AFGIS™ Instruction Set Manual

Version 3.11

December 12, 1994

Preface

This manual describes the AFGIS™ 3.11 firmware instruction set, interface protocol, and associated memory organization on Peritek/RGI graphics boards. AFGIS firmware resides in EPROM on Peritek/RGI graphics boards and provides over 250 highly optimized graphics primitives for graphics programming.

AFGIS firmware supports multi-tasking applications and has been designed for use with the AFGIS C Graphics Library and driver for selected real-time operating systems.

Most users will want to program Peritek/RGI graphics boards with the AFGIS C Graphics Library and driver when using a real-time operating system such as OS-9, pSOS, VRTX, PDOS, VxWorks, VMEexec, etc. In these cases, this manual would be used mostly as a reference document and would not generally be required to program the graphics board.

In cases where a driver is not available for the user’s operating system, the Peritek/RGI graphics board may be programmed directly using the AFGIS Firmware Instruction Set described in this manual. Most users, however, would likely prefer a higher level interface allowing them to use their high level language to program the graphic board.

Users developing their own driver and graphics library will require the information contained in this manual to use the graphics primitives provided by the AFGIS firmware.

Custom driver developement is available from Peritek Corporation

Related Documents

AFGIS C Graphics Library Reference Manual
Standard Drawing Library C Reference Manual

Table of Contents

Introduction

Overview

AFGIS firmware has been designed for use with Peritek/RGI graphics boards and real-time multi-tasking operating systems. AFGIS firmware (pronounced ‘AF-JIS) provides over 250 highly optimized graphics primitives to allow users to easily generate text, lines, polygons, windows, circles, etc., with Peritek/RGI graphics boards.

In addition to the graphics primitives, AFGIS firmware supports serial port, serial mouse, and keyboard interfaces, an on-board memory manager, and polled and interrupt interfaces to the system bus.

AFGIS firmware has been specifically designed for use with the AFGIS C Graphics Library and provides many advanced features which have been optimized for use with RGI graphics boards.

C programers can use the full power and convenience of the C language for graphics programming and achieve impressive graphics performance with this powerful combination of optimized graphics primitives, parallel processing, and the AFGIS C Graphics Library.

Graphics may also be created by programming the Peritek/RGI graphics boards directly with AFGIS opcodes. For most applications, we suggest using the Standfard Drawing Library (SDL) and appropriate driver to program the Peritek/RGI graphics boards.

AFGIS firmware and Peritek/RGI graphics boards are easy to use and work well in simple embedded systems or in sophisticated real-time applications with multiple tasks.

What are AFGIS opcodes, and how are they used?

AFGIS firmware provides over 250 instructions (opcodes) for easy graphics programing. AFGIS opcodes are 16 bit values which may be followed by parameters, similar to most assembly languages.

The programming interface to the Peritek/RGI graphics board is via the AFGIS opcodes. Most users will use C to program the Peritek/RGI graphics boards; however, the following paragraphs discuss briefly how AFGIS opcodes are used to generate graphics and provides a conceptual model of how Peritek/RGI graphics boards work. When using the C Graphics Library, the appropriate AFGIS opcodes are issued by the C function call.

AFGIS opcodes are placed in graphics board memory by the host processor and the AFGIS opcodes are processed by the graphics board’s TMS34010/20 graphics

When the Peritek/RGI graphics boards are programed using the AFGIS C Graphics Library or Standard Drawing Library and driver for a particular operating system, the mechanics of loading AFGIS opcodes, acquiring information from the graphics board, initiating opcode processing, and responding to interrupts is handled by the driver and is transparent to the programmer.

AFGIS Firmware supports Real-time Multi-tasking Operating Systems

AFGIS firmware resides on the graphics board and allows several real-time tasks to use the graphics board. Each task can have its own color, font, screen positions, etc. The color, font, and screen position of one task is not affected by the color, font, and screen position of another task.

AFGIS firmware provides a pointer based graphics environment that includes the color, font, screen position, etc. Typically, the graphics environment is switched for each task by the driver, which provides a private graphics environment for each task. Graphics environment switching time is fast, requiring only the time for the host to change a pointer on the graphics board.

A default environment is provided at power up that may be used for applications with only a single task. The graphics environment is in RAM and is located from a pointer in fixed RAM.

Major Features Provided by AFGIS Firmware

- **Graphics Environment for Real-Time Multi-Tasking applications**

  Fast switching time, typically only a few microseconds.

- **Serial Mouse Interface with polled or interrupt modes**

  Supports Microsoft format
  Built in cursors, local tracking
  Programmable reporting modes
  Mouse queue for temporary data storage

- **Serial Interface with polled or interrupt modes**

  Serial queue for temporary data storage

- **AT Keyboard Interface with polled or interrupt modes**

  Supports AT keyboards
  Provides DOS codes from keyboard scan codes
  Keyboard queue for temporary data storage

- **On board 60 Hz Interrupt to service mouse/serial and keyboard queues & Outbound Interrupt Queue**

- **Control Flow, such as repeat and conditional jumps**
- **Board Status Information for use by host**
  Returns board resolution, busy status, current screen location, etc.
- **Line Drawing with solid and dashed lines**
  Absolute and relative modes
  Pen styles
  Solid, Dashed, Fatline, and Pattern Filled Fatlines
- **Window Instructions with window relative commands**
  Move window and contents by moving the logical origin
  Screen copy and screen save
  Window clipping
- **Text Instructions including**
  Built-in character sets
  User defined character sets
  Rotated text
- **Interrupts from graphics board to host and host to graphics board**
- **Immediate, Indirect, and Variable Indirect instruction operations**
- **Supports calls to TMS34010/20 assembly code**
- **Convex and non-convex polygons with solid and pattern fills**
- **Circles, Ellipses, Sectors, and Arcs**
  Dashed Lines, Fatlines, and Pattern Fills
- **On-Board Memory Manager**
- **Pattern Fills for Polygons, Conics, and Fatlines**

Fixed RAM and pointers to other RAM locations

AFGIS provides fixed RAM (at pre-defined addresses) that holds pointers to other RAM locations which provide information about the graphics board. For example, fixed RAM holds a pointer to the current graphic environment, etc. Fixed RAM starts at 0300 0000h for TMS34010 based graphics boards and at 1000 0000h for TMS34020 based graphics boards. Fixed RAM parameters are shown below (34010 addresses):

ADDRESS	NAME	SIZE	ACCESS	DESCRIPTION
03000000h	EODLFLAG	16	R/W	= 0 when the graphics board is busy. = 1 when the graphics board is not busy.
03000010h	KBDFLAG	16	R/W	= 0 when there is no keyboard data. = 1 when keyboard data is available
03000020h	MSEFLAG	16	R/W	= 0 when there is no mouse/serial data. = 1 when mouse/serial data is available
03000030h	ERRFLAG	16	R/W	= 0 when no errors have been detected = 1 when an error has been detected
03000040h	IDLEFLAG	16	R/W	Set to 1 on each pass of the idle loop, approximately every 10 usecs. Not cleared by AFGIS firmware.
03000050h	DI_COUNT	16	R	60hz continuous counter, updated by AFGIS.
03000060h	INTOUTMASK	16	R/W	Graphics board to host interrupt enable mask.

Figure 1.1 Fixed Ram Parameters

The Graphics Environment, an Overview

The graphics environment is a portion of RAM that contains all the parameters that affect the drawing features of the Peritek/RGI graphics board, such as color, current screen position, current font, etc. The current graphics environment is located by a pointer. There can be several graphics environments. Typically, each is associated with a particular task in a real-time system. However, only one graphics environment can be active at a time.

At power up, the default graphics environment is active. A new graphics environment is created with the R_ENVB or R_ENVC opcode.

The AFGIS opcodes, R_ENVB and R_ENVC, initialize the new graphics environment with default values. The host can then selectively change any of the parameters in the new graphics environment as required. Simply change the pointer in fixed RAM, at ENV_PTR, to make the new graphics environment the current graphics environment. See Appendix A for additional information about the graphics environment.

AFGIS Variables

AFGIS variables, referred to in source form by V, Vs, Vd, @V, Vi, and Vo are identified by a 16 bit number and refer to a 32 bit value which may be used as operands with some AFGIS opcodes.

AFGIS variables are located in the graphics environment (see Appendix A) and are used to pass data to and from the host, and can be used for loops, counting, and for arithmetic and logical operations.

There are 64 AFGIS variables (V0-V63) in the default graphics environment. Users can specify any number of variables when creating a new graphics environment.

An AFGIS variable is actually a 32 bit RAM location that can be manipulated with certain AFGIS opcodes. The major use for AFGIS variables is to pass data from the host to the graphics board and vice versa.

AFGIS variables follow the syntax used with the TI assembler, left to right, source → destination. Vs is used when the variable is the source of the data. Vd is used when the variable is the destination of the data. For example, ADVV Vs Vd adds the value in Vs to the value in Vd and stores the result in Vd.

V is used when the variable is not clearly the source or destination of the data.

@V is used when the variable holds a pointer to memory.

Vi is used to identify the first variable of a series of successive variables that contain input data, typically passed from the host to the graphics board. For example, CIRS Vi, written in hex form (executable format) could be 0015 0005, and would specify that a circle was to be drawn with the parameters in the successive variables V5, V6, V7, and V8. The 0015 is the hex value for the circle opcode, and the 0005 refers to V5, the first of four successive variables containing parameters.

Vo is used to identify the first variable of a series of successive variables that contain output data, typically to be read by the host. For example, R_ARC Vo returns the coordinates of the last arc drawn, and could be written in hex format as 00BA 0003. 00BA is the opcode value to specify return last arc coordinates, and 0003 would specify that V3 would hold the first parameter to be returned, V4-V8 would contain the other parameters, as a total of 6 values are returned with this opcode.

AFGIS variable arithmetic supports signed operations using 2's complement notation (i.e. 00000001=1, 00000000=0, FFFFFFFF=-1, etc.).

AFGIS Opcode Syntax

AFGIS opcodes are shown in source format (as they would be used with an assembler) and the corresponding hex (or executable) values are also shown. An assembler would convert the source form to the corresponding hex format. The AFGIS C graphics library uses a header file to equate the AFGIS opcode source name to the corresponding hex value for execution by the Peritek/RGI graphics board.

AFGIS firmware requires the hex value, a 16 bit value, to identify a variable when executing the code. For example, 0004 is the executable form of V4.

Alphabetical Listing

Name	Hex Value and parameters	Description
ADIV	0002 long V*d*	Add immediate to variable
ADVV	0003 V*s* V*d*	Add variable to variable
ANDIV	0006 long V*d*	AND immediate with variable
ANDVV	0007 V*s* V*d*	AND variable with variable
ARC	0008	Draw arc
	w_*type* w_*x* w_*y* w_*start_angle* w_*end_angle* w_*radx* w_*rady*	
ARCV	0009 V*i*	Vi = line-type
		Vi+1 = X coordinate
		Vi+2 = Y coordinate
		Vi+3 = start angle (θ*0*)
		Vi+4 = end angle (θ*1*)
		Vi+5 = X radius
		Vi+6 = Y radius
ARCTIC	000A	Draw arc tic-marks
	w_*type* w_*x* w_*y* w_*angle* w_*length* w_*xrad* w_*yrad*	
ARCTICV	000B V*i*	Draw arc tic-marks, variable
		Vi = line type
		Vi+1 = X coordinate
		Vi+2 = Y coordinate
		Vi+3 = angle
		Vi+4 = tic-mark length
		Vi+5 = X radius
		Vi+6 = Y radius
BLINK	0162	Set blinking palette entry
	w_*channel* w_*index* w_*rate*	long_*color0* long_*color1*
BLINKV	0163 Vi	Set blinking pallette entry, variable
		Vi = channel ID
		Vi+1 = color index (RAMDAC address)
		Vi+2 = blink rate
		Vi+3 = color 0 (RGB color)
		Vi+4 = color 1 (RGB color)
BLINKON	0164 w_*channel* w_*index* w_*code*	Enable/disable blinking palette entry
BLINKONV	0165 Vi	Enable/disable blinking palette entry, variable
		Vi = channel ID
		Vi+1 = color index (RAMDAC address)
		Vi+2 = function code
BOOL	000C word	Set pixel processing (boolean) operation
BOOLV	000D V	Set pixel processing (boolean) operation, variable
CAL	000E address	Call AFGIS subroutine
CALV	000F V	Call AFGIS subroutine, variable
CALR	010E word	Call AFGIS subroutine relative
CALRV	010F V	Call AFGIS subroutine relative, variable
CASM	0010 address	Call TMS340x0 subroutine
CASMV	0011 V	Call TMS340x0 subroutine, variable
CIR	0012 w_*type* w_*x* w_*y* w_*rad*	Draw circle
CIRV	0013 V*i*	Draw circle, variable
		Vi = line type
		Vi+1 = X coordinate
		Vi+2 = Y coordinate
		Vi+3 = radius
CIRS	0014 w_*type* w_*x* w_*y* w_*rad*	Fill circle
CIRSV	0015 V*i*	Fill circle, variable
		Vi = fill type
		Vi+1 = X coordinate
		Vi+2 = Y coordinate
		Vi+3 = radius
CLIPMODE	0016 word	Set window clipping mode
CLIPMODEV	0017 V	Set window clipping mode, variable
CLIPWIN	0018 w_*x0* w_*y0* w_*x1* w_*y1*	Set clipping window
CLIPWINV	0019 V*i*	Set clipping window, variable
		Vi = Xmin (left)
		Vi+1 = Ymin (top)
		Vi+2 = Xmax (right)
		Vi+3 = Ymax (bottom)

Alphabetical Listing

Name	Hex Value and parameters	Description
CLRM	001A	Clear all video memory to 0 (BLACK)
CLRPAGE	017A w_*channel* w_*page* long_*color* w_*waitflag*	Clear channel/page to color
CLRPAGEV	017B V*i*	Clear channel/page to color, variable Vi = channel ID Vi+1 = page# Vi+2 = color Vi+3 = wait flag
CLRPG	001B w_*page* long_*color*	Clear page to color
CLRPGV	001C V*i*	Clear page to color, variable Vi = page# Vi+1 = color
CLRWIN	0188 long_*color* w_*waitflag*	Clear window to color
CLRWINV	0189 V*i*	Clear window to color, variable Vi = color Vi+1 = wait flag
COLORB	001D long	Set background color
COLORBV	001E V	Set background color, variable
COLORF	001F long	Set foreground color
COLORFV	0020 V	Set foreground color, variable
CONFIG	0021 word	Set video configuration
CONFIGV	0022 V	Set video configuration, variable
CONTREGX	0136 w_clrmask w_setmask	Set/Clear user-configurable bits
CONTREGXV	0137 V*i*	Set/Clear user-configurable bits, variable Vi = clear-mask Vi+1 = set-mask
COPYEE	013E addr_sENV addr_dENV addr_*p*ENV *w_X0 w_Y0 w_X1 w_Y1 w_destX w_destY*	Copy rectangle from environment to environment
COPYEEV	013F V*i*	Copy rectangle from environment to environment, variable Vi = address of source environment Vi+1 = address of destination environment Vi+2 = address of parameter environment Vi+3 = source rectangle Xmin Vi+4 = source rectangle Ymin Vi+5 = source rectangle Xmax Vi+6 = source rectangle Ymax Vi+7 = destination rectangle X Vi+8 = destination rectangle Y
COPYPP	0025 w_*Spage* w_*Dpage*	Copy screen from page to page
COPYPPV	0026 V*i*	Copy screen from page to page, variable Vi = source page# Vi+1 = destination page#
COPYRR	0027 addr_*src* addr_*dest* long_*size*	Copy buffer from RAM to RAM
COPYRRV	0028 V*i*	Copy buffer from RAM to RAM, variable Vi = source address (linear) Vi+1 = destination address (linear) Vi+2 = length of block in bytes
COPYRS	0029 address w_*x* w_*y*	Copy rectangle from RAM buffer to current screen page
COPYRSV	002A V*i*	Copy rectangle from RAM buffer to current screen page, variable Vi = source address (linear) Vi+1 = destination X (screen address) Vi+2 = destination Y (screen address)
COPYRSP	0166 address w_*page w_X w_Y*	Copy rectangle from RAM buffer to screen page
COPYRSPV	0167 V*i*	Copy rectangle from RAM buffer to screen page, variable Vi = source address (linear) Vi+1 = destination page# Vi+2 = destination X Vi+3 = destination Y
COPYSR	002B w_*X0* w_*Y0* w_*X1* w_*Y1* address	Copy rectangle from current screen to RAM buffer

Alphabetical Listing

Name	Hex Value and parameters	Description
COPYSRV	002C V*i*	Copy rectangle from current screen to RAM, variable Vi = Xmin (left) Vi+1 = Ymin (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom) Vi+4 = destination address (linear)
COPYSRP	0168 *w_page w_X0 w_Y0 w_X1*	Copy rectangle from screen page to RAM buffer *w_Y1* address
COPYSRPV	0169 V*i*	Copy rectangle from screen page to RAM buffer, variable Vi = source page# Vi+1 = source Xmin Vi+2 = source Ymin Vi+3 = source Xmax Vi+4 = source Ymax Vi+5 = destination address (linear)
COPYSS	002D w_*X0* w_*Y0* w_*X1* w_*Y1* w_dest*X*	Copy rectangle from current screen to current screen w_dest*Y*
COPYSSV	002E V*i*	Copy rectangle from current screen to current screen, variable Vi = Xmin (left) Vi+1 = Ymin (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom) Vi+4 = destination X (screen address) Vi+5 = destination Y (screen address)
COPYSSP	016A w_*Spage* w_*Dpage* w_*X0*	Copy rectangle from screen page to screen page w_*Y0* w_*X1* w_*Y1* w_*destX* w_*destY*
COPYSSPV	016B V*i*	Copy rectangle from screen page to screen page, variable Vi = source page# Vi+1 = destination page# Vi+2 = source Xmin Vi+3 = source Ymin Vi+4 = source Xmax Vi+5 = source Ymax Vi+5 = destination X Vi+5 = destination Y
CPFILL	002F w_*type* w_*count* address	Convex polygon fill
CPFILLV	0030 V*i*	Convex polygon fill, variable Vi = fill type Vi+1 = vertex count Vi+2 = address of vertex list
CPFILLO	0140 *w_type w_count* address	Convex polygon fill (offset)
CPFILLOV	0141 V*i*	Convex polygon fill (offset), variable Vi = fill type Vi+1 = vertex count Vi+2 = address of vertex list
CPFILLR	0031 w_*type* w_*count* address	Convex polygon fill (relative)
CPFILLRV	0032 V*i*	Convex polygon fill (relative), variable Vi = fill type Vi+1 = vertex count Vi+2 = address of vertex list
CPIV	0033 long V*d*	Compare immediate to variable
CPVV	0034 V*s* V*d*	Compare variable to variable
CTEXTA	0035 address	Print character text, indirect address
CTEXTI	0036 <string>	Print character text, immediate (in-line)
CTEXTV	0037 V	Print character text, variable
CTEXTLXY	0038 w_*x* w_*y*	Set current CTEXT location
CTEXTLXYV	0039 V*i*	Set current CTEXT location, variable Vi = X coordinate Vi+1 = Y coordinate
CTEXTMXY	003A w_*x* w_*y*	Set current CTEXT margin
CTEXTMXYV	003B Vi	Set current CTEXT margin, variable Vi = X coordinate Vi+1 = Y coordinate
CTEXTXA	0102 word long	Print character text, explicit format, indirect address
CTEXTXV	0103 word	Print character text, explicit format, variable indirect

Alphabetical Listing

Name	Hex Value and parameters	Description
DASHCON	003C word	Select dash pattern continue mode
DASHCONV	003D V	Select dash pattern continue mode,variable
DASHOFFS	003E word	Set dash pattern offset
DASHOFFSV	003F V	Set dash pattern offset, variable
DASHPATN	0040 long	Select dashed-line pattern
DASHPATNV	0041 V	Select dashed line pattern, variable
DCRV	0042 V	Decrement variable
DELAY	0043 word	Delay opcode processing
DELAYV	0044 V	Delay opcode processing, variable
DIVV	0045 V*s* V*d*	Divide variable by variable
DPAGE	017C w_*channel* w_*page* w_*waitflag*	Set current display for channel and page
DPAGEV	017D V*i*	Set current display for channel and page, variable Vi = channel ID Vi+1 = page# Vi+2 = wait flag
DPG	0046 word	Set current display page
DPGA	0047 address	Set current display page address
DPGV	0048 V	Set current display page, variable
ELP	0049 w_*type* w_*x* w_*y* w_*radx* w_*rady*	Draw ellipse
ELPV	004A V*i*	Draw ellipse, variable Vi = line type Vi+1 = X Vi+2 = Y Vi+3 = x radius Vi+4 = y radius
ELPS	004B w_*type* w_*x* w_*y* w_*radx* w_rady	Fill ellipse
ELPSV	004C V*i*	Fill ellipse, variable Input: Vi = fill type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = x radius Vi+4 = y radius
EODL	0001	End of display list
ERPT	004F	End AFGIS repeat loop
FATLNC	0050 word	Select fatline cap-style
FATLNCV	0051 V	Select fatline cap-style, variable
FATLNJ	0052 word	Select fatline joint-style
FATLNJV	0053 V	Select fatline joint-style, variable
FATLNW	0054 word	Select fatline width
FATLNWV	0055 V	Select fatline width, variable
FONT	0056 long	Set current font
FONTV	0057 V	Set current font, variable
GETPALETTE	0198 w_*channel* w_*iColor* w_*nColors*	Read color palette address
GETPALETTEV	0199 V*i*	Read color palette, variable Vi = channel ID Vi+1 = index of initial entry to read Vi+2 = address of (contiguous) entries to read Vi+3 = address of destination buffer
GTEXTA	005A address	Print graphics text, indirect address
GTEXTI	005B <string>	Print graphics text, immediate (in-line)
GTEXTV	005C V	Print graphics text, variable
GTEXTXA	0104 word long	Print graphics text, explicit format, indirect address
GTEXTXV	0105 word	Print graphics text, explicit format, variable indirect
INCV	005D V	Increment variable
INITGCB	0148 *addr_GC addr_DB addr_DB_*	Initialize graphics context for draw buffer params
INITGCBV	0149 V*i*	Initialize graphics context for draw buffer, variable Vi = address of graphics context Vi+1 = address of draw buffer Vi+2 = address of draw buffer parameter structure

Alphabetical Listing

Name	Hex Value and parameters	Description
INITGCC	0178 addr_*GC* w_*channel* w_*page*	Initialize graphics context for channel and page
INITGCCV	0179 V*i*	Initialize graphics context for channel and page, variable Vi = address of graphics context Vi+1 = channel ID Vi+2 = page#
JUMPA	010A word long	Jump absolute
JUMPAV	010B V*i*	Jump absolute, variable
JUMPR	010C word word	Jump relative
JUMPRV	010D V*i*	Jump relative, variable
KBMODE	0060 word	Set keyboard interrupt mode
KBMODEV	0061 V	Set keyboard interrupt mode, variable
KBQFL	0062	Flush (reset) keyboard queue
KBRST	0063	Reset keyboard
KBTEST	0064	Test keyboard
LDIV	0065 word V*d*	Load immediate to variable (short)
LDIVL	0066 long V*d*	Load immediate to variable (long)
LDMV	0067 long V*d*	Load memory to variable (short)
LDMVL	0068 long V*d*	Load memory to variable (long)
LDPCV	0069 V	Load AFGIS PC to variable
LDPMV	006A @V V*d*	Load memory variable-indirect to variable (short)
LDPMVL	006B @V V*d*	Load memory variable-indirect to variable (long)
LDSPV	006C V	Load AFGIS SP to variable
LDVM	006D V*s* address	Load variable to memory (short)
LDVML	006E V*s* address	Load variable to memory (long)
LDVPM	006F V*s* @V	Load variable to memory variable-indirect (short)
LDVPML	0070 V*s* @V	Load variable to memory variable-indirect (long)
LDVV	0071 V*s* V*d*	Load variable to variable (move)
LED	0072 word	Set RED/GREEN LEDs
LEDV	0073 V	Set RED/GREEN LEDs, variable
LINE	0074 w_*type* w_*x0* w_*y0* w_*x1* w_*y1*	Draw line point to point
LINEV	0075 V*i*	Draw line point to point, variable Vi = line type Vi+1 = X0 Vi+2 = Y0 Vi+3 = X1 Vi+4 = Y1
LINECON	0076 word	Select line pattern continue mode
LINECONV	0077 V	Select line pattern continue mode, variable
LINEPATN	0078 long	Select binary line pattern
LINEPATNV	0079 V	Select binary line pattern, variable
LINER	007A w_*type* w_*dx0* w_*dy0* w_*dx1* w_*dy1*	Draw line point-to-point (relative)
LINERV	007B V*i*	Draw line point to point (relative), variable Input: Vi = line type Vi+1 = dX0 (X0 offset) Vi+2 = dY0 (Y0 offset) Vi+3 = dX1 (X1 offset) Vi+4 = dY1 (Y1 offset)
LINETO	007C w_*type* w_*x1* w_*y1*	Draw line from current position to point
LINETOV	007D V*i*	Draw line from current position to point, variable Vi = line type Vi+1 = X Vi+2 = Y
LINETOR	007E w_*type* w_*dx1* w_*dy1*	Draw line from current position to point (relative)
LINETORV	007F V*i*	Draw line from current position to point (relative), variable Vi = line type Vi+1 = X*1* offset Vi+2 = Y*1* offset
MARKER	0108 long	Set current marker
MARKERV	0109 V	Set current marker, variable
MLTV	0080 V*s* V*d*	Multiply variable by variable

Alphabetical Listing

Name	Hex Value and parameters	Description
MODV	0081 V*s* V*d*	Modulus variable with variable
MOVETO	0082 w_*x* w_*y*	Set current X,Y location
MOVETOV	0083 V*i*	Set current X,Y location, variable Vi = X coordinate Vi+1 = Y coordinate
MOVETOR	0084 w_*dx* w_*dy*	Set current X,Y location (relative)
MOVETORV	0085 V*i*	Set current X,Y location (relative), variable Vi = X offset Vi+1 = Y offset
MSCSRON	0088 word	Enable/Disable mouse cursor display
MSCSRONV	0089 V	Enable/Disable mouse cursor display, variable
MSCSRPAGE	012C w_*channel* w_*page*	Configure mouse cursor for channel and page
MSCSRPAGEV	012D V*i*	Configure mouse cursor for channel and page, variable Vi = mouse cursor channel ID Vi+1 = mouse cursor display page
MSCSRXY	008A w_*x* w_*y*	Set mouse cursor location
MSCSRXYV	008B V*i*	Set mouse cursor location, variable Vi = new mouse cursor X Vi+1 = new mouse cursor Y
MSCURSOR	016C long_*cursor* long_*color1*	Select mouse cursor long_*color2* addr_*save* long_*save_pitch*
MSCURSORV	016D V*i*	Select mouse cursor, variable Vi = address of cursor structure (or index of default) Vi+1 = shape #1 color Vi+2 = shape #2 color Vi+3 = save buffer address Vi+4 = save buffer pitch
MSMODE	008C word	Set mouse interrrupt service mode
MSMODEV	008D V	Set mouse interrupt service mode, variable
MSQFL	008E	Flush mouse queue
MSREG	0156 w_*reg* long_*value*	Set mouse register
MSREGV	0157 V*i*	Set mouse register, variable Vi = mouse register # Vi+1 = mouse register value
MSSCALE	0158 *w_Xscale w_Yscale*	Set mouse scale factors
MSSCALEV	0159 V*i*	Set mouse scale factors, variable Vi = mouse X scale factor Vi+1 = mouse Y scale factor
MSTEST	008F	Mouse test
MSWIN	015A *w_X0 w_Y0 w_X1 w_Y1*	Set mouse window
MSWINV	015B V*i*	Set mouse window,variable Vi = mouse window Xmin Vi+1 = mouse window Ymin Vi+2 = mouse window Xmax Vi+3 = mouse window Ymax
NOOP	0000	No-operation (null)
ORIV	0090 long V*d*	OR immediate with variable
ORVV	0091 V*s* V*d*	OR variable with variable
PANX	0116 word	Pan display horizontally (absolute)
PANXV	0117 word	Pan display horizontally (absolute), variable
PANY	0118 word	Pan display vertically (absolute)
PANYV	0119 word	Pan display vertically (absolute), variable
PANXY	011A word word	Pan display (absolute)
PANXYV	011B word	Pan display (absolute), variable
PANXYR	011C word word	Pan display (relative)
PANXYRV	011D word	Pan display (relative), variable
PATRNMODE	0094 word	Set pattern-fill reference mode
PATRNMODEV	0095 V	Set pattern-fill reference mode, variable
PATRNREF	0096 w_*x* w_*y*	Set pattern-fill reference point (offset)
PATRNREFV	0097 V*i*	Set pattern-fill reference point (offset), variable Vi = X coordinate Vi+1 = Y coordinate
PENDEF	0098 w_*type* w_*x* w_*y*	Define pen parameters
PENDEFV	0099 V*i*	Define pen parameters, variable Vi = type code Vi+1 = X half size Vi+2 = Y half-size

Alphabetical Listing

Name	Hex Value and parameters	Description
PFILL	009A w_*type* w_*count* address	General polygon fill
PFILLV	009B V*i*	General polygon fill, variable Vi = fill type Vi+1 = vertex count Vi+2 = address of vertex list
PFILLO	0142 w_*type* w_*count* address	General polygon fill (offset)
PFILLOV	0143 V*i*	General polygon fill (offset), variable Vi = fill type Vi+1 = vertex count Vi+2 = address of vertex list
PFILLR	009C w_*type* w_*count* address	General polygon fill (relative)
PFILLRV	009D V*i*	General polygon fill (relative), variable Vi = fill type Vi+1 = vertex count Vi+2 = address of vertex list
PIXEL	009E w_*x* w_*y*	Set pixel to current foreground color
PIXELV	009F V*i*	Set pixel to current foreground color, variable Vi = X coordinate Vi+1 = Y coordinate
PIXELC	0150 w_*X* w_*Y* long_*color*	Set pixel to color
PIXELCV	0151 V*i*	Set pixel to color, variable Vi = pixel X coordinate Vi+1 = pixel Y coordinate Vi+2 = color
PLINE	00A0 w_*type* w_*count* address	Polyline
PLINEV	00A1 V*i*	Polyline, variable Vi = line type Vi+1 = vertex count Vi+2 = address of vertex list
PLINEO	0144 w_*type* w_*count* address	Polyline (offset)
PLINEOV	0145 V*i*	Polyline (offset), variable Vi = line type Vi+1 = vertex count Vi+2 = address of vertex list
PLINER	00A2 w_*type* w_*count* address	Polyline (relative)
PLINERV	00A3 V*i*	Polyline (relative), variable Vi = line type Vi+1 = vertex count Vi+2 = address of vertex list
PLINES	0182 w_type w_*count* address	Draw poly-line-segments
PLINESV	0183 V*i*	Draw poly-line-segments, variable Vi = line type Vi+1 = line-segment count Vi+2 = address of vertex list
PLINESR	0184 w_*type* w_*count* address	Draw poly-line-segments (relative)
PLINESRV	0185 V*i*	Draw poly-line-segments (relative), variable Vi = line type Vi+1 = line-segment count Vi+2 = address of vertex list
PMARK	00A4 w_*count* address	Poly-marker
PMARKV	00A5 V*i*	Poly-marker, variable Vi = vertex count Vi+1 = address of vertex list
PMARKO	0146 w_*count* address	Poly-marker (offset)
PMARKOV	0147 V*i*	Poly-marker (offset), variable Vi = vertex count Vi+1 = address of vertex list
PMARKR	00A6 w_*count* address	Poly-marker (relative)
PMARKRV	00A7 V*i*	Poly-marker (relative), variable Vi = vertex count Vi+1 = address of vertex list
PMASK	00A8 long	Set plane mask
PMASKV	00A9 V	Set plane mask, variable
POPV	00AA V	Pop variable from AFGIS stack
POPVARS	00AB w_count V	Pop contiguous variables from AFGIS stack
PPIXEL	0186 w_*count* address	Poly-pixel

Alphabetical Listing

Name	Hex Value and parameters	Description
PPIXELV	0187 V*i*	Poly-pixel , variable Vi = vertex count Vi+1 = address of vertex list
PPIXELO	0196 w_*count* address	Poly-pixel (offset)
PPIXELOV	0197 V*i*	Poly-pixel (offset), variable Vi = vertex count Vi+1 = address of vertex list
PPIXELR	0194 w_*count* address	Poly-pixel (relative)
PPIXELRV	0195 V*i*	Poly-pixel (relative), variable Vi = vertex count Vi+1 = address of vertex list
PUSHV	00AC V	Push variable to AFGIS stack
PUSHVARS	00AD w_count V	Push contiguous variables to AFGIS stack
RANDRANGE	00AE long long	Set range for random-number
RANDRANGEV	00AF Vi	Set range for random-number, variable Vi = random number range low value Vi+1 = random number range high value
RANDSEED	00B0 long	Sets random-number seed value
RANDSEEDV	00B1 V	Sets random-number seed value, variable
RECT	00B3 w_*type* w_*x0* w_*y0* w_*x1* w_*y1*	Draw rectangle
RECTV	00B4 V*i*	Draw rectangle, variable Vi = line type Vi+1 = Xmin Vi+2 = Ymin Vi+3 = Xmax Vi+4 = Ymax
RECTS	00B5 w_*type* w_*x0* w_*y0* w_*x1* w_*y1*	Fill rectangle
RECTSV	00B6 V*i*	Fill rectangle, variable Vi = fill type Vi+1 = Xmin Vi+2 = Ymin Vi+3 = Xmax Vi+4 = Ymax
RPT	00B7 word	Begin AFGIS repeat loop
RPTV	00B8 V	Begin AFGIS repeat loop, variable
RRECT	015E w_*type* w_*x0* w_*y0* w_*x1* w_*y1* w_*Xradius* w_*Yradius*	Draw rounded rectangle
RRECTV	015F V*i*	Draw rounded rectangle, variable Vi = line type Vi+1 = rectangle Xmin (left) Vi+2 = rectangle Ymin (top) Vi+3 = rectangle Xmax (right) Vi+4 = rectangle Ymax (bottom) Vi+5 = corner-fillet Xradius Vi+6 = corner-fillet Yradius
RRECTS	0130 w_*type* w_*x0* w_*y0* w_*x1* w_*y1* w_*Xradius* w_*Yradius*	Fill rounded rectangle
RRECTSV	0131 V*i*	Fill rounded rectangle, variable Vi = fill type Vi+1 = rectangle Xmin (left) Vi+2 = rectangle Ymin (top) Vi+3 = rectangle Xmax (right) Vi+4 = rectangle Ymax (bottom) Vi+5 = corner-fillet Xradius Vi+6 = corner-fillet Yradius
RTRN	00B9	Return from AFGIS subroutine
RWMEM	0138 address w_*clrmask w_setmask*	Read/write graphics board memory
RWMEMV	0139 V*i*	Read/write graphics board memory, variable Vi = address Vi+1 = clear-mask Vi+2 = set-mask
R_ALLOC	0004 long V*o*	Allocate memory from heap

Alphabetical Listing

Name	Hex Value and parameters	Description
R_ALLOCV	0005 V*i* V*o*	Allocate memory from heap, variable Input: Vi = allocation size in bytes Output: Vo = allocation address (or NULL if error)
R_ARC	00BA V*o*	Return coordinates of last arc drawn Output: Vo = X@ center Vo+1 = Y@ center Vo+2 = X@ theta0 endpoint Vo+3 = Y@ theta0 endpoint Vo+4 = X@ theta1 endpoint Vo+5 = Y@ theta1 endpoint
R_ARCPTS	00BB w_*θ0* w_*θ1* w_*xrad* w_*yrad* V*o*	Return coordinates of arc endpoints
R_ARCPTSV	00BC V*i* V*o*	Return coordinates of arc endpoints, variable Input: Vi = start angle (theta0) Vi+1 = end angle (theta1) Vi+2 = X radius Vi+3 = Y radius Output: Vo = X@ theta0 endpoint (center relative) Vo+1 = Y@ theta0 endpoint (center relative) Vo+2 = X@ theta1 endpoint (center relative) Vo+3 = Y@ theta1 endpoint (center relative)
R_ARCTIC	00BD w_*angle* w_*length* w_*Xrad* w_*Yrad* V*o*	Return arc tic-mark coordinates
R_ARCTICV	00BE V*i* V*o*	Return arc tic-mark coordinates, variable Input: Vi = angle Vi+1 = tic-mark length Vi+2 = X radius Vi+3 = Y radius Output: Vo = X0 (outer endpoint, center relative) Vo+1 = Y0 (outer endpoint, center relative) Vo+2 = X1 (inner endpoint, center relative) Vo+3 = Y1 (inner endpoint, center relative)
R_CPW	00BF w_*x* w_*y* V*o*	Return "clipcode"
R_CPWV	00C0 V*i* V*o*	Return "clipcode", variable Input: Vi = X coordinate Vi+1 = Y coordinate Output: Vo = clipcode
R_ENVB	0152 addr_ENV_*params* addr_DB addr_DB_*params* V*o*	Allocate environment and initialize for draw buffer
R_ENVBV	0153 V*i* V*o*	Allocate environment and initialize for draw buffer, variable input: Vi = address of environment parameter structure Vi+1 = address of draw buffer Vi+2 = address of draw buffer parameter structure output: Vo = address of new environment
R_ENVC	0154 addr_ENV_*params* w_*channel* w_*page* V*o*	Allocate environment and initialize for channel and page
R_ENVCV	0155 V*i* V*o*	Allocate environment and initialize for channel and page, variable input: Vi = address of environment parameter structure Vi+1 = channel ID Vi+2 = page# output: Vo = address of new environment
R_FREE	0058 address V*o*	Deallocate memory from heap

Alphabetical Listing

Name	Hex Value and parameters	Description
R_FREEV	0059 V*i* V*o*	Deallocate memory from heap, variable Input: Vi = address of memory to de-allocate Output: Vo = success flag
R_ISIZE	00C1 w_*X0* w_*Y0* w_*X1* w_*Y1* V*o*	Return image size for COPYSR
R_ISIZEV	00C2 V*i* V*o*	Return image size for COPYSR, variable Input: Vi = X min (left) Vi+1 = Y min (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom) Output: Vo = image size in bytes
R_PIXEL	00C5 w_*x* w_*y* V*o*	Return color at pixel location
R_PIXELV	00C6 V*i* V*o*	Return color at pixel location, variable Input: Vi = X coordinate Vi+1 = Y coordinate Output: Vo = color
R_RAND	00B2 V*o*	Generate random number
R_RGB	0190 w_*channel* w_*index* Vo	Return RGB color of single palette entry
R_RGBV	0191 V*i* V*o*	Return RGB color of single palette entry, variable Input: Vi = channel ID Vi+1 = color index (RAMDAC address) Output: V*o* = RGB color
R_TEXTDA	00C7 address V*o*	Return text dimensions, indirect address
R_TEXTDV	00C8 V*i* V*o*	Return text dimensions, variable Input: Vi = address of string Output: Vo = X extent Vo+1 = Y extent
R_TEXTDXA	0106 w_*mode* address V*o*	Return text dimensions, explicit format, indirect address
R_TEXTDXV	0107 V*i* V*o*	Return text dimensions, explicit format, variable indirect Input: Vi = string format mode Vi+1 = address of string Output: Vo = X extent Vo+1 = Y extent
R_TEXTP	00C9 word V*o*	Return text parameter
R_TEXTPV	00CA V*i* V*o*	Return text parameter, variable Input: Vi= function code Output: Vo= value of parameter requested
R_XYLADD	00CB w_*x* w_*y* V*o*	Convert XY to linear address
R_XYLADDV	00CC V*i* V*o*	Convert XY to linear address, variable Input: Vi = X coordinate Vi+1 = Y coordinate Output: Vo = corresponding linear address
SBIV	00CD long V	Subtract immediate from variable
SBVV	00CE V*s* V*d*	Subtract variable from variable
SECT	00CF w_*type* w_*x* w_*y* w_*start_angle* w_*end_angle* w_*radx* w_*rady*	Draw sector

Alphabetical Listing

Name	Hex Value and parameters	Description
SECTV	00D0 V*i*	Draw sector, variable Vi = line type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = start angle Vi+4 = end angle Vi+5 = X radius Vi+6 = Y radius
SECTS	00D1 w_type w_x w_y w_start_angle w_end_angle w_radx w_rady	Fill sector
SECTSV	00D2 Vi	Fill sector, variable Vi = fill type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = start angle Vi+4 = end angle Vi+5 = X radius Vi+6 = Y radius
SEEDFILL	00D3 w_*type* w_*seedX* w_*seedY*	Flood seed fill
SEEDFILLV	00D4 V*i*	Flood seed fill, variable Vi = fill type Vi+1 = seed X Vi+2 = seed Y
SEG	0160 w_*type* w_*x* *w_y* w_*start_angle* w_*end_angle* w_*radx* w_*rady*	Draw segment
SEGV	0161 V*i*	Draw segment, variable Vi = line type Vi+1 = arc center X coordinate Vi+2 = arc center Y coordinate Vi+3 = start angle Vi+4 = end angle Vi+5 = arc X radius Vi+6 = arc Y radius
SEGS	015C w_*type* w_*x* *w_y* w_*start_angle* w_*end_angle* w_*radx* w_*rady*	Fill segment
SEGSV	015D V*i*	Fill segment, variable Vi = fill type Vi+1 = arc center X coordinate Vi+2 = arc center Y coordinate Vi+3 = start angle Vi+4 = end angle Vi+5 = arc X radius Vi+6 = arc Y radius
SERBAUD	00D5 word	Set baud rate for 2691 UART
SERBAUDV	00D6 V	Set baud rate for 2691 UART, variable
SERMODE	00D7 word	Set serial port interrupt service mode
SERMODEV	00D8 V	Set serial port interrupt service mode, variable
SEROUT	00D9 word	Output character byte to serial port
SEROUTV	00DA V	Output character byte to serial port, variable
SERQFL	00DB	Flush serial port queue
SERUART	00DC address	Initialize 2691 UART
SERUARTV	00DD @V	Initialize 2691 UART, variable
SETPALETTE	0112 w_*channel* long_*palette*	Select palette
SETPALETTEV	0113 V*i*	Select palette, variable Vi = channel ID Vi+1 = address of palette structure (or index of default)
SETRGB	018E w_*channel* w_*index* long_*rgb_color*	Set single palette entry to RGB color
SETRGBV	018F V*i*	Set single palette entry to RGB color, variable Vi = channel ID Vi+1 = color index (RAMDAC address) Vi+2 = RGB color
SHIFT	00DE long	Shift screen area
SHIFTV	00DF V	Shift screen area, variable
SLLV	00E0 w_*count* V	Shift left logical variable
SRLV	00E1 w_*count* V	Shift right logical variable
SSYM	00E2 w_*rotation* long_*ssymbol*	Draw “simple symbol”

Alphabetical Listing

Name	Hex Value and parameters	Description
SSYMV	00E3 V*i*	Draw “simple symbol”, variable Vi = rotation Vi+1 = address of symbol structure (or index of default)
SSYMX	00E4 w_*rotation* w_*width* w_*height* w_*pitch* address	Draw “simple symbol” (explicit parameters)
SSYMXV	00E5 Vi	Draw “simple symbol” (explicit param), variable Vi = rotation Vi+1 = symbol width Vi+2 = symbol height Vi+3 = symbol pitch Vi+4 = pointer to symbol (pixblt) data
STIPPLE	00E6 long_*pattern*	Select stipple (binary) fill pattern
STIPPLEV	00E7 V	Select stipple (binary) fill pattern, variable
STIPPLEX	018A w_*width* w_*height* w_*pitch* address	Select stipple fill pattern, explicit parameters
STIPPLEXV	018B V*i*	Select stipple fill pattern, explicit parameters, variable Vi = pattern width Vi+1 = pattern height Vi+2 = pattern pitch Vi+3 = address of stipple pattern data
TEXTP	00E8 w_code long	Set text parameter
TEXTPV	00E9 Vi	Set text parameter, variable Vi = function code Vi+1 = parameter value
TEXTSVC	00EA long	Select text service routine
TEXTSVCV	00EB V	Select text service routine, variable
TILE	00EC long_*pattern*	Select tile (pixel mapped) fill pattern
TILEV	00ED V	Select tile (pixel mapped) fill pattern, variable
TILEX	018C w_*width* w_*height* w_*depth* w_*pitch* address	Select tile fill pattern, explicit parameters
TILEXV	018D V*i*	Select tile fill pattern, explicit parameters, variable Vi = tile pattern width Vi+1 = tile pattern height Vi+2 = tile pattern depth (pixel size) Vi+3 = tile pattern array pitch Vi+4 = address of tile pattern data
TRANS	00EE word	Set graphics transparency mode
TRANSV	00EF V	Set graphics transparency mode, variable
TXCSRON	0172 w_*flag*	Enable/disable text cursor display
TXCSRONV	0173 V	Enable/disable text cursor display, variable
TXCSRPAGE	0174 w_*channel* *w_page*	Configure text cursor for channel and page
TXCSRPAGEV	0175 V*i*	Configure text cursor for channel and page, variable Vi = text cursor channel ID Vi+1 = text cursor display page
TXCSRWIN	0192 w_*x0* w_*y0* w_*x1* w_*y1*	Set text cursor window
TXCSRWINV	0193 V*i*	Set text cursor window, variable Vi = xmin (left) Vi+1 = ymin (top) Vi+2 = xmax (right) Vi+3 = ymax (bottom)
TXCSRXY	0176 w_*x* w_*y*	Set text cursor location
TXCSRXYV	0177 V*i*	Set text cursor location, variable Vi = new text cursor x Vi+1 = new text cursor y
TXCURSOR	0170 long_*cursor* long_*color1* long_*color2* addr_*save* long_*save_pitch* w_*blink_rate*	Select text cursor
TXCURSORV	0171 V*i*	Select text cursor, variable Vi = address of cursor structure (or index of default) Vi+1 = shape #1 color Vi+2 = shape #2 color Vi+3 = save buffer address Vi+4 = save buffer pitch Vi+5 = blink rate
USCSRON	00F4 word	Set user cursor state on/off
USCSRONV	00F5 V	Set user cursor state on/off, variable

Alphabetical Listing

Name	Hex Value and parameters	Description
USCSRXY	00F6 w_*x* w_*y*	Set current user cursor location
USCSRXYV	00F7 V*i*	Set current user cursor location, variable Vi = X coordinate Vi+1 = Y coordinate
USCURSOR	016E long_*cursor* long_*color1* long_*color2* addr_*save* long_*save_pitch*	Select user cursor
USCURSORV	016F V*i*	Select user cursor, variable Vi = address of cursor structure (or index of default) Vi+1 = shape #1color Vi+2 = shape #2color Vi+3 = address of save buffer Vi+4 = save buffer pitch
VWAIT	00F8	Wait for vertical blanking interval
WPAGEB	017E *addr_DB addr_DB_params*	Initialize drawing parameters for draw buffer
WPAGEBV	017F V*i*	Initialize drawing parameters for draw buffer, variable Vi = address of draw buffer Vi+1 = address of draw buffer parameter structure
WPAGEC	0180 w_*channel* w_*page*	Initialize drawing parameters for channel and page
WPAGECV	0181 V*i*	Initialize drawing parameters for channel and page, variable Vi = channel ID Vi+1 = page#
WPG	00F9 word	Set current write page
WPGA	00FA address	Set current write page address
WPGV	00FB V	Set current write page, variable
XCHGPC	00FC V	Exchange AFGIS PC with variable
XCHGSP	00FD V	Exchange AFGIS SP with variable
XORIV	00FE long V*d*	XOR immediate with variable
XORVV	00FF V*s* V*d*	XOR variable with variable
XYORG	0100 w_*x* w_*y*	Set logical origin
XYORGV	0101 V*i*	Set logical origin, variable Vi = X coordinate Vi+1 = Y coordinate
ZOOM	011E word	Set display zoom factor (enlarge display)
ZOOMV	011F word	Set display zoom factor (enlarge display), variable

Fill Types for Area-Fill Opcodes

CIRS, CPFILL, CPFILLO, CPFILLR, ELPS, PFILL, PFILLO, PFILLR, RECTS, RRECTS, SECTS, SEGS, SEEDFILL (types 0,1 only)

w_type=0= solid color
w_type=1= stipple pattern
w_type=2= tile pattern

Line Types for Line-Draw Opcodes

ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINEO, PLINER, PLINES, PLINESR, RECT, RRECT, SECT, SEG

w_type=0= solid line
w_type=1= dash line (32 bit binary pattern)
w_type=2= dash line (arbitrary - segment-length word list)
w_type=3= fatline - solid
w_type=4= fatline - stipple pattern (binary)
w_type=5= fatline - tile pattern (pixel-mapped)
w_type=6= pen line - solid
w_type=7= pen line - stipple

Functional Listing

Name	Hex Value and parameters	Description
AFGIS Variable Operations		
ADIV	0002 long V*d*	Add immediate to variable
ADVV	0003 V*s* V*d*	Add variable to variable
ANDIV	0006 long V*d*	AND immediate with variable
ANDVV	0007 V*s* V*d*	AND variable with variable
CPIV	0033 long V*d*	Compare immediate to variable
CPVV	0034 V*s* V*d*	Compare variable to variable
DCRV	0042 V	Decrement variable
DIVV	0045 V*s* V*d*	Divide variable by variable
INCV	005D V	Increment variable
LDIV	0065 word V*d*	Load immediate to variable (short)
LDIVL	0066 long V*d*	Load immediate to variable (long)
LDMV	0067 long V*d*	Load memory to variable (short)
LDMVL	0068 long V*d*	Load memory to variable (long)
LDPCV	0069 V	Load AFGIS PC to variable
LDPMV	006A @V V*d*	Load memory variable-indirect to variable (short)
LDPMVL	006B @V V*d*	Load memory variable-indirect to variable (long)
LDSPV	006C V	Load AFGIS SP to variable
LDVM	006D V*s* address	Load variable to memory (short)
LDVML	006E V*s* address	Load variable to memory (long)
LDVPM	006F V*s* @V	Load variable to memory variable-indirect (short)
LDVPML	0070 V*s* @V	Load variable to memory variable-indirect (long)
LDVV	0071 V*s* V*d*	Load variable to variable (move)
MLTV	0080 V*s* V*d*	Multiply variable by variable
MODV	0081 V*s* V*d*	Modulus variable with variable
ORIV	0090 long V*d*	OR immediate with variable
ORVV	0091 V*s* V*d*	OR variable with variable
POPV	00AA V	Pop variable from AFGIS stack
POPVARS	00AB w_count V	Pop contiguous variables from AFGIS stack
PUSHV	00AC V	Push variable to AFGIS stack
PUSHVARS	00AD w_count V	Push contiguous variables to AFGIS stack
SBIV	00CD long V	Subtract immediate from variable
SBVV	00CE V*s* V*d*	Subtract variable from variable
SLLV	00E0 w_*count* V	Shift left logical variable
SRLV	00E1 w_*count* V	Shift right logical variable
XCHGPC	00FC V	Exchange AFGIS PC with variable
XCHGSP	00FD V	Exchange AFGIS SP with variable
XORIV	00FE long V*d*	XOR immediate with variable
XORVV	00FF V*s* V*d*	XOR variable with variable
Area Fill		
CIRS	0014 w_*type* w_*x* w_*y* w_*rad*	Fill circle
CIRSV	0015 V*i*	Fill circle, variable Vi = fill type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = radius
CPFILL	002F w_*type* w_*count* address	Convex polygon fill
CPFILLV	0030 V*i*	Convex polygon fill, variable Vi = fill type Vi+1 = vertex count Vi+2 = pointer to vertex list
CPFILLO	0140 w_*type* w_*count* address	Convex polygon fill (offset)
CPFILLOV	0141 V*i*	Convex polygon fill (offset), variable Vi = fill type Vi+1 = vertex count Vi+2 = pointer to vertex list
CPFILLR	0031 w_*type* w_*count* address	Convex polygon fill (relative)
CPFILLRV	0032 V*i*	Convex polygon fill (relative), variable Vi = fill type Vi+1 = vertex count Vi+2 = pointer to vertex list
ELPS	004B w_*type* w_*x* w_*y* w_*radx* w_rady	Fill ellipse

Functional Listing

Name	Hex Value and parameters	Description
ELPSV	004C V*i*	Fill ellipse, variable Vi = fill type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = x radius Vi+4 = y radius
PATRNMODE	0094 word	Set pattern-fill reference mode
PATRNMODEV	0095 V	Set pattern-fill reference mode, variable
PATRNREF	0096 w_*x* w_*y*	Set pattern-fill reference point
PATRNREFV	0097 V*i*	Set pattern-fill reference point, variable Vi = X coordinate Vi+1 = Y coordinate
PFILL	009A w_type w_count address	General polygon fill
PFILLV	009B Vi	General polygon fill, variable Vi = fill type Vi+1 = vertex count Vi+2 = address of vertex list
PFILLO	0142 w_type w_count address	General polygon fill (offset)
PFILLOV	0143 Vi	General polygon fill (offset), variable Vi = fill type Vi+1 = vertex count Vi+2 = pointer to vertex list
PFILLR	009C w_*type* w_*count* address	General polygon fill (relative)
PFILLRV	009D V*i*	General polygon fill (relative), variable Vi = fill type Vi+1 = vertex count Vi+2 = address of vertex list
RECTS	00B5 w_*type* w_*x0* w_*y0* w_*x1* w_*y1*	Fill rectangle
RECTSV	00B6 V*i*	Fill rectangle, variable Vi = fill type Vi+1 = Xmin Vi+2 = Ymin Vi+3 = Xmax Vi+4 = Ymax
RRECTS	0130 w_*type* w_*x0* w_*y0* w_*x1* w_*y1*	Fill rounded rectangle w_*Xradius* w_*Yradius*
RRECTSV	0131 V*i*	Fill rounded rectangle, variable Vi = fill type Vi+1 = Xmin Vi+2 = Ymin Vi+3 = Xmax Vi+4 = Ymax Vi+5 = corner-fillet Xradius Vi+6 = corner-fillet Yradius
SECTS	00D1 w_type w_x w_y w_start_angle w_end_angle w_radx w_rady	Fill sector
SECTSV	00D2 Vi	Fill sector, variable Vi = fill type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = start angle Vi+4 = end angle Vi+5 = X radius Vi+6 = Y radius
SEGS	015C w_type w_x w_y w_start_angle w_end_angle w_radx w_rady	Fill segment
SEGSV	015D Vi	Fill segment, variable Vi = fill type Vi+1 = arc center X coordinate Vi+2 = arc center Y coordinate Vi+3 = start angle Vi+4 = end angle Vi+5 =arc X radius Vi+6 = arc Y radius
SEEDFILL	00D3 w_*type* w_*seedX* w_*seedY*	Flood seed fill

Functional Listing

Name	Hex Value and parameters	Description
SEEDFILLV	00D4 V*i*	Flood seed fill, variable Vi = fill type Vi+1 = seed X Vi+2 = seed Y
STIPPLE	00E6 long_*pattern*	Select stipple (binary) fill pattern
STIPPLEV	00E7 V	Select stipple (binary) fill pattern, variable
STIPPLEX	018A w_*width* w_*height* w_*pitch*	Select stipple fill pattern, explicit parameters address
STIPPLEXV	018B V*i*	Select stipple fill pattern, explicit parameters, variable Vi = pattern width Vi+1 = pattern height Vi+2 = pattern pitch Vi+3 = address of stipple pattern data
TILE	00EC long_pattern	Select tile (pixel mapped) fill pattern
TILEV	00ED V	Select tile (pixel mapped) fill pattern, variable
TILEX	018C w_*width* w_*height* w_*depth*	Select tile fill pattern, explicit parameters w_*pitch* address
TILEXV	018D V	Select tile fill pattern, explicit parameters, variable Vi = tile pattern width Vi+1 = tile pattern height Vi+2 = tile pattern depth Vi+3 = tile pattern array pitch Vi+4 = address of tile pattern data

Clipping

Name	Hex Value and parameters	Description
CLIPMODE	0016 word	Set window clipping mode
CLIPMODEV	0017 V	Set window clipping mode, variable
CLIPWIN	0018 w_*x0* w_*y0* w_*x1* w_*y1*	Set clipping window
CLIPWINV	0019 V*i*	Set clipping window, variable Vi = Xmin (left) Vi+1 = Ymin (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom)

Colors/Palettes

Name	Hex Value and parameters	Description
BLINK	0162 w_*channel* w_*index* w_*rate*	Set blinking palette entry long_*color0* long_*color1*
BLINKV	0163 V*i*	Set blinking palette entry, variable Vi = channel ID Vi+1 = color index (RAMDAC address) Vi+2 = blink rate Vi+3 = color 0 (RGB color) Vi+4 = color 1 (RGB color)
BLINKON	0164 w_*channel* w_*index* w_*code*	Enable/disable blinking palette entry
BLINKONV	0165 V*i*	Enable/disable blinking palette entry, variable Vi = channel ID Vi+1 = color index (RAMDAC address) Vi+2 = function code
COLORB	001D long	Set background color
COLORBV	001E V	Set background color, variable
COLORF	001F long	Set foreground color
COLORFV	0020 V	Set foreground color, variable
GETPALETTE	0198 w_*channel* w_*iColor* w_*nColors*	Read color palette address
GETPALETTEV	0199 V*i*	Read color palette, variable Vi = channel ID Vi+1 = index of initial entry to read Vi+2 = number of (contiguous) entries to read Vi+3 = address of destination buffer
R_RGB	0190 w_*channel* w_*index* V*o*	Read RGB color of single palette entry
R_RGBV	0191 V*i* V*o*	Read RGB color of single palette entry, variable Input: Vi = channel ID Vi+1 = color index (RAMDAC address) Output: Vo = RGB color
SETPALETTE	0112 w_*channel* long_*palette*	Select palette

Functional Listing

Name	Hex Value and parameters	Description
SETPALETTEV	0113 V*i*	Select palette, variable Vi = channel ID Vi+1 = address of palette structure (or index of default)
SETRGB	018E w_*channel* w_*index* long_*rgb_color*	Set single palette entry to RGB color
SETRGBV	018F V*i*	Set single palette entry to RGB color, variable Vi = channel ID Vi+1 = color index (RAMDAC address) Vi+2 = RGB color

Current Position

Name	Hex Value and parameters	Description
MOVETO	0082 w_*x* w_*y*	Set current X,Y location
MOVETOV	0083 V*i*	Set current X,Y location, variable Vi = X coordinate Vi+1 = Y coordinate
MOVETOR	0084 w_*dx* w_*dy*	Set current X,Y location (relative)
MOVETORV	0085 V*i*	Set current X,Y location (relative), variable Vi = X offset Vi+1 = Y offset
XYORG	0100 w_*x* w_*y*	Set logical origin
XYORGV	0101 V*i*	Set logical origin, variable Vi = X coordinate Vi+1 = Y coordinate

Environment

Name	Hex Value and parameters	Description
R_ENVB	0152 addr_ENV_*params* addr_DB addr_DB_*params* V*o*	Allocate environment and initialize for draw buffer
R_ENVBV	0153 V*i* V*o*	Allocate environment and initialize for draw buffer, variable input: Vi = address of environment parameter structure Vi+1 = address of draw buffer Vi+2 = address of draw buffer parameter structure output: Vo = address of new environment
R_ENVC	0154 addr_ENV_*params* w_*channel* w_*page* V*o*	Allocate environment and initialize for channel and page
R_ENVCV	0155 V*i* V*o*	Allocate environment and initialize for channel and page, variable input: Vi = address of environment parameter structure Vi+1 = channel ID Vi+2 = page# output: Vo = address of new environment
INITGCB	0148 *addr_GC addr_DB addr_DB_*params	Initialize graphics context for draw buffer
INITGCBV	0149 V*i*	Initialize graphics context for draw buffer, variable Vi = address of graphics context Vi+1 = address of draw buffer Vi+2 = address of draw buffer parameter structure
INITGCC	0178 addr_*GC* w_*channel* *w_page*	Initialize graphics context for channel and page
INITGCCV	0179 V*i*	Initialize graphics context for channel and page, variable Vi = address of graphics context Vi+1 = channel ID Vi+2 = page#

Hardware Interface

Name	Hex Value and parameters	Description
CONFIG	0021 word	Set video configuration
CONFIGV	0022 V	Set video configuration, variable
CONTREGX	0136 w_*clrmask* w_*setmask*	Set/Clear user-configurable bits
CONTREGXV	0137 V*i*	Set/Clear user-configurable bits, variable Vi = clear-mask Vi+1 = set-mask
LED	0072 word	Set RED/GREEN LEDs
LEDV	0073 V	Set RED/GREEN LEDs, variable
RWMEM	0138 address w_*clrmask w_setmask*	Read/write graphics board memory

Functional Listing

Name	Hex Value and parameters	Description
RWMEMV	0139 V*i*	Read/write graphics board memory, variable Vi = address Vi+1 = clear-mask Vi+2 = set-mask

Host interface

Name	Hex Value and parameters	Description
R_ENVB	0152 addr_ENV_*params* addr_DB addr_DB_*params* V*o*	Allocate environment and initialize for draw buffer
R_ENVBV	0153 V*i* V*o*	Allocate environment and initialize for draw buffer, variable input: Vi = address of environment parameter structure Vi+1 = address of draw buffer Vi+2 = address of draw buffer parameter structure output: Vo = address of new environment
R_ENVC	0154 addr_ENV_*params* w_*channel* w_*page* V*o*	Allocate environment and initialize for channel and page
R_ENVCV	0155 V*i* V*o*	Allocate environment and initialize for channel and page, variable input: Vi = address of environment parameter structure Vi+1 = channel ID Vi+2 = page# output: Vo = address of new environment
EODL	0001	End of display list
KBMODE	0060 word	Set keyboard interrupt mode
KBMODEV	0061 V	Set keyboard interrupt mode, variable
MSMODE	008C word	Set mouse interrrupt service mode
MSMODEV	008D V	Set mouse interrupt service mode, variable
SERMODE	00D7 word	Set serial port interrupt service mode
SERMODEV	00D8 V	Set serial port interrupt service mode, variable

Image Operations

Name	Hex Value and parameters	Description
COPYEE	013E addr_sENV addr_dENV addr_*p*ENV *w_X0 w_Y0 w_X1 w_Y1 w_destX w_destY*	Copy rectangle from environment to environment
COPYEEV	013F V*i*	Copy rectangle from environment to environment, variable Vi = address of source environment Vi+1 = address of destination environment Vi+2 = address of parameter environment Vi+3 = source rectangle Xmin Vi+4 = source rectangle Ymin Vi+5 = source rectangle Xmax Vi+6 = source rectangle Ymax Vi+7 = destination rectangle X Vi+8 = destination rectangle Y
COPYRS	0029 address w_*x* w_*y*	Copy rectangle from RAM buffer to current screen page
COPYRSV	002A V*i*	Copy rectangle from RAM buffer to current screen page, variable Vi = source address (linear) Vi+1 = destination X (screen address) Vi+2 = destination Y (screen address)
COPYRSP	0166 address w_*page w_X w_Y*	Copy rectangle from RAM buffer to screen page
COPYRSPV	0167 V*i*	Copy rectangle from RAM buffer to screen page, variable Vi = source address (linear) Vi+1 = destination page# Vi+2 = destination X Vi+3 = destination Y
COPYSR	002B w_*X0* w_*Y0* w_*X1* w_*Y1* address	Copy rectangle from current screen to RAM buffer
COPYSRV	002C V*i*	Copy rectangle from current screen to RAM, variable Vi = Xmin (left) Vi+1 = Ymin (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom) Vi+4 = destination address (linear)
COPYSRP	0168 *w_page w_X0 w_Y0 w_X1 w_Y1* address	Copy rectangle from screen page to RAM buffer

Functional Listing

Name	Hex Value and parameters	Description
COPYSRPV	0169 V*i*	Copy rectangle from screen page to RAM buffer, variable Vi = source page# Vi+1 = source Xmin Vi+2 = source Ymin Vi+3 = source Xmax Vi+4 = source Ymax Vi+5 = destination address (linear)
COPYSS	002D w_*X0* w_*Y0* w_*X1* w_*Y1* w_dest*X* w_dest*Y*	Copy rectangle from current screen to current screen
COPYSSV	002E V*i*	Copy rectangle from current screen to current screen, variable Vi = Xmin (left) Vi+1 = Ymin (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom) Vi+4 = destination X (screen address) Vi+5 = destination Y (screen address)
COPYSSP	016A w_*Spage* w_*Dpage* w_*X0* w_*Y0* w_*X1* w_*Y1* w_*destX* w_*destY*	Copy rectangle from screen page to screen page
COPYSSPV	016B V*i*	Copy rectangle from screen page to screen page, variable Vi = source page# Vi+1 = destination page# Vi+2 = source Xmin Vi+3 = source Ymin Vi+4 = source Xmax Vi+5 = source Ymax Vi+5 = destination X Vi+5 = destination Y
R_ISIZE	00C1 w_*X0* w_*Y0* w_*X1* w_*Y1* V*o*	Return image size for COPYSR
R_ISIZEV	00C2 Vi Vo	Return image size for COPYSR, variable Input: Vi = X min (left) Vi+1 = Y min (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom) Output: Vo = image size in bytes
SHIFT	00DE address	Shift screen area
SHIFTV	00DF V	Shift screen area, variable

Keyboard

Name	Hex Value and parameters	Description
KBMODE	0060 word	Set keyboard interrupt mode
KBMODEV	0061 V	Set keyboard interrupt mode, variable
KBQFL	0062	Flush (reset) keyboard queue
KBRST	0063	Reset keyboard
KBTEST	0064	Test keyboard

Line Drawing

Name	Hex Value and parameters	Description
ARC	0008 w_*type* w_*x* w_*y* w_*start_angle* w_*end_angle* w_*radx* w_*rady*	Draw arc
ARCV	0009 V*i*	Vi = line-type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = start angle ($\theta 0$) Vi+4 = end angle ($\theta 1$) Vi+5 = X radius Vi+6 = Y radius
ARCTIC	000A w_*type* w_*x* w_*y* w_*angle* w_*length* w_*xrad* w_*yrad*	Draw arc tic-marks
ARCTICV	000B V*i*	Draw arc tic-marks, variable Vi = line type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = angle Vi+4 = tic-mark length Vi+5 = X radius Vi+6 = Y radius
CIR	0012 w_*type* w_*x* w_*y* w_*rad*	Draw circle

Functional Listing

Name	Hex Value and parameters	Description
CIRV	0013 V*i*	Draw circle, variable Vi = line type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = radius
DASHCON	003C word	Select dash pattern continue mode
DASHCONV	003D V	Select dash pattern continue mode,variable
DASHOFFS	003E word	Set dash pattern offset
DASHOFFSV	003F V	Set dash pattern offset, variable
DASHPATN	0040 long	Select dashed-line pattern
DASHPATNV	0041 V	Select dashed line pattern, variable
ELP	0049 w_*type* w_*x* w_*y* w_*radx* w_*rady*	Draw ellipse
ELPV	004A V*i*	Draw ellipse, variable Vi = line type Vi+1 = X Vi+2 = Y Vi+3 = x radius Vi+4 = y radius
FATLNC	0050 word	Select fatline cap-style
FATLNCV	0051 V	Select fatline cap-style, variable
FATLNJ	0052 word	Select fatline joint-style
FATLNJV	0053 V	Select fatline joint-style, variable
FATLNW	0054 word	Select fatline width
FATLNWV	0055 V	Select fatline width, variable
LINE	0074 w_*type* w_*x0* w_*y0* w_*x1* w_*y1*	Draw line point to point
LINEV	0075 V*i*	Draw line point to point, variable Vi = line type Vi+1 = X0 Vi+2 = Y0 Vi+3 = X1 Vi+4 = Y1
LINECON	0076 word	Select line pattern continue mode
LINECONV	0077 V	Select line pattern continue mode, variable
LINEPATN	0078 long	Select binary line pattern
LINEPATNV	0079 V	Select binary line pattern, variable
LINER	007A w_*type* w_*dx0* w_*dy0* w_*dx1* w_*dy1*	Draw line point-to-point (relative)
LINERV	007B V*i*	Draw line point to point (relative), variable: Vi = line type Vi+1 = dX0 (X0 offset) Vi+2 = dY0 (Y0 offset) Vi+3 = dX1 (X1 offset) Vi+4 = dY1 (Y1 offset)
LINETO	007C w_*type* w_*x1* w_*y1*	Draw line from current position to point
LINETOV	007D V*i*	Draw line from current position to point, variable Vi = line type Vi+1 = X Vi+2 = Y
LINETOR	007E w_*type* w_*dx1* w_*dy1*	Draw line from current position to point (relative)
LINETORV	007F V*i*	Draw line from current position to point (relative), variable Vi = line type Vi+1 = X*1* offset Vi+2 = Y*1* offset
PENDEF	0098 w_*type* w_*x* w_*y*	Define pen parameters
PENDEFV	0099 V*i*	Define pen parameters, variable Vi = type code Vi+1 = X half size Vi+2 = Y half-size
PIXEL	009E w_*x* w_*y*	Write pixel
PIXELV	009F V*i*	Write pixel, variable Vi = X coordinate Vi+1 = Y coordinate
PIXELC	0150 w_*X* w_*Y* long_*color*	Set pixel to color
PIXELCV	0151 V*i*	Set pixel to color, variable Vi = pixel X coordinate Vi+1 = pixel Y coordinate Vi+2 = color
PLINE	00A0 w_*type* w_*count* address	Polyline

Functional Listing

Name	Hex Value and parameters	Description
PLINEV	00A1 V*i*	Polyline, variable Vi = line type Vi+1 = vertex count Vi+2 = address of vertex list
PLINEO	0144 w_*type* w_*count* address	Polyline (offset)
PLINEOV	0145 V*i*	Polyline (offset), variable Vi = line type Vi+1 = vertex count Vi+2 = address of vertex list
PLINER	00A2 w_*type* w_*count* address	Polyline (relative)
PLINERV	00A3 V*i*	Polyline (relative), variable Vi = line type Vi+1 = vertex count Vi+2 = address of vertex list
PLINES	0182 w_type w_*count* address	Draw poly-line-segments
PLINESV	0183 V*i*	Draw poly-line-segments, variable Vi = line type Vi+1 = line-segment count Vi+2 = address of vertex list
PLINESR	0184 w_*type* w_*count* address	Draw poly-line-segments (relative)
PLINESRV	0185 V*i*	Draw poly-line-segments (relative), variable Vi = line type Vi+1 = line-segment count Vi+2 = address of vertex list
PPIXEL	0186 w_*count* address	Poly-pixel
PPIXELV	0187 V*i*	Poly-pixel , variable Vi = vertex count Vi+1 = address of vertex list
PPIXELO	0196 w_*count* address	Poly-pixel (offset)
PPIXELOV	0197 V*i*	Poly-pixel (offset), variable Vi = vertex count Vi+1 = address of vertex list
PPIXELR	0194 w_*count* address	Poly-pixel (relative)
PPIXELRV	0195 V*i*	Poly-pixel (relative), variable Vi = vertex count Vi+1 = address of vertex list
RECT	00B3 w_*type* w_*x0* w_*y0* w_*x1* w_*y1*	Draw rectangle
RECTV	00B4 V*i*	Draw rectangle, variable Vi = line type Vi+1 = Xmin Vi+2 = Ymin Vi+3 = Xmax Vi+4 = Ymax
RRECT	015E w_*type* w_*x0* w_*y0* w_*x1* w_*y1* w_*Xradius* w_*Yradius*	Draw rounded rectangle
RRECTV	015F V*i*	Draw rounded rectangle, variable Vi = line type Vi+1 = rectangle Xmin (left) Vi+2 = rectangle Ymin (top) Vi+3 = rectangle Xmax (right) Vi+4 = rectangle Ymax (bottom) Vi+5 = corner-fillet Xradius Vi+6 = corner-fillet Yradius
SECT	00CF w_*type* w_*x* w_*y* w_*start_angle* w_*end_angle* w_*radx* w_*rady*	Draw sector
SECTV	00D0 V*i*	Draw sector, variable Vi = line type Vi+1 = X coordinate Vi+2 = Y coordinate Vi+3 = start angle Vi+4 = end angle Vi+5 = X radius Vi+6 = Y radius
SEG	0160 w_*type* w_*x* w_*y* w_*start_angle* w_*end_angle* w_*radx* w_*rady*	Draw segment

Functional Listing

Name	Hex Value and parameters	Description
SEGV	0161 V*i*	Draw segment, variable Vi = line type Vi+1 = arc center X coordinate Vi+2 = arc center Y coordinate Vi+3 = start angle Vi+4 = end angle Vi+5 = arc X radius Vi+6 = arc Y radius

Marker

Name	Hex Value and parameters	Description
MARKER	0108 long	Set current marker
MARKERV	0109 V	Set current marker, variable
PMARK	00A4 w_*count* address	Poly-marker
PMARKV	00A5 V*i*	Poly-marker, variable Vi = vertex count Vi+1 = address of vertex list
PMARKO	0146 w_*count* address	Poly-marker (offset)
PMARKOV	0147 V*i*	Poly-marker (offset), variable Vi = vertex count Vi+1 = address of vertex list
PMARKR	00A6 w_*count* address	Poly-marker relative
PMARKRV	00A7 V*i*	Poly-marker relative, variable Vi = vertex count Vi+1 = address of vertex list

Memory Allocation

Name	Hex Value and parameters	Description
R_ALLOC	0004 long V*o*	Allocate memory from heap
R_ALLOCV	0005 V*i* V*o*	Allocate memory from heap, variable Input: Vi = allocation size in bytes Output: Vo = allocation address (or NULL if error)
R_FREE	0058 address V*o*	Deallocate memory from heap
R_FREEV	0059 V*i* V*o*	Deallocate memory from heap, variable Input: Vi = address of memory to de-allocate Output: Vo = success flag
R_ISIZE	00C1 w_*X0* w_*Y0* w_*X1* w_*Y1* V*o*	Return image size for COPYSR
R_ISIZEV	00C2 Vi Vo	Return image size for COPYSR, variable Input: Vi = X min (left) Vi+1 = Y min (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom) Output: Vo = image size in bytes

Mouse

Name	Hex Value and parameters	Description
MSCSRON	0088 word	Enable/Disable mouse cursor display
MSCSRONV	0089 V	Enable/Disable mouse cursor display, variable
MSCSRPAGE	012C w_*channel* w_*page*	Configure mouse cursor for channel and page
MSCSRPAGEV	012D V*i*	Configure mouse cursor for channel and page, variable Vi = mouse cursor channel ID Vi+1 = mouse cursor display page
MSCSRXY	008A w_*x* w_*y*	Set mouse cursor location
MSCSRXYV	008B V*i*	Set mouse cursor location, variable Vi = new mouse cursor X Vi+1 = new mouse cursor Y
MSCURSOR	016C long_*cursor* long_*color1* long_*color2* addr_*save* long_*save_pitch*	Select mouse cursor
MSCURSORV	016D V*i*	Select mouse cursor, variable Vi = address of cursor structure (or index of default) Vi+1 = shape #1 color Vi+2 = shape #2 color Vi+3 = save buffer address Vi+4 = save buffer pitch
MSMODE	008C word	Set mouse interrrupt service mode
MSMODEV	008D V	Set mouse interrupt service mode, variable
MSQFL	008E	Flush mouse queue
MSREG	0156 w_*reg* long_*value*	Set mouse register
MSREGV	0157 V*i*	Set mouse register, variable Vi = mouse register # Vi+1 = mouse register value

Functional Listing

Name	Hex Value and parameters	Description
MSSCALE	0158 *w_Xscale w_Yscale*	Set mouse scale factors
MSSCALEV	0159 V*i*	Set mouse scale factors, variable Vi = mouse X scale factor Vi+1 = mouse Y scale factor
MSTEST	008F	Mouse test
MSWIN	015A *w_X0 w_Y0 w_X1 w_Y1*	Set mouse window
MSWINV	015B V*i*	Set mouse window,variable Vi = mouse window Xmin Vi+1 = mouse window Ymin Vi+2 = mouse window Xmax Vi+3 = mouse window Ymax

Pixel Processing

Name	Hex Value and parameters	Description
BOOL	000C word	Set pixel processing (boolean) operation
BOOLV	000D V	Set pixel processing (boolean) operation, variable
PMASK	00A8 long	Set plane mask
PMASKV	00A9 V	Set plane mask, variable
TRANS	00EE word	Set graphics transparency mode
TRANSV	00EF V	Set graphics transparency mode, variable

Program Flow

Name	Hex Value and parameters	Description
CAL	000E address	Call AFGIS subroutine
CALV	000F V	Call AFGIS subroutine, variable
CALR	010E word	Call AFGIS subroutine relative
CALRV	010F V	Call AFGIS subroutine relative, variable
CASM	0010 address	Call TMS340x0 subroutine
CASMV	0011 V	Call TMS340x0 subroutine, variable
DELAY	0043 word	Delay opcode processing
DELAYV	0044 V	Delay opcode processing, variable
ERPT	004F	End AFGIS repeat loop
JUMPA	010A word long	Jump absolute
JUMPAV	010B V*i*	Jump absolute, variable
JUMPR	010C word word	Jump relative
JUMPRV	010D V*i*	Jump relative, variable
RPT	00B7 word	Begin AFGIS repeat loop
RPTV	00B8 V	Begin AFGIS repeat loop, variable
RTRN	00B9	Return from AFGIS subroutine
VWAIT	00F8	Wait for vertical blanking interval

Random Number Generation

Name	Hex Value and parameters	Description
RANDRANGE	00AE long long	Set random-number range values
RANDRANGEV	00AF Vi	Set random-number range values, variable Vi = random number range low value Vi+1 = random number range high value
RANDSEED	00B0 long	Sets random-number seed value
RANDSEEDV	00B1 V	Sets random-number seed value, variable
R_RAND	00B2 V*o*	Generate random number

Return Parameters

Name	Hex Value and parameters	Description
R_ARC	00BA V*o*	Return coordinates of last arc drawn Output: Vo = X@ center Vo+1 = Y@ center Vo+2 = X@ theta0 endpoint Vo+3 = Y@ theta0 endpoint Vo+4 = X@ theta1 endpoint Vo+5 = Y@ theta1 endpoint
R_ARCPTS	00BB w_$\theta 0$ w_$\theta 1$ w_*xrad* w_*yrad* V*o*	Return coordinates of arc endpoints
R_ARCPTSV	00BC V*i* V*o*	Return coordinates of arc endpoints, variable Input: Vi = start angle (theta0) Vi+1 = end angle (theta1) Vi+2 = X radius Vi+3 = Y radius Output: Vo = X@ theta0 endpoint (center relative) Vo+1 = Y@ theta0 endpoint (center relative) Vo+2 = X@ theta1 endpoint (center relative) Vo+3 = Y@ theta1 endpoint (center relative)
R_ARCTIC	00BD	Return arc tic-mark coordinates

Functional Listing

Name	Hex Value and parameters	Description
	w_*angle* w_*length* w_*Xrad* w_*Yrad* V*o*	
R_ARCTICV	00BE V*i* V*o*	Return arc tic-mark coordinates, variable Input: Vi = angle Vi+1 = tic-mark length Vi+2 = X radius Vi+3 = Y radius Output: Vo = X0 (outer endpoint, center relative) Vo+1 = Y0 (outer endpoint, center relative) Vo+2 = X1 (inner endpoint, center relative) Vo+3 = Y1 (inner endpoint, center relative)
R_CPW	00BF w_*x* w_*y* V*o*	Return “clipcode”
R_CPWV	00C0 V*i* V*o*	Return “clipcode”, variable Input: Vi = X coordinate Vi+1 = Y coordinate Output: Vo = clipcode
R_ENVB	0152	Allocate environment and initialize for draw buffer
	addr_ENV_*params* addr_DB addr_DB_*params* V*o*	
R_ENVBV	0153 V*i* V*o*	Allocate environment and initialize for draw buffer, variable input: Vi = address of environment parameter structure Vi+1 = address of draw buffer Vi+2 = address of draw buffer parameter structure output: Vo = address of new environment
R_ENVC	0154	Allocate environment and initialize for channel and page
	addr_ENV_*params* w_*channel* w_*page* V*o*	
R_ENVCV	0155 V*i* V*o*	Allocate environment and initialize for channel and page, variable input: Vi = address of environment parameter structure Vi+1 = channel ID Vi+2 = page# output: Vo = address of new environment
R_FREE	0058 address V*o*	Deallocate memory from heap
R_FREEV	0059 V*i* V*o*	Deallocate memory from heap, variable Input: Vi = address of memory to de-allocate Output: Vo = success flag
R_ISIZE	00C1	Return image size for COPYSR
	w_*X0* w_*Y0* w_*X1* w_*Y1* V*o*	
R_ISIZEV	00C2 Vi Vo	Return image size for COPYSR, variable Input: Vi = X min (left) Vi+1 = Y min (top) Vi+2 = Xmax (right) Vi+3 = Ymax (bottom) Output: Vo = image size in bytes
R_PIXEL	00C5 w_*x* w_*y* V*o*	Return color at pixel location
R_PIXELV	00C6 V*i* V*o*	Return color at pixel location, variable Input: Vi = X coordinate Vi+1 = Y coordinate Output: Vo = color
R_RAND	00B2 V*o*	Generate random number
R_RGB	0190 w_*channel* w_*index* Vo	Return RGB color of single palette entry

Functional Listing

Name	Hex Value and parameters	Description
R_RGBV	0191 V*i* V*o*	Return RGB color of single palette entry, variable Input: Vi = channel ID Vi+1 = color index (RAMDAC address) Output: V*o* = RGB color
R_TEXTDA	00C7 address V*o*	Return text dimensions, indirect address
R_TEXTDV	00C8 V*i* V*o*	Return text dimensions, variable Input: Vi = address of string Output: Vo = X extent Vo+1 = Y extent
R_TEXTDXA	0106 w_*mode* address V*o*	Return text dimensions, explicit format,
R_TEXTDXV	0107 V*i* V*o*	Return text dimensions, explicit format, variable Input: Vi = string format mode Vi+1 = address of string Output: V*o* = x extent V*o*+1 = y extent
R_TEXTP	00C9 word V*o*	Return text parameter
R_TEXTPV	00CA V*i* V*o*	Return text parameter, variable Input: Vi= function code Output: Vo= value of parameter requested
R_XYLADD	00CB w_*x* w_*y* V*o*	Convert XY to linear address
R_XYLADDV	00CC V*i* V*o*	Convert XY to linear address, variable Input: Vi = X coordinate Vi+1 = Y coordinate Output: Vo = corresponding linear address

Serial Port

Name	Hex Value and parameters	Description
SERBAUD	00D5 word	Set baud rate for 2691 UART
SERBAUDV	00D6 V	Set baud rate for 2691 UART, variable
SERMODE	00D7 word	Set serial port interrupt service mode
SERMODEV	00D8 V	Set serial port interrupt service mode, variable
SEROUT	00D9 word	Output character byte to serial port
SEROUTV	00DA V	Output character byte to serial port, variable
SERQFL	00DB	Flush serial port queue
SERUART	00DC address	Initialize 2691 UART
SERUARTV	00DD @V	Initialize 2691 UART, variable

Symbols

Name	Hex Value and parameters	Description
SSYM	00E2 w_*rotation* long	Draw “simple symbol”
SSYMV	00E3 V*i*	Draw “simple symbol”, variable Vi = rotation Vi+1 = address of symbol structure (or index of default)
SSYMX	00E4 w_*rotation* w_*width* w_*height* w_*pitch* address	Draw “simple symbol” (explicit parameters)
SSYMXV	00E5 Vi	Draw “simple symbol” (explicit param), variable Vi = rotation Vi+1 = symbol width Vi+2 = symbol height Vi+3 = symbol pitch Vi+4 = pointer to symbol (pixblt) data

Text

Name	Hex Value and parameters	Description
CTEXTA	0035 address	Print character text, indirect address
CTEXTI	0036 <string>	Print character text, immediate (in-line)
CTEXTV	0037 V	Print character text, variable
CTEXTLXY	0038 w_*x* w_*y*	Set current CTEXT location
CTEXTLXYV	0039 V*i*	Set current CTEXT location, variable Vi = X coordinate Vi+1 = Y coordinate
CTEXTMXY	003A w_*x* w_*y*	Set current CTEXT margin

Functional Listing

Name	Hex Value and parameters	Description
CTEXTMXYV	003B Vi	Set current CTEXT margin, variable Vi = X coordinate Vi+1 = Y coordinate
CTEXTXA	0102 word long	Print character text, explicit format, indirect address
CTEXTXV	0103 V*i*	Print character text, explicit format, variable Vi = String format mode Vi+1 = address of string
FONT	0056 long	Set current font
FONTV	0057 V	Set current font, variable
GTEXTA	005A address	Print graphics text, indirect address
GTEXTI	005B <string>	Print graphics text, immediate (in-line)
GTEXTV	005C V	Print graphics text, variable
GTEXTXA	0104 word long	Print graphics text, explicit format, indirect address
GTEXTXV	0105 word	Print graphics text, explicit format,variable Vi = String format mode Vi+1 = address of string
R_TEXTDA	00C7 address V*o*	Return text dimensions, indirect address
R_TEXTDV	00C8 V*i* V*o*	Return text dimensions, variable Input: Vi = address of string Output: Vo = X extent Vo+1 = Y extent
R_TEXTDXA	0106 w_*mode* address V*o*	Return text dimensions, explicit format, indirect address
R_TEXTDXV	0107 V*i* V*o*	Return text dimensions, explicit format, variable indirect Input: Vi = String format mode Vi+1 = address of string Output: Vo = X extent Vo+1 = Y extent
R_TEXTP	00C9 word V*o*	Return text parameter
R_TEXTPV	00CA V*i* V*o*	Return text parameter, variable Input: Vi= function code Output: Vo= value of parameter requested
TEXTP	00E8 w_code long	Set text parameter
TEXTPV	00E9 Vi	Set text parameter, variable Vi = function code Vi+1 = parameter value
TEXTSVC	00EA long	Select text service routine
TEXTSVCV	00EB V	Select text service routine, variable

Text Cursor

TXCSRON	0172 w_*flag*	Enable/disable text cursor display
TXCSRONV	0173 V	Enable/disable text cursor display, variable
TXCSRPAGE	0174 w_*channel* w_*page*	Configure text cursor for channel and page
TXCSRPAGEV	0175 V*i*	Configure text cursor for channel and page, variable Vi = text cursor channel ID Vi+1 = text cursor display page
TXCSRWIN	0192 w_*x0* w_*y0* w_*x1* w_*y1*	Set text cursor window
TXCSRWINV	0193 V*i*	Set text cursor window, variable Vi = xmin (left) Vi+1 = ymin (top) Vi+2 = xmax (right) Vi+3 = ymax (bottom)
TXCSRXY	0176 w_*x* w_*y*	Set text cursor location
TXCSRXYV	0177 V*i*	Set text cursor location, variable Vi = new text cursor x Vi+1 = new text cursor y
TXCURSOR	0170	Select text cursor

long_*cursor* long_*color1* long_*color2* addr_*save* long_*save_pitch* w_*blink_rate*

Functional Listing

Name	Hex Value and parameters	Description
TXCURSORV	0171 Vi	Select text cursor, variable
		Vi = address of cursor structure (or index of default)
		Vi+1 = shape #1 color
		Vi+2 = shape #2 color
		Vi+3 = save buffer address
		Vi+4 = save buffer pitch
		Vi+5 = blink rate
USCSRON	00F4 word	Set user cursor state on/off
USCSRONV	00F5 V	Set user cursor state on/off, variable
USCSRXY	00F6 w_x w_y	Set current user cursor location
USCSRXYV	00F7 Vi	Set current user cursor location, variable
		Vi = X coordinate
		Vi+1 = Y coordinate
USCURSOR	016E long_cursor long_color1 long_color2 addr_save long_save_pitch	Select user cursor
USCURSORV	016F Vi	Select user cursor, variable
		Vi = address of cursor structure (or index of default)
		Vi+1 = shape #1color
		Vi+2 = shape #2color
		Vi+3 = address of save buffer
		Vi+4 = save buffer pitch

Utilities

Name	Hex Value and parameters	Description
COPYRR	0027 addr_src addr_dest long_size	Copy RAM to RAM
COPYRRV	0028 Vi	Copy RAM to RAM, variable
		Vi = source address (linear)
		Vi+1 = destination address (linear)
		Vi+2 = length of block in bytes
NOOP	0000	No-operation (null)
R_XYLADD	00CB w_x w_y Vo	Convert XY to linear address
R_XYLADDV	00CC Vi Vo	Convert XY to linear address, variable
		Input:
		Vi = X coordinate
		Vi+1 = Y coordinate
		Output:
		Vo = corresponding linear address

Video Page Operations

Name	Hex Value and parameters	Description
CLRM	001A	Clear all video memory to 0 (BLACK)
CLRPAGE	017A w_channel w_page long_color w_waitflag	Clear channel/page to color
CLRPAGEV	017B Vi	Clear channel/page to color, variable
		Vi = channel ID
		Vi+1 = page#
		Vi+2 = color
		Vi+3 = wait flag
CLRPG	001B w_page long_color	Clear page to color
CLRPGV	001C Vi	Clear page to color, variable
		Vi = page#
		Vi+1 = color
CLRWIN	0188 long_color w_waitflag	Clear window to color
CLRWINV	0189 Vi	Clear window to color, variable
		Vi = color
		Vi+1 = wait flag
COPYPP	0025 w_src w_dest	Copy page to page
COPYPPV	0026 Vi	Copy page to page, variable
		Vi = source page#
		Vi+1 = destination page#
DPAGE	017C w_channel w_page w_waitflag	Set current display for channel and page
DPAGEV	017D Vi	Set current display for channel and page, variable
		Vi = channel ID
		Vi+1 = page#
		Vi+2 = wait flag
DPG	0046 word	Set current display page
DPGA	0047 address	Set current display page address
DPGV	0048 V	Set current display page, variable
PANX	0116 word	Pan display horizontally (absolute)
PANXV	0117 V	Pan display horizontally (absolute), variable

Functional Listing

Name	Hex Value and parameters	Description
PANY	0118 word	Pan display vertically (absolute)
PANYV	0119 V	Pan display vertically (absolute), variable
PANXY	011A word word	Pan display (absolute)
PANXYV	011B V*i*	Pan display (absolute), variable
PANXYR	011C word word	Pan display (relative)
PANXYRV	011D V*i*	Pan display (relative), variable
WPAGEB	017E *addr_DB addr_DB_params*	Initialize drawing parameters for draw buffer
WPAGEBV	017F V*i*	Initialize drawing parameters for draw buffer, variable Vi = address of draw buffer Vi+1 = address of draw buffer parameter structure
WPAGEC	0180 w_*channel* w_*page*	Initialize drawing parameters for channel and page
WPAGECV	0181 V*i*	Initialize drawing parameters for channel and page, variable Vi = channel ID Vi+1 = page#
WPG	00F9 word	Set current write page
WPGA	00FA address	Set current write page address
WPGV	00FB V	Set current write page, variable
ZOOM	011E word	Set display zoom factor (enlarge display)
ZOOMV	011F V	Set display zoom factor (enlarge display), variable

Fill Types for Area-Fill Opcodes

CIRS, CPFILL, CPFILLO, CPFILLR, ELPS, PFILL, PFILLO, PFILLR, RECTS, RRECTS, SECTS, SEGS, SEEDFILL (types 0,1 only)

w_type=0= solid color
w_type=1= stipple pattern
w_type=2= tile pattern

Line Types for Line-Draw Opcodes

ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINEO, PLINER, PLINES, PLINESR, RECT, RRECT, SECT, SEG

w_type=0= solid line
w_type=1= dash line (32 bit binary pattern)
w_type=2= dash line (arbitrary - segment-length word list)
w_type=3= fatline - solid
w_type=4= fatline - stipple pattern (binary)
w_type=5= fatline - tile pattern (pixel-mapped)
w_type=6= pen line - solid
w_type=7= pen line - stipple
w_type=8= pen line - tile

Add Immediate to Variable

ADIV long V*d*

Syntax: 0002 long word

Description: The 32-bit signed, immediate value is added to the value of the destination variable.

Related opcodes: ADVV, JUMPA, JUMPR

Flags Affected: NCVZ

Add Variables

ADVV V_s V_d

Syntax: 0003 word word

Description: The value of the source variable Vs is added to the value of the destination variable V_d. The result is stored in the destination variable. $V_d = V_d + V_s$.

Related opcodes: ADIV, JUMPA, JUMPR

Flags Affected: NCVZ

AND Immediate with Variable

ANDIV long V*d*

Syntax: 0006 long word

Description: The 32 bit immediate value is ANDed with the destination variable.

Related opcodes: ANDVV, JUMPA, JUMPR

Flags Affected: Z (other flags undefined)

AND Two Variables

ANDVV V_s V_d

Syntax: 0007 word word

Description: The source variable V_s is ANDed with the destination variable V_d. The result is stored in the destination variable, V_d. V_s remains unchanged.

Related opcodes: ANDIV, JUMPA, JUMPR

Flags Affected: Z (other flags undefined)

Draw Arc (immediate)

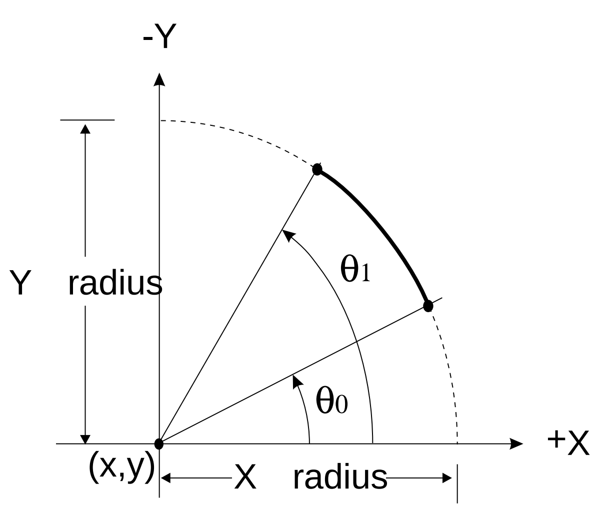
ARC type X Y theta0 theta1 Xradius Yradius

Syntax: 0008 word word word word word word word

Description: Draws an arc between two angles given in degrees counter-clockwise from the positive x axis.

Related opcodes: ARCTIC, R_ARC, R_ARCPTS, SECT, SEG

Line Types:
w_type=0= solid line
w_type=1= dash line (32 bit binary pattern)
w_type=2= dash line (arbitrary - segment-length byte list)
w_type=3= fatline - solid
w_type=4= fatline - stipple pattern (binary)
w_type=5= fatline - tile pattern (pixel-mapped)
w_type=6= pen line - solid
w_type=7= pen line - stipple
w_type=8= pen line - tile



Draw Arc (variable)

ARCV V*i*

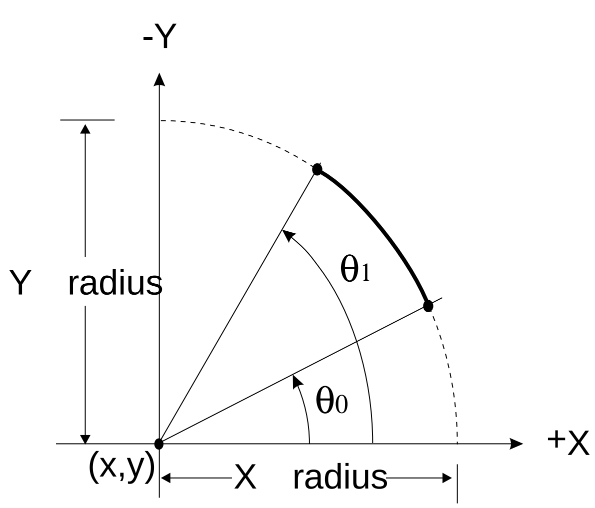
Syntax: 0009 word

Parameters

Vi	= line-type
Vi+1	= X coordinate of ARC center
Vi+2	= Y coordinate of ARC center
Vi+3	= start angle (θ_0)
Vi+4	= end angle (θ_1)
Vi+5	= X radius
Vi+6	= Y radius

Description: Draws an arc between two angles, given in degrees counterclockwise from the horizontal axis. 0 degrees is to the right along the X-axis. Vi is the first of 7 consecutive variables containing the parameters.

Related Opcodes: ARC, ARCTIC, R_ARC, R_ARCPTS



Draw Arc Tic-mark

ARCTIC type X Y angle length Xrad Yrad

Syntax: 000A word word word word word word word

Description: Draws a “tic-mark” for the specified arc. The mark is drawn at the position corresponding to the specified angle, perpendicular to the path of the arc at that point (i.e. along a radius line), of the specified length with the outer endpoint on the circumference of the arc.

Related Opcodes: ARC, R_ARCTIC, R_ARCPTS

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

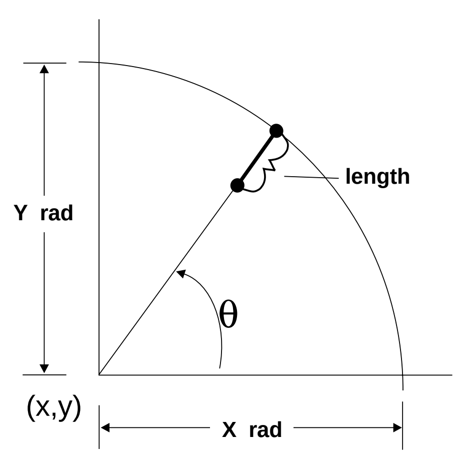
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



Draw Arc Tic-mark (variable)

ARCTICV V*i*

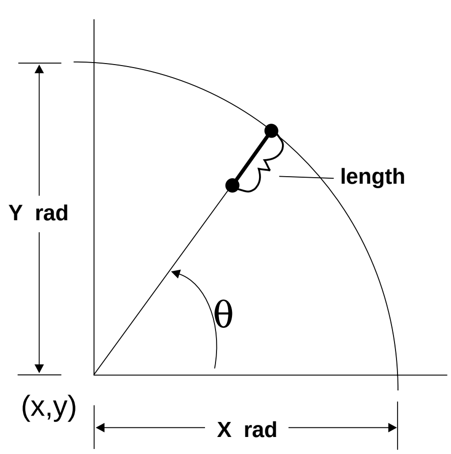
Syntax: 000B word

Parameters:

Vi	= type
Vi+1	= X coordinate
Vi+2	= Y coordinate
Vi+3	= angle
Vi+4	= tic-mark length
Vi+5	= X radius
Vi+6	= Y radius

Related Opcodes: ARC, R_ARCTIC

Description: Draws a “tic-mark” for the specified arc. The mark is drawn at the position corresponding to the specified angle, perpendicular to the path of the arc at that point (i.e. along a radius line), of the specified length with the outer endpoint on the circumference of the arc. Vi is the first of 7 consecutive variables containing the parameters.



Set Blinking Palette Entry

BLINK channel index rate color0 color1

Syntax: 0162 word word word long long

Description: Configures one of the palette entries as a “blinking color”. *channel* specifies which video output channel is to be affected; *index* identifies the palette entry (RAMDAC offset or address); *rate* is the blink half-period; color0 and color1 are RGB color values that are used alternately to re-program the palette entry to accomplish the blink function.

Related Opcodes: BLINKON, R_RGB, SETRGB

Discussion: The blink-color function is accomplished by re-programming the RAMDAC color look-up table at regular intervals. The pixel-values in VRAM remain constant, but the color that is displayed changes because the color-map is being changed. The palette index merely identifies the pixel-value or color-code that the blinking entry will be associated with. E.g., if index “9" is configured as a blinking color, then all pixels in the displayed portion of the relevant video output channel with a value of ”9" will appear to blink according to the parameters specified for that palette index.

The blink rate parameter is the number of vertical intervals (typically 1/60 second) between the onset of one color and it’s alternate. The blink “period” (i.e., the interval for a full cycle through both colors) is thus twice the rate.

The color parameters are specified as 32-bit RGB values for direct programming in the RAMDAC lookup table. Either the GETPALETTE or R_RGB opcodes can be used to determine the RGB value of an existing palette entry. RGB colors are formatted as follows:

FIELD	DESCRIPTION
bits 0...7	RED level (0..255)
bits 8...15	GREEN level (0..255)
bits 16...23	BLUE level (0..255)
bits 24...31	(padding) (0)

Up to 8 blinking palette entries may be defined. Subsequent entries are “synchronized” to the first entry with respect to the onset of blinking and “state” (color0, color1). If ALL blinking palette entries are programmed with the same rate, then ALL blinking colors will blink at exactly the same time and will maintain synchrony of colors (i.e., all entries will display “color0" at the same time, and likewise for color1).

Blinking commences immediately when a palette entry is programmed. A blinking palette entry may subsequently be disabled (and re-enabled) or deleted with the BLINKON opcode. The original RGB value for each palette entry is saved when it is first programmed, and will be restored when the blinking palette entry is deleted.

This function is not implemented on graphics boards that do not use a RAMDAC device (such as the RG-752).

Set Blinking Palette Entry, variable

BLINKV V*i*

Syntax: 0163 word

Parameters:

Vi	= channel ID
Vi+1	= color index (RAMDAC address)
Vi+2	= blink rate
Vi+3	= color 0 (RGB color)
Vi+4	= color 1 (RGB color)

Description: Configures one of the palette entries as a “blinking color”. channel specifies which video output channel is to be affected; index identifies the palette entry (RAMDAC offset or address); rate is the blink half-period; color0 and color1 are RGB color values that are used alternately to re-program the palette entry to accomplish the blink function. See BLINK for more information.

Related Opcodes: BLINK

Note: This function is not implemented on graphics boards that do not use a RAMDAC device (such as the RG-752).

Enable/Disable Blinking Palette Entry

BLINKON channel index code

Syntax: 0164 word word word

Description: Manages blinking palette entries according to the function code specified. The channel and index parameters identify the blinking palette entry to be affected.

CODE	FUNCTION
0	disable blinking palette entry (temporarily restores previous color and suspends blinking)
1	enable blinking palette entry (resumes blinking)
2	delete blinking palette entry (restores previous color and terminates blinking)

Blinking palette entries maintain synchronization even when disabled, such that when subsequently re-enabled they will resume with the proper state and timing.

Related Opcodes: BLINK

Enable/Disable Blinking Palette Entry, variable

BLINKONV V*i*

Syntax: 0165 word

Parameters:

Vi	= channel ID
Vi+1	= color index (RAMDAC address)
Vi+2	= function code

Description: Manages blinking palette entries according to the function code specified. The channel and index parameters identify the blinking palette entry to be affected.

CODE	FUNCTION
0	disable blinking palette entry (temporarily restores previous color and suspends blinking)
1	enable blinking palette entry (resumes blinking)
2	delete blinking palette entry (restores previous color and terminates blinking)

Blinking palette entries maintain synchronization even when disabled, such that when subsequently re-enabled they will resume with the proper state and timing.

Related Opcodes: BLINKON

Boolean Operation

BOOL word

Syntax: 000C word

Description: The value of the word specifies the Boolean operation for pixel processing. There are 22 options. The default operation is replace.

Word Value	Pixel Operation
0000	Replace
0001	Source AND destination=destination
0002	Source AND NOT destination=destination
0003	Zeros are written to destination
0004	Source OR NOT destination=destination
0005	Source XNOR destination=destination
0006	NOT destination=destination
0007	Source NOR destination=destination
0008	Source OR destination=destination
0009	Destination=destination
000A	Source XOR destination=destination
000B	NOT source AND destination=destination
000C	Ones are written to destination
000D	NOT source OR destination=destination
000E	Source NAND destination=destination
000F	NOT source=destination
0010	Source+destination=destination
0011	ADDS(source,destination)=destination
0012	Destination-source=destination
0013	SUBS(source,destination)=destination
0014	MAX(source,destination)=destination
0015	MIN(source,destination)=destination

Related opcodes: BOOLV, TRANS

Boolean Operation (variable)

BOOLV V

Syntax: 000D word

Description: The value stored in variable V specifies the Boolean operation for pixel processing. There are 22 options. The default operation is replace.

Value	Pixel Operation
0000	Replace
0001	Source AND destination=destination
0002	Source AND NOT destination=destination
0003	Zeros are written to destination
0004	Source OR NOT destination=destination
0005	Source XNOR destination=destination
0006	NOT destination=destination
0007	Source NOR destination=destination
0008	Source OR destination=destination
0009	Destination=destination
000A	Source XOR destination=destination
000B	NOT source AND destination=destination
000C	Ones are written to destination
000D	NOT source OR destination=destination
000E	Source NAND destination=destination
000F	NOT source=destination
0010	Source+destination=destination
0011	ADDS(source,destination)=destination
0012	Destination-source=destination
0013	SUBS(source,destination)=destination
0014	MAX(source,destination)=destination
0015	MIN(source,destination)=destination

Related opcodes: BOOL, TRANS

Set Background Color, variable

COLORBV V_{color}

Syntax: 001E word

Description: Specifies the background color for text characters and pixblt symbols.

The number of bits that must be specified corresponds to the channel pixel size. The color coding for the default 4-bit palette (CGA) is shown below:

VALUE	COLOR
0	BLACK
1	DARK BLUE
2	DARK GREEN
3	DARK CYAN
4	DARK RED
5	DARK MAGENTA
6	BROWN
7	LIGHT GRAY
8	DARK GRAY
9	LIGHT BLUE
10	LIGHT GREEN
11	LIGHT CYAN
12	LIGHT RED
13	LIGHT MAGENTA
14	YELLOW
15	WHITE

Related opcodes:COLORF

Discussion: Text characters and PIXBLT symbols are stored as binary patterns and are expanded to the specified color when written to VRAM. The value specified by COLORB determines the background color of the character or PIXBLT. Viewed another way, the zeros of the cell are transformed to the color specified by COLORB. For 256 color configurations, the color coding will depend on how the color look-up table is programmed. The default for 256 colors is D7, D6 = Blue, D5, D4, D3 = Green, and D2, D1, D0 = Red. For a bright red background, the above example would be coded as 00000111B.

Text/Line Foreground 32-bit Color

COLORF long

Syntax: 001F long

Description: Specifies the foreground color for text characters, pixblt symbols, and all other graphics.

The number of bits that must be specified corresponds to the channel pixel size. The color coding for the default 4-bit palette (CGA) is shown below:

VALUE	COLOR
0	BLACK
1	DARK BLUE
2	DARK GREEN
3	DARK CYAN
4	DARK RED
5	DARK MAGENTA
6	BROWN
7	LIGHT GRAY
8	DARK GRAY
9	LIGHT BLUE
10	LIGHT GREEN
11	LIGHT CYAN
12	LIGHT RED
13	LIGHT MAGENTA
14	YELLOW
15	WHITE

Related opcodes:COLORB

Discussion: Text and PIXBLT characters are stored as one bit values and are expanded to the specified color when written to VRAM. The value specified by COLORF determines the color of the character or PIXBLT. Viewed another way, the ones of the cell are transformed to the color specified by COLORF. For 256 color configurations, the color coding will depend on how the color look-up table is programmed. The default for 256 colors is D7, D6 = Blue, D5, D4, D3 = Green, and D2, D1, D0 = Red. For a bright red background, the above example would be coded as 00000111B.

Text/Line Foreground 32-bit Color (variable)

COLORFV V*color*

Syntax: 0020 word

Description: Specifies the foreground color for text characters, pixblt symbols, and all other graphics.

The number of bits that must be specified corresponds to the channel pixel size. The color coding for the default 4-bit palette (CGA) is shown below:

VALUE	COLOR
0	BLACK
1	DARK BLUE
2	DARK GREEN
3	DARK CYAN
4	DARK RED
5	DARK MAGENTA
6	BROWN
7	LIGHT GRAY
8	DARK GRAY
9	LIGHT BLUE
10	LIGHT GREEN
11	LIGHT CYAN
12	LIGHT RED
13	LIGHT MAGENTA
14	YELLOW
15	WHITE

Related opcodes:COLORB

Discussion: Text and PIXBLT characters are stored as one bit values and are expanded to the specified color when written to VRAM. The value specified by COLORF determines the color of the character or PIXBLT. Viewed another way, the ones of the cell are transformed to the color specified by COLORF. For 256 color configurations, the color coding will depend on how the color look-up table is programmed. The default for 256 colors is D7, D6 = Blue, D5, D4, D3 = Green, and D2, D1, D0 = Red. For a bright red background, the above example would be coded as 00000111B.

Set video configuration

CONFIG code

Syntax: 0021 word

Description: Specifies the video configuration for those boards that support multiple resolutions or programmable display formats, such as resolution and bits per pixel. Refer to the hardware manual for the particular graphics board to determine what modes are supported and the corresponding codes.

Related opcodes: CONFIGV, CONTREG

Set video configuration (variable)

CONFIGV V

Syntax: 0022 word

Parameters: V = configuration code

Description: Specifies the video configuration, for those boards that support multiple resolutions or programable display formats. Refer to the hardware manual for the particular graphics board to determine what modes are supported and the corresponding codes.

Related opcodes: CONFIG, CONTREG

Set/Clear User-configurable Bits

CONTREGX clr_mask set_mask

Syntax: 0136 word word

Description: Sets the user-configurable bits in the on-board hardware control register on the graphics board. One-bits in the clear-mask will clear the corresponding bit position in the hardware control register—similarly, one-bits in the set-mask will set the corresponding bit. Zero-bits in the masks have no effect. Refer to the User's manual for the particular graphics board for a description of user-configurable bits.

Related opcodes: CONFIG

Set/Clear User-configurable Bits, variable

CONTREGXV V*i*

Syntax: 0137 word

Parameters: V_i =clear-mask of user-configurable bits in HW control register

V_{i+1} = set-mask of user-configurable bits in HW control register

Description: Sets the user-configurable bits in the on-board hardware control register on the graphics board. One-bits in the clear-mask will clear the corresponding bit position in the hardware control register—similarly, one-bits in the set-mask will set the corresponding bit. Zero-bits in the masks have no effect. Refer to the User's manual for the particular graphics board for a description of user-configurable bits.

Related opcodes: CONTREGX

Copy Rectangle from Environment to Environment

COPYEE addr_sENV addr_dENV addr_pENV x_0 y_0 x_1 y_1 destX destY

Syntax: 013E long long long word word word word word word

Description: Copies a rectangle from the drawing context of one environment to that of another. Parameters controlling the processing applied to the rectangle copy operation are taken from a third environment. The x0, y0, x1, y1, parameters define the source rectangle and the destX and destY parameters specify the destination rectangle location (each relative to the pertinent draw-buffer origin and coordinate logical origin for their respective environments).

COPYEE facilitates the copying of rectangular areas to or from off-screen draw-buffers (as configured by R_ENVB, INITGCB, OR WPAGEB). Other copy functions (such as COPYSSP) primarily support copy operations to or from displayable draw-buffers (as configured by R_ENVC, INITGCC, or WPAGEC).

The parameters used from the various environments are as follows:

	SOURCE ENVIRONMENT
psize	pixel size (source and destination **MUST MATCH**)
offset	source draw-buffer address
dptch	source draw-buffer pitch
convdp	source draw-buffer convert-pitch value
org	source logical origin
wpage	page # if displayable buffer
wchannel	channel ID if displayable buffer

DESTINATION	**ENVIRONMENT**
offset	destination draw-buffer address
dptch	destination draw-buffer pitch
convdp	destination draw-buffer convert-pitch value
org	destination logical origin
wpage	page # if displayable buffer
wchannel	channel ID if displayable buffer
wstart, wend	clipping window parameters
uwstart, uwend	user clipping window parameters
clipmode	clipping mode

	PROCESS ENVIRONMENT
pmask	plane-mask
control	transparency mode, pixel processing operation

The process environment may be unique, or may be mapped to either the source or destination environment.

Related opcodes: COPYSS, COPYSSP

Copy Rectangle from Environment to Environment, variable

COPYEEV V*i*

Syntax: 013F word

Parameters:

Vi	= address of source environment
Vi+1	= address of destination environment
Vi+2	= address of parameter environment
Vi+3	= source rectangle X_0
Vi+4	= source rectangle Y_0
Vi+5	= source rectangle X_1
Vi+6	= source rectangle Y_1
Vi+7	= destination rectangle X (destX)
Vi+8	= destination rectangle Y (destY)

Description: Copies a rectangle from the drawing context of one environment to that of another. Parameters controlling the processing applied to the rectangle copy operation are taken from a third environment. The x0, y0, x1, y1, parameters define the source rectangle and the destX and destY parameters specify the destination rectangle location (each relative to the pertinent draw-buffer origin and coordinate logical origin for their respective environments).

COPYEE facilitates the copying of rectangular areas to or from off-screen draw-buffers (as configured by R_ENVB, INITGCB, OR WPAGEB). Other copy functions (such as COPYSSP) primarily support copy operations to or from displayable draw-buffers (as configured by R_ENVC, INITGCC, or WPAGEC).

Vi is the first of 9 consecutive variables containing the parameters.

Related opcodes: COPYEE

Copy Page to Page

COPYPP source destination

Syntax: 0025 word word

Description: Copies the contents of the video RAM *source* page into the video RAM *destination* page. Both pages are assumed to be in the current channel.

Related opcodes: COPYPPV, COPYSR, COPYRS, COPYSSP, COPYEE

Copy Page to Page (variable)

COPYPPV V*i*

Syntax: 0026 word

Parameters: Vi = source page #

Vi+1 = destination page #

Description: Copies the contents of the video RAM *source* page into the video RAM *destination* page. Both pages are assumed to be in the current channel. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: COPYPP

Copy RAM to RAM

COPYRR source_addr dest_addr size_bytes

Syntax: 0027 long long long

Description: Copies a block of bytes from the specified source address to the destination address.

Related opcodes: COPYRRV

Copy RAM to RAM (variable)

COPYRRV V*i*

Syntax: 0028 word

Parameters:

Vi	= source address (linear)
Vi+1	=destination address (linear)
Vi+2	= length of block in bytes

Description: Copies a block of bytes from the specified source address to the destination address. Vi is the first of 3 consecutive variables that contain the parameters.

Related opcodes: COPYRR

Copy Rectangle from RAM Buffer to Current Screen Page

COPYRS address X Y

Syntax: 0029 long word word

Description: Copies the image stored in DRAM at the address specified to the screen (VRAM), with the upper left corner of the image mapped to the specified X,Y location. The width and height of the image are taken from the image data structure.

Related opcodes: COPYSR, COPYSS, COPYRSV, COPYRSP

Discussion: The COPYSS opcode copies the contents of a rectangle from one part of the screen to another. The rectangle contents are copied to memory (DRAM) with the COPYSR opcode. COPYRS copies the rectangle from memory back to the screen.

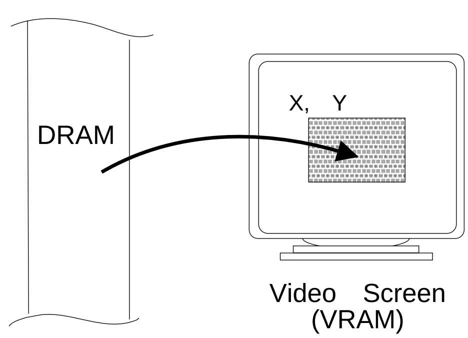
The image structure used by the COPYSR and COPYRS opcodes is as follows:

OFFS	SIZE	DESCRIPTION
0000	16	image width in pixels
0010	16	image height in pixels
0020	16	image depth (bits per pixel)
0030	16	row pitch of packed image (bits)
0040	32	(unused)
0060		beginning of image data

Note that on TMS34010 systems, the row pitch must be a multiple of 16. COPYSR always forces this restriction to provide for compatibility of image data between TMS34010 and TMS34020 systems. COPYRS uses the pitch specified in the image structure. The size returned by the R_ISIZE opcode already accounts for the image header specified above, as well as the pitch restriction enforced by COPYSR.

An image may be constructed in other ways without the use of COPYSR, but once generated can then be copied to the screen with COPYRS.

The image data structure is exactly the same as that used to specify a tile pattern. Therefore, tile patterns may easily be created from on-screen data by first copying a rectangle to a buffer with COPYSR (see TILE opcode).



Copy Rectangle from RAM Buffer to Current Screen Page, variable

COPYRSV V*i*

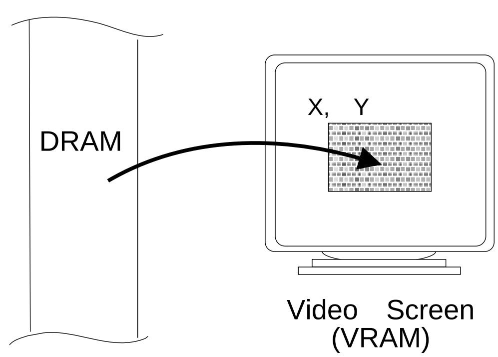
Syntax: 002A word

Parameters:

Vi	= source address (linear)
Vi+1	= destination X (screen address)
Vi+2	= destination Y (screen address)

Description: Copies the image stored in DRAM at the address specified to the screen (VRAM), with the upper left corner of the image mapped to the specified X,Y location. The width and height of the image are taken from the image data structure. Vi is the first of 3 consecutive variables that contain the parameters.

Related opcodes: COPYRS



Copy Rectangle from RAM Buffer to Screen Page

COPYRSP address page X Y

Syntax: 0166 long word word word

Description: Copies the image stored in DRAM at the address specified to a particular screen page (VRAM), with the upper left corner of the image mapped to the specified X,Y location. COPYRS is similar, but uses the current page as the destination. The width and height of the image are taken from the image data structure.

Related opcodes: COPYRS, COPYSRP, COPYSSP

Discussion: The COPYSS opcode copies the contents of a rectangle from one part of the screen to another. The rectangle contents are copied to memory (DRAM) with the COPYSR opcode. COPYRS copies the rectangle from memory back to the screen.

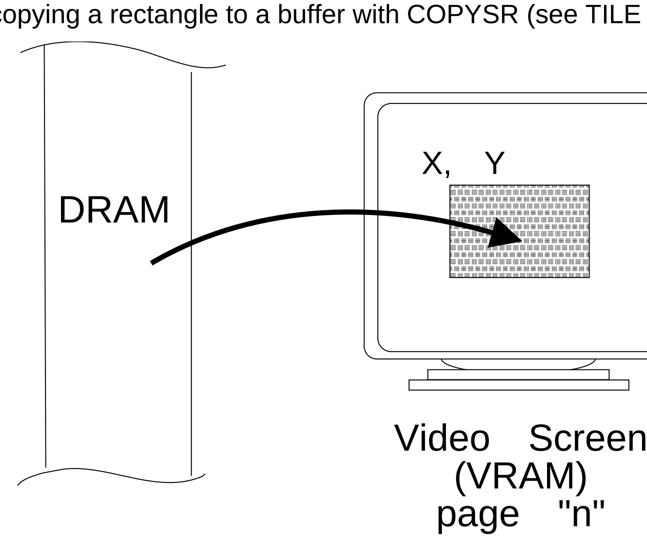
The image structure used by the COPYSR and COPYRS opcodes is as follows:

OFFS	SIZE	DESCRIPTION
0000	16	image width in pixels
0010	16	image height in pixels
0020	16	image depth (bits per pixel)
0030	16	row pitch of packed image (bits)
0040	32	(unused)
0060		beginning of image data

Note that on TMS34010 systems, the row pitch must be a multiple of 16. COPYSR always forces this restriction to provide for compatibility of image data between TMS34010 and TMS34020 systems. COPYRS uses the pitch specified in the image structure. The size returned by the R_ISIZE opcode already accounts for the image header specified above, as well as the pitch restriction enforced by COPYSR.

An image may be constructed in other ways without the use of COPYSR, but once generated can then be copied to the screen with COPYRS.

The image data structure is exactly the same as that used to specify a tile pattern. Therefore, tile patterns may easily be created from on-screen data by first copying a rectangle to a buffer with COPYSR (see TILE opcode).



Copy Rectangle from RAM Buffer to Screen Page, variable

COPYRSPV V*i*

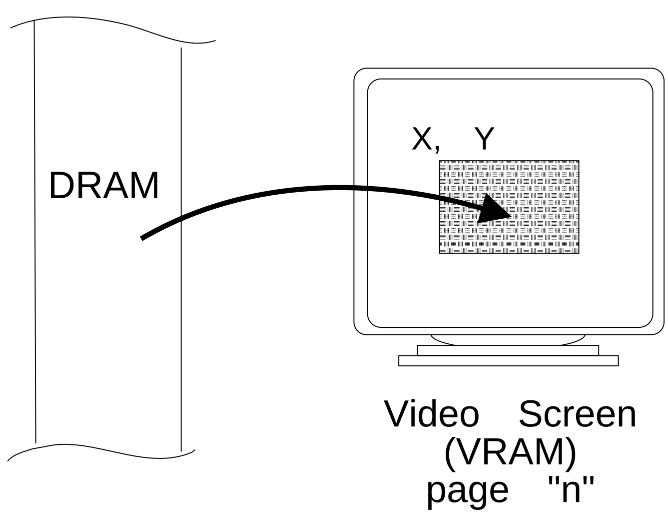
Syntax: 0167 word

Parameters:

Vi	= source address (linear)
Vi+1	= destination page
Vi+2	= destination X (screen address)
Vi+3	= destination Y (screen address)

Description: Copies the image stored in DRAM at the address specified to a particular screen page (VRAM), with the upper left corner of the image mapped to the specified X,Y location. COPYRS is similar, but uses the current page as the destination. The width and height of the image are taken from the image data structure. Vi is the first of 4 consecutive variables that contain the parameters.

Related opcodes: COPYRSP



Copy Rectangle from Current Screen to RAM Buffer

COPYSR *X0 Y0 X1 Y1* address

Syntax: 002B word word word word long

Description: Stores the image rectangle to memory starting at the *address* specified. The width and height of the image are stored in the image data structure. The size required for the image data buffer is determined with the R_ISIZE opcode.

Related opcodes: COPYRS, COPYSS, COPYSRP, R_ISIZE

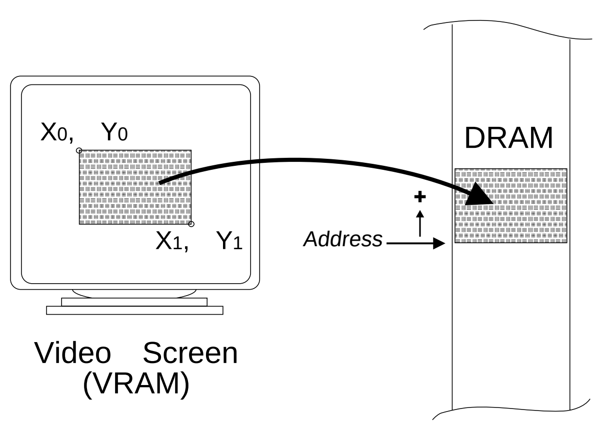
Discussion: The image structure used by the COPYSR and COPYRS opcodes is as follows:

OFFS	SIZE	DESCRIPTION
0000	16	image width in pixels
0010	16	image height in pixels
0020	16	image depth (bits per pixel)
0030	16	row pitch of packed image (bits)
0040	32	(unused)
0060		beginning of image data

Note that on TMS34010 systems, the row pitch must be a multiple of 16. COPYSR always forces this restriction to provide for compatibility of image data between TMS34010 and TMS34020 systems. COPYRS uses the pitch specified in the image structure. The size returned by the R_ISIZE opcode already accounts for the image header specified above, as well as the pitch restriction enforced by COPYSR.

An image may be constructed in other ways without the use of COPYSR, but once generated can then be copied to the screen with COPYRS.

The image data structure is exactly the same as that used to specify a tile pattern. Therefore, tile patterns may easily be created from on-screen data by first copying a rectangle to a buffer with COPYSR (see TILE opcode).



Copy Rectangle from Current Screen to RAM Buffer, variable

COPYSRV V*i*

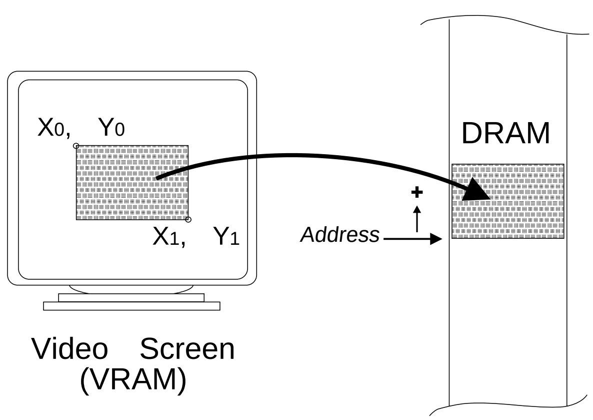
Syntax: 002C word

Parameters:

Vi	= Xmin (left)
Vi+1	= Ymin (top)
Vi+2	= Xmax (right)
Vi+3	= Ymax (bottom)
Vi+4	= destination address (linear)

Description: Stores the image rectangle to memory starting at the *address* specified. The width and height of the image are stored in the image data structure. The size required for the image data buffer is determined with the R_ISIZE opcode.

Related opcodes: COPYSR



Copy Rectangle from Screen Page to RAM Buffer

COPYSRP page X*0* Y*0* X*1* Y*1* address

Syntax: 0168 word word word word word long

Description: Stores the image rectangle on the page specified, to memory starting at the *address* specified. COPYSR is similar, but uses the current page as the source. The width and height of the image are stored in the image data structure. The size required for the image data buffer is determined with the R_ISIZE opcode.

Related opcodes: COPYSR, COPYRS, COPYSS, R_ISIZE

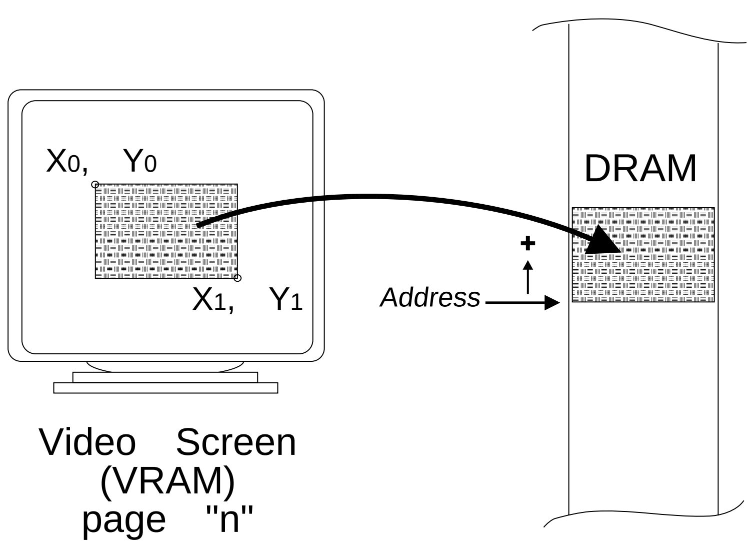
Discussion: The image structure used by the COPYSR and COPYRS opcodes is as follows:

OFFS	SIZE	DESCRIPTION
0000	16	image width in pixels
0010	16	image height in pixels
0020	16	image depth (bits per pixel)
0030	16	row pitch of packed image (bits)
0040	32	(unused)
0060		beginning of image data

Note that on TMS34010 systems, the row pitch must be a multiple of 16. COPYSR always forces this restriction to provide for compatibility of image data between TMS34010 and TMS34020 systems. COPYRS uses the pitch specified in the image structure. The size returned by the R_ISIZE opcode already accounts for the image header specified above, as well as the pitch restriction enforced by COPYSR.

An image may be constructed in other ways without the use of COPYSR, but once generated can then be copied to the screen with COPYRS.

The image data structure is exactly the same as that used to specify a tile pattern. Therefore, tile patterns may easily be created from on-screen data by first copying a rectangle to a buffer with COPYSR (see TILE opcode).



Copy Rectangle from Screen Page to RAM Buffer, variable

COPYSRPV V*i*

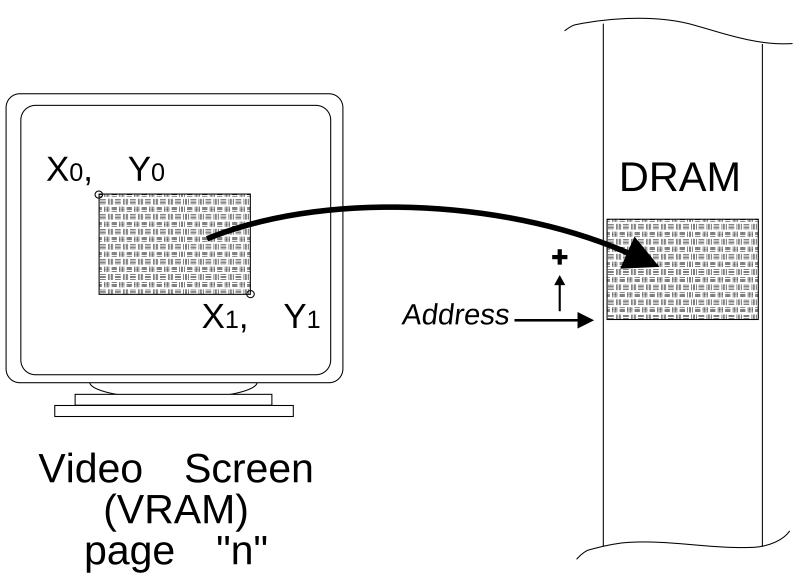
Syntax: 0169 word

Parameters:

Vi	= source page #
Vi+1	= Xmin (left)
Vi+2	= Ymin (top)
Vi+3	= Xmax (right)
Vi+4	= Ymax (bottom)
Vi+5	= destination address (linear)

Description: Stores the image rectangle on the page specified, to memory starting at the *address* specified. COPYSR is similar, but uses the current page as the source. The width and height of the image are stored in the image data structure. The size required for the image data buffer is determined with the R_ISIZE opcode.

Related opcodes: COPYSRP



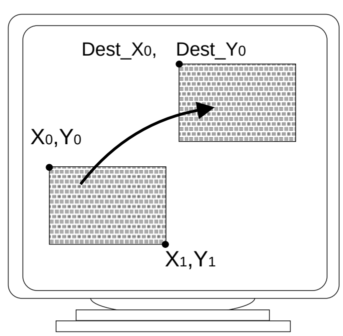
Copy Rectangle from Current Screen to Current Screen

COPYSS X_0 Y_0 X_1 Y_1 Dest_X_0 Dest_Y_0

Syntax: 002D word word word word word word

Description: Copies the image rectangle to another place on the screen. The upper left corner of the original image is located at X_0 Y_0 and the lower right corner at X_1, Y_1. The upper left corner of the copy is located by the (*Dest_X_0, Dest_Y_0)* parameters specified.

Related opcodes: COPYRS, COPYSR, COPYPP, COPYSSP



Video Screen
(VRAM)

Copy Rectangle from Current Screen to Current Screen, variable

COPYSSV Vi

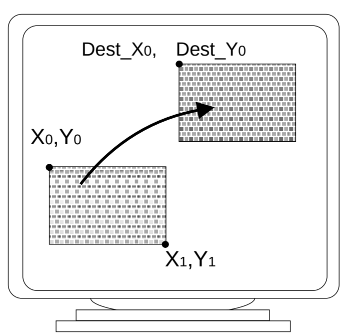
Syntax: 002E word

Parameters:

Vi	= X0 (left)
Vi+1	= Y0 (top)
Vi+2	= X1 (right)
Vi+3	= Y1 (bottom)
Vi+4	= Dest_X0 (screen address)
Vi+5	= Dest_Y0 (screen address)

Description: Copies the image rectangle to another place on the screen. The upper left corner of the oiginal image is located at X0 Y0 and the lower right corner at X1, Y1 The upper left corner of the copy is located by the (Dest_X0, Dest_Y0) parameters specified. Vi is the first of 6 consecutive variables containing the parameters.

Related opcodes: COPYSS



Video Screen
(VRAM)

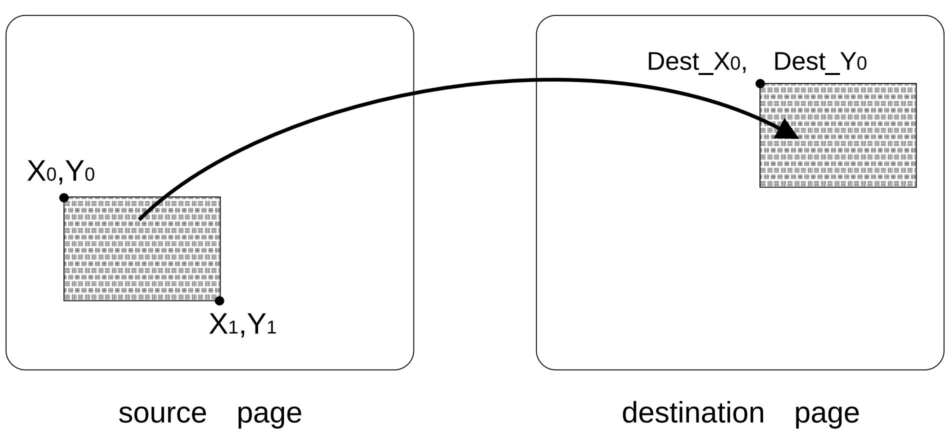
Copy Rectangle from Screen Page to Screen Page

COPYSSP Src_page Dest_page X0 Y0 X1 Y1 Dest_X0 Dest_Y0

Syntax: 016A word word word word word word word word

Description: Copies an image rectangle between video pages. The upper left corner of the source image is located at X0 Y0 and the lower right corner at X1, Y1. The upper left corner of the copy is located by the (Dest_X0, Dest_Y0) parameters specified. COPYSS is similar, but uses the current page for both the source and the destination.

Related opcodes: COPYSS, COPYRS, COPYSR, COPYPP



Copy Rectangle from Screen Page to Screen Page, variable

COPYSSPV V*i*

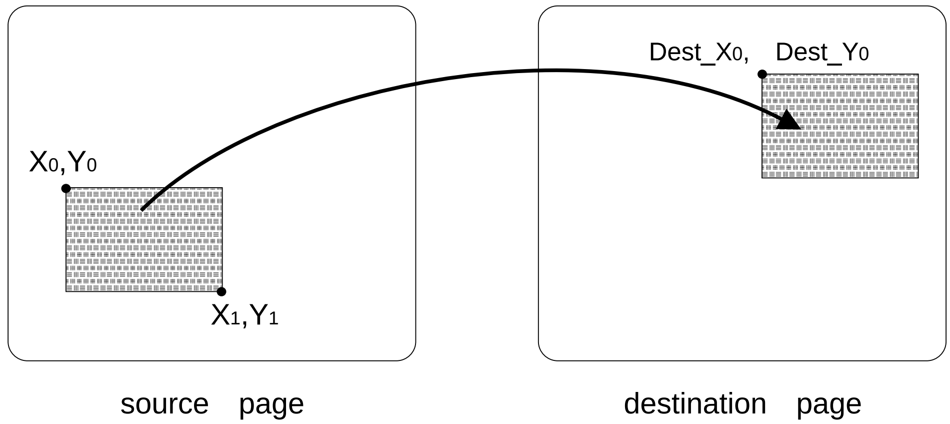
Syntax: 016B word

Parameters:

Vi	= source page
Vi+1	= destination page
Vi+2	= X0 (left)
Vi+3	= Y0 (top)
Vi+4	= X1 (right)
Vi+5	= Y1 (bottom)
Vi+6	= destination_X0 (screen address)
Vi+7	= destination_Y0 (screen address)

Description: Copies an image rectangle between video pages. The upper left corner of the original image is located at X0 Y0 and the lower right corner at X1, Y1 The upper left corner of the copy is located by the (Dest_X0, Dest_Y0) parameters specified. Vi is the first of 8 consecutive variables containing the parameters.

Related opcodes: COPYSSP



Convex Polygon Fill

CPFILL type count address

Syntax: 002F word word long

Description: Draws and fills the convex polygon defined by the coordinate list at the specified address. The polygon is drawn and filled according to the fill-type specified. The polygon must be convex.

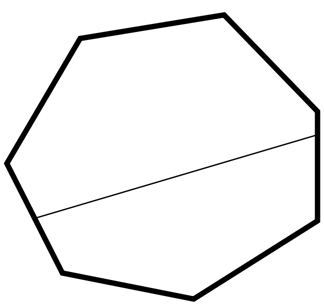
Related opcodes: CPFILLR,CPFILLO, PFILL

Discussion: Any two points of a convex polygon connected by a line will produce a line that lies inside the polygon, and does not cross any of the sides. All triangles are convex polygons.

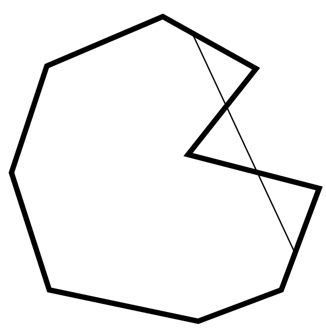
Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



CONVEX POLYGON



NON-CONVEX POLYGON

Convex Polygon Fill (variable indirect)

CPFILLV V*i*

Syntax: 0030 word

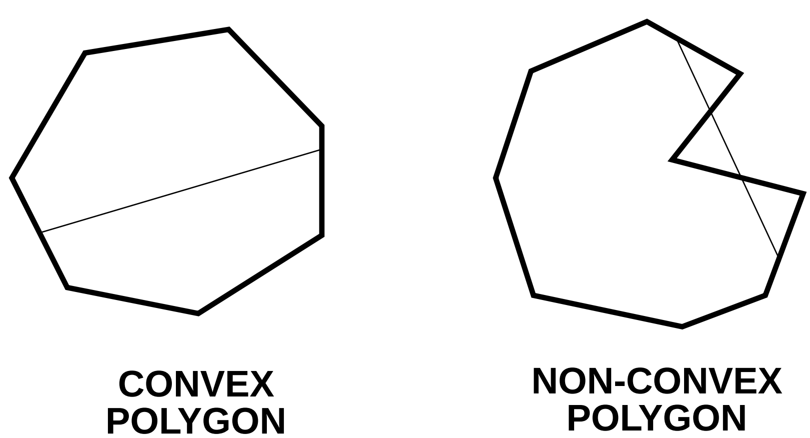
Parameters:

Vi	= fill type
Vi+1	= vertex count
Vi+2	= pointer to vertex list

Description: Draws and fills the convex polygon defined by the coordinate list at the specified address. The polygon is drawn and filled according to the fill-type spec. The polygon must be convex. Vi is the first of 3 consecutive variables that contain the parameters.

Related opcodes: CPFILLR, PFILL, PFILLR

Discussion: Any two points of a convex polygon connected by a line will produce a line that lies inside the polygon, and does not cross any of the sides. All triangles are convex polygons.



Convex Polygon Fill (offset)

CPFILLO type count address

Syntax: 0140 word word long

Description: Draws and fills the polygon specified by the coordinate list located at the address specified. The polygon is drawn and filled in the current color. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated.

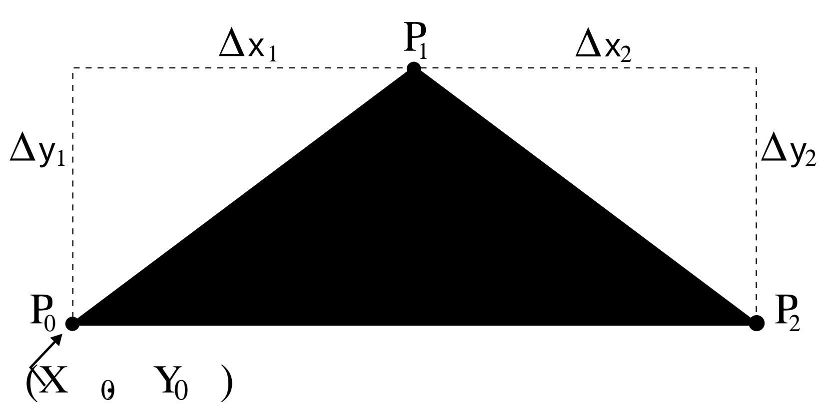
Related opcodes: CPFILL, CPFILLR, PFILL, PFILLO, PFILLR, MOVETO

Discussion: Any two points of a convex polygon connected by a line will produce a line that lies inside the polygon, and does not cross any of the sides. All triangles are convex polygons.

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



Convex Polygon Fill (offset), variable

CPFILLOV V*i*

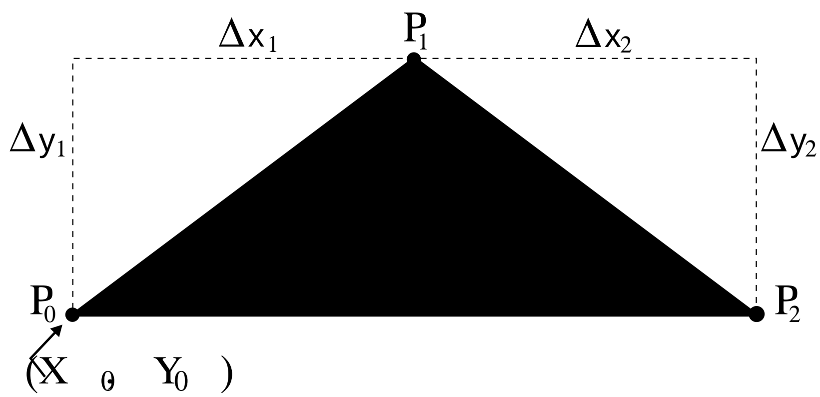
Syntax: 0141 word

Parameters:
Vi = fill type
Vi+1 = vertex count
Vi+2 = pointer to vertex list

Description: Draws and fills the polygon specified by the coordinate list located at the address specified. The polygon is drawn and filled in the current color. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated.

Related opcodes: CPFILLO

Discussion: Any two points of a convex polygon connected by a line will produce a line that lies inside the polygon, and does not cross any of the sides. All triangles are convex polygons.



Convex Polygon Fill (relative)

CPFILLR type count address

Syntax: 0031 word word long

Description: Draws and fills the polygon specified by the coordinate list located at the address specified. The polygon is drawn and filled in the current color. The points specified are relative to (offsets from) the current position.

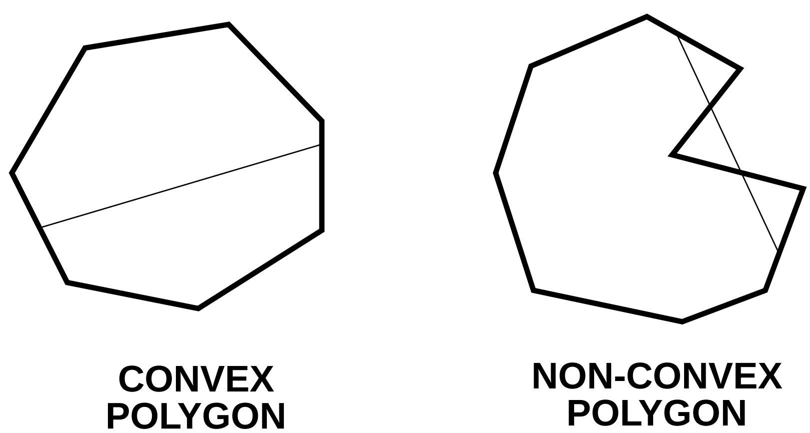
Related opcodes: CPFILL, CPFILLO, PFILL, PFILLR, MOVETO

Discussion: Any two points of a convex polygon connected by a line will produce a line that lies inside the polygon, and does not cross any of the sides. All triangles are convex polygons.

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



Convex Polygon Fill (variable relative)

CPFILLRV V*i*

Syntax: 0032 word

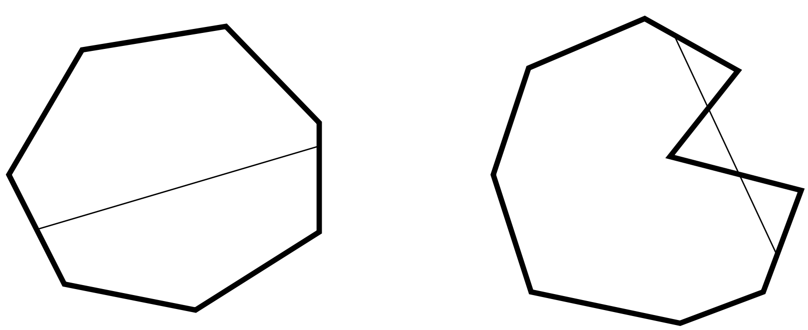
Parameters:

Vi	= fill type
Vi+1	= vertex count
Vi+2	= pointer to vertex list

Description: Draws and fills the polygon specified by the coordinate list located at the address specified. The polygon is drawn and filled in the current color. The points specified are relative to (offsets from) the current position. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: CPFILLR, PFILL, PFILLR, MOVETO

Discussion: Any two points of a convex polygon connected by a line will produce a line that lies inside the polygon, and does not cross any of the sides. All triangles are convex polygons.



CONVEX POLYGON **NON-CONVEX POLYGON**

Compare Immediate to Variable

CPIV long V*d*

Syntax: 0033 long word

Description: The value specified is compared to the destination variable. The carry flag is set if the value is greater than the variable value. The zero flag is set if the value equals the variable value. The value of the variable is unchanged.

Related opcodes: CPVV, JUMPA, JUMPR

Flags Affected: NCVZ

Compare Variables

CPVV V_s V_d

Syntax: 0034 word word

Description: The source variable specified is compared to the destination variable. The carry flag is set if the source variable is greater than the destination variable. The zero flag is set if the source variable equals the destination variable.The variables are unchanged.

Related opcodes: CPIV, JUMPA, JUMPR

Flags Affected: NCVZ

Character Text string (indirect)

CTEXTA address

Syntax: 0035 long

Description: Draw the characters contained in the string located at the *address* specified, starting from the text location specified by CTEXTLXY or from the end of the last CTEXT string, with a margin specified by CTEXTMXY. The position of CTEXT-generated text is located independently of the position used for graphics and GTEXT. The current position used for locating graphics and GTEXT is not used or altered by this instruction.

Related opcodes: CTEXT, CTEXTV, CTEXTLXY, CTEXTMXY, FONT, TEXTP, GTEXT

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguishes it from GTEXT which uses the current graphics x,y position (set with MOVETO). The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

The string must be in the proper format for the current text service routine (text driver). Alternate text drivers that support a different font set may expect different string formats. Or conversely, an alternate text driver may be installed in order to implement a specific string format as a matter of convenience owing to host-system byte-ordering or language conventions. The default string format for the default text driver is byte-packed, little-endian byte-order, NULL-terminated.

Character Text string (immediate)

CTEXTI <string>

Syntax: 0036 <string>

Description: Draw the characters contained in the string immediately following the opcode, starting from the text location specified by CTEXTLXY or from the end of the last CTEXT string, with a margin specified by CTEXTMXY. The position of CTEXT-generated text is located independently of the position used for locating graphics and GTEXT. The current position used for locating graphics and GTEXT is not used or altered by this instruction. The in-line string must be in the proper format for the current text service routine. If necessary, the end of the string must be padded so that the following opcode is word-aligned.

Related opcodes: CTEXT, CTEXTV, CTEXTLXY, CTEXTMXY, FONT, TEXTP, GTEXT

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguishes it from GTEXT which uses the current graphics x,y position (set with MOVETO). The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

The string must be in the proper format for the current text service routine (text driver). Alternate text drivers that support a different font set may expect different string formats. Or conversely, an alternate text driver may be installed in order to implement a specific string format as a matter of convenience owing to host-system byte-ordering or language conventions. The default string format for the default text driver is byte-packed, little-endian byte-order, NULL-terminated.

Character Text string (variable indirect)

CTEXTV V

Syntax: 0037 word

Description: Draw the characters contained in the string located at the *address* contained in the variable V, starting from the text location specified by CTEXTLXY or from the end of the last CTEXT string, with a margin specified by CTEXTMXY. The position of CTEXT-generated text is located independently of the position used for locating graphics and GTEXT. The current position used for locating graphics and GTEXT is not used or altered by this instruction.

Related opcodes: CTEXT, CTEXTI, CTEXTLXY, CTEXTMXY, FONT, TEXTP, GTEXT

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguished it from GTEXT which uses the current graphics x,y position (set with MOVETO). The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

The string must be in the proper format for the current text service routine (text driver). Alternate text drivers that support a different font set may expect different string formats. Or conversely, an alternate text driver may be installed in order to implement a specific string format as a matter of convenience owing to host-system byte-ordering or language conventions. The default string format for the default text driver is byte-packed, little-endian byte-order, NULL-terminated.

Set Current CTEXT location

CTEXTLXY X Y

Syntax: 0038 word word

Description: Sets the current X,Y location used by the CTEXT routines

Related opcodes: CTEXT, CTEXTMXY

Set current CTEXT location, variable

CTEXTLXYV V*i*

Syntax: 0039 word

Parameters: Vi = X coordinate

Vi+1 = Y coordinate

Description: Sets the current X,Y location used by the CTEXT routines. Vi is the first of 2 consecutive variables that contain the parameters.

Related opcodes: CTEXT, CTEXTMXY

Set current CTEXT margin

CTEXTMXY X Y

Syntax: 003A word word

Description: Sets the X,Y margin locations that will be used when processing control characters for CTEXT.

Related opcodes: CTEXT, CTEXTLXY

Discussion: A carriage-return character (CR=0dh) occurring in a string is processed by the CTEXT routine by setting the CTEXT current X location to the CTEXT X margin parameter last set with the CTEXTMXY opcode.

Set current CTEXT margin (variable)

CTEXTMXYV V*i*

Syntax: 003B word

Parameters: Vi = X coordinate

Vi+1 = Y coordinate

Description: Sets the X,Y margin locations that will be used when processing control characters for CTEXT. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: CTEXT, CTEXTLXY, CTEXTMXY

Discussion: A carriage-return character (CR=0dh) occurring in a string is processed by the CTEXT routine by setting the CTEXT current X location to the CTEXT X margin parameter last set with the CTEXTMXY opcode.

Character Text string, explicit format (indirect)

CTEXTXA mode address

Syntax: 0102 word long

Description: Draw the characters contained in the string located at the *address* specified, starting from the text location specified by CTEXTLXY or from the end of the last CTEXT string, with a margin specified by CTEXTMXY. The string is assumed to be in the format as specified by the mode parameter as follows:

mode	string format
0	byte-packed, NULL-terminated
1	byte-packed, byte-swapped, NULL-terminated

Related opcodes: CTEXT, CTEXTLXY, CTEXTMXY, FONT, TEXTP, GTEXT

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguishes it from GTEXT which uses the current graphics x,y position (set with MOVETO). The current position used for locating graphics and GTEXT is not used or altered by this instruction. The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

Character Text string, explicit format (variable indirect)

CTEXTXV

Syntax: 0103 word

Parameters: V_i = string format mode
V_{i+1} = address of string

Description: Draw the characters contained in the string located at the address contained in the variable V, starting from the text location specified by CTEXTLXY or from the end of the last CTEXT string, with a margin specified by CTEXTMXY. The string is assumed to be in the format as specified by the mode parameter as follows:

mode	string format
0	byte-packed, NULL-terminated
1	byte-packed, byte-swapped, NULL-terminated

Related opcodes: CTEXT, CTEXTLXY, CTEXTMXY, FONT, TEXTP, GTEXT

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguished it from GTEXT which uses the current graphics x,y position (set with MOVETO). The current position used for locating graphics and GTEXT is not used or altered by this instruction. The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

Call AFGIS Display List

CAL address

Syntax: 000E long

Description: The next display opcode to be processed will be at the address specified. A RTRN opcode is required at the end of the called display list.

Related opcodes: RTRN, CASM, CALV

Call AFGIS Display List (variable indirect)

CALV V

Syntax: 000F word

Description: The next display opcode to be processed is located at the address stored in the variable *V*. A RTRN opcode is required at the end of the called display list.

Related opcodes: RTRN, CASM, CAL

Call AFGIS Subroutine relative

CALR offset

Syntax: 010E word

Description: Calls an AFGIS subroutine at an address which is relative to the current program counter. The offset value is a signed word-offset from the beginning of the next AFGIS opcode. The new program counter address (PC') is calculated as PC' = PCn + (16*offset), where PCn is the address of the opcode following the CALR opcode.

Related opcodes: CAL

Call AFGIS Subroutine relative (variable)

CALRV V

Syntax: 010F word

Description: Calls an AFGIS subroutine at an address which is relative to the current program counter. The specified variable contains an offset value which is a signed word-offset from the beginning of the next AFGIS opcode. The new program counter address (PC') is calculated as PC' = PCn + (16*offset), where PCn is the address of the opcode following the CALR opcode.

Related opcodes:CAL

Call TMS340x0 Assembly Code

CASM address

Syntax: 0010 long

Description: This instruction allows the user to execute TMS340x0 assembly language code at the address specified. In-line parameters can be passed to the assembly routine through TMS340x0 register A13, which points to the first parameter. The called code must end with the assembly language RETS instruction.

Related opcodes:CAL, CASMV

Call TMS340x0 Assembly Code (variable indirect)

CASMV V

Syntax: 0011 word

Description: This instruction allows the user to execute TMS340x0 assembly language code at the address stored in variable *V*. In-line parameters can be passed to the assembly routine through TMS340x0 register A13, which points to the first parameter. The called code must end with the assembly language RETS instruction.

Related opcodes: CALV, CASM

Draw Circle

CIR type X Y radius

Syntax: 0012 word word word word

Description: A circle of the specified radius is drawn at the specified xy coordinate. The color of the circle is specified by the last COLORF opcode.

Related opcodes: CIRS, CIRV, elp

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

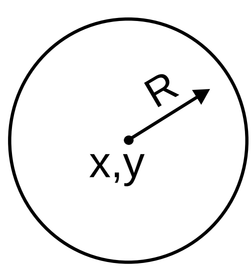
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



Draw Circle (variable)

CIRV V*i*

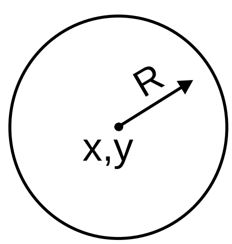
Syntax: 0013 word

Parameters:

Vi	= line type
Vi+1	= X
Vi+2	= Y
Vi+3	= radius

Description: A circle of the specified radius is drawn at the specified xy coordinate. The color of the circle is specified by the last COLORF opcode. Vi is the first of 4 consecutive variables containing the parameters.

Related opcodes: CIRSV, CIR



Draw a Solid Circle

CIRS type X Y radius

Syntax: 0014 word word word word

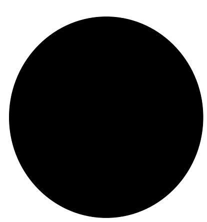
Description: A solid circle of the specified radius is drawn at the specified xy coordinate. The color of the circle is specified by the last COLORF opcode.

Related opcodes: CIR, CIRSV, elps

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



Draw a Solid Circle (variable)

CIRSV V*i*

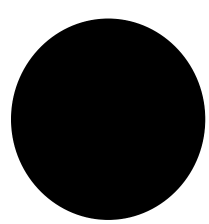
Syntax: 0015 word

Parameters:

Vi	= fill type
Vi+1	= X
Vi+2	= Y
Vi+3	= radius

Description: A solid circle of the specified radius is drawn at the specified xy coordinate. The color of the circle is specified by the last COLORF opcode. Vi is the first of 4 consecutive variables that contain the parameters

Related opcodes: CIRV, CIRS



Set Window Clipping Mode

CLIPMODE mode

Syntax: 0016 word

Description: Sets the window clipping mode according to the specified parameter.

Mode	Description
0	selects “default” clipping window (corresponds to channel resolution)
1	selects clipping window specified by CLIPWIN (relative to logical origin)
2	selects clipping window specified by CLIPWIN (absolute coordinates)

Related opcodes:CLIPWIN, CLIPMODEV

Set Window Clipping Mode (variable)

CLIPMODEV V

Syntax: 0017 word

Description: Sets the window clipping mode according to the specified parameter.

Mode	Description
0	selects “default” clipping window (corresponds to channel resolution)
1	selects clipping window specified by CLIPWIN (relative to logical origin)
2	selects clipping window specified by CLIPWIN (absolute coordinates)

Related opcodes:CLIPWINV, CLIPMODE

Set Clipping Window

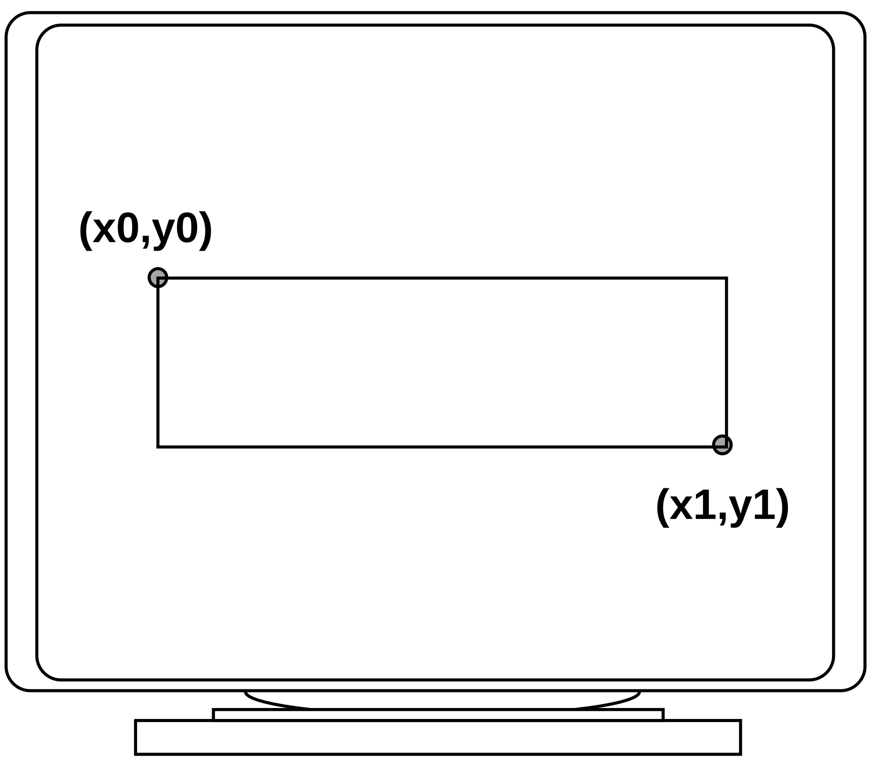
CLIPWIN X_0 Y_0 X_1 Y_1

Syntax: 0018 word word word word

Description: Sets the window clipping rectangle to x_0, y_0 (upper left corner of window) and x_1, y_1 (lower right corner of window). This opcode sets the clipping rectangle limits, where the points (X_0,Y_0) and (X_1, Y_1) are considered to be <u>inside</u> the clipping window. Use the CLIPMODE opcode to activate the clipping rectangle.

If clipmode=1 is specified, the clipping window coordinates are assumed to be relative to the logical origin. If clipmode=2 is specified, the clipping window coordinates are assumed to be absolute.

Related opcodes: CLIPMODE, CLIPWINV



Set Clipping Window (variable)

CLIPWINV V*i*

Syntax: 0019 word

Parameters:

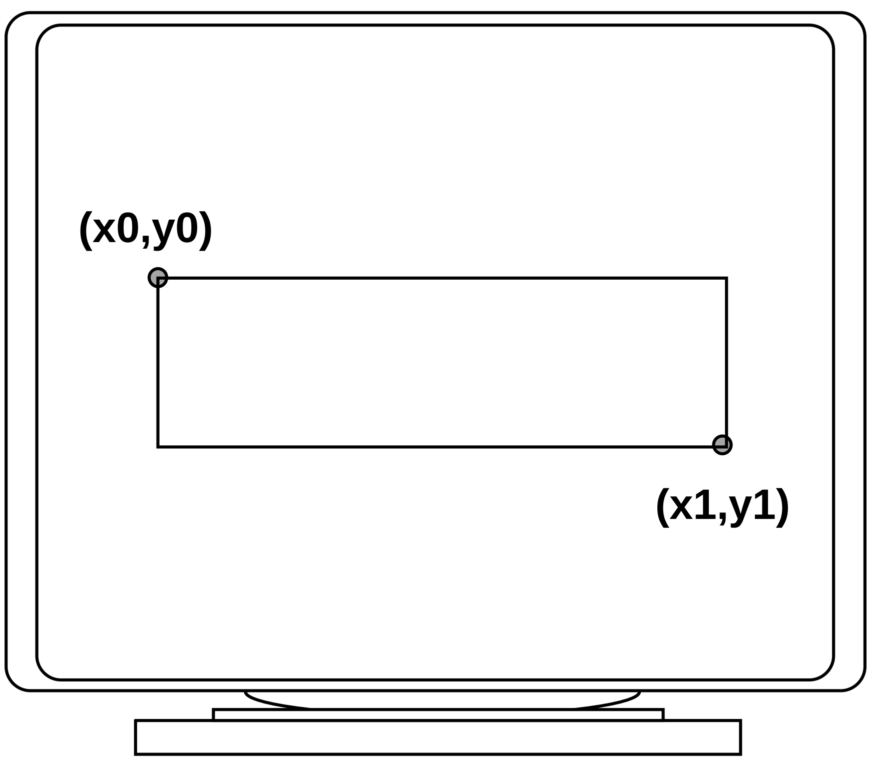
Vi	= X0 (left)
Vi+1	= Y0 (top)
Vi+2	= X1 (right)
Vi+3	= Y1 (bottom)

Description: Sets the window clipping rectangle to x_0, y_0 (upper left corner of window) and x_1, y_1 (lower right corner of window). This opcode sets the clipping rectangle limits, where the points (X_0,Y_0) and (X_1, Y_1) are considered to be inside the clipping window. Use the CLIPMODE opcode to activate the clipping rectangle.

If clipmode=1 is specified, the clipping window coordinates are assumed to be relative to the logical origin. If clipmode=2 is specified, the clipping window coordinates are assumed to be absolute.

Vi is the first of 4 consecutive variables containing the parameters.

Related opcodes: CLIPMODE, CLIPWIN



Clear all Video RAM

CLRM

Syntax: 001A

Description: Clears all video RAM (all channels) to zeros. Waits for the next vertical blanking interval before initiating the clear operation.

Related opcodes: CLRPG, CLRPAGE

Discussion: CLRM clears *all* video memory on *all* channels to *zero* (BLACK). CLRPG clears the video memory corresponding to a particular *page* on the *current* channel to a *color*. CLRPAGE allows the specification of the *channel, page,* and *color,* as well as whether to perform the clear immediately, or at the beginning of the next vertical blanking interval. CLRM and CLRPG always wait for the next vertical blanking interval.

Clear Channel/Page to Color

CLRPAGE channel page color waitflag

Syntax: 017A word word long word

Description: Clears the video RAM corresponding to the specified *channel* and *page* to a *color*. If *waitflag* is FALSE (zero) the clear operation is performed immediately, otherwise it is done at the beginning of the next vertical blanking interval.

Related opcodes: CLRM, CLRPG

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751, RG-752 and RG-753. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

For boards that only support a single channel, specify a channel ID of “0" (default underlay).

CLRM clears *all* video memory on *all* channels to *zero* (BLACK). CLRPG clears the video memory corresponding to a particular *page* on the *current* channel to a *color*. CLRPAGE allows the specification of the *channel, page,* and *color,* as well as whether to perform the clear immediately, or at the beginning of the next vertical blanking interval. CLRM and CLRPG always wait for the next vertical blanking interval.

Clear Channel/Page to Color, variable

CLRPAGEV V*i*

Syntax: 017B word

Parameters:

Vi		= channel ID
Vi+1		= page #color
Vi+2		= color
Vi+3		= waitflag

Description: Clears the video RAM corresponding to the specified *channel* and *page* to a *color*. V_i is the first of 4 consecutive variables containing the parameters. If *waitflag* is FALSE (zero) the clear operation is performed immediately, otherwise it is done at the beginning of the next vertical blanking interval.

Related opcodes:CLRPAGE

Clear Page to Color

CLRPG page color

Syntax: 001B word long

Description: Clears the video RAM corresponding to the specified *page* to a *color*. The clear operation is performed at the beginning of the next vertical blanking interval.

Related opcodes: CLRM, CLRPAGE

Discussion: CLRM clears *all* video memory on *all* channels to *zero* (BLACK). CLRPG clears the video memory corresponding to a particular *page* on the *current* channel to a *color*. CLRPAGE allows the specification of the *channel, page,* and *color,* as well as whether to perform the clear immediately, or at the beginning of the next vertical blanking interval. CLRM and CLRPG always wait for the next vertical blanking interval.

Clear Page to Color, variable

CLRPGV V*i*

Syntax: 001C word

Parameters: Vi = page #

Vi+1 = color

Description: Clears the video RAM corresponding to the specified *page* to a color. V*i* is the first of 2 consecutive variables containing the parameters.

Related opcodes: CLRPG

Clear Window to Color

CLRWIN color waitflag

Syntax: 0188 long word

Description: Clears the rectangle corresponding to the current clipping window to a color. If *waitflag* is FALSE (zero) the clear operation is performed immediately, otherwise it is done at the beginning of the next vertical blanking interval.

Related opcodes: CLIPWIN, CLIPMODE

Discussion: If the current graphics context is configured for an off-screen draw-buffer, the clear operation may always be performed immediately (waitflag = 0). Otherwise, it may be made to wait as required.

Clear Window to Color (variable)

CLRWINV V*i*

Syntax: 0189 word

Parameters: Vi = color

Vi+1 = wait flag

Description: Clears the rectangle corresponding to the current clipping window to a color. V_i is the first of 2 consecutive variables containing the parameters. If *waitflag* is FALSE (zero) the clear operation is performed immediately, otherwise it is done at the beginning of the next vertical blanking interval.

Related opcodes: CLRWIN

Set Background Color

COLORB long

Syntax: 001D long

Description: Specifies the background color for text characters and pixblt symbols.

The number of bits that must be specified corresponds to the channel pixel size. The color coding for the default 4-bit palette (CGA) is shown below:

VALUE	COLOR
0	BLACK
1	DARK BLUE
2	DARK GREEN
3	DARK CYAN
4	DARK RED
5	DARK MAGENTA
6	BROWN
7	LIGHT GRAY
8	DARK GRAY
9	LIGHT BLUE
10	LIGHT GREEN
11	LIGHT CYAN
12	LIGHT RED
13	LIGHT MAGENTA
14	YELLOW
15	WHITE

Related opcodes: COLORF

Discussion: Text characters and PIXBLT symbols are stored as binary patterns and are expanded to the specified color when written to VRAM. The value specified by COLORB determines the background color of the character or PIXBLT. Viewed another way, the zeros of the cell are transformed to the color specified by COLORB. For 256 color configurations, the color coding will depend on how the color look-up table is programmed. The default for 256 colors is D7, D6 = Blue, D5, D4, D3 = Green, and D2, D1, D0 = Red. For a bright red background, the above example would be coded as 00000111B.

Select Dash Pattern Continue Mode

DASHCON flag

Syntax: 003C word

Description: Sets the dash pattern continue mode according to the following parameter values: *0=continue mode off 1=continue mode on 2=reset continue pattern*

Related opcodes: DASHOFFS, DASHPATN

Discussion: When continue mode is off, each subsequent dashed line will begin at the same point in it's pattern (as specified by DASHOFFS). When continue mode is on, subsequent dashed lines will continue the line pattern from the last point drawn by the previous dashed line. Specifying a parameter of "2" will temporarily reset the dashed line pattern without canceling continue mode.

Select dash pattern continue mode (variable)

DASHCONV V

Syntax: 003D word

Parameters: V = function code

Description: Sets the dash pattern, continue mode according to the following parameter values: *0=continue mode off 1=continue mode on 2=reset continue pattern*

Related opcodes: DASHOFFS, DASHPATN

Discussion: When continue mode is off, each subsequent dashed line will begin at the same point in it's pattern (as specified by DASHOFFS). When continue mode is on, subsequent dashed lines will continue the line pattern from the last point drawn by the previous dashed line. Specifying a parameter of "2" will temporarily reset the dashed line pattern without canceling continue mode.

Set dash pattern offset

DASHOFFS offset

Syntax: 003E word

Description: Specifies the starting offset (in pixels) for dashed lines.

Related opcodes: DASHCON, DASHPATN

Discussion: The initial pixel drawn by a subsequent dashed line will correspond to the specified offset within the dashed line pattern. If the continue mode is **off** (“DASHCON 0") every subsequent dashed-line will begin at the same point in its pattern. If continue mode is **on** (”DASHCON 1"), then resetting the dashed line continue pattern with “DASHCON 2" will temporarily reset the pattern to begin at the offset specified by DASHOFFS.

Set dash pattern offset (variable)

DASHOFFSV V

Syntax: 003F word

Description: Specifies the starting offset (in pixels) for dashed lines.

Related opcodes: DASHCON, DASHPATN

Discussion: The initial pixel drawn by a subsequent dashed line will correspond to the specified offset within the dashed line pattern. If the continue mode is **off** (“DASHCON 0") every subsequent dashed-line will begin at the same point in its pattern. If continue mode is **on** (”DASHCON 1"), then resetting the dashed line continue pattern with “DASHCON 2" will temporarily reset the pattern to begin at the offset specified by DASHOFFS.

Select dashed line pattern

DASHPATN pointer

Syntax: 0040 long

Description: Select the current dashed line pattern.

Related opcodes: DASHCON, DASHOFFS, LINEPATN

Discussion: If the parameter value is <128, it is assumed to be the index of one of the default dashed line patterns, and the corresponding pattern is made active. Otherwise the parameter is assumed to be the address of a dashed line pattern definition structure as follows:

OFFS	SIZE	FIELD
0000	32	Pattern ID (default patterns)
0020	16	segment count (N)
0030		beginning of segment-length list (N entries, 16 bits each)

Each word in the segment-length list specifies the length of the corresponding segment in pixels. Even numbered segments are drawn with background color. Odd numbered segments are drawn with foreground color. Dashed lines are selected by specifying line type “2" for any of the line drawing opcodes (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, SECT)

Select dashed line pattern, variable

DASHPATNV V

Syntax: 0041 word

Parameters: V = address of dashed-line segment -length list (or index for default)

Description: Select the current dashed line pattern.

Related opcodes: DASHCON, DASHOFFS, LINEPATN

Discussion: If the parameter value is ,128, it is assumed to be the index of one of the default dashed line patterns, and the corresponding pattern is made active. Otherwise the parameter is assumed to be the address of a dashed line pattern definition structure as follows:

OFFS	SIZE	FIELD
0000	32	Pattern ID (default patterns)
0020	16	segment count (N)
0030		beginning of segment-length list (N entries, 16 bits each)

Each word in the segment-length list specifies the length of the corresponding segment in pixels. Even number segments are drawn with background color, Odd number segments are drawn with foreground color. Dashed lines are selected by specifying line type “2" for any of the line drawing opcodes (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, SECT)

Decrement Variable

DCRV V

Syntax: 0042 word

Description: The value of the specified variable is decremented by one.

Related opcodes: INCV, JUMPA, JUMPR

Flags Affected: NCVZ

Discussion: Variables are assumed to hold 32 bit signed values. Decrementing a variable subtracts one from its current value.

Delay Opcode Processing

DELAY delay

Syntax: 0043 word

Description: Display opcode processing is suspended while this opcode is being processed. The minimum delay occurs when delay = 1 and the maximum delay occurs when delay = 0FFFFh. The delay is approximately 1 millisecond per count.

Related opcodes: None

Discussion: Opcode processing is delayed for about 4 seconds with a delay parameter equal to 1000h. The minimum delay is about 1 millisecond and the maximum delay is about 1 minute. If a longer delay is required, use a repeat loop with the delay opcode inside.

Delay Opcode Processing (variable)

DELAYV V

Syntax: 0044 word

Description: Delays opcode processing by the value stored in the variable V. Display opcode processing is suspended while this opcode is being processed. The minimum delay occurs when delay = 1 and the maximum delay occurs when delay = 0FFFFh. The delay is approximately 1 millisecond per count.

Related opcodes: None

Discussion: Opcode processing is delayed for about 4 seconds with a delay parameter equal to 1000h. The minimum delay is about 1 millisecond and the maximum delay is about 1 minute. If a longer delay is required, use a repeat loop with the delay opcode inside.

Divide Variables

DIVV V*s* V*d*

Syntax: 0045 word word

Description: Vd=Vd/Vs. The value of the destination variable Vd, is divided by the value of the source variable Vs. The result is stored in the destination variable. Integer arithmetic is used, with the remainder being discarded. The zero flag is set if the result is zero. If Vs=0, Vd=0FFFFFFFFh.

Related opcodes: MLTV, JUMPA, JUMPR

Flags Affected: ZV (other flags undefined)

Discussion: Integer division is equivalent to a shift operation if the divisor (Vs) is a power of 2. Floating point division is possible with boards that have the TMS34082 math coprocessor.

Set Current Display for Channel and Page

DPAGE channel page waitflag

Syntax: 017C word word word

Description: Selects the *page* number of the particular channel whose contents are to be displayed on the video monitor.

If *waitflag* is FALSE (0), the operation is performed immediately, otherwise it is done at the beginning of the next vertical blanking interval. DPG is similar, but presumes the current channel.

Related opcodes: DPG, DPGA, WPG, WPAGEC

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751, RG-752 and RG-753. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

For boards that only support a single channel, specify a channel ID of “0" (default underlay).

Set Current Display for Channel and Page , variable

DPAGEV V*i*

Syntax: 017D word

Parameters:

V_i	= channel ID
V_{i+1}	= page#
V_{i+2}	= wait flag

Description: Selects the *page* number of the particular channel whose contents are to be displayed on the video monitor.

If *waitflag* is FALSE (0), the operation is performed immediately, otherwise it is done at the beginning of the next vertical blanking interval. DPG is similar, but presumes the current channel.

Related opcodes: DPAGE

Set Current Display Page

DPG word

Syntax: 0046 long

Description: Selects the page number whose contents are to be displayed on the current channel.

Related opcodes: DPAGE, DPGA

Discussion: DPG presumes the current channel. DPAGE allows the specification of a channel ID parameter.

Set Current Display Page Address

DPGA address

Syntax: 0047 long

Description: Selects the contents of video RAM starting at the address specified to be displayed on the video monitor.

Related opcodes: WPG, DPG

Discussion: This opcode allows the user to display the contents of video RAM starting at an arbitrary location.

This opcode should be used with care, and is provided for experienced users that have special display requirements.

Set Current Display Page, variable

DPGV V

Syntax: 0048 word

Description: Selects the page number whose contents are to be displayed on the current channel.

Related opcodes: DPG

Draw Ellipse

ELP type X Y Xrad Yrad

Syntax: 0049 word word word word word

Description: Draws an ellipse. The x axis of the ellipse is specified by Xrad, and the y axis of the ellipse is specified by Yrad.

Related opcodes: ELPS, CIR

Discussion: The ellipse is centered about the xy point. The current xy address remains unchanged after the ellipse is drawn. Use ELPS for a filled ellipse. The color of the ellipse is the last color specified by the COLORF opcode, and is drawn with the line style specified.

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

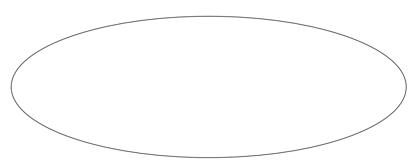
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



Draw Ellipse (variable)

ELPV V*i*

Syntax: 004A word

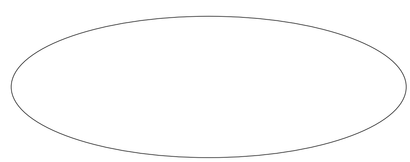
Parameters:

Vi	= line type
Vi+1	= X coordinate
Vi+2	= Y coordinate
Vi+3	= X radius
Vi+4	= Y radius

Description: Draws an ellipse. The x axis of the ellipse is specified by Xrad, and the y axis of the ellipse is specified by Yrad. Vi is the first of 5 consecutive variables containing the parameters.

Related opcodes: ELPS, CIR

Discussion: The ellipse is centered about the xy point. The current xy address remains unchanged after the ellipse is drawn. Use ELPS for a filled ellipse. The color of the ellipse is the last color specified by the COLORF opcode, and is drawn with the line style specified.



Fill Ellipse

ELPS type X Y Xrad Yrad

Syntax: 004B word word word word word

Description: Draws a solid or pattern filled ellipse. The x axis of the ellipse is specified by Xrad, and the y axis of the ellipse is specified with Yrad.

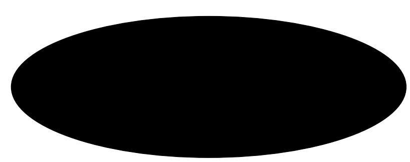
Related opcodes: ELPSV, CIRS

Discussion: The ellipse is centered about the xy point. The xy address remains unchanged after the ellipse is drawn. The colors of the ellipse are specified by COLORF, COLORB.

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



Fill Ellipse (variable)

ELPSV V*i*

Syntax: 004C word

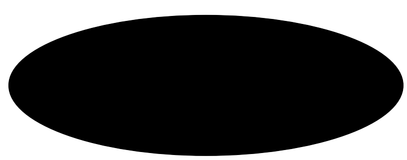
Parameters:

Vi	= fill type
Vi+1	= X coordinate
Vi+2	= Y coordinate
Vi+3	= X radius
Vi+4	= Y radius

Description: Draws a solid or pattern filled ellipse. The x axis of the ellipse is specified by Xrad, and the y axis of the ellipse is specified with Yrad. Vi is the first of 5 consecutive variables containing the parameters.

Related opcodes: ELPS, CIRS

Discussion: The ellipse is centered about the xy point. The xy address remains unchanged after the ellipse is drawn. The colors of the ellipse are specified by COLORF, COLORB.



End of Display List

EODL

Syntax: 0001

Description: Halts display opcode processing. Display lists must end with EODL. When EODL is processed, EODLFLAG in fixed RAM is set. If INTOUTMASK bit D0=1, an interrupt will be sent to the host via the HSTCTL register. The default is for all interrupts to the host to be disabled by INTOUTMASK (all bits are zero). Bits D0 through D7 enable interrupt messages 0-7 respectively from the graphics board to the host CPU. A one in INTOUTMASK enables the interrupt with its corresponding message.

End Repeat

ERPT

Syntax: 004F

Description: Specifies the end of a repeat loop. Use with RPT or RPTV opcodes.

Related opcodes: RPT, RPTV

Discussion: An ERPT belongs to the last RPT or RPTV opcode. Repeats may be nested.

Set Fatline Cap Style (immediate)

FATLNC word

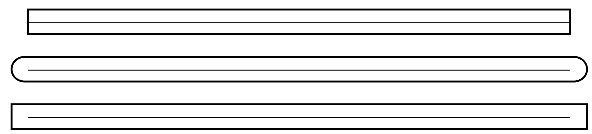
Syntax: 0050 word

Description: Sets the cap style for fatlines.

0 = butting end

1 = rounded end

2 = projecting end



Related opcodes: FATLNJ, FATLNW

Discussion: FATLNC determines the end shape of the line, FATLNJ determines how the line segments are connected, FATLNW specifies the width of the fatline. Fatlines are drawn by specifying line types 3,4, or 5 for any of the line-drawing opcodes. (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, SECT)

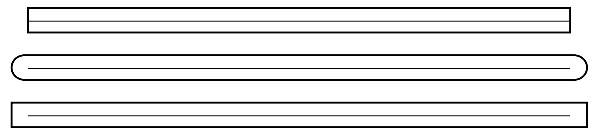
Set Fatline Cap Style (variable)

FATLNCV V

Syntax: 0051 word

Description: Sets the cap style for fatlines to the style stored in variable *V*.

0 = butting end
1 = rounded end
2 = projecting end



Related opcodes: FATLNJ, FATLNW

Discussion: FATLNC determines the end shape of the line, FATLNJ determines how the line segments are connected, FATLNW specifies the width of the fatline. Fatlines are drawn by specifying line types 3,4, or 5 for any of the line-drawing opcodes. (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, SECT)

Set Fatline Joint Style (immediate)

FATLNJ word

Syntax: 0052 word

Description: Sets the joint style for fatlines.

0 = mitered joint 

1 = rounded joint 

2 = bevelled joint 

Related opcodes: FATLNC, FATLNW

Discussion: FATLNC determines the end shape of the line, FATLNJ determines how the line segments are connected, FATLNW specifies the width of the fatline. Fatlines are drawn by specifying line types 3,4, or 5 for any of the line-drawing opcodes. (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, SECT)

Set Fatline Joint Style (variable)

FATLNJV V

Syntax: 0053 word

Description: Sets the joint style for fatlines to the style stored in variable *V*.

0 = mitered joint 

1 = rounded joint 

2 = bevelled joint 

Related opcodes: FATLNC, FATLNW

Discussion: FATLNC determines the end shape of the line, FATLNJ determines how the line segments are connected, FATLNW specifies the width of the fatline. Fatlines are drawn by specifying line types 3,4, or 5 for any of the line-drawing opcodes. (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, SECT)

Set Fatline Width (immediate)

FATLNW word

Syntax: 0054 word

Description: Sets the width (in pixels) for fatlines.

Related opcodes: FATLNC, FATLNJ

Discussion: FATLNC determines the end shape of the line, FATLNJ determines how the line segments are connected, FATLNW specifies the width of the fatline. Fatlines are drawn by specifying line types 3,4, or 5 for any of the line-drawing opcodes. (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, SECT)

Set Fatline Width (variable)

FATLNWV V

Syntax: 0055 word

Description: Sets the width (in pixels) for fatlines. The width value is stored in variable *V*.

Related opcodes: FATLNCV, FATLNJV

Discussion: FATLNC determines the end shape of the line, FATLNJ determines how the line segments are connected, FATLNW specifies the width of the fatline. Fatlines are drawn by specifying line types 3,4, or 5 for any of the line-drawing opcodes. (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, SECT)

Font Selection

FONT font#

Syntax: 0056 long

Description: Specifies the currently active character or font set. There may be up to thirty-two character sets installed in the graphics board, depending on the model. Only one set can be active at a time.

If the font parameter is within the range of 0 to 127, this number is used to specify one of the on-board default fonts. If the parameter is greater than 127, this number is assumed to be the starting address of the font data (no font index number is required). The addressing mode is primarly used for user-defined fonts which have been downloaded into RAM. When a new font is selected, the text spacing parameters are reset to ∅. (See Appendix C for additional font information.)

Related opcodes: CTEXT, GTEXT, TEXTP

Discussion: Only one character set, selected by the FONT opcode, is active at a time. The following table shows the font number, cell size and style of the built-in character sets. Character sets downloaded in the proper format to RAM may be selected by specifiying the address of the font data.

FONT NUMBER	CELL (W x H) DIMENSIONS	CHAR (W x H) DIMENSIONS	DESCENDER HEIGHT	STYLE
0	5 x 9	5 x 7	2	PLAIN
1	8 x 12	7 x 9	3	PLAIN
2	12 x 18	12 x 14	4	PLAIN
3	16 x 37	16 x 26	11	PLAIN
4	32 x 73	32 x 50	23	BOLD
5	7 x 14	5 x 13	1	PLAIN CONDENSED
6	8 x 11	7 x 9	2	PLAIN UC/UC
7	30 x 34	30 x 30	4	ITALICS
8	11 x 24	11 x 22	2	SWISS
9	24 x 33	24 x 30	3	BOLD
10	16 x 23	16 x 16	7	ITALICS
11	24 x 35	24 x 24	11	ITALICS
12	12 x 18	12 x 12	6	ITALICS
13	20 x 31	20 x 28	3	SWISS
14	30 x 47	30 x 42	5	SWISS
15	10 x 17	10 x 14	3	SWISS

Font Selection (variable)

FONTV V

Syntax: 0057 word

Description: Specifies the currently active character or font set. There may be up to thirty-two character sets installed in the graphics board, depending on the model (see table in FONT opcode description). Only one set can be active at a time. The font parameter is contained in the variable V.

If the font parameter is within the range of 0 to 127, this number is used to specify one of the on-board default fonts. If the parameter is greater than 127, this number is assumed to be the starting address of the font data (no font index number is required). The addressing mode is primarily used for user-defined fonts which have been downloaded into RAM. When a new font is selected, the text spacing parameters are reset to Æ. (See Appendix C for additional font information.)

Related opcodes: CTEXT, GTEXT, TEXTP

Discussion: Only one character set, selected by the FONT opcode, is active at a time. The table on the following page shows the font number, cell size, style, and font identification of the built-in character sets. Character sets downloaded in the proper format to RAM may be selected by specifiying the address of the font data.

Read Color Palette

GETPALETTE channel iColor nColors address

Syntax: 0198 word word word long

Description: Reads the RAMDAC color look-up table for the indicated graphics channel and writes the information to a buffer at the address specified. iColor is the index of the first palette entry to be read, and nColors is the number of (contiguous) entries to be read.

Related opcodes: SETPALETTE, R_RGB

Discussion: The buffer is written in exactly the same format as required by the SETPALETTE opcode (32-bit packing). Palette entries may then be modified and the new palette invoked with the SETPALETTE opcode.

The size required for the buffer is 8+(nColors*4) bytes (maximum size required for 256 entries would be 8+(256*4) bytes = 1032 bytes).

This opcode supports those boards having more than one graphics channel, such as the RG-751, RG-752 and RG-753. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

For boards that only support a single channel, specify a channel ID of “0" (default underlay).

Read Color Palette, variable

GETPALETTEV Vi

Syntax: 0199 word

Description: Reads the RAMDAC color look-up table for the indicated graphics channel and writes the information to a buffer at the address specified. iColor is the index of the first palette entry to be read, and nColors is the number of (contiguous) entries to be read.

Parameters:
Vi + 0 = channel ID
Vi + 1 = index of initial entry to be read
Vi + 2 = number of (contiguous) entries to read
Vi + 3 = address of buffer to copy palette to

Related opcodes: GETPALETTE

Discussion: The buffer is written in exactly the same format as required by the SETPALETTE opcode (32-bit packing). Palette entries may then be modified and the new palette invoked with the SETPALETTE opcode.

The size required for the buffer is 8+(nColors*4) bytes (maximum size required for 256 entries would be 8+(256*4) bytes = 1032 bytes).

This opcode supports those boards having more than one graphics channel, such as the RG-751, RG-752 and RG-753. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

For boards that only support a single channel, specify a channel ID of “0" (default underlay).

Print graphics text (indirect address)

GTEXTA address

Syntax: 005A long

Description: Draws the characters contained in the string at the address specified, starting from the current graphics X,Y location. Text processing updates the current X,Y location to the position at which the next character would have been drawn had there been another character in the string.

Related opcodes: CTEXT, FONT, TEXTP, MOVETO

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguishes it from GTEXT which uses the current graphics x,y position (set with MOVETO). The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

The string must be in the proper format for the current text service routine (text driver). Alternate text drivers that support a different font set may expect different string formats. Or conversely, an alternate text driver may be installed in order to implement a specific string format as a matter of convenience owing to host-system byte-ordering or language conventions. The default string format for the default text driver is byte-packed, little-endian byte-order, NULL-terminated.

Print graphics text, immediate (in-line)

GTEXTI <string>

Syntax: 005B <string>

Description: Draws the characters contained in the string at the address specified, starting from the current graphics X,Y location. Text processing updates the current X,Y location to the position at which the next character would have been drawn had there been another character in the string. The in-line string must be in the proper format for the current text service routine. If necessary, the end of the string must be padded so that the following opcodes is word-aligned.

Related opcodes: CTEXT, FONT, TEXTP, MOVETO

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguishes it from GTEXT which uses the current graphics x,y position (set with MOVETO). The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

The string must be in the proper format for the current text service routine (text driver). Alternate text drivers that support a different font set may expect different string formats. Or conversely, an alternate text driver may be installed in order to implement a specific string format as a matter of convenience owing to host-system byte-ordering or language conventions. The default string format for the default text driver is byte-packed, little-endian byte-order, NULL-terminated.

Print graphics text, variable

GTEXTV V

Syntax: 005C word

Description: Draws the characters contained in the string at the address specified, starting from the current graphics X,Y location. Text processing updates the current X,Y location to the position at which the next character would have been drawn had there been another character in the string.

Related opcodes: CTEXT, FONT, TEXTP, MOVETO

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguishes it from GTEXT which uses the current graphics x,y position (set with MOVETO). The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

The string must be in the proper format for the current text service routine (text driver). Alternate text drivers that support a different font set may expect different string formats. Or conversely, an alternate text driver may be installed in order to implement a specific string format as a matter of convenience owing to host-system byte-ordering or language conventions. The default string format for the default text driver is byte-packed, little-endian byte-order, NULL-terminated.

Print graphics text, explicit format (indirect address)

GTEXTXA mode address

Syntax: 0104 word long

Description: Draws the characters contained in the string at the address specified, starting from the current graphics X,Y location. Text processing updates the current X,Y location to the position at which the next character would have been drawn had there been another character in the string. The string is assumed to be in the format as specified by the mode parameter as follows:

mode	string format
0	byte-packed, NULL-terminated
1	byte-packed, byte-swapped, NULL-terminated

Related opcodes: GTEXT, CTEXT, FONT, TEXTP, MOVETO

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguishes it from GTEXT which uses the current graphics x,y position (set with MOVETO). The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

Print graphics text,explicit format (variable indirect)

GTEXTXV

Syntax: 0105 word

Parameters: Vi = string format mode
Vi+1 = address of string

Description: Draws the characters contained in the string at the address specified, starting from the current graphics X,Y location. Text processing updates the current X,Y location to the position at which the next character would have been drawn had there been another character in the string. The string is assumed to be in the format as specified by the mode parameter as follows:

mode	string format
0	byte-packed, NULL-terminated
1	byte-packed, byte-swapped, NULL-terminated

Related opcodes: CTEXT,GTEXT, FONT, TEXTP, MOVETO

Discussion: CTEXT maintains a separate screen position which can be set with CTEXTLXY. This distinguishes it from GTEXT which uses the current graphics x,y position (set with MOVETO). The default text service routine also treats CTEXT and GTEXT differently in that it recognizes and processes ASCII control characters (CR, LF) for CTEXT but not for GTEXT.

Increment Variable

INCV V

Syntax: 005D word

Description: The variable specified is incremented by one.

Related opcodes: DCRV, JUMPA, JUMPR

Flags Affected: NCVZ

Discussion: Variables are 32 bit signed values.

Initialize Graphics Context for Draw Buffer

INITGCB addr_GC addr_DB addr_DB_params

Syntax: 0148 long long long

Description: Initializes the specified graphics context to default values according to the draw buffer parameters specified. *addr_GC* is the address of the graphics context to be initialized; *addr_DB* is the linear address which identifies the upper-left corner (x=0, y=0) of draw-buffer memory; *addr_DB_params* is the address of a draw-buffer parameter structure formatted as follows:

OFFS	SIZE	FIELD
0000	16	draw-buffer width in pixels
0010	16	draw-buffer height in pixels
0020	16	draw-buffer depth (pixel size)
0030	16	draw-buffer pitch

Related opcodes: INITGCC, WPAGEB

Discussion: The draw-buffer pitch must be a multiple of 16, and is rounded up if necessary when initializing the corresponding graphics context field. Clipping is set ON (clipmode = 2) with the clipping window set to the draw-buffer width and height specified. If clipmode is subsequently set to 0 (“off”), the clipping window width reverts to the maximum value corresponding to the specified pitch.

Initialize Graphics Context for Draw Buffer, variable

INITGCBV V*i*

Syntax: 0149 word

Parameters:

Vi	=	address of graphics context
Vi+1	=	address of draw buffer
Vi+2	=	address of draw buffer parameter structure

Description: Initializes the specified graphics context to default values according to the draw-buffer parameters specified (see INITGCB).

Related opcodes: INITGCB

Initialize Graphics Context for Channel and Page

INITGCC addr_GC channel page

Syntax: 0178 long word word

Description: Initializes the specified graphics context to default values for the indicated channel and page number.

Related opcodes: INITGCB, WPAGEC

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751, RG-752 and RG-753. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

For boards that only support a single channel, specify a channel ID of “0" (default underlay).

Initialize Graphics Context for Channel and Page, variable

INITGCCV Vi

Syntax: 0179 word

Parameters:

Parameters:	Vi	= address of graphics context
	Vi+1	= channel ID
	Vi+2	= page#

Description: Initializes the specified graphics context to default values for the indicated channel and page number.

Related opcodes: INITGCC

Jump Absolute

JUMPA cc address

Syntax: 010A word long

Description: Jumps to the specified address if the condition code (cc) is true (see table).

Related opcodes: JUMPR, ADIV, ADVV, ANDIV, ANDVV, CPIV, CPVV, DCRV, INCV, LDIV, LDIVL, LDMV, LDMVL, LDPMV, LDPMVL, LDVV, MLTV, MODV, ORIV, ORVV, SBIV, SBVV, SLLV, SRLV, XORIV, XORVV

CODE	(HEX)	MNEMONIC	CONDITION	STATUS BITS
0000	(0)	UC	(unconditional)	(don't care)
0001	(1)	P	Result is positive	$\overline{N} \cdot \overline{Z}$
0010	(2)	LS	Dst lower or same as Src (unsigned)	$C + Z$
0011	(3)	HI	Dst higher than Src (unsigned)	$\overline{C} \cdot \overline{Z}$
0100	(4)	LT	Dst < Src (signed)	$(N \cdot \overline{V}) + (\overline{N} \cdot V)$
0101	(5)	GE	Dst ≥ Src (signed)	$(N \cdot V) + (\overline{N} \cdot \overline{V})$
0110	(6)	LE	Dst ≤ Src (signed)	$(N \cdot \overline{V}) + (\overline{N} \cdot V) + Z$
0111	(7)	GT	Dst > Src (signed)	$(N \cdot V \cdot \overline{Z}) + (\overline{N} \cdot \overline{V} \cdot \overline{Z})$
1000	(8)	C	Carry set on result	C
1000	(8)	B	Borrow set on result	C
1000	(8)	LO	Dst lower than Src (unsigned)	C
1001	(9)	NC	No Carry on result	$\overline{C}$
1001	(9)	NB	No Borrow on result	$\overline{C}$
1001	(9)	HS	Dst higher or same as Src (unsigned)	$\overline{C}$
1010	(A)	Z	Result = 0	Z
1010	(A)	EQ	Dst = Src (signed, unsigned)	Z
1011	(B)	NZ	Result ≠ 0	$\overline{Z}$
1011	(B)	NE	Dst ≠ Src (signed, unsigned)	$\overline{Z}$
1100	(C)	V	Overflow on result	V
1101	(D)	NV	No Overflow on result	$\overline{V}$
1110	(E)	N	Result is negative	N
1111	(F)	NN	Result is nonegative	$\overline{N}$

Jump Absolute (variable)

JUMPAV V*i*

Syntax: 010B word

Parameters: Vi = condition code (cc)

Vi+1 = address

Description: Jumps to the specified address if the condition code (cc) is true (see table).

Related opcodes: JUMPR, ADIV, ADVV, ANDIV, ANDVV, CPIV, CPVV, DCRV, INCV, LDIV, LDIVL, LDMV, LDMVL, LDPMV, LDPMVL, LDVV, MLTV, MODV, ORIV, ORVV, SBIV, SBVV, SLLV, SRLV, XORIV, XORVV

CODE	(HEX)	MNEMONIC	CONDITION	STATUS BITS
0000	(0)	UC	(unconditional)	(don't care)
0001	(1)	P	Result is positive	$\overline{N} \cdot \overline{Z}$
0010	(2)	LS	Dst lower or same as Src (unsigned)	$C + Z$
0011	(3)	HI	Dst higher than Src (unsigned)	$\overline{C} \cdot \overline{Z}$
0100	(4)	LT	Dst < Src (signed)	$(N \cdot \overline{V}) + (\overline{N} \cdot V)$
0101	(5)	GE	Dst ≥ Src (signed)	$(N \cdot V) + (\overline{N} \cdot \overline{V})$
0110	(6)	LE	Dst ≤ Src (signed)	$(N \cdot \overline{V}) + (\overline{N} \cdot V) + Z$
0111	(7)	GT	Dst > Src (signed)	$(N \cdot V \cdot \overline{Z}) + (\overline{N} \cdot \overline{V} \cdot \overline{Z})$
1000	(8)	C	Carry set on result	C
1000	(8)	B	Borrow set on result	C
1000	(8)	LO	Dst lower than Src (unsigned)	C
1001	(9)	NC	No Carry on result	$\overline{C}$
1001	(9)	NB	No Borrow on result	$\overline{C}$
1001	(9)	HS	Dst higher or same as Src (unsigned)	$\overline{C}$
1010	(A)	Z	Result = 0	Z
1010	(A)	EQ	Dst = Src (signed, unsigned)	Z
1011	(B)	NZ	Result ≠ 0	$\overline{Z}$
1011	(B)	NE	Dst ≠ Src (signed, unsigned)	$\overline{Z}$
1100	(C)	V	Overflow on result	V
1101	(D)	NV	No Overflow on result	$\overline{V}$
1110	(E)	N	Result is negative	N
1111	(F)	NN	Result is nonegative	$\overline{N}$

Jump relative

JUMPR cc offset

Syntax: 010C word word

Description: Jumps relative to the current program counter if the condition code (cc) is true (see table). The offset value is a signed word-offset from the beginning of the next AFGIS opcode. If the condition (cc) is true, the new program counter address (PC’) is calculated as PC’ = PCn + (16·offset), where PCn is the address of the opcode following the JUMPR opcode. If offset = 0, PC’ = PCn and execution continues with the following opcode regardless of the result of the test.

Related opcodes: JUMPA, ADIV, ADVV, ANDIV, ANDVV, CPIV, CPVV, DCRV, INCV, LDIV, LDIVL, LDMV, LDMVL, LDPMV, LDPMVL, LDVV, MLTV, MODV, ORIV, ORVV, SBIV, SBVV, SLLV, SRLV, XORIV, XORVV

CODE	(HEX)	MNEMONIC	CONDITION	STATUS BITS
0000	(0)	UC	(unconditional)	(don’t care)
0001	(1)	P	Result is positive	$\overline{N} \cdot \overline{Z}$
0010	(2)	LS	Dst lower or same as Src (unsigned)	$C + Z$
0011	(3)	HI	Dst higher than Src (unsigned)	$\overline{C} \cdot \overline{Z}$
0100	(4)	LT	Dst < Src (signed)	$(N \cdot \overline{V}) + (\overline{N} \cdot V)$
0101	(5)	GE	Dst ≥ Src (signed)	$(N \cdot V) + (\overline{N} \cdot \overline{V})$
0110	(6)	LE	Dst ≤ Src (signed)	$(N \cdot \overline{V}) + (\overline{N} \cdot V) + Z$
0111	(7)	GT	Dst > Src (signed)	$(N \cdot V \cdot \overline{Z}) + (\overline{N} \cdot \overline{V} \cdot \overline{Z})$
1000	(8)	C	Carry set on result	C
1000	(8)	B	Borrow set on result	C
1000	(8)	LO	Dst lower than Src (unsigned)	C
1001	(9)	NC	No Carry on result	$\overline{C}$
1001	(9)	NB	No Borrow on result	$\overline{C}$
1001	(9)	HS	Dst higher or same as Src (unsigned)	$\overline{C}$
1010	(A)	Z	Result = 0	Z
1010	(A)	EQ	Dst = Src (signed, unsigned)	Z
1011	(B)	NZ	Result ≠ 0	$\overline{Z}$
1011	(B)	NE	Dst ≠ Src (signed, unsigned)	$\overline{Z}$
1100	(C)	V	Overflow on result	V
1101	(D)	NV	No Overflow on result	$\overline{V}$
1110	(E)	N	Result is negative	N
1111	(F)	NN	Result is nonegative	$\overline{N}$

JUMPA relative (variable)

JUMPRV V*i*

Syntax: 010D word

Parameters: Vi = condition code (cc)

Vi+1 = signed word offset

Description: Jumps relative to the current program counter if the condition code (cc) is true (see table). The offset value is a signed word-offset from the beginning of the next AFGIS opcode. If the condition (cc) is true, the new program counter address (PC') is calculated as PC' = PCn + (16·offset), where PCn is the address of the opcode following the JUMPR opcode. If offset = 0, PC' = PCn and execution continues with the following opcode regardless of the result of the test.

Related opcodes: JUMPA, ADIV, ADVV, ANDIV, ANDVV, CPIV, CPVV, DCRV, INCV, LDIV, LDIVL, LDMV, LDMVL, LDPMV, LDPMVL, LDVV, MLTV, MODV, ORIV, ORVV, SBIV, SBVV, SLLV, SRLV, XORIV, XORVV

CODE	(HEX)	MNEMONIC	CONDITION	STATUS BITS
0000	(0)	UC	(unconditional)	(don't care)
0001	(1)	P	Result is positive	$\overline{N} \cdot \overline{Z}$
0010	(2)	LS	Dst lower or same as Src (unsigned)	C + Z
0011	(3)	HI	Dst higher than Src (unsigned)	$\overline{C} \cdot \overline{Z}$
0100	(4)	LT	Dst < Src (signed)	$(N \cdot \overline{V}) + (\overline{N} \cdot V)$
0101	(5)	GE	Dst ≥ Src (signed)	$(N \cdot V) + (\overline{N} \cdot \overline{V})$
0110	(6)	LE	Dst ≤ Src (signed)	$(N \cdot \overline{V}) + (\overline{N} \cdot V) + Z$
0111	(7)	GT	Dst > Src (signed)	$(N \cdot V \cdot \overline{Z}) + (\overline{N} \cdot \overline{V} \cdot \overline{Z})$
1000	(8)	C	Carry set on result	C
1000	(8)	B	Borrow set on result	C
1000	(8)	LO	Dst lower than Src (unsigned)	C
1001	(9)	NC	No Carry on result	$\overline{C}$
1001	(9)	NB	No Borrow on result	$\overline{C}$
1001	(9)	HS	Dst higher or same as Src (unsigned)	$\overline{C}$
1010	(A)	Z	Result = 0	Z
1010	(A)	EQ	Dst = Src (signed, unsigned)	Z
1011	(B)	NZ	Result ≠ 0	$\overline{Z}$
1011	(B)	NE	Dst ≠ Src (signed, unsigned)	$\overline{Z}$
1100	(C)	V	Overflow on result	V
1101	(D)	NV	No Overflow on result	$\overline{V}$
1110	(E)	N	Result is negative	N
1111	(F)	NN	Result is nonegative	$\overline{N}$

Set Keyboard Interrupt Mode

KBMODE mode

Syntax: 0060 word

Description: Sets the current keyboard interrupt mode according to the following parameter values: 0 = keyboard off 1 = keyboard on, polled mode 2 = keyboard on, interrupt mode

Related opcodes: KBQFL, KBRST

Discussion: Use “KBMODE1" at the beginning of a keyboard polling session. This opcode turns on the keyboard interface and allows the bus CPU to poll fixed-RAM location KBDFLAG to determine if keyboard data is available at RAM locations KBDATA0 and KBDATA1.

When the RAM location KBDATA0 or KBDATA1 contain a valid ASCII coded character from the keyboard, KBDFLAG will be set to one. After the bus CPU reads the data from KBDATA0 or KBDATA1, it must clear KBDFLAG. Otherwise KBDATA0 and KBDATA1 will not be updated by the graphics processor, and keyboard data will be stored in the keyboard queue.

The keyboard is serviced with a 60Hz interrupt. Keyboard data stored in the keyboard queue is read at 60Hz, and KBDATA0 and KBDATA1 are updated at this rate.

Set Keyboard Interrupt Mode (variable)

KBMODEV V

Syntax: 0061 word

Description: Sets the current keyboard interrupt mode according to the following parameter values: 0 = keyboard off 1 = keyboard on, polled mode 2 = keyboard on, interrupt mode

Related opcodes: KBQFL, KBRST

Discussion: Use “KBMODE1" at the beginning of a keyboard polling session. This opcode turns on the keyboard interface and allows the bus CPU to poll fixed-RAM location KBDFLAG to determine if keyboard data is available at RAM locations KBDATA0 and KBDATA1.

When the RAM location KBDATA0 or KBDATA1 contain a valid ASCII coded character from the keyboard, KBDFLAG will be set to one. After the bus CPU reads the data from KBDATA0 or KBDATA1, it must clear KBDFLAG. Otherwise KBDATA0 and KBDATA1 will not be updated by the graphics processor, and keyboard data will be stored in the keyboard queue.

The keyboard is serviced with a 60Hz interrupt. Keyboard data stored in the keyboard queue is read at 60Hz, and KBDATA0 and KBDATA1 are updated at this rate.

Flush Keyboard Queue

KBQFL

Syntax: 0062

Description: Clears the keyboard queue.

Related opcodes: KBRST, KBMODE, KBMODEV

Discussion: Use this opcode to clear the keyboard queue. The keyboard reset opcode KBRST also clears the keyboard. However, the keyboard interface may be turned on or off with the KBMODE opcode without affecting the keyboard queue. One use of the KBQFL is to clear the keyboard queue when the keyboard interface is turned on and the contents of the queue are unknown, perhaps holding data from a previous time when the interface was on.

Keyboard Reset

KBRST

Syntax: 0063

Description: Resets the keyboard. This opcode must be used once before any other keyboard opcodes can be used.

Related opcodes: KBMODE, KBMODEV, KBQFL

Discussion: Use this opcode before initiating keyboard communication. This opcode resets the keyboard. KBRST need only be issued once, but it must be the first keyboard opcode to be processed.

Keyboard Test

KBTEST

Syntax: 0064

Description: Runs the built-in keyboard test.

Related opcodes: KBMODE, KBMODEV, KBQFL

Discussion: The keyboard test displays DOS equivalent code on the video monitor screen. Use this opcode to verify the code for each character on the keyboard, and to verify that the keyboard interface is working.

Load Immediate to Variable

LDIV word V*d*

Syntax: 0065 word word

Description: The variable specified is loaded with the signed 16 bit immediate value. The value specified is sign extended when it is loaded into the variable.

Related opcodes: LDIVL, JUMPA, JUMPR

Flags Affected: NZ (other flags undefined)

Discussion: LDIV provides the convenience of a 16-bit parameter for loading into a variable. The variable, however, is loaded with the equivalent 32-bit value, i.e. LDIV -3 V6 and LDIVL -3 V6 both load V6 with the same bit pattern.

Load Immediate (long) to Variable

LDIVL long V*d*

Syntax: 0066 long word

Description: Loads the 32 bit signed long value specified into the specified variable.

Related opcodes: LDIV, JUMPA, JUMPR

Flags Affected: NZ (other flags undefined)

Discussion: Use this opcode to initialize a variable with 32 bits of data.

Load Memory to Variable

LDMV address V*d*

Syntax: 0067 long word

Description: The 16 bits of data at the memory location specified are loaded into the specified variable and are sign extended.

Related opcodes: LDMVL, LDVM, LDVML, JUMPA, JUMPR

Flags Affected: NZ (other flags undefined)

Discussion: The memory contents are unchanged. This opcode provides a way to convert a 16 bit signed value to a 32 bit signed variable. Use LDVM to move data from a variable to memory. Use LDMVL to load a 32 bit signed address value from memory to a variable.

Load Memory (long) to Variable

LDMVL address V*d*

Syntax: 0068 long word

Description: The 32 bits of data at the memory location specified are loaded into the specified variable.

Related opcodes: LDMV, LDVM, LDVML, JUMPA, JUMPR

Flags Affected: NZ (other flags undefined)

Discussion: The memory contents are unchanged. Use LDMV to load 16 bits of memory data to a variable.

Load AFGIS Program Counter to Variable

LDPCV V

Syntax: 0069 word

Description: The current AFGIS program counter (register A13) is moved to the variable specified. The value loaded is the address of the next AFGIS opcode.

Related opcodes: XCHGPC

Load Memory to Variable (indirect)

LDPMV @V V*d*

Syntax: 006A word word

Description: The 16 bits of data pointed to by @V are loaded into the destination variable and are sign extended.

Related opcodes: LDPMVL, JUMPA, JUMPR

Flags Affected: NZ (other flags undefined)

Load Memory (long) to Variable (indirect)

LDPMVL @V V_d

Syntax: 006B word word

Description: Loads 32 bits of data from memory pointed to by @V into the destination variable, V_d.

Related opcodes: LDPMV, JUMPA, JUMPR

Flags Affected: NZ (other flags undefined)

Load AFGIS Stack Pointer to Variable

LDSPV V

Syntax: 006C word

Description: The AFGIS stack pointer from the current environment is moved into the variable specified. The stack pointer remains unchanged.

Related opcodes: XCHGSP

Load Variable to Memory

LDVM V_S address

Syntax: 006D word long

Description: The 16 lsbs in the variable specified are moved to the memory location specified by address. The data in the variable remains unchanged.

Related opcodes: LDVML

Discussion: This opcode provides a way to save the contents of a variable for later use. Use LDMV to move the data back from memory to a selected variable. Use LDVML to move 32 bits from the variable to memory.

Load Variable (long) to Memory

LDVML V_S address

Syntax: 006E word long

Description: The 32 bits of data in the variable specified are moved to the memory location specified by address. The data in the variable remains unchanged.

Related opcodes: LDVM

Discussion: This opcode provides a way to save the contents of a variable for later use. Use LDVML to move the data back from memory to a selected variable. Use LDVM to move only 16 bits of data to memory.

Load Variable to Memory (indirect)

LDVPM V_S @V

Syntax: 006F word word

Description: Loads the 16 bit lsbs in V_S to memory location pointed to by @V.

Related opcodes: LDVPML

Load Variable (long) to Memory (indirect)

LDVPML V_S @V

Syntax: 0070 word word

Description: Loads the 32 bits of the variable specified by V_S into the memory location pointed to by @V.

Related opcodes: LDVPM

Load Variable

LDVV V_s V_d

Syntax: 0071 word word

Description: The value of the source variable is loaded into the destination variable. V_d=V_s.

Related opcodes:LDIV, LDIVL, JUMPA, JUMPR

Flags Affected: NZ (other flags undefined)

Set RED/GREEN LEDs

LED flag

Syntax: 0072 word

Description: The function of the RED and GREEN LEDs is set according to the value of the parameter.

BINARY	(HEX)	LED	ACTION
xx00	(0)	GREEN	no change
xx01	(1)	GREEN	select system mode
xx10	(2)	GREEN	turn off LED (user mode)
xx11	(3)	GREEN	turn on LED (user mode)
00xx	(0)	RED	no change
01xx	(4)	RED	select system mode
10xx	(8)	RED	turn off LED (user mode)
11xx	(C)	RED	turn on LED (user mode)

The RED and GREEN LEDs may be explicitly set or cleared by user code, or they may be set to “system” mode to be used by the operating system for special status functions. The default for both the RED and GREEN LEDs is system mode. Explicitly setting or clearing an LED places it into “user” mode, and has the effect of canceling system mode. When in user mode, an LED displays the last state specified and is not used by the system.

When the GREEN LED is in “system” mode, it is blinked while the processor is in the idle loop. When the RED LED is in “system” mode, it reflects the state of the ERRFLAG location in AFGIS OS Fixed-RAM, turning on when an error is indicated, and off when ERRFLAG has been cleared.

Set RED/GREEN LEDs (variable)

LEDV V

Syntax: 0073 word

Description: The function of the RED and GREEN LEDs is set according to the value of the parameter contained in variable V.

BINARY	(HEX)	LED	ACTION
xx00	(0)	GREEN	no change
xx01	(1)	GREEN	select system mode
xx10	(2)	GREEN	turn off LED (user mode)
xx11	(3)	GREEN	turn on LED (user mode)
00xx	(0)	RED	no change
01xx	(4)	RED	select system mode
10xx	(8)	RED	turn off LED (user mode)
11xx	(C)	RED	turn on LED (user mode)

The RED and GREEN LEDs may be explicitly set or cleared by user code, or they may be set to “system” mode to be used by the operating system for special status functions. The default for both the RED and GREEN LEDs is system mode. Explicitly setting or clearing an LED places it into “user” mode, and has the effect of canceling system mode. When in user mode, an LED displays the last state specified and is not used by the system.

When the GREEN LED is in “system” mode, it is blinked while the processor is in the idle loop. When the RED LED is in “system” mode, it reflects the state of the ERRFLAG location in AFGIS OS Fixed-RAM, turning on when an error is indicated, and off when ERRFLAG has been cleared.

Draw Line Point To Point

LINE type X*0* Y*0* X*1* Y*1*

Syntax: 0074 word word word word word

Description: Draws a line with the specified line type from the point (X0, Y0) to (X1, Y1). The current position is left unchanged

Related opcodes: LINEV, LINER, LINETO

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

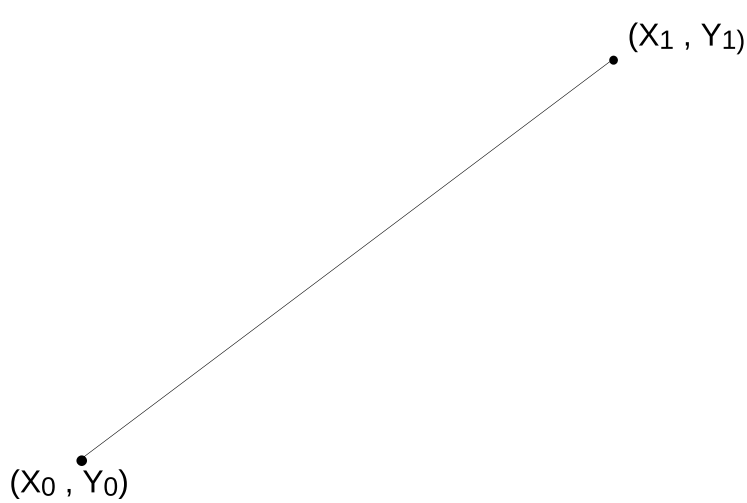
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



Draw Line Point To Point (variable)

LINEV V*i*

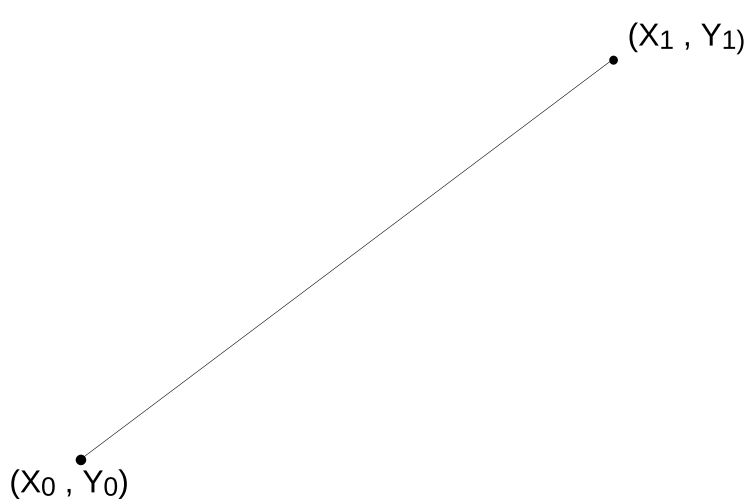
Syntax: 0075 word

Parameters:

Vi	= line type
Vi+1	= X0
Vi+2	= Y0
Vi+3	= X1
Vi+4	= Y1

Description: Draws a line with the specified line type from the point (X0, Y0) to (X1, Y1). The current position is left unchanged. Vi is the first of 5 consecutive variables containing the parameters,

Related opcodes: LINEV, LINER, LINETO



Select Line Pattern Continue Mode

LINECON mode

Syntax: 0076 word

Description: Sets the line pattern continue mode according to the following parameter values:
0=continue mode off 1=continue mode on 2=reset continue pattern

Related opcodes: LINEPATN

Discussion: When continue mode is off each subsequent patterned line will begin drawing with the initial point of its pattern. When continue mode is on, subsequent patterned lines will continue the line pattern from the last point drawn by the previous patterned line. Specifying a parameter of “2" will temporarily reset the line pattern without canceling continue mode.

Select Line Pattern Continue Mode (variable)

LINECONV V

Syntax: 0077 word

Parameter: Vi = function code

Description: Sets the line pattern continue mode according to the following parameter values: 0=continue mode off 1=continue mode on 2=reset continue pattern

Related opcodes: LINEPATN

Discussion: When continue mode is off each subsequent patterned line will begin drawing with the initial point of its pattern. When continue mode is on, subsequent patterned lines will continue the line pattern from the last point drawn by the previous patterned line. Specifying a parameter of “2" will temporarily reset the line pattern without canceling continue mode.

Select Binary Line Pattern

LINEPATN pattern

Syntax: 0078 long

Description: Sets the current line pattern to the 32-bit value specified. Bit 0 of the pattern corresponds to the first pixel to be drawn. Zero-bits in the pattern indicate pixels to be drawn with background color; one-bits are drawn with foreground color.

Patterned lines are selected by specifying line type “1" for any of the line drawing opcodes (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, RRECT, SECT, SEG)

Related opcodes: LINECON, DASHPATN

Select Binary Line Pattern (variable)

LINEPATNV V

Syntax: 0079 word

Description: Sets the current line pattern to the 32-bit value specified. Bit 0 of the pattern corresponds to the first pixel to be drawn zero-bits in the pattern indicate pixels to be drawn with background color; one-bits are drawn with foreground color.

Patterned lines are selected by specifying line type “1" for any of the line drawing opcodes (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, RRECT, SECT, SEG)

Related opcodes: LINECON, DASHPATN

Draw Line Point To Point, Relative

LINER type dX*0* dY*0* dX*1* dY*1*

Syntax: 007A word word word word word

Description: Draws a line with the specified line type from the point (X+dX0, Y+dY0) to (X+dX1, Y+dY1) where (X,Y) is the current position (i.e., the points are relative to the current position and are specified as offsets from same). The current position is left unchanged

Related opcodes: LINERV, LINE, LINETO, MOVETO

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

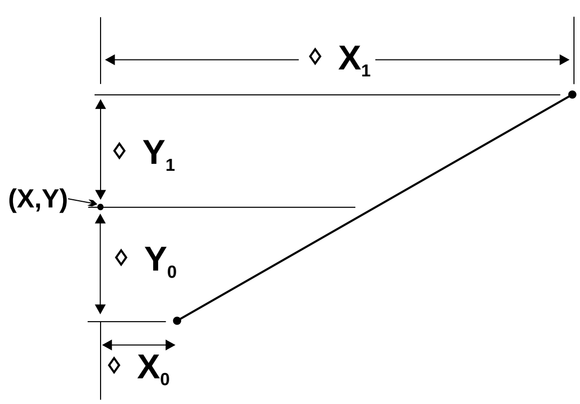
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



Draw Line Point To Point, Relative (variable)

LINERV V*i*

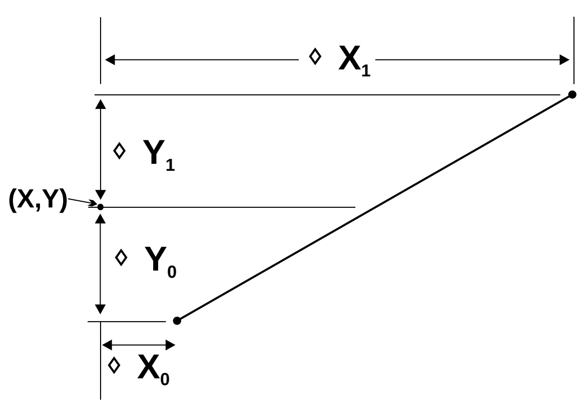
Syntax: 007B word

Parameters:

Vi	= line type
Vi+1	= dX0 (X0 offset)
Vi+2	= dY0 (Y0 offset)
Vi+3	= dX1 (X1 offset)
Vi+4	= dY1 (Y1 offset)

Description: Draws a line with the specified line type from the point (X+dX0, Y+dY0) to (X+dX1, Y+dY1) where (X,Y) is the current position (i.e., the points are relative to the current position and are specified as offsets from same). The current position is left unchanged. Vi is the first of 5 consecutive variables containing the parameters.

Related opcodes: LINERV, LINE, LINETO, MOVETO



Draw Line

LINETO type x y

Syntax: 007C word word word

Description: Draws a line from the current position to the point (x,y). The line is drawn with the specified line type, with the current color(s). Updates the current x,y position to the terminal point of the line.

Related opcodes: LINETO, LINETOR, MOVETO

Line Types:
w_type=0= solid line
w_type=1= dash line (32 bit binary pattern)
w_type=2= dash line (arbitrary - segment-length byte list)
w_type=3= fatline - solid
w_type=4= fatline - stipple pattern (binary)
w_type=5= fatline - tile pattern (pixel-mapped)
w_type=6= pen line - solid
w_type=7= pen line - stipple
w_type=8= pen line - tile

Draw Line (variable)

LINETOV V*i*

Syntax: 007D word

Parameters:

Vi	= line type
Vi+1	= X coordinate
Vi+2	= Y coordinate

Description: Draws a line from the current position to the point (x,y). The line is drawn with the specified line type, with the current color(s). Updates the current x,y position to the terminal point of the line. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: LINETO, LINETOR, MOVETO

Draw Line, Relative

LINETOR type X offset Y offset

Syntax: 007E word word word

Description: Draws a line from the current position to the point defined by (x + xoffset, y + yoffset). The line is drawn with the specified line type.

Updates the current x,y position to the terminal point of the line.

Related opcodes: LINETO, LINETOV, MOVETO

Line Types:
w_type=0= solid line
w_type=1= dash line (32 bit binary pattern)
w_type=2= dash line (arbitrary - segment-length byte list)
w_type=3= fatline - solid
w_type=4= fatline - stipple pattern (binary)
w_type=5= fatline - tile pattern (pixel-mapped)
w_type=6= pen line - solid
w_type=7= pen line - stipple
w_type=8= pen line - tile

Draw Line, Relative (variable)

LINETORV V*i*

Syntax: 007F word

Parameters:

Vi	= line type
Vi+1	= X offset
Vi+2	= Y offset

Description: Draws a line from the current position to the point defined by (x + xoffset, y + yoffset). The line is drawn with the specified line type. Updates the current x,y position to the terminal point of the line. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: LINETO, LINETOV, MOVETO

Set Current Marker

MARKER address

Syntax: 0108 long

Description: Sets the current marker. The specified address is assumed to point to a marker definition structure formatted as follows:

Marker Definition Structure:

OFFS	SIZE	FORMAT	FIELD
00	16	coded	marker type (see table below)
10	16	coded	special parameter field (depends on mark type, see below)
20	32	[X Y]	marker dimensions (half-sizes, see below)
40	32	pointer	address of symbol data (mark type = 8)
60	32	[X Y]	x,y offset from point
80	32	color	marker color (optional- see flag field)
A0	16	coded	flags (see below)

MARKER	MARK TYPE	TYPE PARAM	SIZE	SHAPE	OFFSET	COLOR	FLAGS
outlined ellipse	0	linetype	[Yrad, Xrad]	N/A	[dy,dx]	color	flags
filled ellipse	1	filltype	[Yrad, Xrad]	N/A	[dy,dx]	color	flags
outlined rectangle	2	linetype	[h/2, w/2]	N/A	[dy,dx]	color	flags
filled rectangle	3	filltype	[h/2, w/2]	N/A	[dy,dx]	color	flags
outlined diamond	4	linetype	[h/2, w/2]	N/A	[dy,dx]	color	flags
filled diamond	5	filltype	[h/2, w/2]	N/A	[dy,dx]	color	flags
“+” mark	6	linetype	[h/2, w/2]	N/A	[dy,dx]	color	flags
“X” mark	7	linetype	[h/2, w/2]	N/A	[dy,dx]	color	flags
symbol	8	rotation	N/A	address	[dy,dx]	color	flags
character	9	charcode	N/A	N/A	[dy,dx]	color	flags

Flag Field:

BIT	DESCRIPTION
0	color select
	0 = use current foreground color
	1 = use specified color

Note: for marker types 0 through 7, if the marker “size” is specified as [0,0], then a single pixel is drawn at each of the vertices.

Related opcodes: PMARK, PMARKR, PPIXEL

Set Current Marker (variable)

MARKERV V

Syntax: 0109 word

Description: Sets the current marker. The variable V contains an address which is assumed to point to a marker definition structure formatted as follows:

Marker Definition Structure:

OFFS	SIZE	FORMAT	FIELD
00	16	coded	marker type (see table below)
10	16	coded	special parameter field (depends on mark type, see below)
20	32	[X Y]	marker dimensions (half-sizes, see below)
40	32	pointer	address of symbol data (mark type = 8)
60	32	[X Y]	x,y offset from point
80	32	color	marker color (optional- see flag field)
A0	16	coded	flags (see below)

MARKER	MARK TYPE	TYPE PARAM	SIZE	SHAPE	OFFSET	COLOR	FLAGS
outlined ellipse	0	linetype	[Yrad, Xrad]	N/A	[dy,dx]	color	flags
filled ellipse	1	filltype	[Yrad, Xrad]	N/A	[dy,dx]	color	flags
outlined rectangle	2	linetype	[h/2, w/2]	N/A	[dy,dx]	color	flags
filled rectangle	3	filltype	[h/2, w/2]	N/A	[dy,dx]	color	flags
outlined diamond	4	linetype	[h/2, w/2]	N/A	[dy,dx]	color	flags
filled diamond	5	filltype	[h/2, w/2]	N/A	[dy,dx]	color	flags
“+” mark	6	linetype	[h/2, w/2]	N/A	[dy,dx]	color	flags
“X” mark	7	linetype	[h/2, w/2]	N/A	[dy,dx]	color	flags
symbol	8	rotation	N/A	address	[dy,dx]	color	flags
character	9	charcode	N/A	N/A	[dy,dx]	color	flags

Flag Field:

BIT	DESCRIPTION
0	color select
	0 = use current foreground color
	1 = use specified color

Note: for marker types 0 through 7, if the marker “size” is specified as [0,0], then a single pixel is drawn at each of the vertices.

Related opcodes: MARKER

Multiply Variables

MLTV V_s V_d

Syntax: 0080 word word

Description: The value of Vs is multiplied by the value of V_d. The result is stored in V_d. Integer multiplication is performed. $V_d = V_s \times V_d$.

Related opcodes: DIVV, ADVV, SBVV, JUMPA, JUMPR

Flags Affected: Z (other flags undefined)

Modulus variable with variable

MODV V*s* V*d*

Syntax: 0081 word word

Description: The destination variable is set to the result of the operation V*d* Modulus V*s.* The source variable Vs remains unchanged.

Related opcodes: DIVV, JUMPA, JUMPR

Flags Affected: ZV (other flags undefined)

Set Current x,y Location

MOVETO x y

Syntax: 0082 word word

Description: Moves the current screen position to the new position (x,y)

Related opcodes: MOVETOR, MOVETORV, MOVETOV

Set Current x,y Location (variable)

MOVETOV V*i*

Syntax: 0083 word

Parameters: Vi = X coordinate

Vi+1 = Y coordinate

Description: Moves the current screen position to the new position (x,y). Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: MOVETOR, MOVETORV, MOVETOV

Set Current x,y Location, Relative

MOVETOR x_offset y_offset

Syntax: 0084 word word

Description: Moves the current screen position from (x,y) to the new relative position (x+*xoffset*, y+*yoffset*).

Related opcodes: MOVETO, MOVETORV, MOVETOV

Set Current x,y Location, Relative (variable)

MOVETORV V*i*

Syntax: 0085 word

Parameters: Vi = X offset

Vi+1 = Y offset

Description: Moves the current screen position from (x,y) to the new relative position (x+*xoffset*, y+*yoffset*). Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: MOVETO, MOVETOR, MOVETOV

Enable/Disable mouse cursor display

MSCSRON flag

Syntax: 0088 word

Description: Turns mouse cursor display on/off according to the specified parameter:

0 = mouse cursor off 1 = mouse cursor on

Discussion: The mouse cursor is typically positioned with the MSCSRXY opcode, and then displayed by executing “MSCSRON 1". The MSCSRON opcode saves the screen contents under the cursor. This screen data is restored when the mouse cursor is turned off with ”MSCSRON 0".

Enable/Disable mouse cursor display (variable)

MSCSRONV V

Syntax: 0089 word

Description: Turns mouse cursor display on/off according to the specified parameter:

0 = mouse cursor off 1 = mouse cursor on

Parameters: V = mouse cursor on flag

Discussion: The mouse cursor is typically positioned with the MSCSRXY opcode, and then displayed by executing “MSCSRON 1". The MSCSRON opcode saves the screen contents under the cursor. This screen data is restored when the mouse cursor is turned off with ”MSCSRON 0".

Configure Mouse Cursor for Channel and Page

MSCSRPAGE channel page

Syntax: 012C word word

Description: Initializes the mouse cursor for the specified graphics channel and page.

Related Opcodes: MSCURSOR

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751 and RG-752. It initializes the mouse cursor for the specified graphics channel and page, and thus allows the mouse cursor to be placed on either an underlay or overlay channel. Even for those boards having only a single graphics channel, this opcode may be used to move the mouse cursor to any of the available graphics pages. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

Configure Mouse Cursor for Channel and Page, variable

MSCSRPAGEV Vi

Syntax: 012D word

Description: Initializes the mouse cursor for the specified graphics channel and page.

Parameters: Vi = channel ID
Vi+1 = page #

Related Opcodes: MSCSRPAGE

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751 and RG-752. It initializes the mouse cursor for the specified graphics channel and page, and thus allows the mouse cursor to be placed on either an underlay or overlay channel. Even for those boards having only a single graphics channel, this opcode may be used to move the mouse cursor to any of the available graphics pages. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

Mouse Cursor Location

MSCSRXY X Y

Syntax: 008A word word

Description: Specifies the (x,y) location of the mouse cursor on the screen.

Related opcodes: MSCSRXYV, MSCSRON

Discussion: The MSCSRXY opcode changes the current x,y mouse address, and specifies the starting point for the mouse cursor on the screen. If the mouse cursor is already being displayed on the screen, it is repositioned to the x,y location specified.

Mouse Cursor Location (variable)

MSCSRXYV V*i*

Syntax: 008B word

Parameters: Vi = new mouse cursor X
Vi+1 = new mouse cursor Y

Description: Specifies the (x,y) location of the mouse cursor on the screen. The x coordinate is stored in variable Vi, and the y coordinate is stored in variable Vi+1. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: MSCSRXY, MSCSRON

Discussion: The MSCSRXY opcode changes the current x,y mouse address, and specifies the starting point for the mouse cursor on the screen. If the mouse cursor is already being displayed on the screen, it is repositioned to the x,y location specified.

Select mouse cursor

MSCURSOR cursor color1 color2 save_addr save_pitch

Syntax: 016C long long long long long

Description: Selects the current mouse cursor. If the cursor parameter value is <128, it is assumed to be the index of one of the default cursors, and the corresponding cursor is made active. Otherwise, the parameter is assumed to be the address of a cursor definition structure formatted as follows:

OFFS	SIZE	FORMAT	FIELD	APPLICABILITY					
0000	16	coded	cursor type (see below)	0	1	2	3	4	5
0010	16	coded	boolean operation	X	X	X	X	X	X
0020	32	[XY]	cursor size	X	X	X	X	X	X
0040	32	[XY]	cursor offset	X	X	X	X	X	X
0060	32	address	address of user-specified save buffer	X	X	X	X	X	X
0080	32	linear	pitch of user-specified save buffer	X	X	X	X	X	X
00A0	32	address	shape #1 address			X	X	X	X
00C0	32	linear	shape #1 pitch					X	X
00E0	32	address	shape #2 address				X		X
0100	32	linear	shape #2 pitch						X

Cursor Types

0 = filled rectangle

1 = outlined rectangle

2 = single color symbol

3 = two color symbol

4 = single color bitmap

5 = two color bitmap

For two-color pixblt cursors (types 3 and 5) shape #1 is drawn first with color #1 (background), then shape #2 is drawn with color #2 (foreground). For single-color pixblt cursors (types 2 and 4) only shape1 and color1 are used.

Pixel Processing: Any of the pixel-processing operations listed under the BOOL opcode may be used.

Cursor Size: Specifies the X and Y dimensions of the cursor rectangle. For cursor types 2 and 3, this must match the width and height parameters in the symbol header.

Cursor Offset: Specifies the signed X and Y offsets (in pixels) from the upper left corner of the cursor rectangle that identify the cursor “hot spot.” The cursor will be drawn so that the “hot spot” coincides exactly with the pixel specified as the current cursor X,Y location. E.g., for a “cross-hair” style cursor, the cursor offsets would typically be (width/2,height/2).

MSCURSOR (continued)

User-specified Save Buffer (optional): The firmware maintains a default internal save buffer for use by the on-board default cursors. The size of this default buffer accomodates the largest of default cursors (32x32). For a user defined cursor larger than 32x32 (or, a total "area" of more than 1024 pixels) a save buffer address MUST be specified. A save buffer address specified as an opcode parameter supercedes the parameter contained in the cursor definition structure. If both are NULL (0), then the default save buffer address will be used.

Pitch of User-specified Save Buffer(optional): If a save buffer address is specified, then the save buffer pitch must also be specified. The save buffer pitch (if used) is specified in bits, and must be a multiple of 16. A pitch value specified as an opcode parameter supercedes the parameter contained in the cursor definition structure. If both are NULL (0), then the default pitch value will be used.

Note: If either the save buffer address or save buffer pitch is NULL, the cursor size is truncated to accomodate the default save buffer (32x32), and the default save buffer pitch is used.

Shape Address (cursor types 2, 3, 4, 5): For cursor types 2 and 3 (symbol), the address is that of a symbol structure of the type used by the SSYM opcode. For cursor types 4 and 5, the address is that of a pixel array (bitmap) of the same dimensions as specified by the cursor size, of the specified pitch (see below).

Shape Pitch (cursor types 4, 5): Pitch must be specified for any of the bitmap cursor types, and must be a power of 2 (16, 32,...).

Related opcodes: MSCSRON, MSCSRPAGE, MSCSRXY, TXCURSOR, USCURSOR

Discussion: The mouse cursor, text cursor and user cursor (selected with the MSCURSOR, TXCURSOR and USCURSOR opcodes, respectively) all use the same cursor definition structure and may likewise use any of the default cursors.

The mouse cursor and text cursor both have default shapes at power-up, but the user cursor does not. A user cursor MUST be defined (or selected from the default cursors) before it can be used.

The mouse cursor and text cursor are both global resources (i.e., there is only one of each). The user cursor may be considered a local resource in that there may be one user cursor per environment (graphics context), and therefore there may be as many user cursors in use as there are environments.

The mouse cursor and text cursor each have a defaut save buffer that is used for the default cursors. The default save buffers will accomodate a cursor size of up to 32x32 pixels (or a total "area" of 1024 pixels) and may be used for a user-defined cursor by specifying a NULL (0) save buffer address. There is no default user cursor save buffer—a save buffer address MUST be specified when selecting or defining a user cursor.

The mouse cursor and text cursor may both be configured for automatic save/restore handling by the graphics primitives (both default to this mode at power-up). When in "auto-handling" mode all graphics primitives (circles, lines, text, etc.) will automatically remove and restore the cursors as necessary—thus the user is not required to manage the state of either the mouse or text cursor. If preferred, "auto-handling" mode may be disabled by using the MSREG opcode to disable mouse cursor auto-handling, or by modifying the txcsr_mode field in Global RAM (see appendix A) to disable text cursor auto-handling. The user cursor is NOT handled by the graphics primitives—the user is responsible for managing the state of the user cursor when using any of the graphics primitives.

MSCURSOR (continued)

The mouse cursor may be configured to track the current mouse position when the mouse is enabled (MSMODE≠0). When “mouse-tracking” is enabled, the mouse cursor position may also be changed with the MSCSRXY opcode, but will thereafter track susequent mouse movement inputs. “Mouse-tracking” is enabled as the default mode at power-up, but may be disabled with the MSREG opcode. The mouse cursor position may also be changed “manually” with MSCSRXY when mouse tracking mode is disabled, mouse input is disabled (MSMODE=0), or the mouse is disconnected (a mouse does not need to be connected to the serial port in order to use the mouse cursor). The text cursor and user cursor positions may only be changed with the TXCSRXY and USCSRXY opcodes, respectively.

The mouse cursor has the highest priority in that it will always appear to be “in front of” the text cursor and any user cursors. The text cursor has the next highest priority and will appear to be in front of any user cursors. The user cursor has a lower priority than the mouse or text cursors, but will appear to be in front of any background graphics. The user must manage the relative priorities of overlapping user cursors if more than one is active at a time.

The text cursor may be configured to blink independently of the “blinking palette” function (BLINK opcode) by specifying a non-zero value for the blink-rate parameter of the TXCURSOR opcode. The text cursor will then be removed and restored at regular intervals according to the blink rate specified, and information “behind” the text cursor will become unobscured during the intervals when the text cursor is removed. The blink function of the text cursor will thus operate even on boards that do not make use of a RAMDAC device (such as the RG-752). Any of the cursors may be made to “blink” by specifying colors that have been configured to be “blinking” colors with the BLINK opcode, although information behind the cursor will continue to be obscured.

MOUSE CURSOR	TEXT CURSOR	USER CURSOR
common definition structure	common definition structure	common definition structure
default shape at power-up	default shape at power-up	no default shape (user must define)
global resource (1 only)	global resource (1 only)	1 per environment
default save buffer (32x32)	default save buffer (32x32)	no default save buffer (user must define)
save/restore handled by graphics primitives (may be disabled)	save/restore handled by graphics primitives (may be disabled)	user must manage cursor state
tracks mouse movement (may be disabled and moved with MSCSRXY)	move with TXCSRXY	move with USCSRXY
highest priority	next highest priority	lowest priority
	may be configured to blink	

Select mouse cursor, variable

MSCURSORV V*i*

Syntax: 016D word

Parameters:

Vi	= address of cursor definition structure (or index of default)
Vi+1	= shape #1 color
Vi+2	= shape #2 color
Vi+3	= save buffer address
Vi+4	= save buffer pitch

Description: Selects the current mouse cursor. If the cursor parameter value is <128, it is assumed to be the index of one of the default cursors, and the corresponding cursor is made active. Otherwise, the parameter is assumed to be the address of a cursor definition structure (see MSCURSOR).

Related opcodes: MSCURSOR

Set Mouse Interrrupt Service Mode

MSMODE mode

Syntax: 008C word

Description: Enables the serial port mouse for use and sets the current mouse interrupt mode according to the following parameter values: 0 = mouse off 1 = mouse on, polled mode 2 = mouse on interrupt mode. The mouse cursor will not be displayed until a MSCSRON opcode is executed.

Related opcodes: MSCSRON, MSCSRXY, MSCSRXYV, MSQFL, MSTEST

Discussion: The serial port may be used with a serial mouse or it may be used with an RS-232 peripheral. When used for a serial mouse, the AFGIS firmware defaults to the Microsoft mouse data format.

The serial port generates an interrupt when mouse data is available. If the host is unable to read the data, it is stored in the mouse queue. Data is retrieved from the mouse queue at 60Hz rate and made available to the host.

Mouse report data is stored in the following RAM.

RAM LOCATION	DESCRIPTION
MSEDATA_SW	Switch Closure: D2=Left, D1=Middle, D0=Right, 1=Closed.
MSEDATA_X	X screen coordinate.
MSEDATA_Y	Y screen coordinate.
MSEDATA_TIME	Mouse data time stamp

Additional data and error information is stored in RAM. See appendix A for more information.

Use “MSMODE1" at the beginning of a mouse polling session. This opcode turns on the mouse interface and allows the bus CPU to poll fixed-RAM location MSEFLAG to determine if mouse data is available .

Set Mouse Interrrupt Service Mode (variable)

MSMODEV V

Syntax: 008D word

Parameters: V = mouse mode

Description: Enables the serial port mouse for use and sets the current mouse interrupt mode according to the following parameter values: 0 = mouse off 1 = mouse on, polled mode 2 = mouse on interrupt mode. The mouse cursor will not be displayed until a MSCSRON opcode is executed.

Related opcodes: MSCSRON, MSCSRXY, MSCSRXYV, MSQFL, MSTEST

Discussion: The serial port may be used with a serial mouse or it may be used with an RS-232 peripheral. When used for a serial mouse, the AFGIS firmware defaults to the Microsoft mouse data format.

The serial port generates an interrupt when mouse data is available. If the host is unable to read the data, it is stored in the mouse queue. Data is retrieved from the mouse queue at 60Hz rate and made available to the host.

Mouse report data is stored in the following RAM.

RAM LOCATION	DESCRIPTION
MSEDATA_SW	Switch Closure: D2=Left, D1=Middle, D0=Right, 1=Closed.
MSEDATA_X	X screen coordinate.
MSEDATA_Y	Y screen coordinate.
MSEDATA_TIME	Mouse data time stamp

Additional data and error information is stored in RAM. See appendix A for more information.

Use “MSMODE1" at the beginning of a mouse polling session. This opcode turns on the mouse interface and allows the bus CPU to poll fixed-RAM location MSEFLAG to determine if mouse data is .

Flush Mouse/Serial Port Queue

MSQFL

Syntax: 008E

Description: Clears the mouse queue.

Related opcodes: MSCSRON, MSCSRXY

Set Mouse Register

MSREG register value

Syntax: 0156 word long

Description: Sets the specified mouse register to a new value. The mouse registers and their functions are described below:

Register	Function
0	Mouse tracking mode register (MSETRACKMODE)
1	Mouse reporting mode register (MSEREPORTMODE)

msetrackmode - mouse position tracking mode

BIT	FIELD
0	local tracking mode 0: mouse cursor position controlled externally 1: mouse cursor position controlled internally by graphics board
1	mouse cursor “wrap” mode 0: mouse cursor “sticks” at boundary 1: mouse cursor “wraps” to other side of boundary
2	mouse cursor “confine” mode 0: mouse cursor confined to screen boundaries 1: mouse cursor confined to mouse window boundaries
3	swap mouse X,Y coordinates 0: normal 1: swaps X,Y mouse cursor movement
4	mouse cursor save/restore mode 0: mouse cursor save/restore handled by host 1: mouse cursor save/restore handled by graphics board

msereportmode - mouse reporting mode

VALUE	MOUSE REPORTING MODE
0	report on switch closure
1	report on switch closure or release
2	report on switch release
3	report all movement while any switch closed
4	report all movement

Related opcodes: MSSCALE, MSWIN

Set Mouse Register, variable

MSREGV V*i*

Syntax: 0157 word

Parameters: V*i* = mouse register #

V*i*+1 = mouse register value

Description: Sets the specified mouse register to a new value. See MSREG for a description of the mouse registers and their functions.

Related opcodes: MSREG

Set Mouse Scale Factors

MSSCALE Xscale Yscale

Syntax: 0158 word word

Description: Sets the mouse movement scale factors. The scale factors are 16-bit signed, fixed-point numbers with an 8-bit integer and 8-bit fraction portions. The default values are +1.00 (coded 0100h). Negative scale factors will cause the mouse to move in the opposite direction along the corresponding axis. Mouse X and Y movement may be swapped by setting the "swap X,Y" bit in the "MSETRACKMODE" register (see MSREG).

Related opcodes: MSREG, MSWIN

Set Mouse Scale Factors, variable

MSSCALEV V*i*

Syntax: 0159 word

Parameters: V_i = mouse X scale factor

V_i+1 = mouse Y scale factor

Description: Sets the mouse movement scale factors (see MSSCALE).

Related opcodes: MSSCALE

Mouse Test

MSTEST

Syntax: 008F

Description: Runs the built-in mouse test.

Set Mouse Window

MSWIN X0 Y0 X1 Y1

Syntax: 015A word word word word

Description: Sets the mouse window boundaries (upper-left, lower-right). The mouse window boundaries have no effect unless the “confine” bit is set in the “MSETRACKMODE” register (see MSREG).

Related opcodes: MSREG, MSSCALE

Set Mouse Window, variable

MSWINV V*i*

Syntax: 015B word

Parameters: V*i* = mouse window Xmin

V*i*+1 = mouse window Ymin

V*i*+2 = mouse window Xmax

V*i*+3 = mouse window Ymax

Description: Sets the mouse window boundaries (upper-left, lower-right). The mouse window boundaries have no effect unless the “confine” bit is set in the “MSETRACKMODE” register (see MSREG).

Related opcodes: MSWIN

No Operation

NOOP

Syntax: 0000

Description: This opcode is processed, but has no effect on the display. It takes a short period of time and is used mainly as a place holder for testing code.

Related opcodes: None

Discussion: The NOOP instruction does nothing but is useful to hold a place in the instruction RAM to be filled with a more useful instruction later. Or, when debugging code, an instruction may be removed from a list by replacing it with zeros allowing the rest of the list to be processed.

OR Immediate with Variable

ORIV long V_d

Syntax: 0090 long word

Description: The 32-bit immediate value is ORed with the specified variable.

Related opcodes: ORVV, ANDIV, JUMPA, JUMPR

Flags Affected: Z (other flags undefined)

OR Variables

ORVV V_s V_d

Syntax: 0091 word word

Description: The 32 bit variable V_s is OR'ed with the 32 bit variable V_d.

Related opcodes: ORIV, ANDVV, JUMPA, JUMPR

Flags Affected: Z (other flags undefined)

Discussion: Any or all the bits in V_d may be set to one, depending on the value of V_s.

Pan Display Horizontally (absolute)

PANX X

Syntax: 0116 word

Description: Specifies a new horizontal pan position.

Related opcodes: PANXV, PANY, PANXY, PANXYR

Discussion: The current pan x,y position defines a point, relative to the base address of the current display page, that then becomes the new upper-left corner of the display — i.e., the display is “panned” horizontally and vertically such that the pan-point coincides with the first pixel output to the display device on each frame. This effect could also be accomplished by modifying the display page base address with the DPGA opcode, but would require address calculations that are otherwise performed internally by the pan functions, making them considerably more convenient. However, the pan functions also implement single-pixel horizontal panning on those boards that support it (such as the RG-751). On boards that do not support single-pixel horizontal panning, the horizontal pan position will be rounded off to a multiple of 2 or 4 pixels depending on the hardware configuration and the display may appear to “jump” by this amount during horizontal panning. All boards support single-line vertical pan.

Pan Display Horizontally (absolute), variable

PANXV Vi

Syntax: 0117 word

Description: Specifies a new horizontal pan position.

Parameters: Vi = X-position

Related opcodes: PANX, PANY, PANXY, PANXYR

Discussion: The current pan x,y position defines a point, relative to the base address of the current display page, that then becomes the new upper-left corner of the display — i.e., the display is “panned” horizontally and vertically such that the pan-point coincides with the first pixel output to the display device on each frame. This effect could also be accomplished by modifying the display pate base address with the DPGA opcode, but would require address calculations that are otherwise performed internally by the pan functions, making them considerably more convenient. However, the pan functions also implement single-pixel horizontal panning on those boards that support it (such as the RG-751). On boards that do not support single-pixel horizontal panning, the horizontal pan position will be rounded off to a multiple of 2 or 4 pixels depending on the hardware configuration and the display may appear to “jump” by this amount during horizontal panning. All boards support single-line vertical pan.

Pan Display Vertically (absolute)

PANY Y

Syntax: 0118 word

Description: Specifies a new vertical pan position.

Related opcodes: PANYV, PANX, PANXY, PANXYR

Discussion: The current pan x,y position defines a point, relative to the base address of the current display page, that then becomes the new upper-left corner of the display — i.e., the display is "panned" horizontally and vertically such that the pan-point coincides with the first pixel output to the display device on each frame. This effect could also be accomplished by modifying the display pate base address with the DPGA opcode, but would require address calculations that are otherwise performed internally by the pan functions, making them considerably more convenient. However, the pan functions also implement single-pixel horizontal panning on those boards that support it (such as the RG-751). On boards that do not support single-pixel horizontal panning, the horizontal pan position will be rounded off to a multiple of 2 or 4 pixels depending on the hardware configuration and the display may appear to "jump" by this amount during horizontal panning. All boards support single-line vertical pan.

Pan Display Vertically (absolute), variable

PANYV Vi

Syntax: 0119 word

Description: Specifies a new vertical pan position.

Parameters: Vi = Y-position

Related opcodes: PANY, PANX, PANXY, PANXYR

Discussion: The current pan x,y position defines a point, relative to the base address of the current display page, that then becomes the new upper-left corner of the display — i.e., the display is “panned” horizontally and vertically such that the pan-point coincides with the first pixel output to the display device on each frame. This effect could also be accomplished by modifying the display pate base address with the DPGA opcode, but would require address calculations that are otherwise performed internally by the pan functions, making them considerably more convenient. However, the pan functions also implement single-pixel horizontal panning on those boards that support it (such as the RG-751). On boards that do not support single-pixel horizontal panning, the horizontal pan position will be rounded off to a multiple of 2 or 4 pixels depending on the hardware configuration and the display may appear to “jump” by this amount during horizontal panning. All boards support single-line vertical pan.

Pan Display (absolute)

PANXY X Y

Syntax: 011A word

Description: Specifies a new pan position.

Related opcodes: PANXYV, PANXYR, PANX, PANY

Discussion: The current pan x,y position defines a point, relative to the base address of the current display page, that then becomes the new upper-left corner of the display — i.e., the display is “panned” horizontally and vertically such that the pan-point coincides with the first pixel output to the display device on each frame. This effect could also be accomplished by modifying the display pate base address with the DPGA opcode, but would require address calculations that are otherwise performed internally by the pan functions, making them considerably more convenient. However, the pan functions also implement single-pixel horizontal panning on those boards that support it (such as the RG-751). On boards that do not support single-pixel horizontal panning, the horizontal pan position will be rounded off to a multiple of 2 or 4 pixels depending on the hardware configuration and the display may appear to “jump” by this amount during horizontal panning. All boards support single-line vertical pan.

Pan Display (absolute), variable

PANXYV Vi

Syntax: 011B word

Description: Specifies a new pan position.

Parameters: Vi = X-position
Vi + 1 = Y-position

Related opcodes: PANXY, PANXYR, PANX, PANY

Discussion: The current pan x,y position defines a point, relative to the base address of the current display page, that then becomes the new upper-left corner of the display — i.e., the display is “panned” horizontally and vertically such that the pan-point coincides with the first pixel output to the display device on each frame. This effect could also be accomplished by modifying the display pate base address with the DPGA opcode, but would require address calculations that are otherwise performed internally by the pan functions, making them considerably more convenient. However, the pan functions also implement single-pixel horizontal panning on those boards that support it (such as the RG-751). On boards that do not support single-pixel horizontal panning, the horizontal pan position will be rounded off to a multiple of 2 or 4 pixels depending on the hardware configuration and the display may appear to “jump” by this amount during horizontal panning. All boards support single-line vertical pan.

Pan Display (relative)

PANXYR deltaX deltaY

Syntax: 011C word word

Description: Specifies a new pan position, relative to the previous one.

Related opcodes: PANXYRV, PANXY, PANX, PANY

Discussion: The current pan x,y position defines a point, relative to the base address of the current display page, that then becomes the new upper-left corner of the display — i.e., the display is "panned" horizontally and vertically such that the pan-point coincides with the first pixel output to the display device on each frame. This effect could also be accomplished by modifying the display pate base address with the DPGA opcode, but would require address calculations that are otherwise performed internally by the pan functions, making them considerably more convenient. However, the pan functions also implement single-pixel horizontal panning on those boards that support it (such as the RG-751). On boards that do not support single-pixel horizontal panning, the horizontal pan position will be rounded off to a multiple of 2 or 4 pixels depending on the hardware configuration and the display may appear to "jump" by this amount during horizontal panning. All boards support single-line vertical pan.

Pan Display (relative), variable

PANXYRV V*i*

Syntax: 011D word

Description: Specifies a new pan position, relative to the previous one.

Parameters: Vi+0 = delta X

Vi+1 = delta Y

Related opcodes: PANXYR, PANXY, PANX, PANY

Discussion: The current pan x,y position defines a point, relative to the base address of the current display page, that then becomes the new upper-left corner of the display — i.e., the display is "panned" horizontally and vertically such that the pan-point coincides with the first pixel output to the display device on each frame. This effect could also be accomplished by modifying the display pate base address with the DPGA opcode, but would require address calculations that are otherwise performed internally by the pan functions, making them considerably more convenient. However, the pan functions also implement single-pixel horizontal panning on those boards that support it (such as the RG-751). On boards that do not support single-pixel horizontal panning, the horizontal pan position will be rounded off to a multiple of 2 or 4 pixels depending on the hardware configuration and the display may appear to "jump" by this amount during horizontal panning. All boards support single-line vertical pan.

Set pattern-fill reference mode

PATRNMODE mode

Syntax: 0094 word

Description: Sets the pattern-fill screen reference mode according to the following parameter values:

MODE	PATTERN-FILL REFERENCE
0	screen point
1	cardinal point of figure
2	upper-left corner of figure bounding rectangle
3	lower-left corner of figure bounding rectangle

Related opcodes: PATRNREF

Discussion: **Mode 0:** Pattern-fills are referenced to a single fixed screen point and thus overlapping figures will show no break in continuity of the pattern. The screen point used as the pattern reference point is specified with PATRNREF.

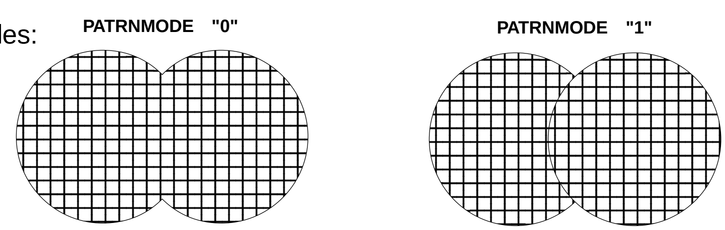
Mode 1: Pattern fills are referenced to the cardinal point of each figure, and overlapping figures may reveal a discontinuity in the pattern along the boundary of a figure overlapping a previous figure. The cardinal point of a figure depends on the type of figure, but generally results in similarly located figures being referenced to the same point. For example, the cardinal point of a polygon is the first point specified in the vertex list, while the cardinal point of a conic figure (circle, ellipse, etc.) is its center. Thus, two coincident circles of different radii will both use the same reference point, whereas they would have different reference points under modes 2 and 3 (see below).

Mode 2: Pattern-fills are referenced to the upper-left corner of a “bounding rectangle” that completely encloses the figure [MINY, MINX].

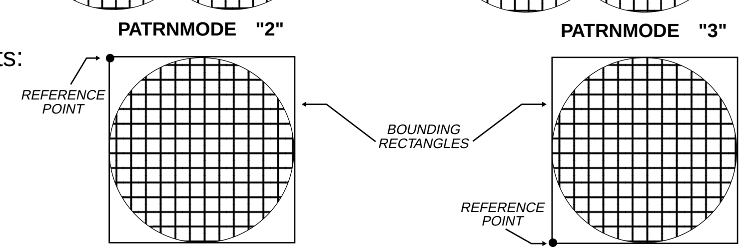
Mode 3: Pattern-fills are referenced to the lower-left corner of the bounding rectangle of the figure [MAXY, MINX].

For mode 0, PATRNREF is used to specify the single fixed screen point that is used as the pattern-fill reference point. For modes 1, 2 and 3, the coordinates specified with PATRNREF are used as signed offsets from the figure reference point (described above) to further refine the actual pattern-fill reference point used.

Example: Overlapping circles:



Example: Reference points:



Set pattern-fill reference mode

PATRNMODEV V

Syntax: 0095 word

Parameters: V = mode

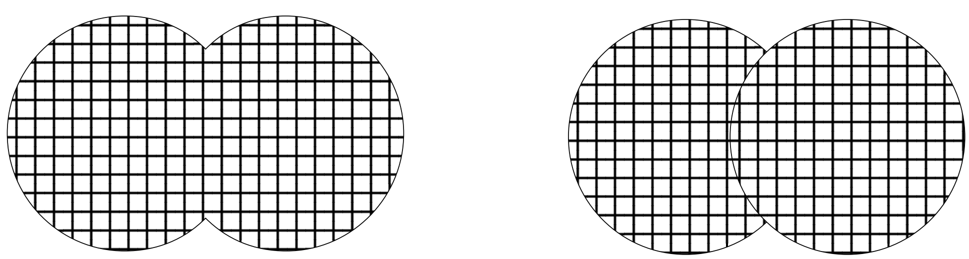
Description: Sets the pattern-fill screen reference mode according to the following parameter values: (See PATRNMODE for more details.)

MODE	PATTERN-FILL REFERENCE
0	screen point
1	cardinal point of figure
2	upper-left corner of figure bounding rectangle
3	lower-left corner of figure bounding rectangle

Related opcodes: PATRNMODE

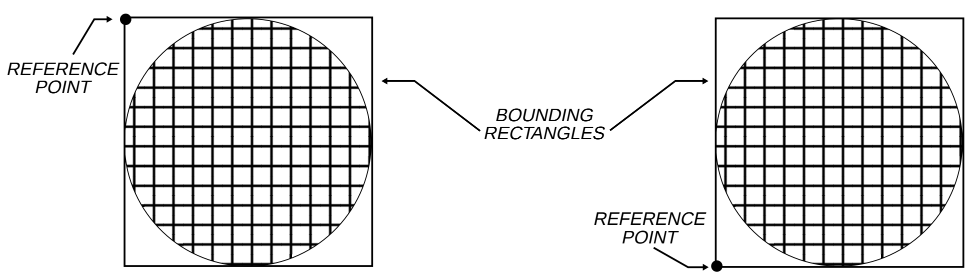
Example: Overlapping circles:

PATRNMODE "0" **PATRNMODE "1"**



Example: Reference points:

PATRNMODE "2" **PATRNMODE "3"**



Set pattern-fill reference point offset

PATRNREF x y

Syntax: 0096 word word

Description: Sets the pattern-fill reference point offset. The interpretation of the reference point offset depends on the pattern-fill reference mode (see PATRNMODE).

PATRNMODE	PATRNREF parameter usage
0	screen point relative to logical origin
1,2,3	signed offsets from figure reference point

Related opcodes: PATRNMODE, XYORG

Set pattern-fill reference point offset,variable

PATRNREFV V*i*

Syntax: 0097 word

Parameters: Vi = X offset

Vi+1 = Y offset

Description: Sets the pattern-fill reference point offset. The interpretation of the reference point offset depends on the pattern-fill reference mode (see PATRNMODE).

PATRNMODE	PATRNREF parameter usage
0	screen point relative to logical origin
1,2,3	signed offsets from figure reference point

Related opcodes: PATRNREF

Define Pen Style

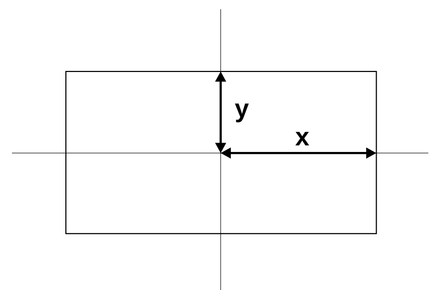
PENDEF mode x y

Syntax: 0098 word word word

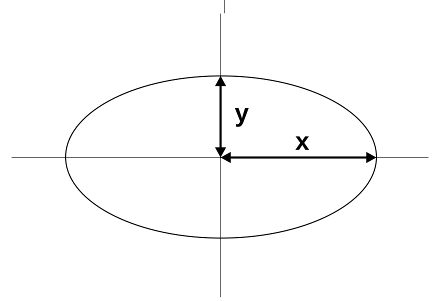
Description: Defines the style of the pen to be used for subsequent pen line draws. Two pen modes are available: 0 = rectangle, 1 = ellipse.

Pen-lines are selected by specifying line type 6,7 or 8 for any of the line-drawing opcodes (ARC, ARCTIC, CIR, ELP, LINE, LINER, LINETO, LINETOR, PLINE, PLINER, RECT, RRECT, SECT, SEG)

If the rectangle mode is selected (mode = 0), the x parameter represents the 1/2 of the width of the rectangle (expressed in pixels), and the y parameter represents 1/2 of the height of the rectangle (expressed in pixels) to be used for the pen point.



If the ellipse mode is selected (mode = 1), the x parameter represents 1/2 of the length of the horizontal axis of the ellipse (expressed in pixels), and the y parameter represents 1/2 of the length of the vertical axis of the ellipse (expressed in pixels) to be used for the pen point.



Define Pen Style (variable)

PENDEFV Vi

Syntax: 0099 word

Parameters: Vi = type code

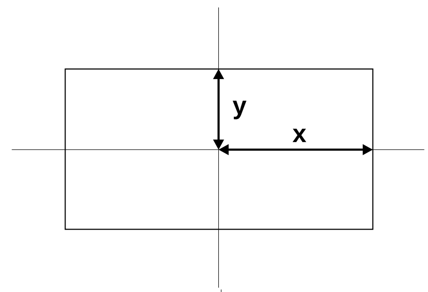
Vi+1 = X half-size

Vi+2 = Y half-size

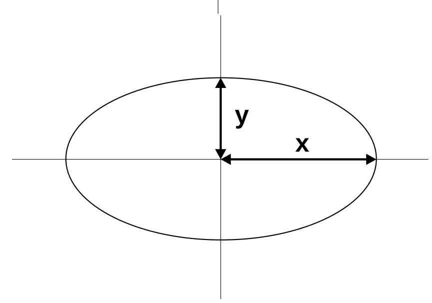
Description: Defines the style of the pen to be used for subsequent pen line draws. Two pen modes are available: 0 = rectangle, 1 = ellipse. Vi is the first of 3 consecutive variables containing the parameters.

Pen-lines are selected by specifying line type 6,7 or 8 for any of the line-drawing opcodes (ARC, CIR, ELP, LINE, LINETO, RECT, RRECT, SECT, SEG)

If the rectangle mode is selected (Vmode = 0), the value stored in Vxx represents 1/2 of the width of the rectangle (expressed in pixels), and the value stored in Vy represents 1/2 of the height of the rectangle (expressed in pixels) to be used for the pen point.



If the ellipse mode is selected (Vmode = 1), the value stored in Vx represents 1/2 of the length of the horizontal axis of the ellipse (expressed in pixels), and the value stored in Vy represents 1/2 of the length of the vertical axis of the ellipse (expressed in pixels) to be used for the pen point.



Polygon Fill (Indirect)

PFILL type count address

Hex Syntax: 009A word word long

Description: Draws a solid polygon from the coordinate list located at the address specified. Fills the polygon according to the specified fill-type. This is a general polygon fill and is capable of filling non-convex polygons. If a polygon is known to be convex, better performance would be attained by using the CPFILL opcode.

Related opcodes: PFILLV, CPFILL, PFILLR, PFILLO

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern

Polygon Fill (variable indirect)

PFILLV V*i*

Hex Syntax: 009B word

Parameters:

Vi	= fill type
Vi+1	= vertex count
Vi+2	= address of vertex list

Description: Draws a solid polygon from the coordinate list located at the address specified. Fills the polygon according to the specified fill-type. This is a general polygon fill and is capable of filling non-convex polygons. If a polygon is known to be convex, better performance would be attained by using the CPFILL opcode. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PFILL

General polygon fill (offset)

PFILLO type count address

Hex Syntax: 0142 word word long

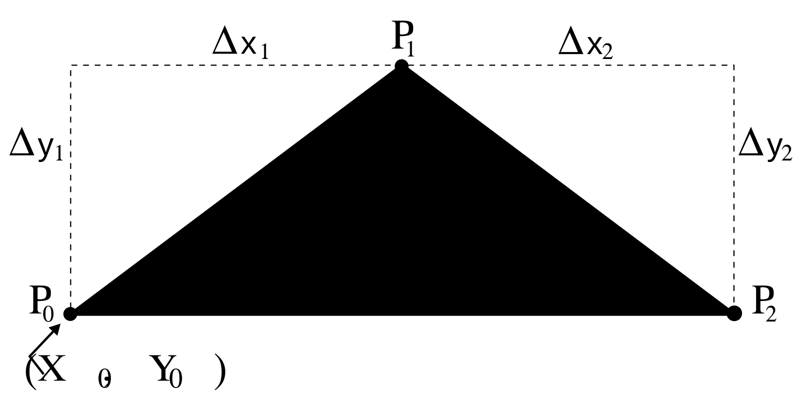
Description: Draws a solid polygon from the coordinate list located at the address specified. Fills the polygon according to the specified fill-type. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated. This is a general polygon fill and is capable of filling non-convex polygons. If a polygon is known to be convex, better performance would be attained by using the CPFILLO opcode.

Related opcodes: PFILL, PFILLR, CPFILLO

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



General polygon fill (offset), variable

PFILLOV V*i*

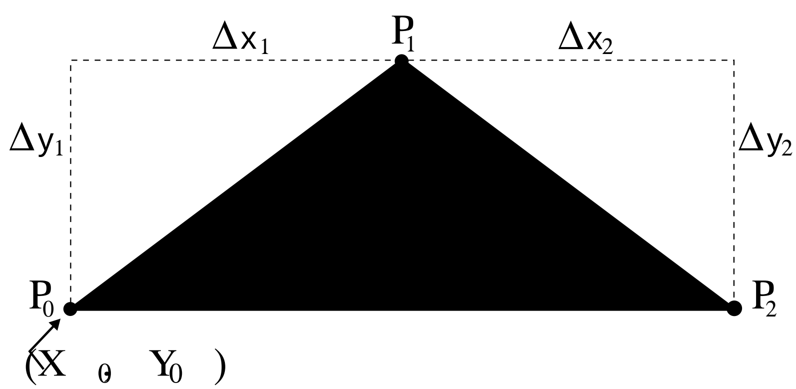
Hex Syntax: 0143 word

Parameters:

Vi		= fill type
Vi+1		= vertex count
Vi+2		= address of vertex list

Description: Draws a solid polygon from the coordinate list located at the address specified. Fills the polygon according to the specified fill-type. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated. This is a general polygon fill and is capable of filling non-convex polygons. If a polygon is known to be convex, better performance would be attained by using the CPFILLO opcode. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PFILLO



General polygon fill (relative)

PFILLR type count address

Hex Syntax: 009C word word long

Description: Draws a solid polygon from the coordinate list located at the address specified. The points specified in the vertex list are assumed to be offsets relative to the current x,y location. Fills the polygon according to the specified fill-type. This is a general polygon fill and is capable of filling non-convex polygons. If a polygon is known to be convex, better performance would be attained by using the CPFILL opcode.

Related opcodes: PFILLRV, PFILL, CPFILL, PFILLO

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern

General polygon fill (relative), variable

PFILLRV V*i*

Hex Syntax: 009D word

Parameters: Vi = fill type

Vi+1 = vertex count

Vi+2 = address of vertex list

Description: Draws a solid polygon from the coordinate list located at the address specified. The points specified in the vertex list are assumed to be offsets relative to the current x,y location. Fills the polygon according to the specified fill-type. This is a general polygon fill and is capable of filling non-convex polygons. If a polygon is known to be convex, better performance would be attained by using the CPFILL opcode. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PFILLR

Write Pixel

PIXEL x y

Syntax: 009E word word

Description: Writes a single pixel at the specified x,y location. Uses the current foreground color (COLORF)

Related opcodes: PIXELC, R_PIXEL, PPIXEL

Write Pixel (variable)

PIXELV V*i*

Syntax: 009F word

Parameters: Vi = X coordinate

Vi+1 = Y coordinate

Description: Writes a single pixel at the specified x,y location. Uses the current foreground color (COLORF). Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: PIXEL

Set Pixel to Color

PIXELC x y color

Syntax: 0150 word word long

Description: Writes a single pixel at the specified x,y location, using the specified color.

Related opcodes: PIXEL, R_PIXEL

Set Pixel to Color, variable

PIXELCV V*i*

Syntax: 0151 word

Parameters:

Vi	= X coordinate
Vi+1	= Y coordinate
Vi+2	= color

Description: Writes a single pixel at the specified x,y location, using the specified color. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PIXELC

Polyline

PLINE type count address

Syntax: 00A0 word word long

Description: Draws a line connecting successive points in the vertex list at the address specified. The vertex list is assumed to consist of "count" points (x,y pairs). The lines are drawn using the specified line-type.

Related opcodes: PLINEO, PLINER, LINE, LINETO

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile

Polyline (variable)

PLINEV V*i*

Syntax: 00A1 word

Parameters:

	Vi	= line type
	Vi+1	= vertex count
	Vi+2	= address of vertex list

Description: Draws a line connecting successive points in the vertex list at the address specified. The vertex list is assumed to consist of “count” points (x,y pairs). The lines are drawn using the specified line-type. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PLINE

Polyline (offset)

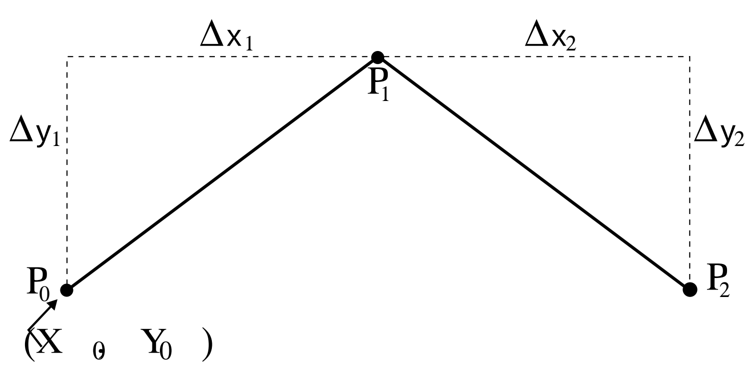
PLINEO type count address

Syntax: 0144 word word long

Description: Draws a line connecting successive points in the vertex list at the address specified. The vertex list is assumed to consist of “count” points (x,y pairs). The lines are drawn using the specified line-type. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated.

Related opcodes: PLINE, PLINER, LINE, LINETO, MOVETO

Line Types:
w_type=0= solid line
w_type=1= dash line (32 bit binary pattern)
w_type=2= dash line (arbitrary - segment-length byte list)
w_type=3= fatline - solid
w_type=4= fatline - stipple pattern (binary)
w_type=5= fatline - tile pattern (pixel-mapped)
w_type=6= pen line - solid
w_type=7= pen line - stipple
w_type=8= pen line - tile



Polyline (offset), variable

PLINEOV V*i*

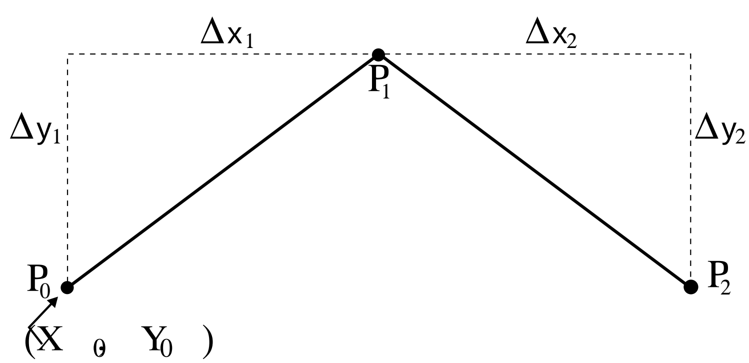
Syntax: 0145 word

Parameters:

Vi	= line type
Vi+1	= vertex count
Vi+2	= address of vertex list

Description: Draws a line connecting successive points in the vertex list at the address specified. The vertex list is assumed to consist of “count” points (x,y pairs). The lines are drawn using the specified line-type. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PLINEO



x listPolyline relative

PLINER type count address

Syntax: 00A2 word word long

Description: Draws a line connecting successive points in the vertex list at the address specified. The vertex list is assumed to consist of “count” points (x,y pairs). The lines are drawn using the specified line-type. The points specified are offsets relative to the current x,y location.

Related opcodes: PLINE, PLINEO, LINE, LINETO, MOVETO

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile

Polyline relative (variable)

PLINERV V*i*

Syntax: 00A3 word

Description: Draws a line connecting successive points in the vertex list at the address specified. The vertex list is assumed to consist of “count” points (x,y pairs). The lines are drawn using the specified line-type. The points specified are offsets relative to the current x,y location. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PLINER

Parameters:

Vi	= line type
Vi+1	= vertex count
Vi+2	= address of vertex list

Draw Poly-Line-Segments

PLINES type count address

Syntax: 0182 word word long

Description: Draws a series of line-segments connecting successive pairs of points in the vertex list at the address specified. The vertex list is assumed to consist of “count” line-segments (pairs of x,y points). The lines are drawn using the specified line-type. The points specified are relative to the current logical origin.

Related opcodes: PLINE, PLINESO, PLINESR

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

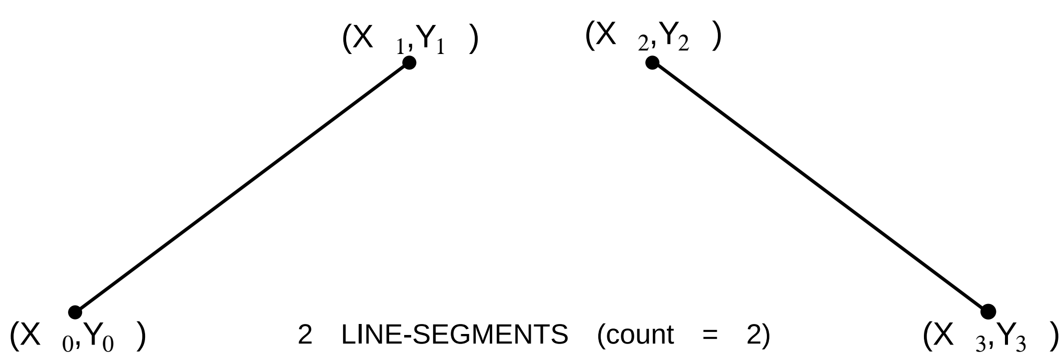
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



Draw Poly-Line-Segments, variable

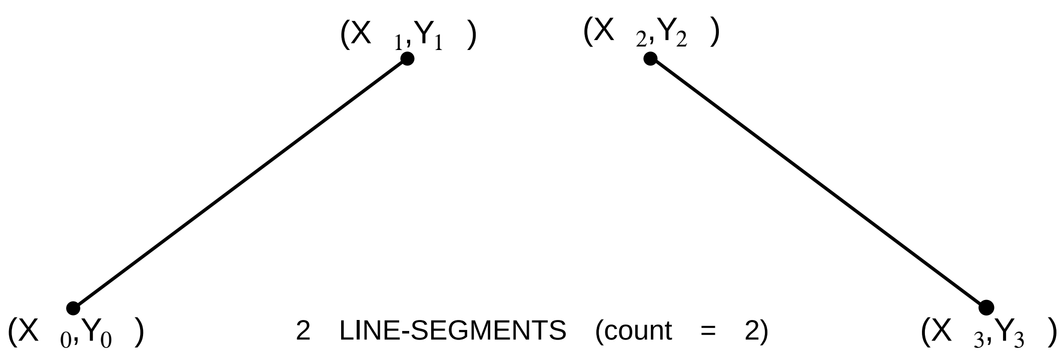
PLINESV V*i*

Syntax: 0183 word

Parameters:
Vi = line type
Vi+1 = line-segment count
Vi+2 = address of vertex list

Description: Draws a series of line-segments connecting successive pairs of points in the vertex list at the address specified. The vertex list is assumed to consist of "count" line-segments (pairs of x,y points). The lines are drawn using the specified line-type. The points specified are relative to the current logical origin. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PLINES



Draw Poly-Line-Segments (Relative)

PLINESR type count address

Syntax: 0184 word word long

Description: Draws a series of line-segments connecting successive pairs of points in the vertex list at the address specified. The vertex list is assumed to consist of "count" line-segments (pairs of x,y points). The lines are drawn using the specified line-type. The points specified are offsets relative to the current x,y location.

Related opcodes: PLINES,PLINER, LINER, LINETO, MOVETO

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

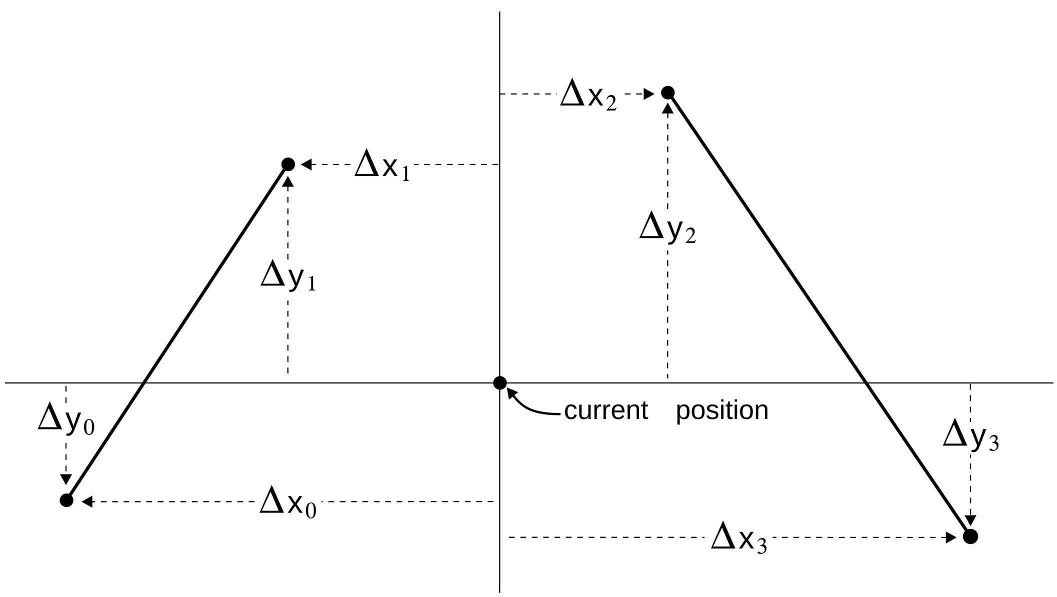
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



2 LINE-SEGMENTS (count = 2)

Draw Poly-Line-Segments (Relative), variable

PLINESRV V*i*

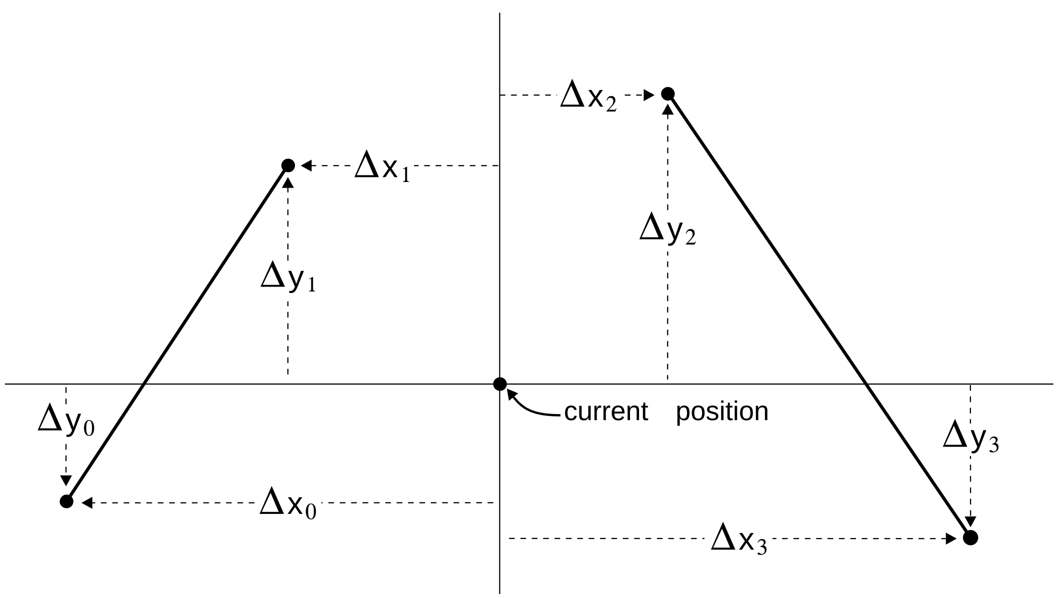
Syntax: 0185 word

Parameters:

Vi	= line type
Vi+1	= line-segment count
Vi+2	= address of vertex list

Description: Draws a series of line-segments connecting successive pairs of points in the vertex list at the address specified. The vertex list is assumed to consist of “count” line-segments (pairs of x,y points). The lines are drawn using the specified line-type. The points specified are offsets relative to the current x,y location. Vi is the first of 3 consecutive variables containing the parameters.

Related opcodes: PLINESR



2 LINE-SEGMENTS (count = 2)

Poly-marker

PMARK count address

Syntax: 00A4 word long

Description: Draws a “marker” at each of the points of the specified vertex list. The marker is specified with the MARKER opcode

Related opcodes: PMARKR, PMARKO, MARKER, PLINE

Discussion: The format of the vertex list is exactly the same as for the PLINE opcodes - thus PLINE & PMARK could be used in turn with the same vertex list to outline and highlight the data points of a graph.

Poly-marker (variable)

PMARKV V*i*

Syntax: 00A5 word

Parameters: Vi = vertex count

Vi+1 = address of vertex list

Description: Draws a “marker” at each of the points of the specified vertex list. The marker is specified with the MARKER opcode. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: PMARK

Discussion: The format of the vertex list is exactly the same as for the PLINE opcodes - thus PLINE & PMARK could be used in turn with the same vertex list to outline and highlight the data points of a graph.

Poly-marker (offset)

PMARKO count address

Syntax: 0146 word long

Description: Draws a “marker” at each of the points of the specified vertex list. The marker is specified with the MARKER opcode. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated.

Related opcodes: PMARK, PMARKR, MARKER, PLINEO

Discussion: The format of the vertex list is exactly the same as for the PLINEO opcodes.

Poly-marker (offset), variable

PMARKOV V*i*

Syntax: 0147 word

Parameters: Vi = vertex count

Vi+1 = address of vertex list

Description: Draws a “marker” at each of the points of the specified vertex list. The marker is specified with the MARKDER opcode. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: PMARKO

Discussion: The format of the vertex list is exactly the same as for the PLINEO opcodes.

Poly-marker (relative)

PMARKR count address

Syntax: 00A6 word long

Description: Draws a “marker” at each of the points of the specified vertex list. The marker is specified with the MARKER opcode. The points in the vertex list are assumed to be relative to (offsets from) the current x,y position. The current position in not changed.

Related opcodes: PMARK, PMARKO, MARKER, PLINER

Discussion: The format of the vertex list is exactly the same as for the PLINER opcodes - thus PLINER & PMARKR could be used in turn with the same vertex list to outline and highlight the data points of a graph.

Poly-marker (relative), variable

PMARKRV V*i*

Syntax: 00A7 word

Parameters: Vi = vertex count

Vi+1 = address of vertex list

Description: Draws a “marker” at each of the points of the specified vertex list. The marker is specified with the MARKER opcode. The points in the vertex list are assumed to be relative to (offsets from) the current x,y position. The current position in not changed. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: PMARKR

Discussion: The format of the vertex list is exactly the same as for the PLINER opcodes - thus PLINER & PMARKR could be used in turn with the same vertex list to outline and highlight the data points of a graph.

Set Plane Masking

PMASK long

Syntax: 00A8 long

Description: Plane masking allows the memory to be treated as bit planes, thereby allowing them to be selectively updated. Zeros in the pattern specify the planes that may be enabled for update. The default is all zeros. A bit-pattern corresponding to the system pixel-size must be replicated to 32 bits.

Related opcodes: None

Discussion: Plane masking may be used to enable selective bit planes for update. The memory on the graphics board is not organized as bit planes, but this opcode allows it to be treated as such.

Set Plane Masking (variable)

PMASKV V

Syntax: 00A9 word

Description: Plane masking allows the memory to be treated as bit planes, thereby allowing them to be selectively updated. Zeros in the pattern specify the planes that may be enabled for update. The default is all zeros. A bit-pattern corresponding to the system pixel-size must be replicated to 32 bits.

Related opcodes: None

Parameters: Vi = plane mask

Discussion: Plane masking may be used to enable selective bit planes for update. The memory on the graphics board is not organized as bit planes, but this opcode allows it to be treated as such.

Pop Variable

POPV V

Syntax: 00AA word

Description: The variable specified is loaded with the last 32 bits pushed on the AFGIS stack.

Related opcodes: PUSHV

Pop Variables

POPVARS number_of_variables V

Syntax: 00AB word word

Description: Consecutive variables, starting with the variable number specified by variable V are loaded with the last 32-bit longs that were pushed onto the AFGIS stack. The number of consecutive variables to be loaded (including the initial variable V) is specified in the *number_of_variables* parameter.

Related opcodes: PUSHVARS

Discussion: Variables are popped from the stack in the reverse order that they are pushed on the stack. Normally PUSHVARS and POPVARS opcodes would be used symmetrically, restoring the same values to the same variables.

Poly-pixel

PPIXEL count address

Syntax: 0186 word long

Description: Draws a pixel at each of the points of the specified vertex list. Uses the current foreground color.

Related opcodes: PPIXELR, PPIXELO, PMARK

Poly-pixel (variable)

PPIXELV V*i*

Syntax: 0187 word

Parameters: Vi = number of pixels
Vi+1 = address of vertex list

Description: Draws a pixel at each of the points of the specified vertex list. Uses the current foreground color. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: PPIXEL

Poly-pixel (offset)

PPIXELO count address

Syntax: 0196 word long

Description: Draws a pixel at each of the points of the specified vertex list. Uses the current foreground color. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated.

Related opcodes: PPIXEL, PPIXELR, PMARKO, PLINEO

Discussion: The format of the vertex list is exactly the same as for the PLINEO opcodes.

Poly-pixel (offset), variable

PPIXELOV V*i*

Syntax: 0197 word

Parameters: Vi = number of pixels

Vi+1 = address of vertex list

Description: Draws a pixel at each of the points of the specified vertex list. Uses the current foreground color. The first point in the coordinate list is relative to the logical origin. Successive points are each relative to (offsets from) the previous point. The current position is not updated. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: PPIXELO

Discussion: The format of the vertex list is exactly the same as for the PLINEO opcodes.

Poly-pixel (relative)

PPIXELR count address

Syntax: 0194 word long

Description: Draws a pixel at each of the points of the specified vertex list. Uses the current foreground color. The points in the vertex list are assumed to be relative to (offsets from) the current x,y position. The current position is not changed.

Related opcodes: PPIXEL, PPIXELO, PMARKR

Poly-pixel (relative), variable

PPIXELRV V*i*

Syntax: 0195 word

Parameters: Vi = vertex count

Vi+1 = address of vertex list

Description: Draws a pixel at each of the points of the specified vertex list. Uses the current foreground color. The points in the vertex list are assumed to be relative to (offsets from) the current x,y position. The current position is not changed. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: PPIXELR

Push Variable

PUSHV V

Syntax: 00AC word

Description: Pushes the variable specified by V onto the AFGIS stack.

Related opcodes: POPV

Push Variables

PUSHVARS number_of_variables V

Syntax: 01BD word word

Description: Pushes the number of consecutive variables (specified by the *number_of_variables* parameter), starting with variable number V onto the AFGIS stack.

Related opcodes: POPVARS

Discussion: Variables are popped from the stack in the reverse order that they are pushed on the stack. Normally PUSHVARS and POPVARS opcodes would be used symmetrically, restoring the same values to the same variables.

Set Range for Random Number

RANDRANGE low_boundary high_boundary

Syntax: 00AE long long

Description: Set the low and high number boundaries for the random integer to be generated by the R_RAND opcode. R_RAND will generate a random number “N” in the range low $\leq$ n $\leq$ high.

Related opcodes: R_RAND

Set Range for Random Number (variable)

RANDRANGEV V_i

Syntax: 00AF word

Parameters: Vi = random number range low value.

Vi+1 = random number range high value

Description: Set the low and high number boundaries for the random integer to be generated by the R_RAND opcode. R_RAND will generate a random number “N” in the range low to high. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: R_RAND

Set Seed Value For Random-number Function

RANDSEED seed

Syntax: 00B0 long

Description: Sets a new seed value for the random number function. There is NO default seed value - if no seed has been explicitly set before the R_RAND opcode is executed the first time, then a random seed is constructed from the on-board video refresh counters.

Related opcodes: RANDSEEDV, RANDRANGE, R_RAND

Set Seed Value For Random-number Function (variable)

RANDSEEDV V

Syntax: 00B1 word

Parameters: V = random-number seed

Description: Sets a new seed value for the random number function. There is NO default seed value - if no seed has been explicitly set before the R_RAND opcode is executed the first time, then a random seed is constructed from the on-board video refresh counters.

Related opcodes: RANDSEED, RANDRANGE, R_RAND

Draw a Rectangle

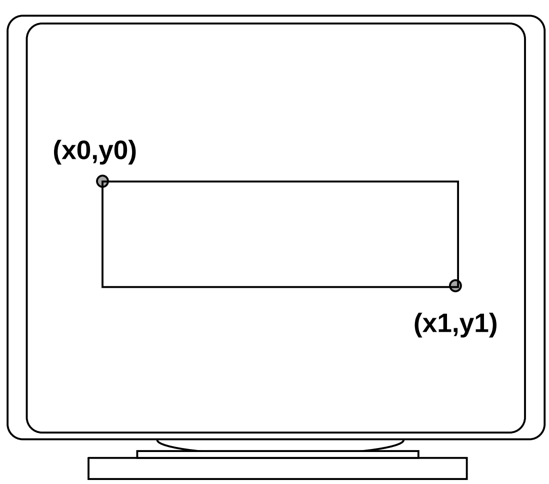
RECT type x0 y0 x1 y1

Syntax: 00B3 word word word word word

Description: Draws a rectangle. The upper left corner of the rectangle is located at (x0,y0) and the lower right corner of the rectangle is located at (x1,y1). Uses the specified line-type.

Related opcodes: RECTV, RECTS, RECTSV, RRECT

Line Types:
w_type=0= solid line
w_type=1= dash line (32 bit binary pattern)
w_type=2= dash line (arbitrary - segment-length byte list)
w_type=3= fatline - solid
w_type=4= fatline - stipple pattern (binary)
w_type=5= fatline - tile pattern (pixel-mapped)
w_type=6= pen line - solid
w_type=7= pen line - stipple
w_type=8= pen line - tile



Draw a Rectangle (variable)

RECTV V*i*

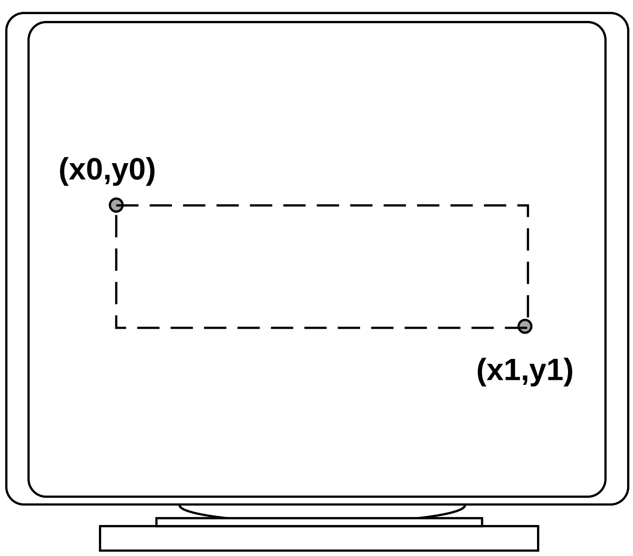
Syntax: 00B4 word

Parameters:

Vi	= line type
Vi+1	= X0
Vi+2	= Y0
Vi+3	= X1
Vi+4	= Y1

Description: Draws a rectangle. The upper left corner of the rectangle is located at (x0,y0) and the lower right corner of the rectangle is located at (x1,y1). Uses the specified line-type. Vi is the first of 5 consecutive variables containing the parameters.

Related opcodes: RECT



Draw a Filled Rectangle

RECTS type x0 y0 x1 y1

Syntax: 00B5 word word word word word

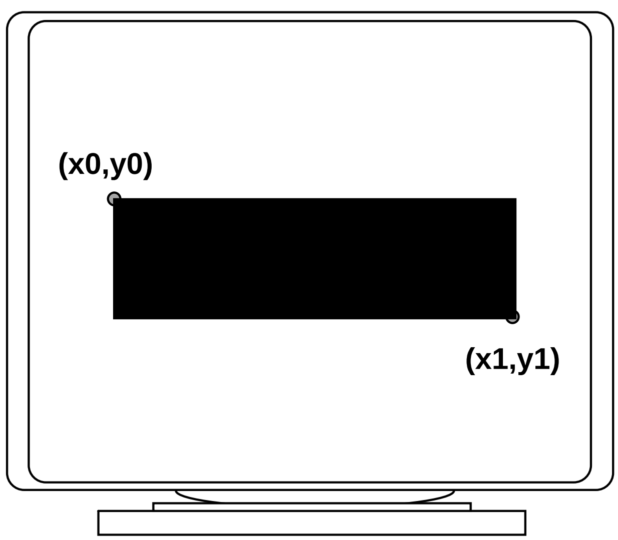
Description: Draws a rectangle. The upper left corner of the rectangle is located at (x0,y0) and the lower right corner of the rectangle is located at (x1,y1). Uses the specified fill-type.

Related opcodes: RECTV, RECTSV, RRECTS

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



Draw a Filled Rectangle (variable)

RECTSV Vi

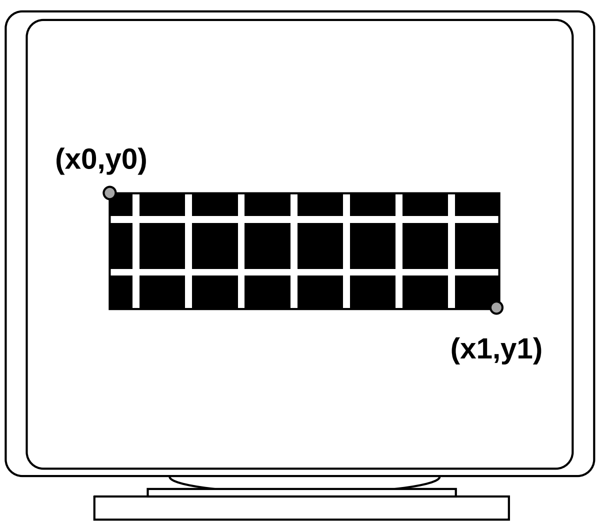
Syntax: 00B6 word

Parameters:

Vi	= fill type
Vi+1	= X0
Vi+2	= Y0
Vi+3	= X1
Vi+4	= Y1

Description: Draws a rectangle. The upper left corner of the rectangle is located at (x0,y0) and the lower right corner of the rectangle is located at (x1,y1). Uses the specified fill-type. Vi is the first of 5 consecutive variables containing the parameters.

Related opcodes: RECTS



Repeat

RPT #_of_times

Syntax: 00B7 word

Description: The display opcodes between RPT and the next ERPT in the display list are executed the number of times specified by word.

Related opcodes: ERPT, RPTV

Repeat (variable)

RPTV V

Syntax: 00B8 word

Description: The display opcodes between RPTV and the next ERPT are executed the number of times specified by V.

Related opcodes: ERPT, RPT

Discussion: Repeats can be nested. The ERPT belongs to the last RPTV.

Draw a Rounded Rectangle

RRECT type x0 y0 x1 y1 xradius yradius

Syntax: 015E word word word word word word word

Description: Draws a rounded rectangle. The upper left corner of the rectangle is located at (x0,y0) and the lower right corner of the rectangle is located at (x1,y1). Uses the specified line-type.

Related opcodes: RECT, RECTS, RRECTS

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

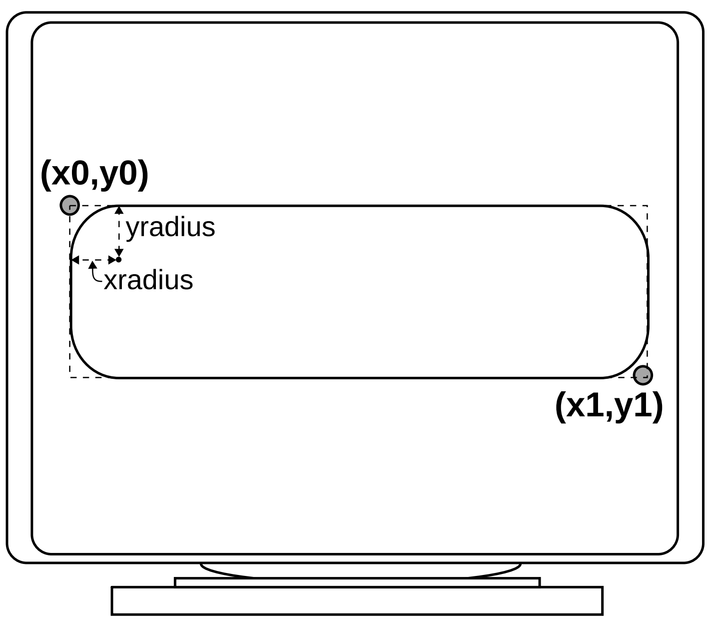
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



Draw a Rounded Rectangle, variable

RRECTV Vi

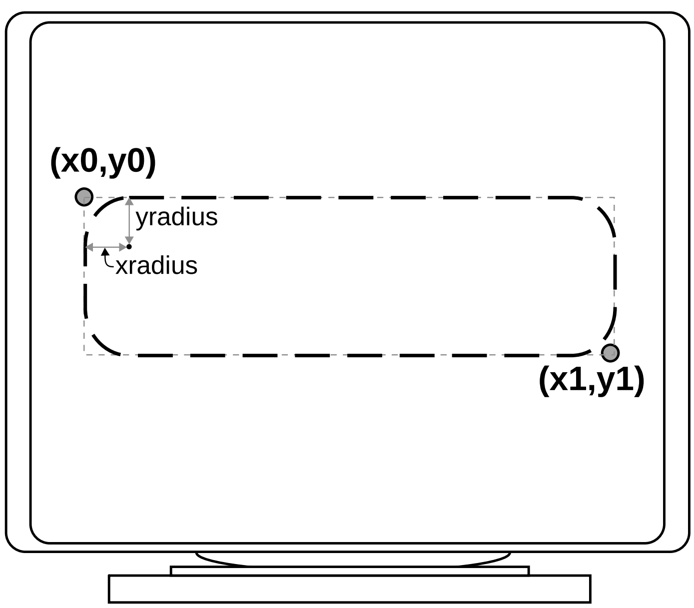
Syntax: 015F word

Parameters:

Vi	= line type
Vi+1	= X0
Vi+2	= Y0s
Vi+3	= X1
Vi+4	= Y1
Vi+5	= corner-fillet x-radius
Vi+6	= corner-fillet y-radius

Description: Draws a rounded rectangle. The upper left corner of the rectangle is located at (x0,y0) and the lower right corner of the rectangle is located at (x1,y1). Uses the specified line-type. Vi is the first of 7 consecutive variables containing the parameters.

Related opcodes: RRECT



Fill a Rounded Rectangle

RRECTS type x0 y0 x1 y1 xradius yradius

Syntax: 0130 word word word word word word word

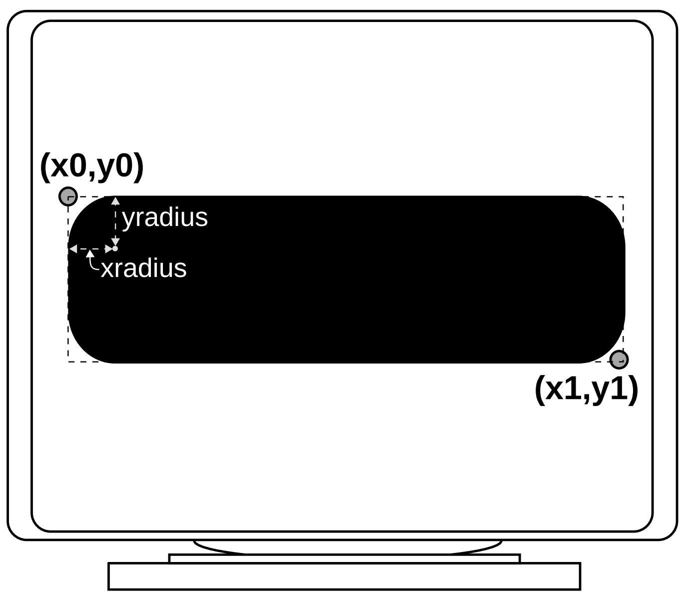
Description: Fills a rounded rectangle. The upper left corner of the rectangle is located at (x0,y0) and the lower right corner of the rectangle is located at (x1,y1). Uses the specified fill-type.

Related opcodes: RRECT, RRECTS

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



Fill a Rounded Rectangle, variable

RRECTSV Vi

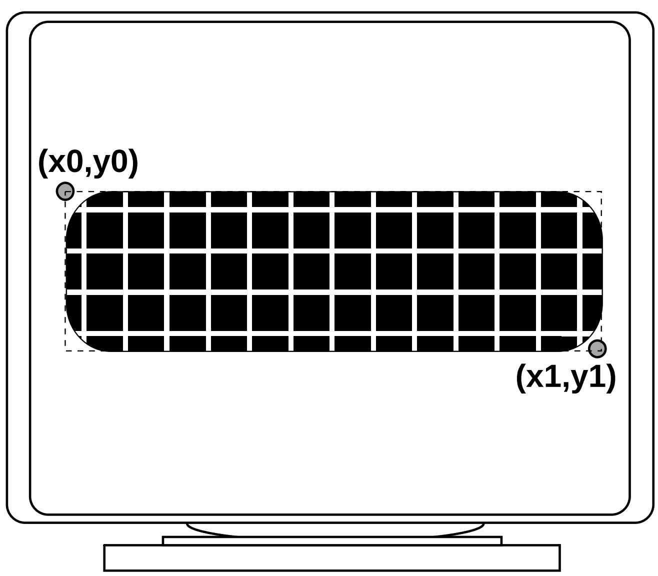
Syntax: 0131 word

Parameters:

Vi	= fill type
Vi+1	= X0
Vi+2	= Y0
Vi+3	= X1
Vi+4	= Y1
Vi+5	= corner-fillet x-radius
Vi+6	= corner-fillet y-radius

Description: Fills a rounded rectangle. The upper left corner of the rectangle is located at (x0,y0) and the lower right corner of the rectangle is located at (x1,y1). Uses the specified fill-type. Vi is the first of 7 consecutive variables containing the parameters.

Related opcodes: RRECTS



Return from Display List

RTRN

Syntax: 00B9

Description: Pops the opcode pointer off the AFIGS stack causing a return to the calling routine. Use this opcode at the end of a list of display opcodes that have been called with CAL.

Related opcodes: CAL

Read/Write Graphics Board Memory

RWMEM address clear_mask set_mask

Syntax: 0138 long word word

Description: Performs a read-modify-write operation on a word location in graphics board memory space. Allows the modification of a memory word in a single operation without intervening accesses by the firmware OS. One-bits in the clear-mask will cause a zero to be written to the corresponding bit position of the target word—simiarly, one-bits in the set-mask will cause a one to be written. Zero bits in the masks have no effect.

Related opcodes: LDVM, CONTREGX

Discussion: To clear (write zeros to) a word, use clearmask=0ffffh and setmask=0. To write a specific value “n” to a word, use clearmask=0ffffh and setmask=n.

Read/Write Graphics Board Memory, variable

RWMEMV Vi

Syntax: 0139

Parameters:

Vi	= address
Vi+1	= clear-mask
Vi+2	= set-mask

Description: Performs a read-modify-write operation on a word location in graphics board memory space. Allows the modification of a memory word in a single operation without intervening accesses by the firmware OS. One-bits in the clear-mask will cause a zero to be written to the corresponding bit position of the target word—simiarly, one-bits in the set-mask will cause a one to be written. Zero bits in the masks have no effect.

Related opcodes: RWMEM

Discussion: To clear (write zeros to) a word, use clearmask=0ffffh and setmask=0. To write a specific value “n” to a word, use clearmask=0ffffh and setmask=n.

Allocate Memory from Heap

R_ALLOC Nbytes V*o*

Syntax: 0004 long word

Description: Requests memory space for the number of bytes specified. A pointer to memory is returned in the variable Vo. If there is insufficient memory available, a value of zero is returned in the variable.

Related opcodes: R_FREE, R_ ISIZE

Allocate Memory from Heap, variable

R_ALLOCV V_i V_o

Syntax: 0005 long word

Parameters: Input:

Vi = allocation size in bytes

Output:

Vo = allocation address (or NULL if error)

Description: Requests memory space for the number of bytes specified. A pointer to memory is returned in the variable Vo. If there is insufficient memory available, a value of zero is returned in the variable.

Related opcodes: R_ALLOC

Return Coordinates Of Last Arc Drawn

R_ARC V*o*

Syntax: 00BA word

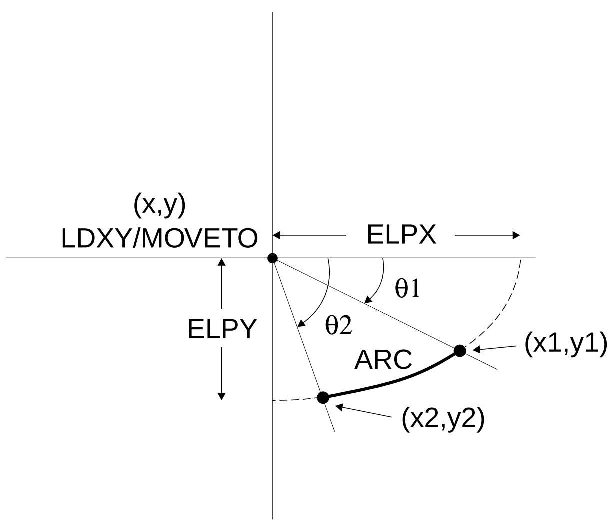
On Exit:

Vo	= X@center
Vo+1	= Y@center
Vo+2	= X@theta0 endpoint
Vo+3	= Y@theta0 endpoint
Vo+4	= X@theta1 endpoint
Vo+5	= Y@theta1 endpoint

Description: Returns the coordinates of the last arc drawn to the output variables listed above. The endpoint coordinates are relative to the arc center.

Vo is the first of 6 consecutive variables containing the data.

Related opcodes: R_ARCPTS, R_ARCTIC



Return Coordinates Of Arc

R_ARCPTS theta0 theta1 Xrad Yrad Vo

Syntax: 00BB word word word word word

On Exit:

Vo	= X@theta0 endpoint (center relative)
Vo+1	= Y@theta0 endpoint (center relative)
Vo+2	= X@theta1 endpoint (center relative)
Vo+3	= Y@theta1 endpoint (center relative)

Description: Returns the endpoints of the specified arc to four contiguous variables starting with V_o. Note that the endpoints returned are offsets relative to the center of an arc.

The R_ARCPTS opcode can be used to determine arc endpoints before they are drawn, to facilitate the dashed-line draw process. Vo is the first of 4 consecutive variables containing the data.

Related opcodes: ARC, SECT, SECTS, R_ARC

Return Coordinates Of Arc Endpoints (variable)

R_ARCPTSV V*i* V*o*

Syntax: 00BC word word

Parameters:

Vi	= start angle (theta0)
Vi+1	= end angle (theta1)
Vi+2	= X radius
Vi+3	= Y radius

On Exit:

Vo	= X@theta0 endpoint (center relative)
Vo+1	= Y@theta0 endpoint (center relative)
Vo+2	= X@theta1 endpoint (center relative)
Vo+3	= Y@theta1 endpoint (center relative)

Description: Returns the endpoints of the arc specified by a series of parameters stored in four variables beginning with V*i*, to four contiguous variables starting with V*o*. Note that the endpoints returned are offsets relative to the center of an arc.

The R_ARCPTS opcode can be used to determine arc endpoints before they are drawn, to facilitate the dashed-line draw process. Vi is the first of 4 consecutive variables containing the parameters. Vo is the first of 4 consecutive variables containing the data.

Related opcodes: ARC, SECT, SECTS, R_ARC

Return Arc Tic-mark Coordinates

R_ARCTIC angle length Xrad Yrad V*o*

Syntax: 00BD word word word word word

On Exit: Vo = X0 (outer endpoint)

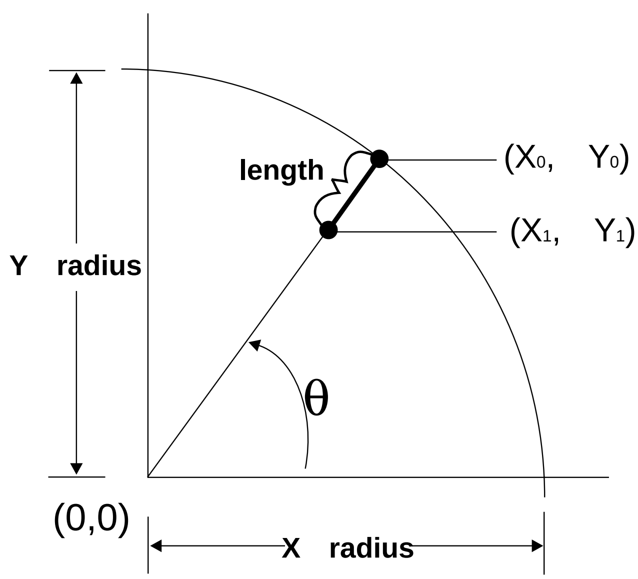
Vo+1 = Y0 (outer endpoint)

Vo+2 = X1 (inner endpoint)

Vo+3 = Y1 (inner endpoint)

Description: Returns the coordinates of the endpoints to an arc “tic-mark” for the specified arc parameters. The coordinates are returned in the output variables listed above, and are specified as offsets relative to the arc center. Vo is the first of 4 consecutive variables containing the data.

Related opcodes: R_ARCTICV, ARCTIC, R_ARCPTS



Return Arc Tic-mark Coordinates (variable)

R_ARCTICV V*i* V*o*

Syntax: 00BE word word

Parameters:

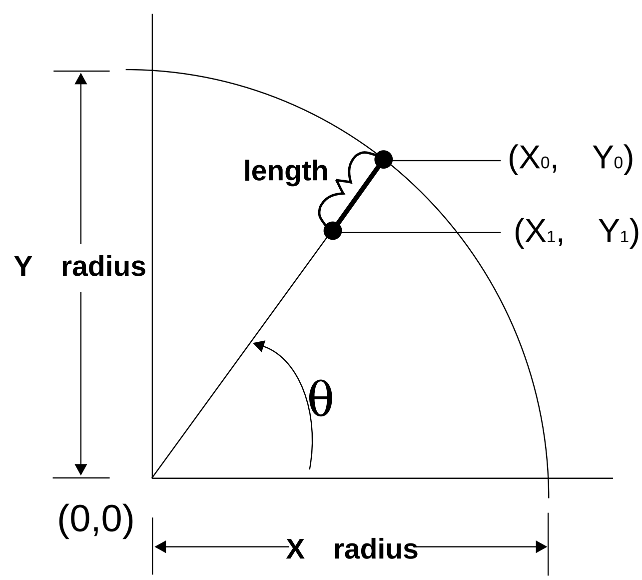
Vi	= angle
Vi+1	= tic-mark length
Vi+2	= X radius
Vi+3	= Y radius

On Exit:

Vo	= Xo (outer endpoint)
Vo+1	= Y0 (outer endpoint)
Vo+2	= X1 (inner endpoint)
Vo+3	= Y1 (inner endpoint)

Description: Returns the coordinates of the endpoints to an arc "tic-mark" for the specified arc parameters. The coordinates are returned in the outout variables listed above, and arc specified as offsets relative to the arc center. Vi is the first of 4 consecutive variables containing the parameters. Vo is the first of 4 consecutive variables containing the data.

Related opcodes: R_ARCTIC, ARCTIC, R_ARCPTS



Return "Clipcode"

R_CPW X Y V*o*

Syntax: 00BF word word word

Description: Compares the specified point to the current clipping window. Returns a "clipcode" in variable V*o* (see TMS340x0 User's Guide, "CPW" instruction for clipcode format).

Related opcodes: R_CPWV, CLIPWIN

Return “Clipcode” (variable)

R_CPWV V*i* V*o*

Syntax: 00C0 word word

Parameters: Vi = X coordinate

Vi+1 = Y coordinate

On Exit: Vo = clipcode

Description: Compares the specified point to the current clipping window. Returns a “clipcode” in variable V*o* (see TMS340x0 User’s Guide, “CPW” instruction for clipcode format). Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: R_CPW, CLIPWIN

Allocate Environment and Initialize for Draw Buffer

R_ENVB addr_ENV_params addr_DB addr_DB_params V_O

Syntax: 0152 long long long word

Description: Allocates memory from the on-board memory manager for a new environment, and initializes the environment header according to parameters contained in an environment parameter structure (addr_ENV_params). The environment graphics context is initialized to default values with the drawing context parameters configured for a draw-buffer at the specified address (addr_DB), in a configuration corresponding to parameters contained in a draw-buffer parameter structure (addr_DB_params). The address of the newly allocated environment is returned in variable Vo — NULL indicates an allocation error (insufficient memory).

Environment parameter buffer structure:

Offset	Bits	Name	Description
0000	16	flags	(see below)
0010	16	TC_len	length of text context area (TC) (bytes)
0020	16	nAFVARS	number of AFGIS variables (<=64)
0030	16	nXFVARS	number of XFORM variables (<=64)
0040	16	buff0_len	length of host buffer 0 (bytes)
0050	16	buff1_len	length of host buffer 1 (bytes)
0060	16	AFGSTK_len	length of AFGIS stack (bytes)
0070	16	GSPSTK_len	length of TMS stack (bytes)

- flags parameter:

<u>bit</u>	<u>(0=false/disable, 1=true/enable)</u>
0	allocate and clear TMS register block

- returns:

V*0* = address of new environment (NULL = error)

Note: the TC_len parameter should specify any environment- specific memory required by a custom text driver. A buffer of the specified size is allocated and a pointer to it placed in the TC_ptr field in the environment header. For resident (default) text drivers, this length parameter can be zero, since it is ALWAYS rounded up to the minimum size required by the resident drivers (so that any text driver buffer will always accommodate any resident text driver).

Draw-buffer parameter structure:

OFFS	SIZE	FIELD
0000	16	draw-buffer width in pixels
0010	16	draw-buffer height in pixels
0020	16	draw-buffer depth (pixel size)
0030	16	draw-buffer pitch

Related opcodes: R_ALLOC, R_FREE, R_ENVC, INITGCB, WPAGEB

Allocate Environment and Initialize for Draw Buffer, variable

R_ENVBV V*i* V*o*

Syntax: 0153 word word

Parameters: Input:

Vi = address of environment parameter structure

Vi +1 = address of draw buffer

Vi +2 = address of draw buffer parameter structure

Output:

On Exit: Vo = address of new environment (or NULL if error)

Description: Allocates memory from the on-board memory manager for a new environment, and initializes the environment header according to parameters contained in an environment parameter structure. The environment graphics context is initialized to default values with the drawing context parameters configured for a draw-buffer at the specified address, in a configuration corresponding to parameters contained in a draw-buffer parameter structure. The address of the newly allocated environment is returned in variable V— NULL indicates an allocation error (insufficient memory).

See R_ENVB for environment and draw-buffer parameter structure tables.

Related opcodes: R_ENVB

Allocate Environment and Initialize for Channel and Page

R_ENVC addr_ENV_params channel page V*o*

Syntax: 0154 long word word word

Description: Allocates memory from the on-board memory manager for a new environment, and initializes the environment header according to parameters contained in an environment parameter structure (addr_ENV_params). The environment graphics context is initialized to default values with the drawing context parameters configured for the specified channel and page. The address of the newly allocated environment is returned in variable V***o***— NULL indicates an allocation error (insufficient memory).

Environment parameter buffer structure:

Offset	Bits	Name	Description
0000	16	flags	(see below)
0010	16	TC_len	length of text context area (TC) (bytes)
0020	16	nAFVARS	number of AFGIS variables (<=64)
0030	16	nXFVARS	number of XFORM variables (<=64)
0040	16	buff0_len	length of host buffer 0 (bytes)
0050	16	buff1_len	length of host buffer 1 (bytes)
0060	16	AFGSTK_len	length of AFGIS stack (bytes)
0070	16	GSPSTK_len	length of TMS stack (bytes)

- flags parameter:

bit (0=false/disable, 1=true/enable)

0 allocate and clear TMS register block

- returns:

V0 = address of new environment (NULL = error)

- note:

The TC_len parameter should specify any environment- specific memory required by a custom text driver. A buffer of the specified size is allocated and a pointer to it placed in the TC_ptr field in the environment header. For resident (default) text drivers, this length parameter can be zero, since it is ALWAYS rounded up to the minimum size required by the resident drivers (so that any text driver buffer will always accommodate any resident text driver).

Related opcodes: R_ALLOC, R_FREE, R_ENVB, INITGCC, WPAGEC

Discussion: This opcode supports those boards having more than one graphics channel.

Allocate Environment and Initialize for Channel and Page, variable

R_ENVCV V*i* V*o*

Syntax: 0155 word word

Parameters:

V*i*	= address of parameter structure
V*i* +1	= channel ID
V*i* +2	= page #

On Exit: V*o* = address of new environment (or NULL if error)

Description: Allocates memory from the on-board memory manager for a new environment, and initializes the environment header according to parameters contained in an environment parameter structure. The environment graphics context is initialized to default values with the drawing context parameters configured for the specified channel and page. The address of the newly allocated environment is returned in variable V— NULL indicates an allocation error (insufficient memory).

See R_ENVC for the environment parameter buffer structure.

Related opcodes: R_ENVC

Deallocate Memory from Heap

R_FREE address V_o

Syntax: 0058 long word

Description: Frees memory previously allocated by the R_ALLOC, R_ENVB, or R_ENVC opcodes. The address specified must be the same as the address previously returned by the allocation opcode. If memory is successfully freed, a value of “1" is returned to the variable Vo . If a corresponding allocation block cannot be found , a value of ”0" is returned to Vo.

Related opcodes: R_ALLOC, R_ENVB, R_ENVC

Deallocate Memory from Heap, variable

R_FREEV V Vo

Syntax: 0059 word word

Description: Frees memory previously allocated by the R_ALLOC, R_ENVB, or R_ENVC opcodes. The variable V specifies the starting address, and must be the same as the address previously returned by the allocation opcode. If memory is successfully freed, a value of "1" is returned to the variable Vo. If a corresponding allocation block cannot be found , a value of "0" is returned to Vo.

Related opcodes: R_FREE

Return Image Size

R_ISIZE X0 Y0 X1 Y1 V*o*

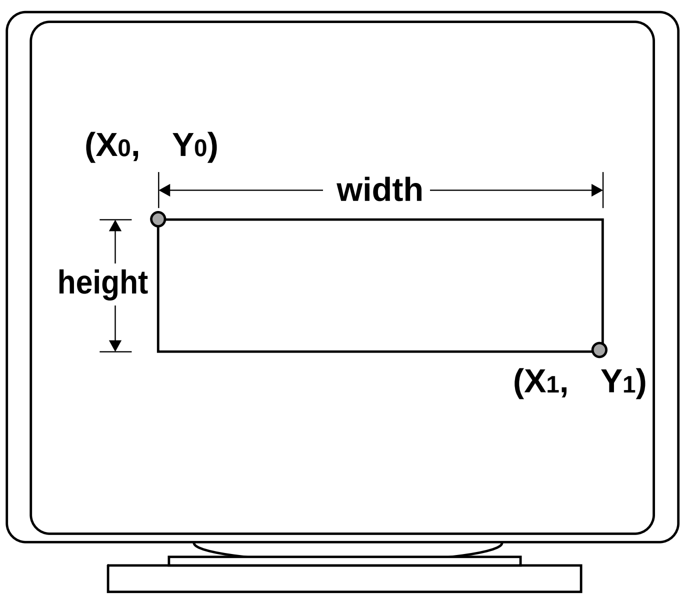
Syntax: 00C1 word word word word word

On Exit: Vo = image size in bytes

Description: Calculates the amount of memory (in bytes) that would be required to save an image bounded by the specified rectangle. The amount of memory required is returned to variable Vo.

Related opcodes: COPYSR, COPYSRV

Discussion: Use R_ISIZE to determine memory requirements for COPYSR save areas, as these functions require extra memory beyond that required strictly for the data save area, incorporating a header, and possibly padding fields to accommodate pitch restrictions for the TMS34010 processor.



Return Image Size (variable)

R_ISIZEV V*i* V*o*

Syntax: 00C2 word word

Parameters:

Vi	= X0 (left)
Vi+1	= Y0 (top)
Vi+2	= X1 (right)
Vi+3	= Y1 (bottom)

On Exit: Vo = image size in bytes

Description: Calculates the amount of memory (in bytes) that would be required to save an image bounded by the specified rectangle. The amount of memory required is returned to variable Vo.

Related opcodes: COPYSR, COPYSRV

Discussion: Use R_ISIZE to determine memory requirements for COPYSR save areas, as these functions require extra memory beyond that required strictly for the data save area, incorporating a header, and possibly padding fields to accommodate pitch restrictions for the TMS34010 processor.

Return Pixel Color

R_PIXEL X Y V*o*

Syntax: 00C5 word word word

Description: Returns the color of the pixel at the (x,y) location specified to the variable Vo.

Related opcodes: PIXEL, PIXELC

Return Pixel Color (variable)

R_PIXELV V*i* V*o*

Syntax: 00C6 word word

Parameters: Vi = X coordinate

Vi+1 = Y coordinate

On Exit: Vo = color

Description: Returns the color of the pixel at the (x,y) location specified to the variable Vo. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: PIXEL, PIXELC

Generate Random Number

R_RAND V

Syntax: 00B2 word

Description: Generates a random number in the range specified by the last RANDRANGE opcode and returns it in the variable V.

Related opcodes: RANDSEED, RANDRANGE

Return RGB color of Single Palette Entry

R_RGB channel index V*o*

Syntax: 0190 word word word

Description: Reads the RAMDAC color lookup table entry corresponding to the specified graphics output channel and palette index (RAMDAC address, pixel color value), and returns the RGB color value in the AFGIS variable specified.

RGB colors are formatted as follows:

FIELD	DESCRIPTION
bits 0...7	RED level (0..255)
bits 8..15	BLUE level (0..255)
bits 16..23	GREEN level (0..255)
bits 24..31	(padding) (0)

Related opcodes: SETRGB, GETPALETTE

Return RGB color of Single Palette Entry, variable

R_RGBV V_i V_o

Syntax: 0191 word

Parameters: V_i = channel ID

V_i +1 = color index (RAMDAC address)

On Exit:

V_o = RGB color

Description: Reads the RAMDAC color lookup table entry corresponding to the specified graphics output channel and palette index (RAMDAC address, pixel color value), and returns the RGB color value in the AFGIS variable specified.

RGB colors are formatted as follows:

FIELD	DESCRIPTION
bits 0...7	RED level (0..255)
bits 8..15	BLUE level (0..255)
bits 16..23	GREEN level (0..255)
bits 24..31	(padding) (0)

Related opcodes: R_RGB

Return Text Dimensions, Indirect Address

R_TEXTDA address V_o

Syntax: 00C7 long word

On Exit: Vo = X extent

Vo+1 = Y extent

Description: This opcode can be used to center a text string with respect to a point on the screen. R_TEXTDA calculates the x and y extents (in pixels) of the text string located at the *address* specified, and returns these values to variables V_o and $V_{o+1.}$ Values are calculated based on the current font and text space settings. Vo is the first of 2 consecutive variables containing the data.

Related opcodes: FONT, TEXTP

Return Text Dimensions (variable indirect)

R_TEXTDV V*i* V*o*

Syntax: 00C8 word word

Parameters: Vi = address of string

On Exit: Vo = X extent

Vo+1 = Y extent

Description: This opcode can be used to center a text string with respect to a point on the screen. R_TEXTDA calculates the x and y extents (in pixels) of the text string located at the *address* specified, and returns these values to variables V_o and V_{o+1}. Values are calculated based on the current font and text space settings. Vi is the first on 1 consecutive variable containing the parameters. Vo is the first of 2 consecutive variables containing the data.

Related opcodes: FONT, TEXTP

Return Text Dimensions, explicit format, Indirect Address

R_TEXTDXA mode address V_o

Syntax: 0106 word long word

On Exit: V_o = X extent

V_{o+1} = Y extent

Description: This opcode can be used to center a text string with respect to a point on the screen. R_TEXTDXA calculates the x and y extents (in pixels) of the text string located at the *address* specified, and returns these values to variables V_o and V_{o+1}. Values are calculated based on the current font and text space settings. Vo is the first of 2 consecutive variables containing the data. The string is assumed to be in the format as specified by the mode parameter as follows:

mode	string format
0	byte-packed, NULL-terminated
1	byte-packed, byte-swapped, NULL-terminated

Related opcodes: R_TEXTD

Return Text Dimensions, explicit format (variable indirect)

R_TEXTDXV V_i V_o

Syntax: 0107 word word

Parameters: Vi = string format mode
V_{i+1} = address of string

On Exit: V_o = X extent

V_{o+1} = Y extent

Description: This opcode can be used to center a text string with respect to a point on the screen. R_TEXTDXA calculates the x and y extents (in pixels) of the text string located at the *address* specified, and returns these values to variables V_o and $V_{o+1.}$ Values are calculated based on the current font and text space settings. Vi is the first on 2 consecutive variables containing the parameters. Vo is the first of 2 consecutive variables containing the data. The string is assumed to be in the format as specified by the mode parameter as follows:

mode	string format
0	byte-packed, NULL-terminated
1	byte-packed, byte-swapped, NULL-terminated

Related opcodes: R_TEXTD

FONT, TEXTP FONT, TEXTPReturn Text Parameter

R_TEXTP FN# V*o*

Syntax: 00C9 word word

Description: Returns the value of the text parameter corresponding to the specified function code. The value is returned in variable Vo.

Related opcodes:TEXTP, FONT

Text parameter codes:

FN#	SIZE	FORMAT	DESCRIPTION
0	32	[XY]	Font ID, type (packed: [id, type])
1	32	pointer	Font address
2	32	[XY]	Font cell dimensions (packed: [dy,dx])
3	32	[XY]	Spacing (packed: [line_space, char_space])
4	32	integer	Text rotation
5	32	integer	Text direction
6	32	boolean	Line-feed sense (zero:+X/+Y, non-zero:-X/-Y (reverse))
7	32	pointer	Font id string (address of NULL-terminated string)
8	32	[XY]	Character baseline measurements: [descent, ascent]
9	32	[XY]	Whitespace (built-in spacing) (packed: [y,x])

NOTE: 1. NOMENCLATURE “[a,b]” indicates a 32-bit quantity composed of 2 separate 16-bit portions, where the “a” parameter is located in the most-significant word, and the “b” parameter in the least-significant word (or, if viewed as 16-bit words in linear memory, the “b” parameter word would be located at the lower address followed by the “a” parameter word).

2. The font ID returned by function 0 is either the index used to select the current font, or -1 (0FFFFh) if the font was selected directly by address (e.g., a downloaded user-defined font).

Return Text Parameter (variable)

R_TEXTPV V*i* V*o*

Syntax: 00CA word word

Parameters: Vi = function code

On Exit: Vo = value of parameter requested

Description: Returns the value of the text parameter corresponding to the specified function code. The value is returned in variable Vo.

Related opcodes: TEXTP, FONT

Text parameter codes:

FN#	SIZE	FORMAT	DESCRIPTION
0	32	[XY]	Font ID, type (packed: [id, type])
1	32	pointer	Font address
2	32	[XY]	Font cell dimensions (packed: [dy,dx])
3	32	[XY]	Spacing (packed: [line_space, char_space])
4	32	integer	Text rotation
5	32	integer	Text direction
6	32	boolean	Line-feed sense (zero:+X/+Y, non-zero:-X/-Y (reverse))
7	32	pointer	Font id string (address of NULL-terminated string)
8	32	[XY]	Character baseline measurements: [descent, ascent]
9	32	[XY]	Whitespace (built-in spacing) (packed: [y,x])

NOTE: 1. NOMENCLATURE "[a,b]" indicates a 32-bit quantity composed of 2 separate 16-bit portions, where the "a" parameter is located in the most-significant word, and the "b" parameter in the least-significant word (or, if viewed as 16-bit words in linear memory, the "b" parameter word would be located at the lower address followed by the "a" parameter word).

2. The font ID returned by function 0 is either the index used to select the current font, or -1 (0FFFFh) if the font was selected directly by address (e.g., a downloaded user-defined font).

Convert X,Y to linear address

R_XYLADD X Y V*o*

Syntax: 00CB word word word

Description: Each (x,y) screen coordinate on every page of video RAM has an associated location (linear address) in memory. The *r_xyladd* opcode returns the linear address of the screen coordinate specified (on the current page) to variable V*o.*

Return Linear Address of X,Y Coordinates

R_XYLADDV V*i* V*o*

Syntax:	00CC word word	
Parameters:	Vi	= X coordinate
	Vi+1	= Y coordinate
On Exit:	Vo	= corresponding linear address

Description: Each (x,y) screen coordinate on every page of video RAM has an associated location (linear address) in memory. The r_xyladd opcode returns the linear address of the screen coordinate specified (on the current page) to variable Vo.

Subtract Immediate from Variable

SBIV long V_d

Syntax: 00CD long word

Description: The value specified is subtracted from the variable specified. The zero flag is set if the result of the subtraction is zero.

Related opcodes: SBVV, ADIV, JUMPA, JUMPR

Flags Affected: NCVZ

Discussion: Variables hold 32 bit signed values. If the result of the subtraction is negative, the carry flag is set.

Subtract Variables

SBVV V_s V_d

Syntax: 00CE word word

Description: The value of V_s is subtracted from V_d. V_d=V_d-V_s. The zero flag is set if the new value of V_d is zero. The carry flag is set is the value of V_s is greater than the value of V_d.

Related opcodes: SBIV, JUMPA, JUMPR

Flags Affected: NCVZ

Discussion: Variables hold 32 bit signed values only. If a larger value is subtracted from a smaller value however, the result in the destination variable will be negative. Use the carry flag to detect this possibility.

Draw a Sector

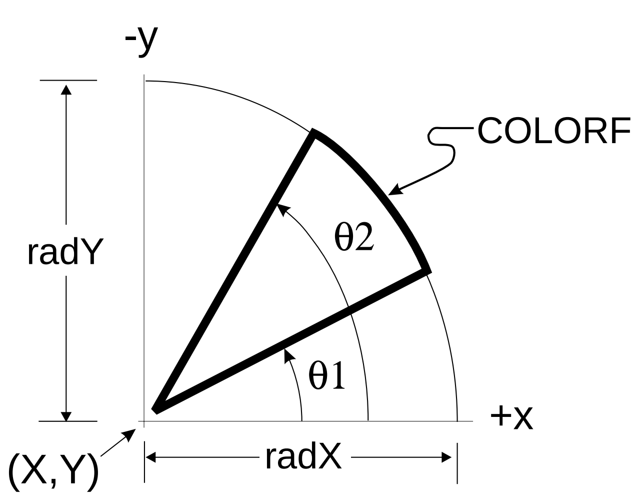
SECT type X Y θ_0 θ_1 xrad yrad

Syntax: 00CF word word word word word word word

Description: Draws a sector between two angles given in degrees counter clockwise from the positive x axis. Uses the specified line type.

Related opcodes: SECTS, ARC, SEG

Line Types:
- w_type=0= solid line
- w_type=1= dash line (32 bit binary pattern)
- w_type=2= dash line (arbitrary - segment-length byte list)
- w_type=3= fatline - solid
- w_type=4= fatline - stipple pattern (binary)
- w_type=5= fatline - tile pattern (pixel-mapped)
- w_type=6= pen line - solid
- w_type=7= pen line - stipple
- w_type=8= pen line - tile



Draw a Sector (variable)

SECTV V*i*

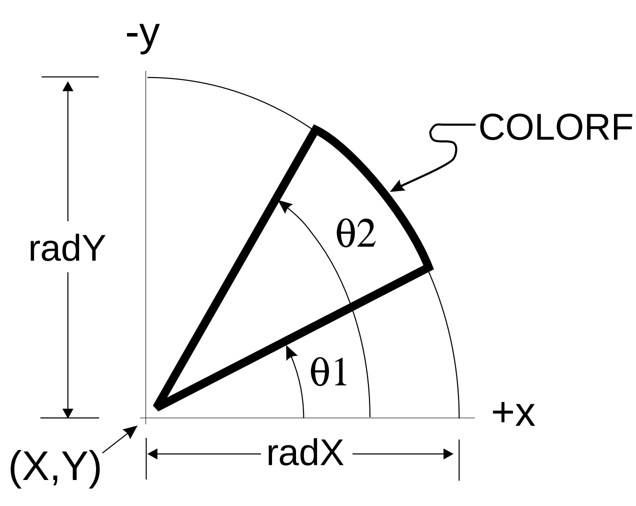
Syntax: 00D0 word

Parameters:

Vi	= line type
Vi+1	= X coordinate
Vi+2	= Y coordinate
Vi+3	= start angle
Vi+4	= end angle
Vi+5	= X radius
Vi+6	= Y radius

Description: Draws a sector between two angles given in degrees counter clockwise from the positive x axis. Uses the specified line type. Vi is the first of 7 consecutive variables containing the parameters.

Related opcodes: SECT



Draw a Solid Sector

SECTS type X Y θ*0* θ*1* Xrad Yrad

Syntax: 00D1 word word word word word word word

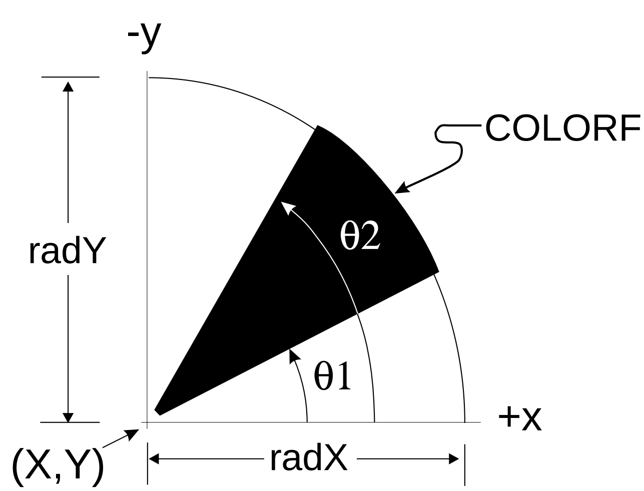
Description: Draws a solid sector between two angles given in degrees counter clockwise from the positive x axis.

Related opcodes: SECT, ARC, SEGS

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



Draw a Solid Sector (variable)

SECTSV V*i*

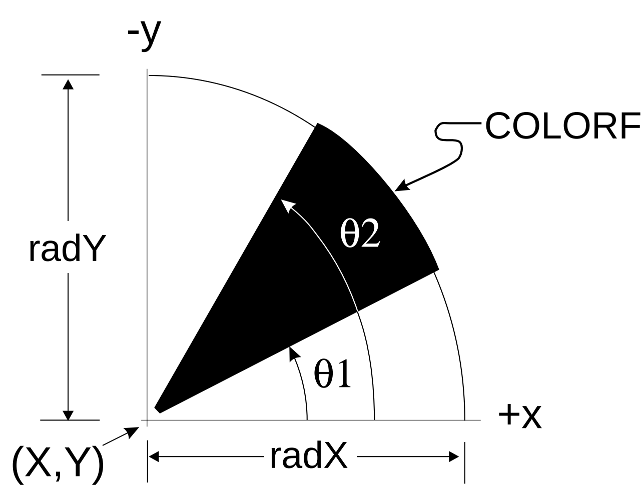
Syntax: 00D2 word

Parameters:

Vi	= fill type
Vi+1	= X coordinate
Vi+2	= Y coordinate
Vi+3	= start angle
Vi+4	= end angle
Vi+5	= X radius
Vi+6	= Y radius

Description: Draws a solid sector between two angles given in degrees counter clockwise from the positive x axis. Vi is the first of 7 consecutive variables containing the parameters.

Related opcodes: SECTS



Flood Seed Fill

SEEDFILL type X Y

Syntax: 00D3 word

Description: Flood-fills on area specified by the X,Y “seed” location. All pixels which are contiguous to and of the same color as the seed pixel are filled according to the specified fill type. The seed fill function is not compatible with boolean pixel processing operations other than “replace” and is not implemented for tile-pattern fills (fill type 2). The seed fill operation is aborted if the seed color (color at the seed pixel before filling) matches the foreground color for fill type 0 (solid) or either the foreground or background colors for fill type 1.

If not otherwise contained, the seedfill will be bounded by the current clipping rectangle.

Note that the seed fill may terminate prematurely with a stack-overflow error if the seed is located in the background of an intricate pattern, such as a field of tiny, non-touching stars.

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

Flood Seed Fill (variable)

SEEDFILLV Vi

Syntax: 00D4 word

Parameters:

Vi	= fill type
Vi+1	= seed X
Vi+2	= seed Y

Description: Flood-fills on area specified by the X,Y “seed” location. All pixels which are contiguous to and of the same color as the seed pixel are filled according to the specified fill type. The seed fill function is not compatible with boolean pixel processing operations other than “replace” and is not implemented for tile-pattern fills (fill type 2). The seed fill operation is aborted if the seed color (color at the seed pixel before filling) matches the foreground color for fill type 0 (solid) or either the foreground or background colors for fill type 1.

If not otherwise contained, the seedfill will be bounded by the current clipping rectangle.

Note that the seed fill may terminate prematurely with a stack-overflow error if the seed is located in the background of an intricate pattern, such as a field of tiny, non-touching stars. Vi is the first of 3 consecutive variables containing the parameters.

Draw Segment

SEG type X Y θ_0 θ_1 xrad yrad

Syntax: 0160 word word word word word word word

Description: Draws a segment between two angles given in degrees counter clockwise from the positive x axis. Uses the specified line type.

Related opcodes: SECT, ARC, SEGS

Line Types: w_type=0= solid line

w_type=1= dash line (32 bit binary pattern)

w_type=2= dash line (arbitrary - segment-length byte list)

w_type=3= fatline - solid

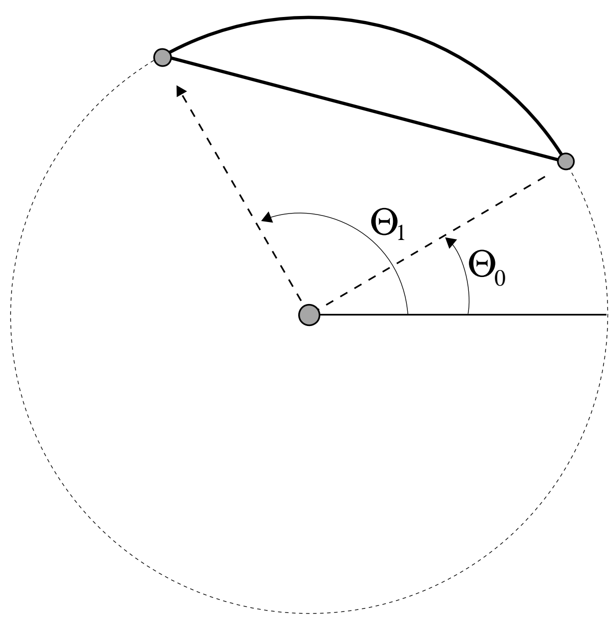
w_type=4= fatline - stipple pattern (binary)

w_type=5= fatline - tile pattern (pixel-mapped)

w_type=6= pen line - solid

w_type=7= pen line - stipple

w_type=8= pen line - tile



Draw Segment, variable

SEGV V*i*

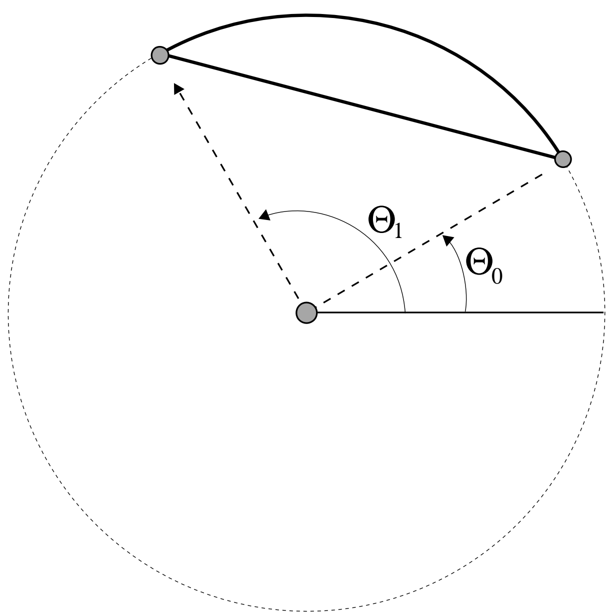
Syntax: 0161 word

Parameters:

Vi	= line type
Vi+1	= X coordinate
Vi+2	= Y coordinate
Vi+3	= start angle
Vi+4	= end angle
Vi+5	= X radius
Vi+6	= Y radius

Description: Draws a segment between two angles given in degrees counter clockwise from the positive x axis. Uses the specified line type. Vi is the first of 7 consecutive variables containing the parameters.

Related opcodes: SEG



Fill Segment

SEGS type X Y θ0 θ1 Xrad Yrad

Syntax: 015C word word word word word word word

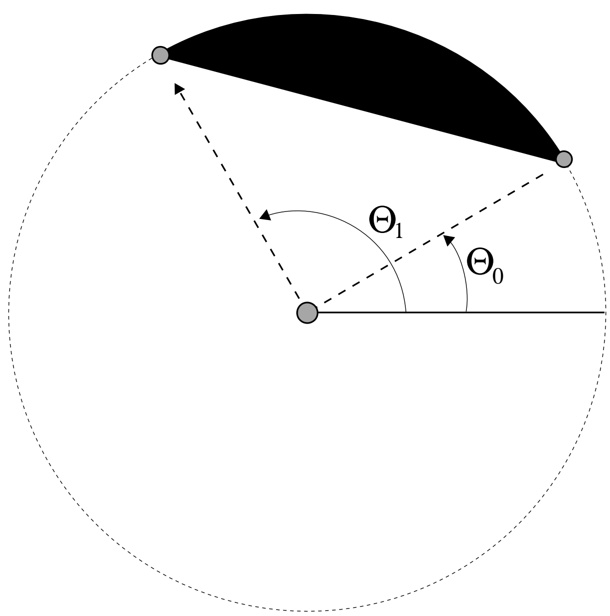
Description: Fills a segment between two angles given in degrees counter clockwise from the positive x axis.

Related opcodes: SECTS, SEG

Fill Types: w_type=0= solid color

w_type=1= stipple pattern

w_type=2= tile pattern



Fill Segment, variable

SEGSV V*i*

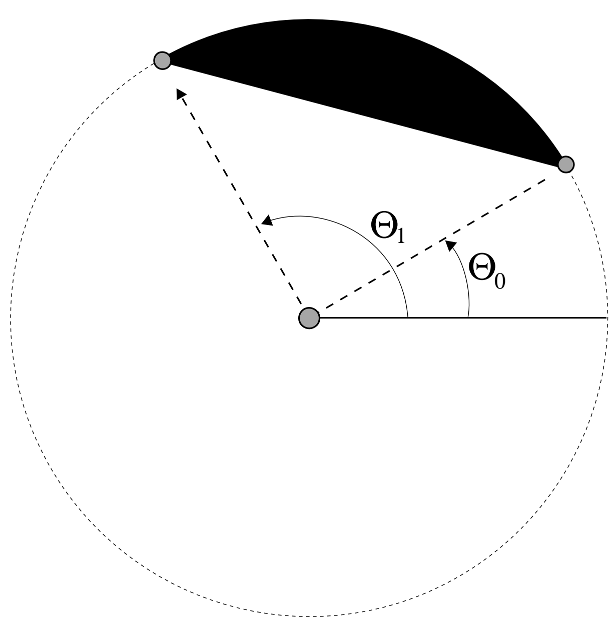
Syntax: 015D word

Parameters:

Vi	= fill type
Vi+1	= X coordinate
Vi+2	= Y coordinate
Vi+3	= start angle
Vi+4	= end angle
Vi+5	= X radius
Vi+6	= Y radius

Description: Fills a segment between two angles given in degrees counter clockwise from the positive x axis. Vi is the first of 7 consecutive variables containing the parameters.

Related opcodes: SEGS



Set baud rate for 2691 UART

SERBAUD word

Syntax: 00D5 word

Description: The value of the specified word sets the baud rate for the serial port as shown below:

Word	Baud Rate
0000	150
0001	300
0002	600
0003	1200
0004	2400
0005	4800
0006	9600
0007	19200
0008	38400

Related Opcodes: SERUART

Set baud rate for 2691 UART (variable)

SERBAUDV V

Syntax: 00D6 word

Parameters: V = baud-rate code

Description: The value of the variable sets the baud rate for the serial port as shown below:

Word	Baud Rate
0000	150
0001	300
0002	600
0003	1200
0004	2400
0005	4800
0006	9600
0007	19200
0008	38400

Related Opcodes: SERUART

Set Serial Port Interrupt Service Mode

SERMODE mode

Syntax: 00D7 word

Description: Enables the serial port for character-based I/O (e.g. an RS-232 port).

Related opcodes: SERBAUD, SERUART, SERQFL, SEROUT

Discussion: SERMODE configures the serial port to default parameters (9600 baud, 8 data bits, no parity, 1 stop bit). The serial port may be reconfigured with the SERUART or SERBAUD opcodes, but this must be done after executing SERMODE.

The serial port generates an interrupt when character data is available. If the host is unable to read the data, it is stored in the serial queue. Data is retrieved from the serial queue at 60Hz rate and made available to the host.

Additional data and error information is stored in RAM. See the RAM Listing for addresses.

Use “SERMODE” at the beginning of a serial polling session. This opcode turns on the serial interface and allows the bus CPU to poll fixed-RAM location SERFLAG to determine if serial data is available at RAM location SERIALDATA.

When the RAM location SERIALDATA contains a valid character, SERFLAG will be set to one. After the bus CPU reads the data from SERIALDATA, it must clear SERFLAG. Otherwise SERIALDATA will not be updated by the graphics processor, and serial data will be stored in the serial queue.

The serial queue is serviced with a 60Hz interrupt. Serial data stored in the serial queue is read at 60Hz, and SERIALDATA is updated at this rate.

Set Serial Port Interrupt Service Mode (variable)

SERMODEV V

Syntax: 00D8 word

Description: Enables the serial port for character-based I/O (e.g. an RS-232 port).

Related opcodes: SERBAUD, SERUART, SERQFL, SEROUT

Discussion: SERMODE configures the serial port to default parameters (9600 baud, 8 data bits, no parity, 1 stop bit). The serial port may be reconfigured with the SERUART or SERBAUD opcodes, but this must be done after executing SERMODE.

The serial port generates an interrupt when character data is available. If the host is unable to read the data, it is stored in the serial queue. Data is retrieved from the serial queue at 60Hz rate and made available to the host.

Additional data and error information is stored in RAM. See the RAM Listing for addresses.

Use “SERMODE” at the beginning of a serial polling session. This opcode turns on the serial interface and allows the bus CPU to poll fixed-RAM location SERFLAG to determine if serial data is available at RAM location SERIALDATA.

When the RAM location SERIALDATA contains a valid character, SERFLAG will be set to one. After the bus CPU reads the data from SERIALDATA, it must clear SERFLAG. Otherwise SERIALDATA will not be updated by the graphics processor, and serial data will be stored in the serial queue.

The serial queue is serviced with a 60Hz interrupt. Serial data stored in the serial queue is read at 60Hz, and SERIALDATA is updated at this rate.

Write Byte to Serial Port

SEROUT word

Syntax: 00D9 word

Description: Writes the 8 lsbs of the word specified to the serial port.

Related opcodes: SERMODE, SERBAUD, SERUART

Discussion: This opcodes writes a single byte to the serial port.

Write Byte to Serial Port (variable)

SEROUTV V

Syntax: 00DA word

Parameters: V = character (8-bits)

Description: Writes the 8 lsbs of the variable to the serial port.

Related opcodes: SERMODE, SERBAUD, SERUART

Discussion: This opcodes writes a single byte to the serial port.

Flush serial port queue

SERQFL

Syntax: 00DB

Description: Flushes (resets) the serial port input queue, discarding any previously received characters that may have been queued and clears the serial ready flag in fixed-RAM.

Related opcodes: SERMODE

Initialize 2691 UART

SERUART address

Syntax: 00DC long

Description: Initializes the 2691 UART with user-specified parameters. The address specified is assumed to point to a buffer containing initialization data to be written into the 2691's configuration registers. The buffer is formatted as follows:

DATA	2691 REGISTER
word 0	MR1
word 1	MR2
word 2	CSR
word 3	CR
word 4	ACR
word 5	IMR
word 6	CTUR
word 7	CTLR

Related opcodes: SERBAUD

Discussion: At power up the 2691 UART is initialized, but the user can change the programming by reprogramming it with the SERUART opcode. The 2691 has many options and the user is urged to consult the Signetics 2691 spec sheet before attempting to reprogram the 2691. Use the SERBAUD opcode to change the baud rate only. The power up configuration for the serial port is: 8 bits, no parity, 9600 baud, and 1 stop bit.

Initialize 2691 UART (variable)

SERUARTV @V

Syntax: 00DD WORD

Parameters: @V = address of initialization data buffer (8 words)

Description: Initializes the 2691 UART with user-specified parameters. The address specified is assumed to point to a buffer containing initialization data to be written into the 2691's configuration registers. The buffer is formatted as follows:

DATA	2691 REGISTER
word 0	MR1
word 1	MR2
word 2	CSR
word 3	CR
word 4	ACR
word 5	IMR
word 6	CTUR
word 7	CTLR

Related opcodes: SERBAUD

Discussion: At power up the 2691 UART is initialized, but the user can change the programming by reprogramming it with the SERUART opcode. The 2691 has many options and the user is urged to consult the Signetics 2691 spec sheet before attempting to reprogram the 2691. Use the SERBAUD opcode to change the baud rate only. The power up configuration for the serial port is: 8 bits, no parity, 9600 baud, and 1 stop bit.

Select Palette

SETPALETTE channel palette

Syntax: 0112 word long

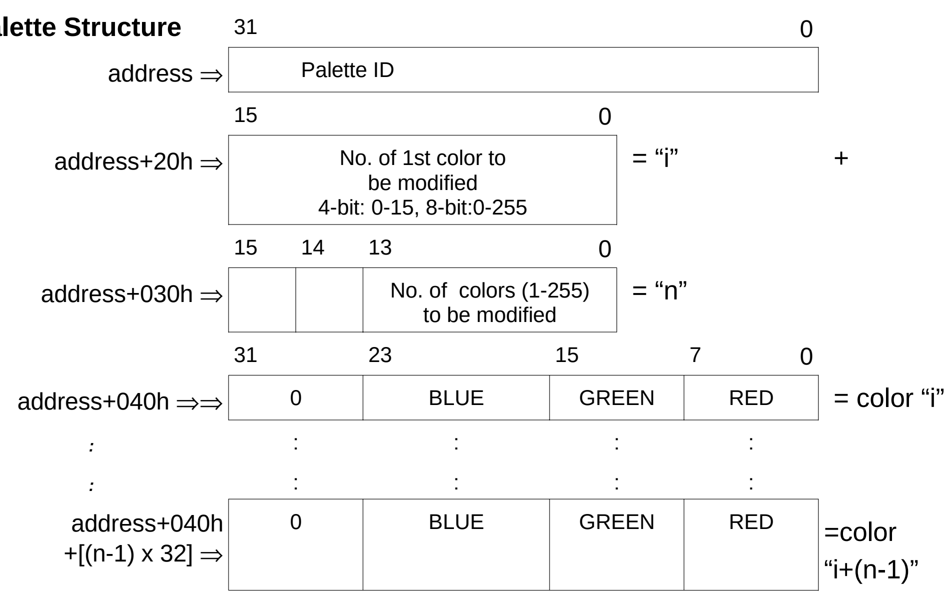
Description: Selects a color palette indicated by the specified parameter. This opcode loads the selected palette into the RAMDAC color look-up table for the specified channel.

If the palette parameter is within the range 0 to 127, this number is used to specify one of the pre-defined palettes.
If the palette parameter is greater than 127, this number is assumed to be the starting address of palette information (see Palette Structure shown below).

The following pre-defined palettes are available:

Palette #	Description
0	Provides 16 shades of gray
1	Provides 16 colors, formatted RGBI
2	Same as 1 except halftones are a little brighter
3	Provides 16 colors, CGA format
8	Provides 256 shades of gray
9	Provides 256 colors coded as follows:
^	D7 D6 D5 D4 D3 D2 D1 D0
^	Blue Green Red
10	Provides 256 colors coded as follows:
^	D7 D6 D5 D4 D3 D2 D1 D0
^	I2 B2 G2 R2 I1 B1 G1 R1

Palette Structure



SETPALETTE (continued)

Pallette ID

The palette ID field contains a unique code for each of the default palettes. This field is ignored for a user-defined palette.

Number of 1st Color ("i")

The RAMDAC color palette can be programmed in its entirety or in selected blocks. To reprogram the entire color palette, place a 0 in the first 16-bit word located at the address specified ("i"). To change a contiguous portion of the RAMDAC palette, indicate the number of the first color to be modified (0-15 for 4-bit systems, 0-255 for 8-bit systems). 0 corresponds to the first color in the RAMDAC table, 1 corresponds to the second color, etc.

Number of Colors ("n")

The second 16-bit word in the Palette Structure is used to indicate the number of contiguous colors in the RAMDAC table that are to be reprogrammed. This number, the color count "n", is stored in bits 0-13. Bit 15 is used as a flag, and should be set = 1 for 4-bit (16 color) systems. If bit 14 is a zero, the 32-bit packing format is used as shown. If bit 14 is a one, the color entries are packed into contiguous 24-bit fields and the 8 bits of padding shown are not present.

Colors ("c1" through "cn")

The remaining data in the Palette Structure contains color information, organized in long (32-bit) words. The first color will be written to the RAMDAC table location "i" (first word of the Palette Structure). The next contiguous locations in the RAMDAC table will be programmed with the colors specified. (The number of colors "n" is stored in the second word of the Palette Structure.) Each 32-bit word is organized into bytes of red, green, and blue information. The upper 8 bits should be zeros.

Related opcodes: GETPALETTE, SETRGB, R_RGB

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751, RG-752 and RG-753. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

For boards that only support a single channel, specify a channel ID of "0" (default underlay).

Select a Color Palette, variable

SETPALETTEV V*i*

Syntax: 0113 word

Parameters: Vi = channel ID

V_i +1 = address of palette structure (or index of default)

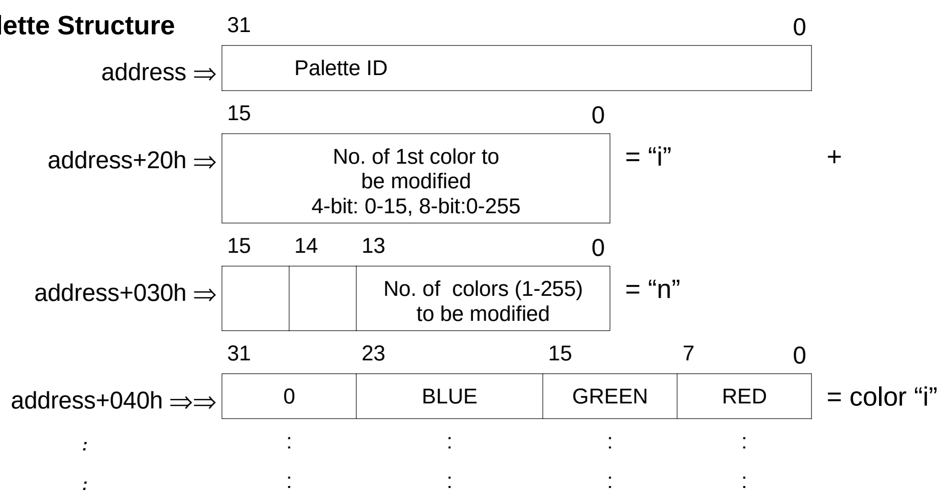
Description: Selects a color palette indicated by the specified parameter. This opcode loads the selected palette into the RAMDAC color look-up tablefor the specified channel.

If the palette parameter is within the range 0 to 127, this number is used to specify one of the pre-defined palettes. If the palette parameter is greater than 127, this number is assumed to be the starting address of palette information (see Palette Structure shown below).

The following pre-defined palettes are available:

Palette #	Description
0	Provides 16 shades of gray
1	Provides 16 colors, formatted RGBI
2	Same as 1 except halftones are a little brighter
3	Provides 16 colors, CGA format
8	Provides 256 shades of gray
9	Provides 256 colors coded as follows: D7 D6 D5 D4 D3 D2 D1 D0 Blue Green Red
10	Provides 256 colors coded as follows: D7 D6 D5 D4 D3 D2 D1 D0 I2 B2 G2 R2 I1 B1 G1 R1

Palette Structure



address+040h +[(n-1) x 32] ⇒	0	BLUE	GREEN	RED	=color "i+(n-1)"

SETPALETTEV (continued)

Pallette ID

The palette ID field contains a unique code for each of the default palettes. This field is ignored for a user-defined palette.

Number of 1st Color ("i")

The RAMDAC color palette can be programmed in its entirety or in selected blocks. To reprogram the entire color palette, place a 0 in the first 16-bit word located at the *address* specified ("i"). To change a contiguous portion of the RAMDAC palette, indicate the number of the first color to be modified (0-15 for 4-bit systems, 0-255 for 8-bit systems). 0 corresponds to the first color in the RAMDAC table, 1 corresponds to the second color, etc.

Number of Colors ("n"*)*

The second 16-bit word in the Palette Structure is used to indicate the number of contiguous colors in the RAMDAC table that are to be reprogrammed. This number, the color count "n", is stored in bits 0-13. Bit 15 is used as a flag, and *should be set = 1 for 4-bit (16 color) systems.* If bit 14 is a zero, the 32-bit packing format is used as shown. If bit 14 is a one, the color entries are packed into contiguous 24-bit fields and the 8 bits of padding shown are not present.

Colors ("c1" through "cn")

The remaining data in the Palette Structure contains color information, organized in long (32-bit) words. The first color will be written to the RAMDAC table location "i" (first word of the Palette Structure). The next contiguous locations in the RAMDAC table will be programmed with the colors specified. (The number of colors "n" is stored in the second word of the Palette Structure.) Each 32-bit word is organized into bytes of red, green, and blue information. The upper 8 bits should be zeros.

Related opcodes: SETPALETTE

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751, RG-752 and RG-753. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

For boards that only support a single channel, specify a channel ID of "0" (default underlay).

Set Single Palette Entry to RGB Color

SETRGB channel index RGB_color

Syntax: 018E word word long

Description: Writes the specified RGB color to the RAMDAC color lookup table entry corresponding to the specified graphics output channel and palette index (RAMDAC address, pixel color value). RGB colors are formatted as follows:

FIELD	DESCRIPTION
bits 0...7	RED level (0...255)
bits 8..15	GREEN level (0...255)
bits 16..23	BLUE level (0...255)
bits 24..31	(padding) (0)

Set Single Palette Entry to RGB Color, variable

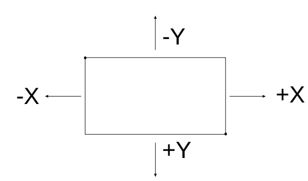
SETRGBV V*i*

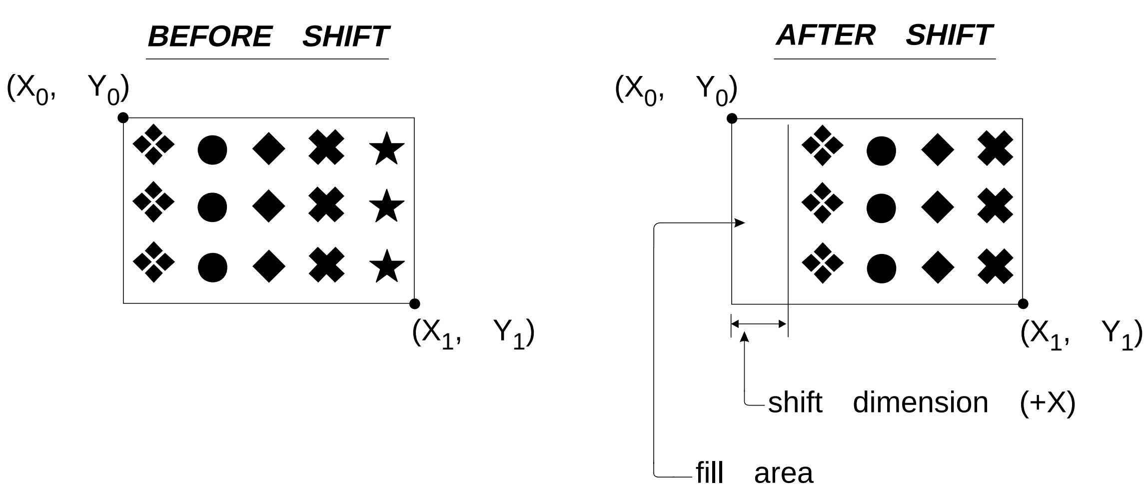
Syntax: 018F Vi

Parameters:
Vi = channel ID
Vi +1 = color index (RAMDAC address)
Vi +2 = RGB color

Description: Writes the specified RGB color to the RAMDAC color lookup table entry corresponding to the specified graphics output channel and palette index (RAMDAC address, pixel color value). RGB colors are formatted as follows:

FIELD	DESCRIPTION
bits 0...7	RED level (0...255)
bits 8..15	GREEN level (0...255)
bits 16..23	BLUE level (0...255)
bits 24..31	(padding) (0)





Shift Screen Area

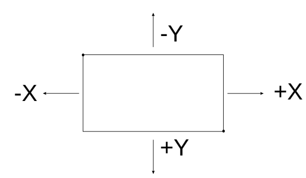
SHIFT pointer

Syntax: 00DE long

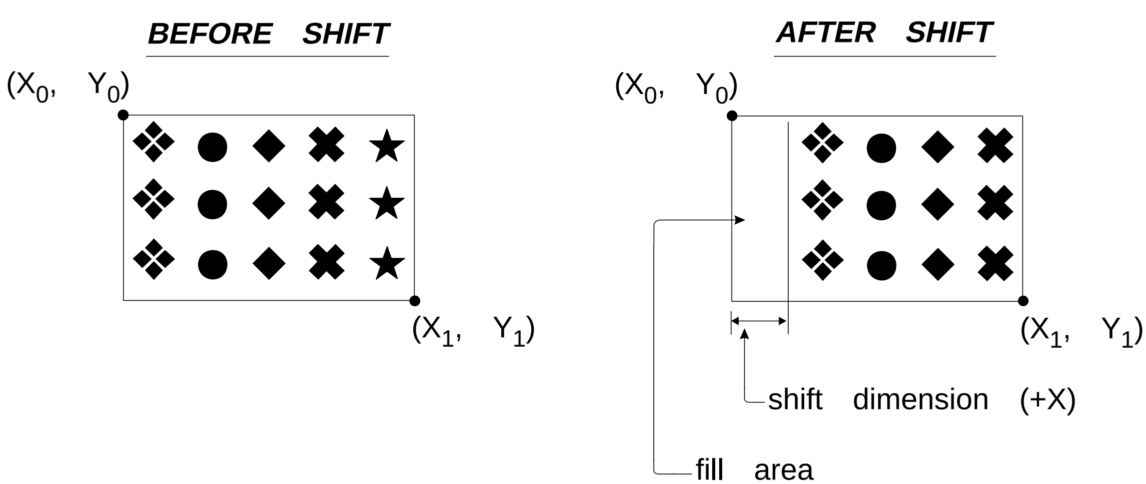
Description: Shifts the contents of a specified screen rectangle. If the parameter is <128 it is assumed to be the index of one of the default shift areas, and the corresponding parameters are used. Otherwise, the parameter is assumed to be the address of a shift-area definition structure. The shift area structure is formatted as follows:

OFFS	SIZE	FORMAT	DESCRIPTION
0000	32	[XY]	Upper left-hand corner of shift area
0020	32	[XY]	Lower right-hand corner of shift area
0040	32	[XY]	Shift dimensions (signed)
0060	32	color	Fill color (optional)
0080	16	coded	Mode field:
			76543210 (bit #'s)
			x00 - fill with specified color
			x01 - fill with background color
			x1x - don't fill
			0xx - use write page for source
			1xx - use display page for source

Discussion: The shift area dimensions are signed. A positive parameter shifts the area in the positive direction along the corresponding axis, and negative parameters shift in the negative direction.



When the area is shifted, some rows and/or columns are truncated (corresponding to the magnitudes of the shift dimensions) so that nothing is written outside of the specified rectangle. The vacated area is optionally filled as specified in the mode field.



Shift the Contents within an Area (variable)

SHIFTV V

Syntax: 00DF word

Parameters: V = address of shift area parameter structure (or index of default)

Description: Shifts the contents of a specified screen rectangle. If the parameter is <128 it is assumed to be the index of one of the default shift areas, and the corresponding parameters are used otherwise the parameter is assumed to be the address of a shift-area definition structure. The shift area structure is formatted as follows:

OFFS	SIZE	FORMAT	DESCRIPTION
0000	32	[XY]	Upper left-hand corner of shift area
0020	32	[XY]	Lower right-hand corner of shift area
0040	32	[XY]	Shift dimensions (signed)
0060	32	color	Fill color (optional)
0080	16	coded	Mode field: 76543210 (bit #'s) x00 - fill with specified color x01 - fill with background color x1x - don't fill 0xx - use write page for source 1xx - use display page for source

Discussion: The shift area dimensions are signed. A positive parameter shifts the area in the positive direction along the corresponding axis, and negative parameters shift in the negative direction.

When the area is shifted, some rows and/or columns are truncated (corresponding to the magnitudes of the shift dimensions) so that nothing is written outside of the specified rectangle. The vacated area is optionally filled as specified in the mode field.

Shift Left Logical Variable

SLLV count V

Syntax: 00E0 word word

Description: Shifts the contents of the variable left the number of bits specified by count.

Related opcodes: SRLV, JUMPA, JUMPR

Flags Affected: CZ (other flags undefined)

Discussion: Zeros are shifted into the variable from the right.

Shift Right Logical Variable

SRLV count V

Syntax: 00E1 word word

Description: Shifts the contents of the variable right by the number of bits specified by count.

Related opcodes: SLLV, JUMPA, JUMPR

Flags Affected: CZ (other flags undefined)

Discussion: Zeros are shifted into the variable from the left.

Display Simple Symbol

SSYM rotation symbol

Syntax: 00E2 word long

Description: Displays the symbol specified by the symbol parameter, with the specified rotation.

If the symbol parameter is <128, it is assumed to be the index of one of the default symbols, and the corresponding data is used. Otherwise, the number is assumed to be the starting address of the symbol data. The symbol is displayed at the current x,y location, in the foreground color specified by the last COLORF opcode.

Symbol structure:

OFFS	SIZE	DESCRIPTION
0000	16	Symbol width in pixels
0010	16	Symbol height in pixels
0020	16	Symbol depth (pixel size)
0030	16	Symbol pitch
0040	16	Symbol ID (default symbols)
0050	16	(unused)
0060		Beginning of symbol data

Related opcodes: SSYMV, SSYMX

Discussion: Symbol depth is assumed to be 1 (binary bitmap). The symbol pitch is the linear difference between successive rows of the symbol bitmap.

Symbol rotation parameter is interpreted as follows:

0 = no rotation

1 = 90° counter clockwise

2 = 180° counter clockwise

3 = 270° counter clockwise

4 = mirrored about center in X-dimension

5 = mirrored + 90° counter clockwise

6 = mirrored + 180° counter clockwise

7 = mirrored + 270° counter clockwise

Note that the symbol structure header is the same format as the STIPPLE or TILE pattern headers.

Display Simple Symbol (variable)

SSYMV V*i*

Syntax: 00E3 word

Parameters: Vi = rotation

Vi+1 = address of symbol structure (or index of default)

Description: Displays the symbol specified by the symbol parameter, with the specified rotation.

If the symbol parameter is <128, it is assumed to be the index of one of the default symbols, and the corresponding data is used. Otherwise, the number is assumed to be the starting address of the symbol data. The symbol is displayed at the current x,y location, in the foreground color specified by the last COLORF opcode.

Symbol structure:

OFFS	SIZE	DESCRIPTION
0000	16	Symbol width in pixels
0010	16	Symbol height in pixels
0020	16	Symbol depth (pixel size)
0030	16	Symbol pitch
0040	16	Symbol ID (default symbols)
0050	16	(unused)
0060		Beginning of symbol data

Related opcodes: SSYMV, SSYMX

Discussion: Symbol depth is assumed to be 1 (binary bitmap). The symbol pitch is the linear difference between successive rows of the symbol bitmap.

Symbol rotation parameter is interpreted as follows:

0 = no rotation

1 = 90° counter clockwise

2 = 180° counter clockwise

3 = 270° counter clockwise

4 = mirrored about center in X-dimension

5 = mirrored + 90° counter clockwise

6 = mirrored + 180° counter clockwise

7 = mirrored + 270° counter clockwise

Note that the symbol structure header is the same format as the STIPPLE or TILE pattern headers.

Draw Simple Symbol (immediate)

SSYMX rotation width height pitch address

Syntax: 00E4 word word word word long

Description: Display a symbol specified by "explicit" parameters. The width and height are the dimensions of the symbol; the pitch is the row-pitch of the symbol pixblt data (difference in bits between the beginning of successive rows of the pixblt data) and the address is that of the pixblt data.

Related opcodes: SSYM

Discussion: The SSYM opcode presumes a prefacing header which contains the symbol parameters (width, height, etc.). SSYMX allows these implicit parameters to be specified explicitly, so that pixblt data can be used as a symbol without the need of a prefacing header (e.g., the contents of a draw-buffer configured with WPAGEB).

Draw Simple Symbol, variable

SSYMXV V*i*

Syntax: 00E5 word

Parameters:

Vi	= rotation
Vi+1	= symbol width
Vi+2	= symbol height
Vi+3	= pitch
Vi+4	= pointer to symbol (pixblt) data

Description: Display a symbol specified by “explicit” parameters. The width and height are the dimensions of the symbol; the pitch is the row-pitch of the symbol pixblt data (difference in bits between the beginning of successive rows of the pixblt data) and the address is that of the pixblt data. Vi is the first of 5 consecutive variables containing the parameters.

Related opcodes: SSYM

Discussion: The SSYM opcode presumes a prefacing header which contains the symbol parameters (width, height, etc.). SSYMX allows these implicit parameters to be specified explicitly, so that pixblt data can be used as a symbol without the need of a prefacing header (e.g., the contents of a draw-buffer configured with WPAGEB).

Select Stipple (binary) Fill Pattern

STIPPLE pattern

Syntax: 00E6 long

Description: Selects a stipple (binary) fill pattern. Used when fill type “1" is specified for any of the area-fill opcodes. (CIRS, CPFILL, CPFILLR, ELPS, PFILL, PFILLR, RECTS, RRECTS, SECTS, SEEDFILL, SEGS)

Related opcodes: TILE

Discussion: If the pattern parameter is <128, it is assumed to be the index of one of the default stipple patterns, and the corresponding pattern is made current. Otherwise, the parameter is assumed to be the address of a stipple pattern definition structure.

Stipple pattern structure:

OFFS	SIZE	FIELD
0000	16	Pattern width
0010	16	Pattern height
0020	16	Pattern depth (pixel-size)
0030	16	Pattern pitch
0040	16	Pattern ID (default patterns)
0050	16	(unused)
0060		Beginning of pattern data

Pattern depth is assumed to be 1 (binary bit-map). The pattern pitch is the linear difference between successive rows of the pattern bit-map. The following constraints apply to pattern parameters: pattern width must be a power of 2 less than or equal to 32 (2, 4, 8, 16, 32); pattern height must be a power of 2; pattern pitch must be a multiple of 16, and for best results should be a power of 2 (16, 32, 64,...).

Note that the stipple pattern structure is exactly the same as the SSYM symbol structure. Therefore, SSYM symbols and stipple patterns may be used interchangeably.

Select Stipple (binary) Fill Pattern, variable

STIPPLEV V

Syntax: 00E7 word

Description: Selects a stipple (binary) fill pattern. Used when fill type “1" is specified for any of the area-fill opcodes. (CIRS, CPFILL, CPFILLR, ELPS, PFILL, PFILLR, RECTS, RRECTS, SECTS, SEEDFILL, SEGS)

Related opcodes: TILE

Discussion: If the parameter is <128, it is assumed to be the index of one of the default stipple patterns, and the corresponding pattern is made current. Otherwise, the parameter is assumed to be the address of a stipple pattern definition structure.

Stipple pattern structure:

OFFS	SIZE	FIELD
0000	16	Pattern width
0010	16	Pattern height
0020	16	Pattern depth (pixel-size)
0030	16	Pattern pitch
0040	16	Pattern ID (default patterns)
0050	16	(unused)
0060		Beginning of pattern data

Pattern depth is assumed to be 1 (binary bit-map). The pattern pitch is the linear difference between successive rows of the pattern bit-map. The following constraints apply to pattern parameters: pattern width must be a power of 2 less than or equal to 32 (2, 4, 8, 16, 32); pattern height must be a power of 2; pattern pitch must be a multiple of 16, and for best results should be a power of 2 (16, 32, 64,...).

Note that the stipple pattern structure is exactly the same as the SSYM symbol structure. Therefore, SSYM symbols and stipple patterns may be used interchangeably.

Select Stipple Fill Pattern, Explicit Parameters

STIPPLEX width height pitch address

Syntax: 018A word word word long

Description: Selects a stipple (binary) fill pattern. width and height are the dimensions of the fill pattern; pitch is the row-pitch of the pattern bitmap data (difference in bits between the beginning of successive rows of the bitmap data); and address is that of the bitmap data. The stipple pattern will be used when fill type “1" is specified for any of the area-fill opcodes. (CIRS, CPFILL, ELPS, PFILL, RECTS, RRECTS, SECTS, SEGS, SEEDFILL)

Stipple Pattern Structure:

OFFS	SIZE	FIELD
0000	16	Pattern width
0010	16	Pattern height
0020	16	Pattern depth (pixel-size)
0030	16	Pattern pitch
0040	16	Pattern ID (default patterns)
0050	16	(unused)
0060		Beginning of pattern data

Related opcodes: STIPPLE, TILE, TILEX, SSYMX

Discussion: This opcode facilitates the use of pattern data existing in a format where the pitch is not equal to the pattern width. This allows the use of a portion of a larger pattern, or the use of bitmap data generated in a 1-bit-per-pixel draw-buffer (as configured by R_ENVB, INITGCB or WPAGEB). The following constraints apply to pattern parameters: pattern width must be a power of 2 less than or equal to 32 (2, 4, 8, 16, 32); pattern height must be a power of 2; pattern pitch must be a multiple of 16, and for best results should be a power of 2 (16, 32, 64,...).

Select Stipple Fill Pattern, Explicit Parameter, variable

STIPPLEXV V

Syntax: 018B word

Parameters:
Vi = stipple pattern width
Vi +1 = stipple pattern height
Vi +2 = stipple pattern pitch
Vi +3 = address of stipple pattern bitmap data

Description: Selects a stipple (binary) fill pattern. width and height are the dimensions of the fill pattern; pitch is the row-pitch of the pattern bitmap data (difference in bits between the beginning of successive rows of the bitmap data); and the address is that of the bitmap data. The stipple pattern will be used when fill type “1" is specified for any of the area-fill opcodes. (CIRS, CPFILL, ELPS, PFILL, RECTS, RRECTS, SECTS, SEGS, SEEDFILL)

Stipple Pattern Structure:

OFFS	SIZE	FIELD
0000	16	Pattern width
0010	16	Pattern height
0020	16	Pattern depth (pixel-size)
0030	16	Pattern pitch
0040	16	Pattern ID (default patterns)
0050	16	(unused)
0060		Beginning of pattern data

Related opcodes: STIPPLEX

Discussion: This opcode facilitates the use of pattern data existing in a format where the pitch is not equal to the pattern width. This allows the use of a portion of a larger pattern, or the use of bitmap data generated in a 1-bit-per-pixel draw-buffer (as configured by R_ENVB, INITGCB or WPAGEB). The following constraints apply to pattern parameters: pattern width must be a power of 2 less than or equal to 32 (2, 4, 8, 16, 32); pattern height must be a power of 2; pattern pitch must be a multiple of 16, and for best results should be a power of 2 (16, 32, 64,...).

Set Text Parameter

TEXTP code value

Syntax: 00E8 word long

Description: Sets a new value for one of several text processing parameters. The “code” parameter indicates which text parameter is being updated, and “value” is the new value for the specified text parameter. Code values and their corresponding text parameters are shown in the following table. The format specified pertains to the “value” field.

Text Parameter Codes:

CODE#	FORMAT	DESCRIPTION
0	pointer	Font index/address (resets all parameters to defaults)
1	pointer	Font index/address (sets spacing parameters to defaults)
2	integer	Character spacing
3	integer	Line spacing
4	integer	Text rotation
5	integer	Text direction
6	boolean	Line-feed sense (zero: +X/+Y, nonzero: -X/-Y (reverse))

Font (code “0", ”1")

Selects the active font. When code “0" is used, all other font parameters are set to defaults. When code ”1" is used, only the character and line spacing parameters are reset for the new font—the previous settings for text rotation, text direction, and line-feed sense are retained. The code “0" defaults for RGI type 0 (”RGI0") fonts (used by the default text driver) are shown below. Alternate text drivers for other font formats may implement different defaults.

Parameter	Code “0" Defaults (”RGI0" Fonts)
character spacing	∅
line spacing	∅
text rotation	0
text direction	0
line-feed sense	0 (+X/+Y)

Set Text Parameter (continued)

Character Spacing: (code “2")

Specifies the amount of spacing (in pixels) that will be inserted between two adjacent character cells printed in succession (i.e., along the axis specified by the “text direction” parameter). “RGI0" fonts have a default character-spacing value at ∅. Note that if transparency is not enabled during text, the background portion of each character cell will be filled in with the current background color, but the area between character cells specified by the character spacing parameter will not.

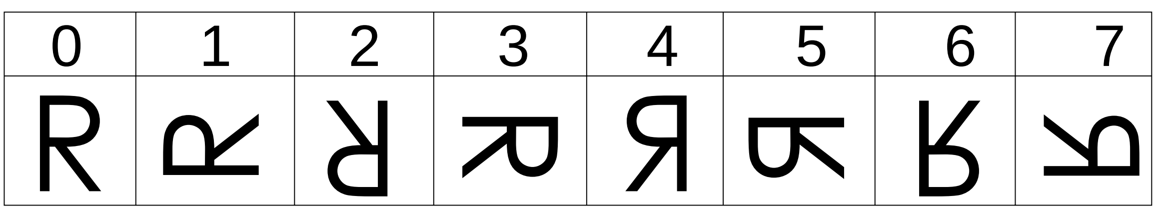
Line Spacing: (code “3")

Specifies the amount of spacing (in pixels) that will be inserted between the character cells of two successive lines of text printed with the CTEXT opcode when an ASCII line-feed character (0Ah) is encountered. GTEXT ignores control characters. “RGI0" fonts have a default line-spacing value at ∅

Text Rotation: (code “4")

“RGI0" fonts implement the following text rotation values (alternate text drivers for other font formats may implement different values):

0 = no rotation
1 = 90° counter-clockwise
2 = 180° counter-clockwise
3 = 270° counter-clockwise
4 = mirrored about center in X-dimension
5 = mirrored + 90° counter-clockwise
6 = mirrored + 180° counter-clockwise
7 = mirrored + 270° counter-clockwise



Examples of possible Text Rotations

Set Text Parameter (continued)

Text Direction: (code “5")

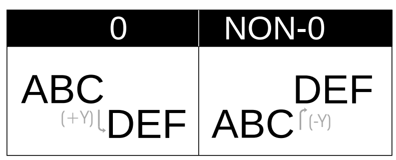
“RGI0" fonts implement the following text direction values (alternate text drivers for other font formats may implement different values):

0 = +X direction (left-to-right)
1 = -Y direction (bottom-to-top)
2 = -X direction (right-to-left)
3 = +Y direction (top-to-bottom)

Line-feed Sense (code “6")

Specifies the direction in which the current text position will be adjusted when text is being printed with the CTEXT opcode and an ASCII line-feed character (0Ah) is encountered. GTEXT ignores control characters. The adjustment is perpendicular to the current text direction. If the line-feed sense parameter is “0", the adjustment is in the negative direction along the perpendicular axis. E.g., if the current text direction is ”0" or “2" (along the X axis in either the +X or -X directions), the line-feed adjustment will be along the Y axis—in the +Y direction if line-feed sense is ”0", or in the -Y direction otherwise.

Example:



Set text rotation at 180 degrees. The code number to specify a new text rotation parameter is “4", the text rotation parameter value to specify 180 degree rotation is ”2".

Opcode: 00E8 (TEXTP)
Code: 0004
Value: 00000002

Words in memory: 00E8 0004 0002 0000

Related opcodes: R_TEXTP, FONT

Set Text Parameter (variable)

TEXTPV V*i*

Syntax: 00E9 word

Parameters: Vi = function code #

Vi+1 = parameter value

Description: Sets a new value for the text parameter corresponding to the specified function code.

Text Parameter Codes:

CODE#	FORMAT	DESCRIPTION
0	pointer	Font index/address (resets all parameters to defaults)
1	pointer	Font index/address (sets spacing parameters to defaults)
2	integer	Character spacing
3	integer	Line spacing
4	integer	Text rotation
5	integer	Text direction
6	boolean	Line-feed sense (zero: +X/+Y, nonzero: -X/-Y (reverse))

Note: See TEXTP for more function code information—Font, Character Spacing, Line Spacing, Text Rotation, Text Direction and Line-feed Sense.

Related opcodes: R_TEXT, FONT

Select Text Service Routine

TEXTSVC pointer

Syntax: 00EA long

Description: Selects a new text service routine. If the parameter is <128, it is assumed to be the index of one of the default text service routines, and the corresponding text driver is made active. Otherwise, the parameter is assumed to be the entry-point address of a text service routine.

Related opcodes: TEXTP, FONT

Discussion: Changing the text service routine should be followed by selecting a new font appropriate to the new text driver.

The default text driver is numbered “0" and supports ”RGI type 0" fonts.

Select Text Service Routine (variable)

TEXTSVCV V

Syntax: 00EB word

Parameters: V = address of text service routine (or index for default)

Description: Selects a new text service routine. If the parameter is <128, it is assumed to be the index of one of the default text service routines, and the corresponding text driver is made active. Otherwise, the parameter is assumed to be the entry-point address of a text service routine.

Related opcodes: TEXTP, FONT

Discussion: Changing the text service routine should be followed by selecting a new font appropriate to the new text driver.

The default text driver is numbered “0" and supports ”RGI type o" fonts.

Select Tile (pixel mapped) Fill Pattern

TILE pointer

Syntax: 00EC long

Description: Selects a tile (pixel mapped) fill pattern to be used when fill type “2" is specified for any of the area fill opcodes (except SEEDFILL) : CIRS, CPFILL, CPFILLR ELPS, PFILL, PFILLR, RECTS, RRECTS, SECTS, SEGS.

If the parameter is <128, it is assumed to be the index of one of the default patterns and the corresponding pattern is made active. Otherwise, the parameter is assumed to be the address of a tile-pattern definition structure.

Tile Pattern Structure:

OFFS	SIZE	FIELD
0000	16	Pattern width
0010	16	Pattern height
0020	16	Pattern depth (pixel-size)
0030	16	Pattern pitch
0040	16	Pattern ID (default patterns)
0050	16	(unused)
0060		Beginning of pattern data

The tile pattern structure is exactly the same as that generated by the COPYSR opcode. Therefore, tile patterns may easily be created from on-screen data by first copying a rectangle to a buffer with COPYSR.

Related opcodes: STIPPLE, COPYSR

Select Tile (pixel mapped) Fill Pattern, variable

TILEV V

Syntax: 00ED word

Parameters: V = address of tile pattern structure (or index of default)

Description: Selects a tile (pixel mapped) fill pattern to be used when fill type “2" is specified for any of the area fill opcodes (except SEEDFILL) : CIRS, CPFILL, CPFILLR, ELPS, PFILL, PFILLR, RECTS, RRECTS, SECTS, SEGS.

If the parameter is <128, it is assumed to be the index of one of the default patterns and the corresponding pattern is made active. Otherwise, the parameter is assumed to be the address of a tile-pattern definition structure.

Tile Pattern Structure:

OFFS	SIZE	FIELD
0000	16	Pattern width
0010	16	Pattern height
0020	16	Pattern depth (pixel-size)
0030	16	Pattern pitch
0040	16	Pattern ID (default patterns)
0050	16	(unused)
0060		Beginning of pattern data

The tile pattern structure is exactly the same as that generated by the COPYSR opcode. Therefore, tile patterns may easily be created from on-screen data by first copying a rectangle to a buffer with COPYSR.

Related opcodes: STIPPLE, COPYSR

Select Tile Fill Pattern, Explicit Parameters

TILEX width height depth pitch address

Syntax: 018C word word word word long

Description: Selects a tile (pixel mapped) fill pattern. *width* and *height* are the dimensions of the tile pattern;*depth* is the pattern pixel-size; *pitch* is the row-pitch of the pattern data (difference in bits between the beginning of successive rows of the tile pattern data); and the *address* is that of the tile pattern data. The tile pattern will be used when fill type “2" is specified for any of the area fill opcodes (except SEEDFILL) : CIRS, CPFILL, ELPS, PFILL, RECTS, RRECTS, SECTS, SEGS.

Tile Pattern Structure:

OFFS	SIZE	FIELD
0000	16	Pattern width
0010	16	Pattern height
0020	16	Pattern depth (pizel-size)
0030	16	Pattern pitch
0040	16	Pattern ID (default patterns)
0050	16	(unused)
0060		Beginning of pattern data

Related opcodes: TILE, STIPPLE, COPYSR

Discussion: TILEX is exactly the equivalent of TILE, except that the tile pattern parameters are specified explicitly rather than in a header structure prefacing the actual pattern data buffer. This facilitates the use of pattern data existing in a format where a parameter header cannot be prefaced to the pattern data—for instance, this allows the use of a portion of a larger pattern, or the use of pattern data generated in an off-screen draw-buffer (as configured by R_ENVB, INITGCB or WPAGEB), or even a portion of a displayable frame-buffer (screen page).

Select Tile Fill Pattern, Explicit Parameters, variable

TILEXV Vi

Syntax: 018D word

Parameters:
Vi = tile pattern width
Vi +1 = tile pattern height
Vi +2 = tile pattern depth (pixel size)
Vi +3 = tile pattern array pitch
Vi +4 = address of tile pattern data

Description: Selects a tile (pixel mapped) fill pattern. *width* and *height* are the dimensions of the tile pattern; depth is the pattern pixel-size; *pitch* is the row-pitch of the pattern data (difference in bits between the beginning of successive rows of the tile pattern data); and the *address* is that of the tile pattern data. The tile pattern will be used when fill type “2" is specified for any of the area fill opcodes (except SEEDFILL) : CIRS, CPFILL, ELPS, PFILL, RECTS, RRECTS, SECTS, SEGS.

Tile Pattern Structure:

OFFS	SIZE	FIELD
0000	16	Pattern width
0010	16	Pattern height
0020	16	Pattern depth (pixel-size)
0030	16	Pattern pitch
0040	16	Pattern ID (default patterns)
0050	16	(unused)
0060		Beginning of pattern data

Related opcodes: TILEX

Discussion: TILEX is exactly the equivalent of TILE, except that the tile pattern parameters are specified explicitly rather than in a header structure prefacing the actual pattern data buffer. This facilitates the use of pattern data existing in a format where a parameter header cannot be prefaced to the pattern data—for instance, this allows the use of a portion of a larger pattern, or the use of pattern data generated in an off-screen draw-buffer (as configured by R_ENVB, INITGCB or WPAGEB), or even a portion of a displayable frame-buffer (screen page).

Set Graphics Transparency Mode

TRANS flag

Syntax: 00EE word

Description: Sets the graphics transparency mode: 0 = off (default) 1 = on

When transparency mode is on, background pixels (text, patterned lines or fills) are not written.

Note that on the TMS34010 processor, transparency may not function properly for some pixel processing modes other than replace or XOR.

Related opcodes: BOOL, PMASK

Set Graphics Transparency Mode (variable)

TRANSV V

Syntax: 00EF word

Parameters: V = flag

Description: Sets the graphics transparency mode: 0 = off (default) 1 = on

When transparency mode is on, background pixels (text, patterned lines or fills) are not written.

Note that on the TMS34010 processor, transparency may not function properly for some pixel processing modes other than replace or XOR.

Related opcodes: BOOL, PMASK

Enable/Disable Text Cursor Display

TXCSRON flag

Syntax: 0172 word

Description: Turns text cursor display on/off according to the specified parameter:

0 = text cursor off 1 = text cursor on

Related Opcodes: TXCSRPAGE, TXCSRXY, TXCURSOR

Discussion: The text cursor is typically positioned with the TXCSRXY opcode, and then displayed by executing “TXCSRON 1". The TXCSRON opcode saves the screen contents under the cursor. This screen data is restored when the text cursor is turned off with ”TXCSRON 0".

Enable/Disable Text Cursor Display, variable

TXCSRONV V

Syntax: 0173 word

Description: Turns text cursor display on/off according to the specified parameter:

0 = text cursor off 1 = text cursor on

Discussion: The text cursor is typically positioned with the TXCSRXY opcode, and then displayed by executing “TXCSRON 1". The TXCSRON opcode saves the screen contents under the cursor. This screen data is restored when the text cursor is turned off with ”TXCSRON 0".

Related Opcodes: TXCSRON

Configure Text Cursor for Channel and Page

TXCSRPAGE channel page

Syntax: 0174 word word

Description: Initializes the text cursor for the specified graphics channel and page.

Related Opcodes: TXCSRON, TXCSRXY, TXCURSOR

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751 and RG-752. It initializes the cursor for the specified graphics channel and page, and thus allows the cursor to be placed on either an underlay or overlay channel. Even for those boards having only a single graphics channel, this opcode may be used to move the cursor to any of the available graphics pages. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

Configure Text Cursor for Channel and Page, variable

TXCSRPAGEV V*i*

Syntax: 0175 word

Description: Initializes the text cursor for the specified graphics channel and page.

Parameters: Vi = text cursor channel ID
Vi+1 = text cursor display page

Related Opcodes: TXCSRPAGE

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751 and RG-752. It initializes the cursor for the specified graphics channel and page, and thus allows the cursor to be placed on either an underlay or overlay channel. Even for those boards having only a single graphics channel, this opcode may be used to move the cursor to any of the available graphics pages. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

Set Text Cursor Window

TXCSRWIN X_0 Y_0 X_1 Y_1

Syntax: 0192 long

Description: Sets the text cursor clipping window boundaries (upper-left, lower-right).

Related opcodes: TXCURSOR

Set Text Cursor Window, variable

TXCSRWINV V_i

Syntax: 0193 word

Parameters:

Vi	= text cursor window Xmin (left)
Vi+1	= text cursor window Ymin (top)
Vi+2	= text cursor window Xmax (right)
Vi+3	= text cursor window Ymax (bottom)

Description: Sets the text cursor clipping window boundaries (upper-left, lower-right).

Related opcodes: TXCSRWIN

Set Text Cursor Location

TXCSRXY x y

Syntax: 0176 word word

Description: Specifies the (x,y) location of the text cursor on the screen.

Related opcodes: TXCSRON, TXCSRPAGE, TXCURSOR

Discussion: The TXCSRXY opcode changes the current x,y text cursor address. If the text cursor is already being displayed on the screen, it is repositioned to the x,y location specified.

Set Text Cursor Location, variable

TXCSRXYV V*i*

Syntax: 0177 word

Parameters: Vi = new text cursor x

Vi+1 = new text cursor y

Description: Specifies the (x,y) location of the text cursor on the screen. The x coordinate is stored in variable V*i*, and the y coordinate is stored in variable V*i*+1. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: TXCSRXY

Discussion: The TXCSRXY opcode changes the current x,y text address. If the text cursor is already being displayed on the screen, it is repositioned to the x,y location specified.

Select Text Cursor

TXCURSOR cursor color1 color2 save_addr save_pitch blink_rate

Syntax: 0170 long long long long long word

Description: Selects the current text cursor. If the cursor parameter value is <128, it is assumed to be the index of one of the default cursors, and the corresponding cursor is made active. Otherwise, the parameter is assumed to be the address of a cursor definition structure formatted as follows:

OFFS	SIZE	FORMAT	FIELD	APPLICABILITY					
0000	16	coded	cursor type (see below)	**0**	**1**	**2**	**3**	**4**	**5**
0010	16	coded	boolean operation	X	X	X	X	X	X
0020	32	[XY]	cursor size	X	X	X	X	X	X
0040	32	[XY]	cursor offset	X	X	X	X	X	X
0060	32	address	address of user-specified save buffer	X	X	X	X	X	X
0080	32	linear	pitch of user-specified save buffer	X	X	X	X	X	X
00A0	32	address	shape #1 address			X	X	X	X
00C0	32	linear	shape #1 pitch					X	X
00E0	32	address	shape #2 address				X		X
0100	32	linear	shape #2 pitch						X

<u>Cursor Types</u>

0 = filled rectangle

1 = outlined rectangle

2 = single color symbol

3 = two color symbol

4 = single color bitmap

5 = two color bitmap

For two-color pixblt cursors (types 3 and 5) shape #1 is drawn first with color #1 (background), then shape #2 is drawn with color #2 (foreground). For single-color pixblt cursors (types 2 and 4) only shape1 and color1 are used.

Pixel Processing: Any of the pixel-processing operations listed under the BOOL opcode may be used.

Cursor Size: Specifies the X and Y dimensions of the cursor rectangle. For cursor types 2 and 3, this must match the width and height parameters in the symbol header.

Cursor Offset: Specifies the signed X and Y offsets (in pixels) from the upper left corner of the cursor rectangle that identify the cursor “hot spot.” The cursor will be drawn so that the “hot spot” coincides exactly with the pixel specified as the current cursor X,Y location. E.g., for a “cross-hair” style cursor, the cursor offsets would typically be (width/2,height/2).

TXCURSOR (continued)

User-specified Save Buffer (optional): The firmware maintains a default internal save buffer for use by the on-board default cursors. The size of this default buffer accomodates the largest of default cursors (32x32). For a user defined cursor larger than 32x32 (or, a total “area” of more than 1024 pixels) a save buffer address MUST be specified. A save buffer address specified as an opcode parameter supercedes the parameter contained in the cursor definition structure. If both are NULL (0), then the default save buffer address will be used.

Pitch of User-specified Save Buffer(optional): If a save buffer address is specified, then the save buffer pitch must also be specified. The save buffer pitch (if used) is specified in bits, and must be a multiple of 16. A pitch value specified as an opcode parameter supercedes the parameter contained in the cursor definition structure. If both are NULL (0), then the default pitch value will be used.

Note: If either the save buffer address or save buffer pitch is NULL, the cursor size is truncated to accomodate the default save buffer (32x32), and the default save buffer pitch is used.

Shape Address (cursor types 2, 3, 4, 5): For cursor types 2 and 3 (symbol), the address is that of a symbol structure of the type used by the SSYM opcode. For cursor types 4 and 5, the address is that of a pixel array (bitmap) of the same dimensions as specified by the cursor size, of the specified pitch (see below).

Shape Pitch (cursor types 4, 5): Pitch must be specified for any of the bitmap cursor types, and must be a power of 2 (16, 32,...).

Related opcodes: TXCSRON, TXCSRPAGE, TXCSRXY, MSCURSOR, USCURSOR

Discussion: The mouse cursor, text cursor and user cursor (selected with the MSCURSOR, TXCURSOR and USCURSOR opcodes, respectively) all use the same cursor definition structure and may likewise use any of the default cursors.

The mouse cursor and text cursor both have default shapes at power-up, but the user cursor does not. A user cursor MUST be defined (or selected from the default cursors) before it can be used.

The mouse cursor and text cursor are both global resources (i.e., there is only one of each). The user cursor may be considered a local resource in that there may be one user cursor per environment (graphics context), and therefore there may be as many user cursors in use as there are environments.

The mouse cursor and text cursor each have a defaut save buffer that is used for the default cursors. The default save buffers will accomodate a cursor size of up to 32x32 pixels (or a total “area” of 1024 pixels) and may be used for a user-defined cursor by specifying a NULL (0) save buffer address. There is no default user cursor save buffer—a save buffer address MUST be specified when selecting or defining a user cursor.

The mouse cursor and text cursor may both be configured for automatic save/restore handling by the graphics primitives (both default to this mode at power-up). When in “auto-handling” mode all graphics primitives (circles, lines, text, etc.) will automatically remove and restore the cursors as necessary—thus the user is not required to manage the state of either the mouse or text cursor. If preferred, “auto-handling” mode may be disabled by using the MSREG opcode to disable mouse cursor auto-handling, or by modifying the **txcsr_mode** field in Global RAM (see appendix A) to disable text cursor auto-handling. The user cursor is NOT handled by the graphics primitives—the user is responsible for managing the state of the user cursor when using any of the graphics primitives.

TXCURSOR (continued)

The mouse cursor may be configured to track the current mouse position when the mouse is enabled (MSMODE≠0). When “mouse-tracking” is enabled, the mouse cursor position may also be changed with the MSCSRXY opcode, but will thereafter track susequent mouse movement inputs. “Mouse-tracking” is enabled as the default mode at power-up, but may be disabled with the MSREG opcode. The mouse cursor position may also be changed “manually” with MSCSRXY when mouse tracking mode is disabled, mouse input is disabled (MSMODE=0), or the mouse is disconnected (a mouse does not need to be connected to the serial port in order to use the mouse cursor). The text cursor and user cursor positions may only be changed with the TXCSRXY and USCSRXY opcodes, respectively.

The mouse cursor has the highest priority in that it will always appear to be “in front of” the text cursor and any user cursors. The text cursor has the next highest priority and will appear to be in front of any user cursors. The user cursor has a lower priority than the mouse or text cursors, but will appear to be in front of any background graphics. The user must manage the relative priorities of overlapping user cursors if more than one is active at a time.

The text cursor may be configured to blink independently of the “blinking palette” function (BLINK opcode) by specifying a non-zero value for the blink-rate parameter of the TXCURSOR opcode. The text cursor will then be removed and restored at regular intervals according to the blink rate specified, and information “behind” the text cursor will become unobscured during the intervals when the text cursor is removed. The blink function of the text cursor will thus operate even on boards that do not make use of a RAMDAC device (such as the RG-752). Any of the cursors may be made to “blink” by specifying colors that have been configured to be “blinking” colors with the BLINK opcode, although information behind the cursor will continue to be obscured.

MOUSE CURSOR	TEXT CURSOR	USER CURSOR
common definition structure	common definition structure	common definition structure
default shape at power-up	default shape at power-up	no default shape (user must define)
global resource (1 only)	global resource (1 only)	1 per environment
default save buffer (32x32)	default save buffer (32x32)	no default save buffer (user must define)
save/restore handled by graphics primitives (may be disabled)	save/restore handled by graphics primitives (may be disabled)	user must manage cursor state
tracks mouse movement (may be disabled and moved with MSCSRXY)	move with TXCSRXY	move with USCSRXY
highest priority	next highest priority	lowest priority
	may be configured to blink	

Select Text Cursor, variable

TXCURSORV V*i*

Syntax: 0171 word

Parameters:

Vi	= address of cursor structure (or index of default)
Vi+1	= shape #1 color
Vi+2	= shape #2 color
Vi+3	= save buffer address
Vi+4	= save buffer pitch
Vi+5	= blink rate

Related Opcodes: TXCURSOR

Description: Selects the current text cursor. If the cursor parameter value is <128, it is assumed to be the index of one of the default cursors, and the corresponding cursor is made active. Otherwise, the parameter is assumed to be the address of a cursor definition structure (see TXCURSOR).

Set User Cursor State on/off

USCSRON flag

Syntax: 00F4 word

Description: Set user cursor on/off. 0 = off 1 = on. The cursor is displayed at the x,y location specified by USCSRXY.

Related opcodes: USCSRXY, USCURSOR

Set User Cursor State on/off, variable

USCSRONV V

Syntax: 00F5 word

Parameters: V = state

Description: Set user cursor on/off. 0 = off 1 = on. The cursor is displayed at the x,y location specified by USCSRXY.

Related opcodes: USCSRON

Set Current User Cursor Location

USCSRXY X Y

Syntax: 00F6 word word

Description: Set the current user cursor position to the specified x,y location

Related opcodes: USCSRON, USCURSOR

Set Current User Cursor Location, variable

USCSRXYV V*i*

Syntax: 00F7 word

Parameters: Vi = X coordinate

Vi+1 = Y coordinate

Description: Set the current user cursor position to the specified x,y location. Vi is the first of 2 consecutive variables containing the parameters.

Related opcodes: USCSRXY

Set User Cursor Parameters

USCURSOR cursor color1 color1 save_addr save_pitch

Syntax: 016E long long long long long

Description: Selects the current user cursor. If the cursor parameter value is <128, it is assumed to be the index of one of the default cursors, and the corresponding cursor is made active. Otherwise, the parameter is assumed to be the address of a cursor definition structure formatted as follows:

OFFS	SIZE	FORMAT	FIELD	APPLICABILITY					
0000	16	coded	cursor type (see below)	**0**	**1**	**2**	**3**	**4**	**5**
0010	16	coded	boolean operation	X	X	X	X	X	X
0020	32	[XY]	cursor size	X	X	X	X	X	X
0040	32	[XY]	cursor offset	X	X	X	X	X	X
0060	32	address	address of user-specified save buffer	X	X	X	X	X	X
0080	32	linear	pitch of user-specified save buffer	X	X	X	X	X	X
00A0	32	address	shape #1 address			X	X	X	X
00C0	32	linear	shape #1 pitch					X	X
00E0	32	address	shape #2 address				X		X
0100	32	linear	shape #2 pitch						X

Cursor Types

0 = filled rectangle

1 = outlined rectangle

2 = single color symbol

3 = two color symbol

4 = single color bitmap

5 = two color bitmap

For two-color pixblt cursors (types 3 and 5) shape #1 is drawn first with color #1 (background), then shape #2 is drawn with color #2 (foreground). For single-color pixblt cursors (types 2 and 4) only shape1 and color1 are used.

Pixel Processing: Any of the pixel-processing operations listed under the BOOL opcode may be used.

Cursor Size: Specifies the X and Y dimensions of the cursor rectangle. For cursor types 2 and 3, this must match the width and height parameters in the symbol header.

Cursor Offset: Specifies the signed X and Y offsets (in pixels) from the upper left corner of the cursor rectangle that identify the cursor “hot spot.” The cursor will be drawn so that the “hot spot” coincides exactly with the pixel specified as the current cursor X,Y location. E.g., for a “cross-hair” style cursor, the cursor offsets would typically be (width/2,height/2).

USCURSOR (continued)

Save BufferAddress: There is no default save buffer for user cursors, therefore a save buffer address MUST be specified. A save buffer address specified as an opcode parameter supercedes the parameter contained in the cursor definition structure.

Save BufferPitch (optional): The save buffer pitch may be specified, otherwise NULL (0) will cause a default pitch to be calculated from the cursor width parameter. The save buffer pitch is specified in bits, and must be a multiple of 16. A pitch value specified as an opcode parameter supercedes the parameter contained in the cursor definition structure. If both are NULL (0), then the default pitch value will be used.

Shape Address (cursor types 2, 3, 4, 5): For cursor types 2 and 3 (symbol), the address is that of a symbol structure of the type used by the SSYM opcode. For cursor types 4 and 5, the address is that of a pixel array (bitmap) of the same dimensions as specified by the cursor size, of the specified pitch (see below).

Shape Pitch (cursor types 4, 5): Pitch must be specified for any of the bitmap cursor types, and must be a power of 2 (16, 32,...).

Related opcodes: USCSRON, USCSRXY, MSCURSOR, TXURSOR

Discussion: The mouse cursor, text cursor and user cursor (selected with the MSCURSOR, TXCURSOR and USCURSOR opcodes, respectively) all use the same cursor definition structure and may likewise use any of the default cursors.

The mouse cursor and text cursor both have default shapes at power-up, but the user cursor does not. A user cursor MUST be defined (or selected from the default cursors) before it can be used.

The mouse cursor and text cursor are both global resources (i.e., there is only one of each). The user cursor may be considered a local resource in that there may be one user cursor per environment (graphics context), and therefore there may be as many user cursors in use as there are environments.

The mouse cursor and text cursor each have a defaut save buffer that is used for the default cursors. The default save buffers will accomodate a cursor size of up to 32x32 pixels (or a total “area” of 1024 pixels) and may be used for a user-defined cursor by specifying a NULL (0) save buffer address. There is no default user cursor save buffer—a save buffer address MUST be specified when selecting or defining a user cursor.

The mouse cursor and text cursor may both be configured for automatic save/restore handling by the graphics primitives (both default to this mode at power-up). When in “auto-handling” mode all graphics primitives (circles, lines, text, etc.) will automatically remove and restore the cursors as necessary—thus the user is not required to manage the state of either the mouse or text cursor. If preferred, “auto-handling” mode may be disabled by using the MSREG opcode to disable mouse cursor auto-handling, or by modifying the **txcsr_mode** field in Global RAM (see appendix A) to disable text cursor auto-handling. The user cursor is NOT handled by the graphics primitives—the user is responsible for managing the state of the user cursor when using any of the graphics primitives.

USCURSOR (continued)

The mouse cursor may be configured to track the current mouse position when the mouse is enabled (MSMODE≠0). When “mouse-tracking” is enabled, the mouse cursor position may also be changed with the MSCSRXY opcode, but will thereafter track susequent mouse movement inputs. “Mouse-tracking” is enabled as the default mode at power-up, but may be disabled with the MSREG opcode. The mouse cursor position may also be changed “manually” with MSCSRXY when mouse tracking mode is disabled, mouse input is disabled (MSMODE=0), or the mouse is disconnected (a mouse does not need to be connected to the serial port in order to use the mouse cursor). The text cursor and user cursor positions may only be changed with the TXCSRXY and USCSRXY opcodes, respectively.

The mouse cursor has the highest priority in that it will always appear to be “in front of” the text cursor and any user cursors. The text cursor has the next highest priority and will appear to be in front of any user cursors. The user cursor has a lower priority than the mouse or text cursors, but will appear to be in front of any background graphics. The user must manage the relative priorities of overlapping user cursors if more than one is active at a time.

The text cursor may be configured to blink independently of the “blinking palette” function (BLINK opcode) by specifying a non-zero value for the blink-rate parameter of the TXCURSOR opcode. The text cursor will then be removed and restored at regular intervals according to the blink rate specified, and information “behind” the text cursor will become unobscured during the intervals when the text cursor is removed. The blink function of the text cursor will thus operate even on boards that do not make use of a RAMDAC device (such as the RG-752). Any of the cursors may be made to “blink” by specifying colors that have been configured to be “blinking” colors with the BLINK opcode, although information behind the cursor will continue to be obscured.

MOUSE CURSOR	TEXT CURSOR	USER CURSOR
common definition structure	common definition structure	common definition structure
default shape at power-up	default shape at power-up	no default shape (user must define)
global resource (1 only)	global resource (1 only)	1 per environment
default save buffer (32x32)	default save buffer (32x32)	no default save buffer (user must define)
save/restore handled by graphics primitives (may be disabled)	save/restore handled by graphics primitives (may be disabled)	user must manage cursor state
tracks mouse movement (may be disabled and moved with MSCSRXY)	move with TXCSRXY	move with USCSRXY
highest priority	next highest priority	lowest priority
	may be configured to blink	

Set User Cursor Parameters (variable)

USCURSORV V*i*

Syntax:	016F	word
Parameters:	Vi	= address of cursor parameter structure
	Vi+1	= shape #1 color
	Vi+2	= shape #2 color
	Vi+3	= save buffer address
	Vi+4	= save buffer pitch

Description: Selects the current user cursor. If the cursor parameter value is <128, it is assumed to be the index of one of the default cursors, and the corresponding cursor is made active. Otherwise, the parameter is assumed to be the address of a cursor definition structure (see USCURSOR).

Related opcodes: USCURSOR

Wait for Vertical Blank

VWAIT

Syntax: 00F8

Description: This opcode will loop and wait until the beginning of the next vertical blanking period.

Related opcodes: None

Discussion: This opcode is useful to update video RAM or the color look-up table during the vertical blank time.

Initialize Drawing Parameters for Draw Buffer

WPAGEB addr_DB addr_DB_params

Syntax: 017E long long

Description: Configures the drawing parameters of the current graphics context according to the draw buffer parameters specified. *addr_DB* is the linear address which identifies the upper-left corner (x=0, y=0) of draw-buffer memory; *addr_DB_params* is the address of a draw-buffer parameter structure formatted as follows:

OFFS	SIZE	FIELD
0000	16	draw-buffer width in pixels
0010	16	draw-buffer height in pixels
0020	16	draw-buffer depth (pixel size)
0030	16	draw-buffer pitch

Related opcodes: INITGCB

Discussion: The draw-buffer pitch must be a multiple of 16, and is rounded up if necessary when initializing the corresponding graphics context field. Clipping is set ON (clipmode = 2) with the clipping window set to the draw-buffer width and height specified. If clipmode is subsequently set to 0 (“off”), the clipping window width reverts to the maximum value corresponding to the specified pitch.

Initialize Drawing Parameters for Draw Buffer, variable

WPAGEBV V*i*

Syntax: 017F word

Parameters: Vi = address of draw buffer

Vi+1 = address of draw buffer parameter structure

Description: Configures the drawing parameters of the current graphics context according to the draw buffer parameters specified. addr_DB is the linear address which identifies the upper-left corner (x=0, y=0) of draw-buffer memory; addr_DB_params is the address of a draw-buffer parameter structure formatted as follows:

OFFS	SIZE	FIELD
0000	16	draw-buffer width in pixels
0010	16	draw-buffer height in pixels
0020	16	draw-buffer depth (pixel size)
0030	16	draw-buffer pitch

Related opcodes: WPAGEB

Discussion: The draw-buffer pitch must be a multiple of 16, and is rounded up if necessary when initializing the corresponding graphics context field. Clipping is set ON (clipmode = 2) with the clipping window set to the draw-buffer width and height specified. If clipmode is subsequently set to 0 (“off”), the clipping window width reverts to the maximum value corresponding to the specified pitch.

Initialize Drawing Parameters for Channel and Page

WPAGEC channel page

Syntax: 0180 word word

Description: Configures the drawing parameters of the current graphics context for the specified channel and page.

Related opcodes: INITGCC

Discussion: This opcode supports those boards having more than one graphics channel, such as the RG-751, RG-752 and RG-753. See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

For boards that only support a single channel, specify a channel ID of “0" (default underlay).

Initialize Drawing Parameters for Channel and Page, variable

WPAGECV V*i*

Syntax: 0181 word

Parameters: Vi = channel ID

Vi+1 = page #

Description: Configures the drawing parameters of the current graphics context for the specified channel and page.

Related opcodes: WPAGEC

Write to Specified Page

WPG page

Syntax: 00F9 word

Description: Directs new data to be written to the specified page in the current channel.

Related opcodes: WPGA, WPAGEC, DPG

Discussion: The graphics board can be configured with one, two, or three pages of video RAM, corresponding to page 0, page 1, and page 2. When two pages are present, the user may wish to display the contents of one page while updating the other page with data. More than one display buffer is desirable for display formats that are continually updated. Switching between two buffers produces the best image for the viewer, as the image is always complete when viewed.

Write to User-defined Page

WPGA address

Syntax: 00FA long

Description: Specifies that new data is to be written to the video RAM page beginning at the address specified.

Related opcodes: WPG, WPGV, DPG, DPGA, DPGV

Discussion: This opcode allows the user to select an arbitrary page of video RAM for update, beginning at the address specified. It should be used with DPGA, which allows the user to display the user selected page on the video monitor. For most applications, WPG or WPGV would be used.

Write to Specified Page (variable)

WPGV V

Syntax: 00FB word

Description: Directs new data to be written to the page specified in the variable (assumes the current channel).

Related opcodes: WPG

Discussion: The graphics board can be configured with one, two, or three pages of video RAM, corresponding to page 0, page 1, and page 2. When two pages are present, the user may wish to display the contents of one page while updating the other page with data. More than one display buffer is desirable for display formats that are continually updated. Switching between two buffers produces the best image for the viewer, as the image is always complete when viewed.

Exchange AFGIS Program Counter with Variable

XCHGPC V

Syntax: 00FC word

Description: Exchanges the contents of the AFGIS program counter with the contents of variable V.

Related opcodes: LDPCV

Exchange AFGIS Stack Pointer with Variable

XCHGSP V

Syntax: 00FD word

Description: Exchanges the contents of the AFGIS stack pointer with the contents of variable V.

Related opcodes: POPV, POPVARS, PUSHV, PUSHVARS, XCHGPC

XOR Immediate with Variable

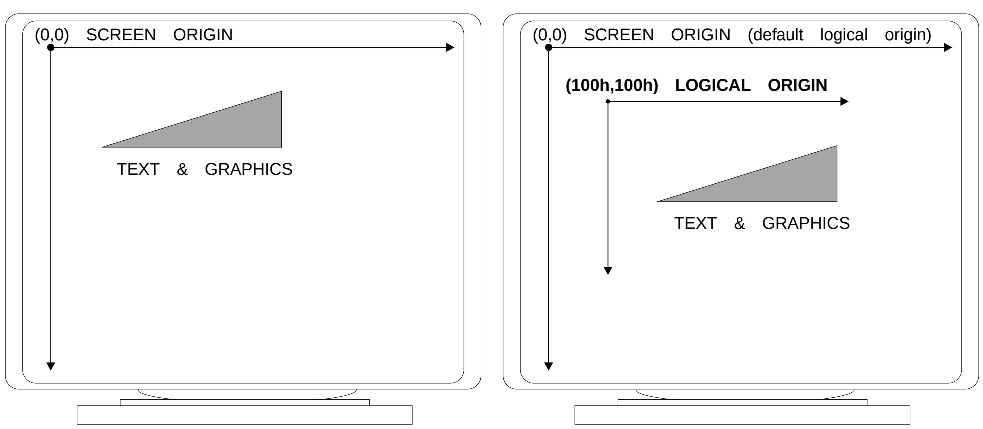
XORIV long Vd

Syntax: 00FE long word

Description: The value specified is XORed with variable V_d.

Related opcodes: XORVV, JUMPA, JUMPR

Flags Affected: Z (other flags undefined)



TRIANGLE AND TEXT LOCATED RELATIVE TO THE DEFAULT LOGICAL ORIGIN

TRIANGLE & TEXT LOCATED RELATIVE TO NEW LOGICAL ORIGIN

XOR Variables

XORVV V_s V_d

Syntax: 00FF word word

Description: The variable V_s is XORed with V_d and the result is stored in V_d.

Related opcodes: XORIV, JUMPA, JUMPR

Flags Affected: Z (other flags undefined)

Discussion: All 32 bits of the destination variable are affected.

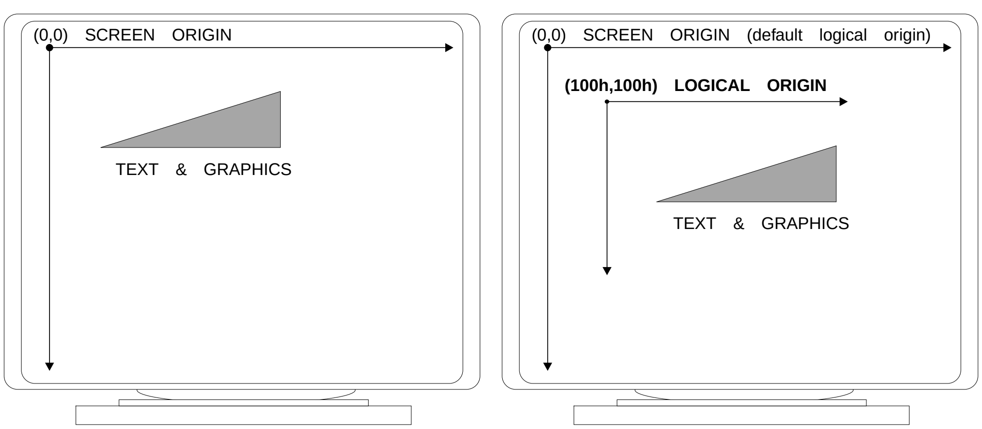
Set Logical Origin

XYORG x y

Syntax: 0100 word word

Description: Sets the logical origin to the (x,y) screen coordinate specified. The position of the logical origin is specified in pixels, relative to the screen origin. All text and graphics are drawn relative to the logical origin. If the logical origin is moved, all text and graphics are drawn relative to the new logical origin location (see diagrams below). The default location of the logical origin is the screen origin (0,0).

Related opcodes: XYORGV



TRIANGLE AND TEXT LOCATED RELATIVE TO THE DEFAULT LOGICAL ORIGIN

TRIANGLE & TEXT LOCATED RELATIVE TO NEW LOGICAL ORIGIN

Logical Origin (variable)

XYORGV V_i

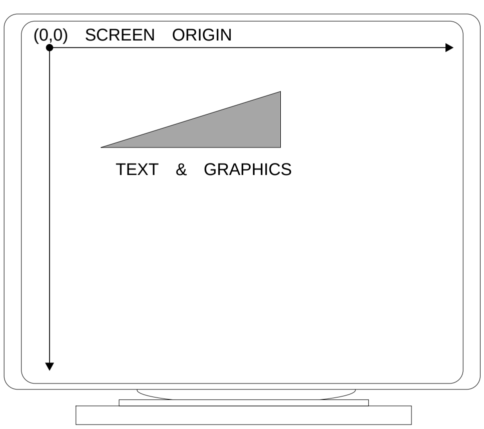
Syntax: 0101 word

Parameters: Vi = X coordinate

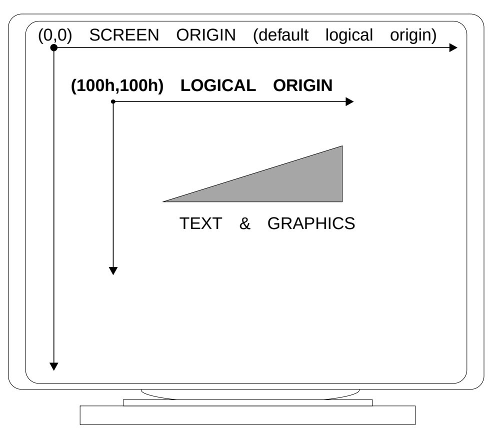
Vi+1 = Y coordinate

Description: Sets the logical origin to the (x,y) screen coordinate specified. The position of the logical origin is specified in pixels, relative to the screen origin. All text and graphics are drawn relative to the logical origin. If the logical origin is moved, all text and graphics are drawn relative to the new logical origin location (see diagrams below). The default location of the logical origin is the screen origin (0,0).

Related opcodes: XYORG



TRIANGLE AND TEXT LOCATED RELATIVE TO THE DEFAULT LOGICAL ORIGIN



TRIANGLE & TEXT LOCATED RELATIVE TO NEW LOGICAL ORIGIN

Set Display Zoom Factor (Enlarge Display)

ZOOM factor

Syntax: 011E word

Description: Specifies the display enlargement factor.

Related opcodes: ZOOMV

Discussion: This function is only effective on boards that support a hardware zoom feature. Refer to the Hardware Manual for the particular graphics board to determine if hardware zoom is supported, and what enlargement factors are implemented.

Set Display Zoom Factor (Enlarge Display), variable

ZOOMV V*i*

Syntax: 011F word

Description: Specifies the display enlargement factor.

Parameters: Vi+0 = zoom factor (hardware display multiplier - 1x, 2x, 4x, ...)

Related opcodes: ZOOM

Discussion: This function is only effective on boards that support a hardware zoom feature. Refer to the Hardware Manual for the particular graphics board to determine if hardware zoom is supported, and what enlargement factors are implemented.

AFGIS-3.11 Appendix A

This page intentionally blank.

APPENDIX A

This Appendix describes AFGIS firmware DRAM organization for Versions 3.0 and up. AFGIS DRAM is divided into four major memory segments, Fixed RAM, Global Host RAM, Graphics Environment RAM, and AFGIS local RAM.

Overview

Fixed RAM contains fields that are used to interface between the host and the graphics board. Fixed RAM fields include flags used to handshake data transactions between the host and graphics board; parameter fields for display list execution; and other address and status fields that may be queried by the host to find out more information about the graphics board. The base address of this Fixed RAM area is constant ("fixed") for a given graphics board model, but may be different for other graphics board models. In general, Fixed RAM starts at 03000000h for TMS34010 based graphics boards and at 10000000h for TMS34020 based graphics boards, but there are exceptions to this rule. Refer to the hardware manual for the particular graphics board to determine the actual base address of Fixed RAM.

At power up, AFGIS firmware initializes Fixed RAM, Global Host RAM, a default Graphics Environment RAM, and AFGIS local RAM.

AFGIS 3.0 supports multi-tasking by providing a pointer based Graphics Environment for each task. A new Graphics Environment is created with the R_ENVB or R_ENVC opcodes. The R_ENVB and R_ENVC opcodes initialize the new environment with default parameters and specify the length of several fields within the environment.

Graphics Environments are switched by the host (typically by the driver) by writing the address of the current Graphics Environment to Fixed RAM location env_ptr before each task is run.

See the AFGIS programing manual for additional information.

Fixed RAM

Fixed RAM provides memory locations at fixed locations for passing parameters between the host and the RGI graphics board. Fixed RAM starts at 03000000h for TMS34010 based graphics boards and at 10000000h for TMS34020 based graphics boards.

Fixed RAM locations, identified with variable names, hold 16 bit or 32 bit values that may have restricted host access. R means the host may only read the variable. R/W means the host may read from or write to the variable.

All other host-accessible DRAM is referenced through either the global pointer table or the Graphics Environment. The address of the global pointer table is located in Fixed RAM at gptable_ptr and the address of the current Graphics Environment is located in Fixed RAM at env_ptr.

Fixed RAM variables are shown below at TMS34010 addresses.

ADDRESS	NAME	SIZE	ACCESS	DESCRIPTION
03000000h	EODLFLAG	16	R/W	= 0 when the graphics board is busy. = 1 when the graphics board is not busy.
03000010h	KBDFLAG	16	R/W	= 0 when there is no keyboard data. = 1 when keyboard data is available
03000020h	MSEFLAG	16	R/W	= 0 when there is no mouse/serial data. = 1 when mouse/serial data is available.
03000030h	ERRFLAG	16	R/W	= 0 when no errors have been detected. = 1 when an error has been detected.
03000040h	IDLEFLAG	16	R/W	= 1 on each pass of the idle loop, approximately every 10 usecs. Not cleared by AFGIS firmware.
03000050h	DI_COUNT	16	R	60hz continuous counter, updated by AFGIS.
03000060h	INTOUTMASK	16	R/W	Graphics bd. to host interrupt enable mask.
03000070h	HOST_FIELD0	16	R/W	Reserved for host use.
03000080h	HOST_FIELD1	32	R/W	Reserved for host use.
030000A0h	ENV_PTR	32	R/W	Address of current graphics environment.
030000C0h	HINT0_AFG_ENTRY	32	R/W	AFGIS display list address. Use with HINT0.
030000E0h	HINT1_TMS_ENTRY	32	R/W	TMS assembly code address. Use with HINT1.
03000100h	GPTABLE_PTR	32	R	Address of global pointer table.
03000120h	DEFAULT_ENV_PTR	32	R	Address of default environment.
03000140h	DPAGEADDR	32	R	Current display page base address.
03000160h	DPAGE	16	R	Current display page number (0,1,...)
03000170h	ZOOM	16	R	Current display page zoom factor
03000180h	PAN	32	R	Current display page pan location [XY]

EODLFLAG, located at 03000000h/10000000h, SIZE=16, ACCESS=R/W EODLFLAG is set to one when the EODL opcode is processed, indicating that display list processing has terminated. If the display list terminates due to an error, ERRFLAG and EODLFLAG will both be set. In polling mode, EODLFLAG is polled to test for display list completion, and should be cleared before initiating execution of the next display list. EODLFLAG should be polled at a low duty cycle, typically 10%.

KBDFLAG, located at 03000010h/10000010h, SIZE=16, ACCESS=R/W KBDFLAG is set to one (in polled mode only) when data is available from the keyboard interface. Keyboard data is accessed through the keyboard_data_ptr pointer in the global pointer table. KBDFLAG is polled to test for keyboard data and should be cleared after reading the keyboard data. KBDFLAG should be polled at a low duty cycle, typically 10%.

MSEFLAG/SERFLAG, located at 03000020h/10000020h, SIZE=16, ACCESS=R/W. MSEFLAG/SERFLAG has a dual function and is used for polling mode only. If the serial interface is configured for a serial mouse, the variable is designated as MSEFLAG and is set when mouse data is available. The mouse data is accessed through the mouse_data_ptr pointer in the global pointer table. MSEFLAG is polled to test for available mouse data, and should be cleared after reading the mouse data.
If the serial interface is configured as an RS-232 port, the variable is designated as SERFLAG and is set when serial data is available. The serial data is accessed through the serial_data_ptr pointer in the global pointer table. In polling mode, SERFLAG is polled to test for the availability of serial data, and should be cleared after reading the serial data. MSEFLAG/SERFLAG should be polled at a low duty cycle, typically 10%.

ERRFLAG, located at 03000030h/10000030h, SIZE=16, ACCESS=R/W. ERRFLAG is set when an error condition is detected (e.g., an illegal AFGIS opcode). Error information is accessed through the error_buffer_ptr pointer in the global pointer table. In polling mode, ERRFLAG should be cleared before executing the next display list.

IDLEFLAG, located at 03000040h/10000040h, SIZE=16, ACCESS=R/W. IDLEFLAG is set when the graphics processor is in the idle loop. The IDLEFLAG is set on each pass of the loop, typically every 10 usecs. The IDLEFLAG can be used to determine if the graphics board is present and functioning properly by clearing the IDLEFLAG and then testing for it to be set again. The test time should be several milliseconds, as the graphics processor breaks out of the idle loop every 16 milliseconds to service the on-board display interrupt and may be busy for a few milliseconds.

DI_COUNT, located at 03000050h/10000050h, SIZE=16, ACCESS=R. DI_COUNT is incremented at field rates and serves as a free running counter. In most configurations it is incremented at a 60 Hz rate, although this may vary according to the video timing.

INTOUTMASK, located at 03000060h/10000060h, SIZE=16, ACCESS=R/W. INTOUTMASK controls the generation of outbound interrupts from the graphics board to the host over the bus interface. Each bit position in INTOUTMASK corresponds to one of the outbound interrupts as shown below. A bit set to one in INTOUTMASK enables the corresponding interrupt. For EODL processing, interrupt or polled mode is specified by the value of D0 in INTOUTMASK. For mouse/serial or keyboard operation, a one enables the interrupt of the corresponding function if it is in the interrupt mode (specified with an AFGIS opcode). Data generated by keyboard/mouse/serial/error functions can be accessed through the indicated pointer in the global pointer table.

BIT	INTERRUPT	DATA POINTER
0	EODL (display list terminated)	n/a
1	keyboard data ready	keyboard_data_ptr
2	mouse/serial data ready	mouse_data_ptr/serial_data_ptr
3	error detected	error_buffer_ptr
4	60Hz	

HOST_FIELD0, located at 03000070h/10000070h, SIZE=16, ACCESS=R/W. HOST_FIELD0 is a 16-bit variable reserved for use by the host (typically used by the driver). It is not updated by AFGIS firmware. A typical use might be as a flag to control access to the graphics board among several host-resident tasks.

HOST_FIELD1, located at 03000080h/10000080h, SIZE=32, ACCESS=R/W. HOST_FIELD1 is a 32-bit variable reserved for use by the host (typically used by the driver). It is not updated by AFGIS firmware. A typical use might be to store host-specific information (task ID, path, etc.) relevant to the currently executing display list.

ENV_PTR, located at 030000A0h/100000A0h, SIZE=16, ACCESS=R/W. ENV_PTR holds the address of the current graphics environment, which provides the graphics parameters such as colors, fill patterns, etc., for the currently executing display list. ENV_PTR would typically be updated by the host (together with the hint0_afg_entry field) before issuing a HINT0 command to begin display list execution. Likewise, ENV_PTR and hint1_tms_entry would be updated by the host (typically by the driver) before issuing a HINT1 command to begin TMS340x0 code execution.

HINT0_AFG_ENTRY, located at 030000C0h/100000C0h, SIZE=32, ACCESS=R/W. HINT0_AFG_ENTRY contains the address of the display list to be executed when the host issues a HINT0 command to the graphics board over the bus interface. Graphics parameters are taken from the graphic environment pointed to by the address contained in env_ptr.

HINT1_TMS_ENTRY, located at 030000E0h/100000E0h, SIZE=32, ACCESS=R/W. HINT1_TMS_ENTRY contains the entry-point address of TMS340x0 assembly code to be executed when the host issues a HINT1 command to the graphics board over the bus interface. Graphics parameters are taken from the graphics environment pointed to by the address contained in env_ptr.

GPTABLE_PTR, located at 03000100h/10000100h, SIZE=32, ACCESS=R. GPTABLE_PTR contains the address of the global pointer table. The global pointer table is located in Global Host Interface RAM, and contains pointers to all host accessible RAM locations in Global Host RAM.

DEFAULT_ENV_PTR, located at 03000120h/10000120h, SIZE=32, ACCESS=R. DEFAULT_ENV_PTR contains the address of the default Graphics Environment. This environment is configured and initialized at reset and would typically be used as the sole environment in single user/task applications, or by a multi-tasking application to create additional graphics environments for each of its tasks.

DPAGEADDR, located at 03000140h/10000140h, SIZE=32, ACCESS=R. DPAGEADDR contains the linear address of the current graphics display page. Each environment may implement a different write-page address which specifies where graphics are generated, but only one portion of video memory may be displayed. DPAGEADDR cannot be written to directly, and may only be modified by executing DPG, DPGA, or DPGV AFGIS opcodes.

DPAGE, located at 03000160h/10000160h, SIZE-16, ACCESS=R. DPAGE contains the page number (0,1,...) of the current graphics display page. DPAGE cannot be written to directly, and may only be modified by executing DPG, DPGA, or DPGV AFGIS opcodes.

ZOOM, located at 03000170h/10000170h, SIZE=16, ACCESS=R. ZOOM contains the current display page zoom factor (0=1x, 2=2x, 4=4x), and is only relevant to graphics boards that support a hardware zoom function, such as the RG-751. ZOOM cannot be written to directly, and may only be modified by executing the ZOOM or ZOOMV AFGIS opcodes.

PAN, located at 03000180h/10000180h, SIZE=32, ACCESS=R. PAN contains the current display page pan location in XY format (low-order word contains the X value, high-order word contains the Y value). PAN cannot be written to directly, and may only be modified by executing the AFGIS "pan" opcodes (PANX, PANXV, PANXR, PANY, PANYV, PANYR, PANYRV, PANXY, PANXYV, PANXYR, PANXYRV).

Global Host Interface RAM

Global Host Interface RAM includes the global pointer table and all data structures that are global to all tasks and graphics environments. The gptable_ptr variable in Fixed RAM contains the address of the global pointer table, which in turn contains the addresses of all other host-accessible data structures. All data structures are thus pointer-based, which allows for a consistent interface between TMS34010 and TMS34020 based graphics boards as only the base address of Fixed RAM must be known for a particular graphics board.

Global Pointer Table

The global pointer table, shown below, contains the addresses of the individual data structures in Global Host Interface RAM. It is an open-ended table whose size is dependent on the firmware version. The first entry is a count of the number of pointers in the table, and allows for a compatibility check between the application code and AFGIS firmware.

OFFSET	SIZE	NAME	ACCESS	DESCRIPTION
0000	32	gptable_nEntries	R	number of entries in gptable
0020	32	keyboard_data_ptr	R	keyboard data buffer
0040	32	mouse_data_ptr	R	mouse data buffer
0060	32	serial_data_ptr	R	serial port data buffer
0080	32	mouse_parameters_ptr	R	mouse parameters
00A0	32		R	(reserved)
00C0	32	env_descriptor_ptr	R	environment header descriptor
00E0	32	afgis_exec_stat_ptr	R	AFGIS execution status buffer
0100	32	error_buffer_ptr	R	error parameter buffer
0120	32	version_params_ptr	R	version parameters
0140	32	screen_params_ptr	R	screen parameters
0160	32	hwconfig_params_ptr	R	hardware configuration parameters
0180	32	memory_config_ptr	R	memory configuration parameters
01A0	32	default_HB0_ptr	R	default host buffer 0
01C0	32	default_HB1_ptr	R	default host buffer 1
01E0	32		R	(reserved)
0200	32	pu_parameters_ptr	R	power-up configuration parameters
0220	32		R	(reserved)
0240	32	sys_mem_alloc_ptr	R	system memory allocation parameters
0260	32	default_palette_ptr	R/W	default palettes
0280	32	default_ssymbol_ptr	R/W	default pixblt symbols (ssymbols)
02A0	32	default_dashpatn_ptr	R/W	default dashed-line patterns
02C0	32	default_stipple_ptr	R/W	default stipple (binary) fill patterns
02E0	32	default_tile_ptr	R/W	default tile (pixel mapped) fill patterns
0300	32	default_font_ptr	R/W	default fonts
0320	32	default_textsvc_ptr	R/W	default text drivers
0340	32	default_cursor_ptr	R/W	default cursors
0360	32	default_shift_ptr	R/W	default shift areas
0380	32	txcursor_parameters_ptr	R	text cursor parameters
03A0	32	bus_struct_ptr	R	bus specific parameter

Each of the data structures whose addresses are contained in the Global Pointer Table are described in the following pages.

Keyboard Data Buffer, (read only, reference: keyboard_data_ptr)

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	kbdata0	keyboard character
0010	16	kbdata1	keyboard character

0020	16	kbdstatus	keyboard status

Mouse Data Buffer, (read only, reference: mouse_data_ptr)

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	msedata_X	mouse report X position (interrupt/polled mode)
0010	16	msedata_Y	mouse report Y position
0020	16	msedata_sw	mouse report switch status
0030	16	msedata_time	mouse report data time stamp
0040	16	mseposition_X	current mouse X position (continuous update)
0050	16	mseposition_Y	current mouse Y position (continuous update)
0060	16	msecursor_on	mouse cursor state (0=off, 1=on)

Serial Port Data Buffer, (read only, reference: serial_data_ptr)

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	serialdata	serial character

Mouse Parameters, (read/write, reference: mouse_parameters_ptr)

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	msetrackmode	mouse position tracking mode
0010	16	msereportmode	mouse reporting mode
0020	16	msescale_X	mouse X-scale factor
0030	16	msescale_Y	mouse Y-scale factor
0040	16	msemin_X	mouse window min X
0050	16	msemin_Y	mouse window min Y
0060	16	msemax_X	mouse window max X
0070	16	msemax_Y	mouse window max Y
0080	16	msecsr_type	cursor type: rectangle, symbol, etc.
0090	16	msecsr_bool	cursor boolean operation
00A0	32	msecsr_size	cursor size [dy,dx]
00C0	32	msecsr_shape1	address of cursor shape #1
00E0	32	msecsr_save	screen data save address
0100	32	msecsr_offset	[yoffset, xoffset]
0120	32	msecsr_color1	mouse cursor color #1
0140	32	msecsr_wpageaddr	mouse cursor write page address
0160	16	msecsr_wpage	mouse cursor write page # (0,1,...)
0170	16	msecsr_wchannel	mouse cursor channel ID
0180	32	msecsr_dptch	mouse cursor destination pitch
01A0	16	msecsr_convdp	mouse cursor convert dptch
01B0	16	msecsr_psize	mouse cursor pixel size
01C0	32	msecsr_shape2	address of cursor shape #2
01D0	32	msecsr_color2	mouse cursor color #2
0200	32	msecsr_pitch1	shape #1 pitch
0220	32	msecsr_pitch2	shape #2 pitch
0240	32	msecsr_save_pitch	save buffer pitch

msetrackmode - mouse position tracking mode

BIT	FIELD
0	local tracking mode 0: mouse cursor position controlled externally 1: mouse cursor position controlled internally by graphics board
1	mouse cursor “wrap” mode 0: mouse cursor “sticks” at boundary 1: mouse cursor “wraps” to other side of boundary
2	mouse cursor “confine” mode 0: mouse cursor confined to screen boundaries 1: mouse cursor confined to mouse window boundaries
3	swap mouse X,Y coordinates 0: normal 1: swaps X,Y mouse cursor movement
4	mouse cursor save/restore mode 0: mouse cursor save/restore handled by host 1: mouse cursor save/restore handled by graphics board

msereportmode - mouse reporting mode

VALUE	MOUSE REPORTING MODE
0	report on switch closure
1	report on switch closure or release
2	report on switch release
3	report all movement while any switch closed
4	report all movement

msescale_X, msescale_Y - mouse cursor-movement X,Y scale factors
Incoming mouse-movement values are multiplied by the X and Y scale-factors in calculating an updated mouse cursor position. The scale-factor fields are in 16-bit fixed-point format (8-bit integer, 8-bit fraction). The default scale-factors are +1.000 (0100h) for both X and Y.

msemin_X, msemin_Y - mouse window minimum X,Y (upper-left corner)

msemax_X, msemax_Y - mouse window maximum X,Y (lower-right corner)
If the “confine” bit is set in the **msetrackmode** field, then the mouse window boundaries define the maximum extent of mouse cursor movement. Otherwise, the mouse cursor is merely constrained to the screen boundaries.

msecsr_type - mouse cursor type

VALUE	MOUSE CURSOR TYPE
0	filled rectangle
1	outlined rectangle
2	single color symbol
3	two color symbol

4	single color bitmap
5	two color bitmap

msecsr_bool - mouse cursor boolean operation

msecsr_size - mouse cursor size

msecsr_shape1, msecsr_shape2 - address of symbol data for mouse cursor types 2, 3, address of bitmap data for types 4, 5. Symbol data is in exactly the same format as required by SSYM:

OFFS	SIZE	DESCRIPTION
0000	16	symbol width in pixels
0010	16	symbol height in pixels
0020	16	symbol depth (pixel size)
0030	16	symbol pitch
0040	16	symbol ID (default symbols)
0050	16	(unused)
0060		beginning of symbol data

msecsr_pitch1, msecsr_pitch2 - pitch of cursor shape bitmap data for cursor types 4,5.

msecsr_save - mouse cursor screen data save address
The mouse cursor save area is a RAM buffer where the previous screen contents are saved when the cursor is displayed. The screen save buffer must be large enough to accomodate the cursor size and save buffer pitch. A default save buffer exists that can accomodate cursors not larger than 32 x 32 pixels (or a total “area” of 1024 pixels).

msecsr_save_pitch - save buffer pitch
The cursor save buffer may be allocated to memory with a pitch different from the cursor width (such as off-screen video memory).

msecsr_offset - mouse cursor X,Y offset
The mouse cursor offset specifies the X,Y offset [yoffset,xoffset] from the upper left hand corner of the cursor rectangle to the cursor “hot-spot.”

msecsr_color1, msecsr_color2 - mouse cursor color(s)

msecsr_wpageaddr - mouse cursor write page address
The mouse cursor page base address is the linear address corresponding to the [0,0] (upper-left) pixel of the page on which the mouse cursor is displayed. The mouse cursor page may be changed with MSCSRPAGE.

msecsr_wpage - mouse cursor write page number (0,1,...)
The mouse cursor page number indicates which of the default pages the mouse cursor is displayed on, and is only valid if the mouse cursor page address had been set with MSCSRPAGE.

msecsr_wchannel - mouse cursor channel ID
The mouse cursor channel ID indicates which channel the mouse cursor is displayed on. This field would be zero for all but those boards having more than one channel.
See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

msecsr_dptch, msecsr_convdp - mouse cursor destination pitch parameters
The mouse cursor destination pitch parameters must contain the proper pitch and pitch conversion values for the page on which the mouse cursor is displayed.

msecsr_psize - mouse cursor pixel size

OPCODE	MOUSE PARAMETER FIELDS MODIFIED

MSCURSOR	msecsr_type msecsr_bool msecsr_size msecsr_shape1 msecsr_save msecsr_offset	msecsr_color1 msecsr_shape2 msecsr_color2 msecsr_pitch1 msecsr_pitch2 msecsr_save_pitch
MSCSRPAGE	msecsr_wpageaddr msecsr_wpage msecsr_wchannel msecsr_dptch msecsr_convdp msecsr_psize	

Environment Header Descriptor (read-only, reference: env_descriptor_ptr). The fields in the Environment Header Descriptor specify the necessary allocation sizes for various portions of the environment which must be known if an application wishes to allocate and configure its own environment(s). The sizes are specified in bytes as a convenience for host memory allocations.

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	ENV_len_bytes	environment header size in bytes
0010	16	GC_len_bytes	graphics context block size in bytes
0020	16	TC_len_bytes	size of default text context buffer

AFGIS Execution Status Buffer (read-only, reference: afgis_exec_stat_ptr)

OFFS	SIZE	FIELD	DESCRIPTION
0000	32	afgis_dladdr	beginning of current AFGIS display list
0020	32	afgis_opaddr	current AFGIS opcode address
0040	16	afgis_opcode	current AFGIS opcode value

Error Parameter Buffer (read-only, reference: error_buffer_ptr). When an illegal AFGIS opcode is detected, the errflag field in Fixed RAM is set and the Error Parameter Buffer fields are loaded with the illegal opcode and the address at which it occurred. Also, if error interrupts are enabled, an interrupt is output to the host.

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	bad_afg_opcode	illegal AFGIS opcode value
0010	16		(reserved)
0020	32	bad_afg_op_addr	illegal AFGIS opcode address

Version Parameters (read-only, reference: version_params_ptr). The Version Parameters fields give access to information about the hardware model, firmware version, and video configuration.

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	model_id	model # (board type) (integer)
0010	16	version_N	firmware version number (integer)

0020	16	version_F	version flag (internal use)
0030	16	version_ops	opcodes supported: STD, 2D/3D, ... (coded)
0040	32	version_N_str	address of version # string
0060	32	version_str	address of version ID string
0080	32	model_str	address of model ID string
00A0	32	config_str	address of video configuration ID string
00C0	32	vers_date_str	address of version date string
00E0	32	copyright_str	address of copyright string

Screen parameters (read-only, reference: screen_params_ptr)

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	Xres	horizontal resolution (pixels)
0010	16	Yres	vertical resolution (pixels)
0020	16	Xmax	right-most pixel (Xres-1)
0030	16	Ymax	bottom-most pixel (Yres-1)
0040	16	fbwidth	frame buffer width (pixels)
0050	16	psize	pixel size (psize)
0060	16	nColors	# of colors
0070	16	pixelmask	pixel mask $((2^{psize})-1)$
0080	32	dptch	display pitch (bits)
00A0	32	dptch_bytes	display pitch (bytes)
00C0	32	gray_palette	default RAMDAC gray-scale palette
00E0	32	color_palette	default RAMDAC color palette
0100	32	redmask	RED color code for default palette
0120	32	greenmask	GREEN color code for default palette
0140	32	bluemask	BLUE color code for default palette
0160	32	graymask	GRAY color code for default palette

Hardware configuration parameters (read-only, reference: hwconfig_params_ptr)

OFFS	SIZE	FIELD	DESCRIPTION
0000	16		reserved
0010	16	interlace	interlaced display (T/F)

Memory configuration parameters (read-only, reference: memory_config_ptr)

OFFS	SIZE	FIELD	DESCRIPTION
0000	32	DRAM	address of beginning of DRAM
0020	32	DRAM_end	address of end of DRAM
0040	32	DRAM_bytes	total DRAM size (bytes)
0060	32	USRRAM	address of beginning of user RAM

0080	32	USRRAM_end	address of end of user RAM
00A0	32	USRRAM_bytes	user DRAM size (bytes)
00C0	32	XOSRAM	address of extra system DRAM (above OS)
00E0	32	XOSRAM_end	address of end of extra system DRAM
0100	32	XOSRAM_bytes	extra system DRAM size (bytes)
0120	16	VRAM_banks	# of installed VRAM banks
0130	16	VRAM_pages	# of usable video pages

Default Host Buffers (read/write, reference: default_HB0_ptr, default_HB1_ptr).
Host Buffers are reserved for use by the host and are not queried or written-to by the graphics firmware. The Default Host Buffers are 32 words each (64 bytes) and are incorporated in the default environment.

System Power-up Configuration Parameters (read-only, reference: pu_parameters_ptr)
These fields are written by system test and configuration routines at power-up or reset and communicate the results of the power-up tests and configuration checks.

OFFS	SIZE	FIELD	DESCRIPTION
0000	16		(reserved)
0010	16	UART_ok	UART test result
0020	16	RAMDAC_ok	RAMDAC test result
0030	16	34082_ok	coprocessor functioning (T/F)
0040	16	bpinstalled	BIT-PROMs installed (T/F)
0050	16	dram_pretest	DRAM pre-test error count
0060	16	vram_pretest	VRAM pre-test error count
0070	16		(reserved)
0080	32	config_ptr	video configuration data base pointer

System Memory Allocation Parameters (read/write, reference: sys_mem_alloc_ptr)
The System Memory Allocation fields contain pointers which specify the location of the AFGIS "heap". The heap is the user memory allocation area that is used by the AFGIS opcodes R_ALLOC, R_FREE, R_ENVB and R_ENVC.

At reset, the heap is configured to contain all user memory, but this may be changed by overwriting the heap_beg and heap_end fields to specify new addresses for the heap boundaries. If these fields need to be modified, it should be done immediately after reset. Otherwise, existing blocks of already-allocated memory may then be located outside of the new heap boundaries.

The heap may also be cleared (all memory de-allocated) by writing a zero to the heap_nonempty field. Pre-existing memory allocations must then be dealt with by the host.

OFFS	SIZE	FIELD	DESCRIPTION
0000	32	heap_beg	pointer to start of heap
0020	32	heap_end	pointer to end of heap
0040	32	heap_nonempty	(NULL $\Rightarrow$ heap empty, non-empty otherwise)

Text cursor parameters (read/write, reference: text_parameters_ptr)

Text cursor parameters (continued)

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	txcsr_mode	text cursor mode flags
0010	16	txcsr_blink_rate	blink rate (number of di intervals)
0020	16	txcsr_X	text cursor X location (absolute)
0030	16	txcsr_Y	text cursor Y location (absolute)
0040	16	txcsr_state	text cursor state: 0=OFF, 1=ON
0060	16	txcsr_minX	text cursor window min X (absolute)
0070	16	txcsr_minY	text cursor window min Y (absolute)
0080	16	txcsr_maxX	text cursor window max X (absolute)
0090	16	txcsr_maxY	text cursor window max Y (absolute)
00A0	16	txcsr_type	cursor type: rectangle, symbol, etc.
00B0	16	txcsr_bool	cursor boolean operation
00C0	32	txcsr_size	cursor size [dy,dx]
00E0	32	txcsr_shape1	address of shape #1
0100	32	txcsr_save	screen data save address
0120	32	txcsr_offset	[yoffset,xoffset]
0140	32	txcsr_color1	text cursor color #1
0160	32	txcsr_wpageaddr	text cursor write page address
0180	16	txcsr_wpage	text cursor write page # (0,1,...)
0190	16	txcsr_wchannel	text cursor channel ID
01A0	32	txcsr_dptch	text cursor destination pitch
01C0	16	txcsr_convdp	text cursor convert dptch
01D0	16	txcsr_psize	text cursor pixel size
01E0	32	txcsr_shape2	address of shape #2
0200	32	txcsr_color2	text cursor color #2
0220	32	txcsr_pitch1	shape #1 pitch
0240	32	txcsr_pitch2	shape #2 pitch
0260	32	txcsr_save_pitch	save buffer pitch

txcsr_mode - text cursor mode flags

BIT	FIELD
0	text cursor save/restore mode 0: text cursor save/restore handled by host 1: text cursor save/restore handled by graphics board

txcsr_blink_rate - text cursor blink rate
A non-zero value enables text cursor blinking and specifies the number of vertical intervals between changes of state (ON-OFF or OFF-ON). The blink period (time for a complete cycle through both ON and OFF states) is therefore twice the blink rate. A zero rate disables blinking— the text cursor state is then only affected by the TXCSRON opcode.

txcsr_X, txcsr_Y - text cursor location (absolute)

txcsr_state - text cursor state: 0=OFF, 1=ON

txcsr_minX, txcsr_minY, txcsr_maxX, txcsr_maxY - text cursor clipping window (absolute)

txcsr_type - text cursor type

VALUE	TEXT CURSOR TYPE
0	filled rectangle
1	outlined rectangle
2	single color symbol
3	two color symbol
4	single color bitmap
5	two color bitmap

txcsr_bool - text cursor boolean operation

txcsr_size - text cursor size

txcsr_offset - text cursor X,Y offset
The text cursor offset specifies the X,Y offset [yoffset,xoffset] from the upper-left corner of the cursor rectangle to the cursor "hot spot."

txcsr_color1, txcsr_color2 - text cursor colors

txcsr_shape1, txcsr_shape2 - text cursor shape address
Address of symbol data (cursor types 2,3), or bitmap data (cursor types 4,5). Unused for cursor types 0 and 1.

txcsr_pitch1, txcsr_pitch2 - text cursor shape data pitches
Pitch of cursor shape bitmap data for cursor types 4,5.

txcsr_save - text cursor screen data save address
The text cursor save area is a RAM buffer where the previous screen contents are saved when the cursor is displayed. The screen save buffer must be large enough to accomodate the cursor size and save buffer pitch. A default save buffer exists that can accomodate cursors not larger than 32 x 32 pixels (or a total "area" of 1024 pixels).

txcsr_save_pitch - text cursor save buffer pitch
The cursor save buffer may be allocated to memory with a pitch different from the cursor width (such as off-screen video memory).

txcsr_wpageaddr - text cursor write page address
The text cursor page base address is the linear address corresponding to the [0,0] (upper-left) pixel of the page on which the text cursor is displayed. The text cursor page may be changed with TXCSRPAGE.

txcsr_wpage - text cursor write page number (0,1,...)
The text cursor page number indicates which of the default pages the text cursor is displayed on, and is only valid if the text cursor page address had been set with TXCSRPAGE.

txcsr_wchannel - text cursor channel ID
The text cursor channel ID indicates which channel the text cursor is displayed on. This field would be zero for all but those boards having more than one channel.
See the hardware manual for the particular graphics board to determine what channels are supported and the corresponding channel ID codes.

txcsr_dptch, txcsr_convdp - text cursor destination pitch and convert pitch parameters
The text cursor destination pitch parameters must contain the proper pitch and pitch conversion values for the page on which the text cursor is displayed.

txcsr_psize - text cursor pixel size

OPCODE	TEXT PARAMETER FIELDS MODIFIED	
TXCURSOR	txcsr_type txcsr_bool txcsr_size txcsr_shape1 txcsr_save txcsr_offset	txcsr_color1 txcsr_shape2 txcsr_color2 txcsr_pitch1 txcsr_pitch2 txcsr_save_pitch
TXCSRPAGE	txcsr_wpageaddr txcsr_wpage txcsr_wchannel txcsr_dptch txcsr_convdp txcsr_psize	

Bus Parameter Structure

Access: Read/Write

Reference: bus_parameters_ptr

These fields contain parameters that are specific to the host bus (e.g. VMEbus, ATbus, etc.), and the contents may vary, depending on the bus used.

For VMEbus applications, the Bus Parameters Fields are defined as follows:

OFFS	SIZE	FIELD	DESCRIPTION
0000	32	vmevecreg	address of VME interrupt vector register
0020	16	vector 0	interrupt vector-outbound message #0 (EODL)
0030	16	vector 1	interrupt vector-outbound message #1 (KEYBOARD)
0040	16	vector 2	interrupt vector-outbound message #2 (MOUSE)
0050	16	vector 3	interrupt vector-outbound message #3 (ERROR)
0060	16	vector 4	interrupt vector-outbound message #4 (60 Hz)
0070	16	vector 5	interrupt vector-outbound message #5 not used
0080	16	vector 6	interrupt vector-outbound message #6 not used
0090	16	vector 7	interrupt vector-outbound message #7 not used

The RGI graphics boards can output 8 interrupts, identified by a 3 bit code, that corresponds to activities on the graphics board. For VMEbus applications, an 8 bit interrupt vector is placed on the VMEbus in response to any of the above 8 interrupts. The default mode is for the vector contained in the interrupt vector register to be placed on the VMEbus in response to any of the above 8 interrupts. The value of the vector is determined by the host when it loads the vector into the vector register.

The default implementation outputs the same vector for all of the 8 interrupts generated by the graphics board. For some applications, it may be convenient to have a different vector for each of the 8 possible interrupts output by the graphics board, corresponding to a different host service routine.

The Bus Parameter Structure contains 9 entries. The first entry is the address of the vector register in TMS340x0 address space. This address is used by the host to load the desired vector into the interrupt vector register. The next 8 entries in the table are 8 bit vectors (on word boundaries) that correspond to the 8 possible interrupts output from the graphics board. If the user wants to have a different service routine for each of the 8 possible interrupts output by the graphics boards, the user must write the 8 bit vector into the appropriate field in the Bus Parameter Structure. If this mode is used, then an interrupt vector should be specified for each interrupt enabled in INOUTMASK in Fixed-RAM. At power up all the vector fields are set to zero. Any non-zero value will be loaded into the interrupt vector register when the corresponding interrupt is output. If a zero value for the vector exists in the table, the current value in the interrupt register is output onto the VMEbus in response to the interrupt.

Default parameter table pointers (read/write, reference: default_palette_ptr, default_ssymbol_ptr, default_dashpatn_ptr, default_stipple_ptr, default_tile_ptr, default_font_ptr, default_textsvc_ptr, default_msecsr_ptr, default_shift_ptr)
These fields in the global pointer table each contain the address of a table which in turn specifies a set of pre-defined default parameters for the relevant opcodes which support default ("index") parameters, namely: SETPALETTE, SSYM, DASHPATN, STIPPLE, TILE, FONT, TEXTSVC, MSCURSOR, TXCURSOR, USCURSOR, and SHIFT. These opcodes take a parameter which may either be the address of a user-defined structure, or an "index" ($0 \leq i \leq 127$) specifying one of the pre-defined structures.

As an example, here is how the STIPPLE opcode would process an "index" parameter. First the default table pointer is retrieved from the default_stipple_ptr field. The first entry in the table is the number of entries in the table ("n") which is compared against the index "i" (if $i \geq n$, then the specified index is invalid, and index i=0 is substituted). The next "n" entries in the table are the addresses of the "n" pre-defined structures, and the index "i" is used to retrieve the indicated address from the table (offset = $32 + (32 \cdot i)$). Processing now proceeds exactly as if this address had been passed to the STIPPLE opcode directly. The other opcodes that support index parameters function similarly (using the relevant default table pointer).

By constructing an alternate default table, and overwriting the relevant default table pointer entry, a set of custom defaults may be implemented. User-defined structures may then be selected by index rather than address.

Default Table Structure

OFFS	SIZE	FORMAT	DESCRIPTION
0000	32	integer	number of default parameters in table ("n")
0020	32	address	default parameter 0
0040	32	address	default parameter 1
• ⁝ ⁝	⁝ ⁝ ⁝	⁝ ⁝ ⁝	⁝ ⁝ ⁝
xxxx	32	address	default parameter "n-1"

Graphics Environment RAM

Graphics Environment RAM consists of an Environment Header and it's associated parameters, buffers and data structures, which define the general content of the Graphics Environment and are used for communication between the host and it's tasks. The Graphics Context is one part of the Graphics Environment and contains the specific graphics parameters such as color, position, logical origin, fill pattern, etc.

At power-up, AFGIS firmware configures a default Graphics Environment (the address of which is contained in the default_env_ptr field in Fixed RAM). This single default Graphics Environment could then be used by an application only having need for a single Graphics Environment.

For Multi-tasking applications, each task running on the host would typically have its own Graphics Environment that could be manipulated independently without interfering with the graphics environment of another task. The multi-tasking operating system provides the mechanisms for prioritization and time-slicing of multiple tasks. As each task executes (during its time-slice) it can modify or query parameters within its own Graphics Environment without affecting other tasks' Graphics Environments. When a task is ready to run a display list that it has previously downloaded, the address of the corresponding Graphics Environment must be written to the env_ptr field in Fixed RAM before the display list is started. The end result is that the color, screen position, etc. of one task does not affect the color, screen position, etc., of any other task, and the time required to switch environments is negligible – typically less than 15 µsec.

Environment Header

A Graphics Environment is identified by the address of its environment header. This is the address returned by the R_ENVB OR R_ENVC opcodes, and is the address written to the env_ptr variable in Fixed RAM to specify the current graphics environment. The environment header contains the various pointer and status fields shown below, whose contents in turn define the parameter buffers to be used by the system for graphics generation.

OFFS	SIZE	FIELD	DESCRIPTION
0000	32	GC_ptr	address of graphics context (GC)
0020	32	TC_ptr	address of text context (TC)
0040	32	TMS_ptr	address of TMS register image block
0060	32	AFV_ptr	address of AFGIS variables
0080	32	XFV_ptr	address of XFORM variables
00A0	32	HB0_ptr	address of host buffer 0
00C0	32	HB1_ptr	address of host buffer 1
00E0	32	text_svc	address of text service routine
0100	32	afgstk_lo	AFGIS stack low-address limit
0120	32	afgstk_lp	AFGIS stack low-pointer
0140	32	afgstk_hp	AFGIS stack high-pointer
0160	32	afgstk_hi	AFGIS stack high-address limit
0180	32	gspstk_lo	user TMS stack low-address limit
01A0	32	gspstk_lp	user TMS stack low-pointer
01C0	32	gspstk_hp	user TMS stack high-pointer
01E0	32	gspstk_hi	user TMS stack high-address limit
0200	32	hostres0	(reserved for use by host)
0220	32	hostres1	(reserved for use by host)
0240	32	hostres2	(reserved for use by host)
0260	32	hostres3	(reserved for use by host)
0280	16	num_AFVARS	# of AFGIS variables
0290	16	num_XFVARS	# of XFORM variables
02A0	16	AFVstatus	AFGIS variable status
02B0	16	TC_bytes	length of text context buffer (bytes)
02C0	16	HB0_bytes	length of host buffer 0 (bytes)
02D0	16	HB1_bytes	length of host buffer 1 (bytes)
02E0	16	TC_flag	return flag from text driver

Graphics Context Pointer (GC_ptr)
The GC_ptr field contains the address of the Graphics Context buffer which is used by the system as the source of the drawing parameters that control graphics generation.

Text Context Pointer (TC_ptr)
The TC_ptr field contains the address of the Text Context buffer which is used by the text service routine as the source of parameters for character generation. The text service routine and the Text Context are tightly coupled and replacing the text service routine would generally require the replacement or reinitialization of the Text Context. See the description of the text_svc field.

TMS Register Image Block Pointer (TMS_ptr)
The TMS_ptr field contains the base address of a buffer reserved for storing the images of the TMS340x0 A-file and B-file registers.

AFGIS Variables Pointer (AFV_ptr)
The AFV_ptr field contains the base address of the AFGIS variables. The AFGIS variables can be considered as an array of 32-bit memory locations. Variable V0 is located at the base address, V1 is at base+32, etc. The number of variables present is given by the num_AFVARs field. The AFGIS variables are used to pass input parameters to, and return output values from AFGIS opcodes.

Transform Variables Pointer (XFV_ptr)
The XFV_ptr field contains the base address of the AFGIS Transform variables. The AFGIS transform variables can be considered as an array of 512-bit structures ($4 \cdot 4 \cdot 32 = 512$ bits = 64 bytes). Transform variable T0 is located at the base address, T1 is at base+512, etc. The number of variables present is given by the num_XFVARs field. Transform variables are only used by TMS34020 based graphics boards that have the TMS34082 coprocessor installed with firmware supporting the AFGIS 2D and 3D opcodes.

Host Buffer Pointers (HB0_ptr, HB1_ptr)
The HB0_ptr and HB1_ptr fields contain the addresses of the two host buffers. These are reserved for use by the host and are not written to by the graphics firmware.

Text Service Routine (text_svc)
The text_svc field contains the address of the text service routine (text driver). The default text driver supports “RGI type 0" fonts, but this field can be overwritten to implement other text drivers to support other font data formats.

AFGIS Stack (afgstk_lo, afgstk_lp, afgstk_hp, afgstk_hi)
The AFGIS stack is used by the AFGIS opcodes PUSHV, POPV, PUSHVARS, and POPVARS to save and restore AFGIS variables. It is also used by the AFGIS CAL/RTRN and RPT/ERPT opcodes. The afgstk_lo and afgstk_hi fields define the low and high address boundaries of a stack area. The stack is empty when afgstk_lp=afgstk_lo and afgstk_hp=afgstk_hi.

User TMS Stack (gspstk_lo, gspstk_lp, gspstk_hp, gspstk_hi).
The user TMS Stack is used when executing user TMS assembly code (HINT1 command). The user TMS stack is structured exactly the same as the AFGIS stack described above. Gspstk_lo and gspstk_hi are the low and high stack bounds; the stack is empty when gspstk_lp=gspstk_lo and gspstk_hp=gspstk_hi. When a HINT1 command is issued, the contents of gspstk_lp are copied to TMS register A14 and the contents of gspstk_hp are copied to SP (the TMS stack pointer). At completion, the returned values of A14 and SP are written to gspstk_lp and gspstk_hp.

Host Reserved Pointers (hostres0, hostres1, hostres2, hostres3.)
These fields are reserved for use by the host and are not overwritten by the graphics firmware.

Number of AFGIS Variables (num_AFVARs)
The num_AFVARs field contains the number of AFGIS variables in the memory array specified by the AFV_ptr field.

Number of XFORM Variables (num_XFVARS)
The num_XFVARs field contains the number of Transform variables in the memory array specified by the XFV_ptr field.

AFGIS Variable Status (AFVstatus)
The AFVstatus field contains the status bits that are modified by AFGIS arithmetic opcodes (ADVV, SBVV, etc.).

Text Context Length (TC_bytes)
The TC_bytes field contains the length, in bytes, of the text context buffer specified by the TC_ptr field.

Host Buffer Sizes (HB0_bytes, HB1_bytes)
The HB0_bytes and HB1_bytes fields contain the lengths, in bytes, of the host memory buffers specified by the HB0_ptr and HB1_ptr fields, respectively.

Text Driver Return Flag (TC_flag)
The TC_flag field is used by the text driver (see text_svc) to return status information from text driver calls.

Graphics Context Table, continued

Graphics Context

The Graphics Context is used by the system as the source of the drawing parameters that control graphics generation. It is specified by the GC_ptr field in the environment header. Fields in the graphics context can be safely written by the host whenever the graphics context is not part of the current Graphics Environment, or no display list is currently executing. A graphics context may be initialized to default values with the AFGIS INITGC opcode.

OFFS	SIZE	FIELD	DESCRIPTION
0000	32	pmask	plane mask
0020	16	convdp	convert dptch
0030	16	psize	pixel size
0040	16	control	control register
0050	16	clipmode	clipping mode flag
0060	32	color1	foreground color
0080	32	color0	background color
00A0	32	wend	default clipping window end [XY] (absolute)
00C0	32	wstart	default clipping window start [XY] (absolute)
00E0	32	offset	offset (linear)
0100	32	dptch	destination pitch
0120	32	org	logical origin [XY]
0140	32	uwend	user clipping window end [XY]
0160	32	uwstart	user clipping window start [XY]
0180	32	loc	current location [XY]
01A0	32	fill_size	stipple pattern fill dimension [XY]
01C0	32	fill_ptr	stipple pattern data pointer (binary pixblt format)
01E0	32	tile_size	tile pattern fill dimension [XY]
0200	16	tile_depth	tile pattern depth (pixel size in bits)
0210	16	tile_sptch	pattern pitch (array pitch in bits)
0220	32	tile_ptr	pattern data pointer (pixel-array format)
0240	16	line_cont	line pattern continue flag
0260	32	line_patn	32-bit line-pattern data
02A0	16	dash_len	dash pattern list length (in words)
02B0	16	dash_cont	continue flag
02C0	32	dash_ptr	pointer to segment-length word-list
02E0	16	dash_sbit	start bit offset
02F0	16	dash_sword	start word offset
0330	16	pen_type	pen type (rectangle, ellipse, ...)
0340	32	pen_size	pen dimensions [Y/2, X/2] (half sizes)
0360	32	arc_center	center of last arc/sector
0380	32	arc_start	starting point of last arc/sector
03A0	32	arc_end	ending point of last arc/sector
03C0	16	fatln_width	fat-line width (pixels)
03D0	16	fatln_cap	fat-line cap-style

03E0	16	fatln_join	fat-line join-style
0400	16	patrn_mode	pattern fill reference point mode flag
0420	32	patrn_ref	pattern-fill reference point offset
0460	16	pixmask	pixel-mask = $((2^{psize})$-1)
0470	16	pixrep	pixel-replication count = 32/psize
0480	16	wpage	current write page number (read-only)
0490	16	wchannel	current write page channel ID (read-only)
04A0	32	ctextl	ctext current location [XY]
04C0	32	ctextm	ctext margin [XY]
04E0	16	uscsr_type	user cursor type: rectangle, symbol, etc.
04F0	16	uscsr_bool	boolean operation
0500	32	uscsr_size	size [dy,dx]
0520	32	uscsr_shape1	address of shape #1
0540	32	uscsr_save	screen data save address
0560	32	uscsr_offset	[yoffset, xoffset]
0580	32	uscsr_color	cursor color #1
05B0	16	uscsr_state	state: 0=off, 1=on (read-only)
05C0	32	uscsr_loc	user cursor x,y, location (window relative)
0620	32	rand_seed	current random number seed
0640	32	rand_lo	random number range low value
0660	32	rand_hi	random number range high value
0680	16	rand_initflag	seed initialization flag (0=not, 1=set)
06A0	16	mark_type	marker type (coded)
06B0	16	mark_param	type parameter (depends on marker type)
06C0	32	mark_size	size [dy,dx] (for some types)
06E0	32	mark_shape	data pointer (for some types)
0700	32	mark_offset	[yoffset, xoffset]
0720	32	mark_color	marker color
0740	16	mark_flags	marker flags
0780	32	uscsr_shape2	address of shape #2
07A0	32	uscsr_color2	user cursor color #2
07C0	32	uscsr_pitch1	shape #1 pitch
07E0	32	uscsr_pitch2	shape #2 pitch
0800	32	uscsr_save_pitch	save buffer pitch
08C0	32	fill-pitch	stipple pattern pitch (bits)

Plane Mask (pmask)
The pmask field contains the plane mask parameter which is loaded into the TMS340x0 pmask register. Zero-bits in the plane mask parameter enable modification of the corresponding pixel-bits. Refer to the TMS340x0 User's Guide for more information about the pmask register. This parameter is modified by the AFGIS PMASK opcode. (default = 0)

Destination Pitch Conversion (convdp)
The convdp field contains the destination pitch conversion parameter which is loaded into the

TMS340x0 convdp register. Refer to the TMS340x0 User's Guide for more information about the convdp register. (default value corresponds to the default destination pitch)

Pixel Size (psize)
The psize field contains the pixel size parameter which is loaded into the TMS340x0 psize register. Refer to the TMS340x0 User's Guide for more information about the psize register. (default value corresponds to the system video configuration)

Control Register Image (control)
The control field contains the control register image parameter which is loaded into the TMS340x0 control register. Fields in the control register affect pixel processing and transparency modes. Refer to the TMS340x0 User's Guide for more information about the control register. This parameter is modified by the AFGIS BOOL and TRANS opcodes. (default value specifies PPOP = replace, transparency off)

Clipping Mode Flag (clipmode)
The clipmode field specifies which clipping window parameters are used. If clipmode=0, the “default” clipping window parameters are used (wend, wstart) ; if clipmode=1, the “user” clipping window parameters are used (uwend, uwstart) and are assumed to be relative to the current logical origin; if clipmode=2, the user clipping window parameters are used and are assumed to be absolute. This parameter is modified by the AFGIS CLIPMODE opcode. (default=0: use wstart, wend)

Background Color, Foreground Color (color0, color1)
The color0 and color1 fields contain the background and foreground color parameters, respectively, that are loaded into the TMS340x0 B-file registers B8(COLOR0) and B9(COLOR1). Refer to the TMS340x0 User's Guide for more information about the color registers B8 and B9. The AFGIS COLORB and COLORF opcodes modify the color0 and color1 fields, respectively. (defaults: color0=0, color1=0ffffffffh)

Clipping Window Start, End (wstart, wend)
The wstart and wend fields contain the clipping window parameters that are loaded into the TMS340x0 B-file registers B5(WSTART) and B6(WEND) when clipmode=0 (see clipmode field above). The [XY] parameters contained in wstart and wend are absolute, i.e., they are referenced to the absolute screen origin rather than being relative to the logical origin (org field). Refer to the TMS340x0 User's Guide for more information about the B5(WSTART) and B6(WEND) registers. (defaults: wstart=[0,0], wend corresponds to the system screen resolution)

Screen Origin Offset (offset)
The offset field contains the screen origin address parameter which is loaded into the TMS340x0 B-file register B4(OFFSET). This specifies the linear address of the absolute screen origin for graphics operations and hence defines the current write page. This parameter is modified by the AFGIS WPG and WPGA opcodes. Refer to the TMS340x0 User's Guide for more information about the B4(OFFSET) register. (default value corresponds to write-page 0)

Destination Pitch (dptch)
The dptch field contains the destination pitch parameter which is loaded into the TMS340x0 B-file register B3(DPTCH). This specifies the row-pitch for the current write page (defined by the offset field). Refer to the TMS340x0 User's Guide for more information about the B3(DPTCH) register. (default value corresponds to the system frame-buffer size)

Logical Origin (org)
The org field defines a point relative to the absolute screen origin which is subsequently used as a “logical origin” by the firmware graphics routines. All coordinates specified to AFGIS opcodes are adjusted relative to the current logical origin. This is convenient for window-oriented operations as by merely modifying the logical origin the same display can be drawn at different positions on the screen without having to modify the coordinate parameters. This parameter is modified by the AFGIS XYORG opcode. (default = [0,0])

User Clipping Window Start, End (uwstart, uwend)
The uwstart and uwend fields contain the clipping window parameters that are loaded into the TMS340x0 B-file registers B5(WSTART) and B6(WEND) when clipmode≠0 (see clipmode field above). The [XY] parameters contained in uwstart and uwend are relative to the logical origin (org field) if clipmode=1, or are absolute if clipmode=2. Refer to the TMS340x0 User's Guide for more

information about the B5(WSTART) and B6(WEND) registers. (defaults: uwstart=[0,0], uwend corresponds to the system screen resolution)

Current Location (loc)
The loc field defines a point relative to the logical origin (org) which is subsequently used as the “current location” by the firmware graphics routines. In particular, the current location is used by the AFGIS opcodes LINETO, LINETOR, SSYM, and GTEXT, and is modified by MOVETO and MOVETOR. (default = [0,0])

Stipple Fill Pattern Parameters (fill_size, fill_ptr, fill_pitch)
These fields define the current stipple (binary) fill pattern that is used by the firmware area-fill routines for filltype 1. The fill_size field specifies the fill pattern dimensions ([XY] format); the fill_ptr field contains the address of the pattern data (binary pixblt format); fill_pitch is the row-pitch of the stipple pattern data. These fields are modified by the AFGIS STIPPLE opcode. (default values correspond to default stipple pattern #0)

Tile Fill Pattern Parameters (tile_size, tile_depth, tile_sptch, tile_ptr)
These fields define the current tile (pixel-mapped) fill pattern that is used by the firmware area-fill routines for filltype 2. The tile_size field specifies the fill pattern dimension ([XY] format); tile_depth is the pixel size (in bits) of the encoded pattern data; tile_sptch is the row-pitch of the pattern data (difference in bits between adjacent rows of the pattern); tile_ptr is the address of the fill pattern data. These fields are modified by the AFGIS TILE opcode. (default values correspond to default tile pattern #0)

Binary Line Pattern Parameters (line_cont, line_patn)
These fields define the current binary line pattern that is used by the firmware line drawing routines for linetype 1. The line_cont field is the pattern continue flag. If line_cont = 0, subsequent patterned lines will each begin drawing at the same point in the pattern – otherwise subsequent patterned lines will each begin drawing at the next point in the pattern where the previous line ended. The line_patn field is the 32-bit binary line pattern. Zero-bits in the pattern are drawn with background color and one-bits are drawn with foreground color. Drawing proceeds from the least-significant bit of the 32-bit pattern to the most-significant bit, repeating as necessary. These fields are modified by the AFGIS LINECON and LINEPATN opcodes. (default = 0F0F0F0Fh).

Dashed Line Pattern Parameters
(dash_len, dash_cont, dash_ptr, dash_sbit, dash_sword)
These fields define the current dashed line pattern that is used by the firmware line drawing routines for linetype 2. The dash_len field specifies the number of words contained in a segment-length word-list array located at the address specified by the dash_ptr field. The segment-length list is an array of words, each of which specifies the length (in pixels) of successive dash segments. Segment 0 (corresponding to word 0 — the initial word in the list) and all other even-numbered segments are drawn with background color; odd-numbered segments are drawn with foreground color. The dash_cont field is the pattern continue flag. If dash_cont=0, subsequent dashed lines will each begin drawing at the same point in the pattern – otherwise subsequent dashed lines will each begin drawing at the next point in the pattern where the previous line ended. The dash_sbit and dash_sword fields define the starting pixel within the pattern (when dash_cont = 0) dash_sword specifies the word number within the word-list array that encompasses the starting pattern pixel, and dash_sbit specifies the starting pixel number within the corresponding segment. These fields are modified by the AFGIS opcodes, DASHCON, DASHOFFS and DASHPATN (note that DASHOFFS calculates the dash_sbit and dash_sword field values from the specified pixel-offset parameter). (default values correspond to default dashed-line pattern #0)

Pen Line Parameters (pen_type, pen_size)
These fields define the current pen type that is used by the firmware line drawing routines for linetypes 6, 7 and 8. The pen_type and pen_size fields define the pen stylus shape and size as follows

pen_type	pen_size
0 (rectangle)	[height/2, width/2]
1 (ellipse)	[Yradius, Xradius]

These fields are modified by the AFGIS PENDEF opcode. (default values specify a rectangular pen 8 pixels square)

Arc Parameters (arc_center, arc_start, arc_end)
These fields record the parameters of the last arc or sector drawn by the firmware graphic routines. The arc_center field contains the coordinates of the arc center, relative to the logical origin at the time the arc was drawn. The arc_start and arc_end fields contain the endpoints of the arc corresponding to the start and end angles, relative to the arc center. All fields are in [XY] format. These fields are modified by the AFGIS opcodes ARC, SECT, SECTS, SEG, and SEGS and are queried by the R_ARC opcode (note that R_ARC resolves the center-relative coordinates and reports all points as relative to the logical origin).

Fat Line Parameters (fatln_width, fatln_cap, fatln_join)
These fields define the current fat line parameters that are used by the firmware line drawing routines for linetypes 3,4 and 5. The fatln_width field specifies the fat-line width in pixels; fatln_cap specifies the fat-line cap-style (shape of the terminal ends of line segments); fatln_join specifies the fat-line joint-style (shape of the corner joints between two contiguous line segments).

fatln_cap	**fatln_join**
0 = BUTTED	0 = MITERED
1 = ROUNDED	1 = ROUNDED
2 = PROJECTING	2 = BEVELED

Pattern Fill Mode Parameters (patrn_mode, patrn_ref)
These fields specify how the pattern-fill reference point is determined by the firmware area-fill routines for filltypes 1 and 2. If patrn_mode = 0, successive pattern-filled figures are referenced to the same point and overlapping figures will show no break in continuity of the pattern. If patrn_mode=1, patterns are referenced to a point relative to each figure, and overlapping figures may reveal a discontinuity in the pattern along the boundary of a figure overlapping a previous figure. For patrn_mode 2 or 3, patterns are referenced to the figure bounding rectangle. The patrn_ref field specifies the pattern reference point (relative to the logical origin) for patrn_mode=0, origin offset by which the pattern reference point is adjusted for pattern modes 1,2, and 3. These fields are modified by the AFGIS opcodes PATRNMODE and PATRNREF. (default: patrn_mode=0, patrn_ref=[0,0])

Pixel Mask Parameters (pixmask, pixrep)
These fields complement the psize field and are a convenience for the firmware graphics routines in performing certain pixel-oriented operations. The pixmask field is a pixel mask corresponding to the pixel size and consists of a contiguous group of 1-bits as wide as a pixel (i.e., a pixel consisting of all 1's, pixmask = ((2^{psize})-1). The pixrep field contains the value 32 divided by the current pixel size. (default values correspond to the system pixel size)

psize	pixmask	pixrep
1	1	32
2	3	16
4	0fh	8
8	0ffh	4

Write Page Number (wpage)
The wpage field contains the number of the current write page, and corresponds to the value in the offset field. This field is modified by the AFGIS WPG and WPGA opcodes (WPGA sets wpage=-1 to indicate that the specified offset address does not necessarily correspond to one of the default page boundaries). (default = page 0)

Write Page Channel ID (wchannel)
The wchannel field contains the channel ID parameter for the current write page. This field is modified by the R_ENVC, INITGCC or WPAGEB opcodes, but is only relevant to those boards having more than one graphics output channel (such as the RG-751, RG-752, or RG-753). Refer

to the hardware manual for the particular graphics board to determine what output channels are supported. On boards that support only a single output channel, the channel ID parameter is ignored, and defaults to “0" (main underlay channel).

CTEXT Current Location (ctextl)
The ctextl field specifies the current [XY] location used by the CTEXT routines. CTEXT maintains a separate screen position independent of the position used by other graphics routines (loc field). This field is modified by the AFGIS CTEXTLXY opcode and updated by the CTEXT opcodes after each string is drawn. (default = [0,0])

CTEXT Margin Location (ctextm)
The ctextm field specifies the current [XY] margin location used by the CTEXT routines for processing control characters. This field is modified by the AFGIS CTEXTMXY opcode. (default = [0,0])

User Cursor Parameters
(uscsr_type, uscsr_bool, uscsr_size, uscsr_shape, uscsr_save, uscsr_offset, uscsr_color, uscsr_state, uscsr_loc). These fields define the current user cursor.

FIELD	SIZE	FORMAT	DESCRIPTION
uscsr_type	16	integer	user cursor type (see below)
uscsr_bool	16	coded	cursor boolean operation
uscsr_size	32	[XY]	cursor size [dy,dx] (types 0,1)
uscsr_shape1	32	linear	address of shape #1
uscsr_save	32	linear	screen data save address
uscsr_offset	32	[XY]	cursor offset [yoffset, xoffset]
uscsr_color1	32	color	cursor color #1
uscsr_state	16	boolean	cursor state: 0=off, 1=on (read-only)
uscsr_loc	32	[XY]	user cursor x,y location (window relative)
uscsr_shape2	32	linear	address of shape #2
uscsr_color2	32	color	user cursor color #2
uscsr_pitch1	32	linear	shape #1 pitch
uscsr_pitch2	32	linear	shape #2 pitch
uscsr_save_pitch	32	linear	save buffer pitch

VALUE	USER CURSOR TYPE
0	filled rectangle
1	outlined rectangle
2	single color symbol
3	two color symbol
4	single color bitmap
5	two color bitmap

The uscsr_state field is read-only and should only be modified by executing the AFGIS USCSRON opcode. The user cursor should be OFF before attempting to modify any of the other fields. These fields are modified by the AFGIS opcodes USCURSOR, USCSRON and USCSRXY.

Random Number Parameters (rand_seed, rand_lo, rand_hi, rand_initflag)
These fields control the generation of random numbers by the AFGIS R_RAND opcode. The rand_seed field contains the current random number seed value, and rand_lo and rand_hi are the random number range low and high values, respectively. R_RAND generates a random number N in the range rand_lo ≤ N ≤ rand_hi. Note that there is NO default seed value. The rand_initflag field should be set to 1 at the time a seed value is specified in the rand_seed field. Otherwise, if

rand_initflag=0 at the time R_RAND is executed, a random seed is constructed from the on-board video refresh counters and rand_initflag is then set to 1. These fields are modified by the AFGIS opcodes RANDRANGE and RANDSEED. (defaults: rand_lo=0, rand_hi=65535, rand_initflag=0)

Marker Parameters
(mark_type, mark_param, mark_size, mark_shape, mark_offset, mark_color, mark_flags)
These fields define the current marker that is used by the AFGIS PMARK opcode, and are modified by the AFGIS MARKER opcode.

FIELD	SIZE	FORMAT	DESCRIPTION
mark_type	16	coded	marker type
mark_param	16	coded	type parameter (depends on marker type)
mark_size	32	[XY]	size [dx,dy] (for some types)
mark_shape	32	linear	data pointer (for some types)
mark_offset	32	[XY]	[yoffset, xoffset]
mark_color	32	color	marker color
mark_flags	16	coded	marker flags

FLAG FIELD	DESCRIPTION
bit 0:	0 = use current foreground color (color1 field) 1 = use specified marker color (mark_color field)

MARKER	TYPE	PARAM	SIZE	SHAPE
outlined ellipse	0	linetype	[yrad, xrad]	N/A
filled ellipse	1	filltype	[yrad, xrad]	N/A
outlined rectangle	2	linetype	[h/2, w/2]	N/A
filled rectangle	3	filltype	[h/2, w/2]	N/A
outlined diamond	4	linetype	[h/2, w/2]	N/A
filled diamond	5	filltype	[h/2, w/2]	N/A
“+” mark	6	linetype	[h/2, w/2]	N/A
“X” mark	7	linetype	[h/2, w/2]	N/A
symbol	8	rotation	N/A	address
character	9	charcode	N/A	N/A

The mark_offset field contains a signed [XY] offset by which each coordinate is adjusted before a marker is drawn; mark_color is the optional marker color; and mark_flags is the flag field described above. (default values correspond to an outlined circle of radius 8, linetype = 0, color = 0ffffffffh).

AFGIS-3.11 Appendix B

Keyboard Interface Keycodes

All keycodes are in hexadecimal. Keycodes preceded by an asterisk (*) are Extended ASCII codes, i.e., they are two byte codes, the first byte being 00, e.g., *29 = 00 29, KBDATA0 = 0000h, KBDATA1 = 0029h. For standard ASCII codes, KBDATA1 = FFFFh and KBDATA0 contains the ASCII code for the character.

Key Legend	BASE	SHIFT	CTRL	ALT	Notes
‘~	60	7E	—	*29	
1!	31	21	—	*78	
2@	32	40	00	*79	
3#	33	23	—	*7A	
4$	34	24	—	*7B	
5%	35	25	—	*7C	
6^	36	5E	1E	*7D	
7&	37	26	—	*7E	
8*	38	2A	—	*7F	
9(	39	28	—	*80	
0)	30	29	—	*81	
-_	2D	5F	1F	*82	
=+	3D	2B	—	*83	
Backspace	08	08	7F	*0E	
Tab	09	*0F	*94	*A5	
Q	71	51	11	*10	
W	77	57	17	*11	
E	65	45	05	*12	
R	72	52	12	*13	
T	74	54	14	*14	
Y	79	59	19	*15	
U	75	55	15	*16	

Figure 6.1 KEYBOARD INTERFACE KEYCODES

Most keycodes emitted by the keyboard interface firmware conform to those found in a PC/AT/DOS system. The exceptions are the numeric keypads “Enter” and “/” keys and the “PrtSc”, “Scroll Lock” and “Pause” keys. In the case of more than one shift type (Shift, Ctrl, or Alt) asserted simultaneously, the order of precedence from

Key Legend	BASE	SHIFT	CTRL	ALT	Notes
I	69	49	09	*17	
O	6F	4F	0F	*18	
P	70	50	10	*19	
[{	5B	7B	1B	*1A	
]}	5D	7D	1D	*1B	
Enter	0D	0D	0A	*1C	
Caps Lock	—	—	—	—	
A	61	41	01	*1E	
S	73	53	13	*1F	
D	64	44	04	*20	
F	66	46	06	*21	
G	67	47	07	*22	
H	68	48	08	*23	
J	6A	4A	0A	*24	
K	6B	4B	0B	*25	
L	6C	4C	0C	*26	
;:	3B	3A	—	*27	
‘"	27	22	—	*28	
Left Shift	—	—	—	—	
Z	7A	5A	1A	*2C	
X	78	58	18	*2D	
C	63	43	03	*2E	
V	76	56	16	*2F	
B	62	42	02	*30	
N	6E	4E	0E	*31	

Keyboard Interface Keycodes *(continued)*

Key Legend	BASE	SHIFT	CTRL	ALT	Notes
M	6D	4D	0D	*32	
,<	2C	3C	—	*33	
.>	2E	3E	—	*34	
/?	2F	3F	*95	*35	(1)
Right Shift	—	—	—	—	
Left CTRL	—	—	—	—	
Left ALT	—	—	—	—	
Space Bar	20	20	20	*20	(1)
Right ALT	—	—	—		
\\|	5C	7C	1C	*2B	
Right CTRL	—	—	—		

Figure 6.2 MAIN KEY CLUSTER KEYCODES

Key Legend	BASE	SHIFT	CTRL	ALT	Notes
Esc	1B	1B	1B	*01	
F1	*3B	*54	*5E	*68	
F2	*3C	*55	*5F	*69	
F3	*3D	*56	*60	*6A	
F4	*3E	*57	*61	*6B	
F5	*3F	*58	*62	*6C	
F6	*40	*59	*63	*6D	
F7	*41	*5A	*64	*6E	
F8	*42	*5B	*65	*6F	
F9	*43	*5C	*66	*70	

AFGIS-3.11 Appendix C

Font Definitions

Several pre-defined fonts are included as part of the AFGIS firmware, and can be selected with the FONT opcode by specifying the font number. User-defined fonts can be down-loaded into graphics board RAM and selected with the FONT opcode by specifying the address of a font definition structure.

The font definition structure consists of a font header which contains the various parameters pertaining to the font, and the font bit-map data.

Font Header

OFFS	SIZE	FIELD	DESCRIPTION
0000	16	font_type	font type code (RGI type 0 = 3052h)
0010	32	font_len	font length (bytes)
0030	16	font_ID	font ID number
0040	32	str_offs	offset to font ID string (bits)
0060	32	data_offs	offset to character bit-map data (bits)
0080	16	fontX	character cell width (pixels)
0090	16	fontY	character cell height (pixels)
00A0	32	cellsize	character cell size (bits) (fontX • fontY)
00C0	16	char_min	code number of first character in font
00D0	16	char_max	code number of last character in font
00E0	16	char_def	code number of default character
00F0	16	ascent	top of character cell to baseline (pixels)
0100	16	descent	bottom of character cell to baseline (pixels)
0110	16		(unused)
0120	16	whiteX	built-in horizontal cell spacing (internal)
0130	16	whiteY	built-in vertical cell spacing (internal)
0140	16	spaceX	default horizontal cell spacing (external)
0150	16	spaceY	default vertical cell spacing (external)
0160	2048		(parameters derived from fontX and fontY fields)

Font Type code (font_type)

The font_type field contains a unique code that identifies the format of the font structure. For RGI type 0 fonts (the format described here), the type code is 3052h (“R0").

Font Definitions (continued)

Font Length (font_len)

The font length encompasses both the font header and the character bit-map data, and indicates the total size of the font data. The host can use this size parameter in downloading a user-defined font from host memory to the graphics board. The size is specified in bytes for host convenience.

Font ID Number (font_ID)

The font ID field contains a unique number corresponding to each distinct font database in the default font set. The value specified for this field can be considered to be the user's perogative, but for compatibility should be -1 (0ffffh) for downloaded or user-defined fonts.

Offset to Font ID String (str_offs)

The str_offs field specifies the offset (in bits) from the beginning of the font header to the beginning of a font ID string. The font ID string can provide a brief description of the font (in words) and can be queried with the R_TEXTP opcode (function #7 returns the address of the font ID string). The string may be a maximum of 30 characters, and for the default text service routine, the string would be expected to be byte-packed and NULL-terminated. This field is defined as an offset (relative to the beginning of the font header) so as to make the font structure relocatable - i.e., a user-defined font may be downloaded anywhere in graphics board memory without having to modify any embedded absolute addresses.

Offset to character bit-map data (data_offs)

The data_offs field specifies the offset (in bits) from the beginning of the font header to the beginning of the character bit-map data. The format of the character bit-map data is described below. This field is defined as an offset (relative to the beginning of the font

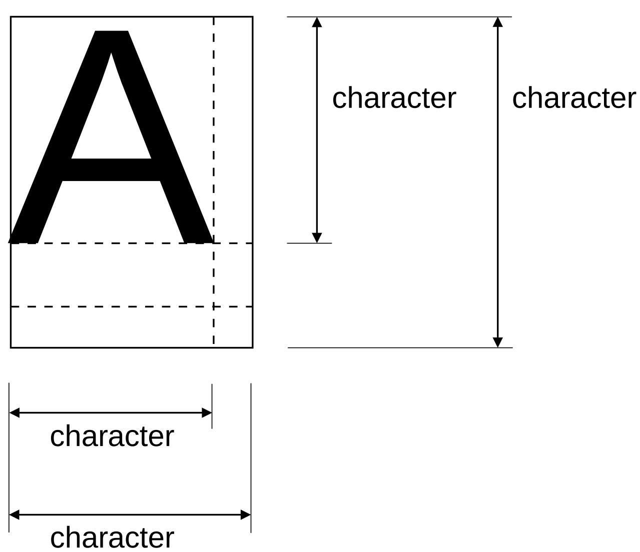


Figure C.1 CHARACTER CELL

Font Definitions (continued)

header) so as to make the font structure relocatable—i.e., a user defined font may be downloaded anywhere in graphics board memory without having to modify any embedded absolute addresses.

Character Cell dimensions (fontX, fontY)

The fontX and fontY fields specify the width and height (in pixels) of the rectangular character cell. The fontX and fontY values are the fundamental parameters describing the font and are used in calculating the derived parameters that make up the latter portion of the font header. Note that these dimensions are that of the enclosing rectangle—the characters defined in a font may not necessarily fill the entire character cell.

Character Cell size (cellsize)

The cellsize field specifies the total linear size (in bits) of a packed character cell in the character bit-map data array—i.e., cellsize=fontX• fontY.

Code Numbers of first and last characters in font (char_min, char_max)

The char_min field specifies the code number of the first character defined in the character bit-map data array. For fonts using ASCII character coding (such as the default fonts) this would typically be SP (space, 20h). The char_max field likewise corresponds to the last defined character. Character definitions in the bit-map data array are assumed to be contiguous, and thus the total number of characters defined in a font is char_max - char_min+1. The default text service routine presumes ASCII coding to the extent that "ctext" processing recognizes some of the ASCII control characters (such as CR and LF).

Code Number of Default Character (char_def)

The char_def field specifies the code number of a default character to be substituted for an invalid character code (i.e., char_code<char_min OR char_code>char_max).

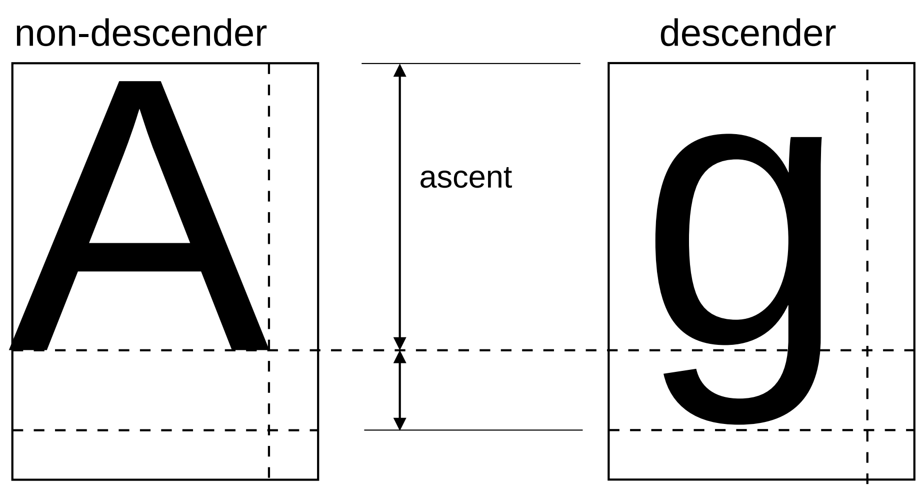


Figure C.2 CHARACTER CELL BASELINE

Font Definitions *(continued)*

Character Cell baseline parameters (ascent, descent)

The ascent and descent fields specify the distance (in pixels) from the character “baseline” (bottom line of an upper-case character) to the first row of the character cell (ascent), and from the baseline to the last row of the character cell (descent). This provides extra information that can be used for vertical justification of characters with or without descenders.

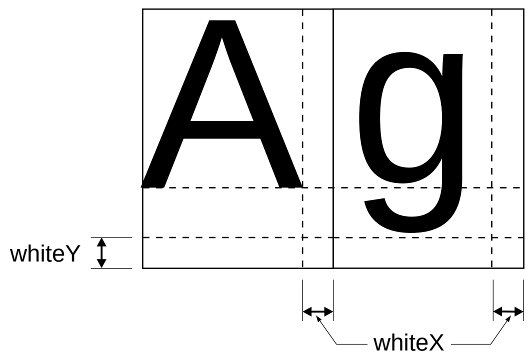


Figure C.3 BUILT-IN CELL SPACING

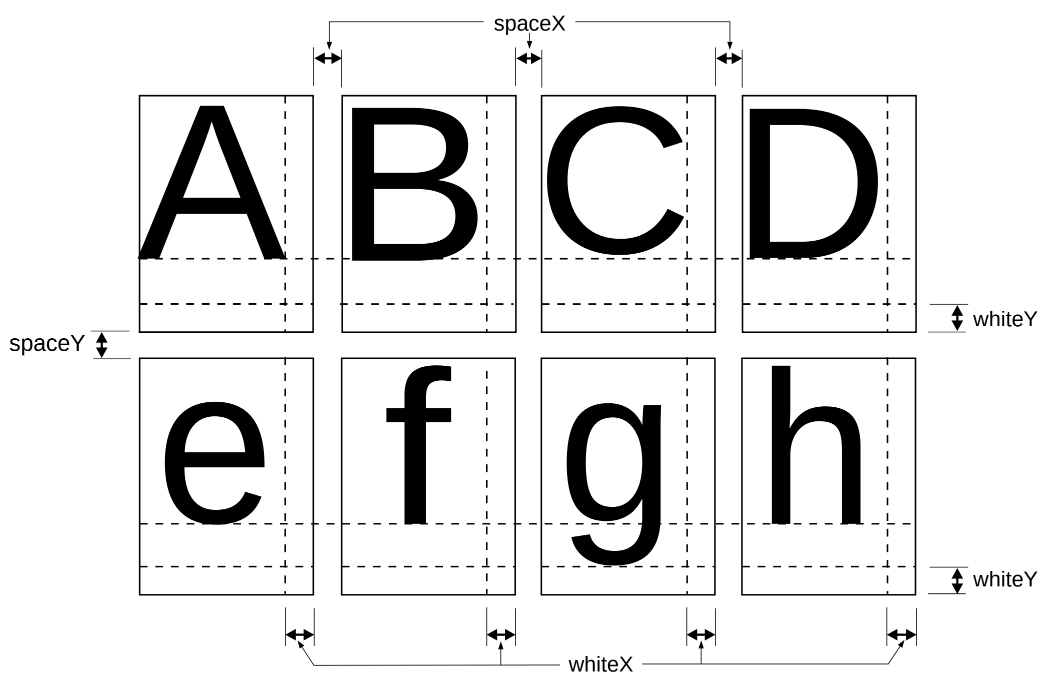


Figure C.4 DEFAULT CELL SPACING

Font Definitions (continued)

The shape of an individual character is defined by a bit-map array of the same dimensions as the character cell (fontX• fontY). Each bit in the array corresponds to a pixel within the character cell. The pixels within the character cell can be numbered from left-to-right, top-to-bottom, contiguously throughout the cell—this corresponds to the numbering of the bits in the bit-map array in order of their bit-addresses. Successive character cells are packed contiguously in the font shape data.

Number of bits per character cell:
Nc=fontX•fontY
Number of characters in font:
N=char_max-char_min+1
Total number of bits comprising font shape data:
Nb=N•Nc=N•fontX•fontY

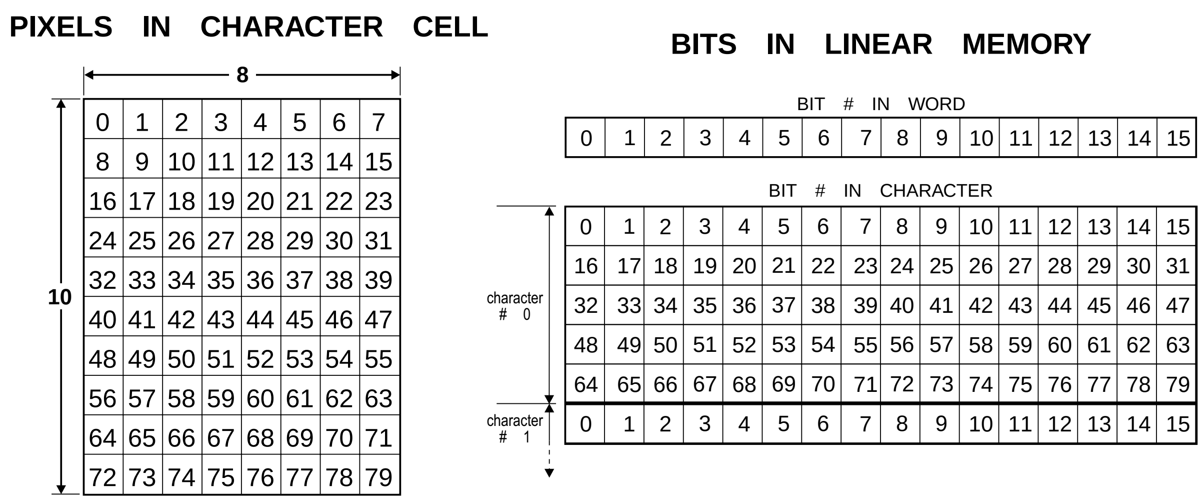


Figure C.5 EXAMPLE 8x10 CHARACTER CELL

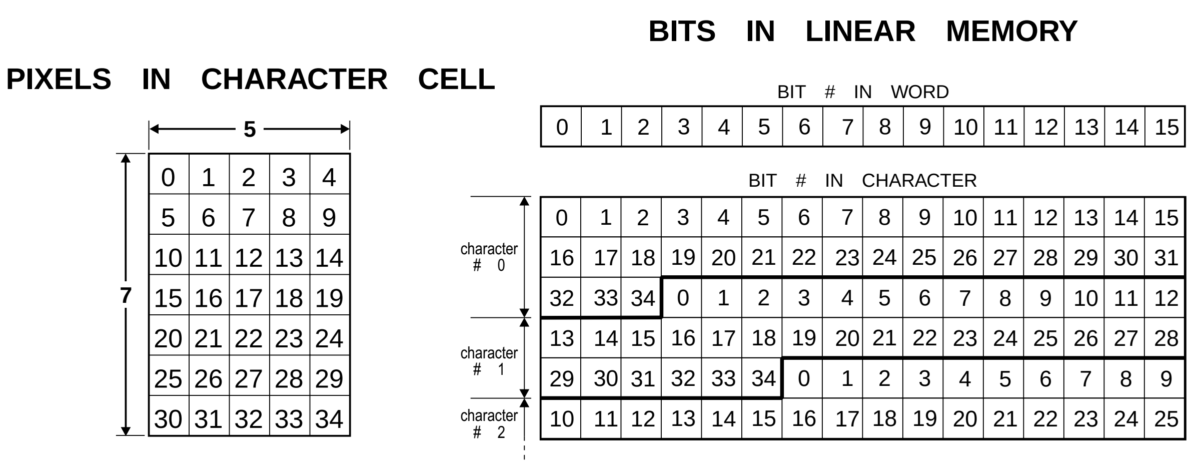


Figure C.6 EXAMPLE 5x7 CHARACTER CELL

Font Definitions *(continued)*

The pixels in the character cell are drawn according to the value of the corresponding bit in the bit-map array (0 or 1) as follows:

BIT	PIXEL	RENDERING
0	off	background color (or transparent)
1	on	foreground color

The pixels in the character cells are viewed in “little-endian” order (LSB on the left), but in order to create the bit-map source data format they must be encoded into hexadecimal words, which are essentially in “big-endian” format. The following examples illustrate this source-format encoding process. Note that for 5x7 characters, successive character cells do not align on word boundaries, whereas 8x10 characters do.

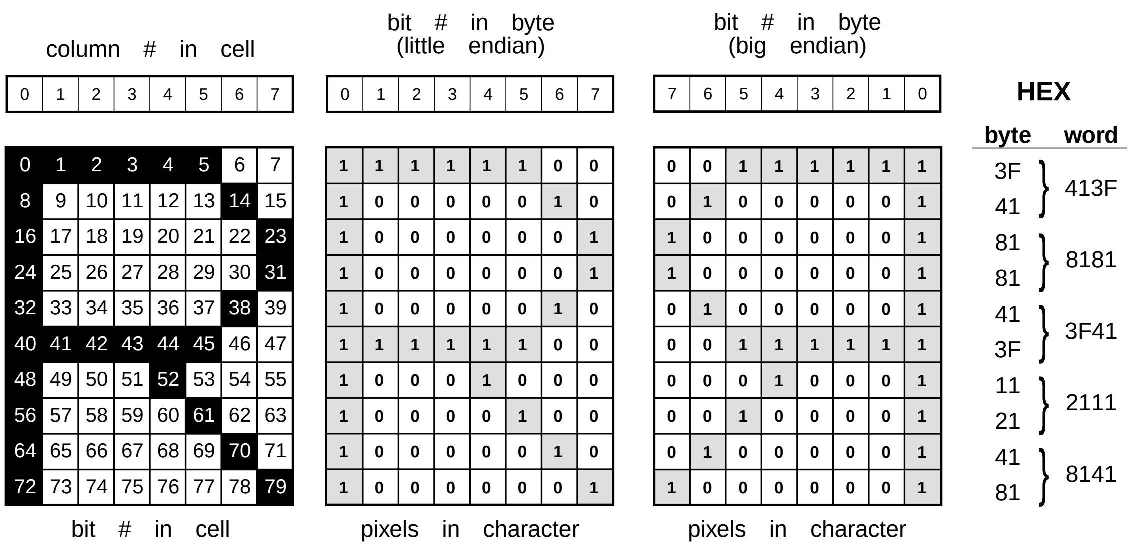


Figure C.7 PIXELS IN THE 8 x 10 CHARACTER CELL

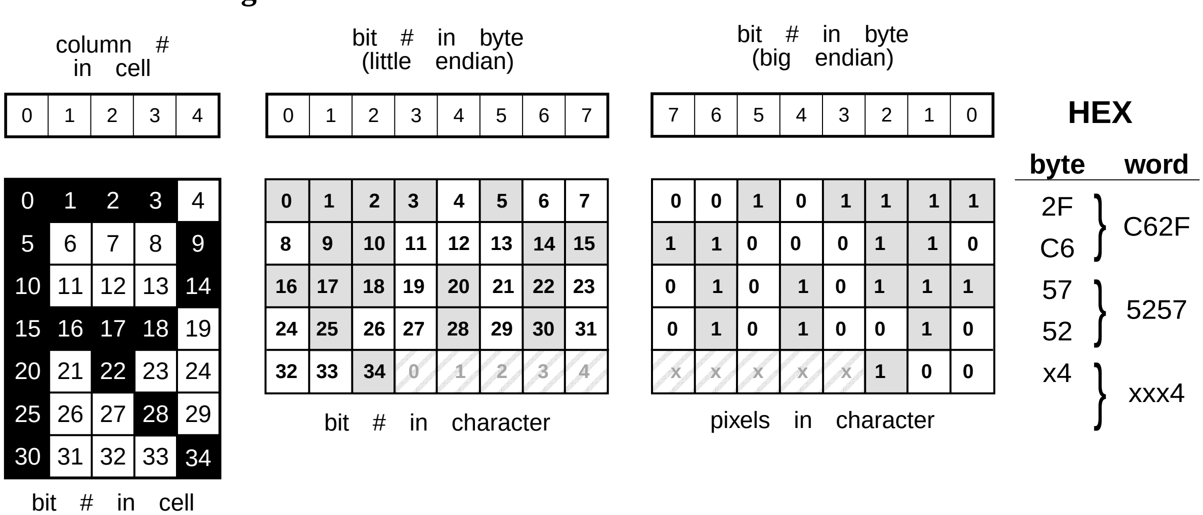


Figure C.8 PIXELS IN THE 5x7 CHARACTER CELL

Font Definitions *(continued)*

Font header include file

The following include file implements the entire font header structure. Symbolic names referenced correspond to the font parameters defined above. The syntax used is that for TMS340 family assembly language, but the file should be easily converted for other assembly languages, or even 'C' language source format.

```
;*************************************************************************
; RGI0FONT.INC - font header for RGI type 0 text
;*************************************************************************
;define the following symbols:
;    font_ID        - font ID # to return
;    fontX          - character cell width (pixels)
;    fontY          - character cell height (pixels)
;    char_min       - code number of first character in font
;    char_max       - code number of last character in font
;    char_def       - code number of default character (substituted for illegal chars)
;    ascent         - top of character cell to baseline (pixels)
;    descent        - bottom of character cell to baseline (pixels)
;    whiteX         - built-in horizontal cell spacing (internal)
;    whiteY         - built-in vertical cell spacing (internal)
;    spaceX         - default horizontal cell spacing (external)
;    spaceY         - default vertical cell spacing (external)
;define the following labels:
;    font_str       - start of font ID string (NULL-terminated, max. 30 chars.)
;    font_data      - start of font bit-map data array
;    font_end       - address of next bit beyond end of font data (last+1)
;note: char_min is also the character code of the first character in the bit-map data
;        base
;*************************************************************************
        ;___________
        ; font header
        ;___________
font_start      ; beginning of header (for length, offset calculations)
        .word   3052h                   ; RGI font type 0 (3052h = "R0")
        .long   (font_end-font_start)/8 ; font length in bytes
        .word   font_ID                 ; font ID #
        .long   font_str-font_start     ; bit offset to ID string (NULL term)
        .long   font_data-font_start    ; bit offset to pixblt data array
        .word   fontX                   ; character cell width (pixels)
        .word   fontY                   ; character cell height (pixels)
        .long   fontX*fontY             ; cellsize (linear)
        .word   char_min                ; code number of first character in font
        .word   char_max                ; code number of last character in font
        .word   char_def                ; code number of default character
        .word   ascent                  ; top of character cell to baseline (pixels)
        .word   descent                 ; bottom of character cell to baseline (pixels)
        .word   0                       ; (padding)
        .word   whiteX                  ; built-in horizontal cell spacing (internal)
        .word   whiteY                  ; built-in vertical cell spacing (internal)
        .word   spaceX                  ; default horizontal cell spacing (external)
        .word   spaceY                  ; default vertical cell spacing (external)
```

Font Header *(continued)*

```
;------------------------------
; derived parameters
;------------------------------

.word    0, 0, 0, fontX-1, fontX-1, fontY-1, fontY-1, 0
.word    fontX-1, 0, 0, 0, 0, fontY-1, fontY-1, fontX-1
.word    0, -(fontY-1), 0, 0, fontX-1, 0, fontY-1, -(fontX-1)
.word    fontX-1, -(fontY-1), 0, -(fontX-1), 0, 0, fontY-1, 0
.word    -(fontX-1), 0, -(fontY-1), fontX-1, 0, fontY-1, 0, 0
.word    0, 0, -(fontY-1), 0, -(fontX-1), fontY-1, 0, fontX-1
.word    0, 0, 0, fontX-1, fontX-1, fontY-1, fontY-1, 0
.word    fontX-1, 0, 0, 0, 0, fontY-1, fontY-1, fontX-1
.word    fontX, fontY, 0, 0, 0, 0, fontX, 0
.word    1, fontX, fontX, 0, 3, fontY, 1, 0
.word    1, fontY, 1, 0, 3, fontX, fontX, 0
.word    1, fontX, fontX, 0, 1, fontY, 1, 0
.word    1, fontY, 1, 0, 1, fontX, fontX, 0
.word    1, fontX, fontX, 0, 1, fontY, 1, 0
.word    fontX, fontY, 0, 0, 2, 0, fontX, 0
.word    1, fontX, fontX, 0, 3, fontY, 1, 0
;------------------------------------------------------
; font ID string, bit-map data array follows
;------------------------------------------------------
```

Font file example

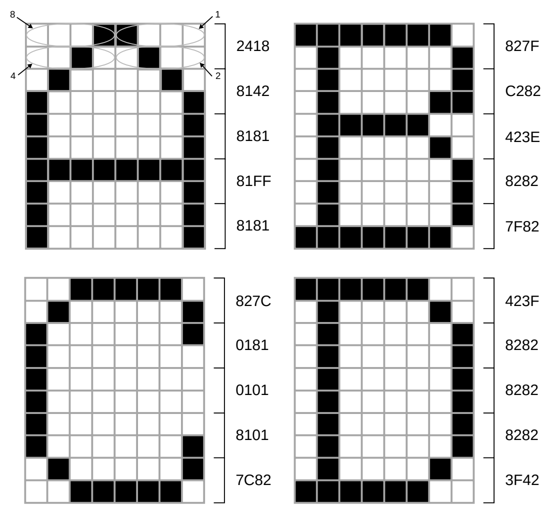
The following example demonstrates the creation of a font with 5 characters. It uses the include file shown above, defining symbolic names for each of the font parameters.

```
;***********************************************************************
; FONT_X.ASM - example 8x10 font
;***********************************************************************
font_ID      .set    0     ; font ID # to return
fontX        .set    8     ; character cell width (pixels)
fontY        .set    10    ; character cell height (pixels)
char_min     .set    40h   ; code number of first character in font
char_max     .set    44h   ; code number of last character in font
char_def     .set    40h   ; code number of default character
ascent       .set    10    ; top of character cell to baseline (pixels)
descent      .set    0     ; bottom of character cell to baseline (pixels)
whiteX       .set    0     ; built-in horizontal cell spacing (internal)
whiteY       .set    0     ; built-in vertical cell spacing (internal)
spaceX       .set    2     ; default horizontal cell spacing (external)
spaceY       .set    6     ; default vertical cell spacing (external)

;------------------------------

font_x       ; font header

             .copy   rgi0font.inc
```

Font File Example (continued)

```
font_str        ; font ID string

                .string    "example 8x10 font"
                .byte       0
                .even

font_data ; font bit-map data

                .word      0000h, 0000h, 0000h, 0000h, 0000h      ; (blank)
                .word      2418h, 8142h, 8181h, 81FFh, 8181H      ; A
                .word      827Fh,0C282h, 423Eh, 8282h, 7F82H      ; B
                .word      827Ch, 0181h, 0101h, 8101h, 7C82H      ; C
                .word      423Fh, 8282h, 8282h, 8282h, 3F42H      ; D

font_end
                .end
```

The character shapes used in the previous example are shown below:



Installing a user-defined font

The following outline illustrates the algorithm for downloading and selecting a user-defined font.

1. Open the font file on the host system and read the “font_len” field to determine the number of bytes required for the font.

2. Allocate a section of memory on the graphics board with the R_ALLOC opcode, passing the length parameter described in step 1. R_ALLOC will return the address of a buffer in graphics board memory. (AFGIS ‘C’ library function: rg_alloc)

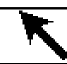
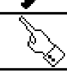
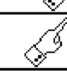
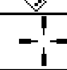
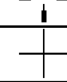
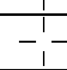
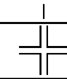
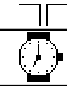
3. Download the font file from the host to the graphics board at the address returned in step 2. (AFGIS ‘C’ library function: rg_blockwrite)

4. Select the downloaded font with the FONT opcode (or TEXTP, function # 0 or 1), passing the address where the font was downloaded to on the graphics board (returned from R_ALLOC in step 2). (AFGIS ‘C’ library function: rg_settextstyle)

AFGIS-3.11 Appendix D

The various bitmaps of the predefined cursors are shown below.

Cursor Number	Cursor Size	Cursor Name	Cursor Bitmap
0	16x16	MSC_left_arrow	
1	16 x16	MSC_arrow	
2	16 x16	MSC_center_ptr	
3	16 x16	MSC_down_center_ptr	
4	16 x16	MSC_double_arrow	
5	16 x16	MSC_lr_double_arrow	
6	16 x16	MSC_fleur	
7	16 x16	MSC_exchange	
8	16 x16	MSC_left_side	
9	16 x16	MSC_right_side	
10	16 x16	MSC_top_side	
11	16 x16	MSC_bottom_side	
12	16 x16	MSC_top_left_corner	
13	16 x16	MSC_top_right_corner	
14	16 x16	MSC_bottom_left_corner	
15	16 x16	MSC_bottom_right_corner	
16	16 x16	MSC_sb_left_arrow	
17	16 x16	MSC_sb_right_arrow	
18	16 x16	MSC_sb_up_arrow	
19	16 x16	MSC_sb_down_arrow	
20	16 x16	MSC_sb_h_double_arrow	
21	16 x16	MSC_sb_v_double_arrow	
22	16 x16	MSC_circle	
23	16 x16	MSC_target	
24	16 x16	MSC_cross	
25	16 x16	MSC_crosshair	
26	16 x16	MSC_plus	
27	16 x16	MSC_tcross	

28	16 x16	MSC_left_hand1	
29	16 x16	MSC_hand1	
30	16 x16	MSC_hand2	
31	16 x16	MSC_right_hand2	
32	16 x16	MSC_leftbutton	
33	16 x16	MSC_middlebutton	
34	16 x16	MSC_rightbutton	
35	16 x16	MSC_xterm	
36	16 x16	MSC_watch	
37	16 x16	MSC_pencil	
38	16 x16	MSC_gumby	
39	16x21	MSC_hour_glass16	
40	32x32	MSC_NW_arrow32	
41	32x32	MSC_NE_arrow32	
42	32x32	MSC_NW_hand32	
43	32x32	MSC_NE_hand32	
44	32x32	MSC_xhair32a	
45	32x32	MSC_xhair32b	
46	32x32	MSC_xhair32c	
47	32x32	MSC_xhair32d	
48	32x32	MSC_watch32	
49	32x32	MSC_hour_glass32