VSV21

digital™

VSV21
Programmer's Guide

Order Number AA-FV67D-TK

VSV21 PROGRAMMER'S GUIDE

Order Number: AA-FV67D-TK

digital equipment corporation maynard, massachusetts

First Edition, July 1985
Second Edition, March 1986
Third Edition, June 1987
Fourth Edition, August 1988

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem–10	PDP	VT
DECSYSTEM–20	PDT	digital™
DECUS	RSTS	
DECwriter	RSX	

This document was prepared using VAX DOCUMENT, Version 1.0

63454

Contents

Preface

Document Structure

This manual is divided into five parts, as follows:

1 Overview of VSV21 System Software (Chapter 1)

2 Overview of VIVID and Display Lists (Chapters 2 and 3)

3 High Level Interface - VSL (Chapters 4 and 5)

4 Low Level Interface - VIVID Instruction Set (Chapters 6 to 16)

5 VSV11 and FORTRAN Draw (Chapters 17 to 19)

The manual should be read in conjunction with the *VSV21 User's Guide.*

VSL is the high level interface for producing VIVID Display Lists.

Use of the VIVID Instruction Set directly represents the low level interface for generating display lists.

Intended Audience

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.

Associated Documents

The following list shows the related documents for the *VSV21 Programmer's Guide*:

- *VSV21 Programmer's Reference Card* AV-FV68D-TK
- *VSV21 Installation Manual* AZ-FV71D-TK
- *VSV21 User's Guide* AZ-FV70D-TK
- *VSV21 Peripheral Concentrator User's/Installation Guide*, EK-VSV21-UM
- *VSV21 Pocket Service Guide* EK-VSV21-PS
- *VR241-A Video Monitor Installation Manual* EK-VR241-IN

Part I Overview of VSV21 System Software

1 INTRODUCTION

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:

- MicroVAX, running MicroVMS Version 4.2 or later
- MicroVAX II, running VMS Version 5.0 or later
- MicroPDP-11, running:
  - RSX-11M-PLUS Version 3.0 or later
  - Micro/RSX Version 3.0 or later

The VSV21 can run any one of three processes.

1. VIVID Interpreter

   VIVID (VSV21 Instructions for Visual Display) is the VSV21 instruction set. It allows the VSV21 to display images defined using the VIVID instruction set.

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

2. 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.

   VS11/VSV11 emulation is referred to as VSV11 emulation in the rest of this guide.

Note: 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.

3. 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. See the *VSV21 User's Guide* 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 4. 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

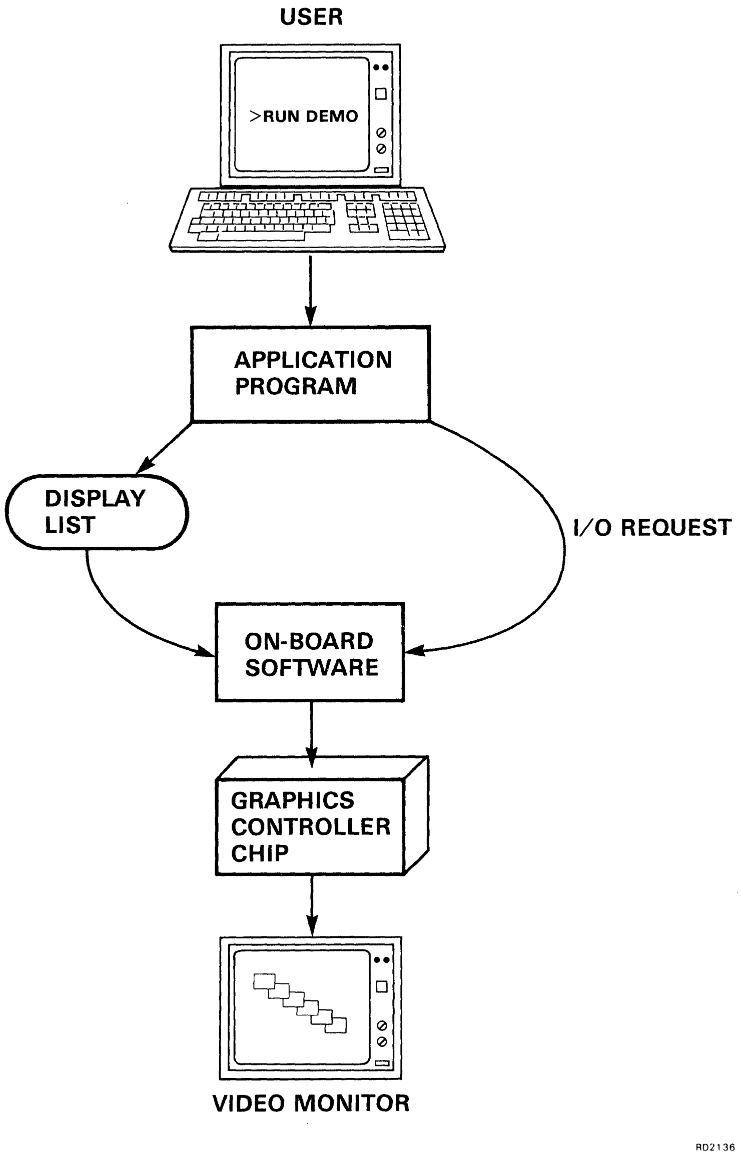
The list of instructions and data is known as a display list. The VSV21 supports two different types of display list:

1. VIVID display lists, defined by the VIVID command set and the VIVID Subroutine Library (VSL). They are described in Chapter 3.
2. VS11/VSV11 display lists, defined by the VSV11 instruction set and the FORTRAN Draw library. They are described in Chapter 18.

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 lists can also be stored in 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.

Figure 1–1 How Pictures are Created and Displayed



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.

The high-level method is recommended for those users new to graphics processing and/or software techniques, and require an easy-to-use mode of operation to get started.

VSL is described in Chapter 4. The method of using the FORTRAN Draw package is described in Chapter 17.

- 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 optimum performance 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, both high and low level) and Chapter 18 (VSV11). Processing display lists and programming the VSV21 are described in Chapter 4 (VIVID high level - VSL), Chapter 6 (VIVID low level - QIOs) and Chapter 19 (VSV11 QIOs). Guides to getting started with VIVID are in Chapter 5 (high level VSL) and Chapter 6 (low level QIOs)

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

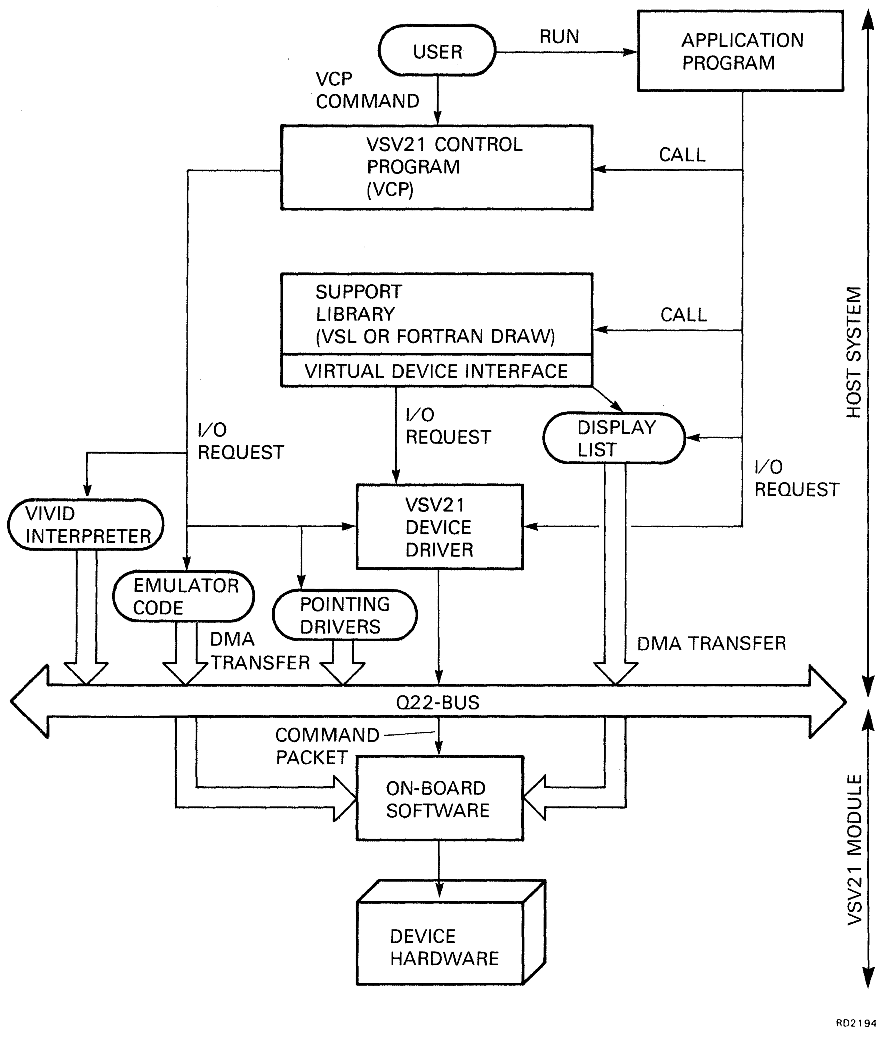
Most of the VSV21 system software is supplied on a distribution kit and must be installed on the host system before any applications can be run. The procedure for performing and verifying the software installation is described in the *VSV21 Installation Manual.*

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

- Host software
  - VSV21 device driver
  - VSV21 Control Program (VSVCP)
  - 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
- VSVCP (VSV21 Control Program)
- Subroutine libraries
- Diagnostics

1.4.1.1 VSV21 Device Driver

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 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 (VSVCP)

The VSV21 Control Program (VSVCP) 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 VSV11 Emulation 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 VSVCP commands enable you to configure the VSV21 system, set graphics attributes, and 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 VSVCP commands are described in the *VSV21 User's Guide.*

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.

1.4.1.4 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.

1.4.2 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 VSVCP. 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) X 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 VSV21 memory, it can be downloaded separately by using either a QIO load segment (Chapter 6) or the VSVCP (*VSV21 User's Guide*).

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 VSVCP can download software with individual commands or a command procedure. For a description of the method of downloading software and fonts, refer to the *VSV21 User's Guide*.

1.5 Constraints When Using VSL Calls

User's programs are constrained by the number of VAX memory mapping registers available. This constraint is more noticeable in VMS V4 than in VMS V5. This is due to the available allocation of mapping registers in the two versions which are as follows:

- VMS V4 = 496
- VMS V5 = 7696

These are not all available to the user because each device driver on the system will probably use some of the allocation. Thus:

$$Remaining\ Mapping\ Registers = Max.\ Available - system\ usage.$$

Each VSV21 on a system will need mapping registers of its own, and must share, not necessarily equally, the Remaining Mapping Registers, calculated above.
One mapping register is needed for each ½K bytes of memory.

Under VMS V4 the VV driver imposes a second limitation of 255 mapping registers per device. This means that the maximum amount of memory which can allocated to each device is 127K bytes.

Under VMS V5 there are 3848K bytes of memory available for sharing among the VSV21s.

Part II Overview of VIVID and Display Lists

2 OVERVIEW OF VSV21 VIVID PROGRAMMING

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 VMS/MicroVMS, RSX-11M-PLUS, or Micro/RSX. The programs make calls to stored graphical information (display lists, described in Chapter 3) and library routines (VSL, described in Chapter 4).

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

The VSV21 can be programmed, using the VIVID instruction set, at low level (QIOs) or high level (VSL) to perform the following range of tasks:

- Control 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

Both VSL (the high level VIVID subroutine library) and the low level VIVID QIO Interface provide access to the VIVID instruction set to generate display list graphics instructions for output to the VSV21. The VSL routines provide an easy-to-use method of constructing VIVID display lists but remember that both the high level and low level approaches ultimately provide the VIVID interpreter with appropriate display lists of instructions for output through the graphics chip.

Note that for most VSL library routines there is an equivalent VIVID instruction, for example, VVVZMF equivalent to ZOOM_FACTOR.

The VIVID instruction set consists of the following types:

- Control instructions
  These initialize the VIVID interpreter, begin and end display list segments and control the general operation of VIVID (Chapter 8).
- Transformation instructions
  These control the magnification of the display and the position of the window and viewport (Chapter 9).
- Global Attribute instructions
  These set the drawing and screen display characteristics (Chapter 10).
- Drawing instructions
  These generate the individual lines that make up an image (Chapter 11).
- Filled Figure instructions
  These are used to paint or flood specified areas (Chapter 12).
- Text instructions
  These control the selection of character fonts, the magnification of characters and output of text to the screen (Chapter 13).
- Area Operation instructions
  These control such operations as scroll, clear screen, copy and pixel read (Chapter 14).
- Interactive instructions
  These control cursor positioning and keyboard operation (Chapter 15).
- Report Handling instructions
  These place a report packet in the current report segment (Chapter 16).

The chapters referenced in this section describe the use of each instruction and its parameters at VIVID interpreter level and provide a Macro-32 example.

A description of the parameters for each VSL routine is in Chapter 4.

A brief description of each VIVID instruction is given in this chapter.

2.3 ACCESS TO VIVID INSTRUCTIONS

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

- As VSL routines, called directly by the program, to produce valid VIVID display lists.
- As user-defined display lists, for direct use with Queue Input/Output (QIO) instructions.

2.3.1 Display Lists

A display, list or segment, is a list of VIVID graphics output instructions and data.

Each segment is a list of VIVID instructions and data that have 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 6.

2.3.2 VIVID Subroutine Library (VSL)

VSL is a library of functions and subroutines (see Chapter 4). 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 VIVID 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 items 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 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 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 co-ordinates 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.

Note: **8000 Hex is not a valid X,Y co-ordinate. If used it can cause unpredictable results.**

- 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

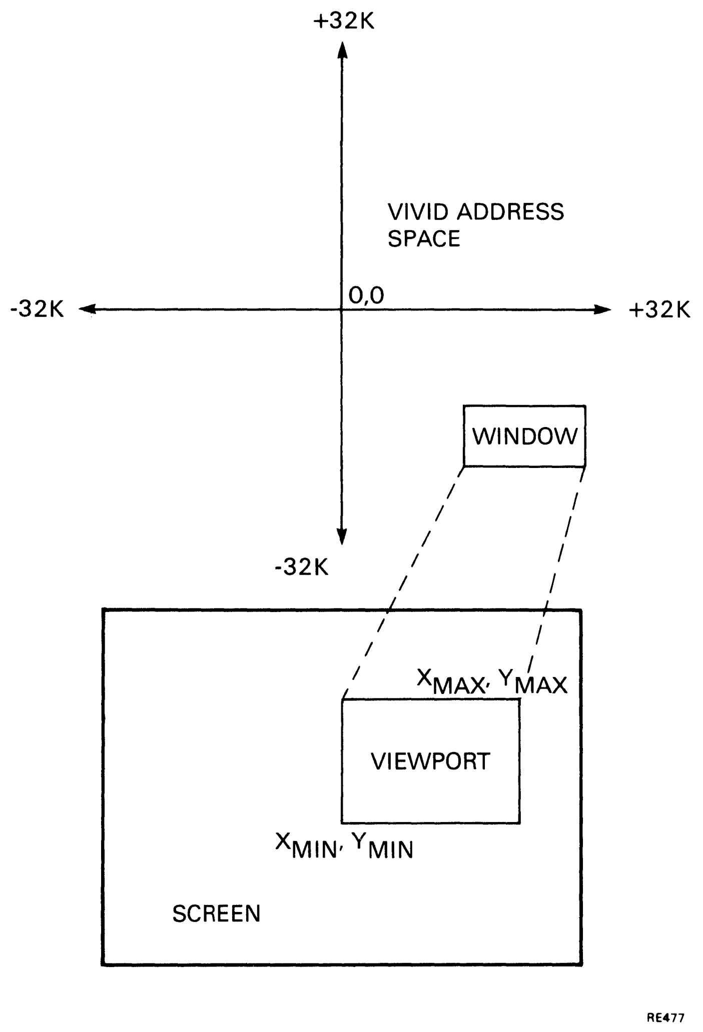
  The VIVID window is the area of VAS which will be mapped to the viewport. You can set the origin (lower left corner) of the window. The extent of the window can be input as a parameter or determined by the screen dimensions.

- 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.

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



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 that require transformation before display and enables the magnification and translation factors.

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 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 the *VSV21 User's Guide*.

- 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:

1 Absolute

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

2 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 and MOVE_REL

  Moves to the specified position. Nothing is drawn.

- MOVE_TO_CURSOR

  Moves the current drawing position to the cursor position.

- LINES_ABS and LINES_REL

  Draws lines from the current position to specified points.

- POLYMARKS_ABS and POLYMARKS_REL

  Draws the specified marker character at the specified points.

- ARCS_ABS and ARCS_REL

  Draws the specified sequence of circular arcs.

- ELLIPSE_ARCS_ABS and ELLIPSE_ARCS_REL

  Draws the specified sequence of elliptical arcs.

- RECTANGLE_ABS and 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 that is filled by a pattern. The instruction used to fill the area determines the boundary conditions.

- FILLED_RECT_ABS and 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 that 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 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 and 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.

- 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 that 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 mailbox on VMS/MicroVMS systems, and to the AST on RSX-11M-PLUS and Micro/RSX systems. The input is echoed to the screen.

- STOP_KEYBOARD_INPUT

  Disables keyboard input for asynchronous processing.

2.12 REPORT HANDLING INSTRUCTION

Reports are information packets that are generated during execution of VIVID display lists.

These reports are either generated automatically by events that occur during display list execution, or can be specifically requested by the application program.

Reports are queued into a report segment, which is created by the application program.

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 VSV21 or host memory
- VIVID version number
- Nested segment calls to 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.

3 DISPLAY LISTS

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

A segment is a list of VIVID instructions and data that has 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:

1 Instruction segment, consisting of VIVID instructions, stored as opcodes and parameters.

2 Font segment, consisting of a set of character cells.

3 Pixel data segment, consisting of a pixel data map.

4 Keyboard segment, consisting of data input from the keyboard.

5 Report segment, consisting of report packets.

6 Attribute segment, consisting of global attributes data.

The segment types are described in Section 3.3.

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

- Use a QIO call to communicate directly with the VIVID interpreter. The VIVID I/O functions are described in Chapter 6.
- Call functions from the VIVID Subroutine Library (VSL) described in Chapter 4.

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

3.1 IDENTIFYING SEGMENTS

A display list or Segment, is a list of VIVID graphics output instructions and data.

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:

1 ID number - segments stored on the host are given an ID number as a parameter within the QIO call.

2 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:

1 class

2 number within class

This structure allows you to probe 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 by Report 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 Host Memory

VIVID allows storage of up 512 segments in 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 VSV21 Memory

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

1 Issuing QIOs (Chapter 6).

2 Calling VSL functions (Chapter 4).

3 Using VSVCP as described in the *VSV21 User's Guide*.

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

It is beneficial to download the following segments:

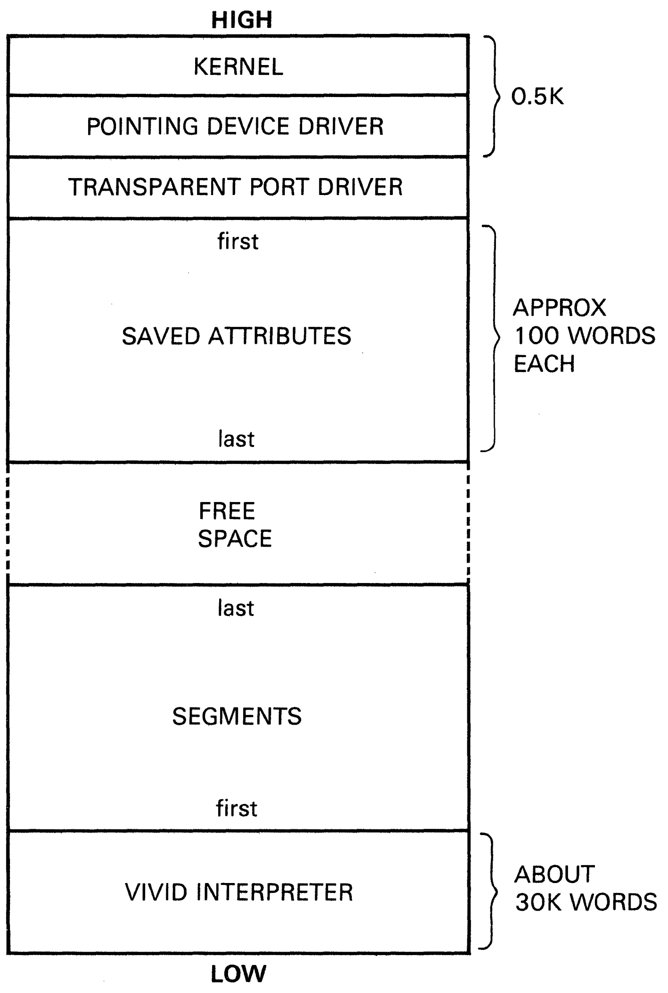
- Commonly used display lists

  For example, those defining icons or symbols repeated at many places in a picture.
- Display lists that are unlikely to change
- Font segments where frequent and fast access is required

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, through the VSVCP, is given in the *VSV21 User's Guide*.

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 a 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 6.

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

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

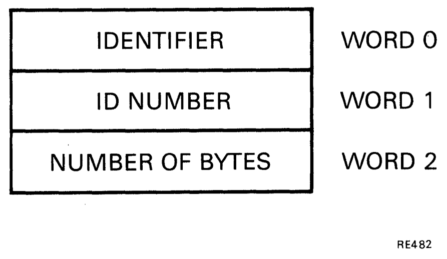
3.2.4 The VIVID Default Font

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 in hexadecimal. 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).

Note: Using the VSL high-level interface, a call to VVBBGN (Start Segment) automatically sets up a Start_Instruction_List instruction in the specified user-created segment.

When using VIVID, it is necessary to use a VIVID instruction to specifically initialize a segment which is to be used for font, pixel, keyboard, report, or attribute data types.

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 in any other way for transmission to the VSV21.

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 in hexadecimal) is stored in VSV21 memory automatically when the VIVID interpreter is downloaded. The segment that defines the default font is listed in Appendix B.

Note: VSL equivalent of Start_Font is the routine VVTIFT.

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
								*

Note: VSL equivalent of Pixel_Readback is the routine VVAPXR.

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.

Note: VSL equivalent of Accept_Keyboard_Input is the routine VVIAKI.

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 Execution 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:

1. Define a new segment as a reporting segment as follows:
   - Define a segment using the Define Segment QIO
   - Define the segment as a report segment and set up the report mask using the Define Reporting QIO
   - Initialize the segment by issuing a Start Segment Execution call without any reporting parameters
2. Define a new segment as a reporting segment as follows:
   - Define a segment using the Define Segment QIO
   - Define the segment as a reporting segment and initialize it by issuing a Start Segment Execution QIO with the optional reporting parameters
3. 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.

Note: Using the VSL high-level interface, the Start_Report_Data instruction is automatically set-up when a report segment is defined.

There are two ways of defining a report segment using VSL:

1. **When calling VVXASS, the routine that assigns a channel to the VSV21 device, supply the third parameter. This is an integer value representing the size of the required report segment in bytes. The report segment is then automatically created and given a segment ID of HEX 2001.**
2. **Using VVMCRS, create an additional segment of user-defined size and ID to be the report segment. Then, when calling VVEEXE, the segment execute routine, supply as the fourth parameter the created segment ID. A Start_ Report_Data instruction is then automatically setup in the created segment and any reports generated during that display list execution will then be stored in the report segment.**

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.

Note: VSL equivalent of DUMP_ATTRIBUTES is the routine VVCDMP.

Part III High Level Interface - VSL

This section describes how to develop graphics applications with the VSV21, and build pictures using the VIVID Subroutine Library (VSL) with a high-level language such as FORTRAN.

4 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.

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 (VVMaaa), used to:
  - Initialize segment building
  - Save and restore segments to and from disk
  - Load segments to the VSV21 from disk and from 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 host memory with the VVXINI function (see VVXINI in Section 4.1). 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.

VMS SYSTEM SPECIFICS

The display area is a 128Kbyte FORTRAN array (VV21RG) in VMS/MicroVMS. Access to the section is thus by array element, each element being one word in the array. This is transparent to VSL users - there is no need to explicitly declare access to the data section at link time.

All input parameters passed in VSL calls should be

$$INTEGER * 2$$

unless otherwise stated. Under VMS, all VSL functions return an

$$INTEGER * 4$$

value. These reply values are additive; if two different errors are detected, the reply is the sum of the error codes.

The return status word ISTAT will work as either

$$INTEGER * 2$$

or

$$INTEGER * 4$$

with one exception - VVRREP (See Section 4.4).

Users should insert an IMPLICIT statement at the top of the application program on VMS to force the status returns from the VSL routines to be

$$INTEGER * 4$$

For example :

$$IMPLICIT\ INTEGER * 4(V)$$

Note: All VSL routines start with the letter "V".

Also, the user should use the /NOOP switch when compiling application programs under VMS. This is to facilitate the use of default parameters with most of the VSL utility routines.

Under VMS, the subroutine library can be accessed from high level languages other than FORTRAN, such as C, PASCAL, BASIC etc. The important stipulation to remember when writing such applications is that all calls to the VSL functions must pass their parameters by reference rather than by value. This is a standard interface within FORTRAN programs.

Again, it is advised to use the "NO OPTIMISE" switch when compiling.

RSX SYSTEM SPECIFICS

The display area is a region in RSX-11M-PLUS and Micro/RSX systems. Access to the region is by means of a window that 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: VV21DA

Window size: 4K words

The Call Formats to the routines are for FORTRAN. All parameters are

$$INTEGER * 2$$

unless otherwise stated. All the VSL functions return a status value ISTAT as an

$$INTEGER * 2$$

Reply values are additive; if two different errors are detected, the reply is the sum of the error codes.

4.1 GENERAL FUNCTIONS

In the following routine descriptions, all parameters other than the status return ISTAT are input parameters; that is, values supplied by the caller, unless otherwise stated.

Where a parameter LUN is stated, this is a Logical Unit Number supplied by the caller for subsequent use in addressing the VSV21 device. The range of the LUN is from 1 to 16.

Assign VSV21 Device - VVXASS

Assign and attach a VSV21 device to VSL processing.

CALL FORMAT: ***istat = VVXASS (dev, lun [, rslen [, clarr]])***

PARAMETERS:

dev

VSV21 device name
This parameter should be passed as the device physical unit number

For example:

VMS
- 0 for VVA0:
- 1 for VVB0:
- 2 for VVC0: and so on

RSX
- 0 for VS0:
- 1 for VS1:
- 2 for VS2: and so on

lun

device logical unit number

rslen

length of report segment. This parameter has no default assumption.

clarr

array containing host classes accessible to unit

REPLY VALUE:

0

not initialized

1

completed successfully

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

4096
area not allocated

NOTES:

You must assign to the VSV21 device before using the device.

You can assign a maximum of eight under VMS/MicroVMS, and a maximum of four devices under RSX-11M-PLUS and Micro/RSX. The device is only attached on RSX systems. 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:

VMS/MicroVMS = 8 devices

RSX-11M-PLUS = 4 devices

End Display Processing - VVXEND

Release the VSV21 processor and free the VSV21 buffers. To restart processing, you must call the VVXINI and VVXASS functions.

CALL FORMAT: ***istat = VVXEND ()***

PARAMETERS: ***None***

REPLY VALUE: ***0***

not initialised

1

completed successfully

Initialize Display Processing - VVXINI

Set up the display area and initialize VIVID processing.

CALL FORMAT: ***istat = VVXINI ([dlen, [maxno]])***

PARAMETERS:

dlen
size of VSL display area in bytes
0 : default of 64K bytes

Under VMS this parameter should be defined as an

$$INTEGER * 4$$

Under RSX this parameter should be a two-word area

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)

Release VSV21 Device - VVXREL

Release 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

8

failed to release

NOTES:

To detach all the devices when display processing is complete, use VVXEND

Get VIVID Version Number - VVXVER

Get 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

4.2 SEGMENT MANIPULATION FUNCTIONS

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

Copy Segment - VVMCPY

Copy a segment to the VSL display area.

The segments must already have 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:

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, no segment is created or updated.

The segment ID and length are derived from the array.

Create Segment - VVMCRS

Create 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.

Delete Segment - VVMDEL

Delete a segment on 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:

0
not initialized

1
completed successfully

4
segment not found

16
device not assigned

NOTES:

The segment ID format is described in Section 3.1.

Load Segments from File - VVMDLD

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

- Instruction
- Font
- Pixel data

CALL FORMAT: ***istat = VVMDLD (lun, fun, filn)***

PARAMETERS:

lun
device logical unit number
-1 : all units for which segment class is valid

fun
FORTRAN unit number

filn
name of disk file

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.

Restore Segments from Disk - VVMGET

Read segments to 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 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.

Segment files opened by this function have shared access under VMS only.

Load Segment from Host - VVMMLD

Load to the VSV21 a segment that 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:

0
not initialized

1
completed successfully

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 3.1.

Save Segments on Disk - VVMSAV

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

CALL FORMAT: ***istat = VVMSAV (fn, filn, idarr, nseg)***

PARAMETERS:

fn
file unit number

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

4.3 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.

Note: **It is important to remember that whilst a segment is being executed, write access is disabled to that segment. Any attempt to insert new instructions into an executing segment, or to delete an executing segment, could result in data corruption and subsequent impaired graphics output.**

Execute Segment - VVEEXE

Initiate 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 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 automatically actioned first.

Resume Segment Execution - VVERES

Resume 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

rsegid
reporting segment ID

NOTES:

If the time out value is zero, or omitted, a default of 5 seconds is used. If pointing device activity 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.

Stop Segment Execution - VVESTP

Stop 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.

4.4 REPORTING FUNCTIONS

Get Keyboard Input - VVRKBD

Get 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
integer array to contain packed ASCII characters (output)

alen
size of array in bytes

dlen
length of data in bytes (output)

segst
segment status (output)

REPLY VALUE:

0
not initialized

1
completed successfully

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).

Get Report - VVRREP

Get a report from the report segment for the indicated logical unit number. The report may be of a specified type, or of any type. The parameter ARR is a user-defined array, typically of about 20 words (INTEGER*2), in which VSL writes any reports of type RTYPE that are in the report segment. Reports are written to the array one at a time. To read a number of reports, further calls to VVRREP should be made.

The report segment is always read from the beginning.

When searching for a report packet in the report segment, any previously read report packets are discarded. Thus, for example, if the report segment contains a number of drawing position report packets, every time the report segment is read for a packet of that type, the next one in the list will be extracted.

CALL FORMAT: *istat = VVRREP (lun, rtype, arr, alen [,rsegid])*

Under VMS, the return status value istat MUST be defined as an

$$INTEGER * 4$$

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 16 for details of report requests.

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

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

Get Segment Block - VVRSEG

Get a block of data from a segment in the VSL display area. This is specifically intended 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:

0
not initialized

1
completed successfully

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.

Get Status - VVRSTA

Provide the status of the preceding 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 report count is the total number of reports issued for a previous call invoking display list processing. See VVEEXE and VVERES (both in Section 4.3) for further details.

The report formats may be found in Chapter 16.

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

4.5 SEGMENT BUILDING FUNCTIONS

Start Segment - VVBBGN

Start 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: ***0***
not initialized

1
success

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 is implied for that segment.

End Segment - VVBEND

End 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:

0
not initialized

1
completed successfully

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.

Set Drawing Parameter Mode - VVBMOD

Set the mode required for subsequent VIVID Instruction Generation call with ABS, REL variations. It is important to set the correct mode before generating any drawing instruction into a segment.

CALL FORMAT: ***istat = VVBMOD (dmode)***

PARAMETERS: ***dmode***

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

REPLY VALUE: ***0***

Invalid parameter

1

Completed successfully

Set Instruction Parameter Mode - VVBPMD

Set 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 4.6.

CALL FORMAT: ***istat = VVBPMD (pmode)***

PARAMETERS: ***pmode***

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

REPLY VALUE: ***0***

Invalid parameter

1

Completed successfully

4.6 INSTRUCTION GENERATION FUNCTIONS

Each VSL call within this section generates an instruction from the VIVID instruction set. VSL instruction calls generate VIVID instruction opcodes into the pre-defined segment together with the appropriate VIVID parameters. The parameters used by VSL will always reflect those required by the specific VIVID calls, plus some extra parameters required for the high level interface. These are documented in the relevant VSL function description.

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

1 Parameter list mode

2 Array list mode

In parameter list mode (VVBPMD in Section 4.5) 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 (VVBPMD in Section 4.5) an alternative Call Format is used for some functions - as noted in the relevant function descriptions in the following sections. 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 (that is, word count). 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 4.5) encountered.

The function call should occur between VVBBGN and VVBEND calls (both in Section 4.5). 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 also available when the VVBEND instruction is executed and the segment is completed.

The VIVID instruction generation VSL routines are specified in detail below. All the referenced examples can be found in the next chapter, "Getting Started With VSL".

Note: "Type Integer" should be a 16-bit value (FORTRAN INTEGER*2) unless otherwise stated.

4.6.1 Control Functions

VVCCAL

Call is made to execute the identified segment from either host or VSV21 memory.

ROUTINE NAME: ***VVCCAL - Call Segment***

CALL FORMAT: ***istat = VVCCAL (segid)***

PARAMETERS: ***segid (type integer)***
Segment ID

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Control Instructions Chapter 8, CALL_SEGMENT.

EXAMPLE OF USE Section 5.5.4, Program TRANSF.FOR

VVCCRS

Call is made to create a segment in VSV21 memory.

ROUTINE NAME:	***VVCCRS - Create Segment***
CALL FORMAT:	***istat = VVCCRS (segid, slen)***
PARAMETERS:	***segid (type integer)*** Segment ID. ***slen (type integer)*** total segment size in bytes.
REPLY VALUE:	***0*** success ***1*** not initialized ***2*** segment overflow
VIVID CROSS REF	Control Instructions Chapter 8, CREATE_SEGMENT.
EXAMPLE OF USE	Section 5.5.7, Program THINGS.FOR

VVCDMP

Call is made to save the current set of attributes in a specified host segment. This segment must already have been created via a call to VVMCRS.

ROUTINE NAME: ***VVCDMP - Dump Attributes***

CALL FORMAT: ***istat = VVCDMP (segid)***

PARAMETERS: ***segid (type integer)***
Segment ID

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Control Instructions Chapter 8, DUMP_ATTRIBUTES.

EXAMPLE OF USE Section 5.5.7, Program THINGS.FOR

VVCDWT

Call is made to wait for a specified time before executing the next display instruction.

ROUTINE NAME:	***VVCDWT - Display Wait***
CALL FORMAT:	***istat = VVCDWT (nfram)***
PARAMETERS:	***nfram (type integer)*** number of video frames delay required. There are 60 frames per second.
REPLY VALUE:	***0*** success ***1*** not initialized ***2*** segment overflow
VIVID CROSS REF	Control Instructions Chapter 8, DISPLAY_WAIT.
EXAMPLE OF USE	Section 5.5.4, Program TRANSF.FOR

VVCERP

Call is made to mark the end of a repeatable loop.

ROUTINE NAME: ***VVCERP - Display End Repeat***

CALL FORMAT: ***istat = VVCREP ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Control Instructions Chapter 8, DISPLAY_END_REPEAT.

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVCINI

Call is made to restore the downloaded (VIVID original) status of one or more graphics-controlled facets.

ROUTINE NAME: ***VVCINI - Initialize***

CALL FORMAT: ***istat = VVCINI (mask)***

PARAMETERS: ***mask (type integer)***

= sum of values indicating requirements.

N.B. -1:all values.

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Control Instructions Chapter 8, INITIALIZE.

EXAMPLE OF USE

Section 5.5.1, Program DRAW.FOR

VVCJMP

Call is made to add the specified number of words to the display list pointer.

ROUTINE NAME: ***VVCJMP - Jump Relative***

CALL FORMAT: ***istat = VVCJMP (nwords)***

PARAMETERS: ***nwords (type integer)***
number of words.

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Control Instructions Chapter 8, JUMP_RELATIVE.

EXAMPLE OF USE Section 5.5.8, Program MATCH.FOR

VVCRCV

Call is made to read the specified attributes from the specified segment.

ROUTINE NAME: ***VVCRCV - Recover Attributes***

CALL FORMAT: ***istat = VVCRCV (segid, mask)***

PARAMETERS:

segid (type integer)
Segment ID

mask (type integer)
bit-mask value defining attributes to be recovered.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Control Instructions Chapter 8, RECOVER_ATTRIBUTES.

EXAMPLE OF USE :

Section 5.5.7, Program THINGS.FOR

VVCREP

Call is made to mark the start of a loop in display list processing.

ROUTINE NAME: ***VVCREP - Display Repeat***

CALL FORMAT: ***istat = VVCREP (nloop)***

PARAMETERS: ***nloop (type integer)***
number of times the loop is to be repeated.
N.B. 0:loop is repeated infinitely.

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Control Instructions Chapter 8, DISPLAY_REPEAT.

EXAMPLE OF USE Section 5.5.4, Program TRANSF.FOR

VVCRES

Call is made to remove the last attributes saved by VVCSAV from the stack and set up as current attributes. The previous attributes are lost.

ROUTINE NAME: *VVCRES - Restore Attributes*

CALL FORMAT: *istat = VVCRES (mask)*

PARAMETERS: *mask (type integer)*

sum of values indicating requirements.

N.B. -1:all values are restored.

See Appendix D for values.

REPLY VALUE: *0*

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Control Instructions Chapter 8, RESTORE_ATTRIBUTES.

EXAMPLE OF USE

Section 5.5.6, Program AREA.FOR

VVCSAV

Call is made to add the current attributes to an attribute stack. This allows you to change attributes in a nested segment and to recover attributes before returning to the calling segment.

ROUTINE NAME: ***VVCSAV - Save Attributes***

CALL FORMAT: ***istat = VVCSAV ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Control Instructions Chapter 8, SAVE_ATTRIBUTES.

EXAMPLE OF USE

Section 5.5.6, Program AREA.FOR

VVCSTP

Call is made to stop display list processing. Control is returned to the application program with a status value, that is, as if execution had completed.

ROUTINE NAME: ***VVCSTP - Stop Display***

CALL FORMAT: ***istat = VVCSTP ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF Control Instructions Chapter 8, STOP_DISPLAY.

EXAMPLE OF USE Section 5.5.7, Program THINGS.FOR

4.6.2 Transformation Functions

VVVDIM

Call is made to define the screen dimensions in logical pixels, for example, 640,480; 640,240; 512,512.

ROUTINE NAME: ***VVVDIM - Screen Dimensions***

CALL FORMAT: ***istat = VVVDIM (width, height)***

PARAMETERS:

width (type integer)
width of display in logical pixels.

height (type integer)
height of display in logical pixels.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - SCREEN_DIMENSIONS

EXAMPLE OF USE

Section 5.5.2, Program PAINTS.FOR

VVVDRM

Call is made to define the magnification of the drawing elements being entered to VAS. This applies to both absolute and relative drawing operations.

ROUTINE NAME: ***VVVDRM - Drawing Magnification***

CALL FORMAT: ***istat = VVVDRM (xmag, ymag)***

PARAMETERS:

xmag (type integer)
magnification along X axis.

ymag (type integer)
magnification along Y axis.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - MAGNIFICATION_FACTOR

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVVDRT

Call is made to define the coordinates by which the transformation origin is shifted relative to the previous transformation origin.

ROUTINE NAME: ***VVVDRT - Drawing Translation***

CALL FORMAT: ***istat = VVVDRT (x, y)***

PARAMETERS:

x (type integer)
X coordinate of translation.

y (type integer)
Y coordinate of translation.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - DRAWING_TRANSLATION

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVVSVU

Call is made to define a screen area to which drawing is restricted. The area units are as defined by the Screen Dimensions instruction (VVVDIM).

ROUTINE NAME: ***VVVSVU - Set Viewport***

CALL FORMAT: ***istat = VVVSVU ([xmin, ymin, width, height])***

PARAMETERS:

xmin (type integer)
X coordinate of lower left corner.

ymin (type integer)
Y coordinate of lower left corner.

width (type integer)
width of viewport in logical pixels.

height (type integer)
height of viewport in logical pixels.

N.B. No parameters supplied, or all parameters zero will set the viewport to the boundaries of the screen.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - SET_VIEWPORT

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVVTRN

Call is made to enable the Drawing Magnification (VVVDRM) and Drawing Translation (VVVDRT) instructions. The instruction can be used with the Drawing VAS (VVVVAS) instruction to turn the transformations on or off as required.

ROUTINE NAME: ***VVVTRN - Drawing Transform***

CALL FORMAT: ***istat = VVVTRN ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - DRAWING_TRANSFORM

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVVVAS

Call is made to disable Drawing Magnification and Drawing Translation. Subsequent input is in VAS units.

ROUTINE NAME: ***VVVVAS - Drawing VAS***

CALL FORMAT: ***istat = VVVVAS ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - DRAWING_VAS

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVVWND

Call is made to define a window in VIVID Address Space. The window is mapped automatically to the viewport.

ROUTINE NAME: ***VVVWND - Set Window***

CALL FORMAT: ***istat = VVVWND (xw, yw, width, height)***

PARAMETERS:

xw (type integer)
X coordinate of lower left corner of window in VAS.

yw (type integer)
Y coordinate of lower left corner of window in VAS.

width (type integer)
width of window in VAS.

height (type integer)
height of window in VAS.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - SET_WINDOW

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVVWOR

Call is made to set the window origin to a VAS position. This defines a window which may be projected into the VSV21 viewport.

ROUTINE NAME:

VVVWOR - Window Origin

CALL FORMAT:

istat = VVVWOR (x, y)

PARAMETERS:

x (type integer)
X coordinate of the window origin in VAS.

y (type integer)
Y coordinate of the window origin in VAS.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - WINDOW_ORIGIN

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVVZMF

Call is made to define the horizontal and vertical magnification factors for the zoom facility. It defines the mapping between the window and the viewport.

ROUTINE NAME: ***VVVZMF - Zoom Factor***

CALL FORMAT: ***istat = VVVZMF (xmag, ymag)***

PARAMETERS:

xmag (type integer)
X direction magnification factor.

ymag (type integer)
Y direction magnification factor.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Transformation Instructions - ZOOM_FACTOR

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

4.6.3 Global Attribute Functions

VVGATX

Call is made to define a cell containing the area texture pattern.

ROUTINE NAME: ***VVGATX - Area Texture***

CALL FORMAT: ***istat = VVGATX (bit, patt1 [,patt2,...pattn])***
where n = 3-16)

PARAMETERS: ***nbit (type integer)***
number of bits in bit pattern

pattn (type integer)
bit pattern of row n
0:background
1:foreground

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Global Attribute Instructions - AREA_TEXTURE

EXAMPLE OF USE Section 5.5.6, Program AREA.FOR

VVGBCL

Call is made to set the background color to be used for subsequent drawing.

ROUTINE NAME:	***VVGBCL - Background Color***
CALL FORMAT:	***istat = VVGBCL (ind)***
PARAMETERS:	***ind (type integer)*** color index number in color look-up table.
REPLY VALUE:	***0*** success ***1*** not initialized ***2*** segment overflow
VIVID CROSS REF	Global Attribute Instructions - BACKGROUND_COLOR
EXAMPLE OF USE	Section 5.5.2, Program PAINTS.FOR

VVGBCT

Call is made to define the number of colors that blink when blink is enabled.

ROUTINE NAME: ***VVGBCT - Blink Count***

CALL FORMAT: ***istat = VVGBCT (ncol)***

PARAMETERS: ***ncol (type integer)***
number of colors to blink

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF
Global Attribute Instructions BLINK_COUNT

EXAMPLE OF USE
Section 5.5.7, Program THINGS.FOR

VVGBLC

Call is made to define CLUT colors (normal colors) and alternate colors (blink colors) for blinking.

ROUTINE NAME: ***VVGBLC - Blink Colors***

CALL FORMAT: ***istat = VVGBLC (bind1, ind1, int1, [bind2, ind2, int2 ... bindn, indn, intn])***

where n = 3-16
N.B. Array Mode accepted.

PARAMETERS:

bindn (type integer)
color index in blink colors look-up table (BCLUT)

indn (type integer)
corresponding CLUT index

intn (type integer)
red, green and blue intensity code (0-15)

Minimum parameters 3, maximum 48.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Global Attribute Instructions - BLINK_COLORS

EXAMPLE OF USE

Section 5.5.7, Program THINGS.FOR

VVGBLK

Call is made to enable/disable screen blinking.

ROUTINE NAME: ***VVGBLK - Screen Blink***

CALL FORMAT: ***istat = VVGBLK (bmod)***

PARAMETERS: ***bmod (type integer)***
blink mode (on/off)
0:screen blink off
non-zero:screen blink on

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Global Attribute Instructions - SCREEN_BLINK

EXAMPLE OF USE Section 5.5.7, Program THINGS.FOR

VVGBLT

Call is made to set screen blink timings.

ROUTINE NAME: ***VVGBLT - Blink Timing***

CALL FORMAT: ***istat = VVGBLT (norm, blnk)***

PARAMETERS:

norm (type integer)
number of frames of normal colors (range 8-64)

blnk (type integer)
number of frames of blink colors (range 8-64)

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Global Attribute Instructions - BLINK_TIMING

EXAMPLE OF USE

Section 5.5.7, Program THINGS.FOR

VVGFCL

Call is made to set the foreground color to be used for subsequent drawing.

ROUTINE NAME: ***VVGFCL - Foreground Color***

CALL FORMAT: ***istat = VVGFCL (ind)***

PARAMETERS: ***ind (type integer)***

color index number in color look-up table.

REPLY VALUE: ***0***

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Global Attribute Instructions - FOREGROUND_COLOR

EXAMPLE OF USE

Section 5.5.1, Program DRAW.FOR

VVGLTX

Call is made to define the line texture. This is a bit pattern that is repeated in the drawn lines.

ROUTINE NAME: ***VVGLTX - Line Texture***

CALL FORMAT: ***istat = VVGLTX (nbit, fbcod)***

PARAMETERS:

nbit (type integer)
number of bits in bit pattern

fbcod (type integer)
bit pattern for foreground/background colors

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Global Attribute Instructions - LINE_TEXTURE

EXAMPLE OF USE

Section 5.5.6, Program AREA.FOR

VVGMOD

Call is made to set the drawing mode so that subsequent drawing operations:-

i replace the display image unconditionally.

ii replace it depending on a logical operation on frame buffer contents.

iii replace it depending on logical/arithmetic comparison with either frame buffer contents or drawing/comparison colors.

ROUTINE NAME: *VVGMOD - Drawing Mode*

CALL FORMAT: *istat = VVGMOD (cmod, pmod [,ccol])*

PARAMETERS:

cmod (type integer)

color mode :
0:draw foreground and background
1:draw foreground only - default
2:draw background only

pmod (type integer)

operational mode:
0:replace display image - 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 (type integer)

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

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Global Attribute Instructions - DRAWING_MODE

EXAMPLE OF USE

Section 5.5.1, Program DRAW.FOR

VVGNLC

Call is made to set up to 16 colors (in terms of index and red, green and blue intensities) in the color look-up table (CLUT).

ROUTINE NAME: ***VVGNLC - Normal Colors***

CALL FORMAT: ***istat = VVGNLC (ind1, int1, [ind2, int2,...indn, intn])***

where n = 3-16

N.B. Array Mode accepted.

PARAMETERS: ***indn (type integer)***

color index number in CLUT

intn (type integer)

intensities of red, green and blue

Minimum parameters 2, maximum 32.

REPLY VALUE: ***0***

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Global Attribute Instructions - NORMAL_COLORS

EXAMPLE OF USE

Section 5.5.7, Program THINGS.FOR

VVGSCB

Call is made to enable/disable screen blanking. Screen blanking gives priority to drawing rather than display. This allows drawing speed to increase by a factor of up to 4.

ROUTINE NAME: ***VVGSCB - Screen Blank***

CALL FORMAT: ***istat = VVGSCB (bmod)***

PARAMETERS: ***bmod (type integer)***
screen mode (blank/not blank)
0:screen not blank. Display has priority.
non-zero :screen blank. Drawing has priority.

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Global Attribute Instructions - SCREEN_BLANK

EXAMPLE OF USE Section 5.5.7, Program THINGS.FOR

4.6.4 Drawing Functions

VVDARC

Call is made to draw the specified sequence of circular arcs starting from the current position.

ROUTINE NAME:

VVDARC - Arcs Absolute/Relative

N.B. Dependent on Drawing Parameter Mode set by VVBMOD and Instruction Parameter Mode set by VVBPMD.

CALL FORMAT:

istat = VVDARC (dir1, xcen1, ycen1, xend1, yend1 [dir2, xcen2, ycen2, xend2, yend2... dirn, xcenn, ycenn, xendn, yendn])

where n has no defined limit

Using Array Mode :

istat = VVDARC (Array of dir/xcen/ycen/xend/yend values, number of elements in array)

PARAMETERS:

dirn (type integer)

drawing direction
0:counterclockwise
1:clockwise

xcenn (type integer)

X coordinate or displacement of center

ycenn (type integer)

Y coordinate or displacement of center

xendn (type integer)

X coordinate or displacement of end position

yendn (type integer)

Y coordinate or displacement of end position

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Drawing Instructions - ARCS_ABS and ARCS_REL

EXAMPLE OF USE

Section 5.5.7, Program THINGS.FOR

VVDCRC

Call is made to draw a circle with specified radius, centered on the current position.

ROUTINE NAME:	***VVDCRC - Circle***
CALL FORMAT:	***istat = VVDCRC (rad)***
PARAMETERS:	***rad (type integer)*** radius
REPLY VALUE:	***0*** success ***1*** not initialized ***2*** segment overflow
VIVID CROSS REF	Drawing Instructions - CIRCLE
EXAMPLE OF USE	Section 5.5.6, Program AREA.FOR

VVDDOT

Call is made to draw a dot at the current position, in the current drawing mode.

ROUTINE NAME: ***VVDDOT - Dot***

CALL FORMAT: ***istat = VVDDOT ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF Drawing Instructions - DOT

EXAMPLE OF USE Section 5.5.7, Program THINGS.FOR

VVDEAR

Call is made to draw the specified sequence of elliptic arcs starting from the current position.

ROUTINE NAME:

VVDEAR - Elliptic Arcs Absolute/Relative

N.B. Dependent on Drawing Parameter Mode set by VVBMOD and Instruction Parameter Mode set by VVBPMD.

CALL FORMAT:

istat = VVDEAR (dir1, ax1, ay1, xcen1, ycen1, xend1, yend1 [dir2, ax2, ay2, xcen2, ycen2, xend2, yend2...dirn, axn, ayn, xcenn, ycenn, xendn, yendn])

where n has no defined limit

Using Array Mode :

istat = VVDEAR (Array of dir/ax/ay/xcen/ycen/xend/ yend values, number of elements in array)

PARAMETERS:

dirn (type integer)
drawing direction
0:counterclockwise
1:clockwise

axn (type integer)
relative X length

ayn (type integer)
relative Y length

xcenn (type integer)
X coordinate or displacement of center

ycenn (type integer)
Y coordinate or displacement of center

xendn (type integer)
X coordinate or displacement of end position

yendn (type integer)
Y coordinate or displacement of end position

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Drawing Instructions - ELLIPSE_ARCS_ABS and ELLIPSE_ARCS_REL

EXAMPLE OF USE

Section 5.5.7, Program THINGS.FOR

VVDELL

Call is made to draw an ellipse of a specified VAS aspect ratio and major axis, with its center on the current position.

ROUTINE NAME:	***VVDELL - Ellipse***
CALL FORMAT:	***istat = VVDELL (ax, by, rad)***
PARAMETERS:	***ax (type integer)*** relative horizontal length ***by (type integer)*** relative vertical length ***rad (type integer)*** radius along X axis in VAS
REPLY VALUE:	***0*** success ***1*** not initialized ***2*** segment overflow
VIVID CROSS REF	Drawing Instructions - ELLIPSE
EXAMPLE OF USE	Section 5.5.7, Program THINGS.FOR

VVDLIN

Call is made to draw the specified sequence of lines, starting from the current position.

ROUTINE NAME:

VVDLIN - Lines Absolute/Relative

N.B. Dependent on Drawing Parameter Mode set by VVBMOD and Instruction Parameter Mode set by VVBPMD.

CALL FORMAT:

istat = VVDLIN (x1 or dx1, y1 or dy1[x2 or dx2, y2 or dy2,...xn or dxn, yn or dyn])

where n has no defined limit

Using Array Mode:

istat = VVDLIN (Array of x/y values, number of elements in array)

PARAMETERS:

xn (type integer)

X coordinate for the end of the line (Absolute)

dxn (type integer)

X displacement for the next end vector (Relative)

yn (type integer)

Y coordinate for the end of the line (Absolute)

dyn (type integer)

Y displacement for the next end vector (Relative)

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Drawing Instructions - LINES_ABS and LINES_REL

EXAMPLE OF USE

Section 5.5.6, Program AREA.FOR

Note: Variations in line thickness can be achieved by using the following techniques:

- **A filled rectangle (use VSL routine VVFRCT or the VIVID instructions FILLED_RECT_ABS or FILLED_RECT_REL)**
- **By PAINT/FLOOD**
- **Several thin lines.**

VVDMOV

Call is made to move the current drawing position to the absolute location or relative position specified.

ROUTINE NAME: ***VVDMOV - Move Absolute/Relative***

CALL FORMAT: ***istat = VVDMOV (x or dx, y or dy)***
N.B. Dependent on Drawing Parameter Mode.

PARAMETERS:

x (type integer)
X coordinate in VAS (Absolute)

dx (type integer)
X displacement from the current position (Relative)

y (type integer)
Y coordinate in VAS (Absolute)

dy (type integer)
Y displacement from the current position (Relative)

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Drawing Instructions - MOVE_ABS and MOVE_REL

EXAMPLE OF USE

Section 5.5.1, Program DRAW.FOR

VVDMTC

Call is made to give a move to the current cursor position.

ROUTINE NAME:	***VVDMTC - Move To Cursor***
CALL FORMAT:	***istat = VVDMTC ()***

PARAMETERS: ***none***

Reply Value:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF Drawing Instructions - MOVE_TO_CURSOR

VVDPMK

Call is made to draw the specified character from the current font at each of the points given by a list of X,Y coordinates or displacements.

ROUTINE NAME:

VVDPMK - Polymarks Absolute/Relative

N.B. Dependent on Drawing Parameter Mode set by VVBMOD and Instruction Parameter Mode set by VVBPMD.

CALL FORMAT:

istat = VVDPMK (ichar, x1 or dx1, y1 or dy1[x2 or dx2, y2 or dy2,...xn or dxn, yn or dyn])

where n has no defined limit

Using Array Mode:

istat = VVDPMK (Array of ichar followed by x/y values, number of elements in array)

PARAMETERS:

ichar (type integer)

index number of character required

xn (type integer)

X coordinate in VAS (Absolute)

dxn (type integer)

X displacement from the current position (Relative)

yn (type integer)

Y coordinate in VAS (Absolute)

dyn (type integer)

Y displacement from the current position (Relative)

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Drawing Instructions - POLYMARKS_ABS and POLYMARKS_REL

EXAMPLE OF USE

Section 5.5.7, Program THINGS.FOR

VVDREC

Call is made to draw a rectangle from a vertex at the current position to the diagonal vertex specified.

ROUTINE NAME: ***VVDREC - Rectangle Absolute/Relative***

N.B. Dependent on Drawing Parameter Mode set by VVBMOD.

CALL FORMAT: ***istat = VVDREC (x or dx, y or dy)***

PARAMETERS:

x (type integer)
X coordinate in VAS of opposite vertex (Absolute)

dx (type integer)
X displacement of opposite vertex (Relative)

y (type integer)
Y coordinate in VAS of opposite vertex (Absolute)

dy (type integer)
Y displacement of opposite vertex (Relative)

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Drawing Instructions - RECTANGLE_ABS and RECTANGLE_REL

EXAMPLE OF USE

Section 5.5.1, Program DRAW.FOR

4.6.5 Filled Figure Functions

VVFFLD

Call is made to fill the area from the current position to the defined edge color or to the current foreground color with the area texture pattern. The area texture pattern is written in Replace mode, irrespective of the current drawing mode.

ROUTINE NAME:

VVFFLD - Flood Area

CALL FORMAT:

istat = VVFFLD ([ind])

PARAMETERS:

ind (type integer)

CLUT index of edge color to which filling occurs. Range 0-15.
-1 or no parameter supplied defaults to the current foreground color.

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Filled Figure Instructions - FLOOD_AREA

EXAMPLE OF USE

Section 5.5.6, Program AREA.FOR

Note:

To change the red interior of a circle that has a green border using VVFFLD and VVFPNT use one of the following two procedures:

1 **Using VVFFLD, flood up to the green border as follows:**

   - **Set current position to a point inside the circle using VVDMOV**
   - **Change background color to a color which is NEITHER red or green, using for example, VVGBCL (15) - white. Also change the foreground color to a color which is NEITHER red or green, BUT is the required replacement color, that is, black, using VVGFCL (0). Note that this produces no immediate visual change, only color table values are changed.**

- The default color index for green is 8, so call VVFFLD as VVFFLD (8). The interior of the circle will be filled with black up to, but not including, the green border.

2 Using VVFPNT to paint over the red interior.

- Set current position to a point inside the circle using VVDMOV
- Change background color to a color which is NEITHER red or green, using for example, VVGBCL (15) - white. Also change the foreground color to a color which is NEITHER red or green, BUT is the required replacement color, that is, black, using VVGFCL (0). Note that these produce no immediate visual change, only color table values are changed.
- The default color index for red is 2, so call VVFPNT as VVFPNT (2). The interior of the circle will be replaced with black.

VVFPNT

Call is made to fill the area of the specified color which includes the current position with the area texture pattern.

ROUTINE NAME:

VVFPNT - Paint Area

CALL FORMAT:

istat = VVFPNT (ind)

PARAMETERS:

ind (type integer)
index of color to be replaced

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Filled Figure Instructions - PAINT

EXAMPLE OF USE

Section 5.5.7, Program THINGS.FOR

Note:

To change the red interior of a circle that has a green border using VVFFLD and VVFPNT use one of the following two procedures:

1 **Using VVFFLD, flood up to the green border as follows:**

- **Set current position to a point inside the circle using VVDMOV**
- **Change background color to a color which is NEITHER red or green, using for example, VVGBCL (15) - white. Also change the foreground color to a color which is NEITHER red or green, BUT is the required replacement color, that is, black, using VVGFCL (0). Note that this produces no immediate visual change, only color table values are changed.**
- **The default color index for green is 8, so call VVFFLD as VVFFLD (8). The interior of the circle will be filled with black up to, but not including, the green border.**

2 Using VVFPNT to paint over the red interior.

- Set current position to a point inside the circle using VVDMOV
- Change background color to a color which is NEITHER red or green, using for example, VVGBCL (15) - white. Also change the foreground color to a color which is NEITHER red or green, BUT is the required replacement color, that is, black, using VVGFCL (0). Note that these produce no immediate visual change, only color table values are changed.
- The default color index for red is 2, so call VVFPNT as VVFPNT (2). The interior of the circle will be replaced with black.

VVFRCT

Call is made to draw a rectangle from a vertex at the current position to the diagonal vertex specified. The rectangle is then filled with the area texture pattern.

ROUTINE NAME:

VVFRCT - Filled Rectangle Absolute/Relative

N.B. Dependent on Drawing Parameter Mode set by VVBMOD.

CALL FORMAT:

istat = VVFRCT (x or dx, y or dy)

PARAMETERS:

x (type integer)

X coordinate in VAS of opposite vertex (Absolute)

dx (type integer)

X displacement of opposite vertex (Relative)

y (type integer)

Y coordinate in VAS of opposite vertex (Absolute)

dy (type integer)

Y displacement of opposite vertex (Relative)

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Filled Figure Instructions - FILLED_RECT_ABS and FILLED_RECT_REL

EXAMPLE OF USE

Section 5.5.2, Program PAINTS.FOR

4.6.6 Text Functions

VVTDRC

Call is made to display the characters specified by each index in the parameter list. There is one index per word.

ROUTINE NAME: ***VVTDRC - Draw Characters***

CALL FORMAT: ***istat = VVTDRC (ind1[,ind2,...indn])***

where n has no defined limit
N.B. Array Mode accepted.

PARAMETERS: ***indn (type integer)***

index to cell in font

REPLY VALUE: ***0***

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Text Instructions - DRAW_CHARS

EXAMPLE OF USE

Section 5.5.3, Program FONT.FOR

VVTDRP

Call is made to display the characters specified by each index in the parameter list. Indices are packed two per parameter word.

ROUTINE NAME:

VVTDRP - Draw Packed Characters

N.B. Dependent on Instruction Parameter Mode set by VVBPMD.

CALL FORMAT:

istat = VVTDRP (i1, j1[i2, j2,...in, jn], wordcount)

where n has no defined limit

Using Array Mode :

istat = VVTDRP (Array of packed characters, number of WORDS in array)

There is an inherent difference in the writing of VSL applications in FORTRAN on VMS and RSX systems. It is to do with the way the operating systems handle strings. On RSX systems, a string buffer can be passed in a FORTRAN call simply by the name of the buffer containing the string, or by the defined string itself, and the address is used directly by the called function.

On VMS systems, the text string is passed by String Descriptor Block.

On RSX, an example of a call to this function would be:

CALL VVTDRP ('A string',4)

that is, a string plus the number of WORDS used in the string.

On VMS, an example of a call to this function would be:

CALL VVTDRP (%REF('A string'),4)

that is, the address of a string plus the number of words.

It will be seen that, under VMS, the use of the %REF qualifier ensures that the string address is passed to the VVTDRP function.

Other high-level languages are supported on VMS only, and therefore the collusion will not occur.

PARAMETERS:

in, jn (type character (byte))

any two characters from byte string

wordcount (type integer)

number of WORDS (that is, 2-character blocks)

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Text Instructions - DRAW_PACKED_CHARS

EXAMPLE OF USE

Section 5.5.1, Program DRAW.FOR

VVTIFT

Call is made to initialize the specified segment as a font, irrespective of the segment contents. This segment must already have been created via a call to VVMCRS.

ROUTINE NAME: ***VVTIFT - Initialize Font***

CALL FORMAT: ***istat = VVTIFT (segid, width, height, ncell [,init])***

PARAMETERS:

segid (type integer)
Segment ID

width (type integer)
cell width in pixels (1-16)

height (type integer)
cell height in pixels (1-16)

ncell (type integer)
number of cells in the font (>0)

init (type integer)
initialization style for cells:
0 or no parameter:blank - default
-1:solid

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Text Instructions - INITIALIZE_FONT

EXAMPLE OF USE

Section 5.5.3, Program FONT.FOR

VVTLDC

Call is made to load a character cell to the current font from the pixel data given as parameters.

ROUTINE NAME: ***VVTLDC - Load Character Cell***

CALL FORMAT: ***istat = VVTLDC (ind, irow1[,irow2,...irown])***
where n = 3-16
N.B. Array Mode accepted.

PARAMETERS: ***ind (type integer)***
cell index

irown (type integer)
image value for a pixel row

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Text Instructions LOAD_CHAR_CELL

EXAMPLE OF USE Section 5.5.3, Program FONT.FOR

VVTMAG

Call is made to define the horizontal and vertical cell magnification, in terms of pixels or relative magnification.

ROUTINE NAME: ***VVTMAG - Cell Magnification***

CALL FORMAT: ***istat = VVTMAG (utyp, xmag, ymag)***

PARAMETERS:

utyp (type integer)
code for magnification unit type
0:pixels
1:relative - default

xmag (type integer)
magnification in the cell X direction (range 1-16). The default is 1.

ymag (type integer)
magnification in the cell Y direction (range 1-16). The default is 2.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Text Instructions - CELL_MAGNIFICATION

EXAMPLE OF USE

Section 5.5.3, Program FONT.FOR

VVTMOV

Call is made to define text spacing, a relative movement from the end of one character cell to a final current position.

ROUTINE NAME: ***VVTMOV - Cell Movement***

CALL FORMAT: ***istat = VVTMOV (xd, yd)***

PARAMETERS: ***xd (type integer)***
horizontal displacement

yd (type integer)
vertical displacement

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Text Instructions - CELL_MOVEMENT

EXAMPLE OF USE Section 5.5.3, Program FONT.FOR

VVTOBL

Call is made to define whether subsequent cells are to be drawn rectangularly, or in italic (45-degree slope) form.

ROUTINE NAME: ***VVTOBL - Cell Oblique***

CALL FORMAT: ***istat = VVTOBL (ital)***

PARAMETERS: ***ital (type integer)***

parameter for rectangular or italic character
0:rectangular character
1:italic character

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Text Instructions - CELL_OBLIQUE

EXAMPLE OF USE

Section 5.5.7, Program THINGS.FOR

VVTROT

Call is made to define the angle at which cells are written to the display image. The angle is defined in 45-degree counterclockwise units.

ROUTINE NAME: ***VVTROT - Cell Rotation***

CALL FORMAT: ***istat = VVTROT (nseg)***

PARAMETERS: ***nseg (type integer)***
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

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Text Instructions - CELL_ROTATION

EXAMPLE OF USE Section 5.5.7, Program THINGS.FOR

VVTSFT

Call is made to set the current font to the identified font segment. This font is used for subsequent VIVID instructions which access fonts.

ROUTINE NAME: ***VVTSFT - Set Font***

CALL FORMAT: ***istat = VVTSFT (segid)***

PARAMETERS: ***segid (type integer)***

Font Segment ID

REPLY VALUE: ***0***

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Text Instructions - SET_FONT

EXAMPLE OF USE

Section 5.5.3, Program FONT.FOR

VVTSIZ

Call is made to define the length and width of the display image cell and the displacement of the stored font cell within the display image cell.

ROUTINE NAME: ***VVTSIZ - Cell Size***

CALL FORMAT: ***istat = VVTSIZ (width, height, xdis, ydis)***

PARAMETERS:

width (type integer
width of display cell in pixels (1-16)

height (type integer)
height of display cell in pixels (1-16)

xdis (type integer)
horizontal displacement of font cell (0-15)

ydis (type integer)
vertical displacement of font cell (0-15)

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF Text Instructions - CELL_SIZE

EXAMPLE OF USE Section 5.5.7, Program THINGS.FOR

4.6.7 Area Operation Functions

VVACLS

Call is made to clear the display.

ROUTINE NAME: ***VVACLS - Clear Screen***

CALL FORMAT: ***istat = VVACLS ([patt])***

PARAMETERS: ***patt (type integer)***
list of color indices for screen no parameter:screen is cleared to the current background color.

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Area Operation Instructions - CLEAR_SCREEN

EXAMPLE OF USE Section 5.5.1, Program DRAW.FOR

VVACLV

Call is made to clear the viewport to the current background color.

ROUTINE NAME: ***VVACLV - Clear Viewport***

CALL FORMAT: ***istat = VVACLV ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF Area Operation Instructions CLEAR_VIEWPORT

EXAMPLE OF USE Section 5.5.2, Program PAINTS.FOR

VVACPY

Call is made to copy a specified source area to an area with a vertex at the current position with a defined attitude. The origin of the source area is expressed either as an absolute position in VAS, or relative to the current position, depending on the current setting of the drawing parameter mode.

ROUTINE NAME: ***VVACPY - Copy Absolute/Relative***

CALL FORMAT: ***istat = VVACPY (amod, xs or dxs, ys or dys, xdim, ydim)***

N.B. Dependent on Drawing Parameter Mode.

PARAMETERS:

amod (type integer)
attitude mode

xs (type integer)
X position of the source area origin in VAS (Absolute)

dxs (type integer)
X VAS displacement of the source area origin (Relative)

ys (type integer)
Y position of the source area origin in VAS (Absolute)

dys (type integer)
Y VAS displacement of the source area origin (Relative)

xdim (type integer)
X dimension of the source copy area in VAS

ydim (type integer)
Y dimension of the source copy area in VAS

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Area Operation Instructions - COPY_ABS and COPY_REL

EXAMPLE OF USE

Section 5.5.6, Program AREA.FOR

VVAFPM

Call is made to write a specified segment that 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.

ROUTINE NAME: ***VVAFPM - Fast Pixel Modify***

CALL FORMAT: ***istat = VVAFPM (segid, mode, mask)***

PARAMETERS:

segid (type integer)
pixel data map segment ID

mode (type integer)
operational mode
0:replace display image
1:OR with display image
2:AND with display image
3:EOR with display image

mask (type integer)
word bit mask)

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Area Operation Instructions - FAST_PIXEL_MODIFY

EXAMPLE OF USE

Section 5.5.6, Program AREA.FOR

VVAFPR

Call is made to write a specified segment that 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.

ROUTINE NAME: ***VVAFPR - Fast Pixel Write***

CALL FORMAT: ***istat = VVAFPR (segid)***

PARAMETERS: ***segid (type integer)***
pixel data map segment ID

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Area Operation Instructions - FAST_PIXEL_WRITE

EXAMPLE OF USE Section 5.5.6, Program AREA.FOR

VVAPXR

Call is made to read a display image to a specified segment in host memory. The segment must already have been created via a call to VVMCRS. The segment may be used for subsequent pixel write operations.

ROUTINE NAME: ***VVAPXR - Pixel Readback***

CALL FORMAT: ***istat = VVAPXR (segid, dxw, dyp)***

PARAMETERS:

segid (type integer)
pixel data map segment ID

dxw (type integer)
area width in words (of 4 pixels each). Positive values indicate displacement to right.

dyp (type integer)
area height in pixels. Positive values indicate upward displacement.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Area Operation Instructions - PIXEL_READBACK

EXAMPLE OF USE

Section 5.5.6, Program AREA.FOR

VVAPXW

Call is made to write a specified segment containing pixel data to the display image at the current drawing position.

ROUTINE NAME: ***VVAPXW - Pixel Write***

CALL FORMAT: ***istat = VVAPXW (segid)***

PARAMETERS: ***segid (type integer)***
pixel data map segment ID

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF
Area Operation Instructions - PIXEL_WRITE

EXAMPLE OF USE
Section 5.5.6, Program AREA.FOR

VVASCL

Call is made to perform the specified logical operation on the rectangular area whose opposite vertices are defined by the current position and the specified displacement.

ROUTINE NAME: ***VVASCL - Selective Clear***

CALL FORMAT: ***istat = VVASCL (mode, mask, [patt], dxw, dyp)***

PARAMETERS:

mode (type integer)
operational mode
0:replace display image
1:OR with display image
2:AND with display image
3:EOR with display image

mask (type integer)
word bit mask

patt (type integer)
color bit pattern for 4 pixels

dxw (type integer)
signed area width in words (of 4 pixels each)

dyp (type integer)
signed area height in pixels

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Area Operation Instructions - SELECTIVE_CLEAR

EXAMPLE OF USE

Section 5.5.8, Program MATCH.FOR

VVASCV

Call is made to move the data within the viewport. The data is moved by the indicated displacement. Data falling outside the viewport is lost.

ROUTINE NAME: ***VVASCV - Scroll Viewport***

CALL FORMAT: ***istat = VVASCV (dx, dy)***

PARAMETERS: ***dx (type integer)***
horizontal displacement of data. Positive values indicate displacement to right.

dy (type integer)
vertical displacement of data. Positive values indicate upward displacement.

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Area Operation Instructions - SCROLL_VIEWPORT

EXAMPLE OF USE Section 5.5.4, Program TRANSF.FOR

4.6.8 Interactive Functions

VVIAKI

Call is made to begin synchronous keyboard input to the identified segment. Input continues until the specified termination character is received, a specified maximum number of characters has been read, or the segment is full.

ROUTINE NAME: ***VVIAKI - Accept Keyboard Input***

CALL FORMAT: ***istat = VVIAKI (segid, chend, chmax [,cind, cfore, cback])***

PARAMETERS:

segid (type integer)
the segment ID for writing the data

chend (type char (byte))
input termination character

N.B. Should be set to zero if unused.

chmax (type integer)
maximum number of characters input

cind (type integer)
cursor index in current font

cfore (type integer)
cursor foreground color index

cback (type integer)
cursor background color index

N.B. Keyboard input will only be echoed at the current drawing position if the last three parameters are supplied.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Interactive Instructions - ACCEPT_KEYBOARD_INPUT

EXAMPLE OF USE

Section 5.5.5, Program KEYIN.FOR

VVICUS

Call is made to set the cursor to the specified pixel data, or to one of the default cursor styles.

ROUTINE NAME: ***VVICUS - Cursor Style***

CALL FORMAT: ***istat = VVICUS (ccode[,dxp, dyp, row1, row2... rown])***

where n is in the range 3-16

PARAMETERS:

ccode (type integer)

cursor style code
0:small cross-hairs
-1:full screen cross-hairs
>0:width of cursor in pixels

N.B. Parameters 2 to n are only valid for ccode > 0.

dxp (type integer)

cell pixel X displacement from cursor point

dyp (type integer)

cell pixel Y displacement from cursor point

rown (type integer)

cursor cell row bit pattern

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Interactive Instructions - CURSOR_STYLE

EXAMPLE OF USE

Section 5.5.8, Program MATCH.FOR

VVICUS

Call is made to define whether or not the cursor is visible.

ROUTINE NAME: ***VVICUS - Cursor Visibility***

CALL FORMAT: ***istat = VVICUS (cmod)***

PARAMETERS: ***cmod (type integer)***
cursor visibility
0:cursor invisible
1:cursor visible

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF Interactive Instructions - CURSOR_VISIBILITY

EXAMPLE OF USE Section 5.5.2, Program PAINTS.FOR

VVIMTD

Call is made to disable match interrupts.

ROUTINE NAME: ***VVIMTD - Match Disable***

CALL FORMAT: ***istat = VVIMTD ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Interactive Instructions - MATCH_DISABLE

EXAMPLE OF USE

Section 5.5.8, Program MATCH.FOR

VVIMTE

Call is made to enable match interrupts such that when subsequent drawing intersects the cursor position, a report is sent to the host. Following this instruction, drawing continues until the maximum number of matches have been detected.

ROUTINE NAME: ***VVIMTE - Match Enable***

CALL FORMAT: ***istat = VVIMTE (nmax)***

PARAMETERS: ***nmax (type integer)***
maximum number of matches

REPLY VALUE: ***0***
success

1
not initialized

2
segment overflow

VIVID CROSS REF
Interactive Instructions - MATCH_ENABLE

EXAMPLE OF USE
Section 5.5.8, Program MATCH.FOR

VVIPCU

Call is made to set the cursor to the specified position when on display. The cursor is restricted by the screen boundaries.

ROUTINE NAME: ***VVIPCU - Position Cursor***

CALL FORMAT: ***istat = VVIPCU ([x,y])***

PARAMETERS:

x (type integer)
cursor X position in VAS

y (type integer)
cursor Y position in VAS

N.B. If no parameters supplied, cursor position defaults to the current drawing position.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Interactive Instructions - POSITION_CURSOR

EXAMPLE OF USE

Section 5.5.2, Program PAINTS.FOR

VVIRUB

Call is made to define the rubber band characteristics and base point.

ROUTINE NAME: ***VVIRUB - Rubber Band***

CALL FORMAT: ***istat = VVIRUB (rcod [,x,y])***

PARAMETERS:

rcod (type integer)
rubber band code
0:no rubber band
1:linear rubber band
2:rectangular rubber band

x (type integer)
X position of base point in VAS

y (type integer)
Y position of base point in VAS

N.B. If no X,y provided, the current drawing position is assumed as the base point of the rubber band.

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Interactive Instructions - RUBBER_BAND

EXAMPLE OF USE

Section 5.5.8, Program MATCH.FOR

VVISWD

Call is made to disable pointing device reporting.

ROUTINE NAME: ***VVISWD - Switch Report Disable***

CALL FORMAT: ***istat = VVISWD ()***

PARAMETERS: ***none***

REPLY VALUE:

0
success

1
not initialized

2
segment overflow

VIVID CROSS REF

Interactive Instructions - SWITCH_DISABLE

EXAMPLE OF USE

Section 5.5.4, Program TRANSF.FOR

VVISWE

Call is made to enable 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.

ROUTINE NAME: ***VVISWE - Switch Report Enable***

CALL FORMAT: ***istat = VVISWE (mask)***

PARAMETERS

mask

If switch mask is 0, then a report is generated for movement and any device input, for example, pushing one of the buttons.
If the switch mask is NOT 0 then if the following bits are set a report will be generated:

Bit Number (from 0)		
0	-	Left button
1	-	Middle button
2	-	Right button
3	-	If no DECtablet - set to 0 If DECtablet - set to 1 for response to 4th switch
4 .. 15	-	Must be zeros

REPLY VALUE:

0

success

1

not initialized

2

segment overflow

VIVID CROSS REF

Interactive Instructions - SWITCH_REPORT_ENABLE

EXAMPLE OF USE

Section 5.5.2, Program PAINTS.FOR

VVIWSW

Call is made to wait for a switch interrupt before continuing with the next VIVID instruction.

ROUTINE NAME:	***VVIWSW - Wait Switch***
CALL FORMAT:	***istat = VVIWSW (mask)***
PARAMETERS:	***mask (type integer)*** switch mask
REPLY VALUE:	***0*** success ***1*** not initialized ***2*** segment overflow
VIVID CROSS REF	Interactive Instructions - WAIT_SWITCH
EXAMPLE OF USE	Section 5.5.2, Program PAINTS.FOR

4.6.9 Report Handling Functions

VVQREP

Call is made to place the specified report in the current report segment.

ROUTINE NAME:	***VVQREP - Request Report***
CALL FORMAT:	***istat = VVQREP (nrep)***
PARAMETERS:	***nrep (type integer)*** report number required (range 0-10)
REPLY VALUE:	***0*** success ***1*** not initialized ***2*** segment overflow
VIVID CROSS REF	Report Handling - REQUEST_REPORT INSTRUCTION
EXAMPLE OF USE	Section 5.5.7, Program THINGS.FOR

5 GETTING STARTED WITH VSL

5.1 INTRODUCTION

VSL is a library of functions and subroutines that 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.

Thus VSL is designed to enable high-level language programmers to obtain easy access to the VSV21 graphics capability via use of the VIVID instruction set.

The VIVID Subroutine Library is provided for both VMS and RSX systems and in both cases is called VSLLIB.OLB.

5.2 ACCESS TO VIVID DISPLAY AREA

The high-level approach of VSL is such that the detailed QIO directive calls required to access the VSV21 graphics interpreter (VIVID) are transparent to the user. What the user DOES have to do is to include in his task build (or linkage) access to the VIVID Display Area.

In VMS terms, the display area is a word array, currently maximum size of 64K words. Access is therefore by array element. All the user needs to do is include access to the library at link time thus:

```
LINK USERFILE,SYS$LIBRARY:VSLLIB/LIB
```

In RSX terms, the display area is a common area of data, or region. Access to the region is via a 4K window VV21DA that must be mapped at APR 7, that is, in program space at 28K. An example of a task build command process, from a FORTRAN source, including access to the VIVID display area is:

```
TKB USERFILE,USERFILE/-SP=USERFILE
TKB LB:[1,1]VSLLIB/LB
TKB LB:[1,1]F4POTS/LB
TKB /
TKB VSECT=VV21DA:160000:20000
TKB WNDWS=1
TKB MAXBUF=512
TKB //
```

Note: The window is mapped as a virtual section at APR 7.

5.3 SEGMENTS

5.3.1 An Introduction To Segments

The VIVID display area resides in host memory. It is used by the VSV21 driver to store graphics information for subsequent processing on-board the VSV21 and by the on-board processor to store data resulting from graphics execution. Data is transferred from the host to on-board and vice-versa via the display area using direct memory access (DMA). The data within the area can be of various types:

- Executable VIVID instructions plus data
- Character cells in a font
- Pixel data maps
- Characters input from a keyboard
- Report packets generated as the result of graphics execution
- Attributes data written back from the board

These identifiable data entities are referred to as SEGMENTS.

Thus the definable segments are:

- Instruction
- Font
- Pixel
- Keyboard
- Report
- Attributes

Segments can also be downloaded directly into VSV21 memory to give increased speed of access. This facility is really designed for use with static segments i.e. those whose data will not change for the duration of processing such as fonts. Any change to a downloaded segment would necessitate the segment being deleted from VSV21 memory, rebuilt and re-downloaded.

5.3.2 Segment Size and Class/ID

The size of segment that you create is dependent on the specific application the segment is intended for. There is no restriction on the size of individual segments other than the maximum imposed by the VIVID Display Area or of VSV21 memory. It is advised to keep instruction segments fairly small, say 100 words maximum, purely for ease of programming. Users can nest calls to lower level segments from a top level segment and in this way pre-define specific graphics operations via segment ID. Editing of smaller entities makes for more efficient use of the segment concept, and thus a library of pictures can be set up as files on disk. VSL gives you the facility to create, store, and restore segments into host or VSV21 memory.

Segment Class/ID is a one-word value and is a reflection of the data entities discussed above. It is, in the main, user-definable but within the restrictions mentioned:

- Class is in the MSB of the word.

Range : Host Segments 1 to 32
VSV21 1 to 16

— Class 16 is reserved for multinational font segments.

— Class 32 is reserved for Report segments.

- ID is in the LSB of the word.
  Range : 1 to 255.

The use and application of each segment type will become apparent upon reference to the various VSL routine calls itemised in the previous chapter and also within the published examples at the end of this chapter.

5.4 VSL - LOGICAL PROCEDURE

To access the VSV21 via the high-level subroutine library VSL, a number of logical operations must be performed. These are itemised below with the relevant VSL routine in brackets:

- initialize VIVID processing (VVXINI)
- assign channel to VSV21 and allocate display area (VVXASS)
- create segment(s) within display area (VVMCRS)
- begin inserting VIVID instructions into segment (VVBBGN)
- generate VIVID instruction calls (various)
- end of segment (VVBEND)
- execute segment (VVEEXE)
- monitor execution status (VVRSTA)
- read any relevant reports (VVRREP)
- release display area and de-assign channel from VSV21 (VVXREL)
- terminate VSV21 processing (VVXEND)

The above stages represent a typical VSL program structure. Obviously, following execution of the segment, applications will differ as to the next steps of the program. Before release and termination of the VSV21, for instance, a program may read reports, read keyboard segments, create new segments, generate VIVID instructions within segments and execute them, delete segments, create new fonts, and so forth. The essential ingredients must be present in any program to achieve correct access of the VSV21 via VSL, namely:

- INITIALIZE
- ASSIGN CHANNEL AND ALLOCATE DISPLAY AREA
- CREATE SEGMENT(S)
- EXECUTE
- RELEASE DISPLAY AREA AND DEASSIGN CHANNEL
- TERMINATE

5.5 EXAMPLES

All the following examples are written in VAX FORTRAN and developed under VMS. RSX users should be able to use these examples but with special attention to two points:

1 All references to INTEGER*4 should be INTEGER*2.

2 All calls to "VVTDRP - Draw Packed Characters" should **NOT** use the %REF qualifier. (See VVTDRP in Section 4.6.6)

5.5.1 Draw a Picture

This is an example of using VSL to produce a simple display. It should especially aid new VSV21 users to access the VIVID instruction set by the use of the higher level calls within VSL.

Program Name: DRAW.FOR

```
        PROGRAM DRAW

C A simple example to illustrate VSL and its ease of use.
C Program sets up VSV21 processing and creates a segment to output
C a header message and sequence of colored rectangles.
C Full error handling is included.
C
C To compile and link the program :-
C
C VMS -
C       FOR/NOOP DRAW
C       LINK DRAW,SYS$LIBRARY:VSLLIB/LIB
C
C RSX -
C       F77 DRAW,DRAW/-SP=DRAW
C       TKB DRAW,DRAW/-SP=DRAW
C       TKB LB:[1,1]VSLLIB/LB
C       TKB LB:[1,1]F4POTS/LB
C       TKB /
C       TKB VSECT=VV21DA:160000:20000
C       TKB WNDWS=1
C       TKB MAXBUF=512
C       TKB //

        IMPLICIT INTEGER*4 (V)

        INTEGER*4       DRAWIT
        INTEGER*2       ISTAT,LUN

C Initialize VIVID processing.
C Set up the display area to be 4096 bytes long,
C maximum segments to be 10.

        ISTAT = VVXINI (4096, 10)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXINI  ', ISTAT)  ! Check status

C Assign VSV21 device for VSL processing.
C First, set up a logical unit number for all subsequent device access.

        LUN = 1

C Device physical unit 0 will assign to device VVA0:
C                      1 will assign to device VVB0:
C                      2 will assign to device VVC0:  etc.
C Third parameter 1024 sets up a report segment of that size,
C default segment ID Hex 2001.
```

```
        ISTAT = VVXASS (0, LUN, 1024)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXASS   ', ISTAT)  ! Check status

C Create a VIVID instruction segment of length 1000 bytes

        ISTAT = VVMCRS (LUN, '201'X, 1000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS1 ', ISTAT)  ! Check status

C Call subroutine to build up the display segment and output the picture
C Supply logical unit number and segment ID.

        ISTAT = DRAWIT (LUN, '201'X)

        IF (ISTAT .NE. 1)  CALL SRERR ('DRAWIT   ', ISTAT)  ! Check status

C Release VSV21 device from VSL processing.

        ISTAT = VVXREL (LUN)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL   ', ISTAT)  ! Check status

C Release VSV21 processor and free the VSV21 buffers.

        ISTAT = VVXEND ()

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL   ', ISTAT)  ! Check status

        STOP

        END

C Set up title "THIS IS VSV21" and draw colored rectangles

        INTEGER*4 FUNCTION  DRAWIT (LUN, SEGID)

        INTEGER*2       LUN,SEGID,ISTAT,RECX,RECY,I

        CALL  VVBBGN (SEGID)            ! start of segment
        CALL  VVCINI (-1)               ! initialize all
        CALL  VVACLS ()                 ! clear screen
        CALL  VVGMOD (1,0)              ! drawing mode - foreground only
        CALL  VVGFCL (14)               ! foreground color yellow
        CALL  VVDMOV (36,420)           ! move abs
        CALL  VVTMAG (1,3,3)            ! cell mag 3x
        CALL  VVBPMD (1)                ! set param mode for arrays
        CALL  VVTDRP (%REF('T H I S  I S  V S V 2 1 '),12)
        CALL  VVBPMD (0)                ! reset param mode
        CALL  VVCINI (-1)               ! initialize all
        CALL  VVDMOV (176,128)          ! move abs
        RECX = 448                      ! init rectangle x-val
        RECY = 368                      ! init rectangle y-val

C 2 loops to produce 30 colored rectangles

        DO 100 I=1,15,1
            CALL  VVGFCL (I)            ! foreground colors 1-15
            CALL  VVBMOD (1)            ! instruction mode - relative
            CALL  VVDMOV (4,4)          ! move rel
            CALL  VVBMOD (0)            ! instruction mode - absolute
            CALL  VVDREC (RECX,RECY)    ! rectangle absolute
            RECX = RECX - 4             ! rectangles in by 4
            RECY = RECY - 4
100     CONTINUE

        DO 200 I=1,15,1
            CALL  VVGFCL (I)            ! foreground colors 1-15
            CALL  VVBMOD (1)            ! instruction mode - relative
            CALL  VVDMOV (4,4)          ! move rel
            CALL  VVBMOD (0)            ! instruction mode - absolute
            CALL  VVDREC (RECX,RECY)    ! rectangle absolute
            RECX = RECX - 4             ! rectangles in by 4
            RECY = RECY - 4
200     CONTINUE

        CALL  VVBEND ()                 ! end of segment
```

```
C Now execute the segment

        CALL   VVEEXE (LUN,SEGID,32000)  ! execute segment
        CALL   VVRSTA (LUN, DRAWIT, IDUM) ! get execute status

        RETURN
        END

C Error handling.
C Routine prints function and error code.

        SUBROUTINE  SRERR (ISRNAM, ISTAT)       ! Handle errors

        CHARACTER*8        ISRNAM
        INTEGER*2           ISTAT

1000    FORMAT  (/, ' **** Failed: routine ', a8,
        1        '   Status = ', I7, ' ****', /)

        WRITE (5, 1000) ISRNAM, ISTAT
        STOP
        END
```

5.5.2 Reporting

This example illustrates a simple use of the reporting facility. The program draws a sequence of colored boxes down the right hand side of the screen as a palette of colors. Using a pointing device, users can then position the cursor over one of the boxes, hit a switch and the left-hand remainder of the screen will flood to the selected color. This demonstrates a simple menu selection facility using cursor-position reports and wait-switch instruction.

Program Name: PAINTS.FOR

```
        PROGRAM PAINTS
C
C An exercise in programming VSV21 graphics using the
C       VIVID SUBROUTINE LIBRARY (VSL).
C
C IMPORTANT. This test requires Pointing Device input.
C
C Object is to create a 16-color paint box as a menu, from which to
C select a color and clear the viewport to that color.
C Exercise demonstrates VIVID drawing facilities plus pointing device
C interaction and the reading of reports.
C
C To compile and link the program :-
C
C VMS -
C       FOR/NOOP PAINTS
C       LINK PAINTS,SYS$LIBRARY:VSLLIB/LIB
C
C RSX -
C       F77 PAINTS,PAINTS/-SP=PAINTS
C       TKB PAINTS,PAINTS/-SP=PAINTS
C       TKB LB:[1,1]VSLLIB/LB
C       TKB LB:[1,1]F4POTS/LB
C       TKB /
C       TKB VSECT=VV21DA:160000:20000
C       TKB WNDWS=1
C       TKB MAXBUF=512
C       TKB //

        IMPLICIT INTEGER*4 (V)

        INTEGER*4       ICNTRL
        INTEGER*2       ISTAT,LUN
        DATA  SEGID  / '0201'X /

C Initialize VIVID processing.
C Set up the display area to be 4096 bytes long,
C maximum segments to be 10.

        ISTAT = VVXINI (4096, 10)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXINI  ', ISTAT)  ! Check status

C Assign VSV21 device for VSL processing.
C First, set up a logical unit number for all subsequent device access.

        LUN = 1

        ISTAT = VVXASS (0, LUN, 1024)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXASS  ', ISTAT)  ! Check status

C Create the top-level segment .

        ISTAT = VVMCRS (LUN, SEGID, 2000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS 1', ISTAT)

C Call control routine

        ISTAT = ICNTRL (LUN, SEGID)

        IF (ISTAT .NE. 1)  CALL SRERR ('ICNTRL  ', ISTAT)

C Release VSV21 device from VSL processing.

        ISTAT = VVXREL (LUN)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status

C Release VSV21 processor and free the VSV21 buffers.

        ISTAT = VVXEND ()

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXEND  ', ISTAT)  ! Check status

        CALL EXIT

        END

C
C Control routine
C

        INTEGER*4  FUNCTION ICNTRL (LUN,SEGID)

        INTEGER*2  LUN,SEGID

C
C Reporting
C
        INTEGER*4       VVRREP,IREP
        INTEGER*2       FCOL,IARR
        DIMENSION       IARR(20)

C Set up paintbox menu

        ICNTRL = IDINIT(LUN,SEGID)

        IF (ICNTRL .NE. 1) THEN
            CALL SRERR ('IDINIT  ', ICNTRL)
        ENDIF

C
C Read Switch Reports.
C
```

```
110     IREP = VVRREP (LUN,65,IARR,20)
        IF (IREP .EQ. 1) THEN
            IVALID = 1
            GOTO 110
        ELSE IF (IREP .GE. 0) THEN
            CALL SRERR ('VVRREP  ',IREP)
        ENDIF

C
C Read X/y coordinates to set up correct color
C
        IF  (IARR(3) .GE. 488) THEN
            IF  (IARR(4) .GE. 430) THEN         ! Exit
                GOTO  999
            ELSE IF  (IARR(4) .GE. 400) THEN
                FCOL = 1
            ELSE IF  ((IARR(4) .GE. 372) .AND. (IARR(4) .LE. 390)) THEN
                FCOL = 2
            ELSE IF  ((IARR(4) .GE. 344) .AND. (IARR(4) .LE. 362)) THEN
                FCOL = 3
            ELSE IF  ((IARR(4) .GE. 316) .AND. (IARR(4) .LE. 334)) THEN
                FCOL = 4
            ELSE IF  ((IARR(4) .GE. 288) .AND. (IARR(4) .LE. 306)) THEN
                FCOL = 5
            ELSE IF  ((IARR(4) .GE. 260) .AND. (IARR(4) .LE. 278)) THEN
                FCOL = 6
            ELSE IF  ((IARR(4) .GE. 232) .AND. (IARR(4) .LE. 250)) THEN
                FCOL = 7
            ELSE IF  ((IARR(4) .GE. 204) .AND. (IARR(4) .LE. 222)) THEN
                FCOL = 8
            ELSE IF  ((IARR(4) .GE. 176) .AND. (IARR(4) .LE. 194)) THEN
                FCOL = 9
            ELSE IF  ((IARR(4) .GE. 148) .AND. (IARR(4) .LE. 166)) THEN
                FCOL = 10
            ELSE IF  ((IARR(4) .GE. 120) .AND. (IARR(4) .LE. 138)) THEN
                FCOL = 11
            ELSE IF  ((IARR(4) .GE. 92) .AND. (IARR(4) .LE. 110)) THEN
                FCOL = 12
            ELSE IF  ((IARR(4) .GE. 64) .AND. (IARR(4) .LE. 82)) THEN
                FCOL = 13
            ELSE IF  ((IARR(4) .GE. 36) .AND. (IARR(4) .LE. 54)) THEN
                FCOL = 14
            ELSE IF  ((IARR(4) .GE. 8) .AND. (IARR(4) .LE. 26)) THEN
                FCOL = 15
            ENDIF
        ENDIF

C
C Call 'clear viewport' routine.
C Even if no valid color selection, the routine will wait for next switch.
C
        CALL CLEAR (LUN, SEGID, FCOL)

        IF (ICNTRL .NE. 1) THEN
            CALL SRERR ('CLEAR   ', ICNTRL)
        ENDIF

        GOTO  110                        ! keep looping for switch reports

999     RETURN
        END

C
C Draw the Paintbox Menu
C
        FUNCTION IDINIT (LUN, SEGID)

        INTEGER*2  LUN,SEGID
        CALL VVBMOD (0)                 ! Drawing instruction mode ABS
        CALL VVBBGN (SEGID)             ! Start Segment
        CALL VVCINI (-1)                ! Initialize everything
        CALL VVVDIM (640, 480)          ! Screen Dimensions 640x480
        CALL VVACLS ('0044'X)           ! Clear screen(black/green stripes)
        CALL VVGFCL (11)                ! Foreground color pink
        CALL VVDMOV (  0,   0)          ! Move abs
        CALL VVFRCT (479, 479)          ! Filled rectangle
        CALL VVGFCL (0)                 ! Foreground color black
        CALL VVDMOV (  3,   3)          ! Move abs
        CALL VVFRCT (476, 476)          ! Filled rectangle
        CALL VVTMAG (1, 1, 2)           ! Cell magnification
        CALL VVGFCL (14)                ! Foreground color yellow
        CALL VVGBCL (0)                 ! Background color black
        CALL VVGMOD (1, 0)              ! Drawing mode,foreground only
        CALL VVDMOV (480, 458)          ! Move abs
        CALL VVBPMD (1)                 ! Set array mode (for text)
        CALL VVTDRP (%ref('    MY PAINTBOX     '), 10)
        CALL VVDMOV (488,430)           ! Move abs
        CALL VVGBCL (14)                ! Background color yellow
        CALL VVGFCL (2)                 ! Foreground color red
        CALL VVGMOD (0,0)               ! Drawing mode,fore/background
        CALL VVTDRP (%ref('       EXIT       '), 9)
C
C Now set up the colors of the paint box
C
        CALL VVGMOD (1,0)               ! Drawing mode,foreground only
        CALL VVGBCL (0)                 ! Background color black
        CALL VVGFCL (1)                 ! Foreground color deep blue
        CALL VVDMOV (500, 400)          ! Move abs
        CALL VVBMOD (1)                 ! Drawing instruction mode RELATIVE
        CALL VVFRCT (120, 18)           ! Filled rectangle
        CALL VVGFCL (2)                 ! Deep Red
        CALL VVDMOV (0, -28)            ! Move rel X by 0,y by -28
        CALL VVFRCT (120, 18)           ! Filled rectangle
        CALL VVGFCL (3)                 ! Purple
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (4)                 ! Dark Green
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (5)                 ! Blue
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (6)                 ! Orange
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (7)                 ! Deep pink
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (8)                 ! Green
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (9)                 ! Light blue
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (10)                ! Pale orange
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (11)                ! Pink
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (12)                ! Bright green
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (13)                ! Pale blue
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (14)                ! Yellow
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
        CALL VVGFCL (15)               ! White
        CALL VVDMOV (0, -28)
        CALL VVFRCT (120, 18)
C
C Draw the rectangle around the screen and put cursor on
C
        CALL VVBMOD (0)                ! Drawing instruction mode ABS
        CALL VVDMOV (0,0)              ! Move abs
        CALL VVGFCL (15)               ! Foreground color white
        CALL VVDREC (639,479)          ! Rectangle abs
        CALL VVIPCU (320,240)          ! Position cursor
        CALL VVICUS (1)                ! Cursor visibility
        CALL VVISWE (7)                ! Switch enable (1,2,3)
        CALL VVIWSW (7)                ! Wait switch (1,2,3)
        CALL VVBEND ()                 ! End segment
        CALL VVEEXE (LUN,SEGID,32000)  ! Execute segment
        CALL VVRSTA (LUN, IDINIT, IDUM) ! Check execute completion status
        RETURN
        END

C
C Clear viewport to supplied color.
C Note. Viewport is set so as not to clear the paintbox menu also.
C       It must be reset to enable the cursor position checks to work.
C

        FUNCTION  CLEAR (LUN,SEGID,FCOL)

        INTEGER*2  LUN,SEGID,FCOL

        CALL VVBBGN (SEGID)            ! Start segment
        CALL VVVSVU (4,4,472,472)      ! Set viewport to non-menu size
        CALL VVGBCL (FCOL)             ! Background color to that selected
        CALL VVACLV ()                 ! Clear viewport (to background)
        CALL VVVSVU (0,0,640,480)      ! Reset viewport
        CALL VVIWSW (7)                ! Wait switch (1,2,3)
        CALL VVBEND ()                 ! End segment
        CALL VVEEXE (LUN,SEGID,32000)  ! Execute segment
        CALL VVRSTA (LUN, CLEAR, IDUM) ! Check execute completion status
        RETURN
        END

C Error handling.
C Routine prints function and error code.

        SUBROUTINE  SRERR (ISRNAM, ISTAT)       ! Handle errors

        CHARACTER*8       ISRNAM
        INTEGER*2          ISTAT

1000    FORMAT  (/, ' **** Failed: routine ', a8,
        1        '   Status = ', I7, ' ****', /)

        WRITE (5, 1000) ISRNAM, ISTAT
        STOP
        END
```

5.5.3 Font Creation

Note:

When designing a character (or texture, or special cursor) a 16 by 16 grid is used in which:

a. Rows are numbered 1 to 16 from the bottom of the grid.

b. **In terms of bit positions, the columns are numbered 0 to 15 from left to right. Thus, if a shaded unit of the grid represents a 1, and an unshaded unit represents a 0, then the correct row values supplied to VSL functions or VIVID instructions are obtained by REVERSING each row value.**

See Appendix B for an example.

This example illustrates how to create a user-font of special symbols. It demonstrates defining and initializing the font, displaying characters from it and switching back to the default font.

Program Name: FONT.FOR

```
      PROGRAM FONT

C Program shows how to create a font of six special characters. These
C symbols are :-
C               STAR sign
C               INFINITY sign
C               ARROW righthand pointer
C               'bb' as one symbol
C               A matchstick man symbol
C               FEMALE logical symbol
C
C A string of opening text is displayed from the default font. Special
C font is then created and the six symbols displayed with movement and
C magnification. We then switch back to defaults and display a closing
C message.
C Full error handling is included.
C
C To compile and link the program :-
C
C VMS -
C       FOR/NOOP FONT
C       LINK FONT,SYS$LIBRARY:VSLLIB/LIB
C
C RSX -
C       F77 FONT,FONT/-SP=FONT
C       TKB FONT,FONT/-SP=FONT
C       TKB LB:[1,1]VSLLIB/LB
C       TKB LB:[1,1]F4POTS/LB
C       TKB /
C       TKB VSECT=VV21DA:160000:20000
C       TKB WNDWS=1
C       TKB MAXBUF=512
C       TKB //

      PROGRAM FONT
C
C An exercise in programming VSV21 graphics using the
C       VIVID SUBROUTINE LIBRARY (VSL).
C
C Object is to create a small font of special characters for subsequent
C text output to the screen.
C Exercise demonstrates VIVID text facilities and cell manipulation.
C

      IMPLICIT INTEGER*4 (V)

      INTEGER*4       ICNTRL
      INTEGER*2       ISTAT,LUN

      DATA  SEGID  / '0201'X /              ! Instruction segment ID

      DATA  FONTID  / '1004'X /             ! Special font segment ID

C Initialize VIVID processing.
C Set up the display area to be 4096 bytes long,
C maximum segments to be 10.
```

```
        ISTAT = VVXINI (4096, 10)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXINI  ', ISTAT)  ! Check status

C Assign VSV21 device for VSL processing.
C First, set up a logical unit number for all subsequent device access.

        LUN = 1

        ISTAT = VVXASS (0, LUN, 1024)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXASS  ', ISTAT)  ! Check status

C Create the top-level segment .

        ISTAT = VVMCRS (LUN, SEGID, 2000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS 1', ISTAT)

C Create the font segment

        ISTAT = VVMCRS (LUN, FONTID, 800)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS 2', ISTAT)

C Call control routine

        ISTAT = ICNTRL (LUN, SEGID, FONTID)

        IF (ISTAT .NE. 1)  CALL SRERR ('ICNTRL  ', ISTAT)

C Release VSV21 device from VSL processing.

        ISTAT = VVXREL (LUN)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status

C Release VSV21 processor and free the VSV21 buffers.

        ISTAT = VVXEND ()

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXEND  ', ISTAT)  ! Check status

        CALL EXIT

        END

C
C Control routine
C

        INTEGER*4  FUNCTION ICNTRL (LUN, SEGID, FONTID)

        INTEGER*2  LUN, SEGID, FONTID

        INTEGER*2  STAR (11)
        DATA STAR / 1, '111'X, '92'X, '54'X, '38'X, '1FF'X, '38'X,
        1           '54'X, '92'X, '111'X, '8000'X /

        INTEGER*2  INFINI (11)
        DATA INFINI / 2, 0, 0, 0, 'EE'X, '111'X, 'EE'X, 0, 0, 0, '8000'X /

        INTEGER*2  ARROW (11)
        DATA ARROW / 3, 0, '20'X, '40'X, '80'X, '1FF'X, '80'X,
        1            '40'X, '20'X, 0, '8000'X /

        INTEGER*2  BB (11)
        DATA BB / 4, 0, '1EF'X, '129'X, '129'X, '1EF'X, '21'X, '21'X, '21'X, 0
        1        '8000'X /

        INTEGER*2  MAN (11)
        DATA MAN / 5, '101'X, '82'X, '44'X, '28'X, '10'X, '7C'X,
        1          '92'X, '38'X, '38'X, '8000'X /

        INTEGER*2  FEMALE (11)
        DATA FEMALE / 6, '38'X, '44'X, '82'X, '82'X, '44'X, '38'X,
        1             '10'X, '38'X, '10'X, '8000'X /
        CALL VVBBGN (SEGID)            ! Start Segment
        CALL VVCINI (-1)               ! Initialise everything
        CALL VVACLS ('FFFF'X)          ! Clear screen (white)
        CALL VVGFCL (5)                ! Foreground color blue
        CALL VVGBCL (15)               ! Background color white
        CALL VVDMOV (20, 440)          ! Move abs
        CALL VVBPMD (1)                ! Set array mode (for text)
        CALL VVTDRP (%ref('TEXT FROM DEFAULT FONT'), 11)

C Create a new font of special characters :
C       Each cell 9 X 9 pixels, 6 cells in all

        CALL VVTIFT (FONTID, 9, 9, 6)
        CALL VVTSFT (FONTID)           ! Point at new font
        CALL VVTLDC (STAR)
        CALL VVTLDC (INFINI)
        CALL VVTLDC (ARROW)
        CALL VVTLDC (BB)
        CALL VVTLDC (MAN)
        CALL VVTLDC (FEMALE)
        CALL VVBPMD (0)                ! Set array mode (off)

C Now output each cell as text - magnify X 6 first

        CALL VVTMAG (1, 6, 6)          ! Cell magnification X 6
        CALL VVDMOV (20, 240)          ! Move abs
        CALL VVGFCL (2)                ! Foreground color red
        CALL VVTMOV (2,0)              ! Cell movement X+2
        CALL VVTDRC (1,2,3,4,5,6)      ! Output the six cells
        CALL VVTSFT ('10FF'X)          ! Point back at default font
        CALL VVDMOV (20, 140)          ! Move abs
        CALL VVGFCL (5)                ! Foreground color blue
        CALL VVTMAG (1, 1, 2)          ! Cell magnification default
        CALL VVTMOV (0,0)              ! Cell movement default
        CALL VVBPMD (1)                ! Set array mode (for text)
        CALL VVTDRP (%ref('BACK TO DEFAULT FONT'), 10)
        CALL VVBEND ()                 ! End segment
        CALL VVEEXE (LUN,SEGID,32000)  ! Execute segment
        CALL VVRSTA (LUN, ICNTRL, IDUM) ! Check execute completion status
        RETURN
        END

C Error handling.
C Routine prints function and error code.

        SUBROUTINE  SRERR (ISRNAM, ISTAT)        ! Handle errors

        CHARACTER*8        ISRNAM
        INTEGER*2           ISTAT

1000    FORMAT  (/, ' **** Failed: routine ', a8,
        1        '   Status = ', I7, ' ****', /)

        WRITE (5, 1000) ISRNAM, ISTAT
        STOP
        END
```

5.5.4 Transformations

This example illustrates the use of the graphics transformation instructions, their interaction, and their effect upon the display output.

Program Name: TRANSF.FOR

```
        PROGRAM TRANSF

C A simple example to illustrate all transformations performed
C upon an image in VAS.

C Program sets up a main picture of 8 colored rectangles numbered 1-8.
C These are large enough to take up most of the display and so
C subsequent transformations are easy to discern. Program flow is
C controlled by the pointing device switches. The top-level picture is
C always returned to after each transformation example, user is then
C requested to press the switch to observe the next transformation.

C
C IMPORTANT. This test requires Pointing Device input.
C
C
C To compile and link the program :-
C
C VMS -
C       FOR/NOOP TRANSF
C       LINK TRANSF,SYS$LIBRARY:VSLLIB/LIB
C
C RSX -
C       F77 TRANSF,TRANSF/-SP=TRANSF
C       TKB TRANSF,TRANSF/-SP=TRANSF
C       TKB LB:[1,1]VSLLIB/LB
C       TKB LB:[1,1]F4POTS/LB
C       TKB /
C       TKB VSECT=VV21DA:160000:20000
C       TKB WNDWS=1
C       TKB MAXBUF=512
C       TKB //

        IMPLICIT INTEGER*4 (V)

        INTEGER*4       DRAWIT
        INTEGER*2       ISTAT,LUN


C Initialize VIVID processing.
C Set up the display area to be 4096 bytes long,
C maximum segments to be 10.

        ISTAT = VVXINI (6000, 10)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXINI  ', ISTAT)  ! Check status


C Assign VSV21 device for VSL processing.
C First, set up a logical unit number for all subsequent device access.

        LUN = 1

        ISTAT = VVXASS (0, LUN, 1024)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXASS  ', ISTAT)  ! Check status

C Create a VIVID instruction segment.

        ISTAT = VVMCRS (LUN, '201'X, 1000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS1 ', ISTAT)  ! Check status

        ISTAT = VVMCRS (LUN, '301'X, 3000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS1 ', ISTAT)  ! Check status

C Call subroutine to build up the display segment and output the picture
C Supply logical unit number and segment ID.

        ISTAT = DRAWIT (LUN, '201'X)

        IF (ISTAT .NE. 1)  CALL SRERR ('DRAWIT ', ISTAT)  ! Check status

C Release VSV21 device from VSL processing.

        ISTAT = VVXREL (LUN)
        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status
C Release VSV21 processor and free the VSV21 buffers.
        ISTAT = VVXEND ()
        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status
        STOP
        END

C Set up picture in VAS and perform transformations on it.
        INTEGER*4 FUNCTION  DRAWIT (LUN, SEGID)
        INTEGER*2       LUN,SEGID,ISTAT
        CALL  VVBBGN (SEGID)            ! start of segment
        CALL  VVGMOD (1,0)              ! drawing mode - foreground only
        CALL  VVGFCL (4)                ! foreground color dark green
        CALL  VVDMOV (40,240)           ! move abs
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVFRCT (140,200)          ! filled rect
        CALL  VVDMOV (50,80)            ! move rel
        CALL  VVTMAG (1,6,6)            ! call mag x6
        CALL  VVGFCL (14)               ! foreground color yellow
        CALL  VVTDRC (49)               ! draw 1
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (180,240)          ! move abs
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVFRCT (140,200)          ! filled rect
        CALL  VVDMOV (50,80)            ! move rel
        CALL  VVTMAG (1,6,6)            ! call mag x6
        CALL  VVGFCL (4)                ! foreground color dark green
        CALL  VVTDRC (50)               ! draw 2
        CALL  VVGFCL (5)                ! foreground color blue
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (320,240)          ! move abs
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVFRCT (140,200)          ! filled rect
        CALL  VVDMOV (50,80)            ! move rel
        CALL  VVTMAG (1,6,6)            ! call mag x6
        CALL  VVGFCL (12)               ! foreground color light green
        CALL  VVTDRC (51)               ! draw 3
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (460,240)          ! move abs
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVFRCT (140,200)          ! filled rect
        CALL  VVDMOV (50,80)            ! move rel
        CALL  VVTMAG (1,6,6)            ! call mag x6
        CALL  VVGFCL (5)                ! foreground color blue
        CALL  VVTDRC (52)               ! draw 4
        CALL  VVGFCL (15)               ! foreground color white
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (40,40)            ! move abs
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVFRCT (140,200)          ! filled rect
        CALL  VVDMOV (50,80)            ! move rel
        CALL  VVTMAG (1,6,6)            ! call mag x6
        CALL  VVGFCL (11)               ! foreground color pink
        CALL  VVTDRC (53)               ! draw 5
CALL   VVBMOD (0)                 ! absolute mode
CALL   VVDMOV (180,40)            ! move abs
CALL   VVBMOD (1)                 ! relative mode
CALL   VVFRCT (140,200)           ! filled rect
CALL   VVDMOV (50,80)             ! move rel
CALL   VVTMAG (1,6,6)             ! call mag x6
CALL   VVGFCL (15)                ! foreground color white
CALL   VVTDRC (54)                ! draw 6

CALL   VVGFCL (13)                ! foreground color pale blue
CALL   VVBMOD (0)                 ! absolute mode
CALL   VVDMOV (320,40)            ! move abs
CALL   VVBMOD (1)                 ! relative mode
CALL   VVFRCT (140,200)           ! filled rect
CALL   VVDMOV (50,80)             ! move rel
CALL   VVTMAG (1,6,6)             ! call mag x6
CALL   VVGFCL (2)                 ! foreground color red
CALL   VVTDRC (55)                ! draw 7

CALL   VVBMOD (0)                 ! absolute mode
CALL   VVDMOV (460,40)            ! move abs
CALL   VVBMOD (1)                 ! relative mode
CALL   VVFRCT (140,200)           ! filled rect
CALL   VVDMOV (50,80)             ! move rel
CALL   VVTMAG (1,6,6)             ! cell mag x6
CALL   VVGFCL (13)                ! foreground color light blue
CALL   VVTDRC (56)                ! draw 8

CALL   VVBEND ()                  ! end of segment

CALL   VVBBGN ('301'X)            ! start of segment
CALL   VVCINI (-1)                ! init all
CALL   VVISWE (7)                 ! enable switches
CALL   TOPPIC (SEGID,'PRESS SWITCH:   SET WINDOW 440,340',1)
CALL   VVVWND (0,0,440,340)       ! set window
CALL   VVACLS (0)                 ! clear screen black
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   TOPPIC (SEGID,'PRESS SWITCH: SET WINDOW 1000,1000',0)
CALL   VVVWND (0,0,1000,1000)     ! set window
CALL   VVACLS (0)                 ! clear screen black
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   TOPPIC (SEGID,'PRESS SWITCH: SET VIEWPORT         ',0)
CALL   VVVSVU (10,360,120,120)    ! set viewport
CALL   VVACLS (0)                 ! clear screen black
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (150,360,120,120)   ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (290,360,120,120)   ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (430,360,120,120)   ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (10,220,120,120)    ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (150,220,120,120)   ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (290,220,120,120)   ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (430,220,120,120)   ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (10,80,120,120)     ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (150,80,120,120)    ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (290,80,120,120)    ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
CALL   VVVSVU (430,80,120,120)    ! set viewport
CALL   VVCCAL (SEGID)             ! redraw pic
```

```
        CALL  TOPPIC (SEGID,'SWITCH: WINDOW ORIGIN +-300, +-200',0)
        CALL  VVACLS (0)                ! clear screen black
        CALL  VVVWOR (300,200)          ! window origin
        CALL  VVCCAL (SEGID)            ! redraw pic
        CALL  VVVWOR (-300,-200)        ! window origin
        CALL  VVCCAL (SEGID)            ! redraw pic

        CALL  TOPPIC (SEGID,'PRESS SWITCH:SET VIEWPORT AND ZOOM',0)
        CALL  VVACLS (0)                ! clear screen black
        CALL  VVVSVU (170,170,190,190)  ! set viewport
        CALL  VVVZMF (90,90)            ! zoom factor
        CALL  VVCCAL (SEGID)            ! redraw pic

        CALL  TOPPIC (SEGID,'SWITCH: TRANSFORMATIONS ON/OFF/ON ',0)
        CALL  VVACLS (0)                ! clear screen black
        CALL  VVVDRT (-10,-10)          ! drawing translation
        CALL  VVVDRM ('200'X,'200'X)    ! drawing mag x2
        CALL  VVCCAL (SEGID)            ! redraw pic
        CALL  VVCDWT (180)              ! wait 3 secs
        CALL  VVACLS (0)                ! clear screen black
        CALL  VVVVAS ()                 ! drawing VAS
        CALL  VVCCAL (SEGID)            ! redraw pic
        CALL  VVCDWT (180)              ! wait 3 secs
        CALL  VVACLS (0)                ! clear screen black
        CALL  VVVTRN ()                 ! drawing transform
        CALL  VVCCAL (SEGID)            ! redraw pic

        CALL  TOPPIC (SEGID,'PRESS SWITCH: SCROLL VIEWPORT     ',0)
        CALL  VVVSVU (0, 38, 640, 442)  ! set viewport - don't scroll text
        CALL  VVCREP (50)               ! scroll 50 times
        CALL  VVASCV (10,10)            ! scroll viewport
        CALL  VVCERP ()                 ! end repeat
        CALL  VVVSVU ()                 ! reset viewport

        CALL  TOPPIC (SEGID,'END OF TRANSFORMATIONS TESTING    ',2)

        CALL  VVISWD ()                 ! disable switch reports
        CALL  VVBEND ()                 ! end of segment

        CALL  VVEEXE (LUN,'301'X,32000) ! execute segment
        CALL  VVRSTA (LUN, DRAWIT, IDUM) ! get execute status

        RETURN
        END

C Error handling.
C Routine prints function and error code.

        SUBROUTINE  SRERR (ISRNAM, ISTAT)       ! Handle errors

        CHARACTER*8        ISRNAM
        INTEGER*2           ISTAT

1000    FORMAT  (/, ' **** Failed: routine ', a8,
        1        '   Status = ', I7, ' ****', /)

        WRITE (5, 1000) ISRNAM, ISTAT
        STOP
        END

        SUBROUTINE  TOPPIC (SEGID,TEX,KEY) ! Display top level picture

        INTEGER*2       SEGID, KEY
        CHARACTER*34    TEX
        CALL  VVCINI ('FDFF'X)          ! init all
        CALL  VVGFCL (14)               ! foreground color yellow
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (60,10)            ! move abs
        CALL  VVBPMD (1)                ! array list
        CALL  VVTMAG (1,2,2)            ! cell mag x2
        IF  (KEY .NE. 1) THEN
            CALL  VVTDRP (%ref('PRESS SWITCH: BACK TO MAIN PICTURE'),17)
            CALL  VVIWSW (7)            ! wait on switch
        ENDIF
        CALL  VVACLS (0)                ! clear screen black
        CALL  VVDMOV (60,10)            ! move abs
        CALL  VVTDRP (%ref(TEX),17)
        CALL  VVTMAG (1,1,2)            ! cell mag reset
        CALL  VVBPMD (0)                ! not array list
        CALL  VVCCAL (SEGID)            ! redraw pic
        IF  (KEY .NE. 2) THEN
            CALL  VVIWSW (7)            ! wait on switch
        ENDIF

        RETURN

        END
```

5.5.5 Keyboard Input

This example illustrates the use of the VSL keyboard input routines.

Program Name: KEYIN.FOR

```
        PROGRAM KEYIN

C Test program takes graphics keyboard input and echoes it to the
C screen. Characters are entered up to a termination character '!' or a
C maximum of 20. Keyboard segment is then interrogated and the number of
C characters typed is displayed as a message.
C
C To compile and link the program :-
C
C VMS -
C       FOR/NOOP KEYIN
C       LINK KEYIN,SYS$LIBRARY:VSLLIB/LIB
C
C RSX -
C       F77 KEYIN,KEYIN/-SP=KEYIN
C       TKB KEYIN,KEYIN/-SP=KEYIN
C       TKB LB:[1,1]VSLLIB/LB
C       TKB LB:[1,1]F4POTS/LB
C       TKB /
C       TKB VSECT=VV21DA:160000:20000
C       TKB WNDWS=1
C       TKB MAXBUF=512
C       TKB //
C
C
        IMPLICIT INTEGER*4 (V)

        INTEGER*4       DRAWIT
        INTEGER*2       ISTAT, LUN, SEGID, KEYID

        DATA  SEGID  / '0201'X /        ! Instruction segment ID
        DATA  KEYID  / '0477'X /        ! Keyboard input segment ID

C Initialize VIVID processing.
C Set up the display area to be 4096 bytes long,
C maximum segments to be 10.

        ISTAT = VVXINI (4096, 10)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXINI  ', ISTAT)  ! Check status
```

```
C Assign VSV21 device for VSL processing.
C First, set up a logical unit number for all subsequent device access.

        LUN = 1

C Device physical unit 0 will assign to device VVA0:
C                      1 will assign to device VVB0:
C                      2 will assign to device VVC0:  etc.
C Third parameter 1024 sets up a report segment of that size,
C default segment ID Hex 2001.

        ISTAT = VVXASS (0, LUN, 1024)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXASS  ', ISTAT)  ! Check status

C Create a VIVID instruction segment, length 400 bytes.

        ISTAT = VVMCRS (LUN, SEGID, 400)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS1 ', ISTAT)  ! Check status

C Create a Keyboard Input segment, length 400 bytes.

        ISTAT = VVMCRS (LUN, KEYID, 400)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS2 ', ISTAT)  ! Check status

C Call subroutine to build up the display segment and output the picture
C Supply logical unit number and segment ID.

        ISTAT = DRAWIT (LUN, SEGID, KEYID)

        IF (ISTAT .NE. 1)  CALL SRERR ('DRAWIT ', ISTAT)  ! Check status

C Release VSV21 device from VSL processing.

        ISTAT = VVXREL (LUN)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status

C Release VSV21 processor and free the VSV21 buffers.

        ISTAT = VVXEND ()

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status

        CALL EXIT

        END

C
C Initialise graphics and print message "ENTER MESSAGE NOW"
C

        INTEGER*4 FUNCTION  DRAWIT (LUN, SEGID, KEYID)

        INTEGER*2       LUN,SEGID,KEYID,ISTAT,COUNT
        BYTE            CHARIN (20)
        CHARACTER*2     NUMBERS (20)

        DATA NUMBERS / '01','02','03','04','05','06','07','08','09','10',
        1              '11','12','13','14','15','16','17','18','19','20' /

        CALL  VVBBGN (SEGID)            ! start of segment
        CALL  VVCINI (-1)               ! initialise all
        CALL  VVACLS (0)                ! clear screen
        CALL  VVGMOD (1,0)              ! drawing mode - foreground only
        CALL  VVGFCL (14)               ! foreground color yellow
        CALL  VVDMOV (36,240)           ! move abs
        CALL  VVVDRM (768,768)          ! drawing mag 3x
        CALL  VVBPMD (1)                ! set param mode for arrays
        CALL  VVTDRP (%REF('ENTER MESSAGE NOW '),9)
        CALL  VVBPMD (0)                ! reset param mode
        CALL  VVVDRM ()                 ! drawing mag reset
        CALL  VVDMOV (36,100)
```

```
C
C Now get keyboard input
C
        CALL  VVGFCL (2)                ! foreground color red
        CALL  VVIAKI(KEYID,33,20,62,2,15) ! Cursor = 'red >'
        CALL  VVBEND ()

C Now execute the display list

        CALL  VVEEXE (LUN,SEGID,32000)  ! execute segment
        CALL  VVRSTA (LUN, DRAWIT, IDUM) ! get execute status

C
C Interrogate the keyboard input segment and count the number of characters.
C Then formulate a text message in the instruction segment and execute it.
C

C First get keyboard input segment chars into the array

        CALL  VVRKBD(KEYID,CHARIN,20,IDAT,ISTAT)

C Now start building an instruction segment

        CALL  VVBBGN (SEGID)            ! start of segment
        CALL  VVCINI (-1)               ! initialise all
        CALL  VVGMOD (1,0)              ! drawing mode - foreground only
        CALL  VVGFCL (14)               ! foreground color yellow

        COUNT = 0                       ! initialise character counter

        DO 100 I=1,20,1                 ! step through keyboard input segment
            IF  (CHARIN(I) .EQ. 0) THEN ! test for NULL
                GOTO 200
            ENDIF
            COUNT = COUNT+1             ! increment count
100     CONTINUE

200     CALL  VVDMOV (36,50)
        CALL  VVBPMD (1)                ! set param mode for arrays
        CALL  VVTDRP (%REF('NUMBER OF CHARACTERS INPUT :  '),15)
        CALL  VVGFCL (13)               ! foreground color pale blue
        IF  (COUNT .EQ. 0) THEN
            CALL  VVTDRP (%REF('00'),1)
        ELSE
            CALL  VVTDRP (%REF(NUMBERS(COUNT)),1)
        ENDIF
        CALL  VVBPMD (0)                ! reset param mode

C Now execute the display list

        CALL  VVEEXE (LUN,SEGID,32000)  ! execute segment
        CALL  VVRSTA (LUN, DRAWIT, IDUM) ! get execute status

        RETURN
        END

C Error handling.
C Routine prints function and error code.

        SUBROUTINE  SRERR (ISRNAM, ISTAT)        ! Handle errors

        CHARACTER*8         ISRNAM
        INTEGER*2            ISTAT

1000    FORMAT  (/, ' **** Failed: routine ', a8,
        1        '   Status = ', I7, ' ****', /)

        WRITE (5, 1000) ISRNAM, ISTAT
        STOP
        END
```

5.5.6 Area Operations

This example illustrates the use of the Area Operations instructions. The program prints a header underlined three times with dotted lines, then draws a striped circle. Part of the circle area is read to a pixel segment and then the segment is displayed using FAST_PIXEL_WRITE, PIXEL_WRITE and FAST_PIXEL_MODIFY. Finally a corner of the circle area is redisplayed using the COPY instruction.

Program Name: AREA.FOR

```
        PROGRAM AREA
C
C This example shows the use of the Area Operation Instructions.
C Full error handling is included.
C
C To compile and link the program :-
C
C VMS -
C       FOR/NOOP AREA
C       LINK AREA,SYS$LIBRARY:VSLLIB/LIB
C
C RSX -
C       F77 AREA,AREA/-SP=AREA
C       TKB AREA,AREA/-SP=AREA
C       TKB LB:[1,1]VSLLIB/LB
C       TKB LB:[1,1]F4POTS/LB
C       TKB /
C       TKB VSECT=VV21DA:160000:20000
C       TKB WNDWS=1
C       TKB MAXBUF=512
C       TKB //

        IMPLICIT INTEGER*4 (V)

        INTEGER*4       DRAWIT
        INTEGER*2       ISTAT,LUN,SEGID,PIXID

        DATA SEGID / '201'X /           ! Instruction segment ID
        DATA PIXID / '401'X /           ! Pixel Data segment ID

C Initialize VIVID processing.
C Set up the display area to be 8192 bytes long,
C maximum segments to be 10.

        ISTAT = VVXINI (8192, 10)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXINI  ', ISTAT)  ! Check status

C Assign VSV21 device for VSL processing.
C First, set up a logical unit number for all subsequent device access.

        LUN = 1

C Device physical unit 0 will assign to device VVA0:
C                      1 will assign to device VVB0:
C                      2 will assign to device VVC0:  etc.
C Third parameter 1024 sets up a report segment of that size,
C default segment ID Hex 2001.

        ISTAT = VVXASS (0, LUN, 1024)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXASS  ', ISTAT)  ! Check status

C Create a VIVID instruction segment of length 1000 bytes

        ISTAT = VVMCRS (LUN, SEGID, 1000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS1 ', ISTAT)  ! Check status

C Create a Pixel Data segment of length 2000 bytes
```

```
        ISTAT = VVMCRS (LUN, PIXID, 2000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS2 ', ISTAT)  ! Check status

C Call subroutine to build up the display segment and perform Area
C operations
C Supply logical unit number and segment IDs.

        ISTAT = DRAWIT (LUN, SEGID, PIXID)

        IF (ISTAT .NE. 1)  CALL SRERR ('DRAWIT  ', ISTAT)  ! Check status

C Release VSV21 device from VSL processing.

        ISTAT = VVXREL (LUN)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status

C Release VSV21 processor and free the VSV21 buffers.

        ISTAT = VVXEND ()

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status

        STOP

        END


C Control routine to draw initial picture and set up Area operations


        INTEGER*4 FUNCTION  DRAWIT (LUN, SEGID, PIXID)

        INTEGER*2       LUN, SEGID, PIXID, ISTAT

        CALL  VVBBGN (SEGID)            ! start of segment
        CALL  VVCINI (-1)               ! initialise all
        CALL  VVACLS ('FFFF'X)          ! clear screen to white
        CALL  VVGMOD (1,0)              ! drawing mode - foreground only
        CALL  VVGFCL (2)                ! foreground color red
        CALL  VVGBCL (15)               ! background color white
        CALL  VVDMOV (10,420)           ! move abs
        CALL  VVTMAG (1,2,2)            ! cell mag 2x
        CALL  VVBPMD (1)                ! set param mode for arrays
        CALL  VVTDRP (%REF('V S V 2 1  A R E A  O P E R A T I O N S '),20)
        CALL  VVBPMD (0)                ! reset param mode
        CALL  VVTMAG (1,1,2)            ! reset cell mag
        CALL  VVGLTX (16,'5555'X)       ! line texture dotted
        CALL  VVDMOV (0,410)            ! move abs
        CALL  VVDLIN (640,410)          ! dotted line
        CALL  VVDMOV (0,400)            ! move abs
        CALL  VVDLIN (640,400)          ! dotted line
        CALL  VVDMOV (0,390)            ! move abs
        CALL  VVDLIN (640,390)          ! dotted line
        CALL  VVDMOV (320,240)          ! move abs to center
        CALL  VVGFCL (5)                ! foreground color blue
        CALL  VVGLTX (16,'FFFF'X)       ! line texture reset
        CALL  VVDCRC (100)              ! circle
C
C use AREA TEXTURE to set up a pattern of stripes
C
        CALL  VVGATX (16, 'FF00'X, 'FF00'X, 'FF00'X, 'FF00'X, 'FF00'X, 'FF00'X
        1             'FF00'X, 'FF00'X, 'FF00'X, 'FF00'X, 'FF00'X, 'FF00'X,
        2             'FF00'X, 'FF00'X, 'FF00'X, 'FF00'X)
        CALL  VVCSAV ()                 ! save current attributes
        CALL  VVGBCL (13)               ! background color pale blue
        CALL  VVFFLD (5)                ! flood within circle
        CALL  VVCRES (4)                ! restore current colors

C Now PIXEL READBACK part of the circled area into the pixel data segment.
C Then
C 1) Write it back with FAST PIXEL WRITE
C       2) Write it back with PIXEL WRITE
C       3) FAST PIXEL MODIFY the segment
C       4) COPY part of the circle area
```

```
        CALL  VVDMOV (300,200)          ! move abs within circle
        CALL  VVAPXR (PIXID,12,48)      ! read back 48x48 pixels
        CALL  VVGFCL (0)                ! foreground color black
        CALL  VVBPMD (1)                ! set param mode for arrays
        CALL  VVDMOV (236,346)          ! move abs
        CALL  VVTDRP (%REF('PIXEL READBACK AREA '),10)
        CALL  VVDMOV (544,340)          ! move abs
        CALL  VVTDRP (%REF('FAST'),2)
        CALL  VVDMOV (544,320)          ! move abs
        CALL  VVTDRP (%REF('PIXEL '),3)
        CALL  VVDMOV (544,300)          ! move abs
        CALL  VVTDRP (%REF('WRITE '),3)
        CALL  VVBPMD (0)                ! reset param mode
        CALL  VVDMOV (544,184)          ! move abs (pixel word boundary)
        CALL  VVAFPR (PIXID)            ! fast pixel write of saved square
        CALL  VVDMOV (38,184)           ! move abs (not pixel word boundary)
        CALL  VVAPXW (PIXID)            ! pixel write square
        CALL  VVBPMD (1)                ! set param mode for arrays
        CALL  VVDMOV (38,320)           ! move abs
        CALL  VVTDRP (%REF('PIXEL '),3)
        CALL  VVDMOV (38,300)           ! move abs
        CALL  VVTDRP (%REF('WRITE '),3)
        CALL  VVBPMD (0)                ! reset param mode
        CALL  VVDMOV (98,10)            ! move abs
        CALL  VVAFPM (PIXID,0,'FF'X)    ! fast pixel modify
        CALL  VVBPMD (1)                ! set param mode for arrays
        CALL  VVDMOV (38,58)            ! move abs
        CALL  VVTDRP (%REF('FAST'),2)
        CALL  VVDMOV (38,38)            ! move abs
        CALL  VVTDRP (%REF('PIXEL '),3)
        CALL  VVDMOV (38,18)            ! move abs
        CALL  VVTDRP (%REF('MODIFY'),3)
        CALL  VVDMOV (320,58)           ! move abs
        CALL  VVTDRP (%REF('COPY'),2)
        CALL  VVBPMD (0)                ! reset param mode
        CALL  VVDMOV (380,10)           ! move abs
C
C Copy a slice of the circle and display it
C
        CALL  VVACPY (0,340,230,100,100)! copy abs
        CALL  VVBEND ()                 ! end of segment

C Now execute the segment

        CALL  VVEEXE (LUN,SEGID,32000)  ! execute segment
        CALL  VVRSTA (LUN, DRAWIT, IDUM) ! get execute status

        RETURN
        END

C Error handling.
C Routine prints function and error code.

        SUBROUTINE  SRERR (ISRNAM, ISTAT)       ! Handle errors

        CHARACTER*8        ISRNAM
        INTEGER*2           ISTAT

1000    FORMAT  (/, ' **** Failed: routine ', a8,
        1        '   Status = ', I7, ' ****', /)

        WRITE (5, 1000) ISRNAM, ISTAT
        STOP
        END
```

5.5.7 General VSL Calls

This example demonstrates some of the VSL routine calls that have not been used in the previous examples. The program sets up a display of dark blue faces drawn with arcs and ellipses against a pale blue background. Some colors are blinked and then the user is able to use the pointing device to position the cursor across the picture and paint sections red. Program creates an Attributes segment in VSV21 memory and dumps and recovers text attributes. Before-and-after cell parameters reports are displayed on the console.

Program Name: THINGS.FOR

```
        PROGRAM THINGS

C This program contains various VSL routine calls not specifically
C covered in the previous examples. It is thus by nature a "hotch potch"
C of drawing, reporting, blinking, cursor movement etc. activities.
C Full error handling is included.
C
C To compile and link the program :-
C
C VMS -
C       FOR/NOOP THINGS
C       LINK THINGS,SYS$LIBRARY:VSLLIB/LIB
C
C RSX -
C       F77 THINGS,THINGS/-SP=THINGS
C       TKB THINGS,THINGS/-SP=THINGS
C       TKB LB:[1,1]VSLLIB/LB
C       TKB LB:[1,1]F4POTS/LB
C       TKB /
C       TKB VSECT=VV21DA:160000:20000
C       TKB WNDWS=1
C       TKB MAXBUF=512
C       TKB //

        IMPLICIT INTEGER*4 (V)

        INTEGER*4       DRAWIT
        INTEGER*2       ISTAT,LUN

C Initialize VIVID processing.
C Set up the display area to be 8192 bytes long,
C maximum segments to be 10.

        ISTAT = VVXINI (8192, 10)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXINI  ', ISTAT)  ! Check status

C Assign VSV21 device for VSL processing.
C First, set up a logical unit number for all subsequent device access.

        LUN = 1

C Device physical unit 0 will assign to device VVA0:
C                      1 will assign to device VVB0:
C                      2 will assign to device VVC0:  etc.
C Third parameter 2048 sets up a report segment of that size,
C default segment ID Hex 2001.

        ISTAT = VVXASS (0, LUN, 2048)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXASS  ', ISTAT)  ! Check status

C Create a VIVID instruction segment of length 1000 bytes

        ISTAT = VVMCRS (LUN, '201'X, 1000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS  ', ISTAT)  ! Check status
```

```
C Call subroutine to build up the display segment and output the picture
C Supply logical unit number and segment ID.

        ISTAT = DRAWIT (LUN, '201'X)

        IF (ISTAT .NE. 1)  CALL SRERR ('DRAWIT  ', ISTAT)  ! Check status

C Release VSV21 device from VSL processing.

        ISTAT = VVXREL (LUN)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status


C Release VSV21 processor and free the VSV21 buffers.

        ISTAT = VVXEND ()

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status

        STOP

        END

C Main drawing routine

        INTEGER*4 FUNCTION  DRAWIT (LUN, SEGID)

        INTEGER*4       IREP, VVRREP
        INTEGER*2       LUN,SEGID,ISTAT,IARR,ATTID,I
        DIMENSION       IARR(20)
        DATA ATTID / '10A'X /           ! Attribute Segment ID

        CALL  VVBBGN (SEGID)            ! start of segment
        CALL  VVCINI (-1)               ! initialise all
C
C Set up normal colors
C
        CALL VVGNLC (0, '000'X,  1, '00F'X,  2, 'F00'X,  3, 'F0F'X,
        1            4, '050'X,  5, '05F'X,  6, 'F50'X,  7, 'F00'X,
        2            8, '0A0'X,  9, '0AF'X, 10, 'FA0'X,
        3           11, 'FAF'X, 12, 'FAF'X, 13, '0FF'X, 14, 'FF0'X,
        4           15, 'FFF'X)
        CALL  VVACLS ('DDDD'X)          ! clear screen to pale blue
        CALL  VVGMOD (1,0)              ! drawing mode - foreground only
        CALL  VVCCRS (ATTID,512)        ! create attribute dump segment
        CALL  VVCDMP (ATTID)            ! dump attributes
        CALL  VVGFCL (2)                ! foreground color red
        CALL  VVDMOV (25,26)            ! move abs
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVCREP (100)              ! display repeat
        CALL  VVDDOT ()                 ! draw a dot
        CALL  VVDMOV (6,0)              ! move rel
        CALL  VVCERP ()                 ! end repeat
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (25,30)            ! move abs
        CALL  VVTMAG (1,3,3)            ! cell mag 3x
        CALL  VVTOBL (1)                ! cell oblique
        CALL  VVTSIZ (16,16,2,0)        ! cell size
        CALL  VVBPMD (1)                ! set param mode for arrays
        CALL  VVTDRP (%REF('VSV21 THINGS'),6)
        CALL  VVBPMD (0)                ! reset param mode
        CALL  VVTMAG (1,1,2)            ! cell mag reset
        CALL  VVTOBL (0)                ! cell not oblique
        CALL  VVDMOV (25,20)            ! move abs
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVCREP (4)                ! display repeat
        CALL  VVDPMK (%REF('*'),20,0,20,0,20,0,20,0,20,0,20,0,20,0)
        CALL  VVCERP ()                 ! end repeat
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVGFCL (5)                ! foreground color blue
        CALL  VVDMOV (296,440)          ! move abs
        CALL  VVGSCB (1)                ! screen blank (drawing has priority)
```

```
C
C Draw arcs clockwise (first circle)
C
      CALL  VVDARC (1,320,440,344,440,1,320,440,296,440)
      CALL  VVDMOV (264,416)
C
C Draw arcs clockwise (second circle)
C
      CALL  VVDARC (0,288,416,312,416,0,288,416,264,416)
      CALL  VVDMOV (328,416)          ! move abs
C
C Draw arcs clockwise (third circle)
C
      CALL  VVDARC (1,352,416,376,416,1,352,416,328,416)
      CALL  VVDMOV (416,416)          ! move abs
C
C Draw ellipse arcs clockwise
C
      CALL  VVDEAR (1,4,2,480,416,544,416,1,4,2,480,416,416,416)
      CALL  VVDMOV (420,410)          ! move abs
      CALL  VVGFCL (14)               ! foreground color yellow
      CALL  VVFPNT (13)               ! paint
      CALL  VVGFCL (5)                ! foreground color blue
      CALL  VVDMOV (96,416)           ! move abs
C
C Draw ellipse arcs anticlockwise
C
      CALL  VVDEAR (0,2,1,160,416,224,416,0,2,1,160,416,96,416)
      CALL  VVDMOV (100,410)          ! move abs
      CALL  VVGFCL (14)               ! foreground color yellow
      CALL  VVFPNT (13)               ! paint
      CALL  VVGFCL (5)                ! foreground color blue
      CALL  VVDMOV (84,224)           ! move abs
      CALL  VVCREP (4)                ! display repeat 4
C
C draw a face
C
      CALL  VVDELL (1,4,32)           ! ellipse
      CALL  VVDELL (2,3,5)            ! ellipse
      CALL  VVBMOD (1)                ! relative mode
      CALL  VVDMOV (24,48)            ! move rel
      CALL  VVDCRC (2)                ! circle
      CALL  VVDELL (4,1,24)           ! ellipse
      CALL  VVDMOV ('FFD0'X,0)        ! move rel
      CALL  VVDMOV (0,2)              ! move rel
      CALL  VVDARC (1,0,'FFFE'X,0,'FFFC'X)    ! arcs
      CALL  VVDARC (1,0,2,0,4)        ! arcs
      CALL  VVDMOV (0,'FFFE'X)        ! move rel
      CALL  VVDELL (4,1,24)           ! ellipse
      CALL  VVDMOV (24,'FFD0'X)       ! move rel
      CALL  VVDMOV (16,'FFE0'X)       ! move rel
      CALL  VVDEAR (1,1,2,'FFF0'X,32,'FFE0'X,0,0,1,1,16,32,32,0)
    ! ellip arcs
      CALL  VVDMOV ('FFF0'X,32)       ! move rel
C
      CALL  VVDMOV (152,0)            ! move rel
      CALL  VVCERP ()                 ! display end repeat
      CALL  VVGSCB (0)                ! screen blank
    ! (display has priority)
      CALL  VVBMOD (0)                ! back to absolute mode
C
C Now blink the yellow and pink
C
      CALL  VVGBLC (1,14,'F0F'X,2,14,'F0F'X)! blink yellow with pink
      CALL  VVGBCT (2)                ! blink count 2
      CALL  VVGBLT (48,24)            ! blink timings
      CALL  VVGBLK (1)                ! blink enable
      CALL  VVTROT (2)                ! cell rotate 90 degrees
      CALL  VVTMAG (1,3,3)            ! cell mag 3x
      CALL  VVDMOV (32,112)           ! move abs
```

```
        CALL  VVGFCL (2)                ! foreground color red
        CALL  VVBPMD (1)                ! array mode
        CALL  VVTDRP (%REF('VSV21 '),3)
        CALL  VVTROT (6)                ! cell rotate 270 degrees
        CALL  VVDMOV (600,340)          ! move abs
        CALL  VVTDRP (%REF('VSV21 '),3)
C
C Use the cursor to paint parts of the picture red
C
        CALL  VVQREP (2)                ! request report cell prams BEFORE
        CALL  VVCRCV (ATTID,64)         ! recover default text attributes
        CALL  VVQREP (2)                ! request report cell prams AFTER
        CALL  VVDMOV (10,360)           ! move abs
        CALL  VVTDRP (%REF('SWITCH 1/2:PAINT WITH CURSOR,SWITCH 4:EXIT'),21)
        CALL  VVBPMD (0)                ! array mode off
        CALL  VVICUS (1)                ! make cursor visible
        CALL  VVISWE (7)                ! enable switches
        CALL  VVIWSW (7)                ! wait on a switch
        CALL  VVCSTP ()                 ! stop display
        CALL  VVBEND ()                 ! end of segment

C Now execute the segment

100     CALL  VVEEXE (LUN,SEGID,32000)  ! execute segment
        CALL  VVRSTA (LUN, DRAWIT, IDUM) ! get execute status

        IF (DRAWIT .NE. 1)  CALL SRERR ('DRAWIT   ', ISTAT)  ! Check status
C
C Read Cell Parameters Reports.
C
105     IREP = VVRREP (LUN,2,IARR,20)
        IF (IREP .EQ. 1) THEN
            PRINT *,'Cell Report :'
            PRINT *,' ',IARR(3),' ',IARR(4),' ',IARR(5),' ',IARR(6)
            PRINT *,' ',IARR(7),' ',IARR(8),' ',IARR(9),' ',IARR(10)
            PRINT *,' ',IARR(11),' ',IARR(12),' ',IARR(13),' ',IARR(14)
            GOTO 105
        ELSE IF (IREP .GE. 0) THEN
            CALL SRERR ('VVRREP  ',IREP)
        ENDIF
C
C Read Switch Reports.
C
110     IREP = VVRREP (LUN,65,IARR,20)
        IF (IREP .EQ. 1) THEN
            IVALID = 1
            GOTO 110
        ELSE IF (IREP .GE. 0) THEN
            CALL SRERR ('VVRREP  ',IREP)
        ENDIF
C
C Read switch number - if logical 4 then exit
C
        IF  (IARR(5) .EQ. 4) THEN
            GOTO 999
        ENDIF

        CALL  VVBBGN (SEGID)            ! set up a small segment
        CALL  VVDMTC ()                 ! move to cursor
        CALL  VVGFCL (2)                ! foreground color red
        CALL  VVFPNT (13)               ! paint it
        CALL  VVISWE (7)                ! enable switches
        CALL  VVIWSW (7)                ! wait on switch
        CALL  VVBEND ()
        GOTO  100

999     RETURN
        END


C Error handling.
C Routine prints function and error code.
```

```
              SUBROUTINE  SRERR (ISRNAM, ISTAT)       ! Handle errors

              CHARACTER*8        ISRNAM
              INTEGER*2           ISTAT

      1000    FORMAT  (/, ' **** Failed: routine ', a8,
              1        '   Status = ', I7, ' ****', /)

              WRITE (5, 1000) ISRNAM, ISTAT
              STOP
              END
```

5.5.8 Match Interrupts/Cursor Style

This program demonstrates the use of the match enable/disable instructions and their effects upon drawing output to the screen. It also includes an example of designing a special cursor.

Program Name: MATCH.FOR

```
        PROGRAM MATCH

C A demonstration of the VSL routines to enable/disable match interrupts.
C Program shows how drawing is disabled whilst match interrupts are enabled.
C Also shown are cursor style creation and rubber banding.
C Full error handling is included.
C
C To compile and link the program :-
C
C VMS -
C       FOR/NOOP MATCH
C       LINK MATCH,SYS$LIBRARY:VSLLIB/LIB
C
C RSX -
C       F77 MATCH,MATCH/-SP=MATCH
C       TKB MATCH,MATCH/-SP=MATCH
C       TKB LB:[1,1]VSLLIB/LB
C       TKB LB:[1,1]F4POTS/LB
C       TKB /
C       TKB VSECT=VV21DA:160000:20000
C       TKB WNDWS=1
C       TKB MAXBUF=512
C       TKB //

        IMPLICIT INTEGER*4 (V)

        INTEGER*4       DRAWIT
        INTEGER*2       ISTAT,LUN


C Initialize VIVID processing.
C Set up the display area to be 4096 bytes long,
C maximum segments to be 10.

        ISTAT = VVXINI (4096, 10)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXINI  ', ISTAT)  ! Check status


C Assign VSV21 device for VSL processing.
C First, set up a logical unit number for all subsequent device access.

        LUN = 1

C Device physical unit 0 will assign to device VVA0:
C                      1 will assign to device VVB0:
C                      2 will assign to device VVC0:  etc.
C Third parameter 1024 sets up a report segment of that size,
C default segment ID Hex 2001.

        ISTAT = VVXASS (0, LUN, 1024)
        IF (ISTAT .NE. 1)  CALL SRERR ('VVXASS  ', ISTAT)  ! Check status

C Create a VIVID instruction segment of length 1000 bytes

        ISTAT = VVMCRS (LUN, '201'X, 1000)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVMCRS1 ', ISTAT)  ! Check status

C Call subroutine to build up the display segment and output the picture
C Supply logical unit number and segment ID.

        ISTAT = DRAWIT (LUN, '201'X)

        IF (ISTAT .NE. 1)  CALL SRERR ('DRAWIT  ', ISTAT)  ! Check status

C Release VSV21 device from VSL processing.

        ISTAT = VVXREL (LUN)

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status


C Release VSV21 processor and free the VSV21 buffers.

        ISTAT = VVXEND ()

        IF (ISTAT .NE. 1)  CALL SRERR ('VVXREL  ', ISTAT)  ! Check status

        STOP

        END

C Draw a picture. Design a star cursor, then with rubber banding, locate
C a new current position on the cursor. Draw a line through the cursor
C and produce a match report. Disable matches and redraw the line. Then
C reenable matches, move the cursor, redraw and produce a second match
C report. At program exit, the two match report packets are printed on
C the console. The first and third lines will not be visible due to
C matches being enabled.

        INTEGER*4 FUNCTION  DRAWIT (LUN, SEGID)

        INTEGER*4       IREP, VVRREP
        INTEGER*2       LUN,SEGID,ISTAT,IARR(20),CSTYLE(20)

        DATA CSTYLE / 16,0,0,'8001'X,'4002'X,'2004'X,'1008'X,'810'X,
        1             '420'X,'240'X,'FFFF'X,'FFFF'X,'240'X,'420'X,
        2             '810'X,'1008'X,'2004'X,'4002'X,'8001'X,'8000'X /

        CALL  VVBBGN (SEGID)             ! start of segment
        CALL  VVCINI (-1)                ! initialise all
        CALL  VVACLS ('FFFF'X)           ! clear screen to white
        CALL  VVGMOD (1,0)               ! drawing mode - foreground only
        CALL  VVGFCL (5)                 ! foreground color blue
        CALL  VVGBCL (15)                ! background color white
        CALL  VVBPMD (1)                 ! array mode
        CALL  VVICUS (CSTYLE)            ! create cursor '*'
        CALL  VVICUS (1)                 ! cursor visible
        CALL  VVDMOV (10,10)             ! move abs
        CALL  VVTDRP (%REF('MOVE CURSOR THEN HIT A SWITCH '),15)
        CALL  VVIRUB (1)                 ! rubber band from current point
        CALL  VVISWE (7)                 ! enable switch reports
        CALL  VVIWSW (7)                 ! wait on a switch
        CALL  VVDMOV (10,10)             ! move abs
        CALL  VVASCL (0,'FFFF'X,'FFFF'X,630,20) ! selective clear
        CALL  VVCDWT (180)              ! wait 3 seconds
        CALL  VVDMTC ()                 ! move to cursor
        CALL  VVIMTE (10)               ! enable matches
        CALL  VVBPMD (0)                ! non array mode
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVDLIN (60,60)            ! draw a line (get a match)
        CALL  VVIMTD ()                 ! disable matches
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (10,10)            ! move abs
        CALL  VVBPMD (1)                ! array mode
        CALL  VVTDRP (%REF('MOVE CURSOR THEN HIT A SWITCH '),15)
        CALL  VVIWSW (7)                ! wait on a switch
        CALL  VVDMOV (10,10)            ! move abs
        CALL  VVASCL (0,'FFFF'X,'FFFF'X,630,20) ! selective clear
        CALL  VVCDWT (180)              ! wait 3 seconds
        CALL  VVDMTC ()                 ! move to cursor
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVBPMD (0)                ! non array mode
        CALL  VVDLIN (60,60)            ! draw a line (no match)
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (10,10)            ! move abs
        CALL  VVBPMD (1)                ! array mode
        CALL  VVTDRP (%REF('MOVE CURSOR THEN HIT A SWITCH '),15)
        CALL  VVIWSW (7)                ! wait on a switch
        CALL  VVDMOV (10,10)            ! move abs
        CALL  VVASCL (0,'FFFF'X,'FFFF'X,630,20) ! selective clear
        CALL  VVCDWT (180)              ! wait 3 seconds
        CALL  VVDMTC ()                 ! move to cursor
        CALL  VVIMTE (10)               ! enable matches
        CALL  VVBMOD (1)                ! relative mode
        CALL  VVBPMD (0)                ! non array mode
        CALL  VVDLIN (60,60)            ! draw a line (second match)
        CALL  VVCJMP (2)                ! jump over the pale blue clear
        CALL  VVACLS ('DDDD'X)          ! clear screen to pale blue (don't)
        CALL  VVIMTD ()                 ! disable matches
        CALL  VVBMOD (0)                ! absolute mode
        CALL  VVDMOV (10,10)            ! move abs
        CALL  VVBPMD (1)                ! array mode
        CALL  VVTDRP (%REF('END OF MATCH/CURSOR TESTING '),14)
        CALL  VVBEND ()                 ! end of segment

C Now execute the segment

        CALL  VVEEXE (LUN,SEGID,32000)  ! execute segment
        CALL  VVRSTA (LUN, DRAWIT, IDUM) ! get execute status

C
C Read Match Parameters Reports.
C
105     IREP = VVRREP (LUN,64,IARR,20)
        IF (IREP .EQ. 1) THEN
            PRINT *,'Match Report :'
            PRINT *,' ',IARR(3),' ',IARR(4),' ',IARR(5)
            PRINT *,' ',IARR(6),' ',IARR(7),' ',IARR(8)
            GOTO 105
        ELSE IF (IREP .GE. 0) THEN
            CALL SRERR ('VVRREP  ',IREP)
        ENDIF

        RETURN
        END

C Error handling.
C Routine prints function and error code.

        SUBROUTINE  SRERR (ISRNAM, ISTAT)       ! Handle errors

        CHARACTER*8        ISRNAM
        INTEGER*2           ISTAT

1000    FORMAT  (/, ' **** Failed: routine ', a8,
        1        '   Status = ', I7, ' ****', /)
```

```
      WRITE (5, 1000) ISRNAM, ISTAT
      STOP
      END
```

Part IV Low Level Interface - VIVID Instruction Set

This section describes how to develop graphics applications with the VSV21, and build pictures using the VIVID Instruction Set and QIOs.

6 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. 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 VMS/MicroVMS. The format of a VIVID QIO call depends on:

- The host operating system:
  - VMS/MicroVMS
  - RSX-11M-PLUS or Micro/RSX
- 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 6.2.

6.1 THE QIO FUNCTIONS AND PARAMETERS

Each of the VIVID QIO functions is described in this section. For each function, the version for VMS/MicroVMS is given first, followed by the RSX-11M-PLUS and Micro/RSX version, each with its associated parameters.

Note: **The functions which attach and detach the VSV21 are not used under VMS/MicroVMS.**

The contents of the I/O status block and the error and warning codes are given in Section 6.2.

An example of a VIVID MACRO-32 program which includes QIO calls is given in Chapter 7.

6.1.1 Allocate Display Area - VSV$_ALLOCATE and IO.ADA

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 VMS/MicroVMS, the allocated display is a shared global section. Applications using this allocated display area must map the section in identical virtual address space.

VMS/MicroVMS

Function:
VSV$_ALLOCATE

Hex Value:
3C

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

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.

6.1.2 Attach VSV21 Device - IO.ATT

This function attaches a VSV21 unit to the task.

VMS/MicroVMS

This function is not used under VMS/MicroVMS.

RSX-11M-PLUS and Micro/RSX

Function:
IO.ATT

Octal value:
1400

Parameters:
None

6.1.3 Define Reporting - VSV$_DEFREP and IO.DRP

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 VMS/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.

VMS/MicroVMS

Function:
VSV$_DEFREP

Hex value:
34

Parameters:
pl = reporting segment ID
p2 = reporting mask
p3 = mailbox channel

RSX-11M-PLUS and Micro/RSX

Function:
IO.DRP

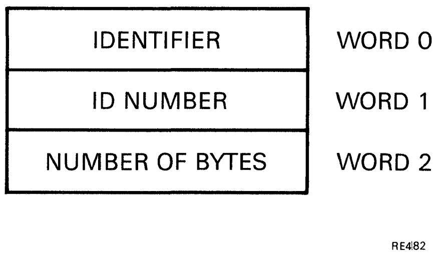
Octal value:
10400 VSVg_item>(Parameters:)
pl = reporting segment ID
p2 = reporting mask
p3 = AST address

6.1.4 Define Segment - VSV$_DEFSEG and IO.DFS

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 6–1). 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 6–1 Format of the First Three Words of a Defined Segment



VMS/MicroVMS

Function:
VSV$_DEFSEG

Hex value:
3A

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

RSX-11M-PLUS and Micro/RSX

Function:
IO.DFS

Octal value:
5400

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

6.1.5 Delete Segment - VSV$_DELSEG and IO.DSG

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

If the segment number is 0 then ALL of the segments of the specific type are deleted. See Section 3.3 for details of segment types.

VMS/MicroVMS

Function:
VSV$_DELSEG

Hex value:
2F

Parameters:
p1 = segment ID

RSX-11M-PLUS and Micro/RSX

Function:
IO.DSG

Octal value:
11000

Parameters:
p1 = segment ID

6.1.6 Detach VSV21 Device - IO.DET

This function detaches the VSV21 unit from the task.

MicroVMS

This function is not used under VMS/MicroVMS.

RSX-11M-PLUS and Micro/RSX

Function:
IO.DET

Octal value:
2000

Parameters:
None

6.1.7 Load Segment - VSV$_LOADSEG and IO.LSG

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.

VMS/MicroVMS

Function:
VSV$_LOADSEG

Hex value:
3B

Parameters:
p1 = virtual address of segment
p2 = number of bytes in transfer block
p3 = segment ID
p4 = block sequence number

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 = segment ID
p4 = block sequence number

6.1.8 Read Data - VSV$_READDATA and IO.RED

VMS/MicroVMS

Function:
VSV$_READDATA

Hex value:
38

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

RSX-11M-PLUS and Micro/RSX

Function:
IO.RED

Octal value:
6400

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

6.1.9 Release Display Area - VSV$_RELEASE and IO.RDA

This releases a display list area which has been allocated using the Allocate function (see VSV$_ALLOCATE in Section 6.1). References to host segments are deleted and all display list processing stops.

VMS/MicroVMS

Function:
VSV$_RELEASE

Hex value:
3D

Parameters:
None

RSX-11M-PLUS and Micro/RSX

Function:
IO.RDA

Octal value:
10000

Parameters:
None

6.1.10 Resume Execution - VSV$_CONTINUE and IO.REX

Using this function, display list execution is resumed at the next instruction. The parameter p1 is optional; the default value is 5 seconds. If display list processing terminates with an error condition, resumption of processing causes an error.

VMS/MicroVMS

Function:
VSV$_CONTINUE

Hex value:
3F

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

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

6.1.11 Start Segment Execution - VSV$_STARTSEG and IO.SSE

This function starts the execution of a single predefined segment. The time-out parameter is optional; the default value is 5 seconds.

VMS/MicroVMS

Function:
VSV$_STARTSEG

Hex value:
3E

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

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 VSV$_DEFREP in Section 6.1.

6.1.12 Stop Display List Execution - VSV$_STOP and IO.STP

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.

VMS/MicroVMS

Function:
VSV$_STOP

Hex value:
35

Parameters:
None

RSX-11M-PLUS and Micro/RSX

Function:
IO.STP

Octal value:
3400

Parameters:
None

6.1.13 Write Data - VSV$_WRITEDATA and IO.WRT

VMS/MicroVMS

Function:
VSV$_WRITEDATA

Hex value:
39

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

RSX-11M-PLUS and Micro/RSX

Function:
IO.WRT

Octal value:
7000

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

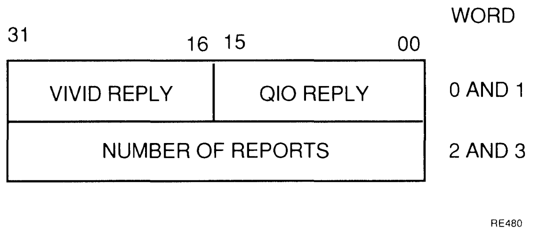
6.2 QIO STATUS REPLIES

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

6.2.1 QIO Replies from VMS/MicroVMS

The contents of the I/O status block are given in Figure 6–2.

Figure 6–2 Contents of VMS/MicroVMS I/O Status Block



The QIO reply word will 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:

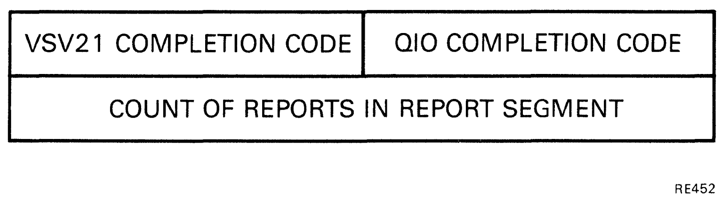
0	-	Normal completion
128	-	Stop acknowledge
129	-	Maximum matches reached
130	-	VIVID error

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.

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

The contents of the I/O status block are shown in Figure 6–3.

Figure 6–3 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
242	-94	IE.PNS	-	Partition/Region not in system
254	-84	IE.ALC	-	Allocation failure define segment
361	-15	IE.ABO	-	QIO aborted
366	-8	IE.DAA	-	Device already attached
372	-6	IE.SPC	-	Illegal user buffer
376	-2	IE.IFC	-	Illegal function code

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.

6.2.3 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.
133	-	Repeat stack overflow
134	-	END_REPEAT found when no matching REPEAT instruction

135	-	Segment already exists
136	-	See Chap 13, LOAD_CHAR_CELL instruction
138	-	Segment is not an attribute segment

7 GETTING STARTED WITH VIVID I/O

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's Guide.*

7.1 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 modules must be downloaded from the host:

- Kernel - controls VSV21 operation
- Pointing device driver - controls joystick, trackball or other devices
- VIVID interpreter - translates display list into a picture

7.1.1 VMS/MicroVMS

The required modules are loaded using the VSV Command Program (VSVCP) described in the *VSV21 User's Guide.* Run VSVCP by entering the following:

VSVCP

This returns the prompt *VSVCP>*. Enter the following commands to load the required modules on the first VSV21:

- *VSVCP> LOAD KERNEL VVA0*
- *VSVCP> LOAD JOY_STICK VVA0*
  (or *DEC_TABLET or PENNY_GILES*)
- *VSVCP> LOAD TRANSPARENT VVA0*
  (If printer is to be used)
- *VSVCP> LOAD VIVID VVA0*
- *VSVCP> EXIT*
  (or CTRL/Z)

The VSV21 is now capable of reduced functionality console emulation. It can interpret VIVID instructions and the VIVID font has been downloaded with the VIVID interpreter.

7.1.2 Compatibility mode on VMS

The required modules are loaded using the VSV Command Program (VSVCP) described in the *VSV21 User's Guide*. Compatibility mode is entered by VSVCP if the logical name VSV$VCP_COMPATIBILITY_MODE has been defined (see the *VSV21 User's Guide*). This is done as follows:

DEFINE VSV$VCP_COMPATIBILITY_MODE 1

Run VSVCP by entering the following:

VSVCP

This returns the following message and prompt:

%VSVCP-I-COMPAT, Using VSV21 Compatibility mode
VCP>

Enter the following commands to load the required modules on the first VSV21:

- *VCP> LOAD KERNEL*
- *VCP> LOAD JSTICK*
  (or *DECTAB or PGSTICK*)
- *VCP> LOAD TRANSP*
  (If printer is to be used)
- *VCP> LOAD VIVID*
- *VCP> CTRL/Z*

The VSV21 is now capable of reduced functionality console emulation. It can interpret VIVID instructions and the VIVID font has been downloaded with the VIVID interpreter.

7.1.3 RSX-11M-PLUS and Micro/RSX

The required modules are loaded using the VSV Command Program (VCP) described in the *VSV21 User's Guide*. Run VSVCP by entering the following:

RUN $VCP

This returns the prompt *VCP>*. Enter the following commands to load the required modules on the first VSV21:

- *VCP> LOAD KERNEL*
- *VCP> LOAD JSTICK*
  (or *DECTAB or PGSTICK*)
- *VCP> LOAD TRANSP*
  (If printer is to be used)
- *VCP> LOAD VIVID*
- *VCP> CTRL/Z*

The VSV21 is now capable of console emulation with reduced functionality. It can interpret VIVID instructions and the VIVID font has been downloaded with the VIVID interpreter.

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 the *VSV21 User's Guide.*

7.2 DEFINING AND EXECUTING A DISPLAY LIST

This section uses a program example to describe how to write a Macro-32 program under VMS/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
        $IODEF
;        Set up I/O function names if necessary. A VIVID application
;        program usually contains the following QIOs:
;
;        RSX and VMS    SECTION      ACTION
;        QIO FUNCTIONS  REFERENCE
;
;
;        IO.ADA or      6.1.1        Allocates a display area for
;       VSV$_ALLOCATE                segments in host memory
;
;        IO.DEF or      6.1.4        Defines a segment by entering its
;       VSV$_DEFSEG                  address and length on the VSV21
;
;        IO.SSE or      6.1.11        Starts execution of a segment
;        VSV$_STARTSEG
;
;
;
VSV$_ALLOCATE == ^X3C
VSV$_DEFSEG   == ^X3A
VSV$_DEFREP   == ^X34
VSV$_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 8 through 16.
;
;        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  /VVA0:/                     ; 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.
;
        $ASSIGN_S       CHAN=CHAN,-
                        DEVNAM=DEV
;
;
;        Allocate a display area for all the concurrent segments
;
;        Use the IO.ADA or VSV$_ALLOCATE function and supply the address
;        of the segment and total length of the display area as
;        parameters
;
        $QIOW_S CHAN=CHAN,-
                FUNC=#VSV$_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 VSV$_DEFSEG function and supply the segment
;        address and length as parameters
;
        $QIOW_S CHAN=CHAN,-                 ; channel number
                FUNC=#VSV$_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 6.2
;
        BLBC    R0,1$
        MOVW    IOSB,R0
        BLBS    R0,2$
```

```
1$:     BRW     EXIT
2$:
;        Define the reporting segment
;
;        Use the IO.DEF or VSV$_DEFSEG function and supply the segment
;        address and length as parameters
;
        $QIOW_S CHAN=CHAN,-             ; channel number
                FUNC=#VSV$_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 VSV$_DEFREP function to define the segment
;        as a reporting segment
;
                $QIOW_S CHAN=CHAN,-     ; channel number
                FUNC=#VSV$_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 VSV$_STARTSEG function and supply the location
;        of the segment ID and the required timeout value as parameters.
;
        $QIOW_S CHAN=CHAN,-
                FUNC=#VSV$_STARTSEG,-
                IOSB=IOSB,-
                P1=@ADA+2,-             ; segment ID from 2nd. 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
```

8 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.

CALL_SEGMENT

Executes the identified segment from the host or VSV21 memory.

ARGUMENTS ***Opcode:*** *7*
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
```

CREATE_SEGMENT

Creates a segment in the VSV21 memory.

ARGUMENTS ***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.
Warning if segment ID is not valid.

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

EXAMPLE:

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

DISPLAY_END_REPEAT

Marks the end of a repeatable loop.

ARGUMENTS ***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 0.,16.        ;length and opcode
```

DISPLAY_REPEAT

Marks the start of a loop.

ARGUMENTS ***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
```

DISPLAY_WAIT

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

ARGUMENTS ***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
```

DUMP_ATTRIBUTES

Saves the current set of attributes in a specified segment.

ARGUMENTS ***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 already defined on the host.

EXAMPLE:

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

INITIALIZE

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

ARGUMENTS ***Opcode:*** ***6***
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
```

JUMP_RELATIVE

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

ARGUMENTS

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
```

NO_OPERATION

No operation is performed and nothing is changed.

ARGUMENTS ***Opcode:*** ***11***
Length: ***0***

FORMAT: ***NO_OPERATION***

PARAMETERS: ***None***

END POSITION: The drawing position is not changed.

ERRORS: Error if length not equal to zero.

EXAMPLE:

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

RECOVER_ATTRIBUTES

Reads the specified attributes from the specified segment.

ARGUMENTS ***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 C.

EXAMPLE:

```
.BYTE  2.,122.         ;length and opcode
.WORD  ^X030A          ;segment class 3, number 10
.WORD  10              ;octal mask for drawing
```

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.

ARGUMENTS

Opcode: ***9***
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:

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 C.

EXAMPLE:

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

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, CHAPTER 8).

ARGUMENTS

Opcode:	***8***
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 C. This includes drawing and cursor positions.

EXAMPLE:

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

SEGMENT_RETURN

Marks the end of an instruction segment.

ARGUMENTS ***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
```

START_ATTRIBUTES_DATA

Identifies the segment contents as attributes data.

ARGUMENTS

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
```

START_FONT

Identifies the segment contents as a font.

ARGUMENTS ***Opcode:*** ***2***
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: The error "START_FONT in instruction list" occurs when this instruction is found in an instruction list, irrespective of the number of parameters.

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
```

START_INSTRUCTION_LIST

Identifies the segment contents as display instructions.

ARGUMENTS ***Opcode:*** ***1***
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
```

START_KEYBOARD_DATA

Identifies the segment contents as keyboard input.

ARGUMENTS ***Opcode:*** ***4***
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: The error "START_KEYBOARD_DATA in instruction list" occurs when this instruction is found in an instruction list, irrespective of the number of parameters.

NOTES: This is set up automatically by the ACCEPT_KEYBOARD_INPUT instruction (Chapter 15).

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
```

START_PIXEL_DATA

Identifies the segment contents as pixel data.

ARGUMENTS ***Opcode:*** ***3***
Length: ***6***

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: The error "START_PIXEL_DATA in instruction list" occurs when this instruction is found in an instruction list, irrespective of the number of parameters.

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 6,3           ;length and op-code

.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 Y dim= 2.0 times
                    ;monitor Y dim
```

START_REPORT_DATA

Identifies the segment contents as report data.

ARGUMENTS ***Opcode:*** ***5***
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: The error "START_REPORT_DATA in instruction list" occurs when this instruction is found in an instruction list, irrespective of the number of parameters.

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 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
```

STOP_DISPLAY

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

ARGUMENTS ***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
```

9 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.

DRAWING_MAGNIFICATION

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

ARGUMENTS

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 Chapter 9).

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
```

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.

ARGUMENTS

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
```

DRAWING_TRANSLATION

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

ARGUMENTS ***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.,22.      ;length and opcode
.WORD 50.         ;relative X coordinate of origin
.WORD 200.        ;relative Y coordinate of origin
```

DRAWING_VAS

This disables DRAWING_MAGNIFICATION and DRAWING_TRANSLATION. Subsequent input is in VAS units.

ARGUMENTS ***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
```

SCREEN_DIMENSIONS

Defines the screen dimensions in logical pixels.

ARGUMENTS ***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, Chapter 9) and display processing stops.

NOTES:

The valid sets of values for the parameters width, height 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 X 480 logical pixels.

EXAMPLE:

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

SET_VIEWPORT

This instruction defines a screen area to which drawing is restricted. The area units are as defined by the SCREEN_DIMENSIONS instruction (Chapter 9).

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.

ARGUMENTS

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
```

SET_WINDOW

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

ARGUMENTS ***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
```

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.

ARGUMENTS ***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
```

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.

ARGUMENTS

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
```

10 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.

AREA_TEXTURE

This defines a cell containing the area texture pattern.

ARGUMENTS ***Opcode:*** ***34***
Length: ***1 + number of rows***

FORMAT: ***AREA_TEXTURE nbit, patt1 [, patt2, ... pattn]***
where n = 1-16

PARAMETERS: ***nbit***
number of bits in bit pattern (that is, within a row)

pattn
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. 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 5.,34.    ;length and opcode (4 rows)
.WORD 12.       ;12 bits in pattern per row
                ;
.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  o  oo  o  oo  o  oo  o  oo
                ;    o  oo  o  oo  o  oo  o  oo
                ;   o  oo  o  oo  o  oo  o  oo
                ;  o  oo  o  oo  o  oo  o  oo  o
```

BACKGROUND_COLOR

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

ARGUMENTS ***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 briefly described in the NORMAL_COLORS section. For a full description of the CLUT, see the *VSV21 User's Guide.*

EXAMPLE:

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

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 the *VSV21 User's Guide* as defined by BLINK_COLORS and by BLINK_COUNT.

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.

ARGUMENTS

Opcode: ***30***
Length: ***3 x number of entries or 255***

FORMAT:

BLINK_COLORS bind1, ind1, int1, bind2, ind2, int2 ... bindn, indn, intn

where n = 1-16

PARAMETERS:

bindn
color index in blink colors look-up table (BCLUT)

indn
corresponding CLUT index

intn
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 the *VSV21 User's Guide*.

```
.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
```

BLINK_COUNT

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

ARGUMENTS ***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. VSVCP command file or display segment is recommended for setting up your own defaults (see NORMAL_COLORS).

EXAMPLE:

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

BLINK_TIMING

This sets screen blink timings.

ARGUMENTS ***Opcode:*** ***35***
Length: ***2***

FORMAT: ***BLINK_TIMING norm, blnk***

PARAMETERS:

norm
number of frames of normal colors (range 8-64)

blnk
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
```

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.

ARGUMENTS

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 $\neq$ 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
```

FOREGROUND_COLOR

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

ARGUMENTS

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 the NORMAL_COLORS section. For a fuller description of the CLUT, see the *VSV21 User's Guide.*

EXAMPLE:

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

LINE_TEXTURE

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

ARGUMENTS ***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"
```

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).

ARGUMENTS

Opcode:	***29***
Length:	***2 x number of colors or 255***

FORMAT:

NORMAL_COLORS ind1, int1 [, ind2, int2, ... indn, intn]

where n = 1-16

PARAMETERS:

indn
color index number in CLUT

intn
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 VSVCP 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 the *VSV21 User's Guide.*

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 the *VSV21 User's Guide.*

EXAMPLE:

The color values used in this example are those used in the CLUT example given in the *VSV21 User's Guide*.

```
.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 ^X088          ;cyan
.WORD 3.
.WORD ^X06C          ;turquoise
.WORD 4.
.WORD ^XF00          ;red
.WORD 5.
.WORD ^XA0A          ;magenta
```

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.

ARGUMENTS ***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
```

SCREEN_BLINK

This instruction enables or disables screen blinking.

ARGUMENTS ***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 are alternated with those specified in the BLINK_COLORS command. The interval is specified by the BLINK_TIMING instruction. Blinking continues for the whole screen until it is disabled.

EXAMPLE:

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

11 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 to draw the last pixel.

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.

ARGUMENTS

Opcode:	***44***
Length:	***5 x number of arcs, or 255***

FORMAT:

ARCS_ABS dir1, xcen1, ycen1, xend2, yend2 [,dir2, xcen2, ycen2, xend2, yend2 ... dirn, xcenn, ycenn, xendn, yendn]

where n has no defined limit

PARAMETERS:

dirn
drawing direction
0 : counterclockwise
1 : clockwise

xcenn
X coordinate of center in VAS

ycenn
Y coordinate of center in VAS

xendn
X coordinate of end position in VAS

yendn
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:

$$\sqrt{\Delta XC^2 + \Delta YC^2} \leq \frac{4095}{(MAX(A,B))^2}$$

where

A	=	transformed relative X length in pixels
B	=	transformed relative Y length in pixels
XC	=	transformed value of xcen - xc
YC	=	transformed value of ycen - 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 11-1. The points A, B, C and D are points at which lines of gradient +1 and -1 are tangential to the arc. Figure 11-1 shows a circle of which the arc is a part. The same principle applies to elliptic 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.

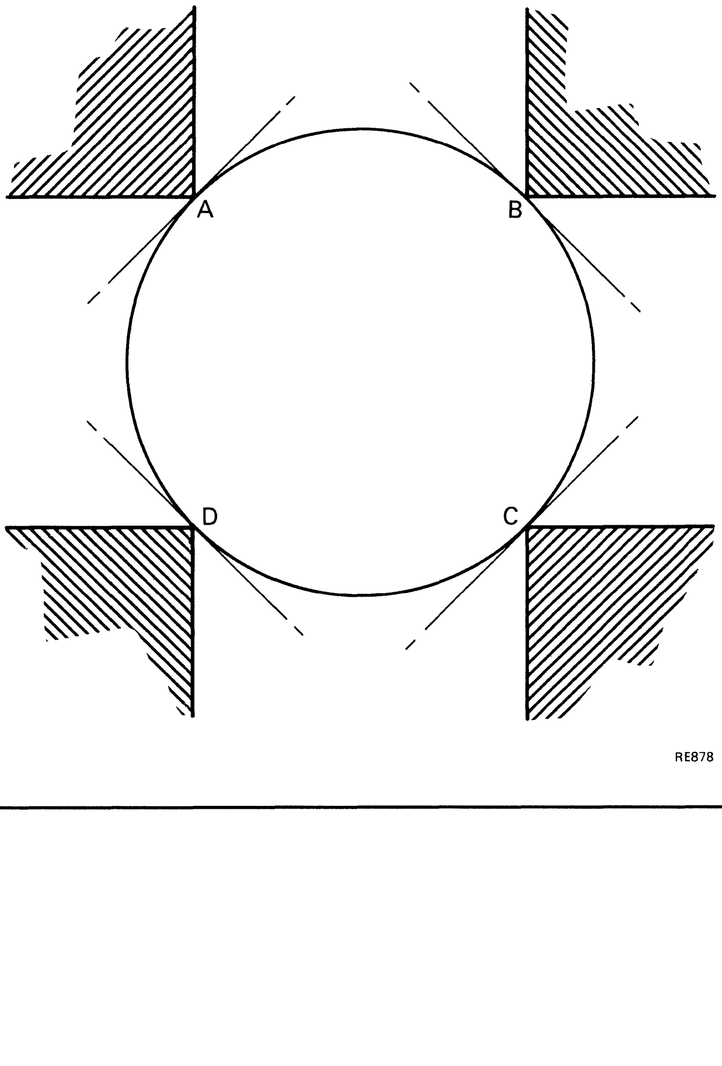
The last pixel is not drawn.

Where a length of 255 is used, the parameter list must be terminated with END_ PARAMETERS (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
```

Figure 11–1 Error Areas for End Points of Circular Arcs



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.

ARGUMENTS

Opcode:	***45***
Length:	***5 x number of arcs, or 255***

FORMAT:

ARCS_REL dir1, xcen1, ycen1, xend2, yend2 [,dir2, xcen2, ycen2, xend2, yend2 ... dirn, xcenn, ycenn, xendn, yendn

where n has no defined limit

PARAMETERS:

dir1

drawing direction
0 : counterclockwise
1 : clockwise

xcen

X displacement of center

ycen

Y displacement of center

xend

X displacement of end position

yend

Y displacement of end position

END POSITION:

As defined by the final coordinates.

ERRORS:

As for ARCS_ABS.

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 (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
```

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.

ARGUMENTS

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:

$$R \leq \frac{4095}{(MAX(A,B))^2}$$

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
```

DOT

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

ARGUMENTS

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
```

ELLIPSE

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

An ellipse is defined by three quantities (Figure 11-3). 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.

ARGUMENTS

Opcode:	***50***
Length:	***3***

FORMAT:

ELLIPSE ax, by, rad

PARAMETERS:

ax
relative horizontal length

bx
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:

$$R \leq \frac{4095}{(MAX(A,B))^2}$$

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

$$B \geq 1.$$

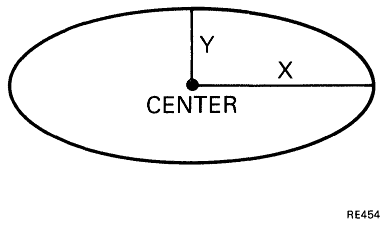
NOTES:

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
```

Figure 11–2 Quantities Used to Define an Ellipse



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.

ARGUMENTS ***Opcode:*** ***46***
Length: ***7 x number of arcs or 255***

FORMAT: ***ELLIPSE_ARCS_ABS dir1, ax1, ay1, xcen1, ycen1, xend1, yend1 [dir2, ax2, ay2, xcen2, ycen2,xend2, yend2 ... dirn,axn, ayn, xcenn, ycenn, xendn, yendn]***

where n has no defined limit

PARAMETERS: ***dir***
= drawing direction
0 : counterclockwise
1 : clockwise

axn
relative X length

ayn
relative Y length

xcenn
X coordinate of center in VAS

ycenn
Y coordinate of center in VAS

xendn
X coordinate of end position in VAS

yendn
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:

$$\sqrt{\Delta XC^2 + \Delta YC^2} \leq \frac{4095}{(MAX(A,B))^2}$$

where

A	=	transformed relative X length in pixels
B	=	transformed relative Y length in pixels
DXC	=	transformed value of (xcen - cpx)
DYC	=	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 ay 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 11-2. The points A, B, C and D are points at which lines of gradient +1 and -1 are tangential to the ellipse.

NOTES:

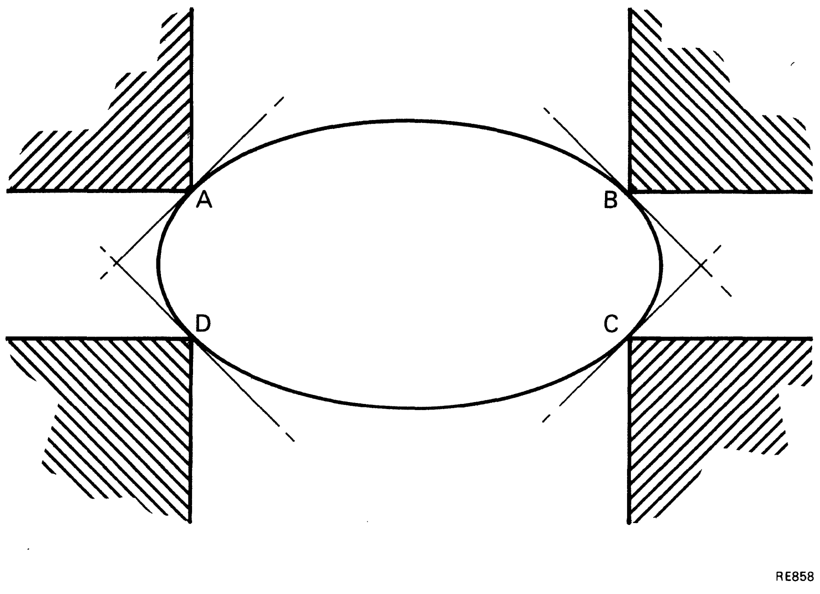
No drawing occurs outside the current viewport, though the drawing position may move outside the viewport. The last pixel is not drawn (see the DOT instruction).

Where a length of 255 is used, the parameter list must be terminated with END_ PARAMETERS (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 11–3 Error Areas for End Points of Elliptic Arcs



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.

ARGUMENTS

Opcode:	***47***
Length:	***7 x number of arcs, or 255***

FORMAT:

ELLIPSE_ARCS_REL dir1, ax1, ay1, dxc1, dyc1, dxe1, dye1 [dir2, ax2, ay2, dxc2, dyc2, dxe2, dye2, ...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

***ay*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

dyen
Y displacement of end position in VAS

END POSITION:

As defined by the final coordinates.

ERRORS:

As for ELLIPSE_ARCS_ABS.

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 (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
```

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.

ARGUMENTS

Opcode:	***40***
Length:	***2 x number of lines, or 255***

FORMAT:

LINES_ABS x1, y1[, x2, y2,... xn, yn]
where n has no defined limit

PARAMETERS:

xn
X coordinate for the end of the line

yn
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
```

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.

ARGUMENTS ***Opcode:*** ***41***
Length: ***2 x number of lines or 255***

FORMAT: ***LINES_REL dx1, dy1 [, dx2, dy2 ... dxn, dyn]***
where n has no defined limit

PARAMETERS: ***dxn***
X displacement for the next end vector

dyn
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
```

MOVE_ABS

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

ARGUMENTS

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
```

MOVE_REL

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

ARGUMENTS ***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
```

MOVE_TO_CURSOR

This instruction gives a move to the current cursor position.

ARGUMENTS ***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
```

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 this chapter.

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

ARGUMENTS

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

FORMAT:

POLYMARKS_ABS ichar, x1, y1 [, x2, y2,... xn, yn]
where n has no defined limit

PARAMETERS:

ichar
index of character required

xn
X coordinate in VAS

yn
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
```

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 13.

ARGUMENTS

Opcode: ***43***

Length: ***1 + (2 x number of points), or 255***

FORMAT:

POLYMARKS_REL ichar, dx1, dy1[, dx2, dy2... dxn, dyn]

where n has no defined limit

PARAMETERS:

ichar
index number of character required

dxn
X displacement

dyn
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.

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 5.,43.          ;length and opcode
.WORD 9               ;index of character
.WORD 100.            ;X displacement
.WORD 200.            ;Y displacement
.WORD -50.            ;X displacement
.WORD 20.             ;Y displacement
```

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.

ARGUMENTS

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
```

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.

ARGUMENTS

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
```

12 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.

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.

ARGUMENTS

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
```

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.

ARGUMENTS

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.

DESCRIPTION

Example:)

```
.BYTE 2.,57.        ;length and opcode
.WORD 200.          ;xd=200
.WORD 100.          ;yd=100
```

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.

ARGUMENTS

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 (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
```

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.

ARGUMENTS

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
```

13 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 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.

    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 9) 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.

CELL_MAGNIFICATION

This instruction defines the horizontal and vertical cell magnification, in terms of pixels or relative magnification.

ARGUMENTS

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
```

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.

ARGUMENTS ***Opcode:*** ***70***
Length: ***2***

FORMAT: ***CELL_MOVEMENT xd, yd***

PARAMETERS:

xd
horizontal displacement

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.

No drawing is performed.

EXAMPLE:

```
.BYTE 2.,70.        ;length and opcode
.WORD 1.            ;X movement
.WORD 2.            ;Y movement
```

CELL_OBLIQUE

This defines whether subsequent cells are to be drawn rectangularly, or in italic (45-degree slope) form.

ARGUMENTS

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
```

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.

ARGUMENTS ***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
```

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.

ARGUMENTS

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.

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
```

DRAW_CHARS

This displays the characters specified by each index in the parameter list. There is one index per word.
See also DRAW_PACKED_CHARS.

ARGUMENTS ***Opcode:*** ***71***
Length: ***number of characters, or 255***

FORMAT: ***DRAW_CHARS ind1[, ind2, ... indn]***
where n has no defined limit

PARAMETERS: ***indn***
index to cell in font

END POSITION: Defined by the CELL_MOVEMENT instruction. 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
```

DRAW_PACKED_CHARS

This instruction displays the characters specified by each index in the parameter list. Indices are packed two per parameter word.
See also DRAW_CHARS.

ARGUMENTS ***Opcode:*** ***72***
Length: ***(number of chars+1)/2 or 255***

FORMAT: ***DRAW_PACKED_CHARS i1 j1 [,i2 j2, ...in jn]***
where n has no defined limit

PARAMETERS: ***in, jn = 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. 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
```

INITIALIZE_FONT

This instruction initializes the specified segment as a font, irrespective of the segment contents.

ARGUMENTS ***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
```

LOAD_CHAR_CELL

A character cell is loaded to the current font from the pixel data given as parameters.

ARGUMENTS ***Opcode:*** ***65***
Length: ***255, or number of rows + 1***

FORMAT: ***LOAD_CHAR_CELL ind, irow1 [, irow2, ... irown]***
where n = 1-16

PARAMETERS: ***ind***
cell index

irown
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. Any rows at the top of the cell not provided are filled solid or blank, depending on font initialization (Chapter 11).

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
```

SET_FONT

This sets the current font to the identified font segment. This font is used for subsequent VIVID instructions which access fonts.

ARGUMENTS

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 VSVCP command.
- 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
```

14 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.

CLEAR_SCREEN

This instruction clears the display.

ARGUMENTS ***Opcode:*** ***76***
Length: ***0 or 1***

FORMAT: ***CLEAR_SCREEN [patt]***

PARAMETERS: ***patt***
list of color indices for screen

END POSITION: The current position is reset to the current window origin.

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 (Chapter 8) for a description of these indices.

The viewport is ignored.

EXAMPLE:

```
.BYTE 1.,76.        ;length and opcode
.WORD ^X1111        ;clear to color 1
```

CLEAR_VIEWPORT

This clears the viewport to the current background color.

ARGUMENTS ***Opcode:*** *77*
Length: *0*

FORMAT: ***CLEAR_VIEWPORT***

PARAMETERS: None

END POSITION: The current position is set to the window origin.

ERRORS: None

EXAMPLE:

```
.BYTE 0.,77.     ;length and opcode
```

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.

ARGUMENTS

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 NOTES ON THE COPY INSTRUCTIONS.

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
```

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.

ARGUMENTS

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 NOTES ON THE COPY INSTRUCTIONS.

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
```

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 14-1). The conventions adopted in Table 14-1 are as follows:

+X	=	left to right
-X	=	right to left
+Y	=	upwards
-Y	=	downwards

Table 14–1 Order of Pixel Write to Destination Area by amod Value

Value of amod	Scan direction 1st	Scan direction 2nd
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		Direction
xdim	ydim	
+	+	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
    V       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
```

Figures 14-1 and 14.2 illustrate 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 14–1 Effect of Parameter Values and Signs on Orientation of Copied Picture; amod range 0 - 7 (1st half)

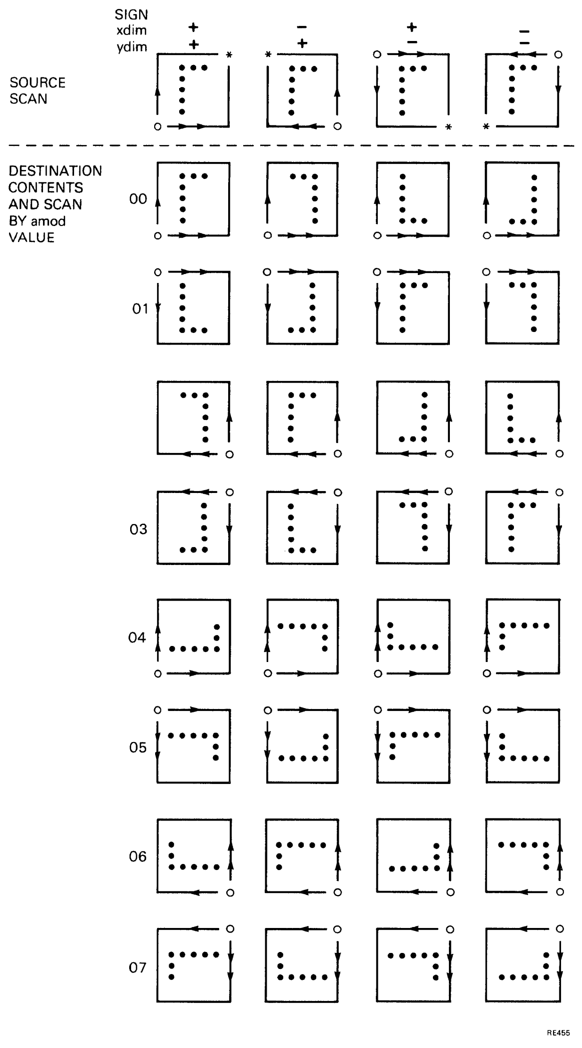
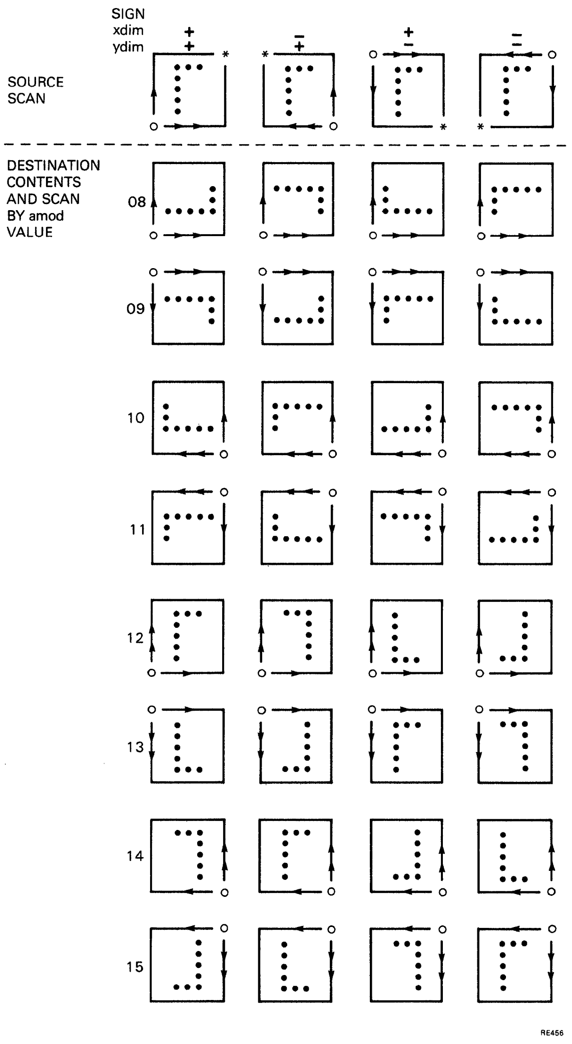


Figure 14–2 Effect of Parameter Values and Signs on Orientation of Copied Picture; amod range 8 - 15 (2nd half)



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 Chapter 3, Font Segment and Pixel Data Segment, 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.

ARGUMENTS

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 given 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
```

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.

ARGUMENTS

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 given 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
```

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.

ARGUMENTS

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 shown 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
```

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.

ARGUMENTS

Opcode:	***81***
Length:	***1***

FORMAT:

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 of the wrong type
- Segment is too small for its defined contents

Warning given 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.

Matches may be detected.

EXAMPLE:

```
.BYTE 1.,81.     ;length and opcode
.WORD ^XD02      ;segment class 13, number 2
```

SCROLL_VIEWPORT

This moves the data within the viewport. The data is moved by the indicated displacement. Data falling outside the viewport is lost.

ARGUMENTS

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
```

SELECTIVE_CLEAR

The specified logical operation is performed on the rectangular area whose opposite vertices are defined by the 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.

ARGUMENTS

Opcode:	***84***
Length:	***4 or 5***

FORMAT:

SELECTIVE_CLEAR mode, mask, [patt,] dxw, dyp

PARAMETERS:

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

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 given 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 parameter mask 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 parameter pattern 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
```

15 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.

Note: Use of pointing devices requires the appropriate device driver to be downloaded prior to program execution.

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.

ARGUMENTS

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 (Chapter 13). 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 chend 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
.BYTE 22.,17.      ;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
```

CURSOR_STYLE

Set the cursor to the specified pixel data, or to one of the default cursor styles.

ARGUMENTS ***Opcode:*** ***90***
Length: ***1 or 3 + number of rows, or 255***

FORMAT: ***CURSOR_STYLE ccode [, dxp, dyp, row1, row2 ... rown]***

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

rown

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 immediately replaced 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 VSVCP (see the *VSV21 User's Guide*).

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.

CURSOR_VISIBILITY

Defines whether or not the cursor is visible.

ARGUMENTS ***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: When visible, the cursor is restricted to the screen bounds.

EXAMPLE:

```
.BYTE 1.,92.        ;length and opcode
.WORD 1.            ;cursor visible
```

MATCH_DISABLE

Disables a match.

ARGUMENTS ***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.

EXAMPLE:

```
.BYTE 0.,98.          ;length and opcode
```

MATCH_ENABLE

When subsequent drawing intersects the cursor position, a report (Chapter 16, MATCH_INTERRUPT REPORT PACKET) is sent to the host. Following this instruction, invisible drawing continues until the maximum number of matches have been detected.

ARGUMENTS ***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 (that is, drawing invisible), to determine which instruction caused the pixel at the cursor position to be drawn. Each match corresponding to the cursor position may 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
```

POSITION_CURSOR

Sets the cursor to the specified position. The cursor is restricted by the screen boundaries.

ARGUMENTS

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
```

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 15-1) or as the ends of the diagonal of a rectangle (rectangular rubber band; Figure 15-2).

ARGUMENTS

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 15-1 and 15-2, x,y is the base point and two successive cursor positions are shown.

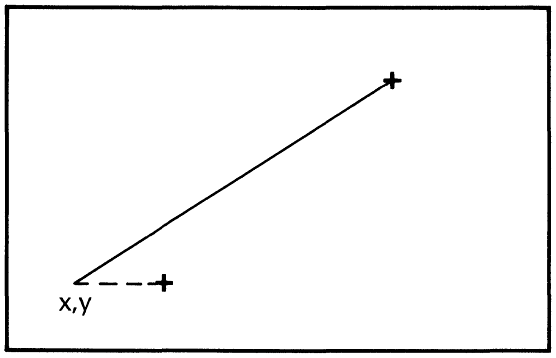
This instruction does not change cursor visibility. If you use the combination of long cross-hair cursor and rectangular rubber band, two sides of the cursor or band will not be seen.

For optimum rendition, it may be necessary to adjust the pointing device sensitivity factors by using the VSVCP. (See the *VSV21 User's Guide.*)

EXAMPLE:

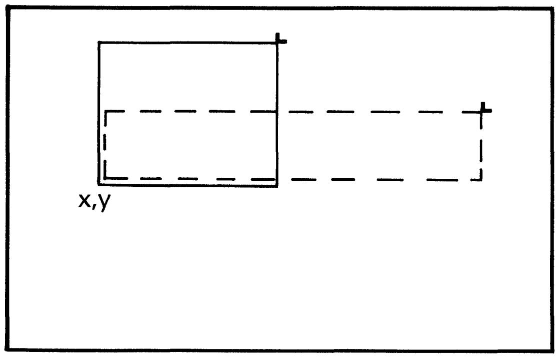
```
.BYTE 3.,93.        ;length and opcode
.WORD 1.            ;linear rubber band
.WORD 100.          ;X position of base point
.WORD 50.           ;Y position of base point
```

Figure 15–1 Linear Rubber Band



RE478

Figure 15–2 Rectangular Rubber Band



RE479

START_KEYBOARD_INPUT

Keyboard input for AST processing is begun. The input is terminated by a STOP_KEYBOARD_INPUT instruction.

ARGUMENTS ***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 a VMS/MicroVMS mailbox, or an RSX AST, 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 VMS/MicroVMS mailbox, or RSX AST, (see Chapter 16, KEYBOARD_INPUT REPORT PACKET). Character codes dispatched are exactly as for host serial input.

No drawing occurs.

EXAMPLE:

```
.BYTE 0.,99.        ;length and opcode
```

STOP_KEYBOARD_INPUT

Stops the keyboard input for AST processing. Subsequent input is by means of the serial interface, if it is connected.

ARGUMENTS ***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 via the host serial port, if connected.

EXAMPLE:

```
.BYTE 0.,100.       ;length and opcode
```

SWITCH_DISABLE

Disables pointing device reporting.

ARGUMENTS	***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
```

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 16.

ARGUMENTS

Opcode: ***94***
Length: ***1***

FORMAT:

SWITCH_REPORT_ENABLE mask

PARAMETERS:

mask

If switch mask is 0, a report is generated for movement with any device input (for example, pushing one of the buttons).
If the switch mask is NOT 0 and if the following bits are set, a report will be generated:

Bit Number (from 0)		
0	-	Left button
1	-	Middle button
2	-	Right button
3	-	If no DECtablet - set to 0 If DECtablet - set to 1 for response to 4th switch
4 .. 15	-	Must be zeros

END POSITION:

The current drawing position is not changed. The cursor position is moved according to calculation from pointing device input.

ERRORS:

None.

NOTES:

The switch mask is ANDed with 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 Chapter 16, SWITCH_INTERRUPT REPORT PACKET.

EXAMPLE:

```
.BYTE 1.,94.        ;length and opcode
.WORD ^B0110        ;mask value
```

WAIT_SWITCH

Waits for a switch interrupt before continuing with the next VIVID instruction.

ARGUMENTS ***Opcode:*** ***103***
Length: ***1***

FORMAT: ***WAIT_SWITCH mask***

PARAMETERS: ***mask***
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 smask.

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 ^B0110         ;bits 110 enable switches 2
                     ;and 3
```

16 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:

  REQUEST_REPORT INSTRUCTION

  DRAWING_POSITION REPORT PACKET

  CURSOR_POSITION REPORT PACKET

  CELL_PARAMETERS REPORT PACKET

  GLOBAL_ATTRIBUTES REPORT PACKET

  TRANSFORMATION REPORT PACKET

  SCREEN_FORMAT REPORT PACKET

  FREE SPACE REPORT PACKET

  VSV21_SEGMENTS REPORT PACKET

  HOST_SEGMENTS REPORT PACKET

  VIVID_VERSION REPORT PACKET

  SEGMENT_TRACE REPORT PACKET

- Errors during segment processing

  VIVID_WARNING REPORT PACKET

  VIVID_ERROR REPORT PACKET

- Match interrupts

  MATCH_INTERRUPT REPORT PACKET

  MAXIMUM_MATCHES REPORT PACKET

- Switch interrupts

  SWITCH_INTERRUPTS REPORT PACKET

- Keyboard input

  KEYBOARD_INTERRUPT REPORT PACKET

- Interruption of segment processing

  VIVID_INTERRUPT REPORT PACKET

Reports REQUEST_REPORT INSTRUCTION through to SEGMENT_TRACE REPORT PACKET are generated by report requests. The remaining reports (VIVID_ WARNING REPORT PACKET through to the end of the section) are generated by events.

An introduction to report handling is given in Section 2.12.

REQUEST_REPORT INSTRUCTION

This instruction places the specified report in the current report segment.

ARGUMENTS ***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 given 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	CELL_PARAMETERS
1	CURSOR_POSITION
2	DRAWING_POSITION
3	FREE_SPACE
4	GLOBAL_ATTRIBUTES
5	HOST_SEGMENTS
6	SCREEN_FORMAT
7	SEGMENT_TRACE
8	TRANSFORMATION
9	VIVID_VERSION
10	VSV21_SEGMENTS

EXAMPLE:

```
.BYTE 1.,108.        ;length and opcode
.WORD 8.             ;report number (host segments)
```

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 16–1.

Figure 16–1 Format of Cell Parameters Report Packet

	WORD
CELL_PARAMETERS = 2	0
NUMBER OF PARAMETERS = 12	1
CURRENT FONT SEGMENT ID	2
FONT CELL WIDTH	3
FONT CELL HEIGHT	4
DISPLAY CELL WIDTH	5
DISPLAY CELL HEIGHT	6
DISPLAY CELL X MOVEMENT	7
DISPLAY CELL Y MOVEMENT	8
DISPLAY CELL UNITS CODE	9
CELL X MAGNIFICATION	10
CELL Y MAGNIFICATION	11
CELL OBLIQUE CODE	12
CELL ROTATION CODE	13

RE459

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 13).

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 16–2 and Figure 16–3.

Figure 16–2 VMS Format of Cursor Position Report Packet

CURSOR_POSITION = 1
NUMBER OF PARAMETERS = 5
CURSOR X COORDINATE IN VAS
CURSOR Y COORDINATE IN VAS
SWITCH STATUS WORD
CURSOR X DEVICE COORDINATE
CURSOR Y DEVICE COORDINATE

RE6906

Figure 16–3 RSX Format of Cursor Position Report Packet

	WORD
CURSOR_POSITION = 1	0
NUMBER OF PARAMETERS = 2	1
CURSOR X COORDINATE IN VAS	2
CURSOR Y COORDINATE IN VAS	3

RE458

NOTES:

This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

For VMS systems the returned device coordinate is logical, that is, on a low resolution device the returned height will be in the range 0 to 479.

The Switch Status Word contains the value of the most recently pressed switch.

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 16–4.

Figure 16–4 Format of Drawing Position Report Packet

	WORD
DRAWING POSITION = 0	0
NUMBER OF PARAMETERS = 2	1
DRAWING X COORDINATE IN VAS	2
DRAWING Y COORDINATE IN VAS	3

RE7114

NOTES:

This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

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 16–5.

Figure 16–5 Format of Free Space Report Packet

	WORD
FREE_SPACE = 6	0
NUMBER OF PARAMETERS = 2	1
RESERVED	2
FREE SPACE IN WORDS	3

RE463

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.

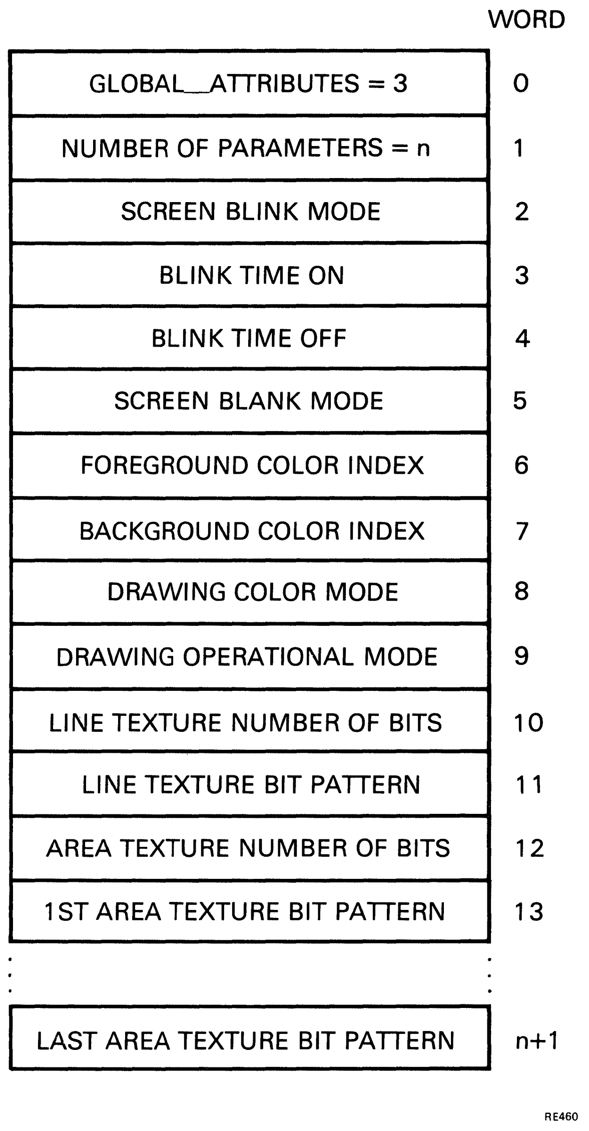
Note that the amount of free space (word 3) is given in words rather than in bytes.

GLOBAL_ATTRIBUTES REPORT PACKET

The report provides the major current global attributes which are not otherwise available by using the VSVCP (VSV21 Control Program). The VSVCP commands are described in the *VSV21 User's Guide*.

FORMAT: The format of the report packet is given in Figure 16–6.

Figure 16–6 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 10).

The maximum report packet size is 29 words.

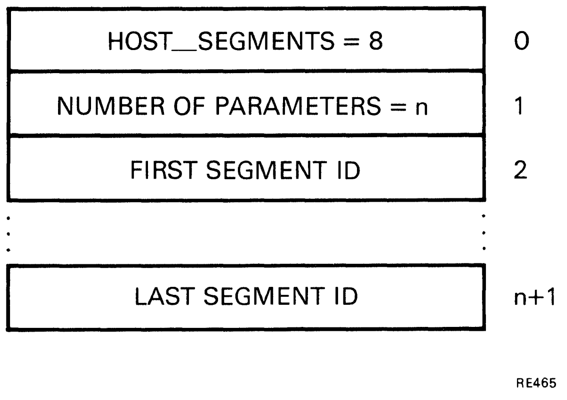
HOST_SEGMENTS REPORT PACKET

This report provides a list of the segments defined in host memory.

FORMAT:

The format of the report packet is given in Figure 16–7.

Figure 16–7 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.

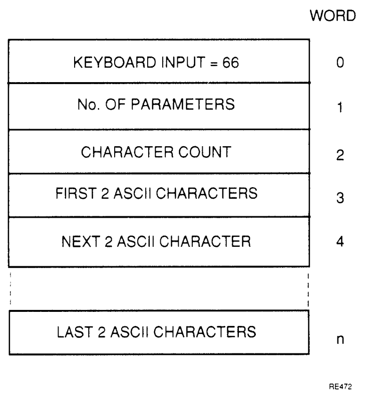
KEYBOARD_INPUT REPORT PACKET

This report provides input from the keyboard to an AST.

FORMAT:

The format of the report packet is given in Figure 16–8.

Figure 16–8 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.

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 16–9.

Figure 16–9 Format of Match Interrupt Report Packet

MATCH_INTERRUPT = 64	0
NUMBER OF PARAMETERS = 6	1
TOP LEVEL SEGMENT ID	2
CURRENT SEGMENT ID	3
CURRENT SEGMENT BYTE OFFSET	4
CURRENT OPCODE	5
DRAWING X COORDINATE IN VAS	6
DRAWING Y COORDINATE IN VAS	7

RE470

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.

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 16–10.

Figure 16–10 Format of Maximum Matches Report Packet

MAXIMUM__MATCHES = 129	0
NUMBER OF PARAMETERS = 4	1
TOP LEVEL SEGMENT ID	2
CURRENT SEGMENT ID	3
CURRENT SEGMENT BYTE OFFSET	4
CURRENT SEGMENT LAST OPCODE	5

RE474

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.

SCREEN_FORMAT REPORT PACKET

The report provides the screen format.

FORMAT:

The format of the report packet is given in Figure 16–11.

Figure 16–11 Format of Screen Format Report Packet

	WORD
SCREEN__FORMAT = 5	0
NUMBER OF PARAMETERS = 2	1
SCREEN WIDTH IN PIXELS	2
SCREEN HEIGHT IN PIXELS	3

RE462

NOTES:

This report is generated only by a REQUEST_REPORT instruction and is provided in the report segment only.

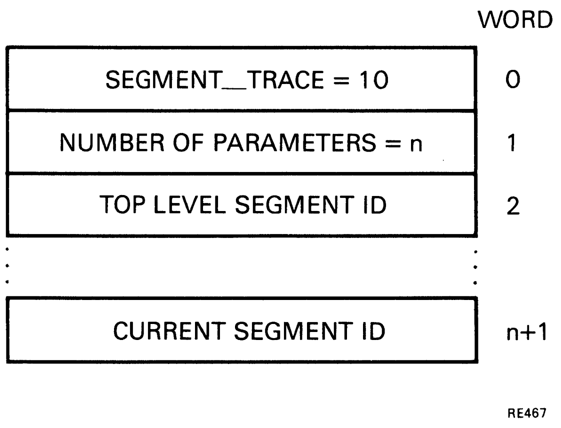
SEGMENT_TRACE REPORT PACKET

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 16–12.

Figure 16–12 Format of Segment Trace 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 calling sequence from the top level down to the segment in which the report request occurs.

The maximum report packet size is 33 words.

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 16–13 and Figure 16–14.

Figure 16–13 VMS Format of Switch Interrupt Report Packet

VIVID REPORT TYPE = 65
NUMBER OF PARAMETERS = 5
CURSOR X COORDINATE
CURSOR Y COORDINATE
SWITCH STATUS WORD
CURSOR X DEVICE COORDINATE
CURSOR Y DEVICE COORDINATE

RE6907

Figure 16–14 RSX Format of Switch Interrupt Report Packet

SWITCH__INTERRUPT = 65	0
NUMBER OF PARAMETERS = 3	1
CURSOR X COORDINATE IN VAS	2
CURSOR Y COORDINATE IN VAS	3
SWITCH STATUS WORD	4

RE471

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.

TRANSFORMATION REPORT PACKET

This report provides the current transformation details.

FORMAT: The format of the report packet is given in Figure 16–15.

Figure 16–15 Format of Transformation Report Packet

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

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 9).

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 16–16.

Figure 16–16 Format of VIVID Error Report Packet

VIVID__ERROR = 130	0
NUMBER OF PARAMETERS = 7	1
ERROR CODE	2
TOP LEVEL SEGMENT ID	3
CURRENT SEGMENT ID	4
CURRENT SEGMENT BYTE OFFSET	5
CURRENT OPCODE	6
DRAWING X COORDINATE IN VAS	7
DRAWING Y COORDINATE IN VAS	8

RE469

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.

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 16–17.

Figure 16–17 Format of VIVID Interrupt Report Packet

VIVID__INTERRUPT = 128	0
NUMBER OF PARAMETERS = 4	1
TOP LEVEL SEGMENT ID	2
CURRENT SEGMENT ID	3
CURRENT SEGMENT BYTE OFFSET	4
CURRENT SEGMENT LAST OPCODE	5

RE473

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.

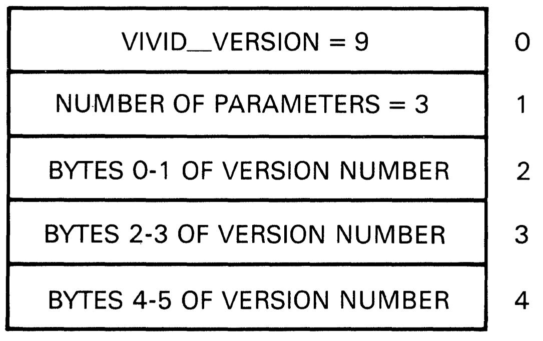
VIVID_VERSION REPORT PACKET

This report provides the downloaded VIVID interpreter version number.

FORMAT:

The format of the report packet is given in Figure 16–18.

Figure 16–18 Format of VIVID Version Report Packet



VIVID__VERSION = 9	0
NUMBER OF PARAMETERS = 3	1
BYTES 0-1 OF VERSION NUMBER	2
BYTES 2-3 OF VERSION NUMBER	3
BYTES 4-5 OF VERSION NUMBER	4

RE466

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.

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 16–19.

Figure 16–19 Format of VIVID Warning Report Packet

VIVID__WARNING = 32	0
NUMBER OF PARAMETERS = 5	1
WARNING CODE	2
TOP LEVEL SEGMENT ID	3
CURRENT SEGMENT ID	4
CURRENT SEGMENT BYTE OFFSET	5
CURRENT OPCODE	6

RE468

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.

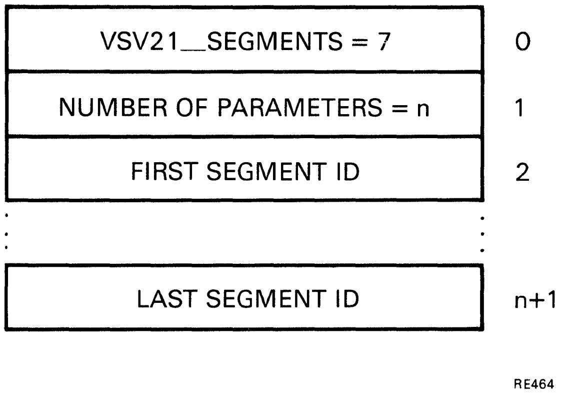
VSV21_SEGMENTS REPORT PACKET

This report provides a list of the segments downloaded to the VSV21.

FORMAT:

The format of the report packet is given in Figure 16–20.

Figure 16–20 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.

Part V VSV11 and Fortran Draw

This section describes how to develop graphics applications with the VSV21 in VSV11 emulation mode, and build pictures using the FORTRAN draw package.

17 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 the *VSV21 User's Guide*.

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:

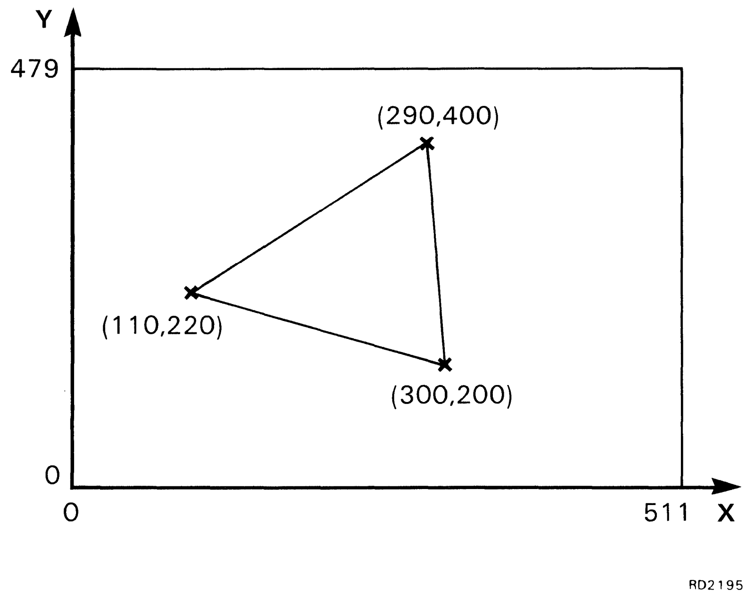
- Draw common graphic shapes
- Control color attributes
- Write text
- Draw graphs and histograms
- Perform screen and drawing position control functions
- Control the cursor position
- Perform initialization and input/output functions
- Access the display list contents
- Control display list processing
- Handle joystick input
- Issue QIO requests

17.1 USING FORTRAN DRAW

17.1.1 Coordinate System

The coordinates used are shown in Figure 17-1.

Figure 17–1 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).

17.1.2 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.

17.1.3 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.

17.2 PROGRAMMING METHOD

The basic method of writing a program to use FORTRAN Draw subroutines is shown in flowchart form in Figure 17-2.

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 VSVCP LOAD command in the *VSV21 User's Guide*. 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.

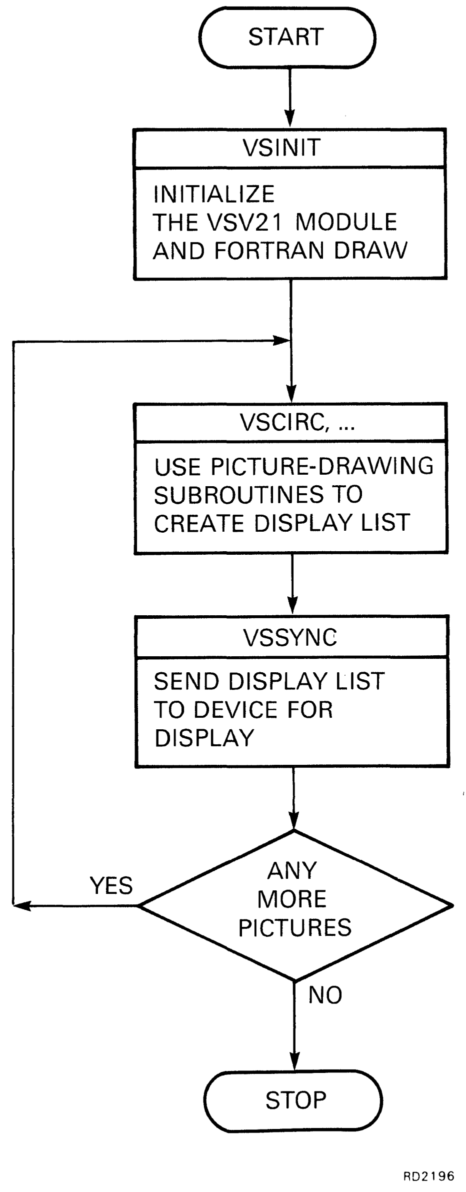
17.3 FORTRAN DRAW SUBROUTINES

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
VSDOT	-	Draws an absolute point
VSDRAW	-	Draws a line to a point
VSDTHK	-	Draws a variable width line
VSFILL	-	Draws a filled rectangle
VSPOLY	-	Draws a filled or unfilled polygon
VSRECT	-	Draws a filled or unfilled rectangle
VSRDRW	-	Draws a relative line

Figure 17–2 Programming Method for FORTRAN Draw


START
VSINIT
INITIALIZE THE VSV21 MODULE AND FORTRAN DRAW
VSCIRC, ...
USE PICTURE-DRAWING SUBROUTINES TO CREATE DISPLAY LIST
VSSYNC
SEND DISPLAY LIST TO DEVICE FOR DISPLAY
ANY MORE PICTURES
YES
NO
STOP
RD2196

Color Control		
VSBACK	-	Sets the background color
VSCOLR	-	Sets the drawing color
VSMIX	-	Mixes a color

Text Control		
VSDFNT	-	Writes text in the current font
VSDSHD	-	Writes text with drop shadow
VSFLEN	-	Gets the length of a text string
VSFONT	-	Selects a text font
VSTEXT	-	Writes a text string

Graphs and histograms:		
VSETHB	-	Sets the histogram base
VSGRFX	-	Adds a point to X graph
VSGRFY	-	Adds a point to Y graph
VSHINC	-	Sets the histogram increment
VSHSTX	-	Adds a point to X histogram
VSHSTY	-	Adds a point to Y histogram

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
VSCURS	-	Performs cursor operation
VSPUTC	-	Sets cursor position

Initialization and configuration:		
VSINIT	-	Initializes device and package
VSMODE	-	Sets channel characteristics
VSSWAP	-	Inverts the state of all active channels

Display list functions:		
VSDJMP	-	Jumps within display list
VSDPLY	-	Puts data into display list buffer
VSFILE	-	Begins saving display list instructions in a file
VSLOAD	-	Loads saved display list
VSSYNC	-	Sends display list to device for display

Joystick control:		
VSJOYS	-	Performs joystick operation
VSWAIT	-	Waits for switch interrupt

Miscellaneous:		
VSDLAY	-	Pauses
VSGADR	-	Gets address
VSQIO	-	Issues a QIO call
VSSTAT	-	Gets I/O status block from last QIO

18 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 the *VSV21 User's Guide.*

The VSV21 can process display lists that 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 that 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 memory before it can be processed.

The display list consists of words of binary information that 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 that identifies the instruction, and parameters that 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 that 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 18.2.

18.1 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.

18.1.1 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.

18.1.2 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 18.1.1.

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.

18.1.3 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.

18.1.4 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.

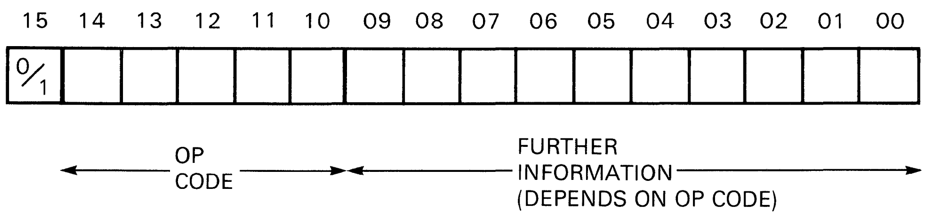
The special graphic instructions available are:

- 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 18-1. 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 18–1 VSV11 Display List Instruction Format



18.2 GENERATING AND PROCESSING VSV11 DISPLAY LISTS

The steps common to both VMS and RSX systems when generating a display list and processing it on the VSV21 are as follows:

1 Create the display list.

  You can use a display list which has previously been created and stored in a file, but you must read the display list into the memory before it can be processed.

2 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.

3 You can interrupt the display list while it is being processed by using the QIO functions described later in Chapter 19.

  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.

4 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.

5 Continue in the same way until all display lists have been displayed.

RSX Specific Steps

Before item 1 above, start the process by attaching the user task to the device.

This dedicates a VSV21 so that it will only accept 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.

After item 5 above, finish the process by detaching your task from the VSV21.

At the end of the program, if your task is attached to the VSV21, detach it to free the device.

19 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 the *VSV21 User's Guide.*

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. The basic steps in generating a VSV11 display list and processing it on the VSV21 are given in Section 18.2.

19.1 QIO FUNCTIONS FOR VMS/MicroVMS

The functions and their hexadecimal values are listed in Table 19–1.

Table 19–1 VSV11 Emulation QIO Functions for VMS/MicroVMS

Description	Function Code	Hexadecimal Value
Read Data	VSV$_READDATA	38
Read Status	VSV$_ READSTATUS	31
Resume Execution	VSV$_RESUME	36
Start Display	VSV$_START	30
Stop Display	VSV$_STOP	33
Timeout	VSV$_TOUT	37
Wait	VSV$_ WAITSWITCH	32
Write Data	VSV$_WRITEDATA	39

Read Data - VSV$_READDATA

This function reads data from the transparent driver.

QIO FORMAT: ***SYS$QIOW VSV$_READDATA efn,chan,func,[iosb],[astadr],[astprm],<baddr,blen,tabid>***

PARAMETERS: **efn, chan, func, iosb, astadr, astprm** are as described in VSV$START.

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 the transparent driver

Read Status - VSV$_READSTATUS

Returns the four device registers containing the following:

- DPC address
- Most recent graphics mode
- Current X and Y positions

QIO FORMAT: ***SYS$QIOW VSV$_READSTATUS efn,chan,func, [iosb],[astadr],[astprm],<baddr,blen>***

PARAMETERS: **efn, chan, func, iosb, astadr, astprm** are as described in VSV$_START.

baddr
address of status buffer

blen
length of status buffer. Minimum length is 8 bytes

Resume Execution - VSV$_RESUME

Resumes execution of a display file after a display stop. The arguments must be the same as those in the VSV$_START function to be stopped.

QIO FORMAT: ***SYS$QIOW VSV$_RESUME efn,chan,func,[iosb], [astadr],[astprm],<baddr,blen, [staddr],[aaddr],[alen],[chan]>***

PARAMETERS: The arguments are as for VSV$_START.

Start Display - VSV$_START

This function starts the display.

QIO FORMAT: ***SYS$QIOW*** ***VSV$_START efn,chan,func,[iosb],[astadr],[astprm],<baddr,blen,[staddr],[aadr],[alen],[chan]>***

PARAMETERS:

efn
event flag number

chan
I/O channel number

func
function code and modifier bits that specify the operation to be performed

iosb
address of input/output status block used for reply status

astadr
address of the entry mask of an AST service routine to be executed when the I/O completes

astprm
AST parameter

baddr
starting address of the display file buffer

blen
length of the display file buffer

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

chan
channel number to use for the display file

Stop Display - VSV$_STOP

This function can be used to stop a looping display file.

QIO FORMAT: ***SYS$QIOW VSV$_STOP efn,chan,func,[iosb],[astadr],[astprm]***

PARAMETERS: **efn, chan, func, iosb, astadr, astprm** are as described in VSV$_START.

Set Timeout Period - VSV$_TOUT

Sets the number of seconds to wait for an I/O complete on VSV$_START. This value is in effect until it is changed by another VSV$_TOUT QIO or by reloading the driver during SYSGEN.

Initially the timeout is 15 seconds.

QIO FORMAT:

SYS$QIOW VSV$_TOUT efn,chan,func,[iosb], [astadr],[astprm], <tout>

PARAMETERS:

efn, chan, func, iosb, astadr, astprm are as described in VSV$_START.

tout
number of seconds to wait; tout > 1

Wait for Switch Interrupt - VSV$_WAITSWITCH

Wait a specified number of seconds for a switch interrupt. This function is complete when a switch interrupt occurs. If a bus timeout occurs, x the function completes with an error status. If any other interrupt occurs, the wait continues.

QIO FORMAT: ***SYS$QIOW VSV$_WAITSWITCH efn,chan,func, [iosb],[astadr],[astprm],<tout,baddr, blen>***

PARAMETERS: **efn, chan, func, iosb, astadr, astprm** are as described in VSV$START,

tout
number of seconds to wait

baddr
address of status buffer

blen
length of status buffer. Minimum length is 8 bytes.

Write Data - VSV$_WRITEDATA

This function writes data to the transparent driver.

QIO FORMAT: ***SYS$QIOW VSV$_WRITEDATA efn,chan,func,[iosb], [astadr],[astprm],<baddr,blen,tabid>***

PARAMETERS: **efn, chan, func, iosb, astadr, astprm** are as described in VSV$START.

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

19.2 QIO FUNCTIONS FOR RSX-11M-PLUS AND MICRO/RSX

The functions and their octal values are listed in Table 19-1.

Table 19–2 VSV11 Emulation QIO Functions for RSX-11M-PLUS and Micro/RSX

Description	Function Code	Octal Code
Attach the VSV21	IO.ATT	1400
Cancel I/O requests	IO.KIL	0012
Connect and display	IO.CON	3000
Connect to auxiliary memory	IO.AUX	2400
Continue display	IO.CNT	4000
Detach the VSV21	IO.DET	2000
Read data	IO.RED	6400
Read joystick location	IO.RJS	5000
Stop display	IO.STP	3400
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.

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. See IO.AUX, for further information.
- 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 DISPLAYJUMP-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 further explained in the IO.CON description.

To set all of these conditions, set the graphics bit by defining TF.GORE = 2 in your program.

QIO FORMAT: ***QIOW$ IO.ATT,lun,efn,,iosb***
or:
QIOW$ IO.ATT!TF.GORE,lun,efn,,iosb

PARAMETERS:

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

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***

lun, efn, and iosb are as described in IO.ATT.

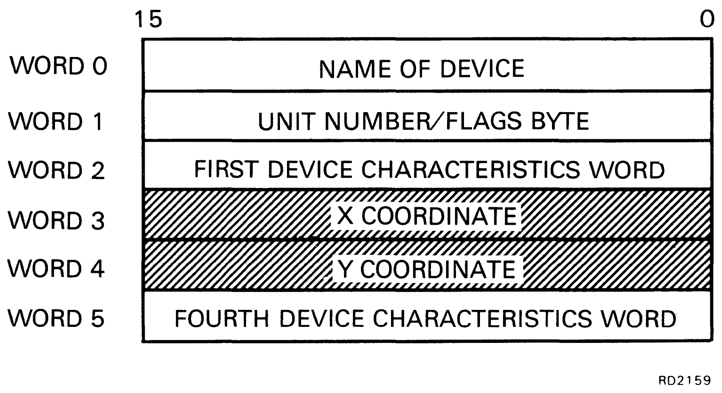
Connect and Display - IO.CON

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.

If you previously set the graphics bit (TF.GORE) 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 the following example. 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 Micro/RSX Executive Reference Manual* (AA-Z508A-TC).

GLUN$ sets the contents of the six-word buffer as shown in Figure 19-1.

Figure 19–1 Contents of GLUN$ Buffer



QIO FORMAT: ***QIOW$ IO.CON,lun,efn,,iosb,,<dsaddr,dlen, addr,chmode,[tout] [,asaddr]>***

PARAMETERS: lun, efn, and iosb are described in IO.ATT

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.GORE bit in the IO.ATT function.

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

Further explanation of the contents of this word is given in the the LOAD STATUS REGISTER C display list instruction. Description in the VSV11/VS11 Option description (YM-C183C-00).

tout

timeout 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). If the timeout 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.

Connect to Auxiliary Memory - IO.AUX

Connects the device to the auxiliary memory.

QIO FORMAT: ***QIOW$ IO.AUX,lun,efn,,iosb,,<axaddr,dlen,addr>***

PARAMETERS: **lun, efn and iosb** are as described in VSV$START.

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

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***

Parameters are as described in IO.ATT.

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***

PARAMETERS: **lun, efn, and iosb** are described in IO.ATT.

Read Data - IO.RED

This function reads data from the transparent driver.

QIO FORMAT: ***QIOW$ IO.RED,lun,efn,,iosb,,<baddr,blen,tabid>***

PARAMETERS: **lun, efn, and iosb** are described in IO.ATT.

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 the transparent driver.

Read Joystick - IO.RJS

This function returns the coordinates of the current position of the joystick.

QIO FORMAT: ***QIOW$ IO.RJS,lun,efn,,iosb,,<baddr,blen,jsnum>***

PARAMETERS: **lun, efn, and iosb** are described in IO.ATT.

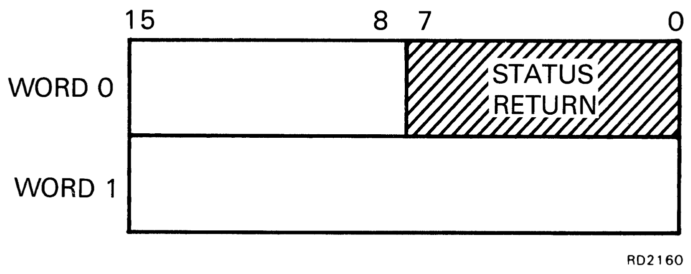
baddr
start address of the data area in which the joystick coordinates will be returned

blen
size of the data area, in bytes (minimum four bytes)

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 19-2.

Figure 19–2 Joystick Data returned by IO.RJS



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***

PARAMETERS: Parameters are as described in IO.ATT.

Write Data - IO.WRT

This function writes data to the transparent driver.

QIO FORMAT: ***QIOW$ IO.WRT,efn,,iosb,,<baddr,blen,tabid>***

PARAMETERS: **lun, efn, and iosb** are described in IO.ATT.

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)

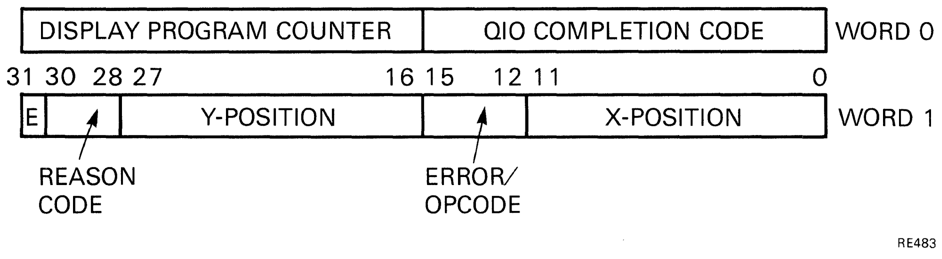
tabid
table ID. Set to zero for the transparent driver.

19.3 QIO STATUS RETURNS

19.3.1 VMS/MicroVMS Systems

In VMS/MicroVMS systems, the I/O status block has the format shown in Figure 19-4.

Figure 19–3 Format of I/O Status Block under VMS/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:

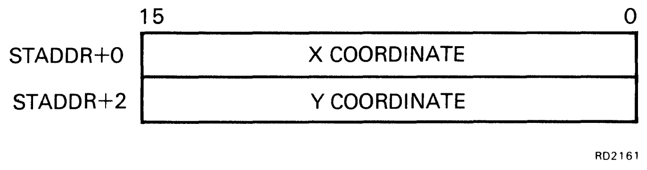
1	=	VS$CR_STOP	normal
2	=	VS$CR_SWITCH	switch
3	=	VS$CR_MATCH	match
4	=	VS$CR_NXM	VSV11 hardware error
5	=	VS$CR_TIMEOUT	timeout
6	=	VS$CR_FORCE	stop acknowledgement
0	=	VS$CR_UNDEFINED	undefined

bit 31	-	error flag

19.3.2 RSX and MicroRSX Systems

In RSX-11M-PLUS and Micro/RSX systems, the I/O status block has the format shown in Figure 19-3.

Figure 19–4 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 the *RSX-11M/M-PLUS I/O Drivers Reference Manual* (AA-L677A-TC).

A THE QIO CALL MECHANISM

A.1 Overview Of 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 VAX 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 appendix.

- 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) and the *RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual* (AA-Z508A-TC). FORTRAN programmers should also refer to these manuals for a description of the subroutine calls which are equivalent to the QIO macro calls.

A.2 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). 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.2.1 The IO.ATT Function

Program example:

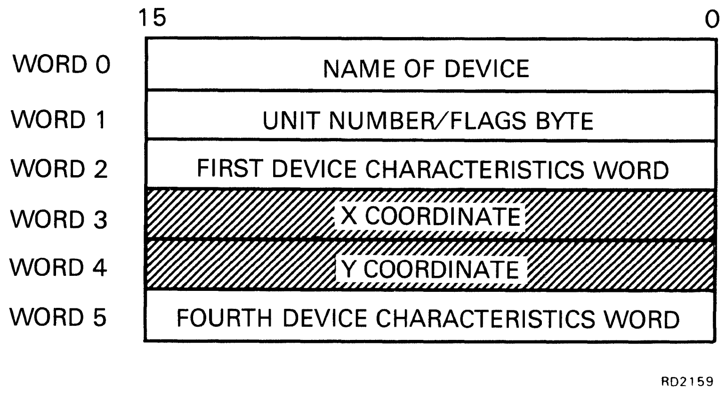
```
IOSB:   .BLKW   2                       ; VSV21 I/O STATUS BLOCK

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.2.2 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 A-1.

Figure A–1 Contents of GLUN$ Buffer



A.2.3 The IO.CON Function

A.2.3.1 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.2.3.2 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.2.4 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
       .
```

B DESIGNING A CHARACTER

The following is a suggested method for designing a special character. The technique can also be applied to designing area texture patterns and special cursors.

The method is:

1 Sketch character using grid

2 Define row values as bit pattern

3 Reverse row values

4 Use these row values in VVTLDC or LOAD_CHAR_CELL

C EXAMPLE OF A VSV11 EMULATION PROGRAM

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 instructions
C       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
```

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 VSVCP command procedure or a VIVID display segment.

Table D–1 VIVID Attribute Mask Values

Mask Value		Attribute Group	Initialisation 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
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)	-

E DEFINING A CURSOR IN VIVID

The maximum size a cursor can have is the maximum cell size (16 x 16 pixels). The cell is a 16 x 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:

```
          0F................
          0E................
          0D................
          0C................
          0B................
          0A................
 row      09................
          08................
number    07................
          06................
          05................
          07................
          06................
          05................
          04................
          03................
          02................
          01................
          00................

            0123456789ABCDEF

               column
               number
```

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 XX..............
              0E XX..............
              0D XX..............
              0C XX..............
              0B XX..............
              0A XX..............
        row   09 XX..............
              08 XX..............
     number   07 XX..............
              06 XX..............
              05 XX..............
              07 XX..............
              06 XX..............
              05 XX..............
              04 XX..............
              03 XX..............
              02 XX..............
              01 XXXXXXXXXXXXXXXX
              00 XXXXXXXXXXXXXXXX

                 0123456789ABCDEF

              column number
```

The display list defining this cursor is as follows:

```
5A13    ! set cursor style and number of rows
0010    ! 16 columns of cursor data (width of cursor)
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
```

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)
001C    !
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
```

Index