Chapter 26

VGA Write Mode 3

Chapter 26

The Write Mode That Grows on You

Over the last three chapters, we've covered the VGA's write path from stem to stern—with one exception. Thus far, we've only looked at how writes work in write mode 0, the straightforward, workhorse mode in which each byte that the CPU writes to display memory fans out across the four planes. (Actually, we also took a quick look at write mode 1, in which the latches are always copied unmodified, but since exactly the same result can be achieved by setting the Bit Mask register to 0 in write mode 0, write mode 1 is of little real significance.)

Write mode 0 is a very useful mode, but some of VGA's most interesting capabilities involve the two write modes that we have yet to examine: write mode 1, and, especially, write mode 3. We'll get to write mode 1 in the next chapter, but right now I want to focus on write mode 3, which can be confusing at first, but turns out to be quite a bit more powerful than one might initially think.

A Mode Born in Strangeness

Write mode 3 is strange indeed, and its use is not immediately obvious. The first time I encountered write mode 3, I understood immediately how it functioned, but could think of very few useful applications for it. As time passed, and as I came to understand the atrocious performance characteristics of **OUT** instructions, and the importance of text and pattern drawing as well, write mode 3 grew considerably in my estimation. In fact, my esteem for this mode ultimately reached the point where in the last major chunk of 16-color graphics code I wrote, write mode 3 was used more than write mode 0 overall, excluding simple pixel copying. So write mode 3 is well worth using, but to use it you must first understand it. Here's how it works.

In write mode 3, set/reset is automatically enabled for all four planes (the Enable Set/Reset register is ignored). The CPU data byte is rotated and then ANDed with the contents of the Bit Mask register, and the result of this operation is used as the contents of the Bit Mask register alone would normally be used. (If this is Greek to you, have a look back at Chapters 23 through 25. There's no way to understand write mode 3 without understanding the rest of the VGA's write data path first.)

That's what write mode 3 does—but what is it *for?* It turns out that write mode 3 is excellent for a surprisingly large number of purposes, because it makes it possible to avoid the bane of VGA performance, **OUT**s. Some uses for write mode 3 include lines, circles, and solid and two-color pattern fills. Most importantly, write mode 3 is ideal for transparent text; that is, it makes it possible to draw text in 16-color graphics mode quickly without wiping out the background in the process. (As we'll see at the end of this chapter, write mode 3 is potentially terrific for opaque text—text drawn with the character box filled in with a solid color—as well.)

Listing 26.1 is a modification of code I presented in Chapter 25. That code used the data rotate and bit mask features of the VGA to draw bit-mapped text in write mode 0. Listing 26.1 uses write mode 3 in place of the bit mask to draw bit-mapped text, and in the process gains the useful ability to preserve the background into which the text is being drawn. Where the original text-drawing code drew the entire character box for each character, with 0 bits in the font pattern causing a black box to appear around each character, the code in Listing 26.1 affects display memory only when 1 bits in the font pattern are drawn. As a result, the characters appear to be painted into the background, rather than over it. Another advantage of the code in Listing 26.1 is that the characters can be drawn in any of the 16 available colors.

LISTING 26.1 L26-1.ASM

```
; Program to illustrate operation of write mode 3 of the VGA.
;  Draws 8x8 characters at arbitrary locations without disturbing
;  the background, using VGA's 8x8 ROM font.  Designed
;  for use with modes 0Dh, 0Eh, 0Fh, 10h, and 12h.
; Runs only on VGAs (in Models 50 & up and IBM Display Adapter
;  and 100% compatibles).
; Assembled with MASM
; By Michael Abrash
;
stack   segment para stack 'STACK'
        db      512 dup(?)
stack   ends
;
VGA_VIDEO_SEGMENT       equ     0a000h  ;VGA display memory segment
SCREEN_WIDTH_IN_BYTES   equ     044ah   ;offset of BIOS variable
FONT_CHARACTER_SIZE     equ     8       ;# bytes in each font char
;
; VGA register equates.
;
```

```
SC_INDEX        equ     3c4h    ;SC index register
SC_MAP_MASK     equ     2       ;SC map mask register index
GC_INDEX        equ     3ceh    ;GC index register
GC_SET_RESET    equ     0       ;GC set/reset register index
GC_ENABLE_SET_RESET equ 1       ;GC enable set/reset register index
GC_ROTATE       equ     3       ;GC data rotate/logical function
                                ; register index
GC_MODE         equ     5       ;GC Mode register
GC_BIT_MASK     equ     8       ;GC bit mask register index
;
dseg    segment para common 'DATA'
TEST_TEXT_ROW   equ     69      ;row to display test text at
TEST_TEXT_COL   equ     17      ;column to display test text at
TEST_TEXT_WIDTH equ     8       ;width of a character in pixels
TestString      label   byte
        db      'Hello, world!',0       ;test string to print.
FontPointer     dd      ?               ;font offset
dseg    ends
;
cseg    segment para public 'CODE'
        assume  cs:cseg, ds:dseg
start   proc    near
        mov     ax,dseg
        mov     ds,ax
;
; Select 640x480 graphics mode.
;
        mov     ax,012h
        int     10h
;
; Set the screen to all blue, using the readability of VGA registers
; to preserve reserved bits.
;
        mov     dx,GC_INDEX
        mov     al,GC_SET_RESET
        out     dx,al
        inc     dx
        in      al,dx
        and     al,0f0h
        or      al,1                    ;blue plane only set, others reset
        out     dx,al
        dec     dx
        mov     al,GC_ENABLE_SET_RESET
        out     dx,al
        inc     dx
        in      al,dx
        and     al,0f0h
        or      al,0fh                  ;enable set/reset for all planes
        out     dx,al
        mov     dx,VGA_VIDEO_SEGMENT
        mov     es,dx                   ;point to display memory
        mov     di,0
        mov     cx,8000h                ;fill all 32k words
        mov     ax,0ffffh               ;because of set/reset, the value
                                        ; written actually doesn't matter
        rep stosw                       ;fill with blue
;
; Set driver to use the 8x8 font.
;
        mov     ah,11h          ;VGA BIOS character generator function,
        mov     al,30h          ; return info subfunction
```

```
        mov     bh,3            ;get 8x8 font pointer
        int     10h
        call    SelectFont
;
; Print the test string, cycling through colors.
;
        mov     si,offset TestString
        mov     bx,TEST_TEXT_ROW
        mov     cx,TEST_TEXT_COL
        mov     ah,0                ;start with color 0
StringOutLoop:
        lodsb
        and     al,al
        jz      StringOutDone
        push    ax                  ;preserve color
        call    DrawChar
        pop     ax                  ;restore color
        inc     ah                  ;next color
        and     ah,0fh              ;colors range from 0 to 15
        add     cx,TEST_TEXT_WIDTH
        jmp     StringOutLoop
StringOutDone:
;
; Wait for a key, then set to text mode & end.
;
        mov     ah,1
        int     21h             ;wait for a key
        mov     ax,3
        int     10h             ;restore text mode
;
; Exit to DOS.
;
        mov     ah,4ch
        int     21h
Start   endp
;
; Subroutine to draw a text character in a linear graphics mode
;  (0Dh, 0Eh, 0Fh, 010h, 012h). Background around the pixels that
;  make up the character is preserved.
; Font used should be pointed to by FontPointer.
;
; Input:
;  AL = character to draw
;  AH = color to draw character in (0-15)
;  BX = row to draw text character at
;  CX = column to draw text character at
;
;  Forces ALU function to "move".
;  Forces write mode 3.
;
DrawChar        proc    near
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    bp
        push    ds
        push    ax      ;preserve character to draw in AL
```

```
;
; Set up set/reset to produce character color, using the readability
; of VGA register to preserve the setting of reserved bits 7-4.
;
        mov     dx,GC_INDEX
        mov     al,GC_SET_RESET
        out     dx,al
        inc     dx
        in      al,dx
        and     al,0f0h
        and     ah,0fh
        or      al,ah
        out     dx,al
;
; Select write mode 3, using the readability of VGA registers
; to leave bits other than the write mode bits unchanged.
;
        mov     dx,GC_INDEX
        mov     al,GC_MODE
        out     dx,al
        inc     dx
        in      al,dx
        or      al,3
        out     dx,al
;
; Set DS:SI to point to font and ES to point to display memory.
;
        lds     si,[FontPointer]        ;point to font
        mov     dx,VGA_VIDEO_SEGMENT
        mov     es,dx                   ;point to display memory
;
; Calculate screen address of byte character starts in.
;
        pop     ax              ;get back character to draw in AL

        push    ds              ;point to BIOS data segment
        sub     dx,dx
        mov     ds,dx
        xchg    ax,bx
        mov     di,ds:[SCREEN_WIDTH_IN_BYTES]   ;retrieve BIOS
                                                ; screen width
        pop     ds
        mul     di              ;calculate offset of start of row
        push    di              ;set aside screen width
        mov     di,cx           ;set aside the column
        and     cl,0111b        ;keep only the column in-byte address
        shr     di,1
        shr     di,1
        shr     di,1            ;divide column by 8 to make a byte address
        add     di,ax           ;and point to byte
;
; Calculate font address of character.
;
        sub     bh,bh
        shl     bx,1            ;assumes 8 bytes per character; use
        shl     bx,1            ; a multiply otherwise
        shl     bx,1            ;offset in font of character
        add     si,bx           ;offset in font segment of character
;
; Set up the GC rotation. In write mode 3, this is the rotation
; of CPU data before it is ANDed with the Bit Mask register to
```

```
; form the bit mask. Force the ALU function to "move". Uses the
; readability of VGA registers to leave reserved bits unchanged.
;
        mov     dx,GC_INDEX
        mov     al,GC_ROTATE
        out     dx,al
        inc     dx
        in      al,dx
        and     al,0e0h
        or      al,cl
        out     dx,al
;
; Set up BH as bit mask for left half, BL as rotation for right half.
;
        mov     bx,0ffffh
        shr     bh,cl
        neg     cl
        add     cl,8
        shl     bl,cl
;
; Draw the character, left half first, then right half in the
; succeeding byte, using the data rotation to position the character
; across the byte boundary and then using write mode 3 to combine the
; character data with the bit mask to allow the set/reset value (the
; character color) through only for the proper portion (where the
; font bits for the character are 1) of the character for each byte.
; Wherever the font bits for the character are 0, the background
; color is preserved.
; Does not check for case where character is byte-aligned and
; no rotation and only one write is required.
;
        mov     bp,FONT_CHARACTER_SIZE
        mov     dx,GC_INDEX
        pop     cx                  ;get back screen width
        dec     cx
        dec     cx                  ; -2 because do two bytes for each char
CharacterLoop:
;
; Set the bit mask for the left half of the character.
;
        mov     al,GC_BIT_MASK
        mov     ah,bh
        out     dx,ax
;
; Get the next character byte & write it to display memory.
; (Left half of character.)
;
        mov     al,[si]             ;get character byte
        mov     ah,es:[di]          ;load latches
        stosb                       ;write character byte
;
; Set the bit mask for the right half of the character.
;
        mov     al,GC_BIT_MASK
        mov     ah,bl
        out     dx,ax
;
; Get the character byte again & write it to display memory.
; (Right half of character.)
;
```

```
        lodsb                       ;get character byte
        mov     ah,es:[di]          ;load latches
        stosb                       ;write character byte
;
; Point to next line of character in display memory.
;
        add     di,cx
;
        dec     bp
        jnz     CharacterLoop
;
        pop     ds
        pop     bp
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
DrawChar        endp
;
; Set the pointer to the font to draw from to ES:BP.
;
SelectFont      proc    near
        mov     word ptr [FontPointer],bp       ;save pointer
        mov     word ptr [FontPointer+2],es
        ret
SelectFont      endp
;
cseg    ends
        end     start
```

The key to understanding Listing 26.1 is understanding the effect of ANDing the rotated CPU data with the contents of the Bit Mask register. The CPU data is the pattern for the character to be drawn, with bits equal to 1 indicating where character pixels are to appear. The Data Rotate register is set to rotate the CPU data to pixel-align it, since without rotation characters could only be drawn on byte boundaries.

*As I pointed out in Chapter 25, the CPU is perfectly capable of rotating the data itself, and it's often the case that that's more efficient. The problem with using the Data Rotate register is that the **OUT** that sets that register is time-consuming, especially for proportional text, which requires a different rotation for each character. Also, if the code performs full-byte accesses to display memory—that is, if it combines pieces of two adjacent characters into one byte—whenever possible for efficiency, the CPU generally has to do extra work to prepare the data so the VGA's rotator can handle it.*

At the same time that the Data Rotate register is set, the Bit Mask register is set to allow the CPU to modify only that portion of the display memory byte accessed that the pixel-aligned character falls in, so that other characters and/or graphics data won't be wiped out. The result of ANDing the rotated CPU data byte with the contents of the Bit Mask register is a bit mask that allows only the bits equal to 1 in the original character pattern (rotated and masked to provide pixel alignment) to be modified by the CPU; all other bits come straight from the latches. The latches should have previously been loaded from the target address, so the effect of the ultimate synthesized bit mask value is to allow the CPU to modify only those pixels in display memory that correspond to the 1 bits in that part of the pixel-aligned character that falls in the currently addressed byte. The color of the pixels set by the CPU is determined by the contents of the Set/Reset register.

Whew. It sounds complex, but given an understanding of what the data rotator, set/reset, and the bit mask do, it's not that bad. One good way to make sense of it is to refer to the original text-drawing program in Listing 25.1 back in Chapter 25, and then see how Listing 26.1 differs from that program.

It's worth noting that the results generated by Listing 26.1 could have been accomplished without write mode 3. Write mode 0 could have been used instead, but at a significant performance cost. Instead of letting write mode 3 rotate the CPU data and AND it with the contents of the Bit Mask register, the CPU could simply have rotated the CPU data directly and ANDed it with the value destined for the Bit Mask register and then set the Bit Mask register to the resulting value. Additionally, enable set/reset could have been forced on for all planes, emulating what write mode 3 does to provide pixel colors.

The write mode 3 approach used in Listing 26.1 can be efficiently extended to drawing large blocks of text. For example, suppose that we were to draw a line of 8-pixel-wide bit-mapped text 40 characters long. We could then set up the bit mask and data rotation as appropriate for the left portion of each bit-aligned character (the portion of each character to the left of the byte boundary) and then draw the left portions only of all 40 characters in write mode 3. Then the bit mask could be set up for the right portion of each character, and the right portions of all 40 characters could be drawn. The VGA's fast rotator would be used to do all rotation, and the only **OUT**s required would be those required to set the bit mask and data rotation. This technique could well outperform single-character bit-mapped text drivers such as the one in Listing 26.1 by a significant margin. Listing 26.2 illustrates one implementation of such an approach. Incidentally, note the use of the 8×14 ROM font in Listing 26.2, rather than the 8×8 ROM font used in Listing 26.1. There is also an 8×16 font stored in ROM, along with the tables used to alter the 8×14 and 8×16 ROM fonts into 9×14 and 9×16 fonts.

LISTING 26.2 L26-2.ASM

```
; Program to illustrate high-speed text-drawing operation of
;  write mode 3 of the VGA.
;  Draws a string of 8x14 characters at arbitrary locations
;  without disturbing the background, using VGA's 8x14 ROM font.
;  Designed for use with modes 0Dh, 0Eh, 0Fh, 10h, and 12h.
; Runs only on VGAs (in Models 50 & up and IBM Display Adapter
;  and 100% compatibles).
; Assembled with MASM
; By Michael Abrash
;
```

```
stack   segment para stack 'STACK'
        db      512 dup(?)
stack   ends
;
VGA_VIDEO_SEGMENT       equ     0a000h          ;VGA display memory segment
SCREEN_WIDTH_IN_BYTES   equ     044ah           ;offset of BIOS variable
FONT_CHARACTER_SIZE     equ     14              ;# bytes in each font char
;
; VGA register equates.
;
SC_INDEX                equ     3c4h            ;SC index register
SC_MAP_MASK             equ     2               ;SC map mask register index
GC_INDEX                equ     3ceh            ;GC index register
GC_SET_RESET            equ     0               ;GC set/reset register index
GC_ENABLE_SET_RESET     equ     1               ;GC enable set/reset register index
GC_ROTATE               equ     3               ;GC data rotate/logical function
                                                ; register index
GC_MODE                 equ     5               ;GC Mode register
GC_BIT_MASK             equ     8               ;GC bit mask register index
;
dseg    segment para common 'DATA'
TEST_TEXT_ROW           equ     69              ;row to display test text at
TEST_TEXT_COL           equ     17              ;column to display test text at
TEST_TEXT_COLOR         equ     0fh             ;high intensity white
TestString      label   byte
        db      'Hello, world!',0               ;test string to print.
FontPointer     dd      ?                       ;font offset
dseg    ends
;
cseg    segment para public 'CODE'
        assume  cs:cseg, ds:dseg
start   proc    near
        mov     ax,dseg
        mov     ds,ax
;
; Select 640x480 graphics mode.
;
        mov     ax,012h
        int     10h
;
; Set the screen to all blue, using the readability of VGA registers
; to preserve reserved bits.
;
        mov     dx,GC_INDEX
        mov     al,GC_SET_RESET
        out     dx,al
        inc     dx
        in      al,dx
        and     al,0f0h
        or      al,1                    ;blue plane only set, others reset
        out     dx,al
        dec     dx
        mov     al,GC_ENABLE_SET_RESET
        out     dx,al
        inc     dx
        in      al,dx
        and     al,0f0h
        or      al,0fh                  ;enable set/reset for all planes
        out     dx,al
        mov     dx,VGA_VIDEO_SEGMENT
```

```
        mov     es,dx               ;point to display memory
        mov     di,0
        mov     cx,8000h            ;fill all 32k words
        mov     ax,0ffffh           ;because of set/reset, the value
                                    ; written actually doesn't matter
        rep stosw                   ;fill with blue
;
; Set driver to use the 8x14 font.
;
        mov     ah,11h          ;VGA BIOS character generator function,
        mov     al,30h          ; return info subfunction
        mov     bh,2            ;get 8x14 font pointer
        int     10h
        call    SelectFont
;
; Print the test string.
;
        mov     si,offset TestString
        mov     bx,TEST_TEXT_ROW
        mov     cx,TEST_TEXT_COL
        mov     ah,TEST_TEXT_COLOR
        call    DrawString
;
; Wait for a key, then set to text mode & end.
;
        mov     ah,1
        int     21h             ;wait for a key
        mov     ax,3
        int     10h             ;restore text mode
;
; Exit to DOS.
;
        mov     ah,4ch
        int     21h
Start   endp
;
; Subroutine to draw a text string left-to-right in a linear
;  graphics mode (0Dh, 0Eh, 0Fh, 010h, 012h) with 8-dot-wide
;  characters. Background around the pixels that make up the
;  characters is preserved.
; Font used should be pointed to by FontPointer.
;
; Input:
;  AH = color to draw string in
;  BX = row to draw string on
;  CX = column to start string at
;  DS:SI = string to draw
;
;  Forces ALU function to "move".
;  Forces write mode 3.
;
DrawString      proc    near
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    bp
        push    ds
```

```
;
; Set up set/reset to produce character color, using the readability
; of VGA register to preserve the setting of reserved bits 7-4.
;
        mov     dx,GC_INDEX
        mov     al,GC_SET_RESET
        out     dx,al
        inc     dx
        in      al,dx
        and     al,0f0h
        and     ah,0fh
        or      al,ah
        out     dx,al
;
; Select write mode 3, using the readability of VGA registers
; to leave bits other than the write mode bits unchanged.
;
        mov     dx,GC_INDEX
        mov     al,GC_MODE
        out     dx,al
        inc     dx
        in      al,dx
        or      al,3
        out     dx,al
        mov     dx,VGA_VIDEO_SEGMENT
        mov     es,dx                   ;point to display memory
;
; Calculate screen address of byte character starts in.
;
        push    ds                      ;point to BIOS data segment
        sub     dx,dx
        mov     ds,dx
        mov     di,ds:[SCREEN_WIDTH_IN_BYTES]   ;retrieve BIOS
                                                ; screen width
        pop     ds
        mov     ax,bx           ;row
        mul     di              ;calculate offset of start of row
        push    di              ;set aside screen width
        mov     di,cx           ;set aside the column
        and     cl,0111b        ;keep only the column in-byte address
        shr     di,1
        shr     di,1
        shr     di,1            ;divide column by 8 to make a byte address
        add     di,ax           ;and point to byte
;
; Set up the GC rotation. In write mode 3, this is the rotation
; of CPU data before it is ANDed with the Bit Mask register to
; form the bit mask. Force the ALU function to "move". Uses the
; readability of VGA registers to leave reserved bits unchanged.
;
        mov     dx,GC_INDEX
        mov     al,GC_ROTATE
        out     dx,al
        inc     dx
        in      al,dx
        and     al,0e0h
        or      al,cl
        out     dx,al
;
; Set up BH as bit mask for left half, BL as rotation for right half.
;
```

```
        mov     bx,0ffffh
        shr     bh,cl
        neg     cl
        add     cl,8
        shl     bl,cl
;
; Draw all characters, left portion first, then right portion in the
; succeeding byte, using the data rotation to position the character
; across the byte boundary and then using write mode 3 to combine the
; character data with the bit mask to allow the set/reset value (the
; character color) through only for the proper portion (where the
; font bits for the character are 1) of the character for each byte.
; Wherever the font bits for the character are 0, the background
; color is preserved.
; Does not check for case where character is byte-aligned and
; no rotation and only one write is required.
;
; Draw the left portion of each character in the string.
;
        pop     cx              ;get back screen width
        push    si
        push    di
        push    bx
;
; Set the bit mask for the left half of the character.
;
        mov     dx,GC_INDEX
        mov     al,GC_BIT_MASK
        mov     ah,bh
        out     dx,ax
LeftHalfLoop:
        lodsb
        and     al,al
        jz      LeftHalfLoopDone
        call    CharacterUp
        inc     di              ;point to next character location
        jmp     LeftHalfLoop
LeftHalfLoopDone:
        pop     bx
        pop     di
        pop     si
;
; Draw the right portion of each character in the string.
;
        inc     di              ;right portion of each character is across
                                ; byte boundary
;
; Set the bit mask for the right half of the character.
;
        mov     dx,GC_INDEX
        mov     al,GC_BIT_MASK
        mov     ah,bl
        out     dx,ax
RightHalfLoop:
        lodsb
        and     al,al
        jz      RightHalfLoopDone
        call    CharacterUp
        inc     di              ;point to next character location
        jmp     RightHalfLoop
```

```
RightHalfLoopDone:
;
        pop     ds
        pop     bp
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
DrawString      endp
;
; Draw a character.
;
; Input:
;  AL = character
;  CX = screen width
;  ES:DI = address to draw character at
;
CharacterUp     proc    near
        push    cx
        push    si
        push    di
        push    ds
;
; Set DS:SI to point to font and ES to point to display memory.
;
        lds     si,[FontPointer]        ;point to font
;
; Calculate font address of character.
;
        mov     bl,14           ;14 bytes per character
        mul     bl
        add     si,ax           ;offset in font segment of character

        mov     bp,FONT_CHARACTER_SIZE
        dec     cx              ; -1 because one byte per char
CharacterLoop:
        lodsb                   ;get character byte
        mov     ah,es:[di]      ;load latches
        stosb                   ;write character byte
;
; Point to next line of character in display memory.
;
        add     di,cx
;
        dec     bp
        jnz     CharacterLoop
;
        pop     ds
        pop     di
        pop     si
        pop     cx
        ret
CharacterUp     endp
;
; Set the pointer to the font to draw from to ES:BP.
;
SelectFont      proc    near
        mov     word ptr [FontPointer],bp       ;save pointer
```

```
        mov     word ptr [FontPointer+2],es
        ret
SelectFont      endp
;
cseg    ends
        end     start
```

In this chapter, I've tried to give you a feel for how write mode 3 works and what it might be used for, rather than providing polished, optimized, plug-it-in-and-go code. Like the rest of the VGA's write path, write mode 3 is a resource that can be used in a remarkable variety of ways, and I don't want to lock you into thinking of it as useful in just one context. Instead, you should take the time to thoroughly understand what write mode 3 does, and then, when you do VGA programming, think about how write mode 3 can best be applied to the task at hand. Because I focused on illustrating the operation of write mode 3, neither listing in this chapter is the fastest way to accomplish the desired result. For example, Listing 26.2 could be made nearly twice as fast by simply having the CPU rotate, mask, and join the bytes from adjacent characters, then draw the combined bytes to display memory in a single operation.

Similarly, Listing 26.1 is designed to illustrate write mode 3 and its interaction with the rest of the VGA as a contrast to Listing 25.1 in Chapter 25, rather than for maximum speed, and it could be made considerably more efficient. If we were going for performance, we'd have the CPU not only rotate the bytes into position, but also do the masking by ANDing in software. Even more significantly, we would have the CPU combine adjacent characters into complete, rotated bytes whenever possible, so that only one drawing operation would be required per byte of display memory modified. By doing this, we would eliminate all per-character **OUT**s, and would minimize display memory accesses, approximately doubling text-drawing speed.

As a final note, consider that non-transparent text could also be accelerated with write mode 3. The latches could be filled with the background (text box) color, set/reset could be set to the foreground (text) color, and write mode 3 could then be used to turn monochrome text bytes written by the CPU into characters on the screen with just one write per byte. There are complications, such as drawing partial bytes, and rotating the bytes to align the characters, which we'll revisit later on in Chapter 55, while we're working through the details of the X-Sharp library. Nonetheless, the performance benefit of this approach can be a speedup of as much as four times—all thanks to the decidedly quirky but surprisingly powerful and flexible write mode 3.

A Note on Preserving Register Bits

If you take a quick look, you'll see that the code in Listing 26.1 uses the readable register feature of the VGA to preserve reserved bits and bits other than those being modified. Older adapters such as the CGA and EGA had few readable registers, so it was necessary to set all bits in a register whenever that register was modified. Happily, all

VGA registers are readable, which makes it possible to change only those bits of immediate interest, and, in general, I highly recommend doing exactly that, since IBM (or clone manufacturers) may well someday use some of those reserved bits or change the meanings of some of the bits that are currently in use.