Title: Name Rotation Program in Graphics Mode (8086 Assembly)

Student: al amin Hossain nayem
Course: Systemic Programming – Laboratory Work
Tool: emu8086

1. Task description

The task was to write a program in 8086 assembly language which displays the student's name on the graphical screen. The program must react to keyboard keys:

- pressing **R**: each letter turns to its right side;
- pressing **L**: each letter turns to its left side;
- pressing **E**: the program finishes and returns to text mode.

The requirement was to use at least three different images of the letters to simulate rotation.

2. Memory model and program type

I used the directive:

.model tiny

org 100h

MODEL TINY was chosen because:

- it produces a **.COM** program, which is exactly what emu8086 expects for simple DOS-style experiments;
- code and data are in a single segment (maximum 64 KB), which is more than enough for this lab;
- this model matches the examples used in the lecture slides.

Using .COM also simplifies segment initialization (only mov ax, cs / mov ds, ax is needed).

3. Graphics mode and interrupts

The program uses **BIOS video interrupt INT 10h** and **keyboard interrupt INT 16h**:

- Graphics mode is set with function **AH = 00h**, **AL = 13h**:
- mov ah, 0
- mov al, 13h ; 320×200, 256 colors

- int 10h
- Pixels are drawn with **INT 10h, AH = 0Ch (Write Graphics Pixel)**.
  Before calling this function, the program supplies:
    - AL – color (I used 15, white),
    - BH – page number (0),
    - CX – X coordinate,
    - DX – Y coordinate.
- Keyboard input is read with **INT 16h, AH = 00h**, which blocks until a key is pressed and returns its ASCII code in AL.

When leaving the program, video mode is restored to standard text mode 3:

```
mov ah, 0
mov al, 03h
int 10h
```

4. Data structures

The program uses three main types of data:

1. **Glyph bitmaps** (letters):
    - Each letter is represented as an 8×8 bitmap stored in 8 bytes.
    - Each byte corresponds to one row; each bit represents one pixel.
    - Example for letter A:
    - GlyphA db 00111100b
    - db 01000010b
    - ...
2. **Letter table** (LetterTable):
    - An array of pointers to the glyphs in the order "A L _ A M I N".
    - This allows one generic procedure DrawName to draw any sequence of glyphs.
3. **RowShift array**:

- RowShift db 0,0,1,1,2,2,3,3
- Contains horizontal shift values for each scanline (row) of a letter.
- Lower rows have larger shift, which gives a simple perspective effect.

In addition, the variable CURRENT_FRAME holds the active “rotation frame”:

- 0 – left tilt,
- 1 – front,
- 2 – right tilt.

5. Algorithm and control flow

The main algorithm can be summarized as:

1. Set graphics mode 13h.
2. Initialize CURRENT_FRAME = 1 (front view).
3. Call DrawName once.
4. Enter an infinite loop:
    - Wait for key (INT 16h).
    - If key is R/r, increase CURRENT_FRAME up to maximum value 2.
    - If key is L/l, decrease CURRENT_FRAME down to minimum value 0.
    - After changing frame, clear the screen and redraw the name.
    - If key is E/e, return to text mode and terminate.

This control flow is implemented by labelled sections PressR, PressL, and ExitProgram.

6. Rotation method (algorithm choice)

For rotation I used a simple **shear-based algorithm** instead of real trigonometric rotation, because:

- Real 2D rotation would require floating-point arithmetic or fixed-point math, which is unnecessary for a small lab task.
- A shear transformation with row-dependent offsets is enough to create a visual illusion of turning left or right.

The algorithm for each pixel is:

1. For every glyph row (row = 0..7), load one byte from the glyph.
2. Copy this byte into a working register and scan its bits from left to right.
3. When a bit is 1, compute base position:
4. X = LetterBaseX + column
5. Y = LetterBaseY + row
6. Read shift = RowShift[row].
7. Depending on the active frame:
    - left frame (CURRENT_FRAME = 0): X = X − shift
    - front frame (CURRENT_FRAME = 1): X = X
    - right frame (CURRENT_FRAME = 2): X = X + shift
8. Call PutPixel(X, Y, color).

This “three-frame + shear” method satisfies the requirement “at least three different letter images” and uses loops, arrays and bit operations, which are exactly the topics of the course.

7. Procedures

To keep the code structured, I used several procedures:

- ClearScreen – clears the whole 320×200 frame buffer by writing zeros directly into segment A000h.
- PutPixel – draws one pixel via INT 10h, AH=0Ch. It also checks a GraphicsInit flag and sets mode 13h once if needed.
- DrawName – iterates through the LetterTable and draws each glyph.
- DrawLetter – draws one glyph using the rotation algorithm described above.

Using procedures makes the program easier to read and to extend (for example, another name or different rotation values).

8. Conclusion

The final program:

- uses the **tiny memory model** and a .COM executable suited for emu8086;

- enters **graphics mode 13h** and uses **BIOS interrupts** only (no DOS graphics libraries);
- represents letters as 8×8 bitmaps and animates them with a simple shear-based rotation algorithm;
- responds correctly to keyboard input: R and L change the rotation frame, E exits.