Assignment # 3

Carefully study the traffic light problem described below along with its solution. Based on your understanding of the system's logic, design and implement a solution for the second problem, whose statement is provided at the end of this document.

Traffic Lights Problem Statement

Problem:

You are tasked with designing a traffic light controller system for an intersection with four directions:

- **North-South** (NS)
- **East-West** (EW)

Each direction will have a **Green**, **Yellow**, and **Red** light. The traffic lights should operate in a typical pattern:

- **Green light**: Cars can go in the given direction.
- **Yellow light**: Warning for cars that the light will soon turn red.
- **Red light**: Cars must stop.

The traffic lights need to operate with the following requirements:

1. The **North-South** and **East-West** directions should not have the green light at the same time (no collision).
2. The system should run in a **cyclic manner**:
    - North-South green for 10 seconds, followed by 3 seconds of yellow, then red.
    - East-West green for 10 seconds, followed by 3 seconds of yellow, then red.
3. The system should be able to handle **interrupts** (e.g., pedestrian button press) and **traffic sensors** if desired, but for this basic example, we will assume a fixed time cycle.
4. The traffic lights should alternate every 13 seconds (10 seconds of green + 3 seconds of yellow).

Solution Design

We will implement this system using **Timer_A** interrupts for timed transitions between green, yellow, and red lights, and **GPIO** for controlling the lights on each direction (North-South and East-West). The green and yellow light durations will be fixed, and we will toggle the GPIO pins corresponding to each light.

GPIO Pin Assignments (Example MSP430FR2355):

- **P1.0**: NS Green
- **P1.1**: NS Yellow
- **P1.2**: NS Red

- **P2.0**: EW Green
- **P2.1**: EW Yellow
- **P2.2**: EW Red
- **Assembly Code Implementation for Traffic Light Controller:**
- assembly
- CopyEdit
- ; Initialize the system
- INIT:
- ; Configure P1.0, P1.1, P1.2 as output (NS lights)
- BIS.B #BIT0 + BIT1 + BIT2, &P1DIR
- ; Configure P2.0, P2.1, P2.2 as output (EW lights)
- BIS.B #BIT0 + BIT1 + BIT2, &P2DIR
-
- ; Set initial state: NS Green, EW Red
- BIS.B #BIT0, &P1OUT ; P1.0 = Green
- BIC.B #BIT1, &P1OUT ; P1.1 = Yellow off
- BIC.B #BIT2, &P1OUT ; P1.2 = Red off(NS)
- BIC.B #BIT0, &P2OUT ; P2.0 = Green off
- BIC.B #BIT1, &P2OUT ; P2.1 = Yellow off
- BIS.B #BIT2, &P2OUT ; P2.2 = Red on (EW)
-
- ; Set up Timer_B for 1 second interrupt (1Hz)
- MOV.W #TBSSEL_2 + MC_1, &TB0CTL ; Timer B with SMCLK, up mode
- MOV.W #1000, &TB0CCR0 ; Set Timer for 1 second
- BIS.W #CCIE, &TB0CCTL0 ; Enable interrupt for CCR0
  EINT
-
- MAIN_LOOP:
- JMP MAIN_LOOP
-
- TB0CCR0_ISR:
- BIC.W #CCIFG, &TB0CCTL0
- ; Switch to the next light state
- CALL SWITCH_LIGHTS
- RETI
-
- ; Function to switch between light states
- SWITCH_LIGHTS:
- ; Check current state and transition to the next state
- ; NS Green -> NS Yellow -> NS Red -> EW Green -> EW Yellow -> EW
  Red -> Repeat
-
- ; If NS Green, change to NS Yellow
- BIT.B #BIT0, &P1OUT ; Check if NS Green
- JZ NS_GREEN_DONE
- BIS.B #BIT1, &P1OUT ; P1.1 = NS Yellow on
- BIC.B #BIT0, &P1OUT ; P1.0 = NS Green off
- MOV.W #3000, &TB0CCR0 ; 3 seconds for yellow
- JMP LIGHT_DONE
-
- NS_GREEN_DONE:
- ; If NS Yellow, change to NS Red
- BIT.B #BIT1, &P1OUT ; Check if NS Yellow

```
    JZ      NS_YELLOW_DONE
    BIS.B   #BIT2, &P1OUT   ; P1.2 = NS Red on
    BIC.B   #BIT1, &P1OUT   ; P1.1 = NS Yellow off
    MOV.W   #10000, &TB0CCR0; 10 seconds for red
    JMP     LIGHT_DONE

NS_YELLOW_DONE:
    ; If NS Red, change to EW Green
    BIT.B   #BIT2, &P1OUT   ; Check if NS Red
    JZ      NS_RED_DONE
    BIC.B   #BIT2, &P1OUT   ; NS Red off
    BIS.B   #BIT0, &P2OUT   ; EW Green on
    MOV.W   #10000, &TB0CCR0 ; 10 seconds for EW Green
    JMP     LIGHT_DONE

NS_RED_DONE:
    ; If EW Green, change to EW Yellow
    BIT.B   #BIT0, &P2OUT   ; Check if EW Green
    JZ      EW_GREEN_DONE
    BIS.B   #BIT1, &P2OUT   ; EW Yellow on
    BIC.B   #BIT0, &P2OUT   ; EW Green off
    MOV.W   #3000, &TB0CCR0 ; 3 seconds for EW Yellow
    JMP     LIGHT_DONE

EW_GREEN_DONE:
    ; If EW Yellow, change to EW Red
    BIT.B   #BIT1, &P2OUT   ; Check if EW Yellow
    JZ      EW_YELLOW_DONE
    BIS.B   #BIT2, &P2OUT   ; EW Red on
    BIC.B   #BIT1, &P2OUT   ; EW Yellow off
    MOV.W   #10000, &TB0CCR0 ; 10 seconds for EW Red
    JMP     LIGHT_DONE

EW_YELLOW_DONE:
    ; Return to NS Green
    BIT.B   #BIT2, &P2OUT   ; Check if EW Red
    JZ      EW_RED_DONE
    BIC.B   #BIT2, &P2OUT   ; EW Red off
    BIS.B   #BIT0, &P1OUT   ; NS Green on
    MOV.W   #10000, &TB0CCR0 ; 10 seconds for NS Green
    JMP     LIGHT_DONE

EW_RED_DONE:
    JMP     SWITCH_LIGHTS

LIGHT_DONE:
    RET
```

Explanation of Code:

1. **GPIO Initialization**:

    - Pins **P1.0**, **P1.1**, and **P1.2** are configured as outputs for controlling the North-South (NS) lights.
    - Pins **P2.0**, **P2.1**, and **P2.2** are configured as outputs for controlling the East-West (EW) lights.
2. **Timer Configuration**:
    - **Timer_B** is set to use **SMCLK** and **up mode** to generate interrupts every **1 second**.
    - **CCR0** is set to trigger every 1 second, and the interrupt is enabled by setting the **CCIE** bit in the **TB0CCTL0** register.
3. **Interrupt Service Routine (ISR)**:
    - Every time the timer overflows, the ISR is executed.
    - The ISR calls the **SWITCH_LIGHTS** function to change the light states in a cyclic order: NS Green -> NS Yellow -> NS Red -> EW Green -> EW Yellow -> EW Red.
4. **Light State Transitions**:
    - The system toggles between **Green**, **Yellow**, and **Red** lights for both NS and EW directions based on the defined timing. The duration for each state is managed by **Timer_A** interrupts, which triggers the state transition.

Traffic Light Sequence Example:

- **0-10 seconds**: NS Green, EW Red.
- **10-13 seconds**: NS Yellow, EW Red.
- **13-23 seconds**: NS Red, EW Green.
- **23-26 seconds**: NS Red, EW Yellow.
- **26-36 seconds**: NS Green, EW Red (Repeat).

Conclusion:

This solution is a basic **traffic light control system** using **Timer_A interrupts** to handle the cyclic transitions of the traffic lights. Each traffic light direction (NS and EW) has its own set of green, yellow, and red timings, and

ASSIGNMENT # 3

Design an embedded system to **track student attendance** and manage **waiting status** for a class session, using:

- **LEDs** to indicate student status.
- **Timer interrupts** to simulate entry and attendance processing.
- **Assembly language**.
- No extra hardware except GPIO (no UART, sensors, etc.).

Pin Configuration

Pin	Purpose	Description
P1.0	**Student Entry Allowed**	Green LED: student may enter
P1.1	**Student Verifying**	Yellow LED: ID check in progress
P1.2	**Student Attended**	Red LED: attendance marked
P2.0	**Next Student Waiting**	Green LED: another student is waiting
P2.1	**Waiting List Active**	Yellow LED: queue in progress
P2.2	**All Present**	Red LED: session attendance complete

We're assuming students arrive one by one and pass through **three states**:

1. **Allowed to Enter**
2. **Verifying**
3. **Attendance Complete**

State Description & Timing

State No	Description	Duration
0	Allow entry (P1.0, P2.0 ON)	15 seconds
1	Verifying ID (P1.1, P2.1 ON)	9 seconds
2	Attendance Marked (P1.2, P2.2 ON)	12 seconds
3	Repeat for next student	–

```
        .cdecls C,LIST,"msp430.h"
        .text
        .global _start

; ------------------------------
; Start of program
; ------------------------------
init:
        BIS.B   #BIT0+BIT1+BIT2, &P1DIR     ; P1.0–P1.2 as output
        BIS.B   #BIT0+BIT1+BIT2, &P2DIR     ; P2.0–P2.2 as output

        CALL    #STATE_0                ; Initial state: Allow Entry

        ; Setup Timer_B0 for 1s interval
        MOV.W   #TBSSEL__SMCLK + MC__UP, &TB0CTL   ; SMCLK, Up Mode
        MOV.W   #1000, &TB0CCR0                ; 1ms with dividers
        BIS.W   #CCIE, &TB0CCTL0               ; Enable CCR0 interrupt
        Bic.w    #CCIFG, &TB0CCTL0
        EINT                            ; Enable interrupts

; Main loop (idle)
```

```
; ----------------------------
MAIN_LOOP:
        JMP     MAIN_LOOP

; ----------------------------
; Timer_B ISR (every 1 sec)
; ----------------------------
        .sect ".int09"
        .short TB0CCR0_ISR

TB0CCR0_ISR:
        BIC.W   #CCIFG, &TB0CCTL0       ; Clear interrupt flag
        INC     &state_counter          ; Count seconds

        MOV     &current_state, R10
        CMP     #0, R10
        JZ      CHECK_STATE_0
        CMP     #1, R10
        JZ      CHECK_STATE_1
        CMP     #2, R10
        JZ      CHECK_STATE_2
        RETI

; ----------------------------
; Check duration for each state
; ----------------------------
CHECK_STATE_0:
        CMP     #15, &state_counter
        JNZ     END_CHECK
        CLR     &state_counter
        MOV     #1, &current_state
        CALL    #STATE_1
        JMP     END_CHECK

CHECK_STATE_1:
        CMP     #9, &state_counter
        JNZ     END_CHECK
        CLR     &state_counter
        MOV     #2, &current_state
        CALL    #STATE_2
        JMP     END_CHECK

CHECK_STATE_2:
        CMP     #12, &state_counter
        JNZ     END_CHECK
        CLR     &state_counter
        MOV     #0, &current_state
```

```
        CALL    #STATE_0

END_CHECK:
        RETI

; ------------------------------
; STATE FUNCTIONS
; ------------------------------
STATE_0:                    ; Allow Entry
        CALL    #CLEAR_LEDS
        BIS.B   #BIT0, &P1OUT           ; P1.0 = Entry Allowed
        BIS.B   #BIT0, &P2OUT           ; P2.0 = Student Waiting
        RET

STATE_1:                    ; Verifying ID
        CALL    #CLEAR_LEDS
        BIS.B   #BIT1, &P1OUT           ; P1.1 = Verifying
        BIS.B   #BIT1, &P2OUT           ; P2.1 = Queue Active
        RET

STATE_2:                    ; Attendance Complete
        CALL    #CLEAR_LEDS
        BIS.B   #BIT2, &P1OUT           ; P1.2 = Attended
        BIS.B   #BIT2, &P2OUT           ; P2.2 = All Present
        RET

CLEAR_LEDS:                 ; Turn OFF all LEDs
        BIC.B   #BIT0+BIT1+BIT2, &P1OUT
        BIC.B   #BIT0+BIT1+BIT2, &P2OUT
        RET

; ------------------------------
; Data Section
; ------------------------------
        .data
state_counter:  .word   0
current_state:  .word   0
```

```
        .cdecls C,LIST,"msp430.h"
        .text
        .global _start

; ----------------------------
; INIT
; ----------------------------
_start:
        ; Configure P1.0–P1.2 and P2.0–P2.2 as output
        BIS.B   #BIT0+BIT1+BIT2, &P1DIR     ; NS
        BIS.B   #BIT0+BIT1+BIT2, &P2DIR     ; EW

        ; Set initial state: NS Green, EW Red
        CALL    #STATE_0                    ; NS Green ON

        ; Timer_B Setup (1-second interrupt)
        MOV.W   #TBSSEL__SMCLK + MC__UP, &TB0CTL
        MOV.W   #1000, &TB0CCR0             ; adjust for actual 1s
        BIS.W   #CCIE, &TB0CCTL0            ; Enable interrupt

        MOV.W   #WDTPW + WDTHOLD, &WDTCTL    ; Stop watchdog
        EINT                                ; Enable global interrupts

; ----------------------------
; MAIN LOOP (Idle)
; ----------------------------
MAIN_LOOP:
        JMP     MAIN_LOOP

; ----------------------------
; TIMER_B ISR (Every 1 Second)
; ----------------------------
        .sect ".int09"
        .short TB0CCR0_ISR

TB0CCR0_ISR:
        BIC.W   #CCIFG, &TB0CCTL0           ; Clear interrupt flag
        INC     &state_counter              ; Count seconds

        MOV     &current_state, R10
        CMP     #0, R10
        JZ      CHECK_STATE_0
        CMP     #1, R10
        JZ      CHECK_STATE_1
        CMP     #2, R10
        JZ      CHECK_STATE_2
        CMP     #3, R10
```

```
        JZ     CHECK_STATE_3
        CMP    #4, R10
        JZ     CHECK_STATE_4
        CMP    #5, R10
        JZ     CHECK_STATE_5
        RETI

; ----------------------------
; CHECK STATES
; ----------------------------
CHECK_STATE_0:          ; NS Green (10s)
        CMP    #10, &state_counter
        JNZ    END_CHECK
        CLR    &state_counter
        MOV    #1, &current_state
        CALL   #STATE_1
        JMP    END_CHECK

CHECK_STATE_1:          ; NS Yellow (3s)
        CMP    #3, &state_counter
        JNZ    END_CHECK
        CLR    &state_counter
        MOV    #2, &current_state
        CALL   #STATE_2
        JMP    END_CHECK

CHECK_STATE_2:          ; EW Green (10s)
        CMP    #10, &state_counter
        JNZ    END_CHECK
        CLR    &state_counter
        MOV    #3, &current_state
        CALL   #STATE_3
        JMP    END_CHECK

CHECK_STATE_3:          ; EW Yellow (3s)
        CMP    #3, &state_counter
        JNZ    END_CHECK
        CLR    &state_counter
        MOV    #0, &current_state
        CALL   #STATE_0
        JMP    END_CHECK

END_CHECK:
        RETI

; ----------------------------
; STATES (LED Logic)
```

```
; ------------------------------
STATE_0:                ; NS Green, EW Red
        CALL   #CLEAR_LEDS
        BIS.B  #BIT0, &P1OUT    ; NS Green
        BIS.B  #BIT2, &P2OUT    ; EW Red
        RET

STATE_1:                ; NS Yellow, EW Red
        CALL   #CLEAR_LEDS
        BIS.B  #BIT1, &P1OUT    ; NS Yellow
        BIS.B  #BIT2, &P2OUT    ; EW Red
        RET

STATE_2:                ; NS Red, EW Green
        CALL   #CLEAR_LEDS
        BIS.B  #BIT2, &P1OUT    ; NS Red
        BIS.B  #BIT0, &P2OUT    ; EW Green
        RET

STATE_3:                ; NS Red, EW Yellow
        CALL   #CLEAR_LEDS
        BIS.B  #BIT2, &P1OUT    ; NS Red
        BIS.B  #BIT1, &P2OUT    ; EW Yellow
        RET


; ------------------------------
; Data Section
; ------------------------------
        .bss
state_counter:  .word   0
current_state:  .word   0
```