Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 3

Experiment Date : 04.11.2016

Group Number : Friday - 3

Group Members :

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

Laboratory Assistant : Resul Tugay

1 INTRODUCTION

In this experiment, we have dealed with 2 algorithms named Encryption and Bubble Sort. We created the Encryption program according to the diagram in the Experiment Booklet in Part 1 and the Bubble Sort program according to the pseudocode, also given in the Experiment Booklet in Part 2.

2 EXPERIMENT

2.1 PART 1 - ENCRYPTION

Encryption was done according to a certain procedure given in the booklet. Example data and key was used in the program.

The program of this part is given below:

```
Setup       clr.b   &P1OUT                  ; clear the data in P1OUT
            bis.b   #11111111b, &P1DIR      ; P1 is assigned as output
Mainloop    mov.b   #10010011b, &P1OUT      ; data which will be encrypted is
                                            ; loaded into P1

            clr.b   R5                      ; clear the data inside the register R5
            bis.b   #10010011b, R5          ; set bits of R5 as the data which will
                                            ; be treated and encrypted
```

; Beginning of the first step of encryption ($a_7a_6a_5a_4\ a_3a_2a_1a_0 \rightarrow\ a_3a_2a_1a_0\ a_7a_6a_5a_4$)

```
            mov.b   #00001111b, R4
            and.b   R5, R4                  ; do "and" operation on R5 and R4,
                                            ; so the least significant 4 bits of R5
                                            ; are obtained only, and loaded in R4
                                            ; ((10010011).(00001111) = 00000011)

            clrc                            ; clear the carry bit
            rlc.b   R4                      ; rotate the bits of R4 to the left 4 times
            rlc.b   R4
            rlc.b   R4
            rlc.b   R4                      ; then R4 = 0011 0000
            clrc                            ; clear the carry bit before rotating the
                                            ; bits of R5 every time,
                                            ; unless redundant data may come from
                                            ; the carry bit to the MSB of R5.

            rrc.b   R5                      ; rotate the bits of R5 to the right 4 times
            clrc
            rrc.b   R5
            clrc
            rrc.b   R5
            clrc
            rrc.b   R5                      ; then R5 = 0000 1001
            bis.b   R4, R5                  ; set the bits of R4 to R5, do “or” operation,
                                            ; i.e.00110000 + 00001001 = 00111001

            mov.b   R5, &P1OUT              ; loading the data inside R5 to the output
```

; End of the first step of encryption

; Second step of encryption ($a_3 a_2 a_1 a_0\ a_7 a_6 a_5 a_4 \rightarrow\ a_2 a_3 a_0 a_1\ \ a_6 a_7 a_4 a_5$)

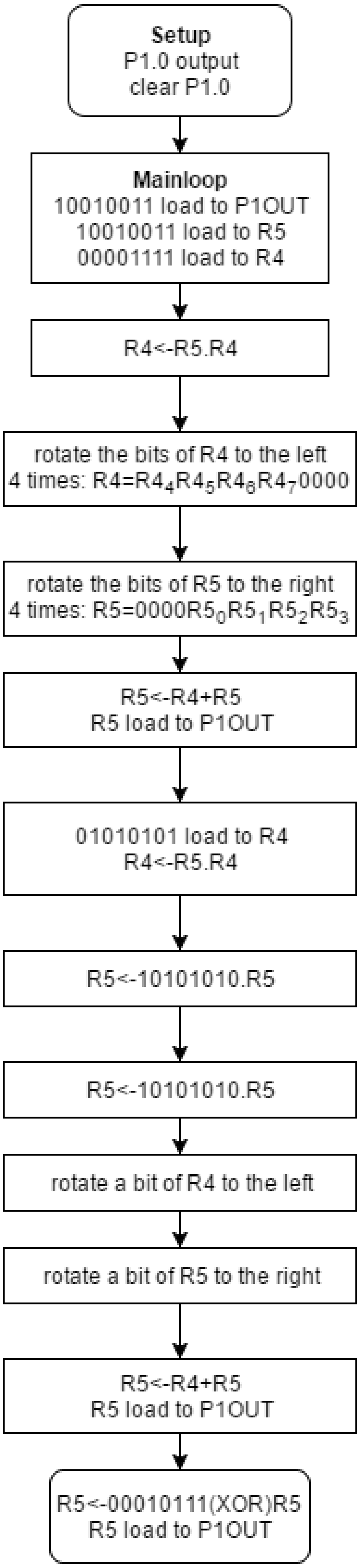
```
            mov.b   #01010101b, R4          ; load the data to R4
            and.b   R5, R4                  ; do “and” operation on R5 and R4, so
                                            ; R4 = 0011 1001 . 0101 0101 = 0001 0001
            and.b   #10101010b, R5          ; R5 = 1010 1010 . 0011 1001 = 0010 1000
            clrc
            rlc.b   R4                      ; rotate a bit of R4 to the left,  R4 = 0010 0010
            clrc
            rrc.b   R5                      ; rotate a bit of R5 to the right R5 = 0001 0100
            add.b   R4, R5                  ; add R4 to R5,
                                            ; 0010 0010 + 0001 0100 = 0011 0110
                                            ; the second step of encryption is done
            mov.b   R5, &P1OUT              ; load the data inside R5 to the output
```

; The last step of encryption: doing “xor” operation on the data with a given key

```
            xor.b   #00010111b, R5          ; R5 = 0001 0111 ‘xor’ 0011 0110
                                            ; = 0010 0001
            mov.b   R5, &P1OUT              ; load the data inside R5 to the output
```

If both three operations are done on the last data in the reverse order, the initial data can be obtained.

The flowchart of the program:



2.2 PART 2 – BUBBLE SORT

In this part of the experiment, bubble sort algorithm was applied on an integer array according to given algorithm in the booklet.

The program is given below:

```
Setup       mov     #array, R5          ; Load the array to the register R5, (R5 = {5, 1, 4})
            mov     #resultArray, R10   ; Load the result array index to R10
            mov.w   R10, R15            ; Load the initial result array which is
                                        ; in R10 to a temporary register R15

            call    #load               ; Call the "load" subroutine 3 times to
            call    #load               ; transfer the data inside the array
            call    #load               ; to the result array (also index of
                                        ; arrays will show the last element of
                                        ; the arrays) (R5 = R10 = {5, 1, 4})

            mov.w   R10, R14            ; Load the changed result array to another temporary
                                        ; register R14  (R10 = R14 = {5, 1, 4})

Mainloop    mov.w   0(R10), R6          ; Load the last element of the result array to R6
            dec     R10                 ; Decrease the index of the result array
            mov.w   0(R10), R7          ; Load the previous element of the result array to R7
            cmp     R7, R6              ; Compare the last element with the previous
            jn      exchange            ; If the last one is less than the previous one, branch to
                                        ; "exchange", since the indexes have to be exchanged

Loopend     cmp     R10, R15            ; If it is 2nd FOR loop of the pseudocode is done
            jne     nextloop            ; If it is not the beginning of the result array, go to
                                        ; "nextloop" that means a loop of the first for given in
                                        ; the algorithm pseudocode is finished.

            jmp     Mainloop            ; Else, only comparison is finished, 2nd FOR loop is
                                        ; ended so branch to "Mainloop"

exchange    mov.w   R6, 0(R10)          ; Exchange operation
            inc     R10
            mov.w   R7, 0(R10)
            dec     R10
            jmp     Loopend

; Next loop of the first for in the pseudocode

nextloop    inc     R15                 ; R15 increased since next iteration has to be held
            mov.w   R14, R10            ; R10 is arranged to point the first index of the array
            cmp     R15, R10            ; Comparison before the 2nd for loop iteration
            jne     finish              ; That means there is no need for the loop since 2 FOR
                                        ; loop pointers point same
            jmp     Mainloop            ; If not, 2nd FOR loop is necessary
```

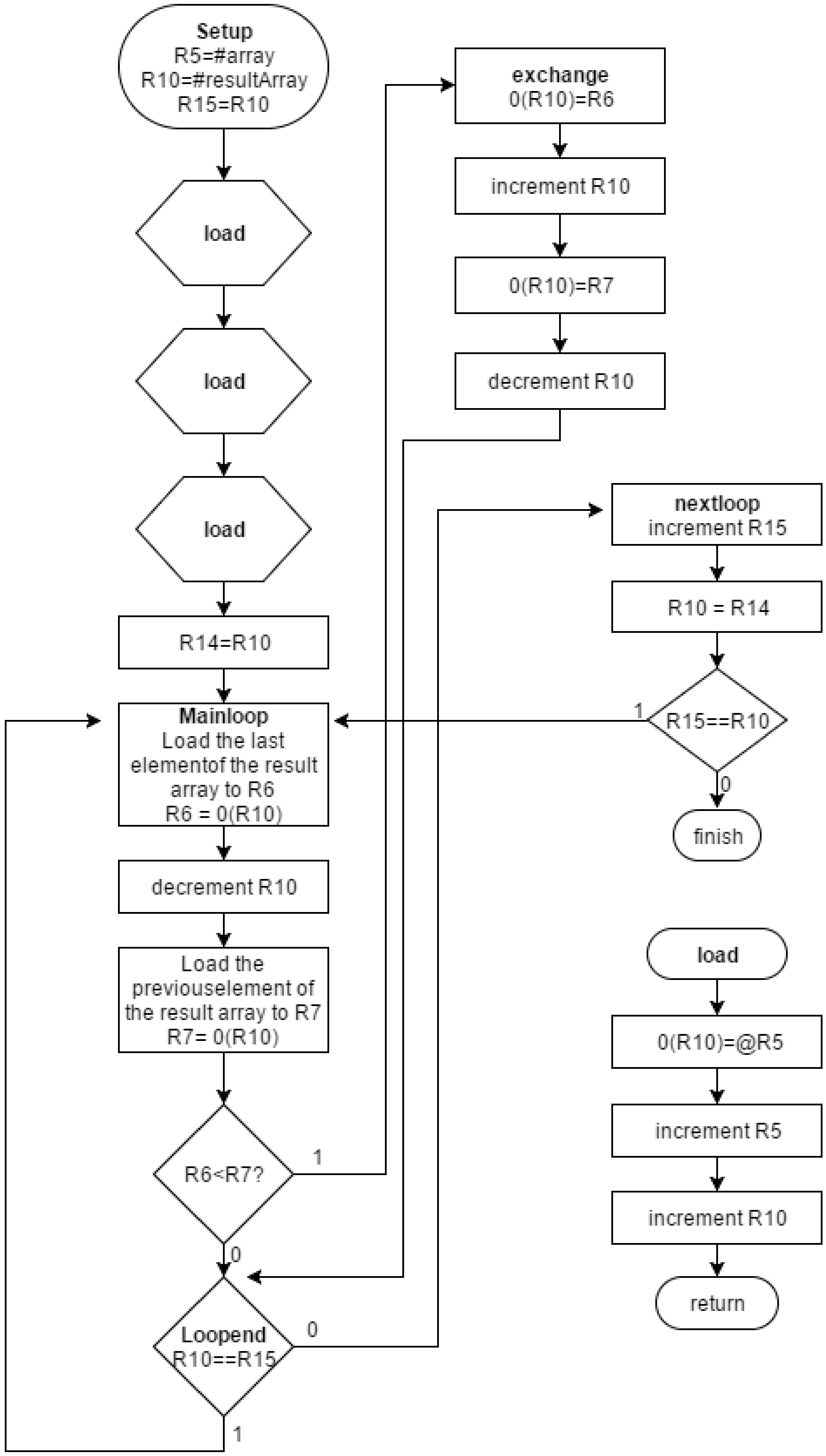
```
load        mov.w  @R5, 0(R10)        ; Load operation (R5 index to R10 index)
            inc    R5
            inc    R10                ; Both array pointers are incremented for the next
            ret                       ; assignment of indexes

; Integer array
            .data
array       .byte  5, 1, 4            ; Elements of the array
lastElement

finish      nop
```

The program is done according to given bubble sort algorithm and it should sort the elements in the array.

The flowchart of the program:



3 CONCLUSION

The most important thing that we learned from the experiment is working with arrays. After this experiment, we have learned how to implement algorithms using arrays. We have also learned how to implement Bubble Sort and Encryption Algorithms in MSP430 Design.

The hardest part of the experiment was chosing the registers to work with in the second part. To illustrate, we could not load register R3 in any case. In the debug mode of the Code Composer Studio we saw the error. Additionally, we saw that the array behaves constant in the 2nd part of the experiment. Actually we could not make this correct in the experiment.

In the first part of the experiment, when we are debugging the program, in the 2nd encryption operation implamantation, the values of the registers were different than needed. Then, we saw that in the rrc command carry bit returns 1 and we wrote clrc command to fix that. Using that command we fixed the problem.