Computer Architecture Project
Traffic Light Control System

202004245 Park Chan Mi
202303594 Lee Hyun Soo

2025.05.30

1. Introduction

This project simulates a real-world traffic light controller using 8086 assembly language. It includes dynamic light switching for vehicles, pedestrian signal processing, and night mode operation with blinking yellow lights. The program also supports user interaction via keyboard inputs. The goal is to demonstrate interrupt-based I/O control and conditional state management in low-level programming.

In order to enhance our understanding of 8086-based traffic control logic, we referred to two GitHub repositories during development. One served as a reference for implementing the day and night mode transitions, while the other provided insight into pedestrian signal management. These resources were used solely for conceptual guidance — all implementations in this project were customized and rebuilt to meet our specific design goals.

Traffic-Light-Control-System-Assembly/main.asm at main · JixCan/Traffic-Light-Control-System-Assembly · GitHub

traffic-light-asm/traffic.asm at master · ayadij/traffic-light-asm · GitHub

2. System Overview

This system is implemented and tested using the emu8086 simulator, with the built-in Virtual Traffic Light Device for visualizing vehicle signals. The program interacts with various virtual hardware components using low-level assembly instructions and BIOS interrupts.

The following components and mechanisms are utilized:

- **Port 4 Output (OUT 4, AX):** Controls the traffic lights for four directions. Each set of bits in AX maps to Red, Yellow, and Green lights for North-South and East-West traffic.
- **Keyboard Input (INT 21h):** Accepts user keypresses to switch between modes:
    - 'P' triggers Day mode
    - 'N' activates Night mode
    - 'B' initiates the pedestrian signal (Day mode only)
- **Screen Display (INT 10h):** Outputs characters such as 'G', 'Y', 'R', '*', and '.' to visualize traffic light and pedestrian signal states, since the emulated LED indicator is not available for pedestrian output.
- **Delay Loops (INT 15h):** Used to simulate realistic time-based transitions between traffic light states. Time delays are precisely controlled using CX:DX register pairs to specify millisecond intervals (e.g., 5s, 2s, 1s, 0.2s).

3. Feature Implementation

3.1 Day Mode

The system cycles through:
- North-South Green (display 'G')
- All Yellow (display 'Y')
- East-West Green (display 'G')
- All Yellow (display 'Y')

3.2 Pedestrian Mode

When the 'B' key is pressed during day mode:
- Red light for pedestrians (display 'R')
- Green light (display 'G')
- Slow blinking: '* . * . * .'
- Fast blinking: '*.*.*.*.'

3.3 Night Mode

When the 'N' key is pressed:
- Yellow lights blink in a loop: '* . * .'
- Loop continues until 'P' is pressed to return to day mode

4. Code Explanation

Key procedures:
- main_loop: Detects user input and dispatches to modes
- day_mode / night_mode: Handle traffic light sequencing
- pedestrian_signal: Manages pedestrian light flow
- delay routines: Create realistic waiting times via INT 15h
- print routines: Show state output to screen using INT 10h

5. Detailed Code Explanation

This section provides an in-depth breakdown of the complete assembly code for the 8086-based traffic light controller system. Each procedure is explained in terms of its function, role in the system, and hardware interaction.

5.1 start – Initialization and Red Light Setup

This procedure is the entry point of the program. It initializes memory segments and displays the red light for all directions as the starting state.

```
6       start:
7           mov ax, @data
8           mov ds, ax
```

Initializes the data segment register (DS) with the appropriate segment value, which is mandatory when using .model small.

```
11          mov ax, 0249h
12          out 4, ax
13          call delay5s
```

Sends the initial traffic light state to **port 4**, where red lights are activated for all directions. A 5-second delay simulates the initial pause after system start-up to stabilize intersection flow.

5.2 main_loop – Mode Selection via Keyboard

This is the main loop of the program, continuously awaiting user input to decide which operational mode (Day/Night) to enter.

```
15  main_loop:
16      mov ah, 1
17      int 21h
18      cmp al, 'N'
19      je night_mode
20      cmp al, 'P'
21      je day_mode
22      jmp main_loop
```

By using INT 21h, function AH=1, the system waits for user input.

- Press 'N' → Enter Night Mode
- Press 'P' → Enter Day Mode
- Any other input → re-loop

5.3 day_mode – Daytime Traffic Light Control

The day mode simulates a realistic traffic control flow by alternating green light direction between north-south and east-west lanes, using yellow lights in between for safe transitions.

```
25  day_mode:
26      call print_day
```

Prints 'D:' on the screen, signaling that the system has entered day mode.

▸ **Step 1: North-South Green**

```
29      mov ax, 0000001100001100b
30      out 4, ax
31      call print_text_green
32      call delay5s
```

Activates green lights in the **north-south** direction and red in the east-west direction. Displays 'G' to represent green.

▸ **Step 2: All Yellow**

```
35          mov ax, 0000011110011110b
36          out 4, ax
37          call print_text_yellow
38          call delay2s
39
```

Sets all directions to yellow, signaling an upcoming change.

▸ **Step 3: East-West Green**

```
41          mov ax, 0000100001100001b
42          out 4, ax
43          call print_text_green
44          call delay5s
45
```

Activates green lights in the east-west direction and red in the north-south direction.

▸ **Step 4: All Yellow (again)**

```
47          mov ax, 0000110011110011b
48          out 4, ax
49          call print_text_yellow
50          call delay2s
```

5.4 Pedestrian Signal Check (Inside Day Mode)

During day mode, the program checks whether the 'B' key is pressed to trigger pedestrian signal routines.

```
53          mov ah, 1
54          int 21h
55          cmp al, 'B'
56          jne no_ped
57
58          call pedestrian_signal
59
```

This allows users to simulate a pedestrian crossing request dynamically.

5.5 pedestrian_signal – Pedestrian Walk Cycle

This routine simulates a real pedestrian crossing signal, including waiting, crossing, slow blinking, and fast blinking indicators.

▸ **Red Signal**

```
88          mov al, 'R'
89          call print_char
90          call delay5s
91
```

Displays 'R' to indicate that pedestrians must wait (red signal).

▸ **Green Signal**

```
92          ; Green ON
93          mov al, 'G'
94          call print_char
95          call delay5s
```

Displays 'G' to indicate it's safe to cross.

▸ **Slow Blinking**

```
109     ; 점멸 (느리게)
110     mov si, 3
111 blink_loop:
112     mov al, '*'
113     call print_char
114     call delay1s
115     mov al, '.'
116     call print_char
117     call delay1s
118     dec si
119     jnz blink_loop
```

Performs 3 slow blinking cycles (* . * . * .), indicating crossing time is ending.

▸ **Fast Blinking**

```
121     ; 빠른 점멸
122     mov si, 4
123 fast_blink:
124     mov al, '*'
125     call print_char
126     call delay200ms
127     mov al, '.'
128     call print_char
129     call delay200ms
130     dec si
131     jnz fast_blink
132
133     ret
```

Performs 4 fast blinking cycles to simulate the end of pedestrian crossing time.

5.6 night_mode – Blinking Yellow at Night

Night mode disables full signal cycles and replaces them with blinking yellow lights, simulating low-traffic nighttime conditions.

```
64      night_mode:
65          call print_night
66
```

Prints 'N:' to indicate night mode is active.

```
67      night_loop:
68          mov ax, 0000001000001000b  ; yellow
69          out 4, ax
70          call print_text_blink
71          call delay1s
72
73          mov ax, 0000000000000000b  ; off
74          out 4, ax
75          call print_text_off
76          call delay1s
77
```

Alternates between yellow on and all-off every second, creating a blinking effect.

```
84      ; 키 입력 여부 확인 (비차단 방식)
85      mov ah, 01h
86      int 16h           ; BIOS: Check for key press
87      jz night_loop     ; ZF=1 → 키 없음 → 계속 루프
88
89      ; 키 있음 → 읽기
90      mov ah, 00h
91      int 16h           ; AL에 입력된 문자
92
93      cmp al, 'P'
94      je main_loop      ; 낮 모드로 전환
95      jmp night_loop
```

Returns to day mode when 'P' is pressed.

5.7 Print Routines – Visual Feedback for Signal States

These subroutines simulate traffic light states using ASCII output on the screen. All of them rely on INT 10h, function AH=0Eh (teletype output).

- print_day: Prints 'D:'
- print_night: Prints 'N:'
- print_text_green: 'G'

- print_text_yellow: 'Y'
- print_text_blink: '*'
- print_text_off: '.'
- print_char: Prints the character stored in AL

This strategy allows visibility of light states even when graphical components are unavailable.

5.8 Delay Routines – Real-Time Simulation

All delay subroutines use BIOS interrupt INT 15h, function AH=86h to pause execution for specific durations.

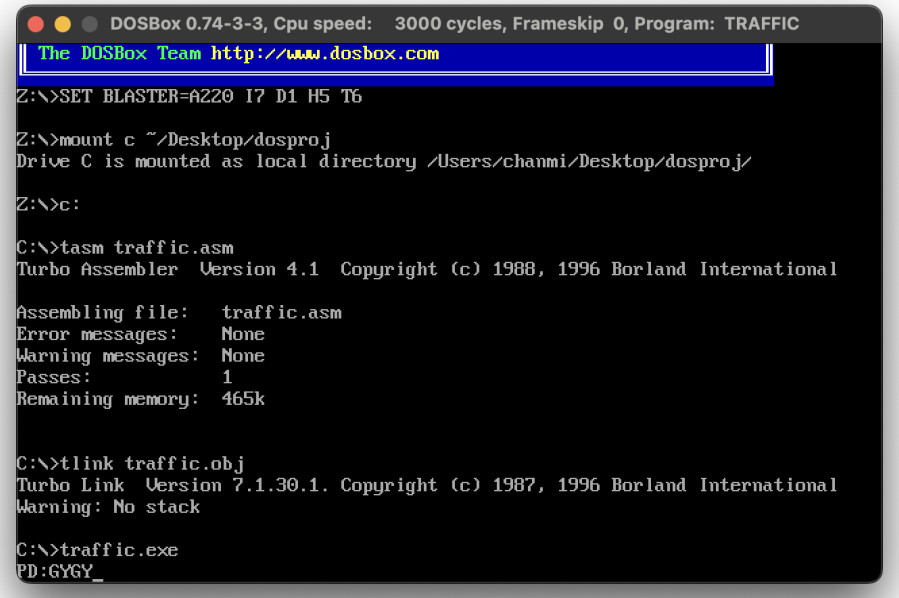
- delay5s: 5 seconds
- delay2s: 2 seconds
- delay1s: 1 second
- delay200ms: 0.2 seconds

Each uses CX:DX to specify delay time. These are crucial for simulating real-world timing between light transitions.

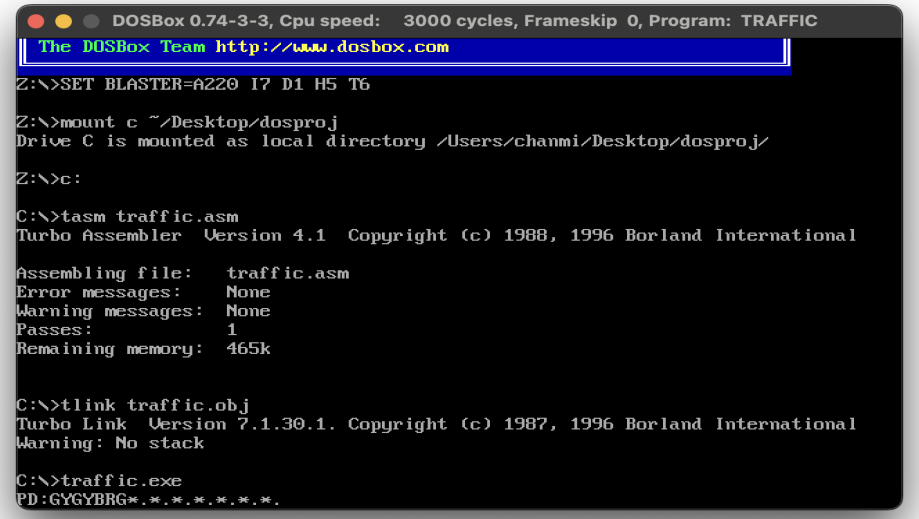
6. Testing and Result

6.1 Screenshot of Each Function

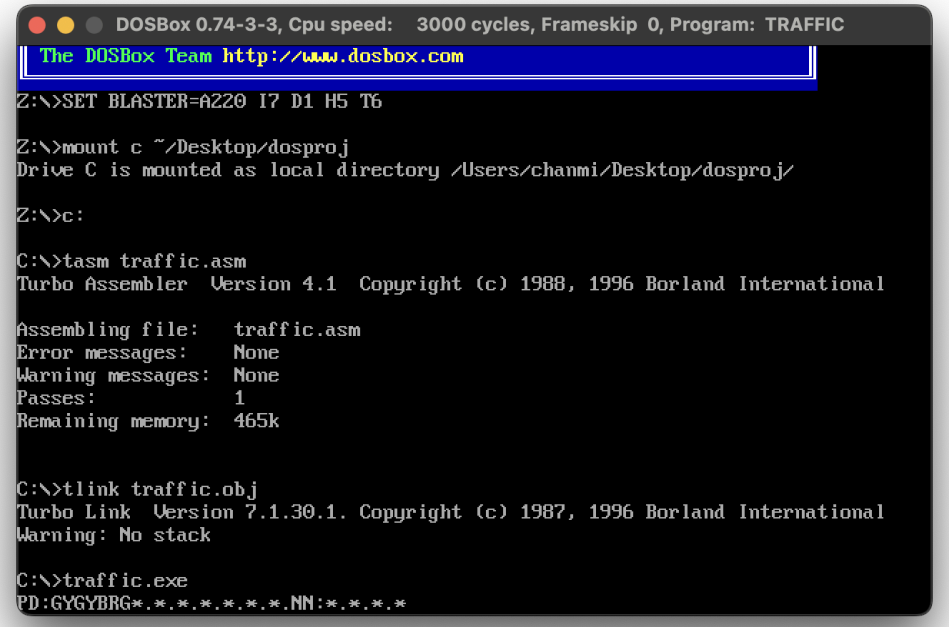
1) Day mode



2) Pedestrian Walk mode



3) Night mode



7. Conclusion

This project was completed as a team of two, which allowed us to divide responsibilities and collaborate effectively throughout the development process. Working together helped us explore multiple perspectives and troubleshoot problems more efficiently, especially when implementing low-level control logic in assembly.

By developing a traffic light control system using 8086 assembly, we both deepened our

understanding of BIOS interrupts, port-based I/O, and real-time signal processing. We worked closely to design modular subroutines for Day mode cycling, pedestrian crossing behavior, and Night mode blinking, while also ensuring user interaction through keyboard input.

Simulating hardware behavior without physical devices was one of the most challenging parts of the project. To overcome this, we used port output and ASCII-based display to represent LED signals and pedestrian states. It required us to think critically about how to emulate timing, state changes, and user feedback using only software tools.

Through this project, we not only improved our technical proficiency in assembly language but also gained valuable experience in pair programming and collaborative problem solving. The process strengthened our ability to manage complexity in a constrained, hardware-simulated environment and deepened our appreciation for how embedded systems operate at a low level.