EXPERIMENT THIRTEEN
ACCESSING VIDEO MEMORY

INTRODUCTION

This experiment concentrates on using the registers within the VGA controller and BIOS INT 10H functions to access video memory using the mode 12H graphics mode. Mode 12H operation of the VGA card displays a 640 x 480, 16 color display on the video screen. Other modes are available, but not explained in this experiment. Although this experiment is meant only as a primer, it illustrates techniques that can be used with higher resolution graphics modes not presented in this experiment because of the differences found in graphics display cards. Eventually the higher resolution modes will be standardized through VESA (video electronics standards association) so these higher resolution modes can be presented generically for all video cards.

OBJECTIVES

1. Use the 640 x 480, 16 color graphics display mode to display any color at any dot position on the video display.
2. Use mode 12H to divide the screen into a 53 line by 80 character per line display so blocks of colors can be displayed.
3. Display text on the 640 x 480, 16 color graphics display without changing the background color.

PROCEDURE

The first section of this experiment concentrates on the 640 X 480, 16 color graphics display mode for displaying graphics. This is mode 12H and requires a VGA (video graphics array) video card to function. This should present no problem today, because this is considered the minimum for most personal computer systems.

Understanding the 640 x 480, 16 color display.

In an earlier experiment we dabbled in directly accessing video text memory for mode 3. This is the mode that DOS normally uses for its presentation. In that experiment we learned that the video text memory was located at address B800:0000 through B800:FFFF and that it contained ASCII data and attribute data for display. The 640 X 480, 16 color mode is a little more complex and uses memory location A000:0000 through A000:FFFF to access graphics data. A little arithmetic shows that to display 16 colors with a resolution of 640 x 480 requires more memory than the 64K bytes found at segment A000H. Because a 16 color display requires 4 bits of memory and the resolution
is 640 x 480 (307,200 bits), the memory system requires 1,228,800 bits or 153,600 bytes of memory in this display mode. Only 64K bytes of memory space is available.

To allow access to this amount of memory, video vendors have standardized mode 12H displays so it is accessed in bit planes. A bit plane is a linear section of memory that contains one of the 4 bits required to display 16 colors. Each bit plane requires 307,200 bits of memory, stored in 38,400 bytes of memory. The 64K bytes of memory at segment A000H are enough to only address a single bit plane at a time. The bit plane is addressed at memory locations A000:0000 through A000:95FF. Location A000:0000 represents the upper leftmost 8 dots in a 640 x 480 display and location A000:95FF represents the lower rightmost 8 dots in a 640 x 480 display. There are four planes or banks of memory that overlap this address range to represent the four bits or color for each dot.

Note the color codes, as represented in an earlier experiment, are arranged so the leftmost bit represents bright and the next three bits represent red, blue and green respectively. (See Figure 13-1).

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Black</td>
</tr>
<tr>
<td>0001</td>
<td>Blue</td>
</tr>
<tr>
<td>0010</td>
<td>Green</td>
</tr>
<tr>
<td>0011</td>
<td>Cyan</td>
</tr>
<tr>
<td>0100</td>
<td>Red</td>
</tr>
<tr>
<td>0101</td>
<td>Magenta</td>
</tr>
<tr>
<td>0110</td>
<td>Brown</td>
</tr>
<tr>
<td>0111</td>
<td>White</td>
</tr>
<tr>
<td>1000</td>
<td>Grey</td>
</tr>
<tr>
<td>1001</td>
<td>Bright Blue</td>
</tr>
<tr>
<td>1010</td>
<td>Bright Green</td>
</tr>
<tr>
<td>1011</td>
<td>Bright Cyan</td>
</tr>
<tr>
<td>1100</td>
<td>Bright Red</td>
</tr>
<tr>
<td>1101</td>
<td>Bright Magenta</td>
</tr>
<tr>
<td>1110</td>
<td>Bright Yellow</td>
</tr>
<tr>
<td>1111</td>
<td>Bright White</td>
</tr>
</tbody>
</table>

* This bit set causes color to be bright

Figure 13 –1: The bit pattern and colors available to VGA Mode 12h
Access to video memory in mode 12H is fairly complex and accomplished by the following steps:

1. Read the byte of memory to be changed. This is a read for the video card latches and the data read from memory does not need to be used.

2. Select the bit or bits in this byte that need to be changed through index 8 of the VGA card address register by sending an 8 out to I/O port 3CEH. Next load AL with the bits to be changed (a one-bit represents a bit to be changed) and send this out to I/O port 3CFH, the bit mask register (BMR).

```
MOV DX, 3CEH ;select VGA card address register
MOV AL, 8 ;index of 8
OUT DX, AL ;select index 8
MOV DX, 3CFH ;select bit mask register
MOV AL, 80H ;place mask in AL
OUT DX, AL
```

3. Next set all mask bits to 1111 in the map mask register (MMR) at sequencer offset 2, and write color 0 to the VGA card to set the color to black. The mask bits select the bit planes to be changed. If all are selected and a color of 0 is written, all four bit-planes are cleared to zero.

```
MOV DX, 3C4H ;select VGA card sequencer register
MOV AL, 2 ;index of 2
OUT DX, AL ;select index 2
MOV DX, 3C5H ;address map mask register
MOV AL, OFH ;mask to 1111 binary
OUT DX, AL
```

; next write a 0000 binary to the selected video memory location

4. Send the desired color number to the map mask register and write an FFH to the video memory. This places a logic one in only the selected bit planes to write a new color to a pixel or dot on the screen.

```
MOV AL, 3 ;choose color cyan
OUT DX, AL ;select color
```

;next write a FFH to the selected video memory location

**Displaying a single pixel in mode 12H.**

Displaying information in this mode (12H) is not a simple task as can be gathered from the steps required. Example 13-1 lists a program that displays a single bright green dot in the uppermost left display position in mode 12H. It also selects mode 12H with the INT 10H instruction and, after waiting for a key, resets the mode to mode 3 at the end. Even
though the dot is bright green, it is very small, so look closely in the upper left corner for a green dot when you execute this program.

Example 13-1

```assembly
CODE SEGMENT 'code' ; indicate start of code segment
ASSUME CS:CODE ; identify CODE as CS
INCLUDE MACS.INC

MAIN PROC FAR ; start main procedure
  MOV AX,12H ; select mode 12H
  INT 10H
  MOV AX,0A000H ; address video memory segment
  MOV DS,AX
  MOV SI,0 ; address first byte
  MOV AL,[SI] ; read first byte
  MOV DX,3CEH ; select VGA address register
  MOV AL,8 ; index 8
  OUT DX,AL
  MOV DX,3CFH ; select bit mask register
  MOV AL,80H ; select leftmost bit
  OUT DX,AL ; select bit
  MOV DX,3C4H ; address sequence register
  MOV AL,2 ; index 2
  OUT DX,AL
  MOV DX,3C5H ; address map mask register
  MOV AL,0FH ; select all planes
  OUT DX,AL
  MOV BYTE PTR [SI1,0] ; write zero to set ; color to black
  MOV AL,0AH ; set color to bright green
  OUT DX,AL ; set color
  MOV BYTE PTR [SI],0FFH ; write color
  MOV AH,7 ; pause for key so we can see dot!
  INT 21H
  MOV AX,3 ; set mode back to 3
  INT 10H
_EXIT ; exit to DOS

MAIN ENDP

CODE ENDS

END MAIN
```

It is obvious, from the length of the program in Example 13-1, that a macro or procedure is needed to display a single dot on the video display in mode 12H. Such a macro, called _DOT is listed in Example 13-2. (Add this to your macro file MACS.INC). This macro contains three parameters: COLOR, X, and Y. The X parameter is the position across the screen (0-639), the Y is the vertical position (0-479), and the COLOR (0-15) is the color of the dot.
Example 13-2

```assembly
_DOT MACRO COLOR,X,Y
; dot plot macro (mode 12H)
PUSH DS
;;;; save registers
PUSH CX
PUSH DX
PUSH SI
MOV AL,COLOR
;;;; get color
PUSH AX
MOV AX,OAOOOH
;;;; address video segment
MOV DS,AX
MOV AX,Y
MOV SI,640
MUL SI
;;;; find Y position
ADD AX,X
;;;; find XY position
ADC DX,0
PUSH AX
AND AL,7
;;;; get position count for bit mask
MOV SI,AL
POP AX
MOV SI,8
;;;; divide by 8 to find byte
DIV SI
MOV SI,AX
;;;; save address of dot in SI
MOV AL,[SI]
;;;; read video location
MOV AH,80H
;;;; leftmost pixel set
SHR AH,CL
MOV DX,3CEH
;;;; select VGA address register
MOV AX,OF02H
OUT DX,AX
INC DX
MOV BYTE PTR [SI],0
;;;; write zero to set
;;;; color to black
POP AX
;;;; get color
OUT DX,AL
;;;; set color
MOV BYTE PTR [SI],OFFH
;;;; write color
POP SI
;;;; restore registers
POP DX
POP CX
POP DS
ENDM
```

STEP 1: Store the _DOT macro in the macro file MACS .INC . Write a program using DOT that places a series of dots in the following positions on the screen: bright blue across the first 3 lines of the display, bright red across the last three lines of the display, and one yellow dot at X = 320 and Y = 220. Enter, assemble, and execute this program.
Displaying blocks of color in mode 12H.

Because the _DOT macro plots a single dot at a time, the process of painting the entire screen or large sections of the screen can take considerable time. To speed up the process of displaying large areas of color information, the screen can be divided into blocks to speed the process. Suppose that the 640 pixels across the screen are divided into 80 positions, each representing a byte in video memory. Also suppose that the 480 lines are divided into 53 lines of 9 scanning lines each with 3 unused lines left over at the bottom of the screen (this matches a 53 line text display). For large areas, the screen would become a 80 x 53 display. Note that other values can divide the screen into other size blocks.

The macro (_BLOCK) listed in Example 13-3 divides the screen into an 80 x 53 display of blocked areas. As with the _DOT macro, _BLOCK uses three parameters: COLOR (0-15), X (0-79), and Y (0-52).

Example 13-3

```
_BLOCK MACRO COLOR,X,Y ;display block
LOCAL B1
PUSH DS ;;save registers
PUSH CX
PUSH DX
PUSH SI
MOV AL,COLOR ;;get color
PUSH AX
MOV AX,OAOOOH ;;address video memory segment
MOV DS,AX
MOV AX,Y ;;find raster line
MOV SI,80*9
MUL SI
ADD AX,X ;;address offset byte
MOV SI,AX
MOV AX,OFF08H
MOV DX,3CEH
OUT DX,AX ;;enable all bits
MOV DX,3C4H
MOV CX,9 ;;load count
B1:

MOV AL, [SI]
MOV AX,OF02H
OUT DX,AX
INC DX
MOV BYTE PTR [SI],0
POP AX
PUSH AX
OUT DX,AL
MOV BYTE PTR [SI],OFFH
DEC DX
ADD SI,80
LOOP B1
POP AX ;;restore registers
POP SI
```
STEP 2: Using the _BLOCK macro developed in Example 13-3, write a program that displays a bright magenta screen. Note the speed at which this changes the screen color when executed.

STEP 3: Develop a program that uses _BLOCK to display a white bar across the top of the screen two block-sized lines deep and a white bar along the bottom of the screen two block-sized lines deep. In the center of the screen, display cyan in all other block-sized locations.

Displaying text in mode 12H.

The video display output from this program is similar to many screens in commercially available software packages. To display text on this screen requires the use of DOS function calls, that always display white text on a black background, which is probably not desirable, or some other technique. Fortunately, the video BIOS ROM contains several different character sets that can be obtained and used to display data on the graphics display. If we use the 80 character, 53 line display generated by the BLOCK macro, we select the 8 x 8 character set from the BIOS ROM. The 8 x 8 character set is a byte wide, as is our blocked display, and eight scanning lines high. Our blocked display is 9 scanning lines high, which leaves a blank scanning line between displayed lines with the 8 x 8 character set.

To obtain the location of the 8 x 8 character set from the ROM requires a call to a BIOS INT 10H function call. The starting address of the character set is returned in ES:BP by using the following sequence of instructions, which use INT 10H, function number 1130H to locate the character set in the ROM:

```
MOV AX, 1130H ;read ROM information
MOV BH, 3 ;return address of 8 x 8 set
INT 10H
```

The starting address of other character sets are returned as indicated in Table 13-2.

Table 13-2. Starting addresses of ROM character sets.

<table>
<thead>
<tr>
<th>BH</th>
<th>Character set</th>
</tr>
</thead>
<tbody>
<tr>
<td>02H</td>
<td>8 (wide) x 14 (high)</td>
</tr>
<tr>
<td>03H</td>
<td>8 x 8 (standard)</td>
</tr>
<tr>
<td>04H</td>
<td>8 x 8 extended characters (ASCII 80H-FFH)</td>
</tr>
<tr>
<td>05H</td>
<td>9 x 14 alternate</td>
</tr>
<tr>
<td>06H</td>
<td>8 x 16</td>
</tr>
<tr>
<td>07H</td>
<td>8 x 16 alternate</td>
</tr>
</tbody>
</table>
Once the address of the character set is known, the bit patterns represented by the character set are used to display any character on the graphics display using any color. Example 13-4 shows a macro that displays any standard ASCII character from AL on the 640 x 480, 16 color display using any character color. This macro assumes that the display is organized as an 80 character per line by 53 line display. Note that this macro does not change the color of the background and it does not display a cursor. The parameters passed to the macro are the character color (0-15), the line number, and the position of the character on the line.

Example 13-4

```
_CHAR MACRO COLOR,LINE,ROW ;display character
LOCAL Cl
PUSH DS
    ;;save registers
PUSH ES
PUSH SI
PUSH BX
PUSH CX
PUSH DX
MOV BL,COLOR ;;save color
PUSH BX
    ;;save row
MOV BX,ROW ;;save line
PUSH BX
PUSH AX ;;save ASCII code
MOV AX, 1130H ;;get 8 x 8 set
MOV BH,3
INT 10H
POP AX ;;get ASCII code
XOR AH,AH
SHL AX,1 ;;multiply by 8
SHL AX, 1
SHL AX, 1
ADD BP,AX ;;index character in ROM
MOV AX,OAO00H ;;address video segment
MOV DS,AX
POP AX ;;get line number
MOV SI,80*9
MUL SI
MOV SI,AX ;;generate raster address
POP AX ;;get row number
ADD SI,AX
MOV CX,8 ;;set count to 8 rows
MOV DX,3CEH ;;address bit mask register
MOV AL,8 ;;load index 8
MOV AH,ES:[BP] ;;get character row
INC BP ;;point to next row
OUT DX,AX
MOV DX,3C4H ;;address map mask register
MOV AX,OF02H
OUT DX,AX ;;select all planes
```
INCDX
MOVAL,[SI];;;;readdata
MOVBYTEPTR[SI],0;;;;writeblack
POPAX;;;;getcolor
PUSHAX
OUTDX,AL;;;;writecolor
MOVBYTEPTR[SI],OFFH
ADDSI,80;;;;addressnextrasterrow
LOOPC1;;;;repeat8times
POPAAX;;;;restoreregisters
POPDX
POPCX
POPBX
POPBFPPOPSSIPOPES
POPDS
ENDM

Notice how the bit mask register is used to select only the logic one bits in the ASCII coded characters. This allows only the character itself to change the color of the display at only the location of the character. The character appears without changing the background color.

STEP 4: Using the program written in Step 3, add a section that uses _CHAR to display your name at the center of the top line in yellow (color - OEH) and the words Main Menu in black at the center of the second line. It is important to write a procedure that uses the _CHAR macro to display a character string. This will reduce the amount of work required to develop this portion of the program.

Although this is an introduction to developing graphics data in the 640 x 480, 16 color mode, the macros presented here and the programming techniques developed can lead to considerable graphics programs.

QUESTIONS

13-1. How much memory space is used to access the 640 x 480, 16 color video display?

13-2. What is a bit plane and how does it apply to the operation of the 640 x 480, 16 color display?

13-3. How is video mode 12H selected?

13-4. What is the purpose of the bit mask register?

13-5. How is the bit mask register addressed?

13-6. What is the purpose of the map mask register?

13-7. How is the map mask register addressed?
13-8. A color must be first set to _____________ before a new color is written.

13-9. Write a program using the DOT macro that draws a solid red line, two raster lines wide, across the bottom of the video display screen.

13-10. Write a program using the BLOCK macro that draws a cyan area across the top of the screen two character lines high. Also draw a cyan area across the bottom of the screen that is 4 character lines deep. Make the center screen area white.