Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 5

Experiment Date : 13.11.2017

Group Number : Friday - 17

Group Members :

ID	Name	Surname
150150302	İrem Nur	Demirtaş
150160706	Merve Elif	Demirtaş
150140719	Cemal	Türkoğlu
150140722	İbrahim	Dolapçı

Laboratory Assistant : Ahmet Arış

1 INTRODUCTION

In this experiment, the 7-segment display and the interrupt subroutine were utilized to implement different counters.

2 EXPERIMENT

2.1 PART 1

In this part, the aim was to write the assembly code for a counter that counts the numbers 0 through 9 and display them on the 7-segment display. Firstly, the table provided in the laboratory sheet was filled to determine which values should be supplied to the input pins of the 7-segment display to make sure the correct segments light up in the pattern of its decimal integer when a number is printed on the display. The values can be seen in Table 1. below.

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	0	1	1	1	1	1

Table 1. Inputs for decimals

Then, these values were stored in memory to be used as an array. Accordingly, the following lines seen in Code Snippet 1. were added to the assembly file above the .text label. Here, array points at the first element of the array, and the lastElement, as the name implies, points to the last element.

```
                    .data
array               .byte         00111111b, 00000110b, 01011011b, 0100111b,
01100110b, 01101101b, 01111101b, 00000111b, 01111111b, 01011111b
lastElement
```

Code Snippet 1. Array of numbers

The counter was implemented as seen below in Code Snippet 2. Setup turns P1 port on so that the numbers can be displayed on the display and sets the least significant bit of P2 port to pick the leftmost one of the 4 available displays. Mainloop prints the values to the display by XORing, the input bits of the ports. Loop one sets the pointers for the first and last elements of the array by copying the addresses of the first and last elements to registers R4 and R5 respectively. After Loop executes, the program passes on to Loop2 where the address of the first element, which is held in R4, is compared to those of the last element. If the last element has not yet been reached, the value R4 points at is forwarded to P1OUT using indexed addressing, so that it is displayed. Then, R4 is incremented so that it points to the last element. Afterwards the function labeled Delay is called which causes a few seconds of delay so that the value at the display can be observed. Then, the program jumps back to Loop2 until the last element is reached and 9 is displayed, at which point the program jumps back to Loop, resets all the pointers and starts counting from 0 again. This goes on in an infinite loop. The flowchart for the code can be seen in Figure 1.

```
Setup     bis.b     #0ffh, &P1DIR      ; Activate all the bits at P1 port
```

```
            bis.b   #001h, &P2DIR        ; Activate the last bit of P2 port

Mainloop    xor.b   #0ffh, &P1OUT        ; Print to P1
            xor.b   #0ffh, &P2OUT        ; Toggle the last bit in P2


Loop        mov.w   #array, R4           ; Make R4 point to the first element of the array
            mov.w   #lastElement, R5     ; Make R5 point to the last element of the array
Loop2       cmp     R4, R5               ; Compare R4 and R5 to make sure end is not reached
            jeq     Loop                 ; If R4 and R5 are equal and the end is reached jump to Loop
            mov.b   0(R4), &P1OUT        ; Else, forward what R4 points at to P1OUT
            inc.w   R4                   ; Make R4 point to the next element
            call    #Delay               ; Call the delay function to observe the changes
            jmp     Loop2                ; Jump to Loop2 unconditionally
; The following is the code provided in the laboratory sheet
Delay       mov.w   #0ah, R14            ; Hold counter value in R14

L2          mov.w   #07a00h, R15         ; Hold counter value in R15
L1          dec.w   R15                  ; Decrement R15
            jnz     L1                   ; Jump to L1 if R15 is not equal 0
            dec.w   R14                  ; Else, decrement R14
            Jnz     L2                   ; Jump to L2 if L2 is not zero
            ret                          ; Return to where the function was called
```

Code Snippet 2. Array of numbers

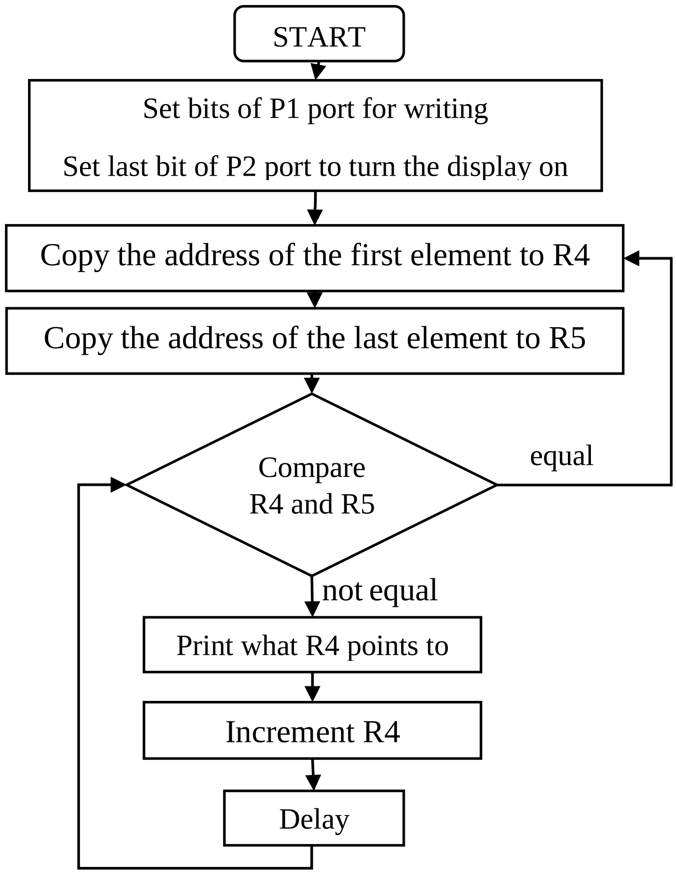

Figure 1. Flowchart for Part 1

2.2 PART 2

In this part, the same array was used to count either even numbers or odd numbers. An interrupt was triggered to switch between the even counter and the odd counter. Code Snippet 1. was also used in this part to store the values corresponding to the digits in the memory to be accessed as an array.

The final code written for this part can be seen in Code Snippet 3. below. The lines after labels init_INT, and ISR were copied from the laboratory sheet to implement the interrupt subroutine. What the code below does is initializing the flag register R10 by clearing and setting the display for printing in Mainloop; setting first even, first odd and last even element pointers and a pointer to the first memory location after the last element in Loop; checking R10, which is declared as the flag that is toggled in the interrupt subroutine, in Loop2, and depending on its value, running either the odd or the even counter and printing the values to the display while calling the function Delay to make sure the changes remain on the display long enough to be observed in EVEN and ODD; toggling the flag R10 when the program enters the interrupt subroutine, ISR, so that it switches to the other counter. Interrupts are triggered by pushing the button P2.6. The program initially starts counting even numbers and unless P2.6 is pressed, it keeps counting even numbers over and over again and returns back to 0 once it reaches 8, same goes for odd numbers as well. However, in the odd case, the next address after the last element's address is checked to make sure that all odd numbers are printed. If it were to check the last element without changing the jump condition, it would not have printed 9. When P2.6 is pressed, the program ceases execution, goes into the interrupt subroutine where it toggles the flag and clears the interrupt flag P2IFG to escape the interrupt routine. Then, the flag is checked again in Loop2 and the other counter starts. The program keeps on running in an infinite loop.

```
Setup       bis.b   #0ffh, &P1DIR        ; Activate all the bits at P1 port
            bis.b   #001h, &P2DIR        ; Activate the last bit of P2 port


init_INT    bis.b   #040h, &P2IE
            and.b   #0bfh, &P2SEL
            and.b   #0bfh, &P2SEL2

            bis.b   #040h, &P2IES
            clr     &P2IFG
            eint

Mainloop    xor.b   #0ffh, &P1OUT        ; Print to P1
            xor.b   #0ffh, &P2OUT        ; Toggle the last bit in P2
            clr     R0


Loop        mov.w   #array, R4           ; Make R4 even pointer
            mov.w   R4, R5               ; Make R5 point to the first element
            inc.w   R5                   ; Increment R5 so that it becomes odd pointer
            mov.w   #lastElement, R6     ; Make R6 point to the last even element
            mov.w   R6, R7               ; Make R7 point to the last even element
            inc.w   R7                   ; Make R7 point to the next place after the array

Loop2       cmp     #000h, R10           ; Check if toggle flag is 0
            jne     ODD                  ; If it is, count even numbers, else count odd ones

EVEN        mov.b   0(R4), &P1OUT        ; Print what R4 points to the display
```

```
            inc.w   R4              ; Increment R4 twice
            inc.w   R4              ; So that it points to the next even number
            call    #Delay          ; Call delay function to observe the chance
            cmp     R4, R6          ; Check if the last even element is reached
            jeq     Loop            ; If it is reached, jump to Loop
            jmp     Loop2           ; Else, jump to Loop2

ODD         mov.b   0(R5), &P1OUT   ; Print what R5 points to the display
            inc.w   R5              ; Increment R5 twice
            inc.w   R5              ; So that it points to the next odd number
            call    #Delay          ; Call delay function
            cmp     R5, R7          ; Check if the last odd element is reached
            jeq     Loop            ; If it is reached, jump to loop
            jmp     Loop2           ; Else, jump to Loop2


Delay       mov.w   #0ah, R14       ; Hold counter value in R14

L2          mov.w   #07a00h, R15    ; Hold counter value in R15
L1          dec.w   R15             ; Decrement R15
            jnz     L1              ; Jump to L1 if R15 is not equal 0
            dec.w   R14             ; Else, decrement R14
            Jnz     L2              ; Jump to L2 if L2 is not zero
            ret                     ; Return to where the function was called


ISR         dint                    ; Disable interrupts

            xor.b   #001h, R10      ; Toggle flag
            clr     &P2IFG          ; Clear interrupt flag

            eint                    ; Enable interrupts
            reti                    ; Return interrupt
```

Code Snippet 3. Code for part 2

Flowchart of this code can be seen in Figure 2.

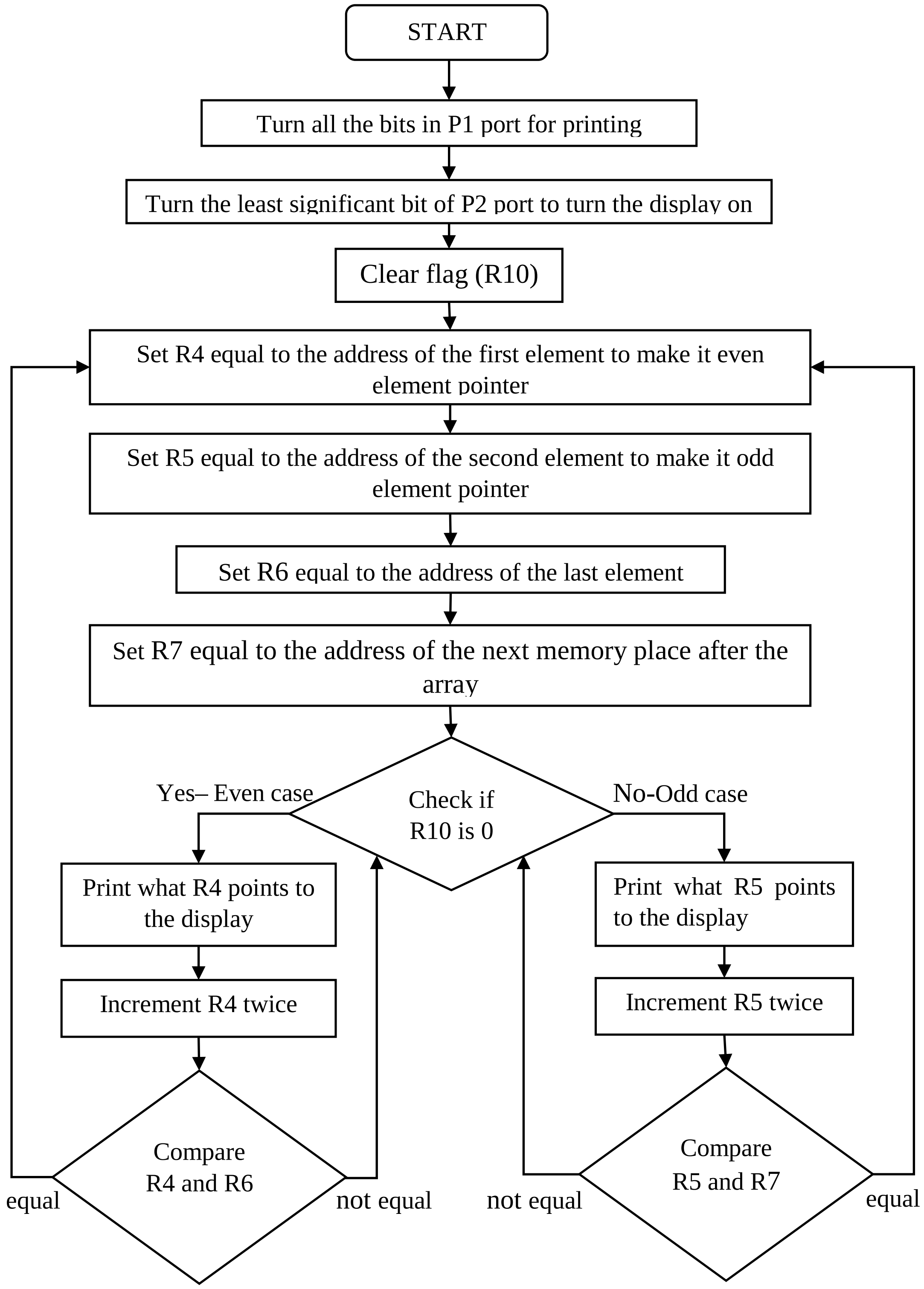

Figure 2. Flowchart for Part 2

3 COMPARISON OF BUSY WAITING AND INTERRUPT

Busy-waiting is constantly checking if a condition has changed in a loop taking action accordingly, while an interrupt notifies the processor when it is triggered, in our case with the press of a button, and the processor stops executing whatever program it was executing, saves the return place, runs the interrupt subroutine and goes back to where it left off. The benefit of interrupts against busy-waiting is not consuming the valuable processor time while running a loop to detect whether a condition has changed, since most of the time the condition is most likely to stay the same. Because any action that would have taken when the condition is satisfied in busy-waiting can be accomplished in the interrupt subroutine, in our case the ISR label that toggles the R10 register, the need for a loop is eliminated. The time that would have been spent in vain while executing the loop in busy waiting can be better utilized by using interrupts instead.

4 CONCLUSION

Overall, we did not have any serious problems, we just needed to play around a little bit to figure out how the interrupt trigger and interrupt subroutine worked. Using the 7-segment display was very easy as it was very similar to using the LEDs. The diagram that explained the connections was clear, precise and very helpful. This experiment helped us understand how to implement interrupt subroutines, how to set its trigger and how interrupts actually operate. Debugging the program was very explanatory in that regard.