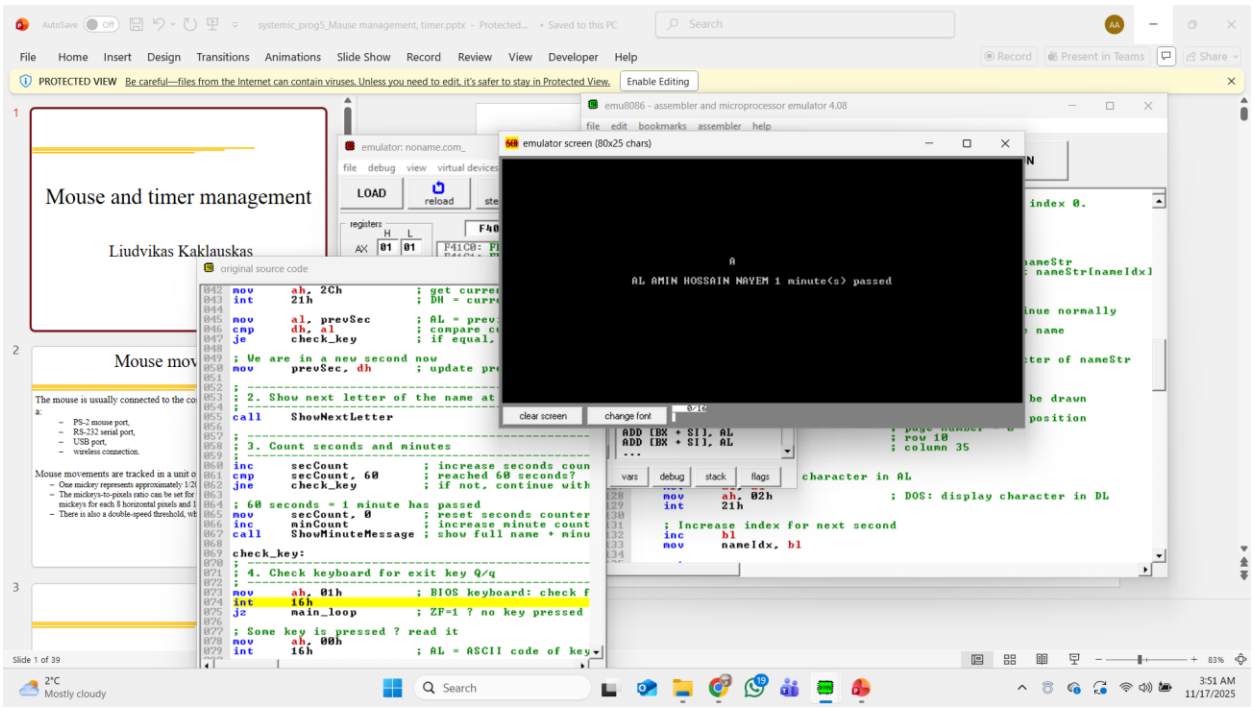

1. Task description

In this lab work I had to create a “text clock” in 8086 assembly language using emu8086. The requirements were:

1. On the screen, in one fixed position, the letters of my name must appear one after another, one letter per second.
2. The letters have to repeat in a loop when the name ends.
3. The program must count real seconds using DOS or BIOS time functions, not just a fake software delay.
4. After every 60 seconds, the program must display my surname and the number of minutes passed.
5. The clock must keep running until the user presses the Q key (upper or lower case), then the program terminates.

My final program uses my name “AL AMIN” for the clock and prints “AL AMIN HOSSAIN NAYEM X minute(s) passed” after each minute.

---

2. Used memory model and program structure

I used the **TINY memory model**:

1. .model tiny and org 100h are used to create a **COM program**.
2. In a COM program, code and data are in a single segment, and the program is loaded at offset 100h. This matches the examples from the lecture slides.
3. At the start of the program I execute:
4. push cs
5. pop ds

to make sure that DS = CS, so all data labels can be accessed correctly in the same segment.

I chose the TINY model because:

1. It is the simplest model for small programs like this.
2. It is directly supported and recommended in the emu8086 examples.
3. It matches how COM programs are explained in the course material, so it is consistent with lectures.

---

3. Used interrupts and instructions

I use several standard BIOS and DOS interrupts:

1. int 21h, AH = 2Ch – Get current time
    - Returns hours (CH), minutes (CL), seconds (DH), and hundredths (DL).
    - I only use DH (seconds) to detect when a new second starts.
2. int 10h, AH = 0 – Set video mode 3 (80x25 text)
    - Ensures the program runs in a clean text screen.
3. int 10h, AH = 06h – Scroll window / clear screen
    - I use this to clear the whole screen at the beginning.
4. int 10h, AH = 02h – Set cursor position
    - Used to place the clock letter at a fixed row/column.
    - Used again to place the minute message in a separate place on the screen.
5. int 21h, AH = 02h – Print one character (DL)

- Used inside ShowNextLetter to display the current letter of my name.

6. int 21h, AH = 09h – Print a string terminated by $
    - Used in ShowMinuteMessage to print my full name, the minute number string, and the static text " minute(s) passed".
7. int 16h, AH = 01h and AH = 00h – Keyboard functions
    - AH = 01h checks if a key is pressed (non-blocking).
    - If a key is available, AH = 00h reads the key into AL.
    - I compare AL with 'Q' and 'q' to exit the program.
8. int 21h, AH = 4Ch – Program termination
    - Used at the end to return control to DOS.

---

4. Algorithm description

The main algorithm is a simple **polling loop** based on comparing the current seconds value with the previous one.
The logic is:

1. At the start, I read the time using int 21h, AH = 2Ch and store the seconds (DH) into prevSec.
2. I enter the main loop main_loop.

Inside main_loop I do the following steps:

1. Call int 21h, AH = 2Ch again to get the current seconds in DH.
2. Compare DH with prevSec.
    1. If they are equal, it means it is still the same second, so I skip the update and only check the keyboard.
    2. If they are different, it means a new second started:
        1. I update prevSec to the new seconds value.
        2. I call ShowNextLetter to show the next character of my name in the same screen position.
        3. I increase secCount. If secCount reaches 60, then:

1. I reset secCount to 0.
2. I increase minCount.
3. I call ShowMinuteMessage to print the full name and the number of minutes passed.

3. At the end of each loop iteration, I check the keyboard with int 16h.
   If the user pressed Q or q, the program jumps to quit and finishes. Otherwise, it returns to main_loop.

The ShowNextLetter procedure:

1. Uses nameIdx as an index into the string nameStr.
2. Fetches the character at that index.
3. If the character is $, it resets nameIdx to 0 and starts again from the first letter.
4. Sets the cursor to a fixed position (row 10, column 35).
5. Prints the character with AH = 02h, int 21h.
6. Increments nameIdx for the next second.

The ShowMinuteMessage procedure:

1. Converts the minCount value (0–9) into an ASCII digit by adding '0'.
2. Saves this digit as the first character of the minStr buffer ("0$").
3. Sets the cursor to another position (row 12, column 20).
4. Prints surnameMsg (my full name), then minStr, then minutesText (“ minute(s) passed”).

---

5. Simple explanation and summary

In simple words, my program is a digital “text clock” that uses the system time provided by DOS. Every second it changes the letter on the screen to the next character of my name. When it reaches the end of the name, it starts again from the beginning. At the same time, it counts how many seconds and minutes have passed. Each time 60 seconds are counted, it prints a line with my full name and how many minutes have passed. The program keeps running until the user presses Q.

This program demonstrates:

1. How to work with the TINY model and COM-style programs in emu8086.
2. How to use DOS and BIOS interrupts for:
    1. Getting the current time.
    2. Printing text.
    3. Controlling the cursor.
    4. Reading the keyboard.
3. How to implement a simple time-based loop (polling algorithm) using real seconds instead of just delay loops.

Overall, the project helped me practice interrupt usage, screen control, and basic time-based logic in 8086 assembly.