Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 5

Experiment Date : 25.11.2016

Group Number : Friday - 3

Group Members :

ID	Name	Surname
150140142	Gamze	Akyol
150130136	Mehmet Barış	Yaman
150130109	Güllü	Katık

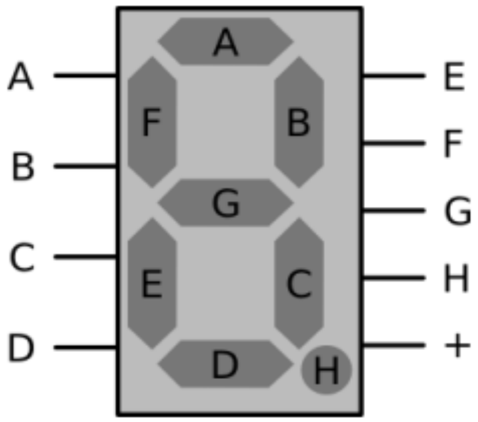
Laboratory Assistant : Enes Albay

1 INTRODUCTION

In this experiment, we learned showing decimal integers on 7-segment display, and initializing interrupt.

2 EXPERIMENT

Before the experiment, we filled in the table below.



7-segment display

Integer	H	G	F	E	D	C	B	A
0	0	0	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0
2	0	1	0	1	1	0	1	1
3	0	1	0	0	1	1	1	1
4	0	1	1	0	0	1	1	0
5	0	1	1	0	1	1	0	1
6	0	1	1	1	1	1	0	1
7	0	0	0	0	0	1	1	1
8	0	1	1	1	1	1	1	1
9	0	1	1	0	1	1	1	1

Input for decimals

2.1 PART 1

In the first part of the experiment, we created an array that included the integers in the table. We incremented R7(pointer that points to array) until it's equal to last element of array, and displayed value of R7 on port 1 and 7-segment display.

The code for part 1:

```
Setup       mov     #array, R7      ;pointer
            bis.b   #11111111b, &P1DIR  ;Initializing P1 as output


            mov.w lastElement, R10      ;last element of array load to R10


main        mov.b   @R7, &P1OUT         ;value pointed by R7 display P1OUT
            call    #Delay          ;call the Delay subroutine
            inc     R7              ;increment R7, so R7 points the next element of array
            cmp.    #lastElement, R7 ;compare R7 and last element of array
            jne     main    ;if R7 not equal to last element brunch to main
            jmp finish      ;else brunch to finish
```

```
finish          nop


;use Delay subroutine to observe change in output
Delay           mov.w #0Ah, R14
L2              mov.w   #07A00h,R15
L1              dec.w   R15
                jnz     L1
                dec.w   R14
                jnz     L2
                ret
; Integer to 7- segment array
array .byte 00111111b, 00000110b, 01011011b,01001111b, 01100110b, 01101101b, 01111101b, 00000111b, 01111111b, 01100111b ; contains 10 values
lastElement
```

2.2 PART 2

In this part, we used the interrupt subroutine given in booklet, in addition to first part of experiment. First, we created a loop that display value pointed by R6, if R7 equal to 1 brunch “revMain”, else increment R6. When R6 is equal to lastElement brunch to “initialize” and first element of array load to R6 for provide to endless loop. “revMain” and “revInit” work similarly “main” and “initiliaze”. Difference is decrementing R6. Thus the loop counts 9 to 0 instead of 0 to 9. We use R7 for decide to direction of counting. If R7 equals to 0 the program count upwards, if R7 equal to 1 the program counts downward. When we push the button interrupt begins running and reverse R7. So count direction changes.

The code for part 2:

```
init_INT        bis.b   #040h ,&P2IE ; enable interrupt at P2 .6
                and.b   #0BFh ,&P2SEL ; set 0 P2SEL .6
                and.b   #0BFh ,&P2SEL2 ; set 0 P2SEL2 .6

                bis.b   #040h ,&P2IES ; high -to -low interrupt mode
                clr             &P2IFG ; clear the flag
                eint                            ; enable interrupts


Setup           mov             #array, R6
                bis.b           #00000000b, &P2DIR ;P2 setup as input
                bis.b           #11111111b, &P1DIR
```

```
                mov.b       #00h, R7 ;initially 0 load to R7
initialize      mov         #array, R6      ;first element of array load to R6
main            mov.b       @R6, &P1OUT
                call        #Delay ;call Delay subroutine
                bit.b       #01000000b, &P2IN ;check 6th bit of port 2
                jnz         ISR ;if it's not equal to 0 brunch ISR(interrupt)
                cmp.b       #00000001b, R7
                jeq         revMain ;if R7 equal to 1 brunch to revMain
                inc         R6 ;increment R6
                cmp.w  #    lastElement, R6
                jne         main ;if R6 isn't equal to lastElement brunch to main
                jmp         initialize ;else brunch to initialize

revInit         mov         #lastElement, R6 ; last element of array load to R6
revMain         dec         R6 ;decrement R6, so R6 points the previous element of array
                mov.b       @R6, &P1OUT
                call        #Delay ;call Delay subroutine
                bit.b       #01000000b, &P2IN ;check 6th bit of port 2
                jnz         ISR ;if it's not equal to 0 brunch ISR(interrupt)
                cmp.b       #00000000b, R7
                jeq         main if R7 equal to 0 brunch to main
                cmp.w       #array, R6
                jne         revMain ;if R6 isn't equal to first element of array brunch to revMain
                jmp         revInit ;else brunch to initialize

ISR             dint                ; disable interrupts
                xor.b       #00000001b, R7 ;reverse last bit of R7
                clr         &P2IFG ; clear the flag
                eint                ; enable interrupts
                reti                ; return from ISR
```

```
finish      nop

Delay       mov.w  #0Ah ,R14       ; Delay to R14
L2          mov.w  #07A00h , R15
L1          dec.w  R15             ; Decrement R15
            jnz    L1
            dec.w  R14
            jnz    L2
            ret

; Integer to 7- segment array
array .byte 00111111b, 00000110b, 01011011b,01001111b, 01100110b, 01101101b, 01111101b, 00000111b, 01111111b, 01100111b ; contains 10 values
lastElement
```

3 CONCLUSION

In the second part of experiment, we had to change the the pin that set input in given code, because it was not working.

We have learned to use 7-segment display to show decimal integers from first part of experiment. We have learned to initialize interrupt and operation principles of interrupt.

Busy waiting is a technique that is used for adjusting process times and shared variables between them. According to the pseudocode of the mechanism, there is a boolean value which is checked by the processors before processes enter the critical section. Busy waiting can be used to generate a time delay for the systems that are low level or don't have a mechanism that has a timer adjustment but not appropriate for process times. On the other hand, busy waiting should not be preferred, since it makes processes time to be consumed on useless activities.

Interrupt mechanism is an another technique that can interrupt the process job temporarily. If this mechanism is designed appropriately, delays are reduced for any events. But, this mechanism can also distort the process activities, generally when there is a conflict that prevents the process to return the job after the interrupt. On that times, process behaviour changes and that causes normal jobs are leaved. This mechanism is usually used for high level, carefully designed systems.