8. Bit-Mapped Graphics

Background

The previous lab introduced you to the basics of text-based graphics on the PC. This lab will introduce you to bit-mapped graphics on the PC. The standard video adapters found on PCs support many different screen resolutions with differing numbers of colors per dot (or pixel) on the screen. Graphics on the PC have gone through a long evolution that started with low resolutions (less than 100,000 pixels), limited colors and primitive graphic capabilities to the high resolutions (greater than 1 million pixels), true color and sophisticated 3-D graphic cards available today. Because of the need for upward compatibility, even the most advanced graphic cards still support the more primitive graphic modes.

Because of the complexity of computer graphics, this lab will look at the some of the earlier graphic modes as a way of introducing a few of the basic concepts of bit-mapped graphics. This lab will concentrate on the graphics modes known as "VGA", which is one step below the current graphics modes that used by Windows (Super-VGA).

Objectives:

Understand:

A. Basic graphic modes on a standard video adapter for the PC

B. Operation of a color palette for controlling pixel colors

Pre-Lab

Read section 5.7 in the Irvine textbook about BIOS-level Video control for graphics modes. Answer the following questions:

1. What is the difference between a text mode and graphics mode for a standard video adapter on the PC?
2. What is a pixel? What does the value of a pixel represent?

Lab

A. Basic Graphic Modes on the PC (VGA)

The program below (*vidtst.asm*) illustrates how to set different graphic modes and the writing of pixels to the screen. **WARNING:** On some laptops only the 640x480 mode will work.

```
        .model small
        .586
        .stack 100h
        .data

        MAXROW  equ   480
        MAXCOL  equ   640
        MAXPIX  equ   16
        VIDMOD  equ   12h  ; 640x480 16 color

;       MAXROW  equ   200
;       MAXCOL  equ   320
;       MAXPIX  equ   16
;       VIDMOD  equ   0Dh  ;320x200 16 color

;       MAXROW  equ   200
;       MAXCOL  equ   320
;       MAXPIX  equ   4
;       VIDMOD  equ   04h  ;320x200 4 color

;       MAXROW  equ   200
;       MAXCOL  equ   320
;       MAXPIX  equ   256
;       VIDMOD  equ   13h  ;320x200 256 color

        BANDSZE equ   MAXROW/MAXPIX
        row    dw  0
        col    dw  0
        bandcnt db  0
        pixel   db  0
        vmode  db  0     ;; current video mode
        vpage  db  0     ;; current video page

        .code


        main          proc
                      mov    ax,@data
                      mov    ds,ax
        ;; read current video mode and save
                      mov    ah,0fh
                      int    10h
                      mov    vmode,al
                      mov    vpage,bh
        ;; set new video mode
                      mov    ah,0
                      mov    al,VIDMOD
                      int    10h
```

```
	lp1:
			call	wpixel
			inc	col
			mov	ax,col
			cmp	ax,MAXCOL
			jne	lp1
	;start new row
			xor	ax,ax
			mov	col,ax ;zero column value
			inc	 bandcnt
			mov	al,bandcnt
			cmp	al,bandsze
			jb	nextrow
			inc	pixel	  ;inc to next color value
			xor	al,al
			mov	bandcnt,al
nextrow:
			inc	row
			mov	ax,row
			cmp	ax,MAXROW
			jne	lp1

doexit:
			mov	ah,1
			int	21h	; get a key
			mov	ah,0
	;; restore old video mode
			mov	bh,vpage
			mov	al,vmode
			int	10h

			Mov	ax, 4c00h	 ;exit
			Int	21h
	Main		endp

	wpixel		proc
			mov	ah,0ch
			mov	al, pixel
			mov	bh,0
			mov	cx,col
			mov	dx, row
			int	10h
			ret
	 wpixel			endp
end main
```

This program sets a graphic mode, and then tests the mode by displaying all possible colors in horizontal bands that progress down the screen. The program then waits for any key to be pressed before restoring the original text mode and exiting the program.

How does this program work?

1. The BIOS function 10h, AH= 0 is used to set the video mode. The VIDMOD equate is used to specify the video modes. The MAXROW, MAXCOL, MAXPIX equates specify the maximum number of rows, columns and colors available in that video mode. The program first saves the current video mode via the BIOS function 10h, AH=0FH, and these sets the video mode as specified by VIDMOD.
2. To display all possible colors as horizontal bands of colors, the program uses the BANDSZE equate to compute how many rows will have the same color via the computation MAXROW/MAXPIX. The pixel value is stored in location 'pixel' and starts out at a value of 0 (color 0). The memory locations *row* and *col* are used to keep track of where the current pixel should be written, and the procedure *wpixel* uses the BIOS function 10H, AH=0CH to write a pixel at that location.

3. The program loops, writing one row of pixels at a time (each pixel is written individually using the *wpixel* procedure with the *col* value being incremented each time). When BANDSZE number of rows has been written, the pixel value is incremented which advances it to the next color value.
4. The program stops once the maximum row position has been reached and waits for a key press.

Lab Question 1: Assemble this program and execute it (you will need to link in the *irvine.lib* library).

A. Test the program with each set of VIDMOD, MAXCOL, MAXROW, MAXPIX values by uncommenting each set in turn and re-assembling, re-executing the program.

B. Modify the program such that the colors are displayed in vertical stripes instead of horizontal stripes. Include a commented listing (at least one comment for every two x86 instructions) of your program in your lab report.

B. Color Representation

A pixel color is composed of three components: Red (R), Green (G), and Blue (B). An RGB monitor has an electron gun for each of these colors; the three beams converge on a pixel to produce a color. The electron beam starts in the upper left corner and is swept left to right for each row and moved down the screen to paint a complete screen (when the beam reaches the right edge, it is turned off and moved quickly back to the left edge, and down a bit for the next row - this is called *horizontal retrace*). When the beam reaches the bottom right corner, it is turned off, moved quickly back to the upper left corner (*vertical retrace*), and the process is repeated. The refresh rate of the monitor is usually between 60 Hz and 80Hz and this defines the number of screens drawn per second. Different intensities for each beam (R,G,B) produce different colors. An analog RGB monitor has an analog voltage input for each beam -- the voltage level on the input determines the intensity of the beam. The video adapter is the device that provides these voltage levels. Within the video card chipset is a device known as a Video DAC (Digital-to-Analog Converter) that converts a digital value that represents a color to the analog voltage needed by the RGB monitor.

The term '24-bit' color means that 8-bits are used for each of the R, G, B color components of a pixel color so each pixel requires 3 bytes of memory. For RGB values, a value of '0' represents the minimum beam intensity while a value of 255 is the maximum beam intensity. An RGB value of 255,0,0 is the color bright RED, a value of 0,0,0 is BLACK, and 255,255,255 is WHITE. If the screen resolution is 1280 x 1024 with 24-bit color, then the number of bytes of memory needed for one screen would be 1280 x 1024 x 3 = 3,932,160 bytes (a little under 4 MB).

A value of 4 MB for video memory does not seem like a lot of memory these days, but it used to be significant. To reduce the amount of memory required to represent pixel colors, a color palette was used. A color palette is a lookup table stored on the video DAC. Each entry in the table contains three values representing R, G and B. A pixel 'color' specifies a table entry, and the RGB values stored for the table entry specifies the color. If the palette had 16 entries, then 16 colors could be represented on the screen. Which colors these values represent depends upon what RGB values are loaded into the color palette for each entry. For 16 colors, only 4 bits is needed for each pixel (1/2 byte) so the memory needed for a 1280 x 1024 screen would be 1280 x 1024 x 0.5 = 655,360 bytes (a little over 1/2 MB).

All of the VGA video modes in the previous example (*vidtst.asm*) use a color palette for pixel colors. A default color palette is loaded by BIOS for each video mode. The program on the next page (*paltst.asm)* illustrates how to change the color palette for a video mode.

WARNING: The *paltst.asm* program is set by default for 320x200 mode. The 320x200 mode might not work on some laptops, try using the 640x480 mode if you have problems with the 320x200 mode.

The tables in the *paltst.asm* program give colors as 8-bit values (0 –255). However, the subroutine that sends these colors to the video card only uses the upper 6 bits because some older video card only had 6-bits of color resolution in their Video DACs.

```
        .model small
        .586
        .stack 100h
        .data
        MAXROW  equ   200
        MAXCOL  equ   320
        MAXPIX  equ   16
        VIDMOD  equ   0Dh  ;320x200 16 color

;       MAXROW  equ   480
;       MAXCOL  equ   640
;       MAXPIX  equ   16
;       VIDMOD  equ   12h  ; 640x480 16 color

        BANDSZE equ   MAXROW/MAXPIX
        row    dw  0
        col    dw  0
        bandcnt db  0
        pixel   db  0
        vmode  db  0     ;; current video mode
        vpage  db  0     ;; current video page

        ;; define a color palette
        ;; 16 entries  each has a R, G, B value
         ;; leave border color alone

        ;; note that color zero is used for border
        ;; on many video cards in VGA mode, only 6 bits of precision so palette
        ;; routine ignores lower 2 bits.
         ;; in VGA mode!!!!
          palette1   db  255,0,0          ;color 0
              db  0,255,0          ;color 1
              db  0,0,255          ;color 2
              db  0,0,0   ;color 3
              db  255,255,255   ;color 4
              db  255,255,0   ;color 5
              db  0,255,255   ;color 6
              db  255,0,255   ;color 7
              db  255,0,0   ;color 8
              db  0,0,255   ;color 9
              db  0,255,0   ;color 10
              db  255,127,0   ;color 11
              db  127,255,0   ;color 12
              db  0,255,127  ;color 13
              db  127,0,127   ;color 14
              db  0,127,127   ;color 15


        ;; note that color zero is used for border
         palette2   db  127,0,127
              db  0,127,127
              db  127,127,0
              db  190,190,190
              db  63,63,63
              db  0,0,127
              db  0,127,0
              db  127,0,0
              db  255,0,255
              db  0,255,255
              db  255,255,0
              db  255,255,255
              db  0,0,0
              db  0,0,255
              db  0,255,0
              db  255,0,0
        .code


        main          proc
                      mov    ax,@data
```

```
                mov     ds,ax
;; read current video mode and save
                mov     ah,0fh
                int     10h
                mov     vmode,al
                mov     vpage,bh
;; set video mode -- changing the video mode changes palette!!
                mov     ah,0
                mov     al,VIDMOD
                int     10h
;; write a screen
                call scrntst
;; change the pallette
mainlp:                 mov     si,offset palette1
                call sndpal
                mov     ah,7    ;get a key, no echo
                int     21h    ; get a key
                cmp     al,20h
                je      main_ex
                mov     si,offset palette2
                call    sndpal
                mov     ah,7    ;get a key, no echo
                int     21h    ; get a key
                cmp     al,20h
                jne     mainlp

main_ex:
;; restore old video mode
                mov     bh,vpage
                mov     al,vmode
                mov     ah,0
                int     10h

                Mov     ax, 4c00h     ;exit
                Int     21h
Main            endp


;; test the current mode by writing a band of colors
scrntst                 proc
                xor     ax,ax
                mov     row,ax
                mov     col,ax
                mov     pixel,al
                mov     bandcnt,al

lp1:
                call    wpixel
                inc     col
                mov     ax,col
                cmp     ax,MAXCOL
                jne     lp1
;start new row
                xor     ax,ax
                mov     col,ax ;zero column value
                inc      bandcnt
                mov     al,bandcnt
                cmp     al,bandsze
                jb      nextrow
                inc     pixel     ;inc to next color value
                xor     al,al
                mov     bandcnt,al
nextrow:
                inc     row
                mov     ax,row
                cmp     ax,MAXROW
                jne     lp1
doexit:
                mov     ah,7    ;get a key, no echo
                int     21h    ; get a key
                ret
scrntst                 endp
```

```
;; send a new palette passed in SI
;; if 640x480 mode, have to send in different sequence
sndpal          proc
                call    retrace
                xor      al,al       ; start at color 0
                mov     dx,3c8h          ; port number for Video card
                out     dx,al

                mov     al,vidmod
                cmp     al,12H  ;; 640?
                je      sndskip

                mov     di,si       ;save si
                mov     cx,8
                call     palsub
                mov     si,di
                mov     cx,8
                call     palsub
;; because of the way the palette registers are arranged
;; send first 8 colors twice
                mov     cx,8
                mov     di,si
                call     palsub
                mov     si,di
                mov     cx,8
                call    palsub
                ret
sndskip:
                mov     cx,16
sndskplp:
                push    cx
                push    si
                mov     cx,16
                call    palsub
                pop     si
                pop     cx
                loop    sndskplp

                ret

 sndpal                 endp

palsub          proc
 sndlp1:
                mov     dx,3c9h          ; port number for Video card
                mov     al,[si]     ;get RED value
                shr     al,2
                out      dx,al
                mov     al,[si+1]   ;get GREEN value
                shr     al,2
                 out      dx,al
                mov     al,[si+2]   ; get BLUE value
                shr     al,2
                  out      dx,al
                add     si,3        ;point to next color
                loop     sndlp1       ;send all colors
                ret
 palsub                 endp

; wait for vertical retrace
retrace           proc

                push     dx
                push     ax

                mov      dx,03dah                ;; wait for end of retrace
lpwaitstart:
                in       al,dx
                and      al,08h
                jnz      lpwaitstart
```

```
                mov     dx,03dah                ;; wait for start of retrace
   lpwaitend:     in      al,dx
                and     al,08h
                jz      lpwaitend

                pop     ax
                pop     dx
                       ret
    retrace     endp

   wpixel       proc
                mov    ah,0ch
                mov    al, pixel
                mov    bh,0
                mov    cx,col
                mov    dx, row
                int    10h
                ret
    wpixel             endp
end main
```

This program uses the code from the first program to display the colors of the 320x200x16 video mode as horizontal bands. The program then switches the color palette on each key press between two new color palettes; the space bar will exit the program.

How does this program work?

1. This program is simply a modification of the previous 'vidtst.asm' program. Two 16-color palette tables called 'palette1' and 'palette2' have been added to the data segment. Each table has 16 colors, with each color represented by three bytes (R, G, B). The colors specified in these tables were arbitrarily chosen. Color '0' (the first entry) is used for the screen border color -- normally this color is black (0,0,0).
2. The *sndpal* procedure is used to load a new palette into the video DAC. The *sndpal* procedure expects the starting address of the palette to be passed in register SI. The *sndpal* procedure operates by first writing a value of '0' to the video DAC register at port 3C8h. This tells the DAC that we want to start loading palette colors starting at color '0'. The *sndpal* procedure then writes the table values sequentially to port 3C9h - every three bytes written to this port defines a color entry and the order of the bytes is R,G,B. Normally, the DAC expects to receive 256 color values -- for 16 colors the register mapping is such that each block of 8 colors needs to be repeated twice. The *sndpal* procedure uses the subroutine palsub to take care of this. This version of the *sndpal* procedure only works with a 16-color palette. Note that the first thing that *sndpal* does is call a procedure called *retrace*. The *retrace* procedure waits for the start of the vertical retrace by reading a status register from the video card. *Vertical retrace* is when the CRT beam is moved from the lower right corner of the screen back to the upper left corner in order to begin drawing another screen. During this time, the beam is turned off. In this way, the changing of the palette color occurs when the beam is turned off and is finished before another screen is drawn.
3. After the initial color bands are displayed, the main program waits for a key press. On each key press, a new color palette is loaded (alternates between *palette1* and *palette2*). The space character causes the program to exit.

Lab Question 2: Assemble this program and execute it. After the initial screen, use any character other than the 'space' character to alternate between the color palettes. Note that color 0 is used as the border color and that both of the new palettes use a non-black color for color 0. For the 'default' palette, create a table in your report in which you use English to describe the displayed colors -- also create columns for the R,G,B values of these colors. Modify the original program so that Palette2 matches the default color palette (DO NOT spend forever trying to get an exact color match - approximate the best that you can and then move on). Include the RGB values that you determined for the 'default' palette in your lab report. Note that the TOP band (color 0) of the default palette is BLACK and merges with the top border. (There is a BIOS function that allows you read the current color palette - if you want to try to use this function to determine the colors of the default palette instead of using trial/error RGB matching, then go right ahead -- Use the HelpPC program linked to the lab WWW page and look under BIOS Video Services to determine what BIOS function to use).

C. A Programming Task - Color Animation

2-D Computer animation involves copying blocks of video memory representing groups of pixels to different places in the video buffer. A technique known as *color animation* can be used to give the appearance of movement but it only involves changing the color palette (this is much less CPU intensive than copying blocks of memory). Write a program that does the following:

A. Define your own 16 color palette -- the only restriction on colors is that color 0 must be black. You can repeat colors if you desire.

B. Use the program you wrote for Lab Question #1 (vertical color stripes) as a starting point. Modify it to use your new color palette as the palette. Use the *sndpal* procedure from the previous example to load your color palette into the video DAC.

C. Achieve color animation in your program by writing a loop that rotates the colors 1 through 15 on each loop iteration. To *rotate* the colors means that after the first iteration, color 1 should be copied to color 2, color 2 to color 3, etc and color 15 to color 1. After the palette is rotated, load the new palette into the video DAC via the *sndpal* procedure. Use the *mywait* procedure from the previous lab to add a delay between each rotation of the color palette. You should be able to achieve an animation effect in which the colors appear to march across the screen from left to right. To achieve more interesting effects, change the *mywait* procedure from Lab7 such that the CX value passed to *mywait* represents ten's of microseconds (.01 milliseconds) instead of milliseconds. Doing this will allow you change the palette very quickly.

D. Start out by having the colors move with a 0.1 second delay between palette rotations. Monitor the keyboard – if a 'w' is pressed, decrease the time between rotations (speed it up). If a 's' is pressed, then increase the time between rotations (slow it down). You can decide on how much to increase/decrease the delay time for each key press

E. If the space bar is pressed, then exit the program.

Lab Question 3: Include the assembled listing of this program in your lab report and make sure that you have at least one comment for every two x86 instruction lines. After your program is working, try executing your program with the call to the *retrace* procedure commented out of the *sndpal* procedure. What visual differences do you see? Why does this happen?

Lab Report

A. Describing What You Learned

Include the answers to all "**Lab Questions**" in your report.

B. Applying What You Learned

Demonstrate the programs you wrote for Lab Questions 1 and 3 to the TA.