APPENDIX B. GRAPHICS USING BIT-MAPPED VIDEO

This appendix summarizes the technique of bit-mapping that is used to produce the display on the GRiD Compass. A collection of assembly language routines at the end of the appendix offers a programmer the fundamental tools to create graphic displays and patterns.

VIDEO DISPLAY ARCHITECTURE

For each pixel that appears on the display screen a bit position is reserved within a portion of RAM that is dedicated to the video display. This one-to-one correspondence allows unlimited manipulation of the video screen graphic displays. Bytes written to the video memory area directly form the bit pattern that is shifted out to produce the display.

The screen dimensions extend 320 pixels horizontally and 240 pixels vertically. Although the bit patterns are written to the video memory 8 or 16 bits at a time by the CPU, they are always accessed 16 bits at a time by the video display circuitry. Twenty of these 16-bit words compose each row of pixels (slice) of the display screen. The full 240 rows of video display require 4800 words which are stored between the RAM memory addresses 00400 through 0297F hex.

Figure B-1 illustrates the manner in which the bit-mapping relates

RAM memory locations to the bit positions on the display screen.

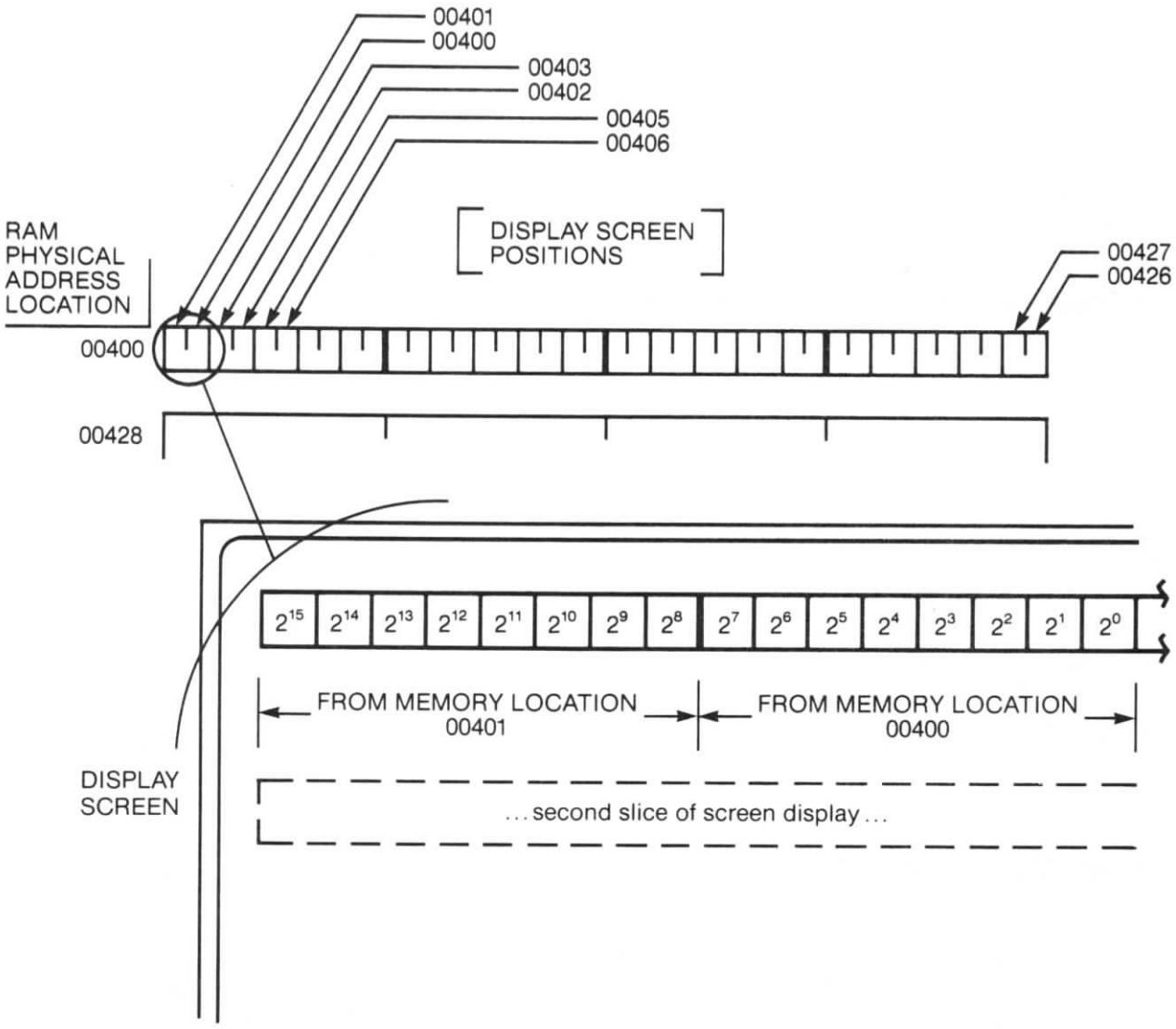


Figure B-1. Bit map for Video Display

Figure B-2 illustrates how a word stored at two consecutive memory locations appears when written to the video display.

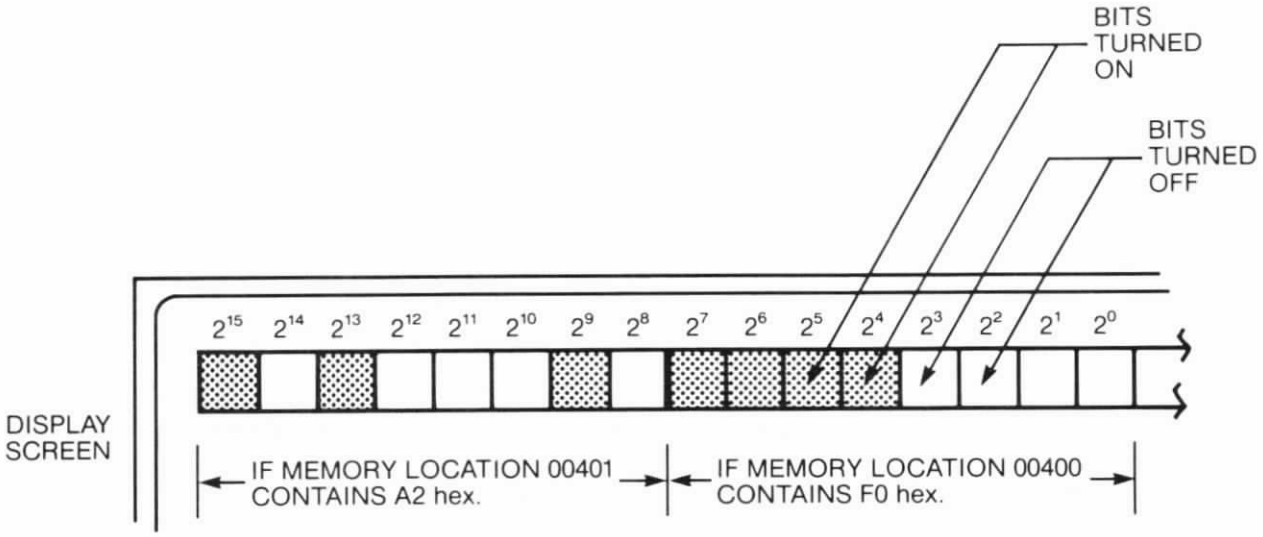
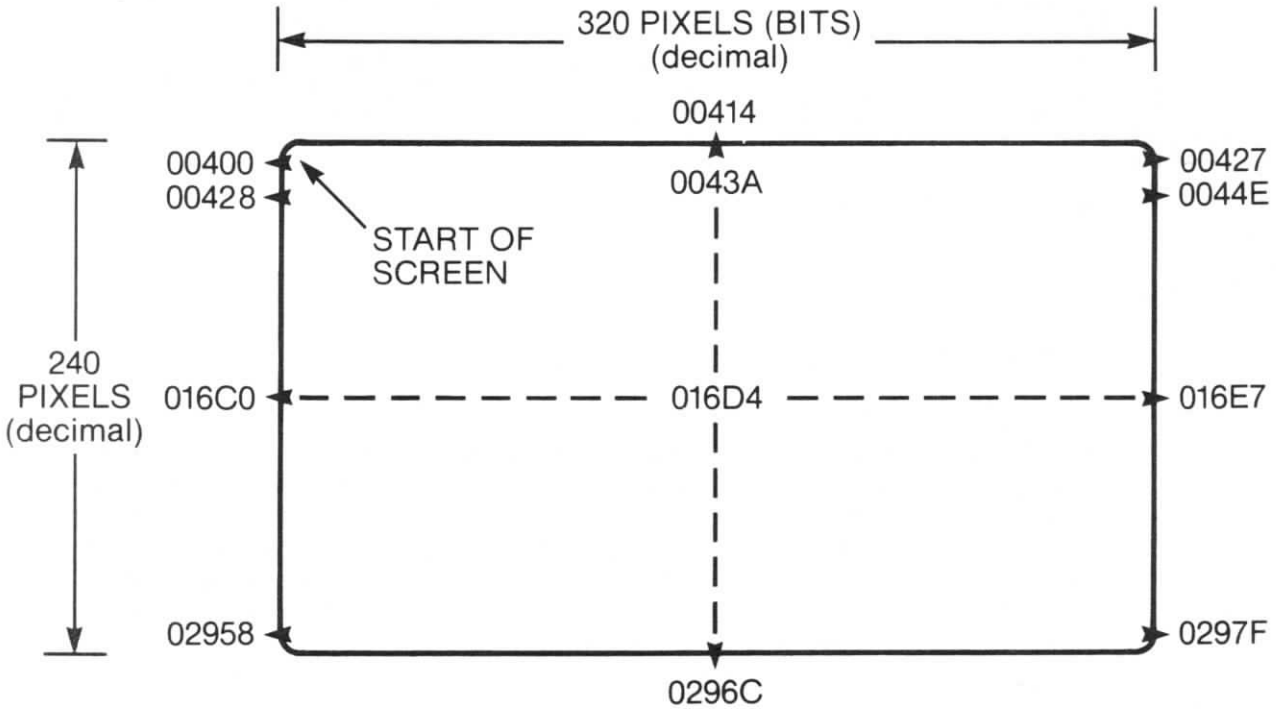


Figure B-2. Example of Bit-Mapping

Writing bytes to the video portion of memory requires no synchronization with the video timing circuitry; the computer's hardware ensures that actual timing matters are transparent to the programmer. By mastering a set of fundamental assembly language routines, the programmer can turn on and off the desired bits to create any possible pattern. Several of these routines follow, including basic descriptions of how to draw a line between two designated points, how to invert the bit pattern, how to erase a line, and how to move pixels about on the display screen.

Additional information concerning algorithms for graphic display can be obtained from the reference, Principles of Interactive Computer Graphics by Newman and Sproull, published by McGraw-Hill in 1979.

Figure B-3 shows addresses in the video memory corresponding with positions on the display screen.



*All addresses expressed in hexadecimal.

Figure B-3. Address Correspondence for Video Display.

The following assembly language routines can be used individually or combined to form intricate graphic patterns.

```
NAME  Grafix

CGROUP GROUP CODE

;*****************************************************
;*   PUBLIC declaration of Graphics Procedures
;*****************************************************

PUBLIC  GfxSetPixel, GfxClrPixel, GfxInvertPixel, GfxTestPixel
PUBLIC  GfxDrawLine, GfxEraseLine, GfxInvertLine

;*****************************************************
;*  constant definitions for screen width, height,
;*  and masks for facilitating working with
;*  bit patterns
;*****************************************************

zero        EQU  0
scrWidth    EQU  320
scrHeight   EQU  240
wordWidth   EQU  scrWidth/16    ; screen width in words
byteWidth   EQU  wordWidth*2
evenMask    EQU  0FFFEH         ; make a word even
true        EQU  0FFFFH

display  SEGMENT AT 40H
display  ENDS
```

```
CODE SEGMENT PUBLIC 'CODE'
ASSUME  CS:CGROUP, DS:display

;*****************************************************
;*
;*     GfxSetpixel(x, y: Word)
;*
;* This will set the pixel at X = horizontal
;* and Y = vertical.
;* (0,0) is the upper lefthand corner.
;*
;* REGISTERS CHANGED: AX, BX, CX, DX, ES, DI, SI
;*
;*****************************************************

GfxSetPixel PROC NEAR
  PUSH  BP
  PUSH  DS
  MOV   BP, SP

  MOV   CX, [BP+6H]          ; CX = y
  MOV   BX, [BP+8H]          ; BX = x
  MOV   SI, 8000H
  CALL  SetUpPixel

  OR    [BX], SI             ; set the pixel

  POP   DS
  POP   BP
  RET   4
GfxSetPixel ENDP
```

```
;*******************************************************
;*
;*    GfxClrPixel(x, y : word)
;*
;* This is exactly the same as SetPixel except
;* for the last two statements
;*
;*******************************************************

GfxClrPixel PROC NEAR
  PUSH  BP
  PUSH  DS
  MOV   BP, SP

  MOV   CX, [BP+6H]
  MOV   BX, [BP+8H]
  MOV   SI, 7FFFH
  CALL  SetUpPixel

  AND   [BX], SI

  POP   DS
  POP   BP
  RET   4
GfxClrPixel ENDP
```

```
;*******************************************************
;*
;*    GfxInvertPixel(x, y : word)
;*
;* This is exactly the same as SetPixel except
;* for the last two statements
;*
;*******************************************************

GfxInvertPixel PROC NEAR
  PUSH  BP
  PUSH  DS
  MOV   BP, SP

  MOV   CX, [BP+6H]
  MOV   BX, [BP+8H]
  MOV   SI, 8000H
  CALL  SetUpPixel

  XOR   [BX], SI

  POP   DS
  POP   BP
  RET   4
GfxInvertPixel ENDP
```

```
;*****************************************************
;*
;*    GfxTestPixel(x, y : word)
;*
;* This is exactly the same as SetPixel except
;* for the last two statements
;*
;*****************************************************

GfxTestPixel PROC NEAR
  PUSH  BP
  PUSH  DS
  MOV   BP, SP

  MOV   CX, [BP+6H]
  MOV   BX, [BP+8H]
  MOV   SI, 8000H
  CALL  SetUpPixel

  MOV   AX, [BX]        ;Get the word
  AND   AX, SI          ;Mask out all other bits
  JZ    NotSet          ;do nothing if not set
  MOV   ASet to true if bit is set.
NotSet:

  POP   DS
  POP   BP
  RET   4
GfxTestPixel ENDP
```

```
;*************************************************
;*
;*    SetUpPixel
;*
;*  entry:  BX = x
;*          CX = y
;*          SI = original mask
;*
;*  Does the setup for Set- and Clr- Pixel.
;*
;*  exit:  BX ^ word in display
;*         ES ^ start of display buffer
;*         SI = rotated mask
;*
;*************************************************

SetUpPixel PROC  NEAR
  MOV  AX, display
  MOV  DS, AX                ; DS ^ start of display

  MOV  AX, CX
  MOV  DX, byteWidth
  MUL  DX
  MOV  CX, AX                ; CX = CX * byteWidth

  MOV  AX, BX                ; AX = x
  SAR  BX, 1
  SAR  BX, 1
  SAR  BX, 1
  AND  BX, evenMask          ; BX = EVEN(x DIV 8)
  ADD  BX, CX                ; BX ^ word in display

  AND  AX, 000FH             ; AX = x MOD 16
  MOV  CX, AX
  ROR  SI, CL                ; rotate to position

  RET
SetUpPixel ENDP
```

```
;*********************************************************
;*
;*    GfxDrawLine(x1, y1, x2, y2)
;*
;* This will draw a line between the two points (x1, y1) and
;* (x2, y2).  It uses the DDA algorithm.  The act of
;* drawing a line is split into two cases for efficiency.
;*
;*  case AB: dx >= dy
;*  case CD: dx < dy
;*
;* During the inner loop of each case, the registers mean:
;*
;*  AX = dx             BX = dy
;*  CX = loop counter   DX = temp
;*  SI = mask           DI ^ display buffer
;*  BP = +- wordWidth
;*
;*********************************************************

GfxDrawLine PROC NEAR
  PUSH  BP
  PUSH  DS
  MOV   BP, SP

  MOV   DX, [BP+06H]          ; DX = y2
  MOV   CX, [BP+08H]          ; CX = x2
  MOV   BX, [BP+0AH]          ; BX = y1
  MOV   AX, [BP+0CH]          ; AX = x1

  CMP   AX, CX                ; x1 <= x2 ?
  JLE   Line10
  XCHG  AX, CX
  XCHG  BX, DX                ; now x1 <= x2
Line10:

  SUB   CX, AX                ; CX = dx ( >= 0)
  SUB   DX, BX                ; DX = dy
  MOV   SI,8000H              ; SI = original mask
  CALL  AInitLine
  OR    [DI],SI
  CMP   AX, BX                ; dx < dy ?
  JL    CaseCD                ; yes -> case CD

CaseAB:             ; case AB
  MOV   CX, AX                ; loop 'dx' times
  MOV   DX, AX
  CMP   DX, 1
  JLE   NoShift
  SAR   DX, 1
NoShift:
```

```
  NEG   DX                      ; DX = -dx/2; DX > 0!!
  JCXZ  LoopDone

LoopOnAB:
  ADD   DX, BX                  ; temp += dy
  JS    A10
  SUB   DX, AX                  ; temp -= dx
  ADD   DI, BP                  ; y += wordWidth
A10:

  ROR   SI, 1                   ; x += 1
  JNC   SameX
  ADD   DI, 2                   ; DI ^ next word
SameX:
  OR    [DI], SI                ; set the bit
  LOOP  LoopOnAB
  JMP   LoopDone

CaseCD:             ; case CD
  MOV   CX, BX                  ; loop 'dy' times
  MOV   DX, BX
  CMP   DX, 1
  JLE   NoShift2
  SAR   DX, 1
NoShift2:
  NEG   DX                      ; DX = -dy / 2
  JCXZ  LoopDone

LoopOnCD:
  ADD   DX, AX                  ; temp += dx
  JS    C10
  SUB   DX, BX                  ; temp -= dy
  ROR   SI, 1                   ; x += 1
  JNC   C10
  ADD   DI, 2                   ; DI ^ next word

C10:
  ADD   DI, BP                  ; next row of bytes
  OR    [DI], SI                ; set the bit
  LOOP  LoopOnCD

LoopDone:
  POP   DS
  POP   BP
  RET   8
GfxDrawLine ENDP
```

```
;**********************************
;*                                *
;*  GfxEraseLine(x1, y1, x2, y2)  *
;*                                *
;**********************************

GfxEraseLine PROC NEAR
  PUSH  BP
  PUSH  DS
  MOV   BP, SP

  MOV   DX, [BP+06H]          ; DX = y2
  MOV   CX, [BP+08H]          ; CX = x2
  MOV   BX, [BP+0AH]          ; BX = y1
  MOV   AX, [BP+0CH]          ; AX = x1

  CMP   AX, CX                ; x1 <= x2 ?
  JLE   ELine10
  XCHG  AX, CX
  XCHG  BX, DX                ; now x1 < x2

ELine10:
  SUB   CX, AX                ; CX = dx ( >= 0 )
  SUB   DX, BX                ; DX = dy
  MOV   SI, 7FFFH             ; SI = original mask
  CALL  AInitLine
  AND   [DI], SI
  CMP   AX, BX                ; dx < dy ?
  JL    ECaseCD               ; yes -> case CD

ECaseAB:            ; case AB
  MOV   CX, AX                ; loop 'dx' times
  MOV   DX, AX
  CMP   DX, 1
  JLE   ENoShift
  SHR   DX, 1
ENoShift:
  NEG   DX                    ; DX = - dx / 2
  JCXZ  ELoopDone

ELoopOnAB:
  ADD   DX, BX                ; temp += dy
  JS    ESameY                ; next y value ?
  SUB   DX, AX                ; temp -= dx
  ADD   DI, BP                ; y += wordWidth
ESameY:
  ROR   SI, 1                 ; x += 1
  JC    ESameX                ; on to next word?
  ADD   DI, 2                 ; yes
ESameX:
  AND   [DI], SI              ; erase the bit
```

```
  LOOP  ELoopOnAB
  JMP   ELoopDone             ; all finished

ECaseCD:            ; case CD
  MOV   CX, BX              ; loop 'dy' times
  MOV   DX, BX
  CMP   DX, 1
  JLE   ENoShift2
  SHR   DX, 1
ENoShift2:
  NEG   DX                  ; DX = -dy / 2
  JCXZ  ELoopDone
ELoopOnCD:
  ADD   DX, AX              ; temp += dx
  JS    EC10
  SUB   DX, BX              ; temp -= dy
  ROR   SI, 1               ; x += 1
  JC    EC10
  ADD   DI, 2               ; DI ^ next word
EC10:
  ADD   DI, BP              ; y += increment
  AND   [DI], SI            ; clear the bit
  LOOP  ELoopOnCD

ELoopDone:
  POP   DS
  POP   BP
  RET   8
GfxEraseLine ENDP
```

```
;****************************************
;*
;*    GfxInvertLine(X, Y, X1, Y1)
;*
;****************************************

GfxInvertLine PROC NEAR
  PUSH  BP
  PUSH  DS
  MOV  BP,SP
  MOV  AX,display
  MOV  DS,AX

  MOV   DX, [BP+06H]          ; DX = y2
  MOV   CX, [BP+08H]          ; CX = x2
  MOV   BX, [BP+0AH]          ; BX = y1
  MOV   AX, [BP+0CH]          ; AX = x1

  CMP  AX, CX    ; IS X <= X1
  JLE  XLine10    ; YES -> JUMP

  XCHG  AX, CX
  XCHG  BX, DX    ; NOW X < X1
XLine10:
  SUB  CX, AX    ; CX = dx (dx >= 0)
  SUB  DX, BX    ; DX = dy

  MOV   SI, 8000H         ; SI = original mask
  CALL  AInitLine
  XOR   [DI], SI

  CMP  AX, BX    ; IS dx < dy ?
  JL  XCaseCD    ; YES -> GOTO CASES C,D

XCaseAB:      ; MUST BE CASE A OR B
  MOV  CX, AX    ; LOOP dx TIMES
  MOV  DX, AX
  CMP   DX, 1
  JLE   XNoShift
  SHR   DX, 1
XNoShift:
  NEG  DX    ; DX = - dx /2
  JCXZ  XLoopDone
XLoopOnAB:
  ADD  DX, BX    ; temp += dy
  JS  XA10    ; JUMP IF RESULT < 0
  SUB  DX, AX    ; temp -= dx
  ADD  DI,BP    ; Y += INC Y
XA10:
  ROR  SI,1    ; X += 1
  JNC  XA20
```

```
  ADD  DI,2     ; DI ^ next word
XA20:
  XOR  [DI],SI     ; SET THE BIT
  LOOP  XLoopOnAB

  JMP   XLoopDone

XCaseCD:       ; MUST BE CASE C OR D
  MOV  CX, BX     ; LOOP dy TIMES
  MOV  DX, BX
  CMP   DX, 1
  JLE   XNoShift2
  SHR   DX, 1
XNoShift2:
  NEG  DX     ; DX = -dy / 2
  JCXZ  XLoopDone
XLoopOnCD:
  ADD  DX, AX     ; temp += dx
  JS  XC10     ; JUMP IF RESULT < 0
  SUB  DX, BX     ; temp -= dy
  ROR  SI,1     ; X += 1
  JNC  XC10
  ADD  DI,2     ; DI ^ next word
XC10:
  ADD  DI,BP     ; Y += INC Y
  XOR  [DI],SI     ; SET THE BIT
  LOOP  XLoopOnCD

XLoopDone:
  POP  DS
  POP  BP
  RET  8
GfxInvertLine ENDP
```

```
;*************************************************
;*
;*    AInitLine
;*
;*  entry:  AX = x
;*          BX = y
;*          CX = dx    (dx >= 0)
;*          DX = dy
;*          SI = original mask
;*
;*  exit:   SI = rotated mask
;*          DI ^ word in display
;*          AX = dx
;*          BX = dy
;*          BP = +- wordWidth
;*
;*************************************************

AInitLine PROC   NEAR
  PUSH  CX
  PUSH  DX

  PUSH  AX              ; save during multiply

  MOV   AX, display     ; screen is data segment
  MOV   DS, AX

  MOV   AX, BX
  MOV   CX, byteWidth
  MUL   CX
  MOV   DI, AX          ; DI = y * byteWidth
  POP   AX

  MOV   CX, AX
  SAR   CX, 1
  SAR   CX, 1
  SAR   CX, 1
  AND   CX, evenMask    ; CX = EVEN(x DIV 8)
  ADD   DI, CX          ; DI ^ word in display

  AND   AX, 000FH       ; AX = x MOD 16
  MOV   CX, AX
  ROR   SI, CL          ; rotate to starting pos

  POP   BX              ; BX = dy
  POP   AX              ; AX = dx

  MOV   BP, byteWidth
  CMP   BX, zero
  JGE   DYGE0           ; dy >= 0
  NEG   BP
```

```
  NEG   BX                ; dy = |dy|
DYGEO:
  RET
AInitLine ENDP

CODE ENDS
  END
```