Zapolska, Anna
EE3123 Embedded Systems
September 6th, 2023
Project 2 Report

I. Introduction

This project aims to familiarize the students with the Assembly language through implementation of the program which flashed green and red LEDs on the MSP430G2553 board at different rates. Functions like **cmp**, **bic**, **bis**, **cmp,** and **inv** were to be used in the program. Additionally, all seven addressing modes were to be used, which are: Immediate, Register, Absolute, Symbolic, Indexed, Indirect Register, Indirect Autoincrement.

II. Methods

The code was set up according to the diagram, shown in Fig. 1.

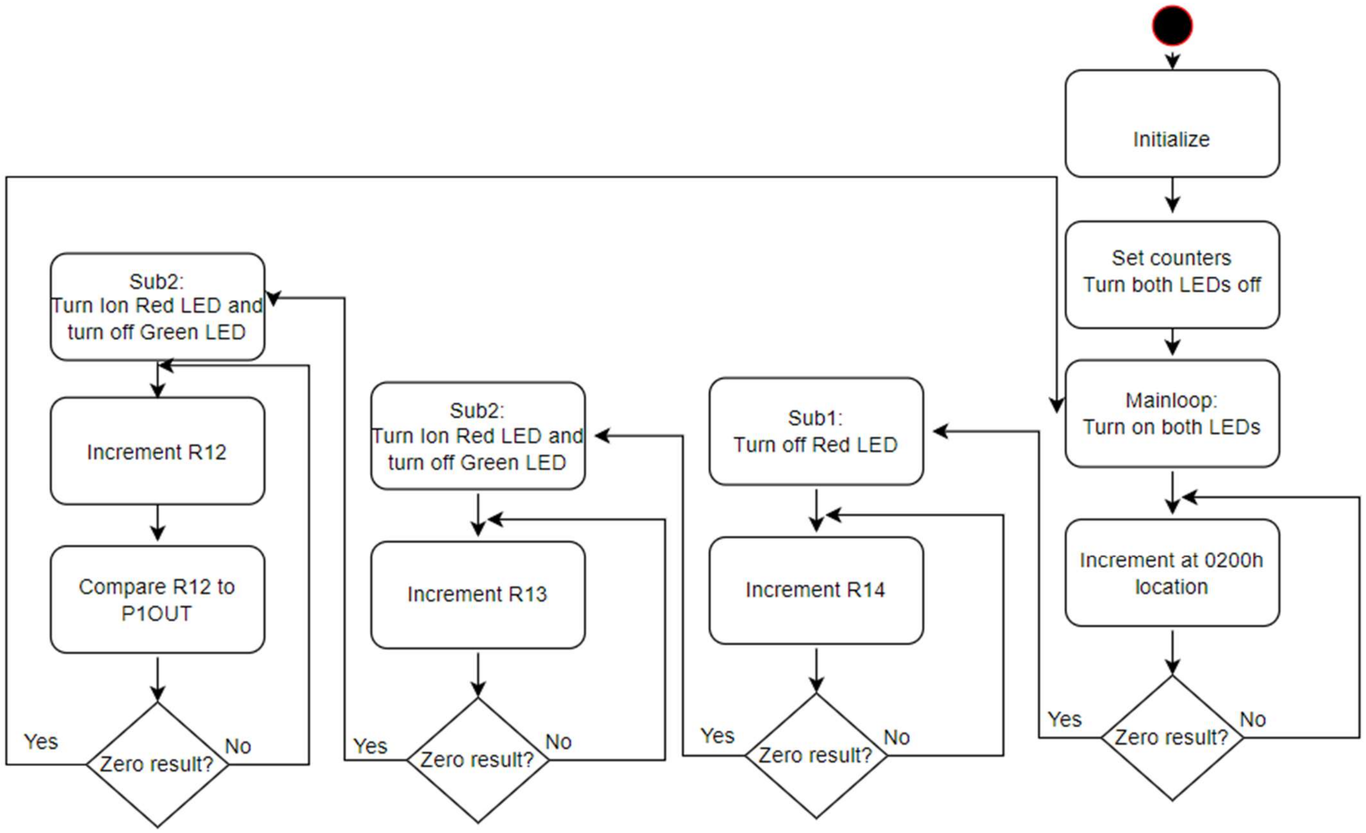

Fig.1 – Flowchart of the program

At first, both LEDs were set up using the P1DIR node. Red LED is connected to P1.0, and green LED is connected to P1.6, so 010h and 040h were used respectively.

```
SetupP1          bis.b   #001h,&P1DIR          ; sets P1.0 as an output
SetupP2          bis.b #040h,&P1DIR            ; sets P1.1 as an output
```

Next, the counters were set up. Four different counters were used for each of the state of the cycle of the program. The locations 0200h to 0206h were designated as counters and were initialized to 0000h by the bic function. Also, the location 0200h was put into the register 15.

```
SetCounters      mov.w #00202h, R15            ; R15 has the location 0202h.
                 bic.w #0FFFFh, &00200h        ; location 0200h contains 0000h
```

```
                  bic.w #0FFFFh, &00202h        ; location 0202h contains 0000h
                  bic.w #0FFFFh, &00204h        ; location 0204h contains 0000h
                  bic.w #0FFFFh, &00206h        ; location 0206h contains 0000h
```

Next, using different addressing modes as described in the comments, the contents of corresponding memory locations were moved to registers 12 through 14 for easier access in the program.

```
                  mov.w @R15, R14               ; moving contents of 0202h into R14
                                                ;(Indirect Register for @R15)
                  mov.w @R15+, R13              ; moving contents of 0204h into R13
                                                ;(Indirect Autoincrement for @R15+)
                  mov.w 2(R15), R12             ; moving contents of 0206h into R12
                                                ; (Indexed mode for 2(R15)
```

Before the beginning of the program, both LEDs are turned off (set to 0) to avoid any confusion. This is done by the bit clear function by setting P1OUT to 041h. This way both LEDs are low.

```
                  bic.b   #041h,&P1OUT          ; turns both LEDs OFF
```

Finally, the main program begins. At first both LEDs are turned on by the bit set function. Next, in the Loop 1 the counter in the 0200h location is incremented. In the next line jump not zero function is used to see if the counter returned back to zero that is the delay is over. In the case if it's not over the program comes back to loop 1 which again increments the counter. If the counter is 0 then the program jumps to the subroutine 1.

```
Mainloop          bis.b  #041h,&P1OUT            ; Turn ON P1.0 and P1.6
L1                inc    &0200h                  ; Incrementing 0200h location
                                                 ; (Absolute mode)
                  jnz     L1                     ; Checking if Delay is over
                  jmp     Sub1                   ; Jump to subroutine
```

In the subroutine 1 the red LED is turned off and the green LED is still on. Next, the counter now in register 14 is incremented. The same logic is used as the Loop 1 to check if the delay is over, and if it is over then the program jumps to the subroutine 2

```
Sub1              bic.b #001h,&P1OUT            ; Turning OFF P1.0 and P1.6 is ON
L2                inc    R14                    ; Incrementing R14 (Register mode)
                  jnz     L2                    ; Checking if Delay is over
                  jmp     Sub2                  ; Jump to the Sub2
```

In the in the subroutine 2, the levels of LEDs are inverted, that is red LED is on and green LED is turned off. In a similar way, Loop 3 increments the counter in the registers 13 and when delay is over, the program jumps to the subroutine 3.

```
Sub2              inv   &P1OUT                  ; Turns ON P1.0 and turns OFF P1.6
L3                inc   R13                     ; Incrementing R13
                  jnz   L3                      ; Checking if Delay is over
```

In the subroutine 3 both LEDs are turned off. In the same way as before loop 4 increments the counter in register 12 and checks if delay is over. It additionally compares register are 12 to P1OUT because P1OUT was set to 0. So, it basically checks if the register 12 is zero and then if that is, the program jumps to the main loop and the program repeats again.

```
Sub3              bic.b #0FFh, &P1OUT           ; Turn OFF both LEDs
L4                inc   R12                     ; Incrementing R12
                  jnz   L4                      ; Checking if Delay is over
                  cmp   R12, P1OUT              ; Comparing R15 to P1OUT because P1OUT
                                                ;has been reset (Symbolic mode for P1OUT)
                  jmp   Mainloop                ; Again to the mainloop
```

III. Results

The program worked as intended. At the very beginning, both LEDs were cleared, and then in the main loop they were both turned on (Fig. 2). After, the red LED was turned off and the green LED remained on (Fig. 3).

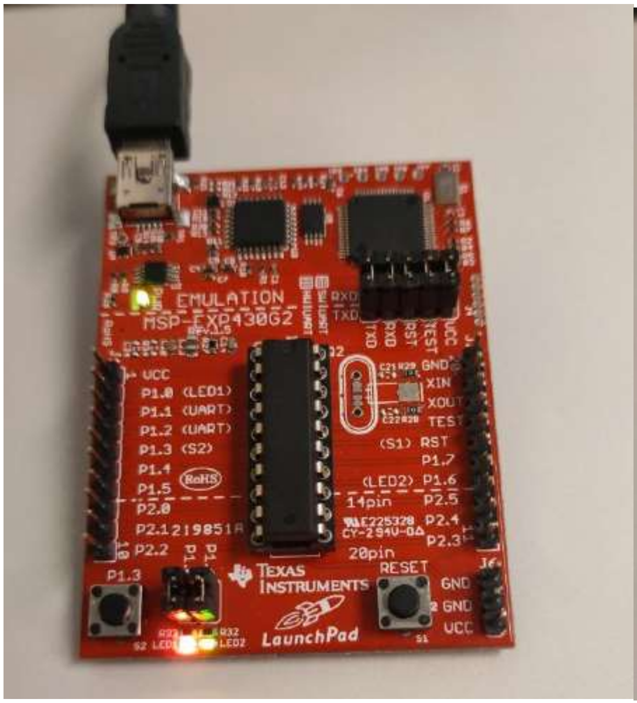

Fig. 2 – Both LEDs turned on in Mainloop

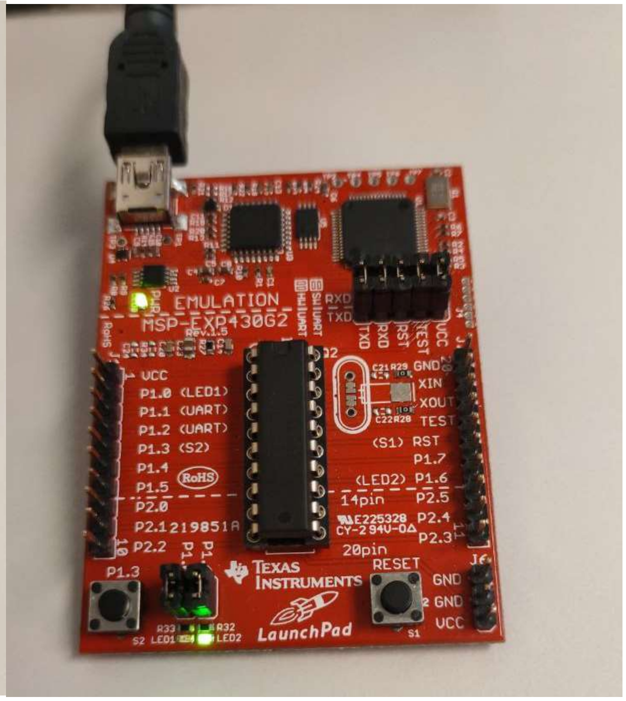

Fig. 3 – Red LED turned off in Sub1

Next, red LED turned off and Green LED was back on (Fig. 4). Finally, both LEDs are off (Fig. 5).

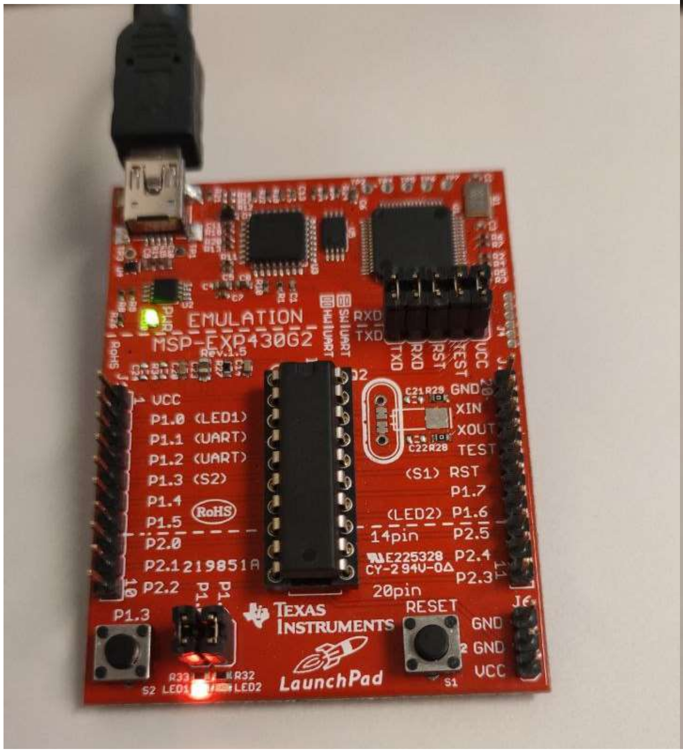

Fig.4 – Red LED is on and Green LED turned off

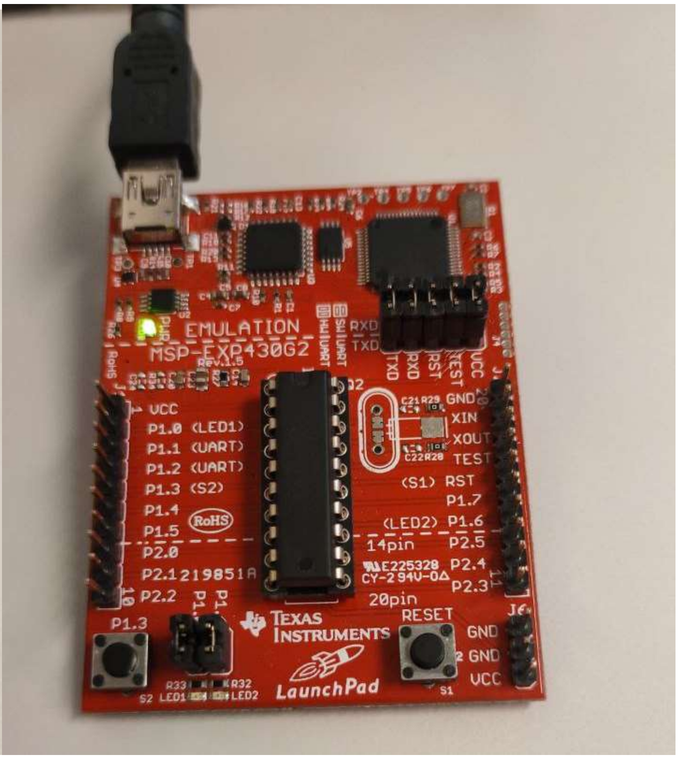

Fig.5 – both LEDs are off

As seen by the results, red LED flashed 2 times faster than the green LED which proves that program worked as intended.

IV. Conclusions

I learned how to properly use the debugger in CCS when working with the code. It is so helpful to see what registers and memory slots contain and it can be immediately tracked if something goes wrong. Also, the step function is very helpful. Decides, the compiler highlights all of the changes. I definitely saw less mistakes than when coding with high level programming languages. I was hard for me to understand and implement all of the 7 addressing modes, especially Indirect Register @R1, Autoincrement Register @R1+, and Indexed mode 2(R1). It was hard to grasp what goes where and which register should contain what information. For next time, I would not rush so much to rush on the project because I started it when it was just assigned and the code was not explained yet in class. I spent 1.5 hours trying to figure out what each line means and how to work it when I could have waited till next day for the explanation and to read in the book the explanation as well.

V. Appendix

```
;-------------------------------------------------------------------------------
; MSP430 Assembler Code Template for use with TI Code Composer Studio
;
;
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.

;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stack pointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
SetupP1     bis.b   #001h,&P1DIR            ; sets P1.0 as an output
                                            (Immediate mode for #001h)
SetupP2     bis.b #040h,&P1DIR              ; sets P1.1 as an output

SetCounters mov.w #00202h, R15              ; R15 has the location 0202h
            bic.w       #0FFFFh, &00200h    ; location 0200h contains 0000h
            bic.w       #0FFFFh, &00202h    ; location 0202h contains 0000h
            bic.w       #0FFFFh, &00204h    ; location 0204h contains 0000h
            bic.w       #0FFFFh, &00206h    ; location 0206h contains 0000h

            mov.w @R15, R14                 ; moving contents of 0202h into R14
                                            ;  (Indirect Register for @R15)
            mov.w @R15+, R13                ; moving contents of 0204h into R13
                                            ;  (Indirect Autoincrement for @R15+)
            mov.w 2(R15), R12               ; moving contents of 0206h into R12
                                            ;  (Indexed mode for 2(R15)

            bic.b   #041h,&P1OUT            ; turns both LEDs OFF

;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
Mainloop    bis.b #041h,&P1OUT              ; Turn ON P1.0 and P1.6
L1          inc   &0200h                    ; Incrementing 0200h location
                                            (Absolute mode)
            jnz   L1                        ; Checking if Delay is over
            jmp   Sub1                      ; Jump to subroutine

Sub1        bic.b #001h,&P1OUT              ; Turning OFF P1.0 and P1.6 is ON
L2          inc   R14                       ; Incrementing R14 (Register mode)
```

```
            jnz    L2                           ; Checking if Delay is over
            jmp    Sub2                         ; Jump to the Sub2

Sub2        inv   &P1OUT                        ; Turns ON P1.0 and turns OFF P1.6
L3          inc   R13                           ; Incrementing R13
            jnz   L3                            ; Checking if Delay is over

Sub3        bic.b #0FFh, &P1OUT                 ; Turn OFF both LEDs
L4          inc   R12                           ; Incrementing R12
            jnz    L4                           ; Checking if Delay is over
            cmp   R12, P1OUT                    ; Comparing R12 to P1OUT because P1OUT
                                                has been reset (Symbolic mode for P1OUT)
            jmp   Mainloop                      ; Again to the mainloop

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                    ; MSP430 RESET Vector
            .short  RESET
```