ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	2
EXPERIMENT DATE	:	15.11.2020
LAB SESSION	:	FRIDAY - 14.30
GROUP NO	:	G7

GROUP MEMBERS:

150210087	:	HİLAL KARTAL
150210100	:	SELİN YILMAZ
150210067	:	ALPER DAŞGIN

FALL 2024

Contents

1 INTRODUCTION [10 points]

During the first section, we put in place a program that would activate a LED (P1.4) when a button was pressed (P2.4). When the user first pressed the button, the micro-controller waited. When pressed, the LED became active, and the program went into an endless loop to keep the LED in that state. This section gave us an overview of basic GPIO configuration and showed us how to control LED outputs and read button states using instructions.

The second step was to write an algorithm that switches between two LEDs (P2.2 and P2.3) depending on the state of a button (P1.5). The LEDs changed states when the button was pressed; If P2.2 was on, P2.3 was turned off and on, and vice versa. To prevent unwanted fast transitions when the button is pressed and held, we also addressed button canceling. In their initial state P2.2 was set to be on and P2.3 was set to be off. This section focused on return methods, button usage, and state transitions.

In the last part, we set the microcontroller to track the number of button presses (P2.1). Instead of using an accumulator to record the count, we created a 4-bit variable in memory. This variable is reset when it reaches 16 decimal values. The count value is displayed on port 1. Our goal was to accurately recognize different presses, so we made sure that holding the button would not cause the number to constantly increase. In this section, we had the opportunity to look at advanced button handling, variable management and memory initialization in assembly language.

2 MATERIALS AND METHODS [40 points]

2.1 MATERIALS

- MSP430
- Selin's Laptop

2.2 METHODS

2.2.1 FIRST PART

The purpose in this part was checking if the third pin of of P2 is pressed. If it is pressed fourth pin P1 will light.

```
SetupP1      mov.b    #00010000b, &P1DIR
             mov.b    #00000000b, &P2DIR
```

```
 3               mov.b   #00000000b, &P1OUT
 4               mov.b   #00000000b, &P2IN
 5
 6 Mainloop1     bit.b   #00100000b, &P2IN
 7               jnz     ledOn
 8               jmp     Mainloop1
 9
10 ledOn         mov.b   #00010000b, &P1OUT
11
12 loop         jmp loop
```

Listing 1: Part1

On the first four lines, we made the setup.

Line 1: We chose fourth led of P1 as output.

Line 2: We chose all of P2 as input.

Line 3: We cleared all the output leds of P1

Line 4: We cleared all the inputs given to P2

Then we started to our first loop: *Mainloop1*

Line 6: Checks if P2's third button is pressed.

Line 7: If it is pressed go to the *ledOn* function.

Line 8: If it is not repeat the *Mainloop1* (until it is pressed).

Now we are out of Mainloop1 function.

Line 10: Lights the fourth pin of P1.

Line 12: Endless loop when the pin is on.

2.2.2 SECOND PART

In this part, the aim is implementing a program that switches back and forth between two leds when P1.5 is pressed. If P1.5 is held down leds should not change.

```
 1 SetupP1       mov.b   #11011111b, &P1DIR  ;enable the P1.5 pin for input
 2               mov.b   #11111111b, &P2DIR  ;p2 is the output port
 3               mov.b   #00000100b, &P2OUT  ;make all outputs zero
 4               mov.b   #00000000b, &P1OUT
 5
 6 wait          mov.w   #50000, R6
 7 decR          dec.w   R6
 8               jnz decR
 9
10 Mainloop1     bit.b   #00100000b, &P1IN     ;check if the button is pressed
11               jz      Mainloop1             ;if not pressed go to Mainloop1
12               xor.b   #00001100b, &P2OUT
13
```

```
14 loo2         bit.b   #00100000b, &P1IN
15              jnz loo2
16              jmp wait
```

Listing 2: Part2

Line 1-2: We chose only P1.5 as input pin. All of the other pins can be used as outputs.

Line 3-4: We initialized P2.2 led as on and all of the other pins as off.

Line 6: This line is named as *wait* to be able to return to here later. And we assigned a word to R6.

Line 7-8: We decrement the word in R6 until it is equal to 0.

Line 10-11: Until P1.5 is pressed we repeatedly check it.

Line 12: When pressed, the 2nd and 3rd pins of P2 are XORed with 1.

Line 14-15: If the button is still pressed (hold down), the program does nothing (stays in *loo2*).

Line 16: If it is released, program goes back to the *Mainloop1*.

2.2.3 THIRD PART

The purpose of the last part is counting how many times P2.1 button is pressed. It needs to be 4-bit so after it counts 15 (decimal) it goes back to 0.

```
1              .data                    ; Data section
2  counter      .byte 0x00              ; 4-bit variable to store the count
3
4  SetupP1      mov.b   #00000000b, &P2SEL
5               mov.b   #11111101b, &P2DIR
6               mov.b   #11111111b, &P1DIR
7               mov.b   #00000000b, &P1OUT
8               mov.b   #00000010b, &P2OUT
9
10 Mainloop1    bit.b   #00000010b, &P2IN   ;Check if P2.1 is pressed
11              jnz     Mainloop1           ;If not pressed, stay in loop
12
13 ; Increment Counter
14              mov.b   counter, R5       ;Counter value into R5
15              inc.b   R5                ;Increment R5
16              cmp.b   #16, R5           ;Check if R5=16
17              jne     UpdateCounter
18              mov.b   #0, R5            ;If 16, reset R5 to 0
19
20 UpdateCounter
21              mov.b   R5, counter
```

```
22              mov.b    counter, &P1OUT
23
24 Wait         bit.b    #00000010b, &P2IN
25              jz       Wait
26              jmp      Mainloop1          ;Go back to main loop
```

Listing 3: Part3

Line 1-2: We initialize 4-bit variable to store the count in memory.

Line 4: We initialized every P2 select to 0 because the laptop we used made P2's 6th and 7th bit 0.

Line 5-6: We chose P2.1 as input pin, and all the other other pins as output pins.

Line 7-8: P2.1 led is set as on, and other leds are set as off.

Line 10-11: We check if P2.1 is pressed.

Line 14: When it is pressed, we move *counter* variable to R5.

Line 15: Increment R5.

Line 16-17-18: If R5 does not equal to 16, directly go to *updateCounter*. If it is equal to 16, make R5 0. Then, go to *updateCounter*.

Line 20: This function is labeled as *updateCounter*.

Line 21-22: Move R5 into counter. Then, move counter to the output of P1.

Line 24-25-26: Wait here until P2.1 is pressed. When pressed go to the *Mainloop1*.

3 RESULTS [15 points]

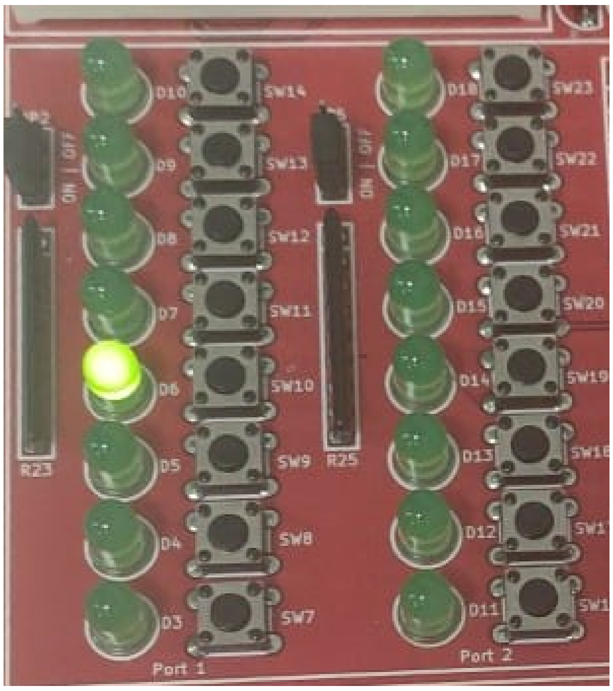


Figure 1: Part 1

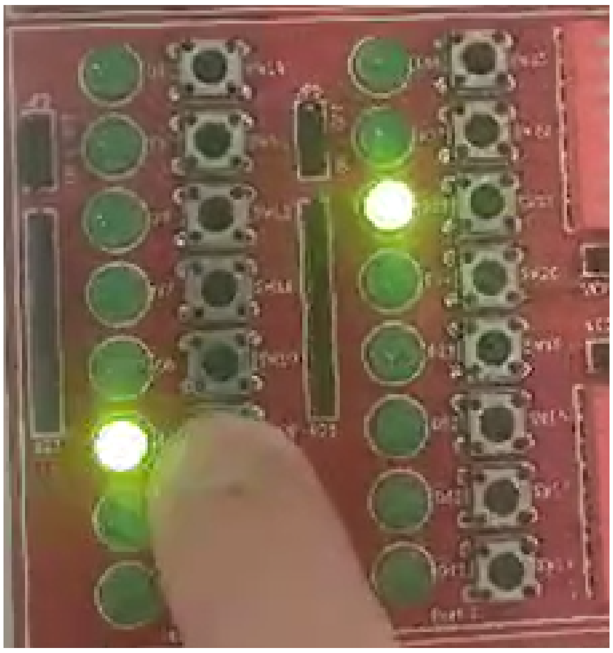
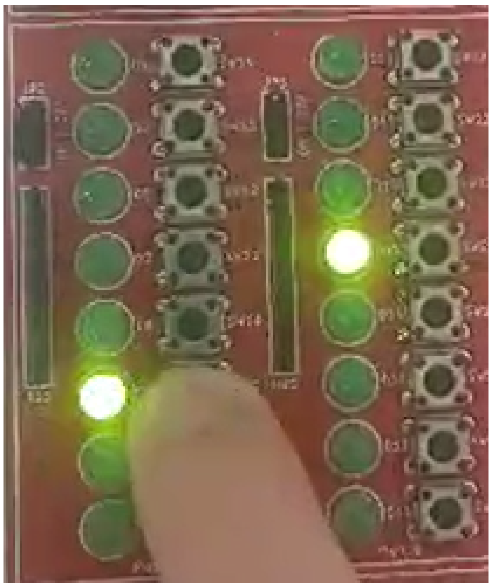
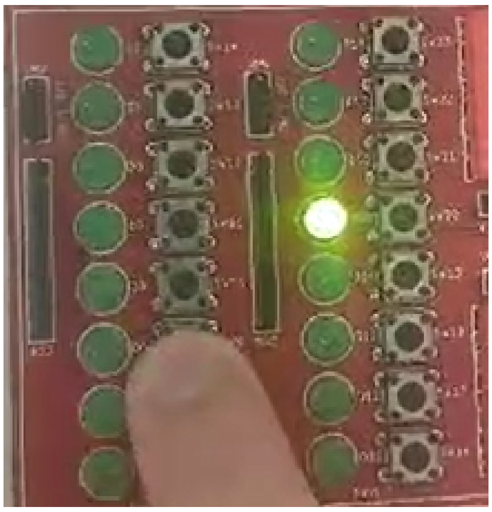


Figure 2: Part 2 - Led 2 On



(a) Hold on Led 3



(b) Do not change

Figure 4: Part 2

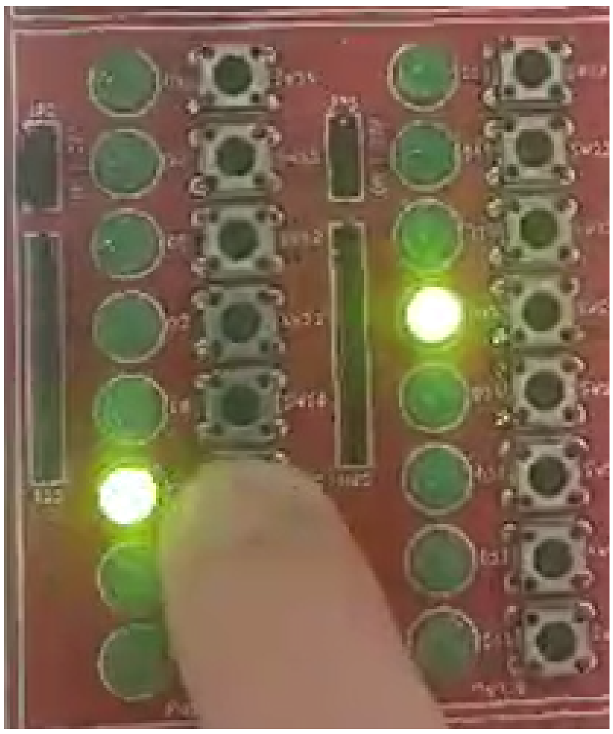


Figure 3: Part 2 - Led 3 On

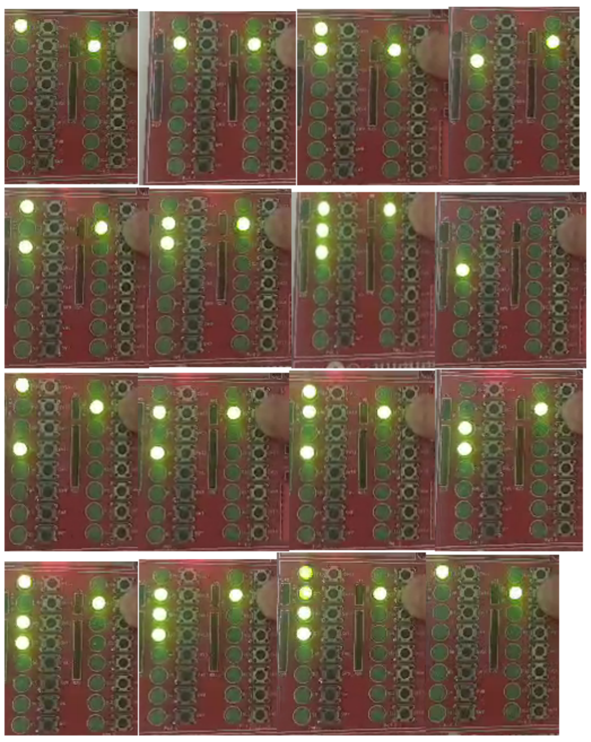


Figure 5: Part 3 results

4 DISCUSSION [25 points]

In the first section, we had to develop an assembly program that, when a button was pressed, activated an LED. We learnt about the distinction between the JMP and JNZ commands from this:

- JMP: An unconditional jump that, under any conditions, gives control to the designated label.
- JNZ: A conditional jump that permits branching according to button state and only happens if the zero flag is not set.

The correct GPIO layout was shown by setting P1.4 as an output and P2.4 as an input. As expected, the LED lighted, and the infinite loop was able to keep its state.

In the second section, we created a program that used button P1.5 to switch between two LEDs on P2.2 and P2.3. In order to avoid fast state changes when holding the button, we addressed switch debouncing. As planned, the debouncing approach ensured constant toggle activity between the LEDs. Our understanding of managing mutual exclusion and initialising LED states was improved by this part. modifying particular LED states without changing others by using logical commands like BIS(bit set) and BIC(bit clear). As a result, the LEDs' states changed easily and regularly as expected.

In the third part, we made a counter for button presses (P2.1), putting away the check in a memory variable rather than a register. Defining and initializing the variable in the data segment of the program allowed us to reset it at 16. The check shown accurately on Port 1, demonstrating: How to manage memory and utilize 4-bit variables in assembly. Avoiding unintentional checking by ensuring the button was completely discharged before counting another press. This segment given important knowledge into advanced GPIO programming, enabling precise and reliable input handling.

5 CONCLUSION [10 points]

- With this experiment, we got familiarized with the usage of LEDs on MSP430 chip and wrote some code to figure out the button usage.
- We learned how to implement a counter to differentiate between press and hold and learned how to store values in memory .
- At first, we struggled to see our results due to a misunderstanding with #...*b* and #... usage. But after trying each we were able to figure out the difference.
  - #...*b*: this is a binary value the number written #101*b* means 5 in decimal.
  - #...: this is a immediate decimal value the number written #101 means 101 in decimal.

REFERENCES

[1] Microcomputer Lab. Micro_experiment_2. *Lab Booklet*, 2024.