THE LONG-AWAITED LINE "A" DOCUMENT

In order to provide "quick-and-dirty" access to the assembler-level graphics routines, ATARI engineers have set up the 68000's LINE "A" exception as an interface to several useful routines. The LINE "A" interface is faster than going through GEM's VDI and has some extra features. Also, LINE "A" calls require less application code than their VDI counterparts. Of course, LINE "A" doesn't replace the VDI completely, but if an application only needs a few primitive graphics functions (and wants maximum performance), then LINE "A" is sufficient (and optimal).

The LINE "A" interface is provided for the hacker-at-heart and no claims are made about its ease of use. The interface may seem unusually inconsistent, but it was not designed; it simply fell out as a freebie from the low-level VDI primitives interface. That is, these routines are the heart of the VDI.

The LINE "A" interface consists of 15 opcodes. The calls to LINE "A" are assembled as 1-word instructions, the highest 4 bits of which are 1010 (A in hexadecimal, hence LINE "A") and the lower 12 bits of which are used as the opcode field. Following is a description of the 15 opcodes:

```
0  = Initialization.
1  = Put pixel.
2  = Get pixel.
3  = Line.
4  = Horizontal line.
5  = Filled rectangle.
6  = Line-by-line filled polygon.
7  = BitBlt.
8  = TextBlt.
9  = Show mouse.
10 = Hide mouse.
11 = Transform mouse.
12 = Undraw sprite.
13 = Draw sprite.
14 = Copy raster form.

15 = Seedfill.  (exists only in versions of TOS after the 1st release)
```

The LINE "A" routines have some features that the VDI doesn't support. BitBlt supports half-tone patterns on the source and TextBlt supports all 16 BitBlt logic operations, not just the 4 GEM VDI writing modes. In addition to these straight-forward extensions LINE "A" also allows the adventurous programmer to experiment with special effects. The BitBlt is especially generous in this area.

(0) Initialization

```
...     ...
dc.w    $A000           ; Init the LINE "A".
...     ...
```

input: none.

output: d0 = ptr to the base address of LINE "A" interface variables.
        a0 = ptr to the base address of LINE "A" interface variables.
        a1 = ptr to array of ptrs to the 3 system font headers.

865

```
                a2 = ptr to array of ptrs to the 15 LINE "A" routines.

        note:   The value returned in a0 is the sine qua non of the LINE "A"
                interface.  Inputs to all the other LINE "A" operations are
                made relative to this value, i.e., the LINE "A" interface
                variables are contained in a structure pointed to by a0.
                The offsets of these variables in the structure are given
                below.

        bugs:   In the first TOS release, a2 is not returned as described
                above.  Instead, it is preserved across the LINE "A" call.
                See Example Program #2 at the end of this document for the
                technique that makes a2 point to the proper place.

(1) Put pixel

        ...     ...
        dc.w    $A001           ; Plot a pixel at x,y.
        ...     ...

        input:  INTIN[0] = pixel value.
                PTSIN[0] = x coordinate.
                PTSIN[1] = y coordinate.

        output: none.

        note:   For a discussion of the CONTRL, INTIN, PTSIN, INTOUT, & PTSOUT
                arrays, see the GEM VDI manual.


(2) Get pixel

        ...     ...
        dc.w    $A002           ; Get the pixel at x,y.
        ...     ...

        input:  PTSIN[0] = x coordinate.
                PTSIN[1] = y coordinate.

        output: d0 = pixel value.


(3) Line

        ...     ...
        dc.w    $A003           ; Draw a line between (x1,y1) and (x2,y2).
        ...     ...

        input:  X1 = x1 coordinate.
                Y1 = y1 coordinate.
                X2 = x2 coordinate.
                Y2 = y2 coordinate.
                COLBIT0 = bit value for plane 0.
                COLBIT1 = bit value for plane 1.
                COLBIT2 = bit value for plane 2.
                COLBIT3 = bit value for plane 3.
                LNMASK  = line style mask.
                WMODE   = writing mode.
                LSTLIN  = always set this to -1, if using xor mode.
                          else ignore it.
```

```
        output: LNMASK is rotated to align with right-most endpoint.

        quirks: 1) If the line is horizontal, LNMASK is a word-aligned
                pattern, not a line style.  That is, a bit other than
                bit 15 of LNMASK may be used at the left-most endpoint.

                2) As the foregoing references imply, the line is always
                drawn from left to right, not from (X1,Y1) to (X2,Y2).
                Thus, LNMASK is always applied from left to right.

        note:   Because of the quirks, an application cannot depend upon the
                phase of the LNMASK being properly updated between calls
                to line-drawing primitives.  If the phase is critical, the
                application must compute and init LNMASK before each line
                is drawn.

                LNMASK is applied to the line-drawing DDA algorithm along
                the direction of greater delta.  If delta Y is greater than
                delta X, then LNMASK is applied in the Y direction.

                These line-drawing quirks and notes apply to the GEM VDI, too


(4) Horizontal line

        ...     ...
        dc.w    $A004           ; Draw a line from (x1,y1) to (x2,y1).
        ...     ...

        input:  X1 = x1 coordinate.
                Y1 = y1 coordinate.
                X2 = x2 coordinate.
                COLBIT0 = bit value for plane 0.
                COLBIT1 = bit value for plane 1.
                COLBIT2 = bit value for plane 2.
                COLBIT3 = bit value for plane 3.
                WMODE   = writing mode.
                PATPTR  = ptr to the fill pattern.
                PATMSK  = pattern index.
                MFILL   = multi-plane pattern flag.


        output: none.


(5)     Filled rectangle

        ...     ...
        dc.w    $A005   ; Draw a filled rectangle with upper left corner at
                        ;    (x1,y1) and lower right corner at (x2,y2).
        ...     ...

        input:  X1 = x1 coordinate.
                Y1 = y1 coordinate.
                X2 = x2 coordinate.
                Y2 = y2 coordinate.
                COLBIT0 = bit value for plane 0.
                COLBIT1 = bit value for plane 1.
                COLBIT2 = bit value for plane 2.
                COLBIT3 = bit value for plane 3.
                WMODE   = writing mode.
```

```
                PATPTR  = ptr to the fill pattern.
                PATMSK  = fill pattern index.
                MFILL   = multi-plane fill pattern flag.
                CLIP    = clipping flag.
                XMINCL  = x minimum for clipping.
                XMAXCL  = x maximum for clipping.
                YMINCL  = y minimum for clipping.
                YMAXCL  = y maximum for clipping.


        output: none.
```

(6) Line-by-line filled polygon.

```
        ...     ...
        dc.w    $A006   ; Draw 1 scan-line of a filled polygon.
        ...     ...

        input:  PTSIN[]    = array of polygon vertices.
                             ((x1,y1),(x2,y2)...,(xn,yn),(x1,y1))
                CONTRL[1] = n = number of vertices.
                Y1         = y coordinate of scan-line to fill.
                COLBIT0    = bit value for plane 0.
                COLBIT1    = bit value for plane 1.
                COLBIT2    = bit value for plane 2.
                COLBIT3    = bit value for plane 3.
                WMODE      = writing mode.
                PATPTR     = ptr to the fill pattern.
                PATMSK     = fill pattern mask.
                MFILL      = multi-plane fill pattern flag.
                CLIP       = clipping flag.
                XMINCL     = x minimum for clipping.
                XMAXCL     = x maximum for clipping.
                YMINCL     = y minimum for clipping.
                YMAXCL     = y maximum for clipping.


        output: X1 and X2 are clobbered.

        note:   The 1st endpoint must be repeated at the end of the list of
        n endpoints.
```

(7) BitBlt

```
        ...     ...
        dc.w    $A007   ; Perform a BIT BLock Transfer.
        ...     ...

        input:  a6 = ptr to a structure of input parameters.

        output: none.


        BIT BLT PARAMETER BLOCK OFFSETS


B_WD            equ     +00     ; width of block in pixels
B_HT            equ     +02     ; height of block in pixels
```

868

```
PLANE_CT        equ     +04     ; number of consecutive planes to blt    {D}

FG_COL          equ     +06     ; foreground color (logic op index:hi bit){D}
BG_COL          equ     +08     ; background color (logic op index:lo bit){D}
OP_TAB          equ     +10     ; logic ops for all fore and background combos
S_XMIN          equ     +14     ; minimum X: source
S_YMIN          equ     +16     ; minimum Y: source
S_FORM          equ     +18     ; source form base address
S_NXWD          equ     +22     ; offset to next word in line  (in bytes)
S_NXLN          equ     +24     ; offset to next line in plane (in bytes)
S_NXPL          equ     +26     ; offset to next plane from start of current

D_XMIN          equ     +28     ; minimum X: destination
D_YMIN          equ     +30     ; minimum Y: destination
D_FORM          equ     +32     ; destination form base address
D_NXWD          equ     +36     ; offset to next word in line  (in bytes)
D_NXLN          equ     +38     ; offset to next line in plane (in bytes)
D_NXPL          equ     +40     ; offset to next plane from start of current

P_ADDR          equ     +42     ; address of pattern buffer (0:no pattern) {D
P_NXLN          equ     +46     ; offset to next line in pattern  (in bytes)
P_NXPL          equ     +48     ; offset to next plane in pattern (in bytes)
P_MASK          equ     +50     ; pattern index mask

P_BLOCK_LEN     equ      76     ; the parameter block must be 76 bytes long
```

*** notes ***

parameters marked with {D} may be altered during the course
of the BIT BLT execution

contents of OP_TAB

```
+00   byte    operation employed when foreground and background colo
              bits for current plane are both clear (0)

+01   byte    operation employed when current plane's foreground colc
              bit is clear (0) and background color bit is set (1)

+02   byte    operation employed when current plane's foreground Aco
              bit is set (1) and background color bit is clear (0)

+03   byte    operation employed when foreground and background colo
              bits for current plane are both set (1)
```

0. PREFACE

Before one floggles one's tormented mind with this tangled nest of arcane knowledge, one ought to be intimately familiar with chapter 6 of the GEM VDI manual. Author assumes that one's knowledge of Raster matters is quite wide and that the rudiments of BIT BLTting are belc discussion. If the author is mistaken then he's sorry (and you're about to become lost in the sea of woe, oh ho!).

I. PARAMETER BLOCK

BIT BLT is accessed via a 76 byte parameter block. Register A6 points to the head of this block upon LINE A entry. Only the first 52 bytes of the block need be attended to by the abuser. The remaining space is maintained internally by the BLT. Note in the following explanations, parameters will be refered to by symbolic offsets into the parameter block.

II. MEMORY FORMS

memory forms are something like a cabbage patch. (a cabbage patch is a place for mentally retarded programmers). Face it, forms are nothing like a cabbage patch. if you think they are, go back and read chapter 6 in the GEM VDI manual. if you know anything at all about memory forms, you know they are almost entirely but not totally unlike a garbage can. memory forms are of two sexes, source and destination.
each sex is defined by the same four parameters: form block address, block width, offset to next contiguous word, and offset to next plane.

S_FORM and D_FORM point to the first words of the source memory form and destination memory forms, respectively. addresses must fall on word boundries or severe hardships fall (as will address exceptions) like plagues upon the ancient egyptians.

S_NXWD and D_NXWD are offsets to the next word in a plane of the memory form. for example, in the monochrome mode the value is 2 while a value 4 is used in medium resolution and 8 is applicable to low resolution.

S_NXLN and D_NXLN are form widths for source and destination. (i can't remember which one belongs to source form and which one belongs to the destination form). widths must be even byte values, as you know, for they represent the offset from one row to the next and forms must be word aligned and an integral number of words wide. (hint: the hi rez screen value is 90 while lo and medium rez values are 160)

S_NXPL and D_NXPL are offsets from the start of one plane to start of the next plane. because of the ST screen's interleaved plane structure, this value is always two (2). alternative universes allow for a series of contiguous planes where NXPL values are number of bytes each plane. thus , it is possible to BLT from the contiguous universe into the interleaved ST universe and vice versa.

the actual bit alligned blocks of memory are defined within the form by an upper left anchor point, a pixel width, and a pixel height: (S_XMIN, S_YMIN, B_WD, and B_HT). the location in the destination form is defined by an anchor point (D_XMIN, D_YMIN). no harm will come if these two areas overlap. Note no clipping is performed andthere is no checking to determine whether bit blocks fall within the confines of the encompasing memory forms. finally, the number of planes to be transfered (the number of itterations of the BLT algorithm) is contained in the PLANE_CT word.

III. RASTER OPERATIONS

OP_TAB is a table of four RASTER OP codes. Each of byte wide entries in OP_TAB contain a code for one of sixteen logical operations between consenting source and destination blocks. For each plane, the logical operation is chosen by indexing the OP_TAB with a value derived from FG_COL and BG_COL words. given plane "n", bit "n" of FG_COL is the hi bit of the two bit index value and bit "n" of BG_COL is the lo bit of the index value.

for those with a furniture fetish, here is a table:

FG(n)	BG(n)	OP_TAB entry
0	0	first entry
0	1	second entry
1	0	third entry
1	1	fourth entry

IV. PATTERNS

Patterns are word wide, word aligned images that are logically anded with source prior to logical combination of source with destination.

Patterns are packed in an imaginary grid anchored left corner (0,0) of the destination memory form.

Patterns are 16 bits wide and repeated every 16 pixels horizontally.

patterns are an integral power of 2 in height and repeat vertically at that frequency.

The source is shifted into alignment with destination rectangle prior to the combination of source with pattern.
Thus, the relationship between source and pattern is dependent upon the X,Y positioning of the destination rectangle.

P_ADDR points to the first word of the pattern. If this pointer is 0, a pattern is not combined with the source rectangle.

P_NXLN offset (in bytes) between consecutive words in the pattern.
For reasons too inane, this number should be an integral power of 2 (such as 2,4, or 8)

P_NXPL is the offset (in bytes) from the beginning of a plane to the beginning of the next plane. In the case of a single plane pattern used in a multi plane environment, this value would be zero. thus, the same pattern is repeated through all planes.

P_MASK works with P_NXLN to specify the length of the pattern.
The length (in words) of the pattern must be an integral power of 2.

```
if    P_NXLN  =  2 ** n

then P_MASK   =  (length in words -1) << n
```

... i don't know why. go ask your father.

V. BAG 'O TRICKS

Q. I want to BLT from a single plane source to multi plane destination.

A. That's not in the form of a question. And besides, i can't think with water pick spurtin in my ear. Hey, that's my cat your puttin in the Cuisinart. Wha you think your doin bustin into my word processor like this. Hey bud, stay away from that delete key. Hey moe foe, i'm serious. How'd you like an unexpected interrupt ?

Q. This key is loaded and it's pointed at your bonus check.

A. ok,ok... i'll talk.

S_NXPL =0 => same source plane is BLTted to all destination planes

Q. yea, i know that but what logic ops do i use ?

A. to map 1's to foreground color and 0's to background color set OP_TAB to:

offset	logic op	
+00	00	all zeros
+01	04	D' <- [not S] and D
+02	07	D' <- S or D
+03	15	all ones

load foreground color into FG_COL and background color into BG_COL

Q. you wanna buy some lake bottom property?

A. to map 1's to foreground color and make 0's transparent set OP_TAB to:

offset	logic op	
+00	04	D' <- [not S] and D
+01	04	D' <- [not S] and D
+02	07	D' <- S or D
+03	07	D' <- S or D

load foreground color into FG_COL
it doesn't matter what you put into BG_COL

don't forget to set S_NXPL to 0

enough smalltalk, let's get down to the core of the issue.
Here are some of my Aunt Marge's flavorful BIT BLT recipes:

1. BLT a pattern without Source to the Destination.

For this number, we'll need a word of ones. Label it "ones:" next, point S_FORM at "ones". Set S_NXLN, S_NXPL, S_NXWD, S_XMIN, and S_YMIN to 0. Set up the pattern as you usually would and before you know it, you'll have a wonderful steaming pattern filled rectangle.

2. this is a nice way to make a sprite like device.

- o you will need to bake a monoplane mask. everywhere there is a 1 in the mask, the background will be removed. wherever a 0 falls the background is left intact.

set OP_TAB to:

offset	logic op	
+00	04	D' <- [not S] and D
+01	04	D' <- [not S] and D
+02	07	D' <- S or D
+03	07	D' <- S or D

load foreground color into FG_COL
it doesn't matter what you put into BG_COL

- o next, take monoplane form (or multiplane form) and "or" it (OP 07 into the area that you just scooped out with the mask

feeds a family of four.

(8) TextBlt

```
...     ...
dc.w    $A008   ; Perform a TEXT BLock Transfer of 1 character.
...     ...
```

input:

```
        WMODE    = writing mode.(0-3 => VDI modes
                                 4-19 => BitBlt modes)
        TEXTFG   = text foreground color.
        TEXTBG   = text background color. (used for modes 4-19)
        FBASE    = ptr to start of font data. (font form)
        FWIDTH   = width of font form.
        SOURCEX  = x coord of character in font form.
        SOURCEY  = y coord of character in font form.
        DESTX    = x coord of character on screen.
        DESTY    = y coord of character on screen.
        DELX     = width of character.
        DELY     = height of character.
        STYLE    = vector of TextBlt special effects flags.
        LITEMASK = the mask to use in lightening text.
        SKEWMASK = the mask to use in skewing text.
        WEIGHT   = the width by which to thicken text.
        ROFF     = offset above character baseline when skewing.
        LOFF     = offset below character baseline when skewing.
```

```
            SCALE    = scaling flag. (0 => no scaling.)
            XDDA     = accumulator for x dda.
            DDAINC   = fractional amount to scale up or down.
            SCALDIR  = scale direction flag. (0 => down)
            CHUP     = character rotation vector.
            MONO     = monospaced font flag.
            SCRTCHP  = ptr to start of text special effects buffer.
            SCRPT2   = offset of scaling buffer in above buffer.

    output: none.
```

(9) Show mouse

```
    ...     ...
    dc.w    $A009    ; Show the mouse.
    ...     ...
```

input: see GEM VDI manual.

output: none.

(10) Hide mouse

```
    ...     ...
    dc.w    $A00A    ; Hide the mouse.
    ...     ...
```

input: see GEM VDI manual.

output: none.

(11) Transform mouse

```
    ...     ...
    dc.w    $A00B    ; Transform the mouse's form.
    ...     ...
```

input: see GEM VDI manual.

output: none.

(12) Undraw sprite

```
    ...     ...
    dc.w    $A00C    ; Undraw the previously drawn sprite.
    ...     ...
```

input: a2 = ptr to sprite save block.

note: The sprite save block is used to save the screen underneath the sprite. Its size is 10 bytes + 64 bytes per plane, i.e. (10 + VPLANES * 64) bytes.

874

output: clobbers a6. ("C" programmers beware.)

(13) Draw sprite

```
...     ...
dc.w    $A00D   ; Draw a sprite.
...     ...
```

input: d0 = x hot-spot.
d1 = y hot_spot.
a0 = ptr to sprite definition block.
a2 = ptr to sprite save block.

SPRITE DEFINTION BLOCK LAYOUT

```
ds.w  1      x offset of hot-spot.
ds.w  1      y offset of hot-spot.
ds.w  1      format flag. (1 => VDI Format,
                          -1 => XOR Format)
```

VDI Format

fg bit	bg bit	action
0	0	transparent to screen
0	1	background color plotted
1	0	foreground color plotted
1	1	foreground color plotted

XOR Format

fg bit	bg bit	action
0	0	transparent to screen
0	1	background color plotted
1	0	xor screen
1	1	foreground color plotted

```
ds.w  1      background color (color table index)
ds.w  1      foreground color (color table index)
ds.w  32     interleaved background/foreground image.
             (word 0 = background line 0.
              word 1 = foreground line 0.
              word 2 = background line 1.
              word 3 = foreground line 1.
              etc.)
```

output: clobbers a6. ("C" programmers beware.)

bugs: This function is not usable as a LINE "A" call in the 1st release of TOS. See Example Program #2 below for the technique one must adopt to use this function.

(14) Copy raster form

```
...     ...
dc.w    $A00E   ; Copy a raster form from source to destination.
...     ...
```

input: See the VDI discussion of Copy Raster, Opaque & Transparent, EXCEPT, CONTRL(0), CONTRL(1), CONTRL(3), and CONTRL(6) are

ignored.
COPYTRAN = Opaque/Transparent mode flag. (0 => Opaque)

output: none.

note: See the BitBlt discussion above.

USING THE LINE "A" INTERFACE

The inputs to the LINE "A" routines are contained in a structure pointed to by the value returned in a0 after an initialization call ($A000) has been made. This initialization only needs to be done once and any returned values can be saved and used as needed.

The LINE "A" interface can be used in cooperation with the VDI and AES, however, one cannot expect the variables below to be unchanged after the VDI or AES has been used. Therefore, if an application wants to mix calls to LINE "A" and VDI/AES, it must reload any variables that it uses as input to the LINE "A" routines.

The caller should assume that registers d0-d2 and a0-a2 are clobbered upon return. The rest are preserved.

The LINE"A" input variables structure:

offset	name	type	description
0	VPLANES	word	number of video planes.
2	VWRAP	word	number of bytes/video line.

note: These variables can be changed to implement special effects, e.g.,doubling VWRAP will cause the routines to skip 1 scan-line between every scanline that is output to the screen. Of course, any modifications made to these variables must be undone when normal operation of the LINE "A" (or VDI) is desired.

offset	name	type	description
4	CONTRL	long	ptr to the CONTRL array.
8	INTIN	long	ptr to the INTIN array.
12	PTSIN	long	ptr to the PTSIN array.
16	INTOUT	long	ptr to the INTOUT array.
20	PTSOUT	long	ptr to the PTSOUT array.

note: See the GEM VDI manual for a discussion of the above arrays.

offset	name	type	description
24	COLBIT0	word	current color bit-plane 0 value.
26	COLBIT1	word	current color bit-plane 1 value.
28	COLBIT2	word	current color bit-plane 2 value.
30	COLBIT3	word	current color bit-plane 3 value.

note: current foreground writing color = 1*COLBIT0 +
2*COLBIT1 +
4*COLBIT2 +
8*COLBIT3.

```
 32     LSTLIN          word    set this to -1 and forget it.
 34     LNMASK          word    equivalent to VDI's line style.
 36     WMODE           word    writing mode.  (0 => replace mode,
                                                1 => transparent mode,
                                                2 => xor mode,
                                                3 => inverse trans mode.)

        note: see VDI manual for discussion of writing modes.

 38     X1              word    x1 coordinate.
 40     Y1              word    y1 coordinate.
 42     X2              word    x2 coordinate.
 44     Y2              word    y2 coordinate.
 46     PATPTR          long    ptr to the current fill pattern.
 50     PATMSK          word    fill pattern "mask".
 52     MFILL           word    multi-plane fill flag.
                                (0 => current fill pattern is single plane)
                                (1 => current fill pattern is multi-plane)

 54     CLIP            word    clipping flag (0 => no clipping)
 56     XMINCL          word    minimum x clipping value.
 58     YMINCL          word    minimum y clipping value.
 60     XMAXCL          word    maximum x clipping value.
 62     YMAXCL          word    maximum y clipping value.

 64     XDDA            word    accumulator for textblt x dda.

        note:   Should be inited to 8000H (.5) before each invocation
                of TextBlt.

 66     DDAINC          word    fractional amount to scale up or down.

        note:   If scaling up, set DDAINC to
                256*(Intended size-Actual size)/Actual size.

                If scaling down, set DDAINC to
                256*Intended size/Actual size.

 68     SCALDIR         word    scale direction flag. (0 => down)
 70     MONO            word    0 => current font is not monospaced OR
                                     its OK for thickening to increase the
                                     width of the current font.
                                1 => current font is monospaced AND thickenin
                                     may not increase the width of the font.

 72     SOURCEX         word    x coord of character in font form.
 74     SOURCEY         word    y coord of character in font form.

        note:   SOURCEX can be computed from the information held in the
                font header. (see Appendix G of VDI manual for header def)
                e.g.    temp = character value;
                        temp -= fnt_ptr->first_ade;
                        SOURCEX = fnt_ptr->off_table(temp);

                SOURCEY is typically set to 0. (top line of font form)

 76     DESTX           word    x coord of character on screen.
 78     DESTY           word    y coord of character on screen.
 80     DELX            word    width of character.
 82     DELY            word    height of character.
```

```
        note:   DELX & DELY can be computed from the font header.
                e.g.    temp = character value;
                        temp -= fnt_ptr->first_ade;
                        SOURCEX = fnt_ptr->off_table(temp);
                        DELX = fnt_ptr->offtable(temp+1)-SOURCEX;
                        DELY = fnt_ptr->form_height;

 84     FBASE           long    ptr to start of font data. (font form)
 88     FWIDTH          word    width of font form.

        note:   FBASE & FWIDTH can be computed from the font header.
                e.g.    FBASE = fnt_ptr->dat_table;
                        FWIDTH = fnt_ptr->form_width;

 90     STYLE           word    vector of TextBlt special effects flags.
                                        Bit 0 = Thicken flag.
                                        Bit 1 = Lighten flag.
                                        Bit 2 = Skewing flag.
                                        Bit 3 = Underline flag. (ignored)
                                        Bit 4 = Outline flag.

        note:   Set the bits to select the desired effects.
                Underlining must be done by the application.

 92     LITEMASK        word    the mask to use in lightening text.
 94     SKEWMASK        word    the mask to use in skewing text.
 96     WEIGHT          word    the width by which to thicken text.
 98     ROFF            word    offset above character baseline when skewing.
100     LOFF            word    offset below character baseline when skewing.

        note:   The above 5 input variables can be computed from the font
                header.
                e.g.    LITEMASK = fnt_ptr->lighten;
                        SKEWMASK = fnt_ptr->skew;
                        WEIGHT = fnt_ptr->thicken;
                        if (skewing) {
                          ROFF = fnt_ptr->right_offset;
                          LOFF = fnt_ptr->left_offset;
                        }
                        else {
                          ROFF = 0;
                          LOFF = 0;
                        }

102     SCALE           word    scaling flag. (0 => no scaling.)
104     CHUP            word    character rotation vector.
                                0 => normal horizontal orientation.
                                900 => rotated 90 degrees clockwise.
                                1800 => rotated 180 degrees clockwise.
                                2700 => rotated 270 degrees clockwise.

106     TEXTFG  word    text foreground color.

108     SCRTCHP         long    ptr to start of text special effects buffer.
112     SCRPT2          word    offset of scaling buffer in above buffer.

        note:   These special effects buffer pointers must be initialized
                before TextBlt effects can be used.

114     TEXTBG  word    text background color. (4/20/85) RAMVDI only.
116     COPYTRAN        word    copy raster form type flag. (4/26/85) RAMVDI.
```

```
                                     0 => Opaque type
                                          n-plane source -> n-plane dest
                                          BitBlt writing modes
                                    ~0 => Transparent type
                                          1-plane source -> n-plane dest
                                          VDI writing modes

118     SEEDABORT       long    ptr to routine which is called within the
                                    seedfill logic to allow the fill to be
                                    aborted.  Initialized to point to a
                                    dummy routine which returns FALSE.
                                    Returning TRUE aborts the seedfill.

        note:   This ptr doesn't exist in 1st release of TOS.  See Example
                Program #2 for the technique to use to identify the 1st TOS
                release.
```

EXAMPLE LINE "A" EQUATES

```
*
*
*
VPLANES         equ             0
VWRAP           equ             2
CONTRL          equ             4
INTIN           equ             8
PTSIN           equ             12
INTOUT          equ             16
PTSOUT          equ             20
COLBIT0         equ             24
COLBIT1         equ             26
COLBIT2         equ             28
COLBIT3         equ             30
LSTLIN          equ             32
LNMASK          equ             34
WMODE           equ             36
X1              equ             38
Y1              equ             40
X2              equ             42
Y2              equ             44
PATPTR          equ             46
PATMSK          equ             50
MFILL           equ             52
CLIP            equ             54
XMINCL          equ             56
YMINCL          equ             58
XMAXCL          equ             60
YMAXCL          equ             62
XDDA            equ             64
DDAINC          equ             66
SCALDIR         equ             68
MONO            equ             70
SRCX            equ             72
SRCY            equ             74
DSTX            equ             76
DSTY            equ             78
DELX            equ             80
DELY            equ             82
```

```
FBASE           equu    84
FWIDTH          equ     88
STYLE           equ     90
LITEMSK         equ     92
SKEWMSK         equ     94
WEIGHT          equ     96
ROFF            equ     98
LOFF            equ     100
SCALE           equ     102
CHUP            equ     104
TEXTFG          equ     106
SCRTCHP         equ     108
SCRPT2          equ     112
TEXTBG          equ     114
COPYTRAN        equ     116
SEEDABORT       equ     118
*
*
*
INIT            equ     $A000
PUTPIX          equ     INIT+1
GETPIX          equ     INIT+2
ABLINE          equ     INIT+3
HABLINE         equ     INIT+4
RECTFILL        equ     INIT+5
POLYFILL        equ     INIT+6
BITBLT          equ     INIT+7
TEXTBLT         equ     INIT+8
SHOWCUR         equ     INIT+9
HIDECUR         equ     INIT+10
CHGCUR          equ     INIT+11
DRSPRITE        equ     INIT+12
UNSPRITE        equ     INIT+13
COPYRSTR        equ     INIT+14
SEEDFILL        equ     INIT+15
```

EXAMPLE PROGRAM #1

```
                text

start:          dc.w    INIT                    ; initialize.
                move.w  #-1,LSTLIN(a0)          ; once and for all.
                move.w  #$5555,LNMASK(a0)       ; dithered line.
                move.w  #0,WMODE(a0)            ; replace mode.
                move.w  #1,COLBIT0(a0)
                move.w  #1,COLBIT1(a0)
                move.w  #1,COLBIT2(a0)
                move.w  #0,COLBIT3(a0)          ; drawing color = 7.
                move.w  #0,X1(a0)               ; X1 = 0.
                move.w  #0,Y1(a0)               ; Y1 = 0.
                move.w  #99,X2(a0)              ; X2 = 99.
                move.w  #99,Y2(a0)              ; Y2 = 99.
                dc.w    ABLINE                  ; draw line.
                .
                .
                .
                move.w  #0,-(sp)
                trap    #1                      ; exit.
                end
```

EXAMPLE PROGRAM #2

```
                text
*
*
*
start:          clr.l   -(sp)
                move.w  #$20,-(sp)
                trap    #1                  ; supervisor mode required to use
*                                           ;    line "A" routines via jsr.
                addq    #6,sp
                move.l  d0,stksave          ; save old stack ptr.
*
*       Find out which version of LINE "A" handler exists.
*
                move.l  #0,a2               ; convenient value for testing.
                dc.w    INIT                ; line "A" initialization.
                move.l  a2,d2               ; old version?
                bne     a2ok                ; no, a2 points to array of line "A"
*                                           ;     routine addresses.
                lea     -4*15(a1),a2        ; yes, a2 is untouched, so use a1 plu
*                                           ;     displacement (15 addresses).
*
*       a2 now points to array of line "A" routine addresses.
*
a2ok:           move.l  4*$D(a2),drawaddr ; fetch draw routine address.
*
*       Bug-workaround/Initialization complete.
*
                move.w  #0,d0               ; init x.
                move.w  #0,d1               ; init y.
                lea     sprite,a0           ; point to sprite.
                lea     save,a2             ; point to save area.

loop:           movem.w d0-d1,-(sp)         ; save x,y.
                movem.l a0/a2,-(sp)         ; save ptrs.
                move.l  a6,-(sp)            ; draw clobbers a6.
                tst.w   old_linea           ; old or new line "A" handler?
                beq     new                 ; new, branch.
                move.l  drawaddr,a3         ; fetch draw routine address.
                jsr     (a3)                ; draw the old way.
                bra     merge
*
new:            dc.w    DRSPRITE            ; draw the new way.
*
merge:          move.l  (sp)+,a6
                movem.l (sp)+,a0/a2         ; restore ptrs.
*
                move.w  #2000,d2
wait:           dbra    d2,wait             ; wait a bit.
*
                movem.l a0/a2,-(sp)         ; save ptrs.
                move.l  a6,-(sp)            ; undraw clobbers a6.
                dc.w    UNSPRITE
                move.l  (sp)+,a6
                movem.l (sp)+,a0/a2         ; restore ptrs.
                movem.w (sp)+,d0-d1         ; restore x,y.
                addq.w  #1,d0               ; inc x.
```

881

```
                cmp.w   #640,d0
                ble     loop
*
                move.l  stksave,-(sp)
                move.w  #$20,-(sp)
                trap    #1                ; user mode.
                addq    #6,sp
*
                move.w  #0,-(sp)
                trap    #1                ; exit.

                data
*
*
*
sprite:         dc.w            0,0       ; x,y offsets of hotspot.
                dc.w            1,0,1     ; format, background, foreground.
bob:            dc.w            $FFFF     ; background line 0.
                dc.w            $07F0     ; foreground line 0.
                dc.w            $FFFF
                dc.w            $0ff8
                dc.w            $FFFF
                dc.w            $1fec
                dc.w            $FFFF
                dc.w            $1804
                dc.w            $FFFF
                dc.w            $1804
                dc.w            $FFFF
                dc.w            $1004
                dc.w            $FFFF
                dc.w            $1e3c
                dc.w            $FFFF
                dc.w            $1754
                dc.w            $FFFF
                dc.w            $1104
                dc.w            $FFFF
                dc.w            $0b28
                dc.w            $FFFF
                dc.w            $0dd8
                dc.w            $FFFF
                dc.w            $0628
                dc.w            $FFFF
                dc.w            $07d0
                dc.w            $FFFF
                dc.w            $2e10
                dc.w            $FFFF
                dc.w            $39e0
                dc.w            $FFFF
                dc.w            $3800


                bss
*
*
*
stksave:        ds.l            1
save:           ds.b            10+64
old_linea:      ds.w            1
drawaddr:       ds.l            1
                end
```