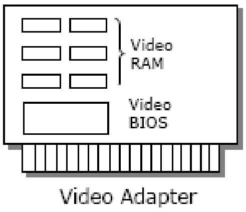
Video Graphic Adaptor (VGA) Programming

Compiled by Ong Wee Hong

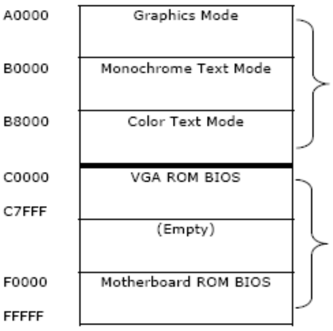
The Hardware: VGA Card



Video Adapter

To get a display a component called video adapter is added to the motherboard. Hardware engineers sometimes call this video adapter as video card. On the video card there is a group of video RAM chips. The video card may have upto 8MB on board, but most of them are used by circuits on the card and cannot be directly accessed by processor. In the basic VGA mode (e.g., DOS mode, Windows safe mode), the processor can directly access upto 128KB (i.e., A0000h to BFFFFh) of video RAM. Usually all video cards also have onboard video BIOS normally addressed at C0000h TO C7FFFh.

Memory map



Not all the memory is used for display purpose because we have so many video modes that support different resolutions. The video modes are usually set by the programs that are stored in video ROM BIOS area. Note that it is ROM, which the content cannot be modified. Whereas the video RAM can be written and is used to effect the display on screen. It is necessary to know which display mode and which memory area is used. Far pointers can be used to write into video RAM. Since VGA and SVGA adapters are used almost everywhere, the memory map for VGA and SVGA is shown beside. Other adapters' memory map will be slightly different and their use should be referred its documentation.

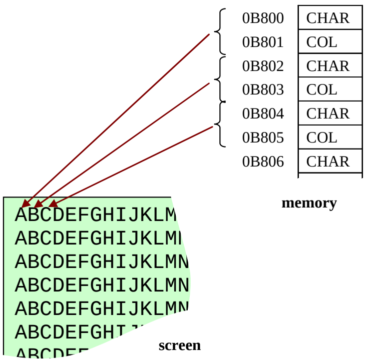
Programming the video RAM

VGA supports each of the modes supported by its predecessors. VGA is backward compatible. So it is enough to know about programming VGA RAM.

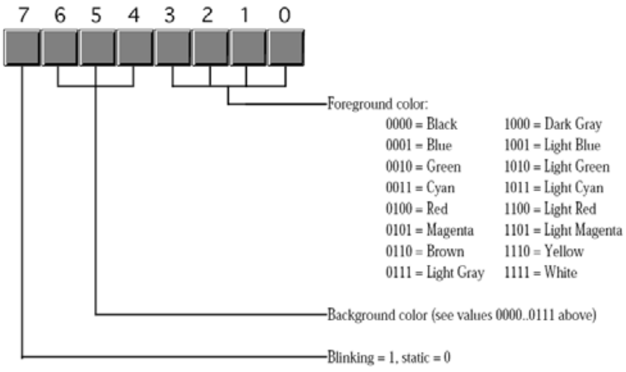
Color Text Mode: Mode 03h

This is the default display mode in DOS. This mode uses the video RAMs addressed at B8000 to BFFFFh. In normal color text mode 3h (80x25x16 mode), the address space is divided into 4 video pages of 4KB each (page 0, page 1, page 2 & page 3). At the same time we can see the characters in any one of the pages. The screen's resolution is 80x25 (i.e. 80 columns x 25 rows). It supports 16 colors at a time. To display a single character, two bytes are being used namely character byte and attribute byte. The character byte contains the ASCII value of the character. The attribute byte is organized as:

Bitfields for character's display attribute				
7	654	3	210	
x				foreground blink or (alternate) background bright
	xxx			background color
		x		foreground bright or (alternate) alternate character set
			xxx	foreground color



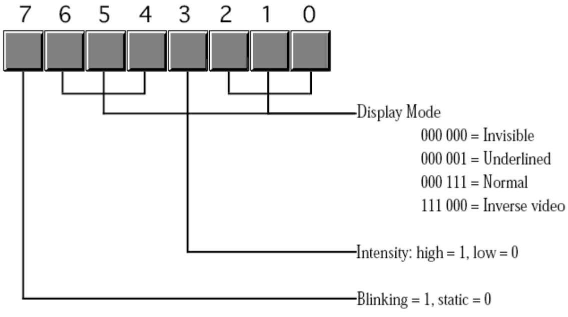
In Mode 03h, we can think of the screen as a grid of 80 x 25 cells, each cell contains a character. In the video RAM, two pieces of information are stored for each character: the character (its ASCII code) and its attribute (as defined in table above). 2 consecutive bytes (starts at 0B800h) represent a character on screen. This makes a total of 80x25x2 = 4000 bytes in the buffer.



Color Display Adapter Attribute Byte Format

Monochrome Text Mode: Mode 07h

Monochrome text mode is similar to color text mode. But this mode uses B000h as a segment address, it displays the character as normal or even reverse video and or underlined for the given attribute colors.

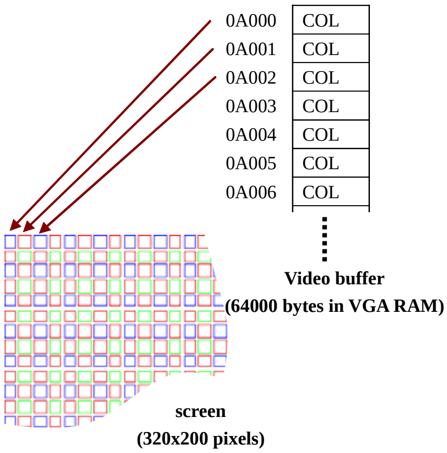


Monochrome Display Adapter Attribute Byte Format

Graphics Mode: Mode 13h

Mode 13h is so widely used for graphics applications in DOS because it is very easy to use. Mode 13h is the 320x200x256 graphics mode, and is fast and very convenient from a programmer's perspective. The screen is constantly being redrawn by the video card. To affect what the card draws, it is necessary to write to the video (screen) buffer. The video buffer begins at address A000:0000 and ends at address A000:F9FF.

In Mode 13h, the screen is 320 by 200, or 320 pixels across and 200 pixels down. In each pixel, there's a possibility of 256 colors, which can be fit into one byte. Thus, 320*200*1 = 64000 bytes, about the size of one segment. This means the buffer is 64000 bytes long and that each pixel in mode 13h is represented by one byte. Think of the screen as an array of colors. The first row takes up addresses A000:0000 to A000:013F (decimal 319), the second row takes up addresses A000:0140 to A000:027F (decimal 639), and so on.



It is easy to set up mode 13h and the video buffer in assembly language:

```
mov ax,0013h    ;Int 10 - Video BIOS Services
int 10h         ;ah = 00 - Set Video Mode, al = 13 - Mode 13h (320x200x256)

mov ax,0A000h   ;point segment register es to A000h
mov es,ax       ;we can now access the video buffer as offsets from register es
```

At the end of your program, it is advisable to restore the text mode. Here's how:

```
mov ax,0003h    ;Int 10 - Video BIOS Services
int 10h         ;ah = 00 - Set Video Mode, al = 03 - Mode 03h (80x25x16 text)
```

Accessing a specific pixel in the buffer is also very easy:

```
; bx = x coordinate
; ax = y coordinate
; pixel (x,y) number = 320*y + x
mul 320         ;multiply y coord by 320 to get row
add ax,bx       ;add this with the x coord to get offset
mov cx,es:[ax]  ;now pixel x,y can be accessed as es:[ax]
```

That was easy but that multiplication is slow and should be avoided. That's easy to do too, simply by using bit shifting instead of multiplication. Shifting a number to the left is the same as multiplying by 2. We want to multiply by 320, which is not a multiple of 2, but 320 = 256 + 64, and 256 and 64 are both even multiples of 2. So a faster way to access a pixel is:

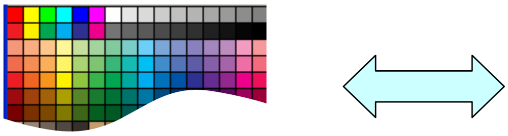
```
; bx = x coordinate
; ax = y coordinate
; pixel (x,y) location = 320*y + x = (256+64)*y + x = (y*256 + y*64) + x = (y*2^8 + y*2^6)
+ x
mov cx,ax       ;copy ax to cx, to save it temporatily
shl cx,8        ;shift left by 8, which is the same as multiplying by 2^8 = 256
shl ax,6        ;now shift left by 6, which is the same as multiplying by 2^6 = 64
add ax,cx       ;now add those two together, which is effectively multiplying by 320
add ax,bx       ;finally add the x coord to this value
mov cx,es:[ax]  ;now pixel x,y can be accessed as es:[ax]
```

Well, the code is a little bit longer and looks more complicated, but it's much faster.

To plot colors, we use a color look-up table. This look-up table is a 768 (3x256) array. Each index of the table is really the offset index*3. The 3 bytes at each index hold the corresponding values (0-63) of the red, green, and blue components. This gives a total of 262144 total possible colors. However, since the table is only 256 elements big, only 256 different colors are possible at a given time. This look-up table is called the color palette.

Note:

Each color can have upto 64 levels, i.e. red has 64 levels, green has 64 levels and blue has 64 levels. This gives a total combination of 64×64×64 = 262144 possible colors.



color palette

3 bytes

Index:			
0	R_0	G_0	B_0
1	R_1	G_1	B_1
2	R_2	G_2	B_2
3	R_3	G_3	B_3
:	:	:	:
:	:	:	:
:	:	:	:
:	:	:	:
254	R_{254}	G_{254}	B_{254}
255	R_{255}	G_{255}	B_{255}

Look-up Table
(Color Registers of the VGA Card)

Changing the color palette is accomplished through the use of the I/O ports of the VGA card:

Port 03C7h is the Palette Register Read port

Port 03C8h is the Palette Register Write port

Port 03C9h is the Palette Data port

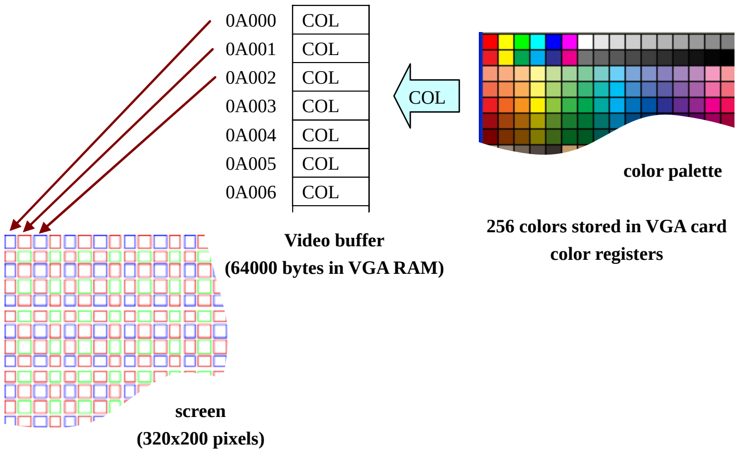
Here is how to change the color palette:

```
; ax = palette index
; bl = red component (0-63)
; cl = green component (0-63)
; dl = blue component (0-63)
mov dx,03C8h    ;03c8h = Palette Register Write port
out dx,ax       ;choose index within palette

mov dx,03C9h    ;03c9h = Palette Data port
                ;note the PORT 03C9h will automatically be saving the 1st value to R, 2nd to
                ;G, 3rd to B
mov al,bl       ;set red value
out dx,al
mov al,cl       ;set green value
out dx,al
mov al,dl       ;set blue value
out dx,al
                ;if we continue to send to port 03c9h, the port will automatically increment
                ;to next color index
```

Thats all there is to it. Reading the color palette is similar:

```
; ax = palette index
; bl = red component (0-63)
; cl = green component (0-63)
; dl = blue component (0-63)
mov dx,03C7h    ;03c7h = Palette Register Read port
out dx,ax       ;choose index within palette

mov dx,03C9h    ;03c9h = Palette Data port
in  al,dx
mov bl,al       ;get red value
in  al,dx
mov cl,al       ;get green value
in  al,dx
mov dl,al       ;get blue value
```

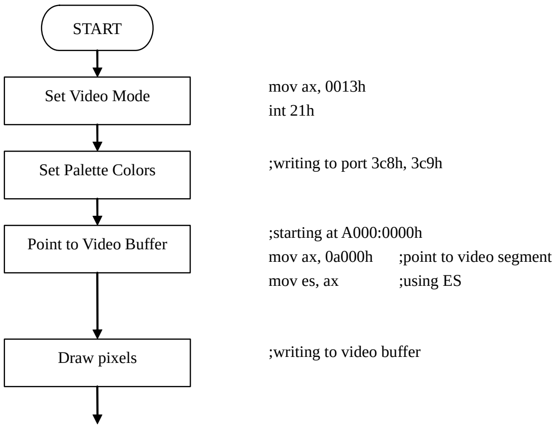
Now all we need to know is how to plot a pixel of a certain color at a certain location. This is achieved by writing the appropriate color index (from the palette) into the corresponding location (of the pixel) in the video buffer (VGA RAM).



It’s very easy, given what we already know:

```
; bx = x coordinate
; ax = y coordinate
; dx = color (0-255); index as in the palette (look-up table)
; pixel (x,y) location = 320*y + x = (256+64)*y + x = (y*256 + y*64) + x = (y*2^8 + y*2^6)
+ x
mov cx,ax       ;copy ax to cx, to save it temporatily
shl cx,8        ;shift left by 8, which is the same as multiplying by 2^8 = 256
shl ax,6        ;now shift left by 6, which is the same as multiplying by 2^6 = 64
add ax,cx       ;now add those two together, which is effectively multiplying by 320
add ax,bx       ;finally add the x coord to this value
mov es:[ax],dx  ;copy color dx into memory location
```

The following flow chart summarizes the steps involve in VGA programming in Mode 13h.



```
mov ax, 0013h
int 21h

;writing to port 3c8h, 3c9h

;starting at A000:0000h
mov ax, 0a000h     ;point to video segment
mov es, ax         ;using ES

;writing to video buffer
```

In very rare cases, Mode 13h may not be supported by the graphic card. It may be useful to check if Mode 13h is possible by adding the following codes before setting up the video mode to Mode 13h.

```
mov ax, 1a00h   ;AH = 1ah, AL = 00, get video display combination
int 10h         ;return AL = 1ah if valid video function
cmp al, 1ah     ;if AL=1ah,
je start        ;set video mode and start VGA programming
```

References

1. “VGA Programming in Mode 13h” by Lord Lucifer at http://www.codebreakers-journal.com/content/view/117/52
2. “A to Z of C” by K. Joseph Wesley and R. Rajesh Jeba Anbiah at http://guideme.itgo.com/atozofc/
3. “Using VGA Mode 13h for Fast Graphics” by Michael Averbuch
4. “Art of Assembly Language” by Randall Hyde