Lab # 13

Drawing Graphics Using INT 10h
Objective:
Drawing Graphics Using INT 10h.

Introduction:
It's easy to draw graphics points and lines using INT 10h Function 0Ch. Before drawing pixels, you have to put the video adapter into one of the standard graphics modes. Each mode can be set using INT 10h function 0 (set video mode).

<table>
<thead>
<tr>
<th>Mode</th>
<th>Resolution (columns X rows, in pixels)</th>
<th>Number of Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>640 X 200</td>
<td>2</td>
</tr>
<tr>
<td>0Dh</td>
<td>320 X 200</td>
<td>16</td>
</tr>
<tr>
<td>0Eh</td>
<td>640 X 200</td>
<td>16</td>
</tr>
<tr>
<td>0Fh</td>
<td>640 X 350</td>
<td>2</td>
</tr>
<tr>
<td>10h</td>
<td>640 X 350</td>
<td>16</td>
</tr>
<tr>
<td>11h</td>
<td>640 X 480</td>
<td>2</td>
</tr>
<tr>
<td>12h</td>
<td>640 X 480</td>
<td>16</td>
</tr>
<tr>
<td>13h</td>
<td>320 X 200</td>
<td>256</td>
</tr>
<tr>
<td>6Ah</td>
<td>800 X 600</td>
<td>16</td>
</tr>
</tbody>
</table>

INT 10h Pixel-Related Functions:
Slow performance, Easy to program
- 0Ch: Write graphics pixel
- 0Dh: Read graphics pixel

<table>
<thead>
<tr>
<th>INT 10h Function 0Ch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>
| **Receives**         | AH = 0Ch
                        AL = pixel value
                        BH = video page
                        CX = x-coordinate
                        DX = y-coordinate |
| **Returns**          | nothing |
Lab work:

**Exercise1:**

This program draws a straight line, using INT 10h function calls.
Note:
- This program will work well on Windows XP.
- But won’t work on Windows 7 or Vista. (Install DOSBox to see the output)
Memory-Mapped Graphics:

- Binary values are written to video RAM
  - Video adapter must use standard address.
- It has Very fast performance
  - No BIOS or DOS routines to get in the way.
Mode 13h: 320 X 200, 256 Colors

- Mode 13h graphics (320 X 200, 256 colors)
  - Fairly easy to program
  - read and write video adapter via IN and OUT instructions
  - Pixel-mapping scheme (1 byte per pixel)

Exercise 2:
This program draws 10 Pixels on the Screen.

```
.model small
.386
.stack 100h
.data
saveMode BYTE ? ; saved video mode
xVal WORD ? ; x-coordinate
yVal WORD ? ; y-coordinate
.code
Main:
    mov ax, @data
    mov ds, ax
    call SetVideoMode
    call Draw_Some_Pixels
    call RestoreVideoMode
    mov ah, 4ch
    int 21h
SetVideoMode PROC
    ; This procedure saves the current video mode, switches to
    ; a new mode, and points ES to the video segment.
    mov ah, 0Fh ; get current video mode
    int 10h
    mov saveMode, al ; save it
    mov ah, 0 ; set new video mode
    mov al, 13h ; to mode 13h
    int 10h
    push 0A000h ; video segment address
        pop es ; ES = A000h (video segment).
ret
SetVideoMode ENDP
RestoreVideoMode PROC
    ; This procedure waits for a key to be pressed and
    ; restores the video mode to its original value.
    mov ah, 10h ; wait for keystroke
        int 16h
    mov ah, 0 ; reset video mode
    mov al, saveMode ; to saved mode
    int 10h
    ret
RestoreVideoMode ENDP
```
; Draw_Some_Pixels PROC
; Calculate the video buffer offset of the first pixel.
; Specific to mode 13h, which is 320 X 200.
    mov xVal,160 ; middle of screen
    mov yVal,100
    mov ax,320 ; 320 for video mode 13h
    mul yVal ; y-coordinate
    add ax,xVal ; x-coordinate
    ; Place the color index into the video buffer.
    mov cx,10 ; draw 10 pixels
    mov di,ax ; AX contains buffer offset
    ; Draw the 10 pixels now.
    Dpi:
        mov BYTE PTR es:[di],1 ; store color index
        add di,5 ; move 5 pixels to the right
        Loop Dpi
    ret

Draw_Some_Pixels ENDP
END main

Output:

............

6