

GenRef  
v1.00

MDOS Reference guide.

Video Library

(C) Copyright 2004  
Beery W. Miller  
**ALL RIGHTS RESERVED**

---

**Video - CONTENTS**


---

VIDEO OVERVIEW.....	4
VIDEO MODES.....	4
VIDEO REGISTERS.....	5
TEXT MODES.....	7
TEXT1.....	7
TEXT2A.....	7
TEXT2B.....	8
MULTICOLOR.....	8
MULTICOLOR.....	8
GRAPHIC MODES.....	9
GRAPHIC1.....	9
GRAPHIC2.....	9
GRAPHIC3.....	10
GRAPHIC4.....	10
GRAPHIC5.....	11
GRAPHIC6.....	11
GRAPHIC7.....	12
VIDEO PAGE CONCEPTS.....	13
VIDEO SPRITES.....	14
SPRITE MODE 1.....	14
SPRITE MODE 2.....	14
VIDEO COLOR TABLE.....	15
CALLING VIDEO FUNCTIONS.....	16
SET VIDEO MODE.....	17
GET VIDEO MODE.....	18
SET CURSOR POSITION.....	19
GET CURSOR POSITION.....	20
SET DISPLAY PAGE.....	21
GET DISPLAY PAGE.....	22
SCROLL WINDOW UP.....	23
SCROLL WINDOW DOWN.....	24
SCROLL WINDOW LEFT.....	25
SCROLL WINDOW RIGHT.....	26
CALL SCREEN.....	27
GET CHARACTER COLOR.....	28
SET BORDER COLOR.....	29
SET COLOR PALETTE.....	30
SET PIXEL COLOR.....	31
GET PIXEL COLOR.....	32
SET VECTOR COLOR.....	33
COLOR SEARCH.....	34
HIGH SPEED BLOCK MOVE.....	35
HIGH SPEED BLOCK COPY.....	36
LOGICAL BLOCK COPY.....	38

GenREF V1.00

BLOCK SCROLL UP.....	39
BLOCK SCROLL DOWN .....	40
BLOCK SCROLL LEFT.....	41
BLOCK SCROLL RIGHT.....	42
SPRITE DEFINE .....	43
DELETE SPRITE.....	44
LOCATE SPRITE .....	45
SPRITE MOTION .....	46
SPRITE COLOR .....	47
DEFINE SPRITE PATTERN .....	48
SPRITE MAGNIFY.....	49
SPRITE PATTERN / DISTANCE.....	50
SPRITE COINCIDENCE .....	51
SPRITE PATTERN DEFINE / GET .....	52
CHARACTER PATTERN DEFINE/GET .....	53
SET TEXT WINDOW .....	54
GET TEXT WINDOW.....	55
WRITE TTY .....	56
RESTORE CHARACTER / SPRITE PATTERN.....	58
SET TEXT COLOR.....	59
WRITE CHARACTER STRING .....	60
PRINT SCREEN.....	61
HORIZONTAL CHARACTER / COLOR (HCHAR).....	62
VERTICAL CHARACTER / COLOR (VCHAR).....	63
HORIZONTAL CHARACTER (HCHAR) .....	64
VERTICAL CHARACTER (VCHAR) .....	65
SET MOUSE .....	66
GET MOUSE.....	67
GET MOUSE RELEASE.....	68
MAKE SOUND.....	69
Musical Tone Frequencies .....	70
SOUND STATUS .....	71
VWTR .....	72
VRFR .....	73
GET TABLES .....	74
GET PALETTE REGISTERS .....	75

---

**VIDEO - OVERVIEW**


---

The video management routines in MDOS are provided to aid a programmer in writing applications requiring video input and output. The V9938 processor contained aboard the Geneve 9640 is upward compatible with the existing TMS9918A used in the TI-99/4A. The V9938 was developed through the joint efforts of ASCII Corporation, Microsoft Inc., and YAMAHA.

The following functions are supported on the V9938.

- Full bit-mapped mode
- 80-column text display
- Access using x- and y- coordinates independent of the screen mode.
- Hardware commands internal the V9938 including AREA MOVE, LINE, SEARCH, RASTER OPERATION, etc.
- More sprites per horizontal line than the TMS9918A.
- Maximum 512 x 424 pixels, 16 colors
- Bit-mapped graphics
- Interfaces the bus mouse
- Maximum 8 sprites per horizontal line
- Logical operation function
- And more

---

**VIDEO MODES**


---

The video management routines in MDOS support 11 different modes of display. The following table describes those modes and their properties.

<b>Mode</b>	<b>Size</b>	<b>Video Mode #</b>	<b>Colors</b>	<b>Sprite Mode</b>	<b>Number of Display Pages</b>
TEXT1	40 x 24	>0000	2	N	32
TEXT2A	80 x 24	>0001	2	N	16
MULTICOLOR	64 x 48	>0002	16	1	32
GRAPHIC1	32 x 24	>0003	16	1	32
GRAPHIC2	32 x 24	>0004	16	1	8
GRAPHIC3	256 x 212	>0005	16	2	8
GRAPHIC4	512 x 212	>0006	16	2	4
GRAPHIC5	512 x 212	>0007	4	2	4
GRAPHIC6	512 x 212	>0008	16	2	2
GRAPHIC7	256 x 212	>0009	256	2	2
TEXT2B	80 x 26.5	>000A	2	N	16

---

**VIDEO REGISTERS**


---

The V9938 uses 49 internal registers for its screen operations. These registers are referred to as "VDP registers" in this book. VDP registers are classified by function into three groups as described below. The control register group and status register group can be referred to using VDP(n) system variables. By using the MDOS Video XOP functions described later in this manual, we do not need to be concerned with directly accessing these registers. Further information on direct programming of the V9938 can be obtained by obtaining a copy of the V9938 Technical Manual available on <ftp.whitech.com> as a PDF file or on <http://map.tni.nl/> as either a PDF file or HTML document.

**Control register group (R#0 to R#23, R#32 to R#46)**

This is a read-only 8-bit register group controlling V9938 actions. Registers are expressed using the notation R#n. R#0 to R#23 are used to set the screen mode. R#32 to R#46 are used to execute VDP commands. These VDP commands will be described in detail later. Control registers R#24 to R#31 do not exist. The roles of the different control registers are listed below.

Control register list

R#n	Corres- ponding VDP(n)	Function
R#0	VDP(0)	mode register #0
R#1	VDP(1)	mode register #1
R#2	VDP(2)	pattern name table
R#3	VDP(3)	colour table (LOW)
R#4	VDP(4)	pattern generator table
R#5	VDP(5)	sprite attribute table (LOW)
R#6	VDP(6)	sprite pattern generator table
R#7	VDP(7)	border colour/character colour at text mode
R#8	VDP(9)	mode register #2
R#9	VDP(10)	mode register #3
R#10	VDP(11)	colour table (HIGH)
R#11	VDP(12)	sprite attribute table (HIGH)
R#12	VDP(13)	character colour at text blinks
R#13	VDP(14)	blinking period
R#14	VDP(15)	VRAM access address (HIGH)
R#15	VDP(16)	indirect specification of S#n
R#16	VDP(17)	indirect specification of P#n
R#17	VDP(18)	indirect specification of R#n
R#18	VDP(19)	screen location adjustment (ADJUST)
R#19	VDP(20)	scanning line number when the interrupt occurs
R#20	VDP(21)	colour burst signal 1
R#21	VDP(22)	colour burst signal 2
R#22	VDP(23)	colour burst signal 3
R#23	VDP(24)	screen hard scroll

## GenREF V1.00

R#n	Corres- ponding VDP(n)	Function
R#32	VDP(33)	SX: X-coordinate to be transferred (LOW)
R#33	VDP(34)	SX: X-coordinate to be transferred (HIGH)
R#34	VDP(35)	SY: Y-coordinate to be transferred (LOW)
R#35	VDP(36)	SY: Y-coordinate to be transferred (HIGH)
R#36	VDP(37)	DX: X-coordinate to be transferred to (LOW)
R#37	VDP(38)	DX: X-coordinate to be transferred to (HIGH)
R#38	VDP(39)	DY: Y-coordinate to be transferred to (LOW)
R#39	VDP(40)	DY: Y-coordinate to be transferred to (HIGH)
R#40	VDP(41)	NX: num. of dots to be transferred in X direction (LOW)
R#41	VDP(42)	NX: num. of dots to be transferred in X direction (HIGH)
R#42	VDP(43)	NY: num. of dots to be transferred in Y direction (LOW)
R#43	VDP(44)	NY: num. of dots to be transferred in Y direction (HIGH)
R#44	VDP(45)	CLR: for transferring data to CPU
R#45	VDP(46)	ARG: bank switching between VRAM and expanded VRAM
R#46	VDP(47)	CMR: send VDP command

### Status register (S#0 to S#9)

This is a read-only 8-bit register group which reads data from the V9938. Registers are expressed using the notation S#n. The functions of the registers are listed below.

### Status register list

S#n	Corres- ponding VDP(n)	Function
S#0	VDP(8)	interrupt information
S#1	VDP(-1)	interrupt information
S#2	VDP(-2)	DP command control information/etc.
S#3	VDP(-3)	coordinate detected (LOW)
S#4	VDP(-4)	coordinate detected (HIGH)
S#5	VDP(-5)	coordinate detected (LOW)
S#6	VDP(-6)	coordinate detected (HIGH)
S#7	VDP(-7)	data obtained by VDP command
S#8	VDP(-8)	X-coordinate obtained by search command (LOW)
S#9	VDP(-9)	X-coordinate obtained by search command (HIGH)

### Colour palette register group (P#0 to P#15)

These registers are used to set the colour palette. Registers are expressed using the notation P#n where 'n' is the palette number which represents one of 512 colours. Each palette register has 9 bits allowing three bits to be used for each RGB colour (red, green, and blue).

## Text Modes

### Video Mode – TEXT1

<b>Characteristics</b>	Pattern Size	6 dots (w) x 8 dots (h)
	Patterns	256 types
	Screen pattern count	40 (w) x 24 (h) patterns
	Pattern colors	Two colors out of 512 colors (per screen)
	VRAM area per screen	4K
<b>Controls</b>	Pattern font	VRAM pattern generator table
	Screen pattern location	VRAM pattern name table
	Pattern color code 1	High-order four bits of R#7
	Pattern color code 0	Low-order four bits of R#7
	Background color code	Low-order four bits of R#7

The area in which character fonts are stored is called the pattern generator table. This table is located in VRAM, and, although the font is defined by using 8 bytes for each character from the top of the table, the 2 low order bits of each byte representing the right two columns are not displayed on the screen. Thus, the size of one character is 6 x 8 pixels. Each character font set contains 256 different characters numbered from 0 to 255.

### Video Mode – TEXT2A

<b>Characteristics</b>	Pattern Size	6 dots (w) x 8 dots (h)
	Patterns	256 types
	Screen pattern count	80 (w) x 24 (h) patterns
	Pattern blinking	Possible for each character
	Pattern colors	Two colors out of 512 colors (per screen) Four if using blinking
	VRAM area per screen	8K
<b>Controls</b>	Pattern font	VRAM pattern generator table
	Screen pattern location	VRAM pattern name table
	Blink attributes	VRAM color table
	Pattern color code 1	High-order four bits of R#7
	Pattern color code 0	Low-order four bits of R#7
	Background color code	Low-order four bits of R#7
	Pattern color code 1	High-order four bits of R#12 (Used for blinking)
Pattern color code 0	Low-order four bits of R#12 (Used for blinking)	

The area in which character fonts are stored is called the pattern generator table. This table is located in VRAM, and, although the font is defined by using 8 bytes for each character from the top of the table, the 2 low order bits of each byte representing the right two columns are not displayed on the screen. Thus, the size of one character is 6 x 8 pixels. Each character font set contains 256 different characters numbered from 0 to 255.

<b>Video Mode – TEXT2B</b>
----------------------------

<b>Characteristics</b>	Pattern Size Patterns Screen pattern count Pattern blinking Pattern colors  VRAM area per screen	6 dots (w) x 8 dots (h) 256 types 80 (w) x 26.5 (h) patterns Possible for each character Two colors out of 512 colors (per screen) Four if using blinking 8K
<b>Controls</b>	Pattern font Screen pattern location Blink attributes Pattern color code 1 Pattern color code 0 Background color code Pattern color code 1  Pattern color code 0	VRAM pattern generator table VRAM pattern name table VRAM color table High-order four bits of R#7 Low-order four bits of R#7 Low-order four bits of R#7 High-order four bits of R#12 (Used for blinking) Low-order four bits of R#12 (Used for blinking)

The area in which character fonts are stored is called the pattern generator table. This table is located in VRAM, and, although the font is defined by using 8 bytes for each character from the top of the table, the 2 low order bits of each byte representing the right two columns are not displayed on the screen. Thus, the size of one character is 6 x 8 pixels. Each character font set contains 256 different characters numbered from 0 to 255.

## MultiColor Mode

<b>Video Mode – MULTICOLOR</b>
--------------------------------

<b>Characteristics</b>	Screen composition Color blocks Sprite mode VRAM area per screen	64 (w) x 48 (h) color blocks 16 colors out of 512 colors Sprite Mode 1 4K bytes
<b>Controls</b>	Color block color code Color block location Background color code Sprites	VRAM pattern generator table VRAM pattern name table Low-order four bits of R#7 VRAM sprite attribute table VRAM sprite pattern table

The pattern generator table is an area that stores the colors of the color blocks. Each pattern is made up of four color blocks. These patterns are approximately 8 x 8 when the dots available for the screen display area is 256 x 192 dots. In MULTICOLOR mode, two bytes are used for each pattern, and each pattern includes four color blocks.



## Graphic Modes

### Video Mode – GRAPHIC1

<b>Characteristics</b>	Pattern Size	8 dots (w) x 8 dots (h)
	Patterns	256 types
	Screen pattern count	32 (w) x 24 (h) patterns
	Pattern colors	16 colors out of 512 colors (per screen)
	Sprite mode	Sprite Mode 1
	VRAM area per screen	4K
<b>Controls</b>	Pattern font	VRAM pattern generator table
	Screen pattern location	VRAM pattern name table
	Pattern color codes 1 & 0	Can be specified as a group for each 8-pattern set, in the VRAM color table
	Background color code	Low-order four bits of R#7
	Sprites	VRAM sprite attribute table, VRAM sprite Pattern table

The pattern generator table is an area that stores the pattern fonts. Each pattern has a number from 0 to 255. The font for each pattern is constructed from 8 bytes. The pattern name table is composed of one byte for each screen pattern. Each byte specifies a unique pattern.

### Video Mode – GRAPHIC2

<b>Characteristics</b>	Pattern Size	8 dots (w) x 8 dots (h)
	Patterns	768 types
	Screen pattern count	32 (w) x 24 (h) patterns
	Pattern colors	16 colors out of 512 colors (per screen)
	Sprite mode	Sprite Mode 1
	VRAM area per screen	16K
<b>Controls</b>	Pattern font	VRAM pattern generator table
	Screen pattern location	VRAM pattern name table
	Pattern color codes 1 & 0	Can be specified as a group for each raster, in the VRAM color table
	Background color code	Low-order four bits of R#7
	Sprites	VRAM sprite attribute table, VRAM sprite Pattern table

The pattern generator table is an area that stores the pattern fonts. Each pattern group has a number from 0 to 255; and since each group may have three members, 768 patterns may be specified. The font for each pattern is constructed from 8 bytes.

<b>Video Mode – GRAPHIC3</b>
------------------------------

<b>Characteristics</b>	Pattern Size	8 dots (w) x 8 dots (h)
	Patterns	768 types
	Screen pattern count	32 (w) x 24 (h) patterns
	Pattern colors	16 colors out of 512 colors (per screen)
	Sprite mode	Sprite Mode 2
	VRAM area per screen	16K
<b>Controls</b>	Pattern font	VRAM pattern generator table
	Screen pattern location	VRAM pattern name table
	Pattern color codes 1 & 0	Can be specified as a group for each raster, in the VRAM color table
	Background color code	Low-order four bits of R#7
	Sprites	VRAM sprite attribute table, VRAM sprite Pattern table

The pattern generator table is an area that stores the pattern fonts. Each pattern group has a number from 0 to 255; and since each group may have three members, 768 patterns may be specified. The font for each pattern is constructed from 8 bytes.

<b>Video Mode – GRAPHIC4</b>
------------------------------

<b>Characteristics</b>	Bit-mapped Graphics Mode	
	Screen Size	256 (w) x 212 (h) dots 256 (w) x 192 (h) dots
	Screen colors	16 colors out of 512 colors (per screen)
	Sprite mode	Sprite Mode 2
	VRAM area per screen	32K
<b>Controls</b>	Graphics	VRAM pattern name table
	Background color code	Low-order four bits of R#7
	Sprites	VRAM sprite attribute table, VRAM sprite Pattern table

The pattern name table is composed of one byte for every two dots on the screen. A color can be assigned for each dot from a selection of 16 colors out of 512 colors.

<b>Video Mode – GRAPHIC5</b>
------------------------------

<b>Characteristics</b>	Bit-mapped Graphics Mode	
	Screen Size	512 (w) x 212 (h) dots 512 (w) x 192 (h) dots
	Screen colors	4 colors out of 512 colors (per screen)
	Sprite mode	Sprite Mode 2
	VRAM area per screen	32K
<b>Controls</b>	Graphics	VRAM pattern name table
	Background color code	Low-order four bits of R#7
	Sprites	VRAM sprite attribute table, VRAM sprite Pattern table

The pattern name table is composed of one byte for every four dots on the screen. A color can be assigned for each dot from a selection of 4 colors out of 512 colors. A hardware tiling function processes the sprite and background colors. For these colors, you can specify four bits; however, of these four bits, the higher-order two bits specify the color code of the even dots, and the lower-order two bits specify the color code of the odd dots of the x-coordinate (0 to 511). The size of one dot of a sprite is approximately twice that of a graphic dot; however, when this tiling function is used, one dot of a sprite may be displayed in two colors. The even and odd dots of the background color may also be specified in the same manner.

<b>Video Mode – GRAPHIC6</b>
------------------------------

<b>Characteristics</b>	Bit-mapped Graphics Mode	
	Screen Size	512 (w) x 212 (h) dots 512 (w) x 192 (h) dots
	Screen colors	16 colors out of 512 colors (per screen)
	Sprite mode	Sprite Mode 2
	VRAM area per screen	128K (Two screens)
<b>Controls</b>	Graphics	VRAM pattern name table
	Background color code	Low-order four bits of R#7
	Sprites	VRAM sprite attribute table, VRAM sprite Pattern table

The pattern name table is composed of one byte for every two dots on the screen. A color can be assigned for each dot from a selection of 16 colors out of 512 colors. .

<b>Video Mode – GRAPHIC7</b>
------------------------------

<b>Characteristics</b>	Bit-mapped Graphics Mode	
	Screen Size	256 (w) x 212 (h) dots 256 (w) x 192 (h) dots
	Screen colors	256 colors (per screen)
	Sprite mode	Sprite Mode 2
	VRAM area per screen	128K (Two screens)
<b>Controls</b>	Graphics	VRAM pattern name table
	Background color code	Low-order four bits of R#7
	Sprites	VRAM sprite attribute table, VRAM sprite Pattern table

The pattern name table is composed of one byte for every dot on the screen. A color can be assigned for each dot from a selection of 256 colors. .

**VIDEO PAGE CONCEPTS**

The parameters used for the V9938 are all x-y coordinates. In other words, the internal command processor of the V9938 accesses the entire VRAM area as x-y coordinates of the display mode.

When a screen is to be displayed, 212 lines of the same page are displayed. Use the Set Display Page Video Opcode >04 to select the page. The display modes and their relationships to the coordinates are shown in the table below.

**GRAPHIC4**

(0,0)		(255,0)
(0,255)	Page 0	(255,255)
(0,256)		(255,256)
(0,511)	Page 1	(255,511)
(0,512)		(255,512)
(0,767)	Page 2	(255,767)
(0,768)		(255,768)
(0,1023)	Page 3	(255,1023)

**GRAPHIC5**

(0,0)		(511,0)
(0,255)	Page 0	(511,255)
(0,256)		(511,256)
(0,511)	Page 1	(511,511)
(0,512)		(511,512)
(0,767)	Page 2	(511,767)
(0,768)		(511,768)
(0,1023)	Page 3	(511,1023)

**GRAPHIC7**

(0,0)		(255,0)
(0,255)	Page 0	(255,255)
(0,256)		(255,256)
(0,511)	Page 1	(255,511)

**GRAPHIC6**

(0,0)		(511,0)
(0,255)	Page 0	(511,255)
(0,256)		(511,256)
(0,511)	Page 1	(511,511)

## Video Sprites

### VIDEO SPRITE MODES

#### Sprite Mode 1

In SPRITE MODE 1, there are 32 sprites, numbered >00 to >1F. The sprites assigned the lower numbers have a higher priority. On a single CRT horizontal line, up to **4 sprites** with the highest priority are displayed, and the overlapping portions of sprites with lower priorities are not displayed.

When two sprites collide (their pattern color 1 portions have overlapped), this condition can be detected.

#### Characteristics

Sprite size	8 x 8 dots (normal)
	16 x 16 dots (magnified)
Number of sprites	32 sprites

#### Sprite Mode 2

In SPRITE MODE 2, there are 32 sprites, numbered >00 to >1F. The sprites assigned the lower numbers have a higher priority. On a single CRT horizontal line, up to **8 sprites** with the highest priority are displayed, and the overlapping portions of sprites with lower priorities are not displayed.

When two sprites collide (their pattern color 1 portions have overlapped), this condition can be detected.

The colors of the sprite may be specified for each horizontal line.

#### Characteristics

Sprite size	8 x 8 dots (normal)
	16 x 16 dots (magnified)
Number of sprites	32 sprites

<b>VIDEO COLOR TABLE</b>
--------------------------

Color Name	Code	Palette Register Settings		
		Red	Green	Blue
Transparent	>00	0	0	0
Black	>01	0	0	0
Medium Green	>02	1	6	1
Light Green	>03	3	7	3
Dark Blue	>04	1	1	7
Light Blue	>05	2	3	7
Dark Red	>06	5	1	1
Cyan	>07	2	6	7
Medium Red	>08	7	1	1
Light Red	>09	7	3	3
Dark Yellow	>0A	6	6	1
Light Yellow	>0B	6	6	4
Dark Green	>0C	1	4	1
Magenta	>0D	6	2	5
Gray	>0E	5	5	5
White	>0F	7	7	7

---

**CALLING VIDEO FUNCTIONS**


---

The MDOS Video Library must be called from within a machine code program running as a task under MDOS. You pass arguments to the Video Library via the calling registers.

The MDOS Video Library is invoked from a machine code program when software trap number zero (XOP 0) is called with a library number of 6. The calling program's R0 must contain the 16-bit subprogram at the time of the XOP. The following code fragment will set the video mode to 80 x 24 column text mode and write "This is a sample text string <CR/LF>" to the screen before exiting back to the prompt.

```

LI      R0,>0000    Set Video Mode
LI      R1,>0001    Text 2 mode
XOP     @SIX,0      Access subprogram

LI      R0,>0027    Write Text Routine
LI      R1,STR1     String to write
CLR     R2          Null terminated string
XOP     @SIX,0      Access subprogram

BLWP   @0           Exit

SIX     DATA >0006
STR1    TEXT "This is a sample text string"
        BYTE >0D,>0A,0  CR/LF/Null terminated
        EVEN

```

In the preceding example, three hidden assumptions were made. First it is assumed that STR1 is located on a page which is currently mapped into a memory page which has the same 16-bit address page number as its Virtual address page number (read the section on Memory Management.) The second assumption is that SIX is actually at the virtual address SIX, not in some overlay segment with a different virtual address.



## Video Mode Library

### Set Video Mode

#### Function

Sets video mode and returns the current border color. For bitmap modes, the background color will be the border color in effect at time of call. This subprogram enables you to select the graphics or text mode that offers you the combination of text and/or graphics capabilities that best suits the particular needs of your program.

When you call the subprogram, the following occurs:

- Clears the entire screen
- Restores the default character definitions of all characters
- Restores the default foreground color and background color of all characters.
- Restores the default screen color.
- Deletes all sprites.
- Resets all sprites.
- Resets the sprite magnification level to 1
- Restores the default current position (X=0, Y=0)

#### Parameters

R0x = >0000  
R1x = Video Mode

#### Results

None

#### Parameter Description

Video Mode

Mode	Size	Video Mode #
TEXT1	40 x 24	>0000
TEXT2A	80 x 24	>0001
MULTICOLOR	64 x 48	>0002
GRAPHIC1	32 x 24	>0003
GRAPHIC2	32 x 24	>0004
GRAPHIC3	256 x 212	>0005
GRAPHIC4	512 x 212	>0006
GRAPHIC5	512 x 212	>0007
GRAPHIC6	512 x 212	>0008
GRAPHIC7	256 x 212	>0009
TEXT2B	80 x 26.5	>000A

**Get Video Mode**

**Function** Returns the video mode presently and parameters of the video mode.

**Parameters** R0x = >0001

**Results**

- R0x = Returned video mode
- R1x = Number of columns
- R2x = Number of rows
- R3x = Number of Graphic columns (pixels)
- R4x = Number of Graphical rows (pixels)
- R5x = Current page offset (in pixel rows, use for chip commands)
- R6x = Color of screen border
- R7h = Foreground color of text
- R7l = Background color of text

**Parameter Description**

**Set Cursor Position**

**Function**                      Sets the current cursor position to the designated location on the display screen.

**Parameters**                    R0x = >0002  
                                     R1x = Row Number  
                                     R2x = Column Number

**Results**                        None

**Parameter Description**

**Get Cursor Position**

**Function** Gets the current position of the cursor on the display screen.

**Parameters** R0x = >0003

**Results** R0x = Returned row number  
R1x = Returned column number

**Parameter Description**

**Set Display Page**

**Function**                      Set's the current display page for the video mode presently being used.

**Parameters**                      R0x = >0004  
   R1x = Page number

**Results**                              None

**Parameter Description**

**Get Display Page**

**Function** Returns the current display page number.

**Parameters** R0x = >0005

**Results** R0x = Display page number

**Parameter Description**

<b>Scroll Window Up</b>
-------------------------

**Function**                      Scrolls the currently defined window up a defined number of lines.

**Parameters**                    R0x = >0006  
R1x = Number of lines to scroll  
R2x = Row number of upper left corner  
R3x = Column number of upper left corner  
R4x = Row number of lower right corner  
R5x = Column number of lower right corner  
R6x = Character value for blank lines  
R7h = Foreground color for blank lines  
R7l = Background color for blank lines

**Results**                        None

**Parameter Description**

**Scroll Window Down**

**Function**                      Scrolls the currently defined window down a defined number of lines.

**Parameters**                    R0x = >0007  
R1x = Number of lines to scroll  
R2x = Row number of upper left corner  
R3x = Column number of upper left corner  
R4x = Row number of lower right corner  
R5x = Column number of lower right corner  
R6x = Character value for blank lines  
R7h = Foreground color for blank lines  
R7l = Background color for blank lines

**Results**                        None

**Parameter Description**



**Scroll Window Left**

**Function**                      Scrolls the currently defined window left a defined number of lines.

**Parameters**                    R0x = >0008  
R1x = Number of lines to scroll  
R2x = Row number of upper left corner  
R3x = Column number of upper left corner  
R4x = Row number of lower right corner  
R5x = Column number of lower right corner  
R6x = Character value for blank lines  
R7h = Foreground color for blank lines  
R7l = Background color for blank lines

**Results**                        None

**Parameter Description**

**Scroll Window Right**

**Function**                      Scrolls the currently defined window right a defined number of lines.

**Parameters**                    R0x = >0009  
R1x = Number of lines to scroll  
R2x = Row number of upper left corner  
R3x = Column number of upper left corner  
R4x = Row number of lower right corner  
R5x = Column number of lower right corner  
R6x = Character value for blank lines  
R7h = Foreground color for blank lines  
R7l = Background color for blank lines

**Results**                        None

**Parameter Description**

**Call Screen**

**Function** Set's the background and foreground color of the currently displayed screen. The screen color is the color of the border and the color displayed when transparent is specified as the foreground or background color of a character or pixel.

**Parameters** R0x = >000A  
R1h = Foreground color (if R3 = 0)  
R1l = Background color  
R2x = Character Set # (if mode 3)  
R3x = Flag (0 is change foreground, !0 is leave it alone)

**Results** None

**Parameter Description**

Background color A numeric expression whose value specifies a screen color from among the 16 available colors.

Foreground color A numeric expression whose value specifies a screen color from among the 16 available colors.

**Get Character Color**

**Function** Returns the character and the foreground and background color from a defined position on the screen.

**Parameters** R0x = >000B  
R1x = Row  
R2x = Column

**Results** R0x = ASCII character read from screen  
R1h = Foreground color for character  
R1l = Background color for character

**Parameter Description**

**Set Border Color**

**Function**                      Set's the border color for the screen

**Parameters**                      R0x = >000C  
   R1x = Color to render border

**Results**                              None

**Parameter Description**

**Set Color Palette**

**Function**                      Set's the color Palette

**Parameters**                      R0x = >000D  
   R1x = Palette register number  
   R2x = Color to put into palette register  
  
   R2: = xxxx | xGGG | xBBB | xRRR as a 16 bit word

**Results**                              None

**Parameter Description**

<b>Set Pixel Color</b>
------------------------

**Function** Set Pixel Color of a point on a graphic mode screen with a logical operation done on the data of the dot tht is already displayed.

**Parameters** R0x = >000E  
 R1x = X coordinate of pixel  
 R2x = Y coordinate of pixel  
 R3h = Foreground color to render pixel  
 R3l = Background color to render pixel in graphics mode 2 and 3.  
 R4h = Logic operation to be performed

**Results** None

**Parameter Description**

Logic Operation

**Summary of Logical Operations**

Name	Operation	A3	A2	A1	A0
IMP	DC = SC	0	0	0	0
AND	DC = SC*DC	0	0	0	1
OR	DC = SC+DC	0	0	1	0
EOR	DC = !SC*DC +SC*!DC	0	0	1	1
NOT	DC = !SC	0	1	0	0
---		0	1	0	1
---		0	1	1	0
---		0	1	1	1
TIMP	If SC=0 then DC=DC else DC=SC	1	0	0	0
TAND	If SC=0 then DC=DC else DC=SC*DC	1	0	0	1
TOR	If SC=0 then DC=DC else DC=SC+DC	1	0	1	0
TEOR	If SC=0 then DC=DC else DC=!SC*DC +SC*!DC	1	0	1	1
TNOT	If SC=0 then DC=DC else DC=!SC	1	1	0	0
---		1	1	0	1
---		1	1	1	0
---		1	1	1	1

\* SC = Source Color code

\* DC = Destination Color code

\* EOR = Exclusive OR

**Get Pixel Color**

**Function** Returns the pixel color for the defined location.

**Parameters** R0x = >000F  
R1x = X coordinate of pixel  
R2x = Y coordinate of pixel

**Results** R0h = Returned foreground color of pixel  
R0l = Returned background color of pixel in graphics mode 2 & 3

**Parameter Description**



<b>Set Vector Color</b>
-------------------------

**Function**                    The LINE command or Set Vector command draws a straight line between two points with a logical operation. The line drawn is the hypotenuse that results after the long and short sides of a triangle are defined. The two sides are defined as distances from a single point.

**Parameters**                R0x = >10  
R1x = X coordinate of first pixel  
R2x = Y coordinate of first pixel  
R3x = X coordinate of second pixel  
R4x = Y coordinate of second pixel  
R5h = Foreground color to render vector  
R5l = Background color to render vector in graphic modes 2 & 3  
R6l = Logic operation to be performed

**Results**                    None

**Parameter Description**

Logic Operation

**Summary of Logical Operations**

Name	Operation	A3	A2	A1	A0
IMP	DC = SC	0	0	0	0
AND	DC = SC*DC	0	0	0	1
OR	DC = SC+DC	0	0	1	0
EOR	DC = !SC*DC +SC*!DC	0	0	1	1
NOT	DC = !SC	0	1	0	0
---		0	1	0	1
---		0	1	1	0
---		0	1	1	1
TIMP	If SC=0 then DC=DC else DC=SC	1	0	0	0
TAND	If SC=0 then DC=DC else DC=SC*DC	1	0	0	1
TOR	If SC=0 then DC=DC else DC=SC+DC	1	0	1	0
TEOR	If SC=0 then DC=DC else DC=!SC*DC +SC*!DC	1	0	1	1
TNOT	If SC=0 then DC=DC else DC=!SC	1	1	0	0
---		1	1	0	1
---		1	1	1	0
---		1	1	1	1

- \* SC = Source Color code
- \* DC = Destination Color code
- \* EOR = Exclusive OR

**Color Search**

**Function**                    The SEARCH command searches for a border color in the Video RAM to the right or left of a basic point.

**Parameters**                R0x = >0011  
R1x = X coordinate of source point  
R2x = Y coordinate of source point  
R3l = Color for search  
R3h = Direction for search (>00 = LEFT, >FF = RIGHT)

**Results**                    EQ status  
R0x = X coordinate of location where color was found  
R1x = Y coordinate of location where color was found

**Parameter Description**

EQ Status                    The equal status bit will be set if the function was valid and found the color, allowing you to perform a "JEQ Function\$ok" right after the software trap.

## High-Speed Block Move

**Function** Move a portion of the screen contents from one destination to a second destination with VRAM. Since the data to be transferred is done in units of one byte, there is a limitation, according to the display mode, on the value of X.

**Parameters**

- R0x = >0012
- R1x = Row number of upper left corner of source
- R2x = Column number of upper left corner of source
- R3x = Row number of upper left corner of destination
- R4x = Column number of upper left corner of destination
- R5x = Number of rows
- R6x = Number of columns
- R7I = Pixel color for blank pixels

**Results** None

### Parameter Description

Note: In Graphics Mode 4 & Graphics Mode 6, the lower one bit, and in Graphics 5 mode, the lower two bits, are lost.

**High-Speed Block Copy**

**Function** Copies a portion of the screen contents from one destination to a second destination with VRAM. Since the data to be transferred is done in units of one byte, there is a limitation, according to the display mode, on the value of X.

**Parameters** R0x = >0013  
R1x = Row number of upper left corner of source  
R2x = Column number of upper left corner of source  
R3x = Row number of upper left corner of destination  
R4x = Column number of upper left corner of destination  
R5x = Number of rows  
R6x = Number of columns

**Results** None

**Parameter Description**

Note: In Graphics Mode 4 & Graphics Mode 6, the lower one bit, and in Graphics 5 mode, the lower two bits, are lost.

<b>Logical Block Move</b>
---------------------------

**Function** Move a block of video from a source point to a destination point. Since the data to be transferred is done in units of dots, logical operations may be done on the destination data.

**Parameters**

- R0x = >0014
- R1x = Row number of upper left corner of source
- R2x = Column number of upper left corner of source
- R3x = Row number of upper left corner of destination
- R4x = Column number of upper left corner of destination
- R5x = Number of rows
- R6x = Number of columns
- R7l = Pixel color for blank pixels
- R7h = Logic operation to be performed on destination

**Results** None

**Parameter Description**

Logic Operation

**Summary of Logical Operations**

Name	Operation	A3	A2	A1	A0
IMP	DC = SC	0	0	0	0
AND	DC = SC*DC	0	0	0	1
OR	DC = SC+DC	0	0	1	0
EOR	DC = !SC*DC +SC*!DC	0	0	1	1
NOT	DC = !SC	0	1	0	0
---		0	1	0	1
---		0	1	1	0
---		0	1	1	1
TIMP	If SC=0 then DC=DC else DC=SC	1	0	0	0
TAND	If SC=0 then DC=DC else DC=SC*DC	1	0	0	1
TOR	If SC=0 then DC=DC else DC=SC+DC	1	0	1	0
TEOR	If SC=0 then DC=DC else DC=!SC*DC +SC*!DC	1	0	1	1
TNOT	If SC=0 then DC=DC else DC=!SC	1	1	0	0
---		1	1	0	1
---		1	1	1	0
---		1	1	1	1

\* SC = Source Color code

\* DC = Destination Color code

\* EOR = Exclusive OR

<b>Logical Block Copy</b>
---------------------------

**Function** Copy a block of video from a source point to a destination point. Since the data to be transferred is done in units of dots, logical operations may be done on the destination data.

**Parameters**

- R0x = >0015
- R1x = Row number of upper left corner of source
- R2x = Column number of upper left corner of source
- R3x = Row number of upper left corner of destination
- R4x = Column number of upper left corner of destination
- R5x = Number of rows
- R6x = Number of columns
- R7h = Logic operation to be performed on destination

**Results** None

**Parameter Description**

Logic Operation

**Summary of Logical Operations**

Name	Operation	A3	A2	A1	A0
IMP	DC = SC	0	0	0	0
AND	DC = SC*DC	0	0	0	1
OR	DC = SC+DC	0	0	1	0
EOR	DC = !SC*DC +SC*!DC	0	0	1	1
NOT	DC = !SC	0	1	0	0
---		0	1	0	1
---		0	1	1	0
---		0	1	1	1
TIMP	If SC=0 then DC=DC else DC=SC	1	0	0	0
TAND	If SC=0 then DC=DC else DC=SC*DC	1	0	0	1
TOR	If SC=0 then DC=DC else DC=SC+DC	1	0	1	0
TEOR	If SC=0 then DC=DC else DC=!SC*DC +SC*!DC	1	0	1	1
TNOT	If SC=0 then DC=DC else DC=!SC	1	1	0	0
---		1	1	0	1
---		1	1	1	0
---		1	1	1	1

\* SC = Source Color code

\* DC = Destination Color code

\* EOR = Exclusive OR

**Block Scroll Up**

**Function**                      Scroll a block of video on the screen up.

**Parameters**                    R0x = >0016  
R1x = Number of pixels to scroll  
R2x = Row number of upper left corner  
R3x = Column number of upper left corner  
R4x = Row number of lower right corner  
R5x = Column number of lower right corner  
R6x = Pixel color for blank lines

**Results**                        None

**Parameter Description**

**Block Scroll Down**

**Function**                      Scroll a block of video on the screen down.

**Parameters**                    R0x = >0017  
R1x = Number of pixels to scroll  
R2x = Row number of upper left corner  
R3x = Column number of upper left corner  
R4x = Row number of lower right corner  
R5x = Column number of lower right corner  
R6x = Pixel color for blank lines

**Results**                        None

**Parameter Description**



**Block Scroll Left**

**Function**                    Scroll a block of video on the screen left.

**Parameters**                R0x = >0018  
                                R1x = Number of pixels to scroll  
                                R2x = Row number of upper left corner  
                                R3x = Column number of upper left corner  
                                R4x = Row number of lower right corner  
                                R5x = Column number of lower right corner  
                                R6x = Pixel color for blank lines

**Results**                     None

**Parameter Description**

<b>Block Scroll Right</b>
---------------------------

**Function**                      Scroll a block of video on the screen right.

**Parameters**                    R0x = >0019  
                                     R1x = Number of pixels to scroll  
                                     R2x = Row number of upper left corner  
                                     R3x = Column number of upper left corner  
                                     R4x = Row number of lower right corner  
                                     R5x = Column number of lower right corner  
                                     R6x = Pixel color for blank lines

**Results**                        None

**Parameter Description**

<b>Sprite Define</b>
----------------------

<b>Function</b>	<p>Sprites are graphics that can be assigned any valid color and placed anywhere on the screen. Sprites treat the screen as a grid 256 pixels high and 256 pixels wide. However, only the first 192 pixels are visible on the screen.</p> <p>You can create up to 32 sprites in all GRAPHICS modes except TEXT modes, which do not allow sprites.</p> <p>Sprites can be set in motion in any direction at a variety of speeds. A sprite continues its motion until it is specifically changed by the program. Because sprites move from pixel to pixel, their motion can be smoother than that of characters, which can be moved only one character position (6 or 8 pixels) at a time.</p> <p>Sprites “pass over” characters on the screen. When two or more sprites are coincident (occupying the same screen pixel position), the sprite with the lowest sprite-number covers the other sprite(s).</p>
<b>Parameters</b>	<p>R0x = &gt;001A  R1x = Pointer to sprite data  R2x = Number of sprites to define</p>
<b>Results</b>	None
<b>Parameter Description</b>	
Sprite Data Mode 1	<p>1<sup>st</sup> word in list is Sprite # (Base 0)  2<sup>nd</sup> word in list is Character Code 0-255  3<sup>rd</sup> word in list is Position  4<sup>th</sup> word in list is X-Velocity  5<sup>th</sup> word in list is Y-Velocity  6<sup>th</sup> word in list is Color</p>
Sprite Data Mode 2	<p>1<sup>st</sup> word in list is Sprite # (Base 0)  2<sup>nd</sup> word in list is Character Code 0-255  3<sup>rd</sup> word in list is Position  4<sup>th</sup> word in list is X-Velocity  5<sup>th</sup> word in list is Y-Velocity  Next 16 words in list are colors</p>
Character code	<p>A numeric expression with a value from 0 to 255, specifying the character that defines the sprite pattern. If you magnify the change the sprite's size, sprite definition includes the character specified by the character code and three additional characters.</p>

**Delete Sprite**

**Function**                    The DELSPRITE subprogram enables you to delete one or more sprites.

**Parameters**                R0x = >001B  
                                R1x = Pointer to list of sprite #'s  
                                R2x = Number of sprites to delete (>FFFF for all)

**Results**                    None

**Parameter Description**

<b>Locate Sprite</b>
----------------------

**Function**                    The LOCATE subprogram enables you to change the location of one or more sprites. This subprogram can cause a sprite that has been deleted to reappear.

**Parameters**                R0x = >001C  
                                R1x = Pointer to location data  
                                R2x = Number of sprites to locate

**Results**                    None

**Parameter Description**

**Sprite Motion**

**Function** Place a sprite in motion. The MOTION subprogram is used to specify the *row-velocity* and *column-velocity* of a sprite. If both the row- and *column-velocities* are zero, the sprite is stationary. A positive *row-velocity* moves the sprite down and a negative value moves it up. A positive *column-velocity* moves the sprite to the right and a negative value moves it to the left. If both *row-velocity* and *column-velocity* are nonzero, the sprite moves smoothly at an angle in a direction determined by the actual values.

When a moving sprite reaches an edge of the screen, it disappears. The sprite reappears in the corresponding position at the opposite edge of the screen.

**Parameters** R0x = >001D  
R1x = Pointer to motion data  
R2x = Number of sprites to put in motion

**Results** None

**Parameter Description**

**Sprite Color**

**Function**                    Define sprite color

**Parameters**                R0x = >001E  
                                R1x = Pointer to color data  
                                R2x = Number of sprites to color

**Results**                    None

**Parameter Description**

<b>Define Sprite Pattern</b>
------------------------------

**Function** Define sprite pattern. The SPRITE subprogram creates sprites. Sprites are graphics which have a color and a location anywhere on the screen. They can be set in motion in any direction at a variety of speeds, and continue their motion until it is changed by the program or the program stops. They move more smoothly than the usual character which jumps from one screen position to another.

**Parameters** R0x = >001F  
 R1x = Pointer to pattern # data.  
 R2x = Number of sprites to pattern

**Results** None

**Parameter Description**

Sprite-number It is a numeric expression from 1 to 28. If the value is that of a sprite that is already defined, the old sprite is deleted and replaced by the new sprite. If the old sprite has a row- or *column-velocity*, and no new one is specified, the new sprite retains the old *velocities*.

Sprites pass over fixed characters on the screen. When two or more sprites are coincident, the sprite with the lowest sprite number covers the other sprites. While five or more sprites are on the same screen row, the one(s) with the highest sprite number(s) disappear.



<b>Sprite Magnify</b>
-----------------------

**Function** Magnify a sprite's size on the screen

**Parameters** R0x = >0020  
R1x = Magnification factor (1-4, just like extended basic)

**Results** None

**Parameter Description**

Magnification Factor 1 A *magnification-factor* of 1 causes all sprites to be single size and unmagnified. This means that each sprite is defined only by the character specified when the sprite was created and takes up just one character position on the screen.

Magnification Factor 2 A *magnification-factor* of 2 causes all sprites to be single size and magnified. This means that each sprite is defined only by the character specified when it was created, but takes up four character positions on the screen. Each dot position in the character specified expands to occupy four dot positions on the screen. The expansion from a *magnification-factor* of 1 is down and to the right.

Magnification Factor 3 A *magnification-factor* of 3 causes all sprites to be double size and unmagnified. This means that each sprite is defined by four character positions that include the character specified. The first character is the one specified when the sprite was created if its number is evenly divisible by four, or the next smallest number that is evenly divisible by four. That character is the upper left quarter of the sprite. The next character is the lower left quarter of the sprite. The next character is the upper right quarter of the sprite. The final character is the lower right quarter of the sprite. The character specified when the sprite was created is one of the four that makes up the sprite. The sprite occupies four character positions on the screen.

Magnification Factor 4 A *magnification-factor* of 4 causes all sprites to be double size and magnified. This means that each sprite is defined by four character positions that include the character specified. The first character is the one specified when the sprite was created if its number is evenly divisible by four, or the next smallest number that is evenly divisible by four. That character is the upper left quarter of the sprite. The next character is the lower left quarter of the sprite. The next character is the upper right quarter of the sprite. The final character is the lower right quarter of the sprite. The character specified when the sprite was created is one of the four that makes up the sprite. The sprite occupies sixteen character positions on the screen. The expansion from a *magnification-factor* of 3 is down and to the right.

<b>Sprite Position and Sprite Distance</b>
--

**Function** Returns the square of the distance between two sprites. The POSITION subprogram returns the position of the specified sprite(s) in the given *dot-row(s)* and *dot-column(s)* as numbers from 1 to 256. If the square of the distance is greater than 32,767, the number returned is 32,767.

The distance between two sprites is considered to be the distance between the upper-left corners of the sprite. The number returned is the square of the distance.

The distance between a sprite and a screen pixel is considered to be the distance between the upper-left corner of the sprite and the specified pixel. The number returned is the square of the distance.

**Parameters**

- R0x = >0021
- R1x = Number of sprite to get position data
- R2x = Type of Distance, 0 for none, 1 for Sprite, 2 for location
- R3x = Number of second sprite (type 1), or Pixel row (type 2)
- R4x = Pixel column (type 2)

**Results**

- R0x = Returned row of sprite
- R1x = Returned column of sprite
- R2x = Distance (if second sprite number was given)

**Parameter Description**

**Sprite Coincidence**

**Function**                   The COINCIDENCE subprogram enables you to ascertain if sprites are coincident (in conjunction) with each other or a specified screen pixel.

The exact conditions that constitute a coincidence vary depending on whether you are testing for the coincidence of two sprites, a sprite and a screen pixel, or all sprites.

If the sprites are moving very quickly, it may occasionally fail to detect a coincidence.

Two sprites are considered to be coincident if the upper-left corners of the sprites are within a specified number of pixels (coincidence checks) of each other. A coincidence exists if the distance between the pixels in the upper-left corners of the two sprites is less than or equal to the value of the coincidence checks.

A sprite is considered to be coincident with a screen pixel if the upper-left corner of the sprite is within a specified number of pixels (coincidence checks) of the screen pixel or if any pixel in the sprite occupies the screen pixel location.

**Parameters**                   R0x = >0022  
R1x = Type (0 = sprites, 1=locations, 2 = any two sprites)  
R2x = Number of coincidence checks  
R3x = Pointer to test field  
R4x = Pointer to result field

**Results**                     R0x = Number of coincidences detected

**Parameter Description**

**Sprite Pattern Define or Get**


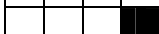







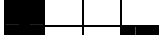






**Function**                      Define or Get Sprite pattern definitions

**Parameters**                    R0x = >0023  
 R1x = CPU address of sprite pattern definitions  
 R2x = Number of sprite patterns to define or get  
 R3x = Starting pattern number  
 R4x = 0 if Define, >FFFF if Get

**Results**                        None

**Parameter Description**

Sprites are created by turning some dots "on" and leaving others "off". The space character (ASCII code 32) is a character with all the dots turned "off". Turning all the dots "on" produces a solid block. The color of the on dots is the foreground color. The color of the off dots is the background color.

BLOCKS	Binary Code 0=Off: 1=On	Hexadecimal Code
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	A
	1011	B
	1100	C
	1101	D
	1110	E
	1111	F

**Character Pattern Define or Get**

















**Function** Define or get a character pattern definition. The CHAR subprogram allows you to define special graphics characters. You can redefine all 0 through 255 character definitions.

**Parameters**  
 R0x = >0024  
 R1x = CPU address of character pattern definitions  
 R2x = Number of patterns to define or get  
 R3x = Starting pattern number  
 R4x = 0 if Define, >FFFF if get

**Results** None

**Parameter Description**

Characters are created by turning some dots "on" and leaving others "off". The space character (ASCII code 32) is a character with all the dots turned "off". Turning all the dots "on" produces a solid block. The color of the on dots is the foreground color. The color of the off dots is the background color.

BLOCKS	Binary Code 0=Off: 1=On	Hexadecimal Code
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	A
	1011	B
	1100	C
	1101	D
	1110	E
	1111	F

**Set Text Window**

**Function** Define a text window on the screen enabling you to define screen margins. The margins you specify define a screen window that affects the operation of several instructions.

The margins cannot “overlap”; that is, the position of the top margin must be higher on the screen than the bottom margin, and the position of the left margin must be farther left on the screen than the right margin.

The valid range for margin location varies according to the graphic mode you are in. In all modes, the margins can extend to the edges of the screen.

**Parameters** R0x = >0025  
R1x = Top row  
R2x = Left column  
R3x = Bottom row  
R4x = Right column

**Results** None

**Parameter Description**

**Get Text Window**

**Function**                    Get the text window screen definitions

**Parameters**                R0x = >0026

**Results**                    R0x = Top row  
                              R1x = Left column  
                              R2x = Bottom row  
                              R3x = Right column None

**Parameter Description**

<b>Write TTY</b>
------------------

**Function** Write characters or strings to the screen.

**Parameters** R0x = >0027  
R1x = CPU address of string  
R2x = Number of characters in string (0 = null terminated string)

**Results** None

**Parameter Description**

Special Characters

>1B,>43,>2f,>2b Control sequence to set color foreground (f) and background (b) colors.

>1B,>3D,>2r,>2c Control sequence to set row (r) and column (c) position.

>0D Carriage return

>01 Home cursor position

>08 Backspace

>09 Tab (8 characters)

>0A or >0B Linefeed

>0C or >1A Clear Screen

ASCII Codes The following predefined characters may be printed or displayed on the screen.

30	(cursor)	63	? (question mark)
31	(edge character)	64	@ (at sign)
32	(space)	65	A
33	! (exclamation point)	66	B
34	" (quote)	67	C
35	# (number or pound sign)	68	D
36	\$ (dollar)	69	E
37	% (percent)	70	F
38	& (ampersand)	71	G
39	' (apostrophe)	72	H
40	( (open parenthesis)	73	I
41	) (close parenthesis)	74	J
42	* (asterisk)	75	K
43	+ (plus)	76	L
44	, (comma)	77	M
45	- (minus)	78	N
46	. (period)	79	O
47	/ (slash)	80	P
48	0	81	Q



GenREF V1.00

49	1	82	R
50	2	83	S
51	3	84	T
52	4	85	U
53	5	86	V
54	6	87	W
55	7	88	X
56	8	89	Y
57	9	90	Z
58	:	91	[ (open bracket)
59	;	92	\ (reverse slash)
60	<	93	] (close bracket)
61	=	94	^ (exponentiation)
62	>	95	_ (underline)
96	`	112	p
97	a	113	q
98	b	114	r
99	c	115	s
100	d	116	t
101	e	117	u
102	f	118	v
103	g	119	w
104	h	120	x
105	i	121	y
106	j	122	z
107	k	123	{ (left brace)
108	l	124	(vertical bar)
109	m	125	} (right brace)
110	n	126	~ (tilde)
111	o	127	DEL (appears as blank)

**Restore Character or Sprite Pattern**

**Function** Restore Character or Sprite Pattern to it's original definition.

**Parameters** R0x = >0028  
R1x = Flag (0 Sprite, <>0 Character)

**Results** None

**Parameter Description**

**Set Text Color**

**Function**                    The COLOR subprogram enables you to specify the colors of characters. In general, each character has two colors. The color of the pixels that make up the character itself is the foreground-color; the color of the pixels that occupy the rest of the character position on the screen is the background-color. If a color is transparent, the color actually displayed is the color specified by the SCREEN subprogram (>000A).

**Parameters**                R0x = >0029  
                                 R1h = Foreground color for text  
                                 R1l = Background color for text

**Results**                    None

**Parameter Description**

**Write Character String**

**Function** Write character string

**Parameters** R0x = >002A  
R1x = Address of string  
R2x = Number of characters in string  
R3x = 0 if change cursor position, >FFFF if leave cursor at beginning

**Results** None

**Parameter Description**

<b>Print Screen</b>
---------------------

**Function**                      Print screen to defined printer.

**Parameters**                      R0x = >002B  
   R1x = 0 for shades, 1 for outline  
   R2x = 0 for normal density (double), 1 for hi density (quad)

**Results**                              None

**Parameter Description**

**Horizontal Character Color**

**Function** Write characters to screen horizontally. The HCHAR subprogram displays a character anywhere on the display screen and optionally repeats it horizontally. The character with the ASCII value of *character-code* is placed in the position described by row and column and is repeated horizontally *repetition* times.

**Parameters** R0x = >002C  
R1x = Row  
R2x = Column  
R3x = ASCII character to write to screen  
R4x = Number of times to write character and color  
R5h = Foreground color for character  
R5l = Background color for character

**Results** None

**Parameter Description**

None

**Vertical Character Color**

**Function** Write characters to screen vertically. The VCHAR subprogram displays a character anywhere on the display screen and optionally repeats it vertically. The character with the ASCII value of *character-code* is placed in the position described by row and column and is repeated vertically *repetition* times.

**Parameters** R0x = >002D  
R1x = Row  
R2x = Column  
R3x = ASCII character to write to screen  
R4x = Number of times to write character and color  
R5h = Foreground color for character  
R5l = Background color for character

**Results** None

**Parameter Description**

None

**Horizontal Character**

**Function** Write characters to screen horizontally. The HCHAR subprogram displays a character anywhere on the display screen and optionally repeats it horizontally. The character with the ASCII value of *character-code* is placed in the position described by row and column and is repeated horizontally *repetition* times.

**Parameters** R0x = >002E  
R1x = Row  
R2x = Column  
R3x = ASCII character to write to screen  
R4x = Number of times to write character and color

**Results** None

**Parameter Description**

None



**Vertical Character**

**Function** Write characters to screen vertically. The VCHAR subprogram displays a character anywhere on the display screen and optionally repeats it horizontally. The character with the ASCII value of *character-code* is placed in the position described by row and column and is repeated horizontally *repetition* times.

**Parameters** R0x = >002F  
R1x = Row  
R2x = Column  
R3x = ASCII character to write to screen  
R4x = Number of times to write character and color

**Results** None

**Parameter Description**

None

**Set Mouse**

**Function**                    Set mouse speed and position

**Parameters**                R0x = >0030  
                                 R1x = New X position for mouse  
                                 R2x = New Y position for mouse  
                                 R3x = Scale factor for mouse speed (0 to 7) 0 = fastest

**Results**                     None

**Parameter Description**

**Get Mouse Speed**

**Function**                      Set mouse speed and position

**Parameters**                    R0x = >0031  
                                     R1x = Returned X position for mouse  
                                     R2x = Returned Y position for mouse  
                                     R3x = b1 b2 b3 0 xxxx xxxx xxxx (highest bits)

                                     b1 = left                    1 = down  
                                     b2 = middle                1 = down  
                                     b3 = right                  1 = down

**Results**                        None

**Parameter Description**

**Get Mouse Release**

**Function**                    Get Mouse release data.

**Parameters**                R0x = >0032

**Results**                    R1x = Returned X displacement since last call to opcode >31 or >32  
R2x = Returned Y displacement since last call to opcode >31 or >32  
R3x = b1 b2 b3 0 xxxx xxxx xxxx (highest bits)

                  b1 = left            1 = down  
                  b2 = middle        1 = down  
                  b3 = right         1 = down

**Parameter Description**

**Make Sound**

**Function**                      Make a sound through the sound generator chip on the Geneve 9640.

**Parameters**

R0x = >0033  
 R1x = Generator 1 frequency in Hz  
 R2x = Generator 2 frequency in Hz  
 R3x = Generator 3 frequency in Hz  
 R4h = Attenuation for Generator 1 (0 to 15)  
 R4l = Attenuation for Generator 2 (0 to 15)  
 R5h = Attenuation for Generator 3 (0 to 15)  
 R6h = Control for noise generator: bits = 0000 0xyz

x = 0 for periodic noise  
 x = 1 for white noise  
 yz = 00 = 6991 Hz  
 yz = 01 = 3496 Hz  
 yz = 10 = 1738 Hz  
 yz = 11 = Same Hz as Generator 3

R6l = Attenuation for Noise Generator  
 R7x = Duration of noise in 60<sup>th</sup> seconds.

**Results**                      None

**Parameter Description**

The SOUND subprogram tells the computer to produce tones or noise. The values given control three aspects of the sound: *Duration*; *frequency*; and *volume*.

Value	Range	Description
Duration	1 to 4250 -1 to -4250	The length of the sound in thousandths of a second
Frequency	(Tone) 110 to 44733 (Noise) -1 to -8	What sound is played

*Duration* is from .001 to 4.250 seconds, although it may vary up to 1/60th of a second. The computer continues performing program statements while a sound is being played. When you call the SOUND subprogram, the computer waits until the previous sound has been completed before performing the new CALL SOUND. However, if a negative *duration* is specified, the previous sound is stopped and the new one is begun immediately.

*Frequency* specifies the frequency of the note to be played with a value from 110 to 44733. (NOTE: This range goes higher than the range of human hearing. People vary in their ability to hear high notes, but generally the highest is approximately a value of 10000.) The actual frequency produced by the computer may vary up to 10 percent. The table below lists lists the frequencies of some musical tones.common notes.

## Musical Tone Frequencies

The following table gives the frequencies (rounded to integers) of four octaves of the tempered scale (one half step between notes). While this list does not represent the entire range of tones that the computer can produce, it can be helpful for programming music.

<b>Frequency</b>	<b>Note</b>	<b>Frequency</b>	<b>Note</b>
110	A	440	A (Above Middle C)
117	A#	466	A#
123	B	494	B
131	C	523	C (High C)
139	C#	554	C#
147	D	587	D
156	D#	622	D#
165	E	659	E
175	F	698	F
185	F#	740	F#
196	G	784	G
208	G#	831	G#
220	A	880	A (Above High C)
233	A#	932	A#
247	B	988	B
262	C (Middle C)	1047	C
277	C#	1109	C#
294	D	1175	D
311	D#	1245	D#
330	E	1319	E
349	F	1397	F
370	F#	1480	F#
392	G	1568	G
415	G#	1661	G#
440	A (Above Middle C)	1760	A

<b>Sound Status</b>
---------------------

**Function**                    Determine status of the sound generator

**Parameters**                R0x = >0034

**Results**                    EQ bit set if no sound is in progress.

**Parameter Description**

**VWTR**

**Function**                    Video Write to Register command with register save.

**Parameters**                R0x = >0035  
                                R1x = VDP Register #  
                                R2I = Value to put into VDP register

**Results**                    None

**Parameter Description**

.



**VRFR**

**Function**                    Video Read From Register, actually a read from stored values.

**Parameters**                R0x = >0036  
                                R1x = VDP register #

**Results**                    R0I = Value read from register

**Parameter Description**

<b>Get Tables</b>
-------------------

**Function**                      Get Tables

**Parameters**                      R0x = >0037  
    R1x = Pointer to user data to put copy of tables (24 bytes)

**Results**                              None

**Parameter Description**

Tables	CTABLE	Data	0,0
	PTABLE	Data	0,0
	SCRIMG	Data	0,0
	SPRATT	Data	0,0
	SPRPAT	Data	0,0
	SPRCOL	Data	0,0

**Get Palette Registers**

**Function**                    Get Palette Registers

**Parameters**                R0x = >0038  
                              R1x = Pointer in user data, to put copy of Palette registers (32 bytes)

**Results**                    None

**Parameter Description**