University of Khartoum
Faculty of Engineering - Department of Electrical and Electronic Engineering.
Microprocessor and Assembly Language

Assembly Language Final Project

Low Level Paint

Prepared by:

1. Fay Majid Ahmed Elhassan 144050
2. Hussien Husham Mansour 144026
3. Israa Mohamed Hamid Hassan 144013
4. Moayad Hassan Mohamed Elamin 144058
5. Mohamed Kamal Ahmed Gomma 144070

Tuesday 17th /April/2018

Objectives:

To design and implement a Mini-Paint software program by changing from text mode to graphics mode and use of interrupts.

What did you want to do?

We wanted to implement a Mini-Paint software program on a bare machine using the mouse as the prime I/O device and its interrupts to visualize different shapes and free-hand drawings on the screen with the aid of x86 Assembly Language & Virtual Box.

The objective of the project is to:

1. Implement a free drawing using the I/O device (mouse)
2. Be able to draw various shapes at click of a button and change color
3. Opportunity to choose between various colors during free drawing

Software Description:

- A bootable bare machine program.
- A virtual hard disk is booted into the virtual machine (VM), from which the program is executed asking the user to choose between two options (1. For Free Drawing 2. For shapes)
- Using a screen that is 320x200 pixel (Video Graphics Array) VGA mode.

- In free drawing you get option to choose between four different colors (Black, Red, Green, Blue)
- In shapes we have:
    - Circle
    - Square
    - Pyramid
    - Rectangle
    - Right-angle Triangle
    - Isosceles Triangle
    - Equilateral Triangle
    - Trapezium
    - Diamond
    - Filled Circle
    - Filled Rectangle
    - Lines

Algorithm Description:

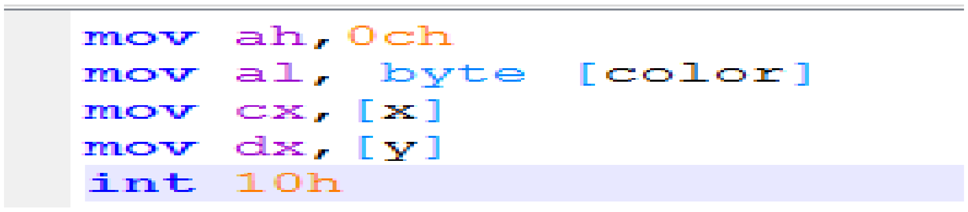
Interacting with pixels is by setting its color in the corresponding Video memory.

We used the Video mode 0x13 which is a standard IBM VGA BIOS mode number for a 256 color 320x200 resolution. It uses a 256-color palette allows access to Video memory as Packed-Pixel Framebuffer.

To be able to draw the shapes we have to move from text mode to VGA mode (Video Graphics Array) using the following syntax:

```
cli
   mov ah,0
   mov al,13h
   int 10h
```

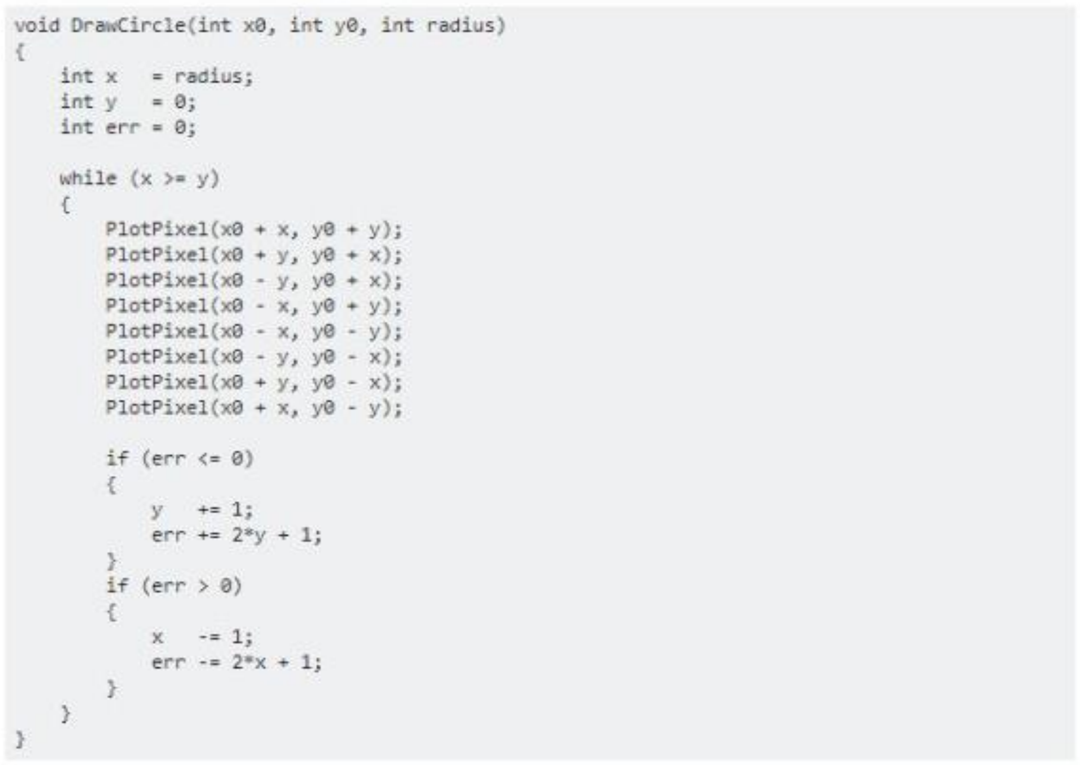
After changing to graphics mode in order to change the pixel color we send the color to the address of the beginning pixel 0x000A000, however using the interrupt (int 10h) we send the appropriate data into the interrupt vector as follow:



```
mov  ah, 0ch
mov  al,  byte  [color]
mov  cx, [x]
mov  dx, [y]
int  10h
```

Shapes:

Using the midpoint algorithm (Bresenham's Algorithm), we have drawn the circle and the square, to construct the other, we used the same concept. Using the center point, (160,100) and the length and width, we calculated the coordinates of the edges of each shape.



```
void DrawCircle(int x0, int y0, int radius)
{
    int x   = radius;
    int y   = 0;
    int err = 0;

    while (x >= y)
    {
        PlotPixel(x0 + x, y0 + y);
        PlotPixel(x0 + y, y0 + x);
        PlotPixel(x0 - y, y0 + x);
        PlotPixel(x0 - x, y0 + y);
        PlotPixel(x0 - x, y0 - y);
        PlotPixel(x0 - y, y0 - x);
        PlotPixel(x0 + y, y0 - x);
        PlotPixel(x0 + x, y0 - y);

        if (err <= 0)
        {
            y   += 1;
            err += 2*y + 1;
        }
        if (err > 0)
        {
            x   -= 1;
            err -= 2*x + 1;
        }
    }
}
```

In some cases, like the right-angle triangle and equilateral triangle, the width and height are equal so we simplified it by calling both of them the radius. After we calculate the coordinates, we draw the lines depending on what the line being drawn is.

It was used in drawing most of the shapes represented in the demo described above.

For drawing the lines (Horizontal, Vertical, Negative slope, Positive Slope) we used the following midpoint algorithms:

```
Horizontal:
Xend=Xbegin+width
Yend=Ybegin
For(i= Xbegin;i<= Xend;i++){
Xbegin ++
ret X
}
```

```
Vertical:
Xend=Xbegin
Yend=Ybegin+height
For(j= Ybegin;j<= Yend;j++){
Y++
Ret Y
}
```

```
Neg Slope:
Xend=Xbegin+width
Yend=Ybegin+height
Neglope[0][0]= Posslope[Xbegin][ Ybegin]
For(i= Xbegin;i<= Xend;i++){
For(j= Ybegin;j<= Yend;j++){
Xbegin++
Ybegin++
Ret Neglope[Xbegin][ Ybegin]
}
}
```

```
Positive Slope:
Xend=Xbegin+width
Yend=Ybegin-height
Posslope[0][0]= Posslope[Xbegin][ Ybegin]
For(i= Xbegin;i<= Xend;i++){
For(j= Yend;j<= Ybegin;j--){
Xbegin++
Ybegin--
Ret Posslope[Xbegin][ Ybegin]
}
}
```

The algorithm used above is known as the midpoint algorithm and we chose it because it works with direct x and y due to its ability to be easily computational and it was straight-forward to implement in x86 language.

Mouse Drive Function:

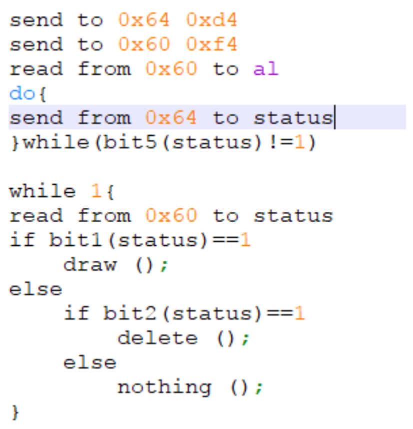
In this part of the project we aimed to define a mouse driver, enable it and use the mouse to draw in a white screen.

The main guideline for our work was the work of Last years project of the same functions. They provided the first step to which we directed our work.

Another guideline was the follow site: https://wiki.osdev.org/Mouse_Input. It gave us a full understanding on the PS2 Mouse its functionality and how we can implement.

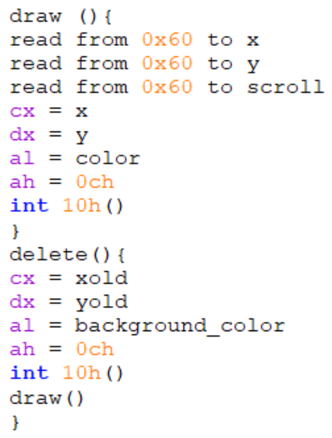
After that the work took the following steps:

1. Turning the screen to VGA mode (using int 10h).
2. Checking if there is a device connected and can receive instructions from the user (checking port 0x64).
3. Preparing the mouse before sending a message (sending 0xd4 to port 0x64).
4. Sending Enabling instruction (sending 0xf4 to port 0x60).
5. Reading the response message to the enable instruction.
6. Beginning the loop:
    A. Checking if the device is a keyboard or a mouse (5th bit in the status read from 0x64).
    B. Reading First input message from the mouse that contains the status (it's read from port 0x60).
    C. Checking the first 2 bits to decide whether we clicked the left click (we jump to the draw function) or the right click (we jump to the delete function) or neither (we do the nothing function).
    D. Reading the X axis displacement from the previous X position and adding it to the previous position to get actual X axis position.
    E. Reading the Y axis displacement from the previous Y position and subtracting it from the previous positon to get the actual Y axis position (the Y movement is reversed).
    F. Using the coordinates we pass them to the draw or delete or nothing part.
    G. We use the int 10h to color the pixel at the positon X, Y passed to CX and DX with color passed to AH.
    H. We read the last scroll information from the mouse.

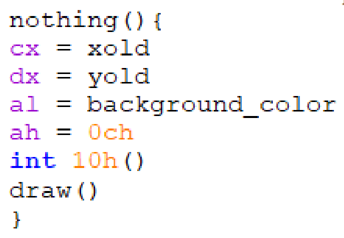


```
send to 0x64 0xd4
send to 0x60 0xf4
read from 0x60 to al
do{
send from 0x64 to status
}while(bit5(status)!=1)

while 1{
read from 0x60 to status
if bit1(status)==1
    draw ();
else
    if bit2(status)==1
        delete ();
    else
        nothing ();
}
```



```
draw (){
read from 0x60 to x
read from 0x60 to y
read from 0x60 to scroll
cx = x
dx = y
al = color
ah = 0ch
int 10h()
}
delete(){
cx = xold
dx = yold
al = background_color
ah = 0ch
int 10h()
draw()
}
```



```
nothing(){
cx = xold
dx = yold
al = background_color
ah = 0ch
int 10h()
draw()
}
```

Above steps are repeated for different mouse movement (delete or draw or do nothing), each having its own specifications that are connected with the background color of the surrounding pixels.

Another functionality that we added was the ability to change the color of the drawing cursor using a left positioned select menu. This was accomplished by dividing the left most 60x200 pixels of the screen and then redevising to 4 40x50 squares with each of them representing a color, and when the cursor is located inside one of the squares the status flag is checked to see if the left click is pressed at then we change the draw color to the color of the square.

What objective was achieved?

- Implement a free drawing using the I/O device (mouse)
- Be able to draw various shapes at click of a button and change color
- Opportunity to choose between various colors during free drawing

If we had more time we would have done the following:

1. A function plotter to draw an inserted function in a specified range of X and corresponding Y.
2. Give the program various points by the mouse and be able to connect them into shapes.
3. A different function for the mouse painter which uses a spray-can like cursor to draw in shapes different than the usual pixel draw.
4. Involving more shapes to our application program and enable the user to change the size of the shapes .

Problems we have faced:

1. The functionality of the mouse itself, in the check read and write it checked more than the needed and it gave back random results of the displacement.
2. We wanted to work with interrupt (int 33h) for the moving of the cursor instead of the ports but we faced difficulty in setting the dimensions of the cx, dx (registers that stores the x and y positions respectively) in order to be changing in loops according to the movement to the cursor.
3. In regard to the shapes the only problem we faced was with the isosceles triangle as in order to it to be drawn we need a phase of 60 degrees between its sides so we had a lot of complications setting this angle.

4. Virtual Machine box wasn't as efficient, we had reinstalled it several times as most times it wasn't able to read our code or accept another virtual hard disk file.

What did you learn?

Through this project we have learned the underlying details of the computer hardware and how to deal with them in order to generate any application program to be used by the user.

We also got a brief knowledge of how to be able to interact with interrupts and we got a better understanding and practice in the idea of virtual machines and how the bootloader works.

This project showed us how the computer deals with the I/O.
It also got us to know how we can generate graphics and how to work in VGA mode (Video Graphics Mode) in means of pixels.
