Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 2

Experiment Date : 16.10.2017

Group Number : Monday - 8

Group Members :

ID	Name	Surname
150150114	EGE	APAK
150140066	ECEM	GÜLDÖŞÜREN
150150701	YUNUS	GÜNGÖR

Laboratory Assistant : Ahmet ARIŞ

1. INTRODUCTION

In this experiment we gained more experience with MSP430 Education Board and its assembly language. The main aim of this experiment was to introduce ourselves to general purpose IO (GPIO).

In the first experiment, we used both Port#1 and Port#2 in order to toggle a LED on Port#1.2 by using the push button on Port#2.3.

In the second part of the experiment, we have implemented a counter and incremented it whenever the push button on Port#2.3 is pressed and displayed the current counter in binary on all the LEDs present on Port#1.

In the third and last part of the experiment, we added a reset button on the counter implemented in Part 2 of the experiment and reset the counter back to zero whenever the reset button is pressed.

2. EXPERIMENT

2.1. PART 1

In this part, we toggled the LED on Port#1.2 when the button on Port#2.3 is pressed. The code we wrote can be seen in Picture 2.1.

```
26
27 START     mov.b #00000100b, &P1DIR
28           mov.b #11111011b, &P2DIR
29           mov.b #000h, R4
30           mov.b #000h, &P1OUT
31           mov.b #000h, &P2OUT
32           jmp CHECK
33
34
35 TOGGLE    xor.b #00000100b, &P1OUT
36
37 CHECK     cmp.b &P2IN, R4
38           mov.b &P2IN, R4
39           jl TOGGLE
40           jmp CHECK
41
42
```

Picture 2.1: Assembly code of Part 1

First of all, we setup the directions for each port, defining all ports on Port#1 except the 2nd is to be input port (Line #27), so we would be working only with the LED we desire and all the only the Port#2.3 to be input (Line #28) so other buttons would not interfere with our experiment.

After that, we decided to use R_4 to hold the value of last button state and start it with non-pressed state (Line #29) so that LED would toggle only when previous input was zero and current input is one (the button is pressed right now). Possible states can be seen on Table 2.1.

R4	P2.3	State
0	**0**	Button stays at default state
0	**1**	Button press is registered
1	**0**	Button is no longer pressed
1	**1**	Button is being held down

Table 2.1: Button states (Shaded state is the desired one)

On the lines #30 & #31, we set the outputs of both ports to zero in order to turn off all LEDs, which might have been left on in previous runs.

Then we jump to the loop (Line #32) to skip toggling the LED on after the setup.

In the TOGGLE part (Line #35), we **xor.b** the content of &P1OUT with #00000100b, which saves the result in &P1OUT, in order to toggle the LED on Port#1.2. If the LED on Port#1.2 were on, XORing with #00000100b would make 3rd least significant bit to zero and turn off the LED and would make it one if it were not one to begin with.

In the loop, we first compare &P2IN with R4 (Line #37) which produces the result on Table 2.2. The result matches with our desired behavior from Table 2.1. After comparing the register and input, we move the input byte to R4 in order to save the last state of the button (Line #38)

R4	P2.3	Result
0	**0**	0
0	**1**	-1
1	**0**	1
1	**1**	0

Table 2.2: Result of cmp.b operation. (CMP.B src dst => dst - src)

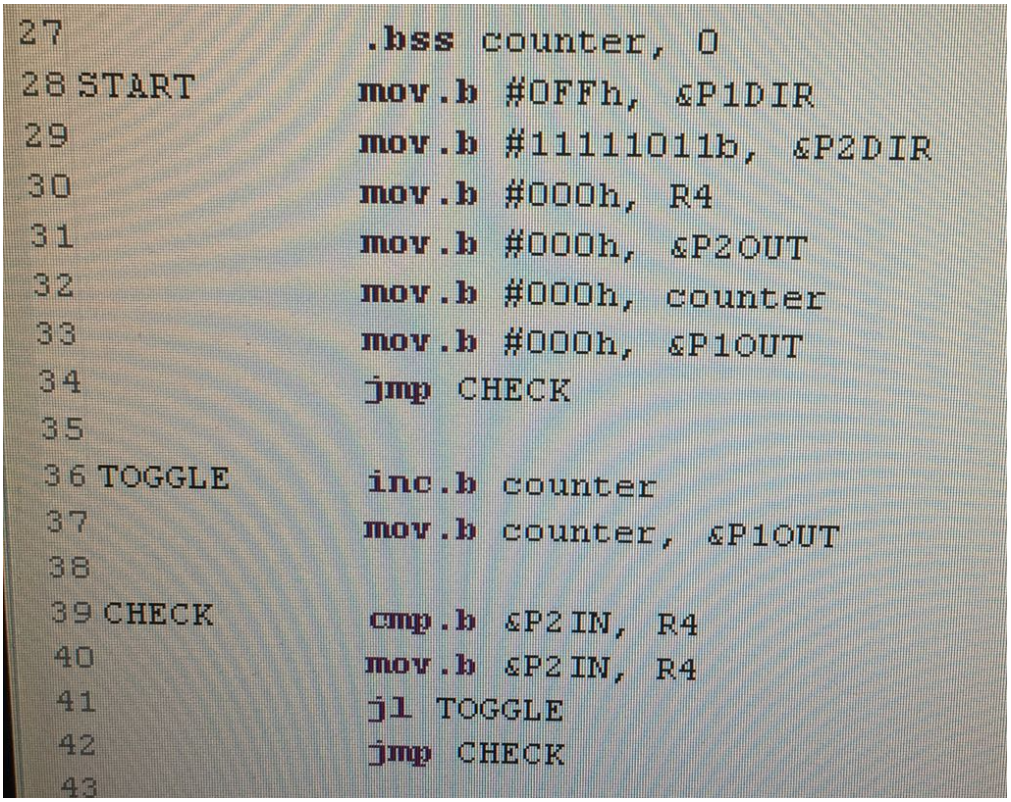
Since **mov.b** command does NOT affect the status bits, we can safely do a conditional jump on Line #39. In this line, we jump to TOGGLE section if the result of compare is less than zero, which only happens in our desired state.

If conditional jump to TOGGLE doesn't happen, we jump back to CHECK (Line #40) to keep checking the input.

In this part of the experiment, our inputs sometimes didn't register (i.e. LED didn't toggle when we pressed the button). After some debugging, we came to conclusion that the change of input value was happening after comparing the R4 and &P2IN but before moving &P2IN to R4, so the state was jumping from not pressed to being held without going through "first press" state.

2.2. PART 2

In this part of the experiment, we have modified the code from the previous part and implemented a counting mechanism by using a variable to store the current value of the counter and displayed the value of the counter in binary form by using all LEDs on Port#1. The code of this part can be seen on Picture 2.2.



```
27                .bss counter, 0
28 START          mov.b #0FFh, &P1DIR
29                mov.b #11111011b, &P2DIR
30                mov.b #000h, R4
31                mov.b #000h, &P2OUT
32                mov.b #000h, counter
33                mov.b #000h, &P1OUT
34                jmp CHECK
35
36 TOGGLE         inc.b counter
37                mov.b counter, &P1OUT
38
39 CHECK          cmp.b &P2IN, R4
40                mov.b &P2IN, R4
41                jl TOGGLE
42                jmp CHECK
43
```

Picture 2.2: Assembly code of Part 2

Before doing anything else, we declared a variable in memory named *counter* by using **.bss** directive.

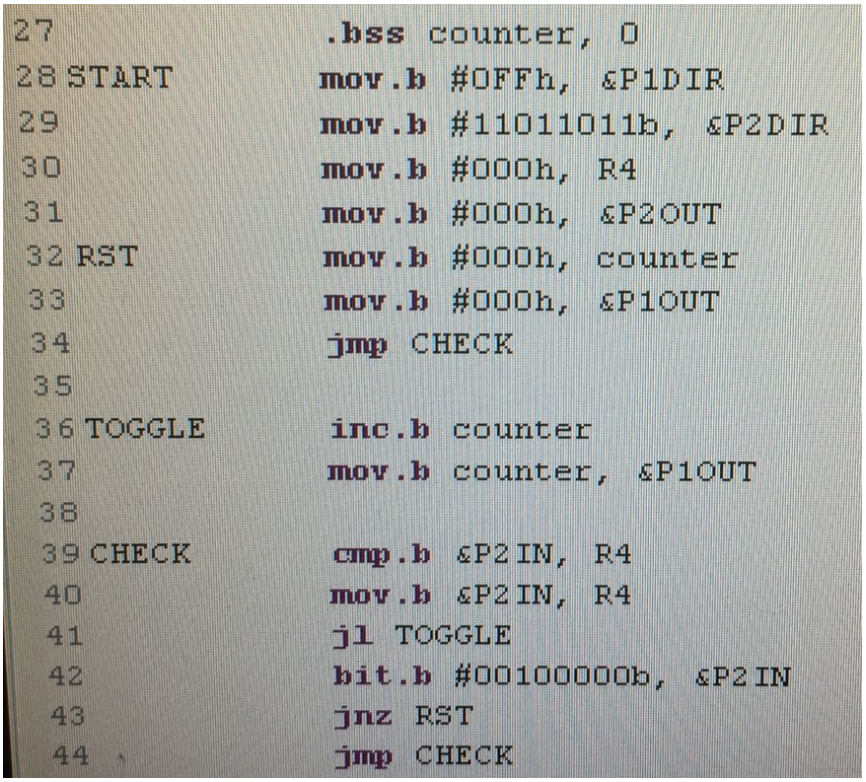
Most of the setup part is the same as before except one line: We assign zero to variable *counter* on Line #32.

On the TOGGLE part, instead of doing **xor.b** operation with &P1OUT and toggling it, we first increment the *counter* (Line #36) and move the value of *counter* to &P1OUT (Line #37) to display it in binary form.

We have faced with the exact same anomaly occurred on first part and haven't found anything unusual other than that.

2.3. PART 3

In the last part of the experiment, we improved the counter we have implemented in the 2nd part and added a reset button to it, which resets the value of counter back to zero. The code we wrote to implement this functionality can be seen in Picture 2.3.



```
27              .bss counter, 0
28 START        mov.b #0FFh, &P1DIR
29              mov.b #11011011b, &P2DIR
30              mov.b #000h, R4
31              mov.b #000h, &P2OUT
32 RST          mov.b #000h, counter
33              mov.b #000h, &P1OUT
34              jmp CHECK
35
36 TOGGLE       inc.b counter
37              mov.b counter, &P1OUT
38
39 CHECK        cmp.b &P2IN, R4
40              mov.b &P2IN, R4
41              jl TOGGLE
42              bit.b #00100000b, &P2IN
43              jnz RST
44              jmp CHECK
```

Picture 2.3: Assembly code of Part 3

Variable declaration and setup parts are same as before, except we have added **RST** label to the line we assign value zero to variable *counter* and reset all the LEDs on Port#1 (Lines #32-33).

The part in which we increment the value of *counter* stays exactly same as before.

In the CHECK loop, we have added checking against the reset button. We have done this by using **bit.b** command to check whether the value of reset button on Port#2 is one or zero (Line #42). After this test, we jump conditionally to RST label (Line #43) if the Z (Zero) flag is not set (which indicates that the bit we have tested on Line #42 is one and the reset button is pressed)

The same issue of not registering the inputs occurred much less compared to the previous parts. We believe that since there are more operations performed in every iteration of loop, the cause we have described in part 1 has a lower chance to occur.

3. CONCLUSION

In this experiment, we have learned about GPIO operations of MSP430 Educational Board and performed some simple IO operations. Aside from that, we have also learned how to declare, read and update variables and how they can be used instead of only using general purpose registers.

As noted at the end of every part, we have faced with an issue of inputs being not registered. We believe that this situation happens when the input values change before moving the &P2IN value to R4 but after comparing &P2IN with R4, which causes a state (which is the one the program decides to perform toggle/increment operations) to be skipped, causing the desired operations to be not performed. As number of operations per iteration of check loop increases, this issue occurs less and less, which strengthens our diagnosis of the issue. However, this anomaly hasn't caused any significant drawbacks during the testing phases.