys	= Y position of the source area origin in VAS
xdim	= X dimension of the source copy area in VAS
ydim	= Y dimension of the source copy area in VAS

End Position: The current drawing position is not changed.

Errors: Warning if amod is out of range 0 to 15. The parameter is masked into range.

Notes: The effects of the parameter amod and the signs of xdim and ydim are described in Section 12.11.

Movement may be simulated by overlapping copies so that each new copy performed deletes the pictorial body of the previous copy. If the pictorial element copied has a border, simple dynamics may be effected.

No drawing occurs outside the viewport. The current drawing mode applies.

Example:
```
.BYTE 5.,85.     ;length and opcode
.WORD 1.         ;attitude mode
.WORD 10.        ;X position of source area origin
.WORD 20.        ;Y position of source area origin
.WORD 200.       ;width of source copy area
.WORD 400.       ;height of source copy area
```

12.10 COPY_REL

The parameter-defined source area is copied to an area with a vertex at the current position with a defined attitude. The origin of the source area is expressed relative to the current position.

No drawing occurs outside the viewport. The current drawing mode applies.

Opcode: 86 Length: 5

Format: COPY_REL amod, dxs, dys, xdim, ydim

Parameters:
amod = attitude mode
dxs = X VAS displacement of the source area origin
dys = Y VAS displacement of the source area origin
xdim = X VAS dimension of the source copy area
ydim = Y VAS dimension of the source copy area

End Position: The current drawing position is not changed.

Errors: Warning if amod is out of range 0 to 15. The parameter is masked into range.

Notes: The effects of the parameter amod and the signs of xdim and ydim are described in Section 12.11.

Movement is simulated by overlapping copies so that each new copy performed deletes the pictorial body of the previous copy. If the pictorial element copied has a border, simple dynamics may be effected.

Example:
```
.BYTE 5.,86.     ;length and opcode
.WORD 1.         ;attitude mode
.WORD 25.        ;X displacement of source area
                 ;origin
.WORD 60.        ;Y displacement of source area
                 ;origin
.WORD 200.       ;width of source copy area
.WORD 400.       ;height of source copy area
```

12.11 NOTES ON THE COPY INSTRUCTIONS

The COPY_ABS and COPY_REL instructions copy part of a picture from one area on the screen to another. Attention to the scan directions is necessary to avoid corruption of the destination area when it overlaps the source area.

The picture is copied pixel by pixel. The instruction parameters define the order in which the pixels are read from the original area and the order in which they are written to the new location. This, for example, allows you to transform the picture by rewriting it as a mirror image or upside down.

The parameter amod defines two things:

- The order of the source scan, as follows:

amod	Scan direction
0 – 7	row by row, bottom to top
8 – 15	column by column, left to right

- The direction of the destination scan

  Unlike the source scan, the destination scan is not restricted to any basic directions. The pixels may be written to the destination in a total of eight ways (Table 12-1). The conventions adopted in Table 12-1 are as follows:

  +X = left to right

  −X = right to left

  +Y = upwards

  −Y = downwards

Table 12-1 Order of Pixel Write to Destination Area by amod Value

Value of amod	Scan direction first	second
0 or 8	+X	+Y
1 or 9	+X	−Y
2 or 10	−X	+Y
3 or 11	−X	−Y
4 or 12	+Y	+X
5 or 13	−Y	+X
6 or 14	+Y	−X
7 or 15	−Y	−X

The parameters xdim and ydim define two things:

- The size of the area to be scanned
- The direction of the source scan. The order of the scan is basically row, column until it is modified by the parameter amod. The direction is specified as follows:

Sign xdim	Sign ydim	Direction
+	+	left to right, bottom to top
+	−	left to right, top to bottom
−	+	right to left, top to bottom
−	−	right to left, bottom to top

Example: Set amod = 6
xdim > 0
ydim < 0

The following happens:

1. The source area is scanned left to right, top to bottom as follows:

	first direction					
second direction	1	2	3	4	5	6
	7	8	9	10	11	12
	13	14	15	16	17	18
	19	20	21	22	23	24

2. The destination area receives data in the following order:

24	18	12	6	
23	17	11	5	
22	16	10	4	
21	15	9	3	
20	14	8	2	first direction
19	13	7	1	

second direction

Figure 12-1 illustrates the effects of the amod value and the signs of the parameters xdim and ydim on the orientation of a simple right-angle figure when it is copied. The source and destination area origins are indicated by "o" and the opposite vertex defined by xdim, ydim is indicated by "*". Scan directions are indicated by ">" and ">>" characters. The first direction is shown by ">>", and the second direction by ">".

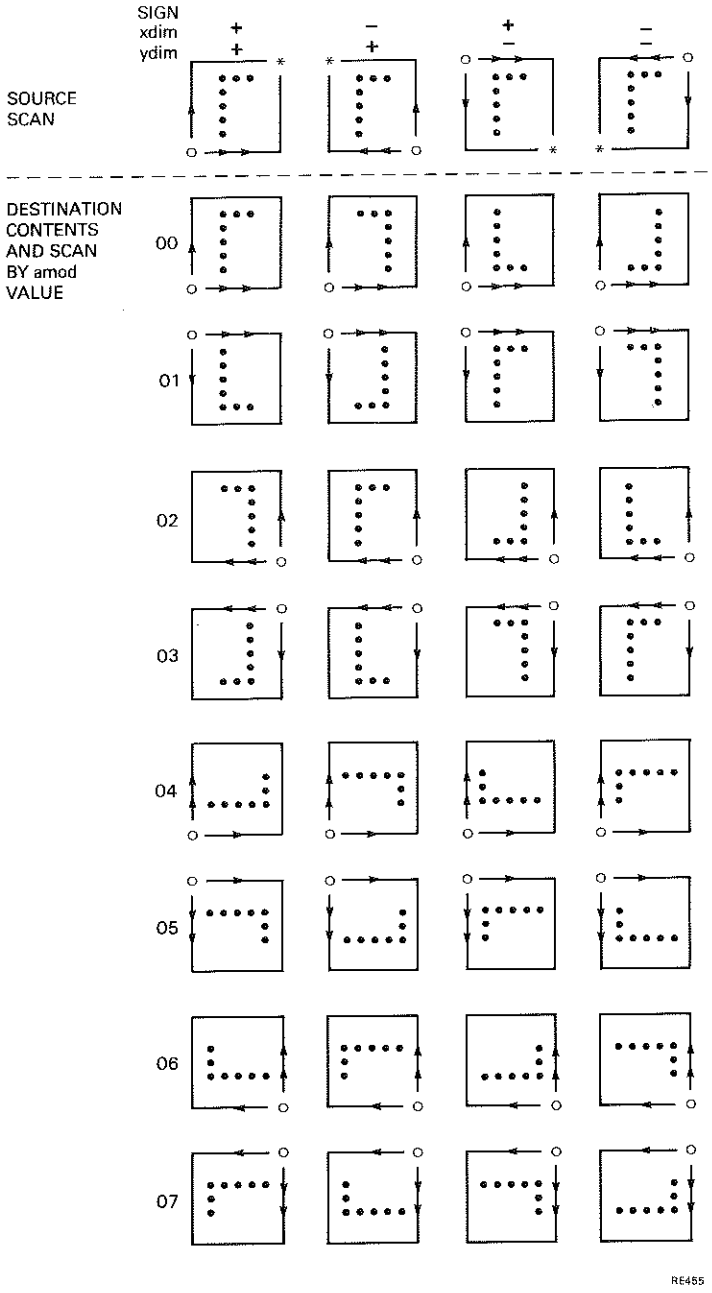


Figure 12-1a Effect of Parameter Values and Signs on Orientation of Copied Picture; amod range 0 – 7

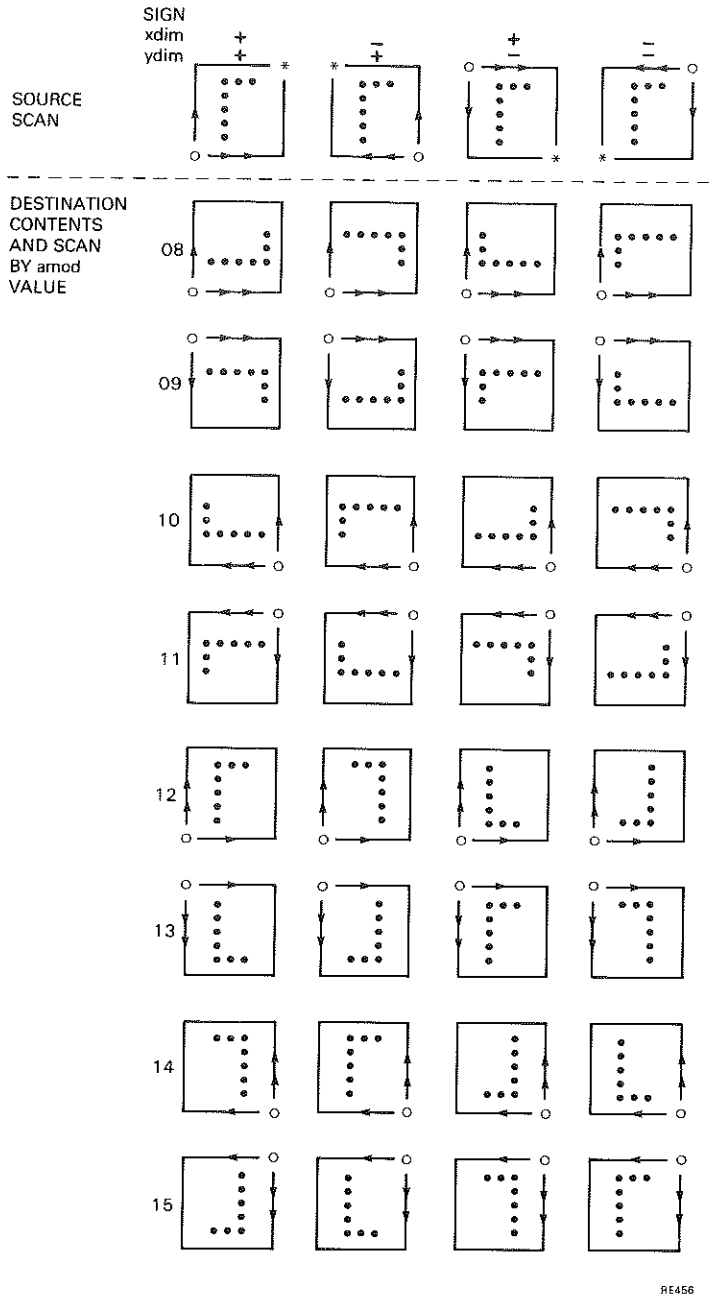


Figure 12-1b Effect of Parameter Values and Signs on Orientation of Copied Picture; amod range 8 – 15

CHAPTER 13
INTERACTIVE OPERATION INSTRUCTIONS

This chapter contains a description of each VIVID interactive operation instruction. Opcodes are given in decimal. A MACRO-32 example of each instruction is provided.

13.1 CURSOR_STYLE

Sets the cursor to the specified pixel data, or to one of the default cursor styles.

Opcode: 90 Length: 1 or 3 + number of rows, or 255

Format: CURSOR_STYLE ccode [, dxp, dyp, row_1, row_2 ... row_n]
where n is in the range 1-16

Parameters: ccode = cursor style code:
−1 : full screen cross-hairs
0 : small cross-hairs
>0 : width of cursor in pixels

dxp = cell pixel X displacement from cursor point
dyp = cell pixel Y displacement from cursor point
row_n = cursor cell row bit pattern

End Position: The current position is not changed.

Errors: An error occurs in the following cases:

- length out of range 1 to 19
- ccode out of range −1 to 16

Notes: For a default cursor, only the parameter ccode should be present and the length must be 1.

Rows are in sequence, the first being the bottom row of the cursor cell.

Details of the format of a row may be found in Section 3.3.2.

To center the cell at the cursor position, use the following parameter values:

dxp = ccod/2
dyp = (length – 3)/2

Setting the cursor style has no effect on cursor visibility. If the cursor is currently visible, it is replaced immediately by the new cursor style.

Where a length of 255 is used, the parameter list must be terminated with END_PARAMETERS (see Section 3.3.1).

For optimum rendition, it may be necessary to adjust the pointing device sensitivity factors by using the VCP (Chapter 18).

An example is given in Appendix E.

Example:

```
.BYTE 1.,90.          ;length and opcode
.WORD 0.              ;small  cross-hairs
```

An example of a user-defined cursor is given in Appendix E.

13.2 POSITION_CURSOR

Sets the cursor to the specified position. The cursor is restricted by the screen boundaries.

Opcode: 91 Length: 0 or 2

Format: POSITION_CURSOR [x, y]

Parameters: x = cursor X position in VAS
y = cursor Y position in VAS

End Position: The current drawing position is not changed.

Errors: None

Notes: If no parameters are provided, the cursor is moved to the current drawing position.

This instruction does not change cursor visibility.

Example:
```
.BYTE 0.,91.        ;length and opcode
                    ;move cursor to current drawing
                    ;position
```

13.3 CURSOR_VISIBILITY

Defines whether or not the cursor is visible.

Opcode: 92 Length: 1

Format: CURSOR_VISIBILITY cmod

Parameters: cmod = cursor visibility
0 : cursor invisible
1 : cursor visible

End Position: The current position is not changed.

Errors: None

Notes: Even when invisible, the cursor is restricted to the screen bounds.

Example:
```
.BYTE 1.,92.        ;length and opcode
.WORD 1.            ;cursor visible
```

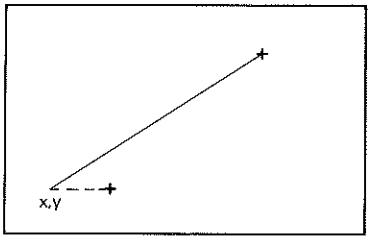


Figure 13-1 Linear Rubber Band

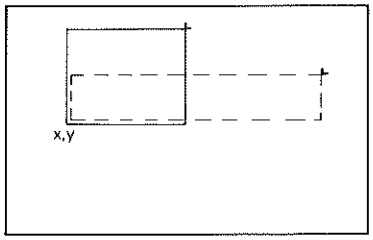


Figure 13-2 Rectangular Rubber Band

13.4 RUBBER_BAND

Defines the rubber band characteristics and base point.

RUBBER_BAND defines two points either as the ends of a line (linear rubber band; Figure 13-1) or as the ends of the diagonal of a rectangle (rectangular rubber band; Figure 13-2).

Opcode: 93 Length: 1 or 3

Format: RUBBER_BAND rcod [,x,y]

Parameters: rcod = rubber band code:
0 : no rubber band
1 : linear rubber band
2 : rectangular rubber band
x = X position of base point in VAS
y = Y position of base point in VAS

End Position: The current position is not changed.

Errors: If rcod is out of range, a warning is issued and no rubber band assumed.

Notes: If no X, Y parameters are provided, the current drawing position is assumed as the base point of the rubber band.

In Figures 13-1 and 13-2, x,y is the base point and two successive cursor positions are shown.

This instruction does not change cursor visibility. If you use the combination of long cross-hair cursor and rectangular rubber band, two sides of the cursor or band will not be seen.

For optimum rendition, it may be necessary to adjust the pointing device sensitivity factors by using the VCP (Chapter 18).

Example:
```
.BYTE 3.,93.        ;length and opcode
.WORD 1.            ;linear rubber band
.WORD 100.          ;X position of base point
.WORD 50.           ;Y position of base point
```

13.5 SWITCH_REPORT_ENABLE

Enables a pointing device so that when a specified switch activity occurs, a report is sent to the host. The condition "No Switch Activity" is also covered, so reports are provided for all cursor movements. Report handling is described in Chapter 14.

Opcode: 94 Length: 1

Format: SWITCH_REPORT_ENABLE mask

Parameters: mask = switch mask

End Position: The current drawing position is not changed. The cursor position is moved according to calculation from pointing device input.

Errors: None

Notes: A logical AND operation is caried out between the switch mask and the pointing-device switch input data. A non-zero result determines that a report shall be sent to the host. A zero switch mask indicates that reports on all pointing device input, including movement only, are directed to the host. For details of reporting, see Section 14.16.

Example:
```
.BYTE 1.,94.        ;length and opcode
.WORD ^B0110        ;mask value
```

13.6 SWITCH_DISABLE

Disables pointing device reporting.

Opcode: 95 Length: 0

Format: SWITCH_DISABLE

Parameters: None

End Position: The current drawing position is not changed.

Errors: None

Example:
```
.BYTE 0.,95.        ;length and opcode
```

13.7 AUTOSWITCH

Defines a wait time after which a switch depression will be assumed at a given cursor position.

Opcode: 96 Length: 2

Format: AUTOSWITCH tout, sval

Parameters: tout = wait time in seconds
sval = the switch value returned if switch is not depressed within the wait time

End Position: Current drawing and cursor positions are not changed.

Errors: None

Notes: Reporting to the host is as for SWITCH__REPORT__ENABLE (Section 13.6).

The switch value need not be supported by the pointing device in use.

Example:
```
.BYTE 2.,96.        ;length and opcode
.WORD 5.            ;wait 5 seconds
.WORD 1.            ;switch value to be returned
```

13.8 WAIT__SWITCH

Waits for a switch interrupt before continuing with the next VIVID instruction.

Opcode: 103 Length: 1

Format: WAIT__SWITCH mask

Parameters: mask = switch mask

End Position: Current drawing and cursor positions are not changed.

Errors: None

Notes: Segment processing waits at this instruction for pointing device input where the switch input is non-zero when masked with the supplied value.

This instruction execution is interrupted by a Stop Execution QIO, or in the case of QIO time-out. Subsequent resumption of processing is at the following instruction.

Example:
```
.BYTE 1.,103        ;length and opcode
.WORD 12.           ;bits 1100 enable switches 2
                    ;and 3
```

13.9 MATCH_ENABLE

When subsequent drawing intersects the cursor position, a report (Section 14.16) is sent to the host. Following this instruction, drawing continues until the maximum number of matches have been detected.

Opcode: 97 Length: 1

Format: MATCH_ENABLE nmax

Parameters: nmax = maximum number of matches

End Position: Drawing and cursor positions are not affected.

Errors: None

Notes: After drawing a picture, the same segments may be processed with match enabled, to determine which instruction caused the pixel at the cursor position to be drawn. Each match corresponding to the cursor position may be be identified until match is disabled.

The parameter nmax determines the maximum number of matches reported before segment processing is terminated. It may be used to provide a single match, after which each further match might be accessed by issuing a Resume Execution QIO, or to limit the number of matches detected for program or report segment size reasons. A value of 32767 implies an unlimited number.

After resuming, if nmax has been decremented to zero, each match detected terminates processing of the segment.

Example:
```
.BYTE 1.,97.        ;length and opcode
.WORD 10.           ;report up to 10 matches
```

13.10 MATCH__DISABLE

Disables a match.

Opcode: 98 Length: 0

Format: MATCH__DISABLE

Parameters: None

End Position: Drawing and cursor positions are not affected.

Errors: None

Notes: See notes for MATCH__ENABLE (Section 13.9).

Example:
```
.BYTE 0.,98.          ;length and opcode
```

13.11 ACCEPT__KEYBOARD__INPUT

Keyboard input to the identified segment begins. Input continues until the specified termination character is received, a specified maximum number of characters has been read, or the buffer is full.

The keyboard input may be automatically echoed to the screen in the current font with the current text attributes applied from the current drawing position.

Opcode: 102 Length: 3 or 6

Format: ACCEPT__KEYBOARD__INPUT segid, chend, chmax
[, cind, cfore, cback]

Parameters:
segid = the segment ID for writing the data
chend = input termination character
chmax = maximum input number of characters
cind = cursor index in current font
cfore = cursor foreground color index
cback = cursor background color index

End Position: The current drawing position is as for DRAW__CHARS (Section 11.9). The cursor position is not changed.

Errors: Error if segment not found.

Warning if the cursor parameters are out of range. In this case, the defaults are as follows:

- cind = 0
- cfore and cback values are taken from the current foreground and background colors

Notes: If no input termination character is required, the parameter cend should be set to zero.

If only three parameters are provided, automatic echo is not performed.

If split screen toggling is enabled, host serial input causes the display to switch to split screen mode, connecting the keyboard to the host port. The keyboard is disabled until <F4> is pressed. This is to avoid ambiguity about the destination of keyboard input during toggling.

Pressing <F4> a second time replaces the full screen and reconnects the keyboard to VIVID. The <F4> key is not accessible to the display application.

In split screen mode, input from the keyboard is directed to the host serial port.

The segment identified for keyboard input is initialized by this instruction and any previous contents are lost. The segment format is given in Section 3.3.4.

The delete key is interpreted as follows:

- Any previous character written is erased
- For echo, the data entry cursor character is repositioned.

No drawing occurs outside the viewport.

The next display instruction is not performed until input has completed.

The instruction execution is interrupted by a Stop Execution QIO, or in the case of QIO time-out. Subsequent resumption of processing is at the next instruction.

Example:

```
.BYTE 6.,102.      ;length and opcode
.WORD ^X1611       ;segment class 22, number 17
.WORD 0.           ;termination character
.WORD 100.         ;maximum number of characters
.WORD 5.           ;cursor index
.WORD 10.          ;cursor foreground color from
                   ;CLUT
.WORD  6.          ;cursor background color from
                   ;CLUT
```

13.12 START_KEYBOARD_INPUT

Keyboard input for AST processing is begun. The input is terminated by a STOP_KEYBOARD_INPUT instruction.

Opcode: 99 Length: 0

Format: START_KEYBOARD_INPUT

Parameters: None

End Position: The current drawing and cursor positions are not changed.

Errors: None. If the application has not set up an RSX AST or MicroVMS mailbox, the input data is lost.

Notes: If split screen toggling is enabled, host serial input causes the display to switch to split screen mode, connecting it to the host port. The keyboard is disabled until <F4> is pressed. This is to avoid ambiguity about the destination of keyboard input during toggling.

Pressing <F4> a second time replaces the full screen and reconnects the keyboard to VIVID. The <F4> key is not accessible to the display application.

Subsequent keyboard input is directed to the RSX AST or MicroVMS mailbox (see Section 14.17). Character codes despatched are exactly as for host serial input.

No drawing occurs.

Example:

```
.BYTE 0.,99.       ;length and opcode
```

13.13 STOP__KEYBOARD__INPUT

Stops the keyboard input for AST processing. Subsequent input is by means of the serial interface, if it is connected.

Opcode: 100 Length: 0

Format: STOP__KEYBOARD__INPUT

Parameters: None

End Position: The current drawing and cursor positions are not changed.

Errors: None

Notes: Keyboard input is subsequently routed by means of the host serial port, if connected.

Example:

```
.BYTE 0.,100          ;length and opcode
```

CHAPTER 14
REPORT HANDLING

This chapter describes the instruction to request reports. The opcode is given in decimal and a MACRO-32 example is provided. The formats of all report packets are also described. The report packets are either written directly to the report segment or provided to an AST or mailbox routine by means of the stack. The packet format is identical in each case, except where specified otherwise.

- Reports may be generated by any of the following:
- Report requests (Sections 14.1 to 14.12)
- Errors during segment processing (Section 14.13 and 14.14)
- Match interrupts (Section 14.15 and 14.19)
- Switch interrupts (Section 14.16)
- Keyboard input (Section 14.17)
- Interruption of segment processing (Section 14.18)

The reports described in Sections 14.1 to 14.12 are generated by report requests. The remaining reports (Sections 14.13 to 14.19) are generated by events.

An introduction to report handling is given in Section 2.13.

14.1 REQUEST_REPORT INSTRUCTION

This instruction places the specified report in the current report segment.

Opcode: 108 Length: 1

Format: REQUEST_REPORT nrep

Parameters: nrep = report number required (range 0-10)

End Position: The current position is not changed.

Errors: Warning if report number is invalid, and no report other than the warning report is generated.

Notes: If there is no current report segment, the instruction is ignored.

The report number is that identified in word 0 of the appropriate report packet.

The following report packets can be specified:

nrep	Report Title
0	DRAWING_POSITION
1	CURSOR_POSITION
2	CELL_PARAMETERS
3	GLOBAL_ATTRIBUTES
4	TRANSFORMATION
5	SCREEN_FORMAT
6	FREE_SPACE
7	VSV21_SEGMENTS
8	HOST_SEGMENTS
9	VIVID_VERSION
10	SEGMENT_TRACE

Example:

```
.BYTE 1.,108.       ;length and opcode
.WORD 8.            ;report number
```

14.2 DRAWING_POSITION REPORT PACKET

The report provides the current graphics drawing position in VAS.

Format: The format of the report packet is shown in Figure 14-1.

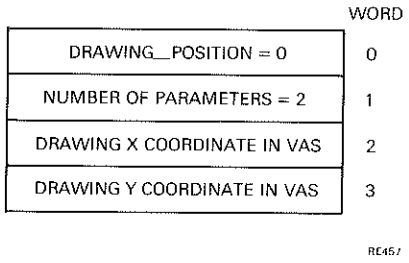


Figure 14-1 Format of Drawing Position Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

14.3 CURSOR_POSITION REPORT PACKET

The report provides the current graphics cursor position in VAS.

Format: The format of the report packet is given in Figure 14-2.

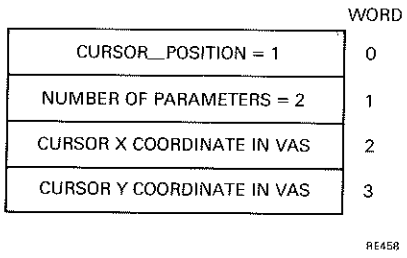


Figure 14-2 Format of Cursor Position Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

14.4 CELL_PARAMETERS REPORT PACKET

The report provides the current attributes applicable to text instructions.

Format: The format of the report packet is given in Figure 14-3.

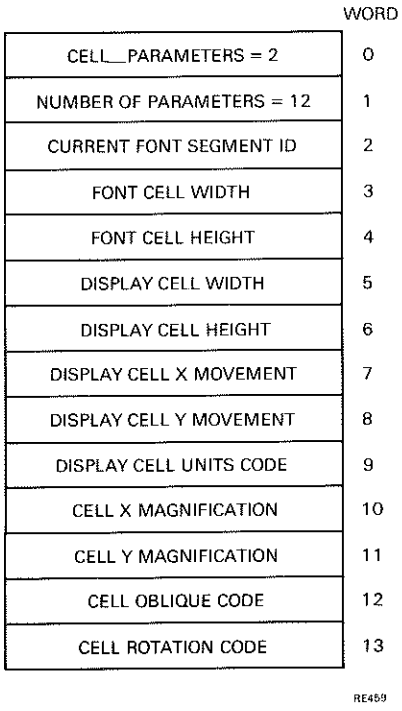


Figure 14-3 Format of Cell Parameters Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only. Details of the parameters are as for the input parameters in the text instructions (Chapter 11).

14.5 GLOBAL_ATTRIBUTES REPORT PACKET

The report provides the major current global attributes which are not otherwise available by using the VCP (VSV21 Control Program). The VCP commands are described in Chapter 18.

Format: The format of the report packet is given in Figure 14-4.

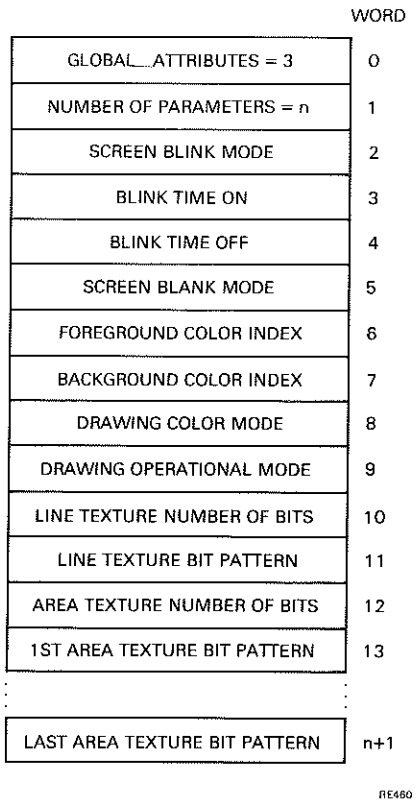


Figure 14-4 Format of Global Attributes Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only. Details of the parameters are as for the input parameters in the global attribute instructions (Chapter 8).

The maximum report packet size is 29 words.

14.6 TRANSFORMATION REPORT PACKET

This report provides the current transformation details.

Format: The format of the report packet is given in Figure 14-5.

TRANSFORMATION REPORT PACKET

	WORD
TRANSFORMATION = 4	0
NUMBER OF PARAMETERS = 17	1
SCREEN X DIMENSION IN VAS	2
SCREEN Y DIMENSION IN VAS	3
WINDOW X ORIGIN IN VAS	4
WINDOW Y ORIGIN IN VAS	5
WINDOW X EXTENT IN VAS	6
WINDOW Y EXTENT IN VAS	7
VIEWPORT MINIMUM X	8
VIEWPORT MINIMUM Y	9
VIEWPORT WIDTH	10
VIEWPORT HEIGHT	11
X ZOOM FACTOR	12
Y ZOOM FACTOR	13
DRAWING TRANSFORMATIONS FLAG	14
X DRAWING MAGNIFICATION	15
Y DRAWING MAGNIFICATION	16
DRAWING X COORDINATE SHIFT	17
DRAWING Y COORDINATE SHIFT	18

RE461

Figure 14-5 Format of Transformation Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only. Details of the parameters are as for the input parameters in the viewing transformation instructions (Chapter 7).

14.7 SCREEN_FORMAT REPORT PACKET

The report provides the screen format.

Format: The format of the report packet is given in Figure 14-6.

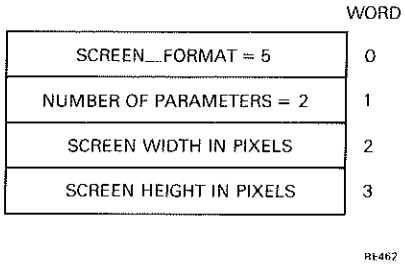


Figure 14-6 Format of Screen Format Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

14.8 FREE_SPACE REPORT PACKET

This report provides the number of words of space free for further download of segments to the VSV21.

Format: The format of the report packet is given in Figure 14-7.

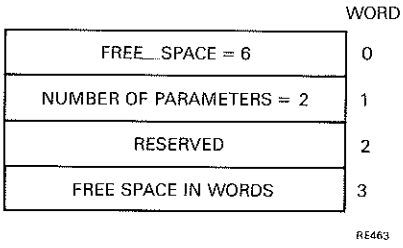


Figure 14-7 Format of Free Space Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

Word 2 is reserved to allow compatibility with any future increase in VSV21 RAM. In this case, the free space would be given by a longword.

Note that the amount of free space (Word 3) is given in words rather than in bytes.

14.9 VSV21_SEGMENTS REPORT PACKET

This report provides a list of the segments downloaded to the VSV21.

Format: The format of the report packet is given in Figure 14-8.

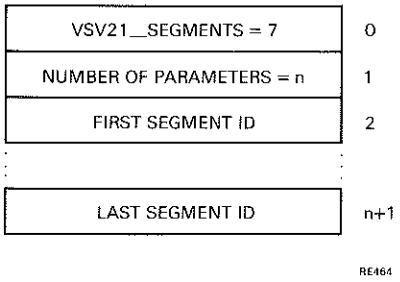


Figure 14-8 Format of VSV21 Segments Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

Segment IDs are in ascending sequence.

The maximum report packet size is 255 words.

14.10 HOST_SEGMENTS REPORT PACKET

This report provides a list of the segments defined in the host memory.

Format: The format of the report packet is given in Figure 14-9.

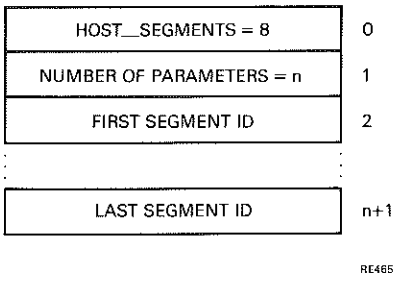


Figure 14-9 Format of Host Segments Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

Segment IDs are in ascending sequence.

The maximum report packet size is 514 words.

14.11 VIVID_VERSION REPORT PACKET

This report provides the downloaded VIVID interpreter version number.

Format: The format of the report packet is given in Figure 14-10.

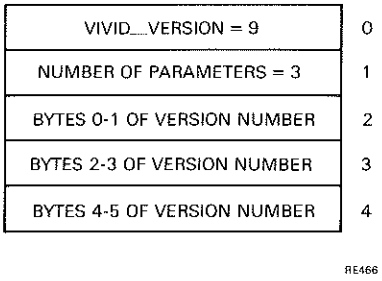


Figure 14-10 Format of VIVID Version Report Packet

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

The first version number byte in each word is stored in bits 7 to 0, and the second in bits 15 to 8.

14.12 SEGMENT_TRACE REPORT PACKET

This report provides a trace of the nested segment calls to the current segment.

Format: The format of the report packet is given in Figure 14-11.

Notes: This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

Segment IDs are in calling sequence from the top level down to the segment in which the report request occurs.

The maximum report packet size is 33 words.

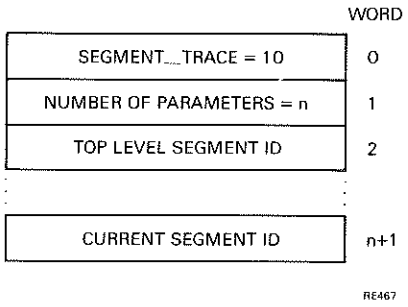


Figure 14-11 Format of Segment Trace Report Packet

14.13 VIVID_WARNING REPORT PACKET

The report indicates that a warning has been encountered and segment processing has continued.

Format: The format of the report packet is given in Figure 14-12.

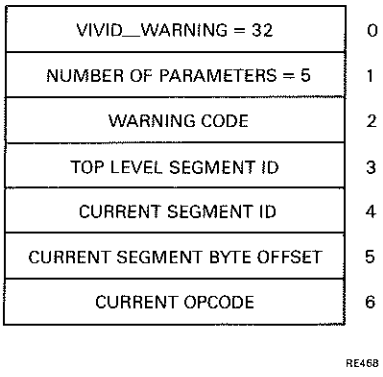


Figure 14-12 Format of VIVID Warning Report Packet

Notes: This report is generated when the VIVID interpreter finds a segment error after which processing can continue.

The report may be directed to the report segment or to an AST or mailbox.

The top-level segment (Word 3) is the segment referenced in the invoking QIO. The current segment (Word 4) is the actual segment currently being processed, and the offset refers to the offset in the segment of the opcode word for which the error was detected.

14.14 VIVID_ERROR REPORT PACKET

The report indicates that an error has been encountered and segment processing has been stopped.

Format: The format of the report packet is given in Figure 14-13.

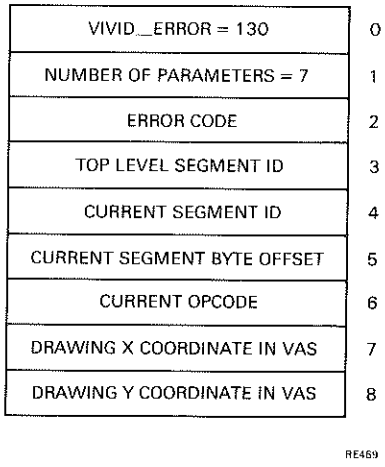


Figure 14-13 Format of VIVID Error Report Packet

Notes: This report is generated when the VIVID interpreter finds an error from which segment processing should not continue.

The report may be directed to the report segment or to an AST.

The top-level segment is the segment referenced in the invoking QIO. The current segment is the actual segment being processed. The offset refers to the offset from the start of that segment of the opcode word for which the error was detected.

14.15 MATCH_INTERRUPT REPORT PACKET

This report indicates that a match has been detected. Processing continues if the match count has not been exhausted.

Format: The format of the report packet is given in Figure 14-14.

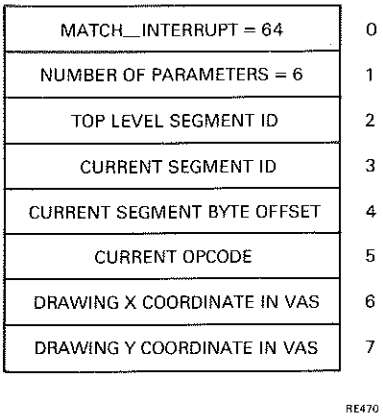


Figure 14-14 Format of Match Interrupt Report Packet

Notes: This report is generated when a match is found while a segment is being processed.

The report may be directed to the report segment or to an AST or mailbox.

The top level segment is the segment referenced in the invoking QIO. The current segment is the actual segment being processed. The offset refers to the offset from the start of that segment of the opcode word for which the error was detected.

14.16 SWITCH_INTERRUPT REPORT PACKET

This report indicates that an operation has been performed on a pointing device for which reporting has been enabled.

Format: The format of the report packet is given in Figure 14-15.

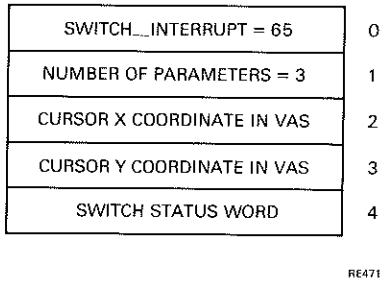


Figure 14-15 Format of Switch Report Report Packet

Notes: The report may be directed to the report segment or to an AST or mailbox.

The switch status word indicates whether switches are depressed (bit = 1), or raised (bit = 0). Bit 0 corresponds to switch 0, bit 1 to switch 1, and so on.

14.17 KEYBOARD_INPUT REPORT PACKET

This report provides input from the keyboard to an AST.

Format: The format of the report packet is given in Figure 14-16.

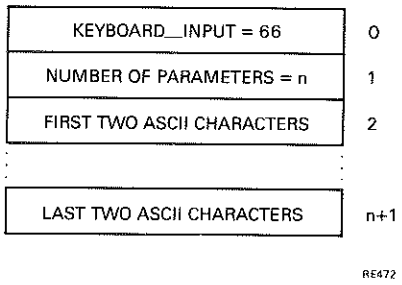


Figure 14-16 Format of Keyboard Input Report Packet

Notes: In each word, the first character is in bits 7 to 0, and the second in bits 15 to 8. If there is only one valid character in the word, the second is set to zero.

The data represents the ASCII character string corresponding to a single key depression. However, if the string is too long for the driver buffers, multiple transfers will occur.

14.18 VIVID_INTERRUPT REPORT PACKET

This report indicates that segment processing has been interrupted by a QIO stop, by time-out or by a cancel protocol from the host.

Format: The format of the report packet is given in Figure 14-17.

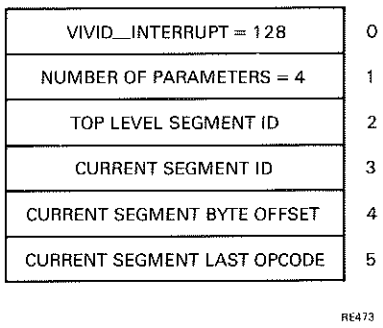


Figure 14-17 Format of VIVID Interrupt Report Packet

Notes: The report may be directed to the report segment or to an AST or mailbox.

The top-level segment is the segment referenced in the invoking QIO. The current segment is the actual segment being processed. The offset refers to the offset from the start of that segment of the opcode word for which the error was detected.

14.19 MAXIMUM_MATCHES REPORT PACKET

The report indicates that segment processing has been stopped as the maximum number of matches has been reported.

Format: The format of the report packet is given in Figure 14-18.

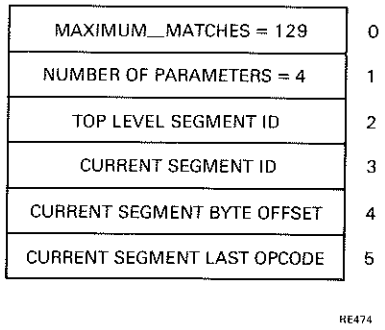


Figure 14-18 Format of Maximum Matches Report Packet

Notes: The report may be directed to the report segment or to an AST or mailbox.

The top-level segment is the segment referenced in the invoking QIO. The current segment is the actual segment being processed. The offset refers to the offset from the start of that segment of the opcode word for which the error was detected.

CHAPTER 15
BUILDING PICTURES USING FORTRAN DRAW

To run VSV11 emulation in the VSV21, download the VSV11 emulator from the host. The method of downloading is described in Section 18.3.

The FORTRAN Draw package is a library of subroutines available to help FORTRAN programmers to create pictures for the VSV21 in VSV11 emulation.

Library subroutines can be called from FORTRAN programs. The library uses FORTRAN-77, so programs using it must be compiled using either the FORTRAN-77 or FORTRAN-IV-PLUS compiler.

The FORTRAN Draw package contains more than forty subroutines which enable programs to:

- Draw common graphic shapes
- Control color attributes
- Write text
- Draw graphs and histograms
- Perform screen and drawing position control functions
- Control the cursor position
- Perform initialization and input/output functions
- Access the display list contents
- Control display list processing
- Handle joystick input
- Issue QIO requests

15.1 USING FORTRAN DRAW

15.1.1 Coordinate System

The coordinates used are shown in Figure 15-1.

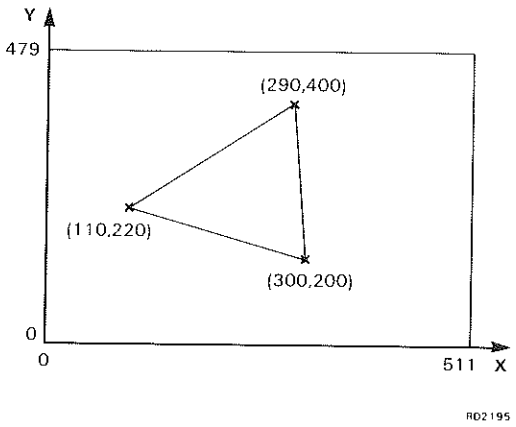


Figure 15-1 FORTRAN Draw Coordinate System

All X- and Y-coordinate positions must be specified as integers. Any scaling and translation operations that are required must be done by the application program.

For further information about the coordinate system used in VSV11 emulation, refer to the description in the *VSV11/VS11 Option Description (YM-C183C-00).*

15.1.2 Common Block Definition

To change data in the COMMON block VSDEFS.FOR, include the following line in the data definition area at the top of the program:

```
INCLUDE  'VSDEFS.FOR'
```

The module VSBLOCK.FTN contains the default values for the COMMON blocks defined in VSDEFS.FOR. VSBLOCK and VSDEFS are contained on the same directory as the FORTRAN Draw package.

15.1.3 Reserved Logical Unit Numbers

The following LUNs (Logical Unit Numbers) are reserved for use by the library subroutines.

- Logical Unit 2 is assigned to the VSV21 device at all times for display output.
- Logical Unit 7 is used by the VSFILE and VSLOAD routines for saving and restoring display buffers.
- Logical Unit 10 is used when loading a new font using VSFONT.

15.2 PROGRAMMING METHOD

The basic method of writing a program to use FORTRAN Draw subroutines is shown in flowchart form in Figure 15-2.

The steps are:

1. Download the kernel, pointing device driver and VSV11 emulation software to the VSV21 module.

   If the VSV11 emulation need not be set under application control, refer to the description of the VCP LOAD command in the *VSV21 User's Guide (AZ-FV70A-TC).* These commands can also be included in the system startup file.

2. Initialize the VSV11 Emulator. Before you can issue any further commands to the VSV21, you must initialize the emulator and the package by calling the VSINIT subroutine.

3. Build the display list. This involves calling the picture-drawing subroutines you need to make up the picture.

4. Display the picture. This is done by calling the VSSYNC subroutine. VSSYNC sends the display list to the VSV11 emulator, where it is processed by the graphics controller chip and displayed on the screen.

15.3 FORTRAN DRAW SUBROUTINES

The following subroutines are listed by the function they perform. For a complete description of each subroutine and its call parameters, refer to the *VSV11-M/M-PLUS Software Driver Guide (AA-J287D-TK).*

Drawing picture shapes:

VSCIRC – Draws a circle

VSCURV – Draws an interpolated curve

VSFILL – Draws a filled rectangle

VSRECT – Draws a filled or unfilled rectangle

VSPOLY – Draws a filled or unfilled polygon

VSDOT – Draws an absolute point

VSDRAW – Draws a line to a point

VSDTHK – Draws a variable width line

VSRDRW – Draws a relative line

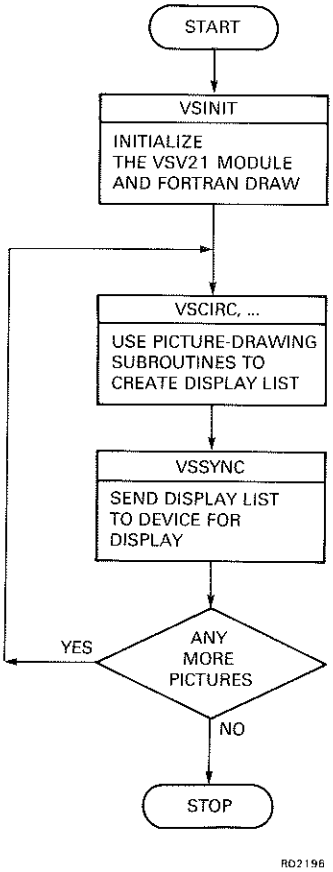


Figure 15-2 Programming Method for FORTRAN Draw

Color control:

VSBACK – Sets the background color

VSCOLR – Sets the drawing color

VSMIX – Mixes a color

Text control:

VSTEXT – Writes a text string

VSFONT – Selects a text font

VSDFNT – Writes text in the current font

VSDSHD – Writes text with drop shadow

VSFLEN – Gets the length of a text string

Graphs and histograms:

VSHINC – Sets the histogram increment

VSETHB – Sets the histogram base

VSHSTX – Adds a point to X histogram

VSHSTY – Adds a point to Y histogram

VSGRFX – Adds a point to X graph

VSGRFY – Adds a point to Y graph

Screen and drawing position control:

VSCLR – Clears screen

VSMOVE – Moves current drawing position

VSRMVE – Moves the current drawing position by a relative amount

Cursor control:

VSCPOS – Gets cursor position

VSPUTC – Sets cursor position

VSCURS – Performs cursor operation

Initialization and configuration:

VSINIT – Initializes device and package

VSMODE – Sets channel characteristics

VSSWAP – Inverts the state of all active channels

Display list functions:

VSLOAD – Loads saved display list

VSFILE – Begins saving display list instructions in a file

VSDPLY – Puts data into display list buffer

VSSYNC – Sends display list to device for display

VSDJMP – Jumps within display list

Joystick control:

VSJOYS – Performs joystick operation

VSWAIT – Waits for switch interrupt

Miscellaneous:

VSDLAY – Pauses

VSQIO – Issues a QIO call

VSSTAT – Gets I/O status block from last QIO

VSGADR – Gets address

CHAPTER 16
BUILDING AND PROCESSING VSV11 DISPLAY LISTS

To run VSV11 emulation in the VSV21, download the VSV11 emulator from the host. The method of downloading is described in Section 18.3.

The VSV21 can process display lists which contain the VS11/VSV11 display list instructions. You can use the VSV11 emulator to run most VS11/VSV11 applications.

A display list is a list of instructions which describes the graphic objects that make up a picture. The instructions tell the graphics controller what shapes to draw on the screen, and how they should appear. Display lists are created by applications programs and output to the VSV21 for display. The display list can be stored in a host file, but it must reside in the VSV21 memory before it can be processed.

The display list consists of words of binary information which describe the primitives, attributes, and control instructions that make up the picture. Each instruction occupies one 16-bit word in the memory. It contains an operation code which identifies the instruction, and parameters which give further information to the graphics controller hardware. For example, this may include the coordinates of the point where the object is to be drawn. To build a display list, you list the instructions which describe the shapes you want to include in the picture, in the order you want them to be drawn on the screen. The steps in generating and processing a VSV11 display list to run on the VSV21 are described in Section 16.2.

16.1 VSV11 DISPLAY LIST CONTENTS

In VSV11 emulation, the VSV21 processes display lists which contain the VSV11 display list instructions. The VSV21 can emulate a single-channel minimum-configuration VSV11 device. Therefore, it cannot support the following VSV11 features:

- Multiple channels
- Eight-bit pixel data
- Hardware register programming

The instructions which can be used in VSV11 display lists are described in this chapter.

16.1.1 Graphic Mode Instructions

In VSV11 emulation, the VSV21 operates in one of the following graphic modes:

- CHARACTER
- SHORT VECTOR
- LONG VECTOR
- ABSOLUTE POINT
- GRAPH/HISTOGRAM X
- GRAPH/HISTOGRAM Y
- RELATIVE POINT
- RUN-LENGTH

The graphic mode determines how the graphic data instructions described in the next section are to be interpreted by the VSV21 hardware.

16.1.2 Graphic Data Instructions

These instructions define coordinates and graphic objects to be drawn on the screen. The interpretation of a graphics data instruction depends on the current graphic mode, previously set by one of the instructions described in Section 16.1.1.

Each graphic mode instruction has one or more corresponding graphic data instructions. For example, in long vector mode, there must be at least two long vector data instructions to specify the endpoints of the vector to be drawn.

16.1.3 Control Instructions

These instructions provide facilities for setting up the pixel memory and joystick channels, branching within the display list, clearing of the pixel memory, and the null operation (NOP).

Control instructions may be inserted anywhere within the display list, except between linked graphic data instructions. For example, control instructions may not be inserted between two long vector data instructions which are associated with the same vector. Control instructions do not affect the current graphic mode.

The following control instructions are available:

- JOYSTICK STATUS – used to enable and disable the cross-hair cursor and joystick interrupts.
- LOAD EXTENDED JOYSTICK CONTROL – can be used to simulate the joystick switch being pressed within the software.
- WRITE CURSOR COORDINATES – enables the program to set up initial cursor coordinates or simulate the action of the joystick.
- SET HISTOGRAM BASE – used to specify the base position for a histogram or bar chart.
- SET CHARACTER BASE – used to specify a table of characters to be processed in character mode.
- DISPLAY JUMP – used to transfer control to another part of the display list. The address can be specified relative to the start of the display list or relative to the start of the task.
- DISPLAY JUMP-TO-SUBROUTINE – used to call a subroutine within a display list.
- DISPLAY POP – used to return from a display list subroutine.
- DISPLAY NOP – null operation.
- LOAD STATUS REGISTER A – used to stop the processor, enable/disable the STOP interrupt, clear or set the pixel memory.
- LOAD STATUS REGISTER C – used to control the channel select, memory read/write select, memory switch enable, and pixel mode select.
- LOAD GRAPHPLOT INCREMENT – sets up the increment between data points plotted in graph/histogram mode.
- LOAD PIXEL-DATA INHIBIT – can be used to erase selectively complex pictures drawn by display lists containing several changes of pixel data.
- MARKER NO-OP – marks locations within the display list.

16.1.4 Special Graphic Instructions

These instructions are used to perform bit-map operations, that is, to transfer data, pixel by pixel, between the host memory and the on-board pixel memory. The special graphic instructions do not affect the current graphic mode.

The special graphic instructions available are:

- BIT-MAP-0 – moves a square array of pixel data from the host memory to the on-board pixel memory.
- BIT-MAP-1 – moves a string of pixels from the host memory to sequential horizontal locations in the pixel memory.
- DMA PIXEL READBACK – reads an area of pixel memory into the host memory using DMA.

The basic format of a display list instruction word is shown in Figure 16-1. All graphic data instructions have bit 15 clear. All other instructions have bit 15 set. Graphic data instructions are interpreted within the context of the current graphic mode. Bits 14 to 10 contain the opcode of the instruction. For example, the opcode for the instruction to set LONG VECTOR mode is 00100. The remainder of the word (bits 0 to 9) contain additional information which is specific to the opcode chosen.

For a full description of the VSV11 display list instructions, refer to the *VSV11/VS11 Option Description (YM-C183C-00)*.

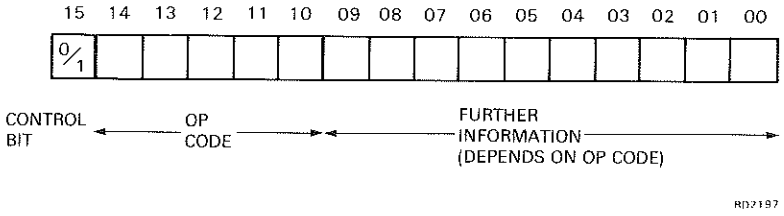


Figure 16-1 VSV11 Display List Instruction Format

16.2 GENERATING AND PROCESSING VSV11 DISPLAY LISTS

The steps in generating a display list and processing it on the VSV21 are as follows:

1. On RSX systems, attach the user task to the device. This dedicates a VSV21 so that it will accept only commands from your task. If the task includes an interrupt service routine for a joystick, attach your task to the device. Otherwise, the Attach is optional.

2. Create the display list. You can use a display list which has been previously created and stored in a file, but you must read the display list into the memory before it can be processed.

3. Tell the VSV21 to process the display list. This is done by issuing a QIO call to the VSV21 driver. One of the parameters you specify is the address of the display list in your program. This is used by the driver to initiate a transfer of the display list to the VSV21 memory using DMA (Direct Memory Access). The graphics controller processes the display list to generate the picture in the pixel memory, and to output the picture to the screen.

4. You can interrupt the display list while it is being processed by using the QIO functions described later in Chapter 17. For example, there are QIO functions which enable you to stop the display or to continue at the point it was stopped. There are also QIO functions which request input from the joystick.

5. When the processing is complete, check the return status. When the QIO transfer finishes, the device driver sends a status code back to the application, to say whether it was successful. Check the status code in case an exception has occurred.

6. Continue in the same way until all display lists have been displayed.

7. On RSX systems, detach your task from the VSV21. If your task is attached to the VSV21 at the end of the program, detach it to free the device.

CHAPTER 17
VSV11 I/O FUNCTIONS

To run VSV11 emulation in the VSV21, download the VSV11 emulator from the host. The method of downloading is described in Section 18.3.

This chapter describes how to write programs which issue QIO calls to the VSV21 device driver in VSV11 emulation. QIO calls can be used to perform a number of different functions associated with processing a display list.

A general introduction to the QIO call mechanism and its interface with the VSV21 device driver is given in Appendix A. Details of the QIO directive are given in the *RSX-11M/M-PLUS Reference Manual (AA-L675A-TC)* and the *VAX/VMS System Services Reference Manual (AA-D018C-TE)*.

The basic steps in generating a VSV11 display list and processing it on the VSV21 are given in Section 16.2.

17.1 QIO FUNCTIONS FOR RSX-11M-PLUS AND MICRO/RSX

The functions and their octal values are listed in Table 17-1.

Table 17-1 VSV11 Emulation QIO Functions for RSX-11M-PLUS and Micro/RSX

Description	Function Code	Octal Code
Attach the VSV21	IO.ATT	1400
Detach the VSV21	IO.DET	2000
Connect and display	IO.CON	3000
Connect to auxiliary memory	IO.AUX	2400
Continue display	IO.CNT	4000
Stop display	IO.STP	3400
Cancel I/O requests	IO.KIL	0012
Read joystick location	IO.RJS	5000
Read data	IO.RED	6400
Write data	IO.WRT	7000

The device-specific functions of the QIO directive that are valid for the VSV21 in VSV11 emulation are described in the following sections.

17.1.1 Attach the VSV21 Device – IO.ATT

Attaches the user task to the VSV21. The IO.ATT function can also connect the task to an auxiliary display list.

You can set the following conditions when you issue the IO.ATT QIO call:

- Make the current X and Y drawing position available to the program when the I/O request completes (Section 17.1.4).
- Make additional data available to the program when an AST (Asynchronous System Trap) is queued as a result of a joystick interrupt.
- Specify that the accessing of instructions within the display list should be relative to the start of the display list. This is used by the DISPLAY JUMP and DISPLAY JUMP-TO-SUBROUTINE display list instructions. The normal condition is task-relative addressing, that is, instructions are accessed relative to the start of the program. This is explained further in Section 17.1.3.

To set all of these conditions, set the graphics bit by defining TF.GRA = 2 in the program.

QIO Format:

```
    QIOW$   IO.ATT,lun,efn,,iosb
or
    QIOW$   IO.ATT!TF.GRA,lun,efn,,iosb
```

where:

lun = logical unit number of the VSV21 device

efn = event flag number (may be omitted)

iosb = address of input/output status block used for reply status

17.1.2 Detach the VSV21 Device – IO.DET

Detaches the VSV21 device. The IO.DET function detaches the user task from a device which was attached using IO.ATT.

QIO Format:

```
    QIOW$   IO.DET,lun,efn,,iosb
```

where lun, efn, and iosb are described in Section 17.1.1

17.1.3 Connect and Display – IO.CON

Processes a specified display list. The display list is transferred to the on-board memory by DMA (Direct Memory Access), and is used by the graphics controller to generate pixel data in the pixel memory. The pixel data is then output to the video screen to display the picture.

QIO Format:

```
    QIOW$   IO.CON,lun,efn,,iosb,,<dsaddr,dlen,addr,chmode, [tout][,asaddr]>
```

where:

lun, efn, and iosb are described in Section 17.1.1

dsaddr = address of the display list

dlen = size of the display list, in bytes

addr = task-relative or display list-relative address

If addressing is relative to start of program (task-relative), addr = dsaddr.

If addressing is relative to start of display list, addr = 0. To specify addressing relative to the start of the display list, start the display list on a 32-word block boundary, and set addr to zero. You must also set the TF.GRA bit in the IO.ATT function (see Section 17.1.1).

chmode = an octal value which is used to set up the VSV21. The following bits may be set:

Bits 8 and 9 – Channel (must be zero)

Bits 5 and 4 – Define the channel access mode, as follows:

0 = Protected

1 = Read-only

2 = Write-only

3 = Read/write

Bit 3 – Enables switching of the access mode

Bits 2 and 1 – Define the pixel drawing mode, as follows:

2 = Replace mode

3 = Logical OR mode

A further explanation of the contents of this word is given in the LOAD STATUS REGISTER C display list instruction. A description of this is given in the *VSV11/VS11 Option Description (YM-C183C-00),* Chapter 3.

tout = time-out value for the display, in seconds (optional). The QIO completion is indicated by a STOP interrupt generated at the end of the display list processing. This interrupt is described in the *VSV11/VS11 Option Description (YM-C183C-00)*, Chapter 3. If the time out expires and the QIO is not complete, an error code is returned in the I/O status block.

You can use the IO.KIL function to get out of the situation in which no I/O completion interrupt is received.

asaddr = address of an optional AST service routine to handle cursor match and joystick switch interrupts.

If the graphics bit (TF.GRA) was set in the IO.ATT QIO call, you can read the X and Y coordinates of the current VSV21 drawing position when the IO.CON completes. To retrieve the coordinates, issue a GLUN$ directive, as shown in Appendix A. For further details, refer to the description of GLUN$ in the *RSX-11M/M-PLUS Executive Reference Manual (AA-L675A-TC)* or the *RSX-11M/M-PLUS and MicroRSX Executive Reference Manual (AA-Z508A-TC).*

GLUN$ sets the contents of the six-word buffer as shown in Figure 17-1

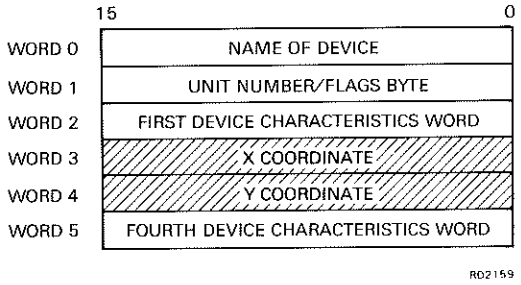


Figure 17-1 Contents of GLUN$ Buffer

17.1.4 Connect to Auxiliary Memory – IO.AUX

Connects the device to the auxiliary memory.

QIO Format:

```
QIOW$   IO.AUX,lun,efn,,iosb,,<axaddr,dlen,addr>
```

where:

lun, efn and iosb are as described in Section 17.1.1

axaddr = address of auxiliary memory segment

dlen = size of display area

addr = 0 if the auxiliary segment is external to the task. Set to the axaddr value on systems without memory management directives

17.1.5 Stop the Display – IO.STP

Stops the display. This can be used to get the display list out of an endless loop.

QIO Format:

```
QIOW$   IO.STP,lun,efn,,iosb
```

where the parameters are as described in Section 17.1.1.

17.1.6 Continue the Display – IO.CNT

Continues the display after it has been interrupted by a joystick switch or cursor match. Processing continues from the point where it was interrupted.

QIO Format:

```
QIOW$   IO.CNT, lun,efn,,iosb
```

where lun, efn, and iosb are as described in Section 17.1.1

17.1.7 Cancel I/O Requests – IO.KIL

Cancels all outstanding I/O requests to the VSV21 device. For I/O requests which are waiting for service or are being processed by the driver, a status code of IE.ABO is returned in the I/O status block.

QIO Format:

```
QIOW$   IO.KIL,lun,efn,,iosb
```

where lun, efn, and iosb are as described in Section 17.1.1

17.1.8 Read Joystick – IO.RJS

This function returns the coordinates of the current position of the joystick.

QIO Format:

```
QIOW$   IO.RJS,lun,efn,,iosb,,<staddr,len,jsnum>
```

where:

lun, efn, and iosb are described in Section 17.1.1

staddr = start address of the data area in which the joystick coordinates will be returned

len = size of the data area, in bytes (minimum four bytes)

jsnum = joystick number (0, 1, 2, or 3) – the default is zero

Results:

On completion, the buffer at staddr will be set as shown in Figure 17-2.

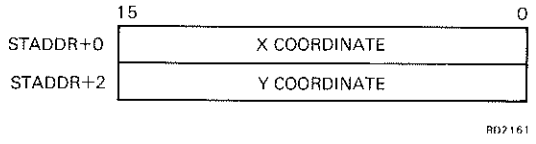


Figure 17-2 Joystick Data Returned by IO.RJS

17.1.9 Read Data – IO.RED

This function reads data from the transparent driver.

QIO Format:

```
QIOW$   IO.RED,lun,efn,,iosb,,<baddr,len,tabid>
```

where:

lun, efn, and iosb are described in Section 17.1.1

baddr = start address of the data area in which the joystick coordinates will be returned

len = size of the data area, in bytes (minimum four bytes)

tabid = table ID. Set to zero for the transparent driver.

17.1.10 Write Data – IO.WRT

This function writes data to the transparent driver.

QIO Format:

```
QIOW$   IO.WRT,lun,efn,,iosb,,<baddr,len,tabid>
```

where:

lun, efn, and iosb are described in Section 17.1.1

baddr = start address of the data area in which the joystick coordinates will be returned

len = size of the data area, in bytes (minimum four bytes)

tabid = table ID. Set to zero for the transparent driver

17.2 QIO FUNCTIONS FOR MICROVMS

The functions and their hexadecimal values are listed in Table 17-2.

Table 17-2 VSV11 Emulation QIO Functions for MicroVMS

Description	Function Code	Hexadecimal Value
Start Display	IO$__START	30
Timeout	IO$__TOUT	37
Stop Display	IO$__STOP	33
Resume Execution	IO$__RESUME	36
Read Status	IO$__READSTATUS	31
Wait	IO$__WAITSWITCH	32
Read Data	IO$__READ	38
Write Data	IO$__WRITE	39

17.2.1 IO$__START – Start Display

This function starts the display.

QIO format:

```
SYS$QIOW  ([efn] ,chan ,IO$__START ,iosb ,,,
          <baddr,blen,staddr,[axaddr],[alen],[dchan]>)
```

where:

efn = event flag number. Defaults to 0 if omitted

chan = number of the I/O channel assigned to the VSV21 device

iosb = address of input/output status block used for reply status

baddr = starting address of the status buffer

blen = length of the display file buffer. Minimum length is eight bytes

staddr = address of display file at which processing is to start. If this is omitted, processing starts at the beginning of the buffer

axaddr = address of the auxiliary buffer

alen = length in bytes of the auxiliary buffer

dchan = channel number to use for the display file

17.2.2 Set Timeout Period – IO$__TOUT

Sets the number of seconds to wait for an I/O complete on IO$__START. This value is in effect until it is changed by another IO$__TOUT QIO or by reloading the driver during SYSGEN. Initially the time out is 15 seconds.

QIO Format:

```
SYS$QIOW ([efn] ,chan ,IO$__TOUT ,iosb ,,,<tout>)
```

where:

efn, chan and iosb are as described in Section 17.2.1

tout = number of seconds to wait; tout > 1

17.2.3 Stop Display – IO$__STOP

This function can be used to stop a looping display file.

QIO format:

```
SYS$QIOW ([efn] ,chan ,IO$__STOP ,iosb)
```

where efn, chan and iosb are as described in Section 17.2.1

17.2.4 Resume Execution – IO$__RESUME

Resumes execution of a display file after a display stop. The arguments must be the same as those in the IO$__START function to be stopped.

QIO format:

```
SYS$QIOW ([efn] ,chan ,IO$__RESUME ,iosb ,,,
          <baddr,blen,[staddr],[aadr],[alen],[dchan]>)
```

where the arguments are as for IO$__START (Section 17.2.1).

17.2.5 Read Status – IO$__READSTATUS

Returns the four device registers containing the following:

- DPC address
- Most recent graphics mode
- Current X and Y positions

QIO format:

SYS$QIOW ([efn] ,chan ,IO$__START iosb,,,<baddr,blen>)

where:

efn, chan and iosb are as described in Section 17.2.1

baddr = address of status buffer

blen = length of status buffer. Minimum length is 8 bytes

17.2.6 Wait for Switch Interrupt – IO$__WAITSWITCH

Wait a specified number of seconds for a switch interrupt. This function is complete when a switch interrupt occurs. If a bus time out occurs, the function completes with an error status. If any other interrupt occurs, the wait continues.

QIO format:

SYS$QIOW ([efn] ,chan ,IO$__WAITSWITCH ,iosb ,,,
<tout,baddr,blen>)

where:

efn, chan and iosb are as described in Section 17.2.1

tout = number of seconds to wait

baddr = address of status buffer

blen = length of status buffer. Minimum length is 8 bytes.

17.2.7 Read Data – IO$__READDATA

This function reads data from the transparent driver.

QIO Format:

```
SYS$QIOW ([efn] ,chan ,IO$__READDATA ,iosb ,,,
          <baddr,blen,tabid>)
```

where:

efn, chan and iosb are described in Section 17.2.1

baddr = start address of the data area in which the joystick coordinates will be returned

blen = size of the data area, in bytes (minimum four bytes)

tabid = table ID, set to zero for transparent driver

17.2.8 Write Data – IO$__WRITEDATA

This function writes data to the transparent driver.

QIO Format:

```
SYS$QIOW ([efn] ,chan ,IO$__WRITEDATA ,iosb ,,,
          <baddr,blen,tabid>)
```

where:

efn, chan and iosb are as described in Section 17.2.1

baddr = start address of the data area in which the joystick coordinates will be returned

blen = size of the data area, in bytes (minimum four bytes)

tabid = table ID, set to zero for transparent driver

17.3 QIO STATUS RETURNS

In RSX-11M-PLUS and Micro/RSX systems, the I/O status block has the format shown in Figure 17-3.

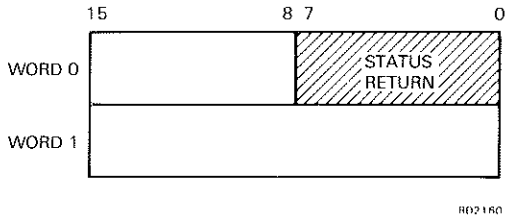


Figure 17-3 Format of I/O Status Block under RSX-11M-PLUS and Micro/RSX

On completion of a QIO transfer, byte 0 of the I/O status block contains a completion code. Successful completion is indicated by the value 1 (IS.SUC) in the status byte. Unsuccessful completion is indicated by a negative value in the status byte. The error codes are listed in Appendix B of the *RSX-11M/M-PLUS I/O Drivers Reference Manual (AA-L677A-TC)*.

In MicroVMS systems, the I/O status block has the format shown in Figure 17-4.

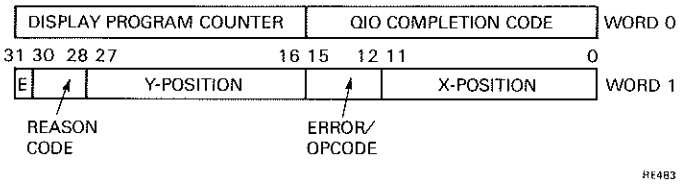


Figure 17-4 Format of I/O Status Block under MicroVMS

The contents of the IOSB are as follows:

Longword 0

bits 0-15 completion code
bits 16-32 display program counter

Longword 1

bits 0-11	current X position
bits 12-15	error code (if bit 31 is set) or last graphics mode opcode (if bit 31 is not set)
bits 16-27	current Y position
bits 28-30	code giving reason for completion. This is one of the following:

1 = VS$CR_STOP	normal
2 = VS$CR_SWITCH	switch
3 = VS$CR_MATCH	match
4 = VS$CR_NXM	VSV11 hardware error
5 = VS$CR_TIMEOUT	time out
6 = VS$CR_FORCE	stop acknowledgement
0 = VS$CR_UNDEFINED	undefined

bit 31	error flag

CHAPTER 18
THE VSV21 CONTROL PROGRAM (VCP)

The VSV21 is configured and controlled by commands from the VSV21 Control Program (VCP). VCP commands perform the following functions:

- Load the kernel by downloading the file KERNEL to the VSV21 module.
- Select either VIVID or VSV11 emulation by downloading either the VIVID interpreter or the VSV11 emulator to the VSV21 module.
- Set the device configuration parameters. For example, describe the peripheral devices currently attached to the serial ports.
- Show the current settings of the device configuration parameters.
- Show the current status of the device.
- Dump VSV21 memory to a file.

The VCP commands can be entered in any of three ways:

- A single command, in response to a system prompt
- A single command, in response to a VCP prompt
- A number of commands in a command procedure, such as a system startup file, to set the VSV21 device to the required state

The available VCP command lines are listed in this chapter, beginning at Section 18.2.

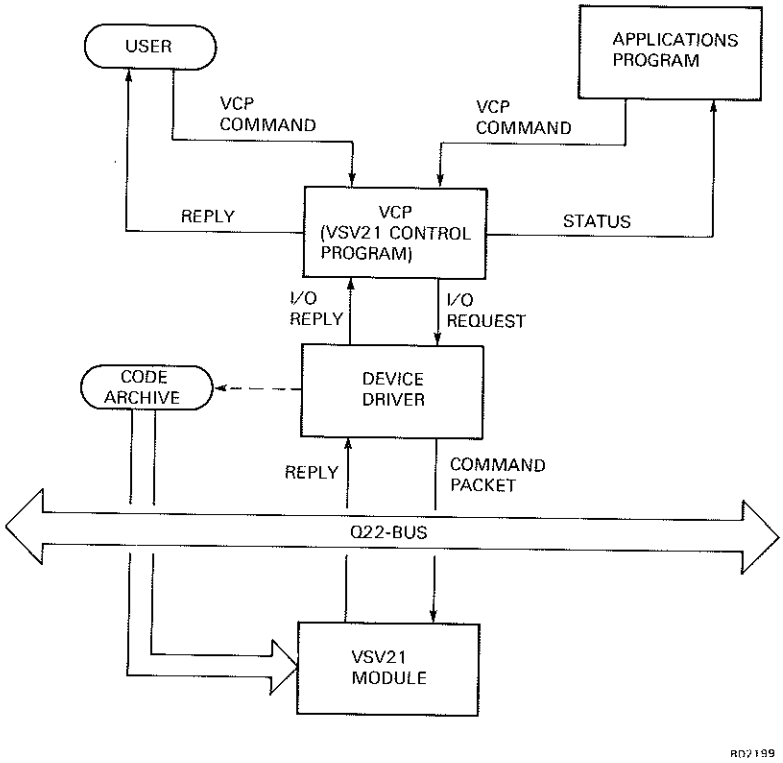


Figure 18-1 User and Program Interfaces to VCP

18.1 CALLING VCP

The user and program interfaces to VCP are shown in Figure 18-1.

18.1.1 Installing VCP on RSX-11M-PLUS and Micro/RSX Systems

Under RSX-11M-PLUS and Micro/RSX, VCP must be either run or installed as an executable task on the host system before you can issue commands to it.

To run VCP, enter

```
RUN $VCP
```

The prompt `"VCP)"` appears. You can now enter individual commands or execute VCP command files. Press CTRL/Z to exit from VCP.

To install VCP, enter

```
INS $VCP/TASK= ...VCP
```

The system installs VCP and returns the system prompt. You can now enter single VCP commands if they are preceded by "VCP". For example, to load the kernel, enter

```
VCP LOAD KERNEL
```

18.1.2 Installing VCP on MicroVMS Systems

Under MicroVMS, the file VCP.EXE must be in the directory SYS$SYSTEM, and all the other files must be in SYS$LIBRARY. The logical name VAXVCP must point to the SYS$LIBRARY directory during software installation.

To run VCP, enter

```
MCR VCP
```

The prompt `"VCP>"` appears. You can now enter individual commands or execute VCP command files. Press CTRL/Z to exit from VCP.

You can enter a single VCP command in response to the dollar prompt without running or loading VCP if it is preceded by " `MCR VCP` ". For example, to load the kernel, enter

```
MCR VCP LOAD KERNEL
```

18.1.3 Calling VCP from a User Program

An application program can use VCP and remain self-contained by spawning the VCP task and passing it a command. Consider the following example:

A system has two VSV21 applications which are run frequently. The first application was originally developed on a VSV11 system with a joystick, so it is run in VSV11 emulation. The second application was developed for VIVID, with a trackball as its pointing device.

The user first downloads the VSV11 emulator or VIVID interpreter. Before each application is run, the characteristics of the pointing device port must be defined, to make sure that the device is in a known state. This could be accomplished in either of two ways:

- Put the VCP commands in a command file which the user invokes before running the applications.
- Spawn VCP in the application program. This allows each application to select its own device and configuration.

This method provides users with an automatic configuration and device control. It is recommended for any system which has a number of different applications requiring different operating characteristics.

The available commands are described in Section 18.2. The VCP task is invoked by issuing the SPWN$ directive from MACRO-11 programs, or by calling the SPAWN subroutine from FORTRAN programs. Some examples of spawn calls under RSX-11M-PLUS and Micro/RSX are as follows:

MACRO-11 Call:

SPWN$ tname,,,,,[efn],,[esb],cmdlin,cmdlen

tname = name (in Radix-50 format) of the task to be spawned (...VCP in this case)

efn = the event flag to be cleared when the directive is issued, and set when VCP exits (optional)

esb = address of an 8-word status block to be written when VCP exits

Word 0 – VCP completion status

Word 1 – TKTN abort code

Words 2 to 7 – Reserved

cmdlin = address of the command line to be sent to VCP

cmdlen = length of the command line in characters (maximum length is 255)

FORTRAN Call:

CALL SPAWN (rtname,,,[iefn],,[iesb],,icmlin,icmlen,,,ids)

rtname = name (in Radix-50 format) of the task to be spawned (...VCP in this case)

iefn = the event flag to be cleared when the directive is issued, and set when VCP exits (optional)

iesb = address of an 8-word status block to be written when VCP exits

Word 0 – VCP completion status

Word 1 – TKTN abort code

Words 2 to 7 – Reserved

icmlin = name of the command line to be sent to VCP

icmlen = length of the command line in characters (maximum length is 255)

FORTRAN Example:

The VSV11 emulation software can be downloaded by a LOAD command (Section 18.3). The following section of FORTRAN code also downloads the VSV11 emulation software.

```
C  FORTRAN VCP Example
C
        INTEGER*4       RTNAME
        DATA            RTNAME /6R...VCP/
C  Name of VCP task in RAD50 (assumes VCP is installed)
C
        INTEGER*2       IESB(8)
C  Status block returned from VCP
C
        CHARACTER*79    ICMLIN
        INTEGER*2       ICMLEN
C  Command line and length
        DATA            ICMLIN /'LOAD VS0: VS11EM'/
        DATA            ICMLEN /16/
C  Store command line and length
C
        INTEGER*2       IDS
C  Directive status word
                .
                .
                .
```

```
      CALL SPAWN (RTNAME,,,1,,IESB,,ICMLIN,ICMLEN,,,IDS)
C  Spawn VCP task and send the command (using event flag 1)
C
      CALL WAITFR(1)
C  Wait for VCP to exit.  Note: IESB(1) now contains the VCP
C  completion status.
C
      IF (IDS.LT.0) GO TO 9999
C  Check the directive status to make sure it was successful
            .
            .
            .

C  End of program
```

A full description of the SPWN$ or SPAWN call is provided in the *RSX-11M/M-PLUS Executive Reference Manual (AA-L675A-TC)* and the *RSX-11M/M-PLUS and MicroIRSX Executive Reference Manual (AA-Z508A-TC).*

18.2 THE VCP COMMANDS

This section describes the commands which can be issued to VCP, and the basic format of the command line. For a description of the individual commands, refer to Sections 18.3 to 18.8.

On-line HELP is available. To see the format of any command, enter HELP and the command. The HELP facility prompts for parameter names and supplies the range of values for each parameter.

18.2.1 Types of VCP Commands

Commands are provided for the following operations:

- Download code from the host

  LOAD [dev] filnam

- Download segments from the host

  VIVID_LOAD_SEG filnam

- Set a VSV21 parameter

  SET_PARAMETER [dev] ctab[/perm] dsp gval[/perm]

- Display configuration and status

  TABLE [dev] ctab
  STATUS dev

- Dump VSV21 memory to a host file

  DUMP filnam

  where:

  filnam = file name

  dev = device specification

  ctab = the table of options to be used

  dsp = a display parameter

  gval = a value associated with dsp

  perm = a qualifier which denotes whether the parameter is to be set permanently, or only for this session

Parameters in a command line must be separated by one or more spaces, commas, or tab characters. The format of these parameters is described in Sections 18.2.2 and 18.4.

18.2.2 Command Syntax

VCP commands are of the form:

command_name [dev] parameters

where:

command_name is one of those listed in Section 18.2.1. You can shorten a command or parameter to the smallest number of characters necessary to identify it uniquely. For example, SET_PARAMETER can be shortened to SET; LOAD may be entered as L.

dev is the device specification, which can be one of the following on RSX-11M-PLUS and Micro/RSX systems:

VS0:
VS1:
VS2:
VS3:

On MicroVMS systems, dev is one of the following:

VSA0:	VSE0:
VSB0:	VSF0:
VSC0:	VSG0:
VSD0:	VSH0:

If the system has just been loaded, VS0: or VSA0: is the default specification. Otherwise, the last device specification entered is the default.

You must enter additional parameters for most commands. There are three categories of parameter as follows.

ctab = a string of up to 16 characters

dsp = a string of up to 16 characters

gval = a parameter containing data of the following types:

- Character string
- Decimal integer
- Hexadecimal string

The format of these parameters is described in Section 18.4.

18.3 DOWNLOADING CODE OR DISPLAY LISTS FROM A HOST FILE

18.3.1 Downloading Code

About 61K words of VSV21 memory space are available for storing the downloaded VSV21 system software and the downloaded segments and saved attributes. The VSV21 memory space is illustrated in Figure 3-1 of Chapter 3. This section describes the procedure and commands for downloading the VSV21 system software (kernel, device drivers, interpreter and emulators) and the constraints to be observed for this memory to be used efficiently. The components of the VSV21 system software are described in Section 1.4.

18.3.1.1 Downloading Procedure – The procedure is as follows. When you have run or installed VCP (Section 18.1), download the software in the following sequence:

1. Load the kernel. The kernel prepares the VSV21 module to receive other files.

2. Load the pointing device driver. The available pointing device drivers and their logical names are as follows:

   - MSI driver: JSTICK
   - Penny and Giles driver: PGSTICK
   - DECTABLET driver: DECTAB

3. Load the transparent port driver TRANSP, if required.

4. Load an interpreter or emulator. This is one of the following:

   - VIVID interpreter: VIVID

     Translates display lists into a picture

   - VSV11 emulator: VS11EM

     Provides VSV11 emulation capability

   - VT220 emulator: VT220EM

     Full-screen emulation is provided in ROM, but it is cancelled when VIVID or VSV11 is loaded. The VT220EM file replaces full-screen VT220 emulation and removes the existing routine.

18.3.1.2 Memory Management – The kernel and drivers are stored in the high memory locations. The interpreter or emulator is stored in the lowest location (Figure 3-1).

When the VIVID SAVE_ATTRIBUTES instruction is executed, saved attributes are stored in memory locations immediately below the last driver loaded. Downloaded VIVID segments are stored immediately above the interpreter.

When the above routines have been loaded, subsequent loading of drivers has the following effects:

- Only the last pointing device driver loaded is accessible to the application.
- The saved attributes remain accessible, but the space occupied by saved attributes is lost when the attributes are deleted; the VSV21 cannot make use of the area of memory immediately above the driver. A loaded driver can be removed only by reloading the kernel.
- VCP is unable to display or modify the pointing device characteristics as described in Section 18.9.

Subsequent loading of an emulator or the VIVID interpreter removes the existing routine.

For example, the user might do the following in a VSV21 session:

1. Load the VIVID interpreter and digitizing tablet driver at the start of the session.
2. Run VIVID and save attributes.
3. Load the tablet driver again.
4. Execute a VIVID display list and restore attributes.
5. Load the VSV11 emulator.

The result is that the saved attributes remain accessible to the application, but when they are deleted by a VIVID RESTORE_ATTRIBUTES instruction, the space they occupied does not become available to VIVID again. Finally, the VIVID interpreter is deleted and the VSV11 emulator replaces it in the memory.

It is recommended that the pointing device driver is loaded before the interpreter or emulator, and is not reloaded while the VIVID interpreter is resident on the VSV21. This avoids the possibility of wasting VSV21 on-board memory.

At any time, the following routines are available to the application:

- The most recently loaded pointing device driver
- The transparent port driver, if loaded
- The most recently loaded emulator or VIVID interpreter

18.3.1.3 The LOAD Command – Code is downloaded by the LOAD command. The general format of the LOAD command is:

LOAD [dev] filnam

where:

dev = device specification as given in Section 18.2.2.

filnam = name, without extension and UIC, of the file to be loaded. It can be one of the following:

KERNEL – kernel

VIVID – VIVID interpreter

VS11EM – VSV11 emulation software

VT220EM – VT220 emulation software

JSTICK – MSI device driver

PGSTICK – Penny and Giles device driver

DECTAB – digitizing tablet driver

TRANSP – transparent port driver

The files are located on the host system as follows:

- RSX-11M-PLUS – LB:[3,54]
- Micro/RSX – LB:[3,54]
- MicroVMS : SYS$LIBRARY

The host system treats these filenames as unprivileged. Future versions of the system may include additional unprivileged filenames, as well as a list of privileged filenames.

18.3.2 Downloading VIVID Segments

VIVID segments are downloaded to the VSV21 with the VIVID_LOAD_SEG command. The general form of the command is:

```
VIVID_LOAD_SEG filnam
```

where filnam is the name of a host file containing the segments. If no file extension is given, the default .VIV is assumed.

The VIVID default font is downloaded automatically with the VIVID interpreter. It is stored as a segment in VSV21 memory. If the font has been deleted from or overwritten in VSV21 memory, it can be reloaded from the host. The parameter filnam is as follows:

- RSX-11M-PLUS and Micro/RSX – [3,54]DFONT
- MicroVMS – SYS$LIBRARY:DFONT

VSV11 display lists are downloaded by using QIO calls (Chapter 17).

18.4 SETTING VSV21 DISPLAY PARAMETERS

The main function of the VSV21 commands is to allow the user to set VSV21 display parameters, such as the color definitions and the peripheral device characteristics.

The general format of the command for setting VSV21 parameters is:

```
SET_PARAMETER [dev] ctab[/perm] dsp gval[/perm]
```

where:

dev = device specification, as defined in Section 18.2.2

ctab = a parameter chosen from the following list:

```
GRAPHICS_DISPLAY

VT220_DISPLAY

SERIAL_KEYBOARD

SERIAL_HOST

SERIAL_POINTING

SERIAL_TRANSP
```

KEYBOARD

HOST

POINTING

dsp = a display parameter, for example:

TESTCHART

SPLIT_SCREEN

LINES_SPLIT

A_COLORS

B_COLORS

C_COLORS

D_COLORS

COUNT_BLINK

INDEX_BLINK

A_BLINK_COLORS

B_BLINK_COLORS

C_BLINK_COLORS

D_BLINK_COLORS

512_TIMINGS

640_TIMINGS

gval = a numeric or logical value associated with dsp

perm = a switch which decides if the parameter is to remain stored in non-volatile memory when the VSV21 is detached from the host. You can set it to one of the following values:

PERMANENT	if the parameter is stored
NOPERMANENT	if the parameter is not stored. This is the default

The /perm switch cannot be used for the ctab parameter POINTING.

You can enter up to 511 characters using one SET__PARAMETER command, as follows.

```
SET__PARAMETER [dev] ctab[/perm] dsp gval[/perm]-
                                 dsp gval[/perm]-
                                 dsp gval[/perm]-
                                 dsp gval[/perm]
```

If the value of perm which qualifies ctab is set to PERMANENT, all values of dsp and gval in the command line which are not qualified by /NOPERMANENT are put in permanent storage. For example, suppose a SET__PARAMETER command has been included in the program as follows:

```
SET GRAPHICS/PERM VT220_COLORS 5 4 3 2 1
                  SPLIT_SCREEN YES/NOPERM
                  LINES_SPLIT 6
```

The VT220 parameters and the COUNT value are put in permanent store, while the SPLIT__SCREEN parameter is not.

The range of available SET__PARAMETER commands is summarized in Table 18-1. The parameters are described in the sections whose numbers are given in brackets.

Table 18-1 Parameter Values for SET__PARAMETER Commands

ctab	dsp	gval
GRAPHICS__DISPLAY (18.5)	TESTCHART	DISPLAY/NODISPLAY
	A__COLORS	24 hex chars
	B__COLORS	24 hex chars
	C__COLORS	24 hex chars
	D__COLORS	24 hex chars
	COUNT__BLINK	integer, range 0-16
	INDEX__BLINK	1 to 16 integers, range 0-15
	A__BLINK__COLORS	1 to 24 hex chars
	B__BLINK__COLORS	1 to 24 hex chars
	C__BLINK__COLORS	1 to 24 hex chars
	D__BLINK__COLORS	1 to 24 hex chars
	512__TIMINGS	20 hex chars
	640__TIMINGS	20 hex chars

Table 18-1 (Cont) Parameter Values for SET_PARAMETER Commands

ctab	dsp	gval
VT220_DISPLAY (18.6)	SPLIT_SCREEN	YES/NO
	LINES_SPLIT	2 to 12
	VT220_COLORS	5 integers, range 0-15
SERIAL_KEYBOARD (18.7)	LINE_STATUS	ENABLE/DISABLE
SERIAL_HOST and SERIAL_POINTING (18.7)	LINE_STATUS	ENABLE/DISABLE
	MODE_PARITY	WITH/FORCE/NONE
	TYPE_PARITY	EVEN/ODD
	BITS_CHAR	7 or 8 for SERIAL_HOST; 5 to 8 for SERIAL_POINTING
	TX_SPEED	75 to 9600
	RX_SPEED	75 to 9600
SERIAL_TRANSP (18.7)	LINE_STATUS	ENABLE/DISABLE
	MODE_PARITY	WITH/FORCE/NONE
	TYPE_PARITY	EVEN/ODD
	BITS_CHAR	integer, range 5-8
	TX_SPEED	75 to 9600
	RX_SPEED	75 to 9600
	XON_CODE	2 hex chars
	XOFF_CODE	2 hex chars
HOST (18.7)	MAX_CHAR	integer, range 1-80
	UK_KEYBOARD	HASH/POUND
	SCROLL	SMOOTH/JUMP
	WRAP	YES/NO
	TAB_SETTINGS	1-20 hex chars
KEYBOARD (18.8)	KEY_CLICK	ON/OFF
	BELL	ON/OFF
	AUTOREPEAT	ON/OFF
	MODE	DP/TYPEWRITER
	BITS_CHAR	7 or 8
	NATIONALITY	language name
POINTING (18.9)	X_SQUARE	YES/NO
	X_FACTOR	integer, range 0-32767
	X_SHIFT_COUNT	integer, range 0-15
	Y_SQUARE	YES/NO
	Y_FACTOR	integer, range 0-32767
	Y_SHIFT_COUNT	integer, range 0-15

18.5 SETTING UP THE GRAPHICS DISPLAY

You can set the following display characteristics:

- Whole screen or split screen display
- The range of colors in the display
- The range of colors to be blinked
- The number of colors to be blinked
- The monitor timings

To set these you use SET_PARAMETER GRAPHICS_DISPLAY, which is one of the commands for setting VSV21 parameters (Section 18.4).

The general format of the GRAPHICS_DISPLAY command is as follows:

SET_PARAMETER [dev] GRAPHICS_DISPLAY[/perm] dsp gval[/perm]

where:

dev = device specification, as defined in Section 18.2.2

dsp = a display parameter

gval = a value associated with dsp

perm = a switch as described in Section 18.4.

SET_PARAMETER can conveniently be abbreviated to SET.

If the graphics software is downloaded, you can choose whether or not to display a test chart by using the TESTCHART command as follows:

```
SET GRAPHICS_DISPLAY TESTCHART DISPLAY
```

or

```
SET GRAPHICS_DISPLAY TESTCHART NODISPLAY
```

18.5.1 The Color Look-Up Table (CLUT)

The VSV21 holds the default VSV11 CLUT in PROM. You can use VCP to put this CLUT in NVRAM, and to change the values.

18.5.1.1 Default Colors – The default contents of the CLUT are the same as the default VSV11 colors. The default colors are as follows:

Position	Color	Position	Color
0	Black	8	Mid green
1	Blue	9	Pale blue
2	Red	10	Light orange
3	Violet	11	Pink
4	Dark green	12	Green
5	Mid blue	13	Magenta
6	Orange	14	Yellow
7	Pale violet	15	White

18.5.1.2 Setting Up the CLUT – If you want to use another set of colors in the VSV21 display, you can choose sixteen from a palette of 4096 colors. The colors parameters are used to define these colors. The colors you specify will be inserted in the color look-up table on the VSV21 module.

Six hexadecimal characters are used to define each of the 16 colors, so for each display a total of 96 characters define the colors. To allow the 96 color and intensity characters to be entered as a number of shorter, more convenient strings, the following four colors parameters are used:

A__COLORS gval	defining colors 0 to 3
B__COLORS gval	defining colors 4 to 7
C__COLORS gval	defining colors 8 to 11
D__COLORS gval	defining colors 12 to 15

where gval is a 24-character hexadecimal string which defines four colors.

The characters are arranged as follows. Each of the sixteen available colors requires six hexadecimal characters, arranged in three pairs, to define it. The three pairs define the intensity of red, green, and blue respectively, and the system combines the data to generate a composite color.

The primary colors are coded as follows:

1. Red
2. Green
4. Blue

You can allocate an intensity value to any of these primary color codes. This value ranges from 0 (zero intensity) to 15 (maximum intensity), that is, from 0 to F in hexadecimal notation. Black is defined by allocating zero intensity to each of the three colors, and white by allocating maximum intensity to each.

No error message is returned if the color-intensity pair does not begin with 1, 2, or 4, but incorrect output will result.

Similarly, errors will result if VCP commands in your program attempt to extract data from unfilled positions in the CLUT. For example, if colors 0 to 7 are defined by A__COLORS and B__COLORS parameters, while colors 8 to F are left undefined, attempts to read colors 8 to F will produce unpredictable results.

18.5.1.3 CLUT Examples – To generate red, the three pairs are 1F, 20, 40. They are entered in a continuous string: 1F2040. In this case, green and blue have zero intensity, so only red is produced.

Yellow could be generated by equal intensities of green and blue, omitting red: 102F4F. A less intense yellow would be 102525, and turquoise could be produced by combining a little green with more blue: 10264B.

Black is 102040. White is a combination of the primary colors in maximum intensities: 1F2F4F. Shades of gray through to white are generated by intermediate intensity values which are the same for each primary color.

Suppose you want to put the following nine colors in the CLUT:

Position	Color	Position	Color
0	Green	5	Black
1	Cyan	6	Magenta
2	Turquoise	7	White
3	Red	8	Blue
4	Cyan		

You can do this by using A__COLORS, B__COLORS, and C__COLORS:

```
SET GRAPHICS_DISPLAY A_COLORS 102F4010284810264C1F2040

SET GRAPHICS_DISPLAY B_COLORS 1028481020401A204A1F2F4F

SET GRAPHICS_DISPLAY C_COLORS 10204F
```

18.5.2 Defining Blink Colors

The four BLINK__COLORS parameters allow the user to define the colors to be shown blinking on the display, and store them in a blink colors look-up table (BCLUT). Sixteen such colors can be entered by using the codes for primary colors and their relative intensities in the way described for the Colors parameters in Section 18.5.1.2.

A__BLINK__COLORS gval Defining colors 0 to 3

B__BLINK__COLORS gval Defining colors 4 to 7

C__BLINK__COLORS gval Defining colors 8 to 11

D__BLINK__COLORS gval Defining colors 12 to 15

where gval is a 24-character hexadecimal string which defines four colors.

18.5.3 Forming Pairs of Blink Colors

18.5.3.1 Defining the Range of Blink Colors – You can enter your choice of alternating colors in the form of a string of integers by using INDEX_BLINK. The value of each integer refers to the position of a color you have already defined in the color look-up table. The position of each integer in the string refers to the position of a color already defined in the blink color look-up table.

Example:

Suppose that the colors and blink colors tables have been set up as follows:

Position	Color	Blink Color
0	Green	Orange
1	Yellow	Indigo
2	Turquoise	Turquoise
3	Red	Black
4	Yellow	Red
5	Black	White
6	Mauve	Yellow
7	White	Blue

Then the command line:

```
SET GRAPHICS_DISPLAY INDEX_BLINK 6 5 7 4
```

means that:

blink color 0 alternates with color 6 – orange/mauve

blink color 1 alternates with color 5 – indigo/black

blink color 2 alternates with color 7 – turquoise/white

blink color 3 alternates with color 4 – black/yellow

18.5.3.2 Selecting Blink Colors from the Range – The number of colors for which blinking is enabled is entered using COUNT_BLINK. COUNT_BLINK allows the user to enter the number of colors to be blinked on the display. Its associated gval is a hexadecimal integer between 0 (no colors blinked) and F (all colors blinked).

Example:

```
SET GRAPHICS_DISPLAY COUNT_BLINK 3
```

This allocates the first three color pairs defined by INDEX_BLINK for use in the VSV21 display. The remaining colors in the blink colors table are ignored.

18.5.4 Setting Monitor Characteristics

If you are using the VSV21 with a monitor which is not recommended by DIGITAL, you may need to set the monitor display characteristics during hardware installation.

The 512_TIMINGS and 640_TIMINGS commands set the characteristics of monitors with 512-pixel and 640-pixel wide displays, used respectively for VSV11 emulation and VT220 emulation.

Each command defines the following:

- Scan time
- Sync width
- Display start position
- Display width

both vertically and horizontally.

Each command has eight parameters, grouped in five 4-character hex strings. The parameter groups are as follows:

Group 1:

- Horizontal Cycle. Horizontal scan time, in units of memory cycles. Range 01 – FF.
- Horizontal Sync Width. Horizontal sync active low time, in units of memory cycles. Range 01 – 1F

Group 2:

- Horizontal Display Start. Horizontal display start interval, in units of memory cycles. Range 00 – FF.
- Horizontal Display Width. Horizontal display period for one raster. Range 00 – FF.

Group 3:

- Vertical Cycle. Vertical scan cycle period in units of rasters. Range 0000 – 0FFF.

Group 4:

- Vertical Display Start. Vertical display start interval in units of memory cycles. Range 00 – FF.
- Vertical Sync Width. Vertical low pulse width in units of rasters. Range 00 – 1F

Group 5:

- Vertical Window Width. Vertical display period of window screen in units of rasters. Range 0000 – 0FFF

18.6 SELECTING VT220 EMULATION

The VSV21 can provide a subset of VT220 functionality to enable you to use it as a terminal on the host system. VT220 emulation operates in two modes.

- Full-screen text
- Split-screen text, where a user-definable number of text lines may be used for terminal emulation

You use the SET VT220_DISPLAY command to:

- Enable or disable switching into split-screen mode on host serial input
- Select the colors to use in VT220 emulation
- Select the number of lines for use in VT220 emulation

The general format of the SET VT220_DISPLAY command is:

SET_PARAMETER [dev] VT220_DISPLAY[/perm] dsp gval[/perm]

where:

dev = device specification, as defined in Section 18.2.2

dsp = a display parameter

gval = a value associated with dsp

perm = a switch as described in Section 18.4

For convenience, you can abbreviate SET_PARAMETER to SET.

18.6.1 Splitting the Screen

You can divide the VSV21 screen display into two sections.

- An upper section consisting of graphics display
- A lower section consisting of host text output

SPLIT_SCREEN determines whether the screen automatically splits on host input. Its associated parameter gval is one of the following:

YES Enables split screen (graphics and text display)

NO Disables split screen (graphics display only)

Example:

```
SET_PARAMETER VT220_DISPLAY SPLIT_SCREEN YES
```

When a new SPLIT_SCREEN value is entered, the terminal VSV11 emulation screen output is automatically reset; that is, the text area is cleared.

You can set the number of text lines shown under the graphics display at between two and twelve lines by using LINES_SPLIT. The default is four lines.

Example:

```
SET_PARAMETER VT220_DISPLAY LINES_SPLIT 2
```

produces a two-line text window under the graphics display.

18.6.2 Choosing Colors for VT220 Emulation

VT220_COLORS allows the user to decide which of the colors defined by the Colors parameters are to be used in the VT220 emulation, and how these colors are to be used. Five colors must be specified, using the positions of those colors in the CLUT, separated by commas, spaces, or tabs as input. The function of each color in the display is defined by its position in the VT220_COLORS string as follows:

Position	Function
1	Normal
2	Background
3	Blink
4	Highlight
5	Blink and highlight

Example:

```
SET VT220_DISPLAY VT220_COLORS 5 4 6 0 7
```

This uses the color allocations described in Section 18.5.3 to define black text on a yellow background. The color of the text blinks to turquoise. The text is highlighted in yellow. Blinking text is highlighted in pale red.

Note that the VT220 scrolling regions are not supported in split-screen mode.

18.7 CONFIGURING THE HOST SERIAL CONNECTION

18.7.1 Setting Up the Host Line Characteristics

You set up the communication characteristics by using the following commands:

SERIAL_KEYBOARD

SERIAL_HOST

SERIAL_POINTING

SERIAL_TRANSP

The SERIAL_KEYBOARD command enables or disables the line to the host. You set it either to ENABLE or to DISABLE.

The SERIAL_HOST command sets up the line characteristics of the link to the host. The link resembles the link to a VT220, and the SERIAL_HOST command enables you to perform the same functions as those defined by the VT220 SET-UP key. For a further explanation of these functions, see the *VT220 User's Guide (EK-VT220-UG-002).*

The SERIAL_HOST command defines:

- Line status
- Mode and type of parity
- Number of bits per character
- Baud rates of transmit and receive

The format of the SERIAL_HOST table command is:

SET_PARAMETER [dev] SERIAL_HOST[/perm] dsp gval[/perm]

where:

dev = device specification as defined in Section 18.2.2

dsp = a display parameter. This can be one of the following:

LINE_STATUS

MODE_PARITY

TYPE_PARITY

BITS_CHAR

TX_SPEED

RX_SPEED

gval = a value associated with dsp

perm = a switch as described in Section 18.4

A description of the dsp parameters follows.

LINE_STATUS defines the status of the line. Its associated gval can be either ENABLE or DISABLE.

MODE_PARITY defines the parity mode. The parameter gval can be either WITH, FORCE, or NONE.

TYPE_PARITY defines the parity type. The parameter gval can be either EVEN or ODD.

BITS_CHAR defines the number of bits in each character. The parameter gval can be either 5, 6, 7, or 8.

TX_SPEED and RX_SPEED define the transmit and receive baud rates respectively. The parameter gval can be one of the following:

75	300	2000
110	600	2400
134	1200	4800
150	1800	9600

The dsp parameters defined here are described in more detail in the *VT220 User's Guide (EK-VT220-UG-002)*.

18.7.2 Setting Pointing Device Characteristics

You use the SERIAL_POINTING command to set the line characteristics of the serial port to which the peripheral device is connected.

The format of the SERIAL_POINTING command is as follows:

SET_PARAMETER [dev] SERIAL_POINTING[/perm] dsp gval[/perm]

where:

dev = device specification, as described in Section 18.2.2

dsp = a display parameter. This can be one of the following:

BITS_CHAR

LINE_STATUS

MODE_PARITY

TYPE_PARITY

TX_SPEED

RX_SPEED

gval = a value associated with dsp

perm = a switch as described in Section 18.4

Example:

To define the number of bits in a character, use BITS_CHAR. You can give BITS_CHAR a value of 5, 6, 7, or 8.

```
SET SERIAL_POINTING BITS_CHAR 8
```

This defines an 8-bit character.

18.7.3 Setting Peripheral Device Characteristics

You use the SERIAL_TRANSP parameter to set the characteristics of the peripheral device. This can be any serial device driven by the on-board transparent driver, such as a keyboard or printer.

The parameters are as for SERIAL_POINTING, with the following additions:

dsp	gval
XON_CODE	two hex chars
XOFF_CODE	two hex chars

XON_CODE and XOFF_CODE are communications protocol characters for your own device.

18.7.4 Setting Up the Host Device

You enter the characteristics of the host device by using the HOST command. The command defines:

- Keyboard type
- Screen display type
- Scrolling mechanism
- Existence of wraparound
- Tab settings

The format of the command is:

SET_PARAMETER [dev] HOST[/perm] dsp gval[/perm]

where:

dev = device specification, as defined in Section 18.2.2

dsp = a display parameter. This can be one of the following:

MAX_CHAR

UK_KEYBOARD

SCROLL

WRAP

TAB_SETTINGS

gval = a value associated with dsp

perm = a switch as described in Section 18.4

MAX_CHAR defines the maximum length of the line. The line can be up to 132 characters in length.

The UK keyboard type is defined by UK_KEYBOARD. It is one of the following:

HASH enables the hash character

POUND enables the pound sign

SCROLL defines the scrolling mechanism. It is one of the following:

SMOOTH Smooth scroll

JUMP Jump scroll

WRAP specifies whether or not word wrap is enabled. It is one of the following:

YES Set wrap

NO Cancel wrap

TAB_SETTINGS specifies the character position of the TAB setting.

It is entered as a number of hex characters. The position of the tabs is derived from the position of the "1" characters in the equivalent binary string.

18.8 SETTING UP KEYBOARD CHARACTERISTICS

You set up the keyboard by using the KEYBOARD command. The KEYBOARD command defines:

- Key click
- Bell
- Autorepeat on or off
- DP or typewriter mode
- Number of bits per character
- Character set

The format of the KEYBOARD command is as follows:

SET_PARAMETER [dev] KEYBOARD[/perm] dsp gval[/perm]

where:

dev = device specification as defined in Section 18.2.2

dsp = a display parameter. This can be one of the following:

KEY_CLICK

BELL

AUTOREPEAT

MODE

BITS_CHAR

NATIONALITY

gval = a value associated with dsp

perm = a switch as described in Section 18.4

18.8.1 Key Sound

KEY_CLICK decides whether or not the keys produce a click. You can set it either to ON or to OFF.

18.8.2 Terminal Bell

BELL is a switch controlling the terminal bell. You can set it either to ON or to OFF.

18.8.3 Autorepeat

AUTOREPEAT sets the autorepeat facility to ON or OFF.

18.8.4 Mode

MODE sets the mode to data processing (DP) or typewriter.

18.8.5 Number of Bits

BITS_CHAR sets the keyboard to 7-bit or 8-bit, depending on the operating system type and version.

18.8.6 Character Sets

NATIONALITY sets the VSV21 to receive the character set of the keyboard being used. The VSV21 can handle the following character sets:

US

UK

SWEDISH

DUTCH

FLEMISH

CANADIAN_FRENCH

DANISH

FINNISH

GERMAN

ITALIAN

SWISS_FRENCH

SWISS_GERMAN

NORWEGIAN

FRENCH

SPANISH

18.9 SETTING UP POINTING DEVICE CHARACTERISTICS

You set up the pointing device characteristics by using the POINTING command. The POINTING command defines the sensitivity of the pointing device. Note that a pointing device driver must be loaded (Section 18.3.1) before you can set up the characteristics.

The format of the command is as follows:

SET_PARAMETER [dev] POINTING dsp gval

where:

dev = device specification as defined in Section 18.2.2

dsp = a display parameter. This can be one of the following:

X_SQUARE

X_FACTOR

X_SHIFT_COUNT

Y_SQUARE

Y_FACTOR

Y_SHIFT_COUNT

gval = a value associated with dsp

Note that the /perm switch cannot be used with this command. The values entered with the POINTING command are not stored in non-volatile memory, and they are lost when the machine is switched off.

18.9.1 Sensitivity

X_SQUARE and Y_SQUARE set the sensitivity of the cursor to the pointing device movement in the X and Y directions respectively. The sensitivity in the X or Y direction is proportional to the movement squared if the parameter is set to YES; that is, the distance the cursor moves across the screen is greater per unit movement of the pointing device for larger movements. Cursor movement is in linear proportion to pointing device movement if the parameter is set to NO.

18.9.2 Sensitivity Factors

X__FACTOR and Y__FACTOR are factors by which the sensitivity can be increased. They can be set to any integer value in the range 1 – 32767.

18.9.3 Shift Counts

The shift count is used to generate non-integer sensitivity factors (Section 18.9.2). A shift count value is combined with the X__FACTOR and Y__FACTOR parameters in the following calculation:

$$FACTOR = FACTOR/2^{SHIFT}$$

where:

FACTOR = value of X__FACTOR or Y__FACTOR

SHIFT = value of SHIFT__COUNT parameter (range 0-15)

This allows precise control of sensitivity.

18.10 DISPLAYING CONFIGURATION AND STATUS

To display the stored parameter values, use the TABLE command:

TABLE [dev] ctab

where:

dev = device specification as defined in Section 18.2.2

ctab = one of the strings listed in Section 18.4

The system responds by displaying the parameter values.

To see the status of any VSV21 device, enter:

STATUS dev

where:

dev = device specification as defined in Section 18.2.2

18.11 DUMPING VSV21 MEMORY TO A FILE

To dump the contents of VSV21 memory, enter the command DUMP. This dumps the memory to a file named VCP.DMP.

CHAPTER 19
GETTING STARTED WITH VIVID

This chapter describes the steps in writing and running a VIVID application after system power-up or initialization. The power-up procedure is described in the *VSV21 User Guide (AZ-FV70B-TC).*

19.1 DOWNLOADING THE SOFTWARE

On power-up, the ROM-resident VT220 terminal emulator is active. Before the VSV21 can be used to run a VIVID application, the following routines must be downloaded from the host:

1. Kernel – controls VSV21 operation
2. Pointing device driver – controls joystick, trackball or other devices
3. VIVID interpreter – translates display list into a picture

Load these with the VSV21 Command Program (VCP). The VCP command set is described in Chapter 18 of this Guide. Install the VCP by entering one of the following commands:

`RUN $VCP` on RSX-11M-PLUS and Micro/RSX systems

`MCR VCP` on MicroVMS systems

This returns the prompt `"VCP)"` . Enter the following commands:

Prompt	Command
`VCP)`	`LOAD KERNEL`
`VCP)`	`LOAD JSTICK` (or DECTAB or PGSTICK)
`VCP)`	`LOAD TRANSP` (if printer is to be used)
`VCP)`	`LOAD VIVID`

The VSV21 displays a split screen, and is capable of reduced functionality console emulation. The VSV21 can now interpret VIVID instructions. The VIVID font is downloaded automatically with the VIVID interpreter.

Exit from VCP by pressing CTRL/Z.

The downloaded routines occupy a limited memory space which they share with VIVID segments. To optimize the use of this memory, the user must observe certain constraints in downloading and deleting system software. These are described in Section 18.3.

19.2 DEFINING AND EXECUTING A DISPLAY LIST

This section uses a program example to describe how to write a MACRO-32 program under MicroVMS to define and execute a display list. The program defines and executes a segment which clears the screen to a pattern of colored stripes.

```
;
;       VIVID MACRO-32 Program
;       ======================
;
;
        .TITLE TEST - VIVID test program
        $IODEF

;        Set up I/O function names if necessary. A VIVID application
;        program usually contains the following QIOs:
;
;        RSX and VMS    SECTION     ACTION
;        QIO FUNCTIONS  REFERENCE
;
;
;        IO.ADA or      4.2.3       Allocates a display area for
;        IO$_ALLOCATE               segments in host memory
;
;        IO.DEF or      4.2.5       Defines a segment by entering its
;        IO$_DEFSEG                 address and length on the VSV21
;
;        IO.SSE or      4.2.8       Starts execution of a segment
;        IO$_STARTSEG
;
;
;
IO$_ALLOCATE == IO$_CONINTREAD
IO$_DEFSEG   == IO$_TTYREADALL
IO$_DEFREP   == IO$_DEACCESS
IO$_STARTSEG =  ^X3E
```

```
;
;
;       Define the display area size in host memory.
;       The defined display area must be big enough to hold all the
;       segments you intend to store in host memory.
;       Enter the list of opcodes and parameters which make up
;       the segments.
;
;       These are entered to the display area as a series of .WORD
;       or .BYTE commands. An example is given with each of the
;       VIVID instructions described in Chapters 6 through 14.
;
;       Define the contents of the executable segment:
ADA:
        .WORD   ^X 102              ; opcode for start of segment
        .WORD   ^X0201              ; segment ID
        .WORD   16                  ; segment length
        .WORD   ^X 601              ; opcode to initialize VIVID
        .WORD   ^X  7F              ; mask value
        .WORD   ^X4C01              ; opcode to clear screen
        .WORD   ^X E00              ; color 14 from default CLUT
                                    ; = yellow stripes
        .WORD   ^X C00              ; stop
;
;
;       Define the contents of the report segment:
;
REPSEG:
        .WORD   0                   ; segment type filled in by VIVID
        .WORD   ^X0202              ; segment ID
        .WORD   200                 ; segment length (octal)
;
;
;
;       Allocate space for the reporting segment and other segments.
;       The display area must be big enough for all the segments you
;       intend to store in host memory. This instruction gives 60000
;       bytes in addition to the 16 used above.
;
BUFF:
        .BLKB   60000
;
;
;       Store the channel number, I/O status block and device name:
;
CHAN:   .LONG                       ; storage for VSV21 channel no.
IOSB:   .BLKB   8                   ; I/O status block
DEV:    .ASCID  /VSA0:/             ; device name for assignment
```

```
;
;
;       Define start of code:
;
        .ENTRY  TEST,   ^M<>
;
;
;       Assign a channel and device name to the VSV21 device:
;
;       This uses the stored information already set up.
;
        $ASSIGN_S       CHAN=CHAN,-
                        DEVNAM=DEV
;
;
;       Allocate a display area for all the concurrent segments:
;
;       Use the IO.ADA or IO$_ALLOCATE function and supply the
;       address of the segment and total length of the display area
;       as parameters.
;
        $QIOW_S CHAN=CHAN,-
                FUNC=#IO$_ALLOCATE,-
                IOSB=IOSB,-
                P1=ADA,-            ; starting address of display
                                    ; area
                P2=#60022           ; length of display area
;
;
;        Define the executable segment:
;
;        Use the IO.DEF or IO_$DEFSEG function and supply the segment
;        address and length as parameters:
;
        $QIOW_S CHAN=CHAN,-         ; channel number
                FUNC=#IO$_DEFSEG,-  ; define segment
                IOSB=IOSB,-         ; I/O status block
                P1=ADA,-            ; starting address of display
                                    ; area
                P2=#16              ; segment length
;
;
;       Check the I/O status block:
;
;       The contents of the I/O status block are described in
;       Section 4.3.
```

```
;
        BLBC    R0,1$
        MOVW    IOSB,R0
        BLBS    R0,2$
1$:     BRW     EXIT
2$:
;
;

;       Define the reporting segment:
;
;       Use the IO.DEF or IO_$DEFSEG function and supply the segment
;       address and length as parameters.
;
        $QIOW_S CHAN=CHAN,-         ; channel number
                FUNC=#IO$_DEFSEG,-  ; define segment
                IOSB=IOSB,-         ; I/O status block
                P1=REPSEG,-         ; starting address of display area
                P2=#200             ; segment length
;
;
;       Check the I/O status block:
;
;
        BLBC    R0,EXIT
        MOVW    IOSB,R0
        BLBC    R0,EXIT
;
;
;        Define reporting:
;
;        Use the IO.DRP or IO_$DEFREP function to define the segment
;        as a reporting segment.
;
                $QIOW_S CHAN=CHAN,- ; channel number
                FUNC=#IO$_DEFREP,-  ; define reporting
                IOSB=IOSB,-         ; I/O status block
                P1=@REPSEG+2        ; segment ID
;
;
;       Check the I/O status block:
;
        BLBC    R0,EXIT
        MOVW    IOSB,R0
        BLBC    R0,EXIT
;
;
;       Start segment execution:
;
```

```
;       Use the IO.SSE or IO$_STARTSEG function and supply the
;       location of the segment ID and the required time-out value
;       as parameters.
;

        $QIOW_S CHAN=CHAN,-
                FUNC=#IO$_STARTSEG,-
                IOSB=IOSB,-
                P1=@ADA+2,-             ; segment ID from second word
                                        ; of seg.
                P2=#10                  ; time-out period in seconds
;
;
;       Check the I/O status block:
;
        BLBC    R0,EXIT
        MOVW    IOSB,R0
;
;
;       Exit
;
EXIT:
       $EXIT_S  R0
       .END     TEST
```

APPENDIX A
THE QIO CALL MECHANISM

A program can use QIO calls to perform a number of different functions. Each QIO call specifies one function which has an associated function code. For example, the IO.CON function sends a display list to the VSV21 to be displayed. In this function, the programmer must specify the address of the display list as a parameter to the QIO call.

The device driver handles all communication between application programs and the VSV21 device. It receives input/output requests from programs, in the form of QIO calls to system service routines in the operating system. The driver passes the requests to the VSV21 processor in the form of command packets, using a programmed I/O mechanism and Direct Memory Access (DMA).

The following types of function are provided for the VSV21 device driver:

- Configuration
- Initialization
- Diagnostic and self-test
- Device control
- Drawing control

The VSV21 device driver provides two sets of QIO functions:

- VIVID functions
- VSV11 emulation functions

The two sets of functions are not compatible. The VSV21 must be set for the VIVID or VSV11 functions by downloading either the VIVID interpreter or the VSV11 emulation code before running an application.

For the VSV21 to display a picture, the display list must be interpreted by code on the VSV21 module. The display list is used to build up the picture in the pixel memory. The device driver does not send the display list across the parallel interface with the rest of the command packet. Instead, the command packet causes the VSV21 to initiate a transfer using a fast Direct Memory Access (DMA) mechanism. DMA is used whenever a large amount of data needs to be transferred to the VSV21. For example, DMA is also used to download the emulation code.

MACRO-11 programs issue QIO requests by calling a system macro, whereas high-level languages such as FORTRAN-77 call subroutines to perform QIO requests. Each request is processed by routines in the executive, and is placed in a request queue. The device driver processes requests from the queue in order of priority. There are two methods by which the program can test whether a transfer is complete, as follows:

- Synchronous I/O

  The program requests return of control only when the transfer is complete. For synchronous I/O you use the QIOW (Queue Input/Output and Wait) form of the QIO call. This method is used in the examples in this chapter.

- Asynchronous I/O

  The program requests immediate return of control before the transfer is complete, so that the program can continue processing while the transfer is in progress. For asynchronous I/O you use the basic QIO (Queue Input/Output) form of the call. When the program reaches a point where it needs to synchronize with the completion of the transfer, it must test whether the transfer is complete. Completion is notified by the setting of the associated event flag, which you specify in the program as a parameter in the QIO call.

The return status code, which notifies whether or not the transfer was completed successfully, is placed in the I/O status block. This is a data area which is set up in the application program. You should check the return status after every QIO call, and provide error-processing routines for each type of error.

Further information about the QIO request is given in the *RSX-11M/M-PLUS Executive Reference Manual (AA-L675A-TC), the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual (AA-Z508A-TC),* and the *VAX/VMS System Services Reference Manual (AA-D018C-TE).* FORTAN programmers should also refer to these manuals for a description of the subroutine calls which are equivalent to the QIO macro calls.

A.1 EXAMPLES OF QIO CALLS UNDER RSX-11M-PLUS AND MICRO/RSX

Each function code mnemonic is listed in the following examples with a corresponding octal value. For example, the following function attaches the task to the VSV21 device on Micro/RSX and RSX-11M-PLUS systems:

IO.ATT = octal code 1400

To use the function code mnemonic in the QIO call, set up the octal equivalents at the top of your program, for example:

```
;
;  VSV21 QIO function definitions
;
IO.ATT=1400      ; ATTACH
   .
   .
   .
```

These examples assume that you have set up these function codes in your program.

You can set up the QIO directives as a Directive Parameter Block (DPB) in your program, and call them with the DIR$ directive. This method speeds up processing. For example:

```
VSVATT: QIOW$   IO.ATT,1,1,,IOSB,,<ACAT,ACLNG,ACAT>
          .
          .
          .

        DIR$    #VSVATT                 ; ATTACH VSV21 DEVICE
```

To put variable data into the DPB at run time, use the local symbol definitions described in the *RSX-11M/M-PLUS Executive Reference Manual (AA-L675A-TC)* and the *RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual (AA-Z508A-TC)* and the *VAX/VMS System Services Reference Manual (AA-D018C-TE).* For example:

```
          MOV     #DL,R0                ; ADDRESS OF DISPLAY LIST
          MOV     R0,VSVCON+Q.IOPL      ; ... STORE IN QIO DPB
```

This method has been used for the examples in this manual.

A.1.1 The IO.ATT Function

Program example:

```
IOSB:   .BLKW   2                       ; VSV21 I/O STATUS BLOCK

VSVATT: QIOW$   IO.ATT,1,1,,IOSB,,<ACAT,ACLNG,ACAT>

          .
          .
          .

        DIR$    #VSVATT                 ; ATTACH TO VSV21 DEVICE
        CMPB    #IS.SUC,IOSB            ; CHECK RETURN STATUS
        BEQ     15$                     ; BRANCH IF OK
        JMP     AERR                    ; ATTACH ERROR
```

A.1.2 The GLUN$ Function

```
BUFF:   .BLKW   6                       ; BUFFER FOR GLUN$
          .
          .
          .
        GLUN$   1,#BUFF
; GET INFORMATION ON LUN 1 DEVICE AND PLACE IN "BUFF"
```

GLUN$ sets the contents of the six-word buffer as shown in Figure 17-1.

A.1.3 The IO.CON Function

A.1.3.1 Task-Relative Addressing – The following code sets up a display list, DL, and specifies task-relative addressing.

```
DL:         .                   ; START OF AN AREA <---------
            .                   ; USED TO BUILD A           !
            .                   ; DISPLAY LIST...           !

        .WORD   160000          ; DISPLAY JUMP INSTRUCTION  !
        .WORD   DL              ; JUMP TO DISPLAY LIST +0 --->
        DLNG = .-DL             ; DISPLAY LIST LENGTH
            .
            .
            .

VSQ:    QIOW$   IO.CON,1,1,,IOSB,,<DL,DLNG,DL,64>
```

A.1.3.2 Display List-Relative Addressing – This example generates a display list, DL, and the QIO user parameters needed for addressing relative to the start of the display list. Note that the display list is aligned to the start of a block (32-word) boundary.

```
DL:         .                   ; START OF AN AREA (---------
            .                   ; USED TO BUILD A            ¦
            .                   ; DISPLAY LIST...            ¦
          .WORD   160000        ; DISPLAY JUMP INSTRUCTION   ¦
          .WORD   DL            ; JUMP TO DISPLAY LIST +0 --->
          DLNG = .-DL           ; DISPLAY LIST LENGTH +N (N<64)
              .
              .
              .

VSQ:      QIOW$   IO.CON,1,1,,IOSB,,<0,DLNG,0,64>
              .
              .
              .

          MOV     #DL,R0        ; ADDRESS OF DISPLAY LIST DL
          NEG     R0            ; ROUND UP TO NEXT BLOCK (32 WORDS)
          BIC     #^C77,R0      ; USE BITS 5-0 ONLY
          ADD     #DL,R0        ; COMPUTE DISPLAY LIST START
          MOV     R0,VSQ+Q.IOPL ; STORE IT IN THE VSV21 QIO CALL
          DIR$    #VSQ          ; ISSUE QIO
```

A.1.4 The IO.RJS Function

```
XC:       .WORD   0                     ; BUFFER TO HOLD X COORDINATE
YC:       .WORD   0                     ; AND Y COORDINATE

                .
                .
                .

VSVRJS:   QIOW$   IO.RJS,1,1,,IOSB,,<XC,4>      ; READ JOYSTICK COORDS
```

MACRO-11 programmers can test for status returns using the mnemonic code given, for example:

```
          CMPB    #IS.SUC,IOSB          ; CHECK RETURN STATUS
          BEQ     15$                   ; BRANCH IF SUCCESSFUL
          JMP     ERR                   ; JUMP TO ERROR ROUTINE
```

FORTRAN programmers should use the numeric code given to check for errors. For example:

```
BYTE    IOSB(4)                 ! I/O STATUS BLOCK
      .
      .

IF (IOSB(0).NE.1) GO TO 5000    ! BRANCH TO ERROR ROUTINE
      .                         ! HERE IF TRANSFER SUCCESSFUL
      .
```

APPENDIX B
DEFINITION OF THE DEFAULT FONT

This table describes the default font. The font length is given, and each character is defined as a 10-row cell.

```
* Describe font length and characteristics

        START_FONT      $F7F,END_FONT-START_DFONT,$FD,$8,$A,$0
        DC.B    2
        DC.B    6
        DC.W    $F7F
        DC.W    END_FONT-START_DFONT
        DC.W    $FD
        DC.W    $8
        DC.W    $A
        DC.W    $0

* Describe matrix for each character

CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  1    Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  2    Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  3    Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  4    Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  5    Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  6    Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  7    Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  10   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  11   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  12   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  13   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  14   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  15   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  16   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  17   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  20   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  21   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  22   Non printing
```

```
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  23   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  24   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  25   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  26   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  27   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  30   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  31   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  32   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  33   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  34   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  35   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  36   Non printing
CELL10  FF,FF,FF,FF,FF,FF,FF,FF,FF,FF   *  37   Non printing
CELL10  00,00,00,00,00,00,00,00,00,00   *  40   Space
CELL10  00,00,10,00,10,10,10,10,10,00   *  41   !
CELL10  00,00,00,00,00,00,00,48,48,00   *  42   "
CELL10  00,00,24,24,7E,24,7E,24,24,00   *  43   #
CELL10  00,00,10,7C,90,7C,12,7C,10,00   *  44
CELL10  00,00,C2,C4,08,10,20,46,86,00   *  45   %
CELL10  00,00,BC,42,A2,1C,22,22,1C,00   *  46   &
CELL10  00,00,00,00,00,00,08,10,30,00   *  47   '
CELL10  00,00,20,10,08,08,08,10,20,00   *  50   (
CELL10  00,00,08,10,20,20,20,10,08,00   *  51   )
CELL10  00,00,00,44,28,FE,28,44,00,00   *  52   *
CELL10  00,00,00,10,10,FE,10,10,00,00   *  53   +
CELL10  00,04,08,18,00,00,00,00,00,00   *  54   ,
CELL10  00,00,00,00,00,FE,00,00,00,00   *  55   -
CELL10  00,00,18,18,00,00,00,00,00,00   *  56   .
CELL10  00,00,02,04,08,10,20,40,80,00   *  57   /
CELL10  00,00,38,44,82,82,82,44,38,00   *  60   0
CELL10  00,00,7C,10,10,10,14,18,10,00   *  61   1
CELL10  00,00,FE,02,0C,70,80,42,3C,00   *  62   2
CELL10  00,00,3C,42,80,70,20,40,FE,00   *  63   3
CELL10  00,00,20,20,FE,24,28,30,20,00   *  64   4
CELL10  00,00,7C,82,80,86,7A,02,FE,00   *  65   5
CELL10  00,00,78,84,86,7A,02,84,78,00   *  66   6
CELL10  00,00,04,08,10,20,40,80,FE,00   *  67   7
CELL10  00,00,7C,82,82,7C,82,82,7C,00   *  70   8
CELL10  00,00,3C,42,80,BC,C2,42,3C,00   *  71   9
CELL10  00,00,18,18,00,00,18,18,00,00   *  72   :
CELL10  00,04,08,18,00,00,18,18,00,00   *  73   ;
CELL10  00,00,80,20,08,02,08,20,80,00   *  74   <
CELL10  00,00,00,00,FE,00,FE,00,00,00   *  75   =
CELL10  00,00,02,08,20,80,20,08,02,00   *  76   >
CELL10  00,00,10,00,10,10,60,82,7C,00   *  77   ?
CELL10  00,00,7C,02,72,92,A2,82,7C,00   * 100   @
CELL10  00,00,82,82,FE,82,44,28,10,00   * 101   A
CELL10  00,00,7E,84,84,7C,84,84,7E,00   * 102   B
CELL10  00,00,78,84,02,02,02,84,78,00   * 103   C
```

```
CELL10  00,00,3E,44,84,84,84,44,3E,00   * 104   D
CELL10  00,00,FE,02,02,3E,02,02,FE,00   * 105   E
CELL10  00,00,02,02,02,3E,02,02,FE,00   * 106   F
CELL10  00,00,78,84,E2,02,02,84,78,00   * 107   G
CELL10  00,00,82,82,82,FE,82,82,82,00   * 110   H
CELL10  00,00,7C,10,10,10,10,10,7C,00   * 111   I
CELL10  00,00,3C,42,40,40,40,40,E0,00   * 112   J
CELL10  00,00,82,62,1A,06,1A,62,82,00   * 113   K
CELL10  00,00,FE,02,02,02,02,02,02,00   * 114   L
CELL10  00,00,82,82,82,92,AA,C6,82,00   * 115   M
CELL10  00,00,82,C2,A2,92,8A,86,82,00   * 116   N
CELL10  00,00,7C,82,82,82,82,82,7C,00   * 117   O
CELL10  00,00,02,02,02,7E,82,82,7E,00   * 120   P
CELL10  00,00,BC,42,A2,82,82,82,7C,00   * 121   Q
CELL10  00,00,82,42,22,7E,82,82,7E,00   * 122   R
CELL10  00,00,7C,82,80,7C,02,82,7C,00   * 123   S
CELL10  00,00,10,10,10,10,10,10,FE,00   * 124   T
CELL10  00,00,7C,82,82,82,82,82,82,00   * 125   U
CELL10  00,00,10,28,28,44,44,82,82,00   * 126   V
CELL10  00,00,28,7C,D6,92,92,82,82,00   * 127   W
CELL10  00,00,82,C6,6C,38,6C,C6,82,00   * 130   X
CELL10  00,00,10,10,10,10,28,44,82,00   * 131   Y
CELL10  00,00,FE,06,0C,38,60,C0,FE,00   * 132   Z
CELL10  00,00,78,08,08,08,08,08,78,00   * 133   [
CELL10  00,00,80,40,20,10,08,04,02,00   * 134   \
CELL10  00,00,3C,20,20,20,20,20,3C,00   * 135   ]
CELL10  00,00,00,00,00,82,44,28,10,00   * 136   ^
CELL10  00,00,FE,00,00,00,00,00,00,00   * 137   _
CELL10  00,00,00,00,00,00,20,10,18,00   * 140   `
CELL10  00,00,FC,82,FC,80,7C,00,00,00   * 141   a
CELL10  00,00,7A,86,82,86,7A,02,02,00   * 142   b
CELL10  00,00,F8,04,02,84,78,00,00,00   * 143   c
CELL10  00,00,BC,C2,82,C2,BC,80,80,00   * 144   d
CELL10  00,00,7C,02,FE,82,7C,00,00,00   * 145   e
CELL10  00,00,08,08,08,3E,08,88,70,00   * 146   f
CELL10  3C,42,40,7C,42,42,7C,00,00,00   * 147   g
CELL10  00,00,82,82,82,86,7A,02,02,00   * 150   h
CELL10  00,00,7C,10,10,10,18,00,10,00   * 151   i
CELL10  3C,42,42,40,40,40,40,00,40,00   * 152   j
CELL10  00,00,C2,32,0E,12,22,02,02,00   * 153   k
CELL10  00,00,38,10,10,10,10,10,18,00   * 154   l
CELL10  00,00,82,92,92,92,7E,00,00,00   * 155   m
CELL10  00,00,82,82,82,86,7A,00,00,00   * 156   n
CELL10  00,00,7C,82,82,82,7C,00,00,00   * 157   o
CELL10  02,02,02,7A,86,86,7A,00,00,00   * 160   p
CELL10  80,80,80,BC,C2,C2,BC,00,00,00   * 161   q
CELL10  00,00,04,04,04,8C,72,00,00,00   * 162   r
CELL10  00,00,7E,80,7C,02,7C,00,00,00   * 163   s
```

```
CELL10  00,00,30,48,08,08,3E,08,08,00   * 164   t
CELL10  00,00,BC,42,42,42,42,00,00,00   * 165   u
CELL10  00,00,10,28,44,82,82,00,00,00   * 166   v
CELL10  00,00,44,AA,92,82,82,00,00,00   * 167   w
CELL10  00,00,42,24,18,24,42,00,00,00   * 170   x
CELL10  3C,42,40,5C,62,42,42,00,00,00   * 171   y
CELL10  00,00,FE,08,30,40,FE,00,00,00   * 172   z
CELL10  00,00,E0,10,10,0C,10,10,E0,00   * 173   {
CELL10  00,10,10,10,00,00,10,10,10,00   * 174   |
CELL10  00,00,0E,10,10,C0,10,10,0E,00   * 175   }
CELL10  00,00,00,00,00,00,62,92,8C,00   * 176   ~
CELL10  00,00,00,00,00,00,00,00,00,00   * 177   blank
CELL10  00,00,10,38,7C,FE,7C,38,10,00   * 200   diamond
CELL10  00,00,92,44,92,44,92,44,92,00   * 201   grill
CELL10  20,20,20,F8,22,22,3E,22,22,00   * 202   HT
CELL10  10,10,30,10,F2,02,0E,02,1E,00   * 203   FF
CELL10  88,88,78,88,78,3C,02,02,3C,00   * 204   CR
CELL10  08,08,78,08,F8,3E,02,02,02,00   * 205   LF
CELL10  00,00,00,00,00,38,44,44,38,00   * 206   degree
CELL10  00,00,FE,10,10,FE,10,10,00,00   * 207   ±
CELL10  f8,08,08,08,22,32,2A,26,22,00   * 210   NL
CELL10  20,20,20,20,F8,08,14,22,22,00   * 211   VT
CELL10  00,00,00,00,00,0F,08,08,08,08   * 212
CELL10  08,08,08,08,08,0F,00,00,00,00   * 213
CELL10  08,08,08,08,08,F8,00,00,00,00   * 214
CELL10  00,00,00,00,00,F8,08,08,08,08   * 215
CELL10  08,08,08,08,08,FF,08,08,08,08   * 216
CELL10  00,00,00,00,00,00,00,00,00,FF   * 217   scan 1
CELL10  00,00,00,00,00,00,00,FF,00,00   * 220   scan 3
CELL10  00,00,00,00,00,FF,00,00,00,00   * 221   scan 5
CELL10  00,00,00,FF,00,00,00,00,00,00   * 222   scan 7
CELL10  00,FF,00,00,00,00,00,00,00,00   * 223   scan 9
CELL10  08,08,08,08,08,F8,08,08,08,08   * 224
CELL10  08,08,08,08,08,0F,08,08,08,08   * 225
CELL10  00,00,00,00,00,FF,08,08,08,08   * 226
CELL10  08,08,08,08,08,FF,00,00,00,00   * 227
CELL10  08,08,08,08,08,08,08,08,08,08   * 230
CELL10  00,FE,80,20,08,02,08,20,80,00   * 231
CELL10  00,FE,02,08,20,80,20,08,02,00   * 232
CELL10  00,00,44,48,48,48,FE,00,00,00   * 233   pi
CELL10  00,00,02,04,FC,10,FC,40,80,00   * 234   not =
CELL10  00,04,CA,3C,08,3E,08,88,70,00   * 235   pound
CELL10  00,00,00,00,00,10,00,00,00,00   * 236   dot
CELL10  00,00,00,00,00,00,00,00,00,00   * 237   -spare-
CELL10  00,00,00,00,00,00,00,00,00,00   * 240   -spare-
CELL10  00,00,10,10,10,10,10,00,10,00   * 241   inverse !
CELL10  00,00,10,F8,14,12,94,78,10,00   * 242   cent
CELL10  00,04,CA,FC,08,FE,08,88,70,00   * 243   pound
CELL10  00,00,00,00,00,00,00,00,00,00   * 244   -spare-
```

```
CELL10  00,00,10,10,7C,10,28,44,82,00   * 245   Yen
CELL10  00,00,00,00,00,00,00,00,00,00   * 246   -spare-
CELL10  7E,80,80,7C,82,7C,02,02,FC,00   * 247   section
CELL10  00,00,82,44,B8,44,B8,44,82,00   * 250   currency
CELL10  7C,82,B2,8A,8A,8A,B2,82,7C,00   * 251   (C)
CELL10  00,00,FC,00,FC,82,FC,80,7C,00   * 252   fem. ord. ind.
CELL10  00,00,90,48,42,12,42,48,90,00   * 253   <<
CELL10  00,00,00,00,00,00,00,00,00,00   * 254   -spare-
CELL10  00,00,00,00,00,00,00,00,00,00   * 255   -spare-
CELL10  00,00,00,00,00,00,00,00,00,00   * 256   -spare-
CELL10  00,00,00,00,00,00,00,00,00,00   * 257   -spare-
CELL10  00,00,00,00,00,38,44,44,38,00   * 260   degree
CELL10  00,00,FE,10,10,FE,10,10,00,00   * 261   ±
CELL10  00,00,00,7C,04,08,30,44,38,00   * 262   superscript 2
CELL10  00,00,00,38,44,40,30,20,7C,00   * 263   superscript 3
CELL10  00,00,00,00,00,00,00,00,00,00   * 264   -spare-
CELL10  20,40,38,44,44,44,44,00,00,00   * 265   micro
CELL10  00,00,90,90,90,90,9E,9E,FE,00   * 266   paragraph
CELL10  00,00,00,00,18,18,00,00,00,00   * 267   middle dot
CELL10  00,00,00,00,00,00,00,00,00,00   * 270   -spare-
CELL10  00,00,00,38,10,10,10,18,10,00   * 271   superscript 1
CELL10  00,00,7C,00,38,44,44,44,38,00   * 272   masc. ord. ind.
CELL10  00,00,12,24,48,90,48,24,12,00   * 273   >>
CELL10  00,00,42,F4,68,52,22,42,82,00   * 274   1/4
CELL10  00,00,C0,44,88,D2,22,42,82,00   * 275   1/2
CELL10  00,00,00,00,00,00,00,00,00,00   * 276   -spare-
CELL10  00,00,3C,42,84,08,10,00,10,00   * 277   inverse ?
CELL10  00,00,44,7C,44,28,10,40,30,00   * 300   A grave
CELL10  00,00,44,7C,44,28,10,04,18,00   * 301   A acute
CELL10  00,00,44,7C,44,28,10,44,38,00   * 302   A circumflex
CELL10  00,00,44,7C,44,28,10,00,28,54   * 303   A tilde
CELL10  00,00,44,7C,44,28,10,00,28,00   * 304   A umlaut
CELL10  00,00,44,7C,44,28,10,28,10,00   * 305   A ring
CELL10  00,00,F2,12,1E,B2,12,14,F8,00   * 306   AE ligature
CELL10  80,10,78,84,02,02,02,84,78,00   * 307   C cedilla
CELL10  00,00,7C,04,3C,04,7C,40,30,00   * 310   E grave
CELL10  00,00,7C,04,3C,04,7C,04,18,00   * 311   E acute
CELL10  00,00,7C,04,3C,04,7C,44,38,00   * 312   E circumflex
CELL10  00,00,7C,04,3C,04,7C,00,28,00   * 313   E umlaut
CELL10  00,00,7C,10,10,10,7C,40,30,00   * 314   I grave
CELL10  00,00,7C,10,10,10,7C,04,18,00   * 315   I acute
CELL10  00,00,7C,10,10,10,7C,44,38,00   * 316   I circumflex
CELL10  00,00,7C,10,10,10,7C,00,28,00   * 317   I umlaut
CELL10  00,00,00,00,00,00,00,00,00,00   * 320   -spare-
CELL10  00,00,44,64,54,4C,44,00,28,54   * 321   N tilde
CELL10  00,00,38,44,44,44,38,40,30,00   * 322   O grave
CELL10  00,00,38,44,44,44,38,04,18,00   * 323   O acute
CELL10  00,00,38,44,44,44,38,44,38,00   * 324   O circumflex
CELL10  00,00,38,44,44,44,38,00,28,54   * 325   O tilde
```

```
CELL10  00,00,38,44,44,44,38,00,28,00   * 326   O umlaut
CELL10  00,00,EC,12,12,B2,12,12,EC,00   * 327   OE ligature
CELL10  00,00,7C,86,8A,92,A2,C2,7C,00   * 330   O slash
CELL10  00,00,38,44,44,44,44,20,18,00   * 331   U grave
CELL10  00,00,38,44,44,44,44,80,30,00   * 332   U acute
CELL10  00,00,38,44,44,44,44,28,10,00   * 333   U circumflex
CELL10  00,00,38,44,44,44,44,00,28,00   * 334   U umlaut
CELL10  00,00,10,10,10,28,44,00,28,00   * 335   Y umlaut
CELL10  00,00,00,00,00,00,00,00,00,00   * 336   -spare-
CELL10  26,04,FC,84,84,7C,84,84,FE,00   * 337   German sml s
CELL10  00,00,FC,82,FC,80,7C,20,18,00   * 340   a grave
CELL10  00,00,FC,82,FC,80,7C,08,30,00   * 341   a acute
CELL10  00,00,FC,82,FC,80,7C,28,10,00   * 342   a circumflex
CELL10  00,00,FC,82,FC,80,7C,00,28,54   * 343   a tilde
CELL10  00,00,FC,82,FC,80,7C,00,28,00   * 344   a umlaut
CELL10  00,00,FC,82,FC,80,7C,28,10,00   * 345   a ring
CELL10  00,00,FC,12,7C,90,6E,00,00,00   * 346   ae ligature
CELL10  80,10,F8,04,02,84,78,00,00,00   * 347   c cedilla
CELL10  00,00,7C,02,FE,82,7C,20,18,00   * 350   e grave
CELL10  00,00,7C,02,FE,82,7C,08,30,00   * 351   e acute
CELL10  00,00,7C,02,FE,82,7C,28,10,00   * 352   e circumflex
CELL10  00,00,7C,02,FE,82,7C,00,28,00   * 353   e umlaut
CELL10  00,00,7C,10,10,10,18,40,30,00   * 354   i grave
CELL10  00,00,7C,10,10,10,18,04,18,00   * 355   i acute
CELL10  00,00,7C,10,10,10,18,28,10,00   * 356   i circumflex
CELL10  00,00,7C,10,10,10,18,00,28,00   * 357   i umlaut
CELL10  00,00,00,00,00,00,00,00,00,00   * 360   -spare-
CELL10  00,00,82,82,82,86,7A,00,28,54   * 361   n tilde
CELL10  00,00,7C,82,82,82,7C,20,18,B0   * 362   o grave
CELL10  00,00,7C,82,82,82,7C,08,30,00   * 363   o acute
CELL10  00,00,7C,82,82,82,7C,28,10,00   * 364   o circumflex
CELL10  00,00,7C,82,82,82,7C,00,28,54   * 365   o tilde
CELL10  00,00,7C,82,82,82,7C,00,28,00   * 366   o umlaut
CELL10  00,00,FC,12,72,92,7C,00,00,00   * 367   oe ligature
CELL10  00,00,7C,8A,B2,C2,7C,00,00,00   * 370   o slash
CELL10  00,00,BC,42,42,42,42,20,18,00   * 371   u grave
CELL10  00,00,BC,42,42,42,42,80,30,00   * 372   u acute
CELL10  00,00,BC,42,42,42,42,28,10,00   * 373   u circumflex
CELL10  00,00,BC,42,42,42,42,00,28,00   * 374   u umlaut
CELL10  00,00,BC,42,42,42,42,00,24,00   * 375   y umlaut
```

APPENDIX C
EXAMPLE OF A VSV11 EMULATION PROGRAM

```
C       EXAMPLE OF A VSV11 FORTRAN-77 PROGRAM

        PROGRAM BARGEN

C       This program draws a test card of
C       parallel lines and colored bars across the screen. These
C       instructions are sent to a user-specified file.

        INTEGER*2     I,MVX(12),MVY(12),LNX(12),LNY(12)

        DATA    MVX   / 30, 60, 90,421,451,481,511,511,511,511,511,511/

        DATA    MVY   /  0,  0,  0,  0,  0,  0, 30, 60, 90,421,451,481/

        DATA    LNX   / 30, 60, 90,421,451,481,  0,  0,  0,  0,  0,  0/

        DATA    LNY   /511,511,511,511,511,511, 30, 60, 90,421,451,481/


        CALL VSINIT

        CALL VSCLR
C       Select colors
        CALL VSBACK(15)
        CALL VSCOLR(0)

C       Draw parallel lines
        DO 100 I = 1,12
           CALL VSMOVE(MVX(I),MVY(I))
           CALL VSDTHK(LNX(I),LNY(I),3)
100     CONTINUE

C       Draw color bars
```

```
C       Set up histogram
        CALL VSMOVE(92,94)
        CALL VSETHB(94)
        CALL VSHINC(40)

C       Draw 15 colors
        DO 200 I = 0,14
           CALL VSCOLR(I)
           CALL VSHSTY(418,1,0)
200     CONTINUE

C       Draw 16th color
        CALL VSMIX(8,11)
        CALL VSHINC(2*(421-93-15*20))
        CALL VSHSTY(418,1,0)

        CALL VSSYNC

        END
```

APPENDIX D
VIVID ATTRIBUTE MASK VALUES

Initialization items and their initialization mask values are listed in Table D-1. The initialization values are given in terms of the equivalent VIVID instructions. The mask values are additive parameters to the VIVID INITIALIZE instruction.

Where the current values of items are also entered to the attribute stack by the VIVID SAVE_ATTRIBUTES instruction, a "yes" appears in the "SAVED" column. The mask value given is also used on RESTORE_ATTRIBUTES to indicate that the item should be restored. It replaces the item's current value.

Note that the color look-up table (CLUT) and associated blink colors are not initialized. This provides the application with complete control of the color palette used. The user can control the palette using a VCP command procedure or a VIVID display segment.

Table D-1 VIVID Attribute Mask Values

Mask Value		Attribute Group	Initialization Equivalent VIVID Instruction	Saved
Dec	Octal			
1	1	Current Pointer	MOVE_ABS 0, 0	Yes
2	2	Cursor	CURSOR_STYLE 0	Yes
			POSITION_CURSOR 0, 0	Yes
			CURSOR_VISIBILITY 0	Yes
			RUBBER_BAND 0	Yes
4	4	Drawing Colors	FOREGROUND_COLOR 15	Yes
			BACKGROUND_COLOR 0	Yes
8	10	Drawing Mode	DRAWING_MODE 0, 0	Yes
16	20	Texture	LINE_TEXTURE 1, 1	Yes
			AREA_TEXTURE 1, 1	Yes
32	40	Transformations	DRAWING_MAGNIFICATION 0, 0	Yes
			SCREEN_DIMENSIONS 640, 480	Yes
			WINDOW_ORIGIN 0, 0	Yes
			ZOOM_FACTOR 1, 1	Yes
			SET_VIEWPORT 0, 0, 0, 0	Yes

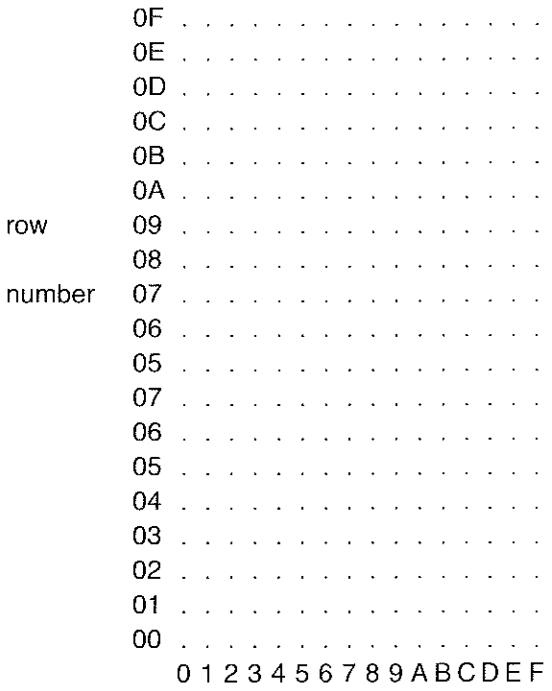
Table D-1(Cont) VIVID Attribute Mask Values

Mask Value		Attribute Group	Initialization Equivalent VIVID Instruction	Saved
Dec	Octal			
64	100	Text	SET__FONT 4223 or undefined	Yes
			CELL__OBLIQUE 0	Yes
			CELL__ROTATION 0	Yes
			CELL__SIZE 8, 10, 0, 0	Yes
			CELL__MAGNIFICATION 0, 1, 2	Yes
			CELL__MOVEMENT 0, 0	Yes
128	200	Screen Blank	SCREEN__BLANK 0	No
256	400	Blink Control	SCREEN__BLINK 0	No
			BLINK__TIMING 32, 28	No
512	1000	Inputs	SWITCH__DISABLE	No
			MATCH__DISABLE	No
			STOP__KEYBOARD__INPUT	No
1024	2000	Attribute Stack	(Clear attribute stack)	

APPENDIX E
DEFINING A CURSOR IN VIVID

The maximum size a cursor can have is the maximum cell size (16 × 16 pixels). The cell is a 16 × 16 dot matrix, where the user can define the dots to be illuminated.

In the example given here, the matrix is regarded as starting at point 0,0 in the bottom left-hand corner and being numbered in hex left to right, bottom to top, as follows:



To define the cursor, the following parameters must be specified:

- Number of cell rows
- X coordinate of cursor point
- Y coordinate of cursor point

Example:

If a cursor is to be defined as the character L and X denotes a pixel, the cursor shape might be as follows:

```
                0F X X . . . . . . . . . . . . . .
                0E X X . . . . . . . . . . . . . .
                0D X X . . . . . . . . . . . . . .
                0C X X . . . . . . . . . . . . . .
                0B X X . . . . . . . . . . . . . .
                0A X X . . . . . . . . . . . . . .
row             09 X X . . . . . . . . . . . . . .
                08 X X . . . . . . . . . . . . . .
number          07 X X . . . . . . . . . . . . . .
                06 X X . . . . . . . . . . . . . .
                05 X X . . . . . . . . . . . . . .
                07 X X . . . . . . . . . . . . . .
                06 X X . . . . . . . . . . . . . .
                05 X X . . . . . . . . . . . . . .
                04 X X . . . . . . . . . . . . . .
                03 X X . . . . . . . . . . . . . .
                02 X X . . . . . . . . . . . . . .
                01 X X X X X X X X X X X X X X X X
                00 X X X X X X X X X X X X X X X X
                   0 1 2 3 4 5 6 7 8 9 A B C D E F

                        column
                        number
```

The display list defining this cursor is as follows:

```
5A13    ! set cursor style
0010    ! 16 rows of cursor data
0000    ! define cursor point x coordinate
0000    ! define cursor point y coordinate
FFFF    ! define the bottom row 1 of cursor data
FFFF
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
```

An alternative way to create the same result is as follows:

```
5AFF    ! set cursor style data to be terminated by
        ! the END_PARAMETERS delimiter (length 255)
0000    ! define cursor point x coordinate
0000    ! define cursor point y coordinate
FFFF    ! define the bottom row 1 of cursor data
FFFF
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
0003
8000    ! END_PARAMETERS delimiter
```

GLOSSARY

attributes
Instructions which describe the appearance of parts of a graphics picture, such as color and size.

CLUT
Color Look-up Table. This table converts the four-bit pixel data, representing a color for each pixel, into intensity values of red, blue, and green.

DMA
Direct Memory Access. DMA operations are initiated by the host processor. They transfer data directly from the host memory to the VSV21 memory across the Q-bus.

display list
The list of instructions that describes the items that make up a picture. A VIVID display list is made up of segments. A VSV11 display list is made up of primitives.

display priority
The pixel memory is updated only when the screen is blank during the frame fly-back period. This eliminates flicker but reduces performance.

download
Copy code from the host system, where it is stored, across the parallel interface to the VSV21 module.

drawing priority
The pixel memory is updated whenever data is ready. This is the fastest method of displaying pictures. It may cause the screen to flicker each time an update is made.

emulation
The VSV21 can operate in one of three ways: VSV11 emulation, VIVID interpreter or replaced VT220 emulation. The interpreter or emulator software must first be downloaded from the host.

host system
The processor to which the VSV21 is attached through its Q-22 bus. Some components of the system software for the VSV21 run under control of the host operating system.

on-board software
This consists of downloaded software (including the VIVID interpreter, VSV11 emulator and pointing device drivers), and resident software (including the VSV21 driver, kernel and self-test and VT220 emulator).

pixel
A single point on the screen; the smallest location that can be addressed separately.

primitive
The fundamental VSV11 graphics instruction, for example, draw a line, or write some text.

programmed I/O
The mechanism by which the host sends commands to and receives status returns from the VSV21, through address-mapped registers on the Q-22 bus.

QIO request
A form of input/output request which uses system service routines in the operating system executive to queue requests to the device driver.

segment
Part of a VIVID display list. A segment can contain either VIVID instructions, font data, pixel data, keyboard data, report information or attribute data.

transformation
The mapping of a picture from the application's coordinate system onto the coordinates of the display surface (the video monitor screen).

VCP
VSV21 Control Program. A utility program to configure and control the VSV21 device.

VIVID
VSV21 Instructions for Visual Display. This is the VSV21 instruction set which is used in display lists.

VSL
VIVID Subroutine Library. This is a library of functions and subroutines which can be used to control and execute segments and handle replies from VIVID.

VSV11 emulation
The VSV21 emulates a minimum-configuration single-channel VSV11 graphics device.

VT220 emulation

The VSV21 emulates a VT220 terminal with a limited set of features.

word

A word is 16 bits (two bytes) of memory. This is the unit of memory normally accessed in one operation by the on-board 68000 microprocessor.

INDEX

W

Z

READER'S COMMENTS

Your comments and suggestions will help us in our efforts to improve the quality and usefulness of our publications.

1. **Which of the following most closely describes your job?** 1 ⊂a⊃ ⊂b⊃ ⊂c⊃ ⊂d⊃ ⊂e⊃ ⊂f⊃ ⊂g⊃ ⊂h⊃ ⊂i⊃

   (a) Administrative support (d) Scientist/Engineer (g) Educator/Trainer
   (b) Programmer/Analyst (e) Systems Manager (h) Computer Operator
   (c) Software support (f) Sales (i) Other _____

2. **How many years of experience do you have with computers?** 2 ⊂a⊃ ⊂b⊃ ⊂c⊃ ⊂d⊃ ⊂e⊃

   (a) Less than 1 (b) 1 to 3 (c) 4 to 6 (d) 7 to 9 (e) 10 or more

3. **What did you like *most* about this manual?**

4. **What did you like *least* about this manual?**

5. **How do you rate this manual?**

   Indicate your opinion of the quality of the manual. For each aspect of quality, darken your response on the five-point scale, where (1) = POOR and (5) = EXCELLENT

(a)	Accuracy	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(b)	Completeness	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(c)	Usefulness of Examples/Figures	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(d)	Clearness of Language	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(e)	Helpfulness of Index/Table of Contents	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(f)	Consistency in Presenting Information	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(g)	Logical Organization	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(h)	Visual Appeal	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(i)	Relevance of Information	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(j)	Ease of Learning	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(k)	Ease of Use	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(l)	YOUR OVERALL IMPRESSION	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(m)	Quality Relative to Other Digital Manuals	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃
(n)	Quality Relative to Other Companies' Manuals	⊂1⊃	⊂2⊃	⊂3⊃	⊂4⊃	⊂5⊃

6. **List any errors you found in the manual. (Reference page, table, or figure numbers.)**

7. **Do you have any additional comments?**

Name _____ Company _____

Title _____ Department _____

Street _____ City _____ State/Country _____ Zip _____

Telephone No. _____ Date _____

Site Code: RE

FOLD HERE AND TAPE

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MA.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Educational Services/Quality Assurance
12 Crosby Drive BUO/E08
Bedford, MA 01730

FOLD HERE