LISTING 28.2 L28-2.ASM

```
; Program to illustrate use of read mode 1 (color compare mode)
; to detect collisions in display memory. Draws a yellow line on a
; blue background, then draws a perpendicular green line until the
; yellow line is reached.
;
; By Michael Abrash
;
stack segment     word stack `STACK'
      db    512 dup (?)
stack ends
;
VGA_SEGMENT       EQU    0a000h
SCREEN_WIDTH            EQU   80    ;in bytes
GC_INDEX          EQU    3ceh        ;Graphics Controller Index register
SET_RESET         EQU    0           ;Set/Reset register index in GC
ENABLE_SET_RESET  EQU    1           ;Enable Set/Reset register index in GC
COLOR_COMPARE     EQU    2           ;Color Compare register index in GC
GRAPHICS_MODE     EQU    5           ;Graphics Mode register index in GC
BIT_MASK          EQU    8           ;Bit Mask register index in GC
;
code              segment   word `CODE'
                  assume    cs:code
Start             proc      near
                  cld
;
; Select graphics mode 10h.
;
     mov    ax,10h
     int    10h
;
; Fill the screen with blue.
;
     mov    al,1                  ;blue is color 1
     call   SelectSetResetColor   ;set to draw in blue
     mov    ax,VGA_SEGMENT
     move   s,ax
     sub    di,di
     mov    cx,7000h
     rep    stosb                 ;the value written actually doesn't
; matter, since set/reset is providing
; the data written to display memory
;
; Draw a vertical yellow line.
;
     mov    al,14                 ;yellow is color 14
     call   SelectSetResetColor   ;set to draw in yellow
     mov    dx,GC_INDEX
     mov    al,BIT_MASK
     out    dx,al                 ;point GC Index to Bit Mask
     inc    dx                    ;point to GC Data
     mov    al,10h
     out    dx,al                 ;set Bit Mask to 10h
     mov    di,40                 ;start in the middle of the top line
     mov    cx,350                ;do full height of screen
VLineLoop:
     mov    al,es:[di]            ;load the latches
     stosb                        ;write next pixel of yellow line (set/reset
```

```
; provides the data written to display
; memory, and AL is actually ignored)
     add    di,SCREEN_WIDTH-1     ;point to the next scan line
loopVLineLoop
;
; Select write mode 0 and read mode 1.
;
     mov    dx,GC_INDEX
     mov    al,GRAPHICS_MODE
     out    dx,al                 ;point GC Index to Graphics Mode register
     inc    dx                    ;point to GC Data
     mov    al,00001000b          ;bit 3=1 is read mode 1, bits 1 & 0=00
                                  ; is write mode 0
     out    dx,al                 ;set Graphics Mode to read mode 1,
                                  ; write mode 0
;
; Draw a horizontal green line, one pixel at a time, from left
; to right until color compare reports a yellow pixel is encountered.
;
; Draw in green.
;
     mov    al,2                  ;green is color 2
     call    SelectSetResetColor  ;set to draw in green
;
; Set color compare to look for yellow.
;
     mov    dx,GC_INDEX
     mov    al,COLOR_COMPARE
     out    dx,al                 ;point GC Index to Color Compare register
     inc    dx                    ;point to GC Data
     mov    al,14                 ;we're looking for yellow, color 14
     out    dx,al                 ;set color compare to look for yellow
     dec    dx                    ;point to GC Index
;
; Set up for quick access to Bit Mask register.
;
     mov    al,BIT_MASK
     out    dx,al                 ;point GC Index to Bit Mask register
     inc    dx                    ;point to GC Data
;
; Set initial pixel mask and display memory offset.
;
     mov    al,80h                ;initial pixel mask
     mov    di,100*SCREEN_WIDTH
                                  ;start at left edge of scan line 100
HLineLoop:
     mov    ah,es:[di]            ;do a read mode 1 (color compare) read.
                                  ; This also loads the latches.
     and    ah,al                 ;is the pixel of current interest yellow?
     jnz    WaitKeyAndDone        ;yes-we've reached the yellow line, so we're
                                  ; done
     out    dx,al                 ;set the Bit Mask register so that we
                                  ; modify only the pixel of interest
     mov    es:[di],al            ;draw the pixel. The value written is
                                  ; irrelevant, since set/reset is providing
                                  ; the data written to display memory
     ror    al,1                  ;shift pixel mask to the next pixel
     adc    di,0                  ;advance the display memory offset if
                                  ; the pixel mask wrapped
;
; Slow things down a bit for visibility (adjust as needed).
;
     mov    cx,0
DelayLoop:
     loop   DelayLoop
```

```
     jmp    HLineLoop
;
; Wait for a key to be pressed to end, then return to text mode and
; return to DOS.
;
WaitKeyAndDone:
WaitKeyLoop:
     mov    ah,1
     int    16h
     jz    WaitKeyLoop
     sub    ah,ah
     int    16h                  ;clear the key
     mov    ax,3
     int    10h                  ;return to text mode
     mov    ah,4ch
     int    21h                  ;done
Startendp
;
; Enables set/reset for all planes, and sets the set/reset color
; to AL.
;
SelectSetResetColorprocnear
     mov    dx,GC_INDEX
     push   ax                   ;preserve color
     mov    al,SET_RESET
     out    dx,al                ;point GC Index to Set/Reset register
     inc    dx                   ;point to GC Data
     pop    ax                   ;get back color
     out    dx,al                ;set Set/Reset register to selected color
     dec    dx                   ;point to GC Index
     mov    al,ENABLE_SET_RESET
     out    dx,al                ;point GC Index to Enable Set/Reset register
     inc    dx                   ;point to GC Data
     mov    al,0fh
     out    dx,al                ;enable set/reset for all planes
     ret
SelectSetResetColorendp
code ends
end  Start
```

When all Planes “Don’t Care”

Still and all, there aren’t all that many uses for basic color compare operations. There is, however, a genuinely odd application of read mode 1 that’s worth knowing about; but in order to understand that, we must first look at the “don’t care” aspect of color compare operation.

As described earlier, during read mode 1 reads the color stored in the Color Compare register is compared to each of the 8 pixels at a given address in VGA memory. But—and it’s a big but—any plane for which the corresponding bit in the Color Don’t Care register is a 0 is always considered a color compare match, regardless of the values of that plane’s bits in the pixels and in the Color Compare register.

Let’s look at this another way. A given pixel is controlled by four bits, one in each plane. Normally (when the Color Don’t Care register is 0FH), the color in the Color Compare register is compared to the four bits of each pixel; bit 0 of the Color Compare register is compared to the plane 0 bit of each pixel, bit 1 of the Color Compare register is compared to the plane 1 bit of each pixel, and so on. That is, when the lower four bits of the Color Don’t Care register are all set to 1, then all four bits of a given pixel must match the Color Compare register in order for a read mode 1 read to return a 1 for that pixel to the CPU.

However, if any bit of the Color Don't Care register is 0, then the corresponding bit of each pixel is unconditionally considered to match the corresponding bit of the Color Compare register. You might think of the Color Don't Care register as selecting exactly which planes should matter in a given read mode 1 read. At the extreme, if all bits of the Color Don't Care register are 0, then read mode 1 reads will always return 0FFH, since all planes are considered to match all bits of all pixels.

Now, we're all prone to using tools the "right" way—that is, in the way in which they were intended to be used. By that token, the Color Don't Care register is clearly intended to mask one or more planes out of a color comparison, and as such, has limited use. However, the Color Don't Care register becomes far more interesting in exactly the "extreme" case described above, where all planes become "don't care" planes.

Why? Well, as I've said, when all planes are "don't care" planes, read mode 1 reads always return 0FFH. Now, when you AND any value with 0FFH, the value remains unchanged, and that can be awfully handy when you're using the bit mask to modify selected pixels in VGA memory. Recall that you must always read VGA memory to load the latches before writing to VGA memory when you're using the bit mask. Traditionally, two separate instructions—a read followed by a write—are used to perform this task. The code in Listing 28.2 uses this approach. Suppose, however, that you've set the VGA to read mode 1, with the Color Don't Care register set to 0 (meaning all reads of VGA memory will return 0FFH). Under these circumstances, you can use a single **AND** instruction to both read and write VGA memory, since ANDing any value with 0FFH leaves that value unchanged.