VIDEO XOP'S

Data Library 6

```
* SetVideoMode          >00
*
* R1x = Video mode
*
* Video modes:
*       0000    Text 1 mode
*       0001    Text 2 mode
*       0002    MultiColor mode
*       0003    Graphic 1 mode
*       0004    Graphic 2 mode
*       0005    Graphic 3 mode
*       0006    Graphic 4 mode
*       0007    Graphic 5 mode
*       0008    Graphic 6 mode
*       0009    Graphic 7 mode
*       000A    Text 2 mode -- 26 lines
*

* GetVideoMode          >01
*
* -----
* R0x = Returned video mode
* R1x = Number of columns
* R2x = Number of rows
* R3x = Number of Graphics columns (pixels)
* R4x = Number of Graphics 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
*

* SetCurPos             >02
*
* R1x = Row number
* R2x = Column number
*

* GetCurPos             >03
*
* -----
* R0x = Returned row number
* R1x = Returned column number
*

* SetDisPage            >04
*
* R1x = Page number
* R2x = Initialize tables? (0=YES !0=NO)
*
```

```
* GetDisPage            >05
*
* -----
* R0x = Returned page number
*

* ScrollWinUp           >06
*
* 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
*

* ScrollWinDown         >07
*
* 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
*

* ScrollWinLeft         >08
*
* 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
*

* ScrollWinRight        >09
*
* 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
*
```

```
* CALL COLOR   (ONLY WORKS IN SCREEN CODE 3) >0A
*
* R1h = Foreground color
* R1l = Background color
* R2x = Charset # (if mode 3)
*

* GCharColor              >0B
*
* R1x = Row
* R2x = Col
* -----
* R0x = ASCII character read from screen
* R1h = Foreground color for character
* R1l = Background color for character
*

* SetBorderColor          >0C
*
* R1x = Color to render border
*

* SetColPalette           >0D
*
* R1x = Palette register number
* R2x = Color to put into palette register
* (G,R,B -- AS BOOK DESCRIBES)  NIBBLE JUSTIFIED MOST 0-G-R-B
*

* SetPixelColor           >0E
*
* R1x = X coordinate of pixel
* R2x = Y coordinate of pixel
* R3h = Foreground color to render pixel
* R3l = Background color to render pixel in graphics 2-3
* R4h = Logic operation to be performed
*

* GetPixelColor           >0F
*
* R1x = X coordinate of pixel
* R2x = Y coordinate of pixel
* -----
* R0h = Returned foreground color of pixel
* R0l = Returned background color of pixel in graphics 2-3
*
```

```
* SetVectorColor          >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 graphics 2-3
* R6h = Logic operation to be performed
*

* ColorSRCH               >11
*
* R1x = X coordinate of source point
* R2x = Y coordinate of source point
* R3l = Color for search
* R3h = Direction for search (>00=LEFT, >FF=RIGHT)
* -----
* ST  = if EQUAL, color found
* R0x = X coordinate of location where color was found
* R1x = Y coordinate of location where color was found
*

* HBlockMove              >12
*
* 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
*

* HBlockCopy              >13
*
* 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
*

* LBlockMove              >14
*
* 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
```

```
* LBlockCopy              >15
*
* 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
*

* BScrollUp               >16
*
* 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
* R6h = Pixel color for blank pixels
*

* BScrollDown             >17
*
* 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
* R6h = Pixel color for blank pixels
*

* BScrollLeft             >18
*
* 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
* R6h = Pixel color for blank pixels
*

* BScrollRight            >19
*
* 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
* R6h = Pixel color for blank pixels
*
```

```
                ##### SEE ADDENDUM FOR MOR ON SPRITES #####
* SpriteDef                >1A
*
* R1x = Pointer to sprite data
* R2x = # of sprites to define
*

* SpriteDel                >1B
*
* R1x = Pointer to list of sprite #'s
* R2x = # of sprites to delete (>FFFF for all)
*

* SpriteLocate             >1C
*
* R1x = Pointer to location data
* R2x = # of sprites to locate
*

* SpriteMotion             >1D
*
* R1x = Pointer to motion data
* R2x = # of sprites to put in motion
*

* SpriteColor              >1E
*
* R1x = Pointer to color data      LIST=SP#,SP COL,....,SP#,SP COL
* R2x = # of sprites to color
*

* SpritePattern            >1F
*
* R1x = Pointer to pattern # data  (SET POINTER TO SPRITE TABLE 0-255)
* R2x = # of sprites to pattern
*

* SpriteMagnify            >20
*
* R1x = MagFac (1-4, just like XB)
*

* SpritePosition / SpriteDistance          >21
*
* R1x = # of sprite to get position of
* R2x = Type of Distance, 0 for none, 1 for Sprite, 2 for Location
* R3x = # of second sprite (type 1), or Pixel row (type 2)
* R4x = Pixel column (type 2)
* -----
* R0x = Returned Row of sprite
* R1x = Returned Column of sprite
* R2x = Distance (if second sprite # was given)
* R3X = Distance squared (if second sprite # was given)
```

```
* SpriteCoinc            >22
*
* R1x = Type (0=sprites, 1=locations, 2=any two sprites)
* R2x = # of coincidence checks (if1,2) IF 2 PUTS THE RESULT IN R4 ONLY 2
* R3x = Pointer to test field     0   SP#, SP#, TOLER  1  SP#,DR,DC,TOL
* R4x = Pointer to result field   0 one COINC PAIR  row,col  if coinc
* -----                           1 SP#, DR, DC
*                                 2 DR,DC         (2 IS ALL, SP TO SP IS 0)
* R0x = # of coincidences detected                    (DR,DC  )
*

* SpritePatDefGet        >23
*
* R1x = CPU address of sprite pattern definitions
* R2x = # of sprite patterns to define or get
* R3x = Starting pattern #
* R4x = 0 if Def, >FFFF if Get
*

* CharPatDefGet          >24
*
* R1x = CPU address of character pattern definitions
* R2x = # of character patterns to define or get     (SPRITE & CHAR DATA SAME
* R3x = Starting pattern #                                ONLY IN MODE 1,1)
* R4x = 0 if Def, >FFFF if Get
*

* SetTextWindow          >25
*
* R1x = Top row
* R2x = Left column
* R3x = Bottom row
* R4x = Right column
*

* GetTextWindow          >26
*
* -----
* R0x = Returned top row
* R1x = Returned left column
* R2x = Returned bottom row
* R3x = Returned right column
*

* WriteTTY               >27
*
* R1x = CPU address of string
* R2x = # of characters in string (0=NULL terminated)
*

* RestoreCharSpritePat   >28
*
* R1x = Restore Chars?   (0=No)
* R2x = Restore Sprites? (0=No)
```

```
* SetTextColor          >29
*
* R1h = Foreground color for text
* R1l = Background color for text
*

* WriteCharString       >2A
*
* R1x = Address of string
* R2x = # of characters in string
* R3x = 0 if change cursor position, >FFFF if leave cursor at beginning
*

* PrintScreen           >2B
* R1x = 0 for shades, 1 for outline
* R2x = 0 for normal density (double), 1 for hi density (quad)

* HCharColor            >2C
*
* R1x = Row
* R2x = Col
* 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
*

* VCharColor            >2D
*
* R1x = Row
* R2x = Col
* 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
*

* HChar                 >2E
*
* R1x = Row
* R2x = Col
* R3x = ASCII character to write to screen
* R4x = Number of times to write character and color
*

* VChar                 >2F
*
* R1x = Row
* R2x = Col
* R3x = ASCII character to write to screen
* R4x = Number of times to write character and color
*
```

TEXT ONLY.

```
* SetMouse                >30
*
* R1x = new Xposition for mouse
* R2x = new Yposition for mouse
* R3x = scale factor for mouse speed  (0 to 7) 0=fastest
* -----
*
* GetMouse                >31
*
* -----
* R1x = Returned Xposition for mouse
* R2x = Returned Yposition for mouse
* R3x = b1 b2 b3 0 xxxx xxxx xxxx   (highest bits)
*       \          b1= left   1=down
*          \       b2= middle 1=down
*             \    b3= right  1=down
*
* GetMouseRel             >32
*
* -----
* R1x = Returned X displacement since last call to GetMouse or GetMouseRel
* R2x = Returned Y displacement since last call to GetMouse or GetMouseRel
*
* MakeSound               >33
*
* R1x = Generator 1 frequency in Hz
* R2x = Generator 2 frequency in Hz
* R3x = Generator 3 frequency in Hz
* R4h = Attenuation for Generator 1 (0-15)
* R4l = Attenuation for Generator 2 (0-15)
* R5h = Attenuation for Generator 3 (0-15)
* R6h = control for noise generator: bits= 0000 0w t1 t2
*                                          |  +--+-- 00= 6991 Hz
*                                          |         01- 3496 Hz
*                                          |         10= 1738 Hz
*                                          |         11= same Hz as #3
*                                          +---------- 0= periodic noise
*                                                      1= white noise
* R6l = Attenuation for Noise Generator
* R7x = duration of noise in 60th seconds
* -----


*
* SoundStatus             >34
*
* -----
* EQ bit set if no sound is in progress
*
```

```
* OPCODE >35
* VWTR - write to video registers, with register save
*
* R1x = VDP register #
* R2l = Value to put into VDP register
*

* OPCODE >36
* VRFR - read from video registers, actually, a read from stored values
*
* R1x = VDP register #
* -----
* R0l = Value read from VDP register
*

* GetTables              >37
*
* R1x = Pointer in user data, to put copy of tables (24 bytes)
* -----
* Copies of this data are put into user data:
*
* CTABLE  DATA    0,0
* PTABLE  DATA    0,0
* SCRIMG  DATA    0,0
* SPRATT  DATA    0,0
* SPRPAT  DATA    0,0
* SPRCOL  DATA    0,0
*

* GetPalRegs
*
* R1x = Pointer in user data, to put copy of Palette Registers (32 bytes)
* R2x = FORMAT (0=SQUASHED !0=BY THE BOOK, EXPANDED)
*
```

9938 does not support Direct Reads.

```
MOUSE SAME VIDEO

R0 2A-SET MOUSE
R1   -X POSTION
R2    Y POSITION
R3    SCALE FACTOR 0-7 0 FASTEST

R0 2B-GET ABSOLUTE (WHOLE REGISTER)
R1    RETURNED X
R2    RETURNED Y
R3    MOST SIGNIFICANT 3 BITS L BUTTON, 1 DOWN, 0 NOT DOWN
```

R0 2C-GET RELATIVE (HOW MUCH MOUSE HAS MOVED SINCE LAST CALL TO GET MOUSE)
R1 RETURNED X
R2 RETURNED Y
R3 MOST 3 BITS L BUTTON,1 DOWN 0 NOT DOWN

DEFINE SPRITES

POINTER IS TO WORDS (SPRITE MODE 1)

1ST WORD IN LIST IS SPRITE # (BASE 0)
2ND WORD IN LIST IS CHAR CODE 0-255
3RD WORD IN LIST IS POSITION
4TH WORD IN LIST IS VOLOCITY
5TH WORD IN LIST IS VOLOCITY
6TH WORD IN LIST IS COLOR
POINTER IS TO WORDS (SPRITE MODE 2)

1ST WORD IN LIST IS SPRITE # (BASE 0)
2ND WORD IN LIST IS CHAR CODE 0-255
3RD WORD IN LIST IS POSITION
4TH WORD IN LIST IS VOLOCITY
5TH WORD IN LIST IS VOLOCITY
NEXT 16 WORDS IN LIST ARE COLORS

SPRITE COLOR

SAME AS COLORS FOR SPRITE MODE 1 OR 2

```
*
* math library ...  XOP @TEN,0
*
*      all floating point arguments must be on an even byte boundary
*      calling registers must be in PAD from >f000 to >f060
*
*      floating point representation, radix 100:
*
*      0  = 00 00 xx xx xx xx xx xx
*     +n  = e0 m0 m1 m2 m3 m4 m5 m6
*
*           e0 is int(log[100](n)) + >40
*           m0 - m6 are numbers from >00 to >63 (0 to 99)
*           m0 is most  significant digits of mantissa,
*           m6 is least significant digits of mantissa.
*
*           in normalized numbers, decimal is between m0 and m1
*     -n  is same as "n" except first word is negated ... -(e0 m0)
*
*      examples:
*      decimal                  floating point
*
*      7                        >40 >07 >00 >00 >00 >00 >00 >00
*      70                       >40 >46 >00 >00 >00 >00 >00 >00
*      2,345,600                >43 >02 >22 >38 >00 >00 >00 >00
*      23,456,000               >43 >17 >2D >3C >00 >00 >00 >00
*      0                        >00 >00 >xx >xx >xx >xx >xx >xx
*      -7                       >BF >F9 >00 >00 >00 >00 >00 >00
*      -70                      >BF >BA >00 >00 >00 >00 >00 >00
*      -2,345,600               >BC >FE >22 >38 >00 >00 >00 >00
*
*  opcode #0
*  FCOMP    floating point compare
*           input:           r0=0
*                            r2=^float1
*                            r3=^float2
*
*           return:          status reg = AG set iff (float2 > float1)
*                                         EQ set iff (float2 = float1)
*
*  opcode #1
*  FSUB     floating point subtract
*           input:           r0=1
*                            r1=^result
*                            r2=^float1
*                            r3=^float2
*           return:          r0=error code
*                            ^r1 = float2 - float1
*
```

```
*  opcode #2
*  FADD     floating point add
*           input:          r0=2
*                           r1=^result
*                           r2=^float1
*                           r3=^float2
*           return:         r0=error code
*                           ^r1 = float2 + float1
*
*  opcode #3
*  FMULT    floating point multiply
*           input:          r0=3
*                           r1=^result
*                           r2=^float1
*                           r3=^float2
*           return:         r0=error code
*                           ^r1 = float2 * float1
*
*  opcode #4
*  FDIV     floating point divide
*           input:          r0=4
*                           r1=^result
*                           r2=^float1
*                           r3=^float2
*           return:         r0=error code
*                           ^r1 = float2 / float1
*
*  opcode #5
*  PWR      floating point power
*           input:          r0=5
*                           r1=^result
*                           r2=^float1
*                           r3=^float2
*           return:         r0=error code
*                           ^r1 = float2 ^ float1
*
*  opcode #6
*  EXP      floating point e^x
*           input:          r0=6
*                           r1=^result
*                           r2=^float1
*           return:         r0=error code
*                           ^r1 = e ^ float1
*
*  opcode #7
*  LOG      floating point ln(x)
*           input:          r0=7
*                           r1=^result
*                           r2=^float1
*           return:         r0=error code
*                           ^r1 = ln( float1 )
*
```

```
*  opcode #8
*  SQR      floating point sqr(x)
*           input:            r0=8
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = sqr( float1 )
*
*  opcode #9
*  COS      floating point cos(x)
*           input:            r0=9
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = cos( float1 )
*
*  opcode #10
*  SIN      floating point sin(x)
*           input:            r0=10
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = sin( float1 )
*
*  opcode #11
*  TAN      floating point tan(x)
*           input:            r0=11
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = tan( float1 )
*
*  opcode #12
*  ATN      floating point atn(x)
*           input:            r0=12
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = atn( float1 )
*
*  opcode #13
*  GRI      floating point greatest integer
*           input:            r0=13
*                             r1=^result
*                             r2=^float1
*           return:           r0=error code
*                             ^r1 = ;; float1 ;;
*
*  opcode #14
*  CFI      convert floating point to integer
*           input:            r0=14
*                             r2=^float1
*           return:           r0=error code
*                             r1 = integer
```

```
*  opcode #15
*  CIF      convert integer to floating point
*           input:            r0=15
*                             r1=^result
*                             r2=integer
*           return:           r0=error code
*                             ^r1 = float( integer )
*
*  opcode #16
*  CSINT    convert string to integer
*           input:            r0=16
*                             r2=^string
*           return:           r0=error code
*                             r1 = integer
*
*  opcode #17
*  CSN      convert string to floating point
*           input:            r0=17
*                             r1=^result
*                             r2=^string
*                             r3=strlen
*           return:           r0=error code
*                             ^r1 = float( string )
*
*  opcode #18
*  CNS      convert float to string
*           input:            r0=18
*                             r1=^string
*                             r2=^float1
*                             r3=opt1
*                                 bit 0: 0=free form (ignore opt2, opt3)
*                                        1=fixed (opt2, opt3 are field sizes)
*                                 bit 1: 1 for explicit sign
*                                 bit 2: 1 to show sign of positive number
*                                        as a '+' instead of as a space.
*                                        (bit 1 must also be on)
*                                 bit 3: 1 for E-notation output
*                                 bit 4: 1 for extended E-notation
*                                        (bit 3 must also be 1)
*                             r4=if fixed format, number of places to left
*                                of decimal point, including explicit sign
*                             r5=if fixed format, number of places to the
*                                right of decimal point and including decimal
*                                point
*                             if fixed format, with exponent, r4,r5 exclude the
*                             3 places for an exponent
*
*           return:           r0=error code
*                             ^r1 = string
*
*
```

```
***************************
*
*
* keyboard    XOP @FIVE,0
*
*      IN:      R0Lsb =  scan mode   (like ti modes, 0-5)
*
*      OUT:     R0Lsb =  returned mode  (same as >8374 in 99/4a)
*               R1Msb =  returned scan code (same as >8375)
*               R2Msb =  joystick Y value     (same as >8376)
*               R2Lsb =  joystick X value     (same as >8377)
*           EQ bit in Status register set if there's a new key in R0
*
*               mode 7 = break key check  return EQ=1 if Break on
*               mode 8 = raw scan code returned in R1h
*                        raw code returned is >FF if there was no code in buffer
*
```

-- There MUST Be A Modification to get this to work.
PATCH 95H - will fix the problem.