Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 3

Experiment Date : 24.10.2016

Group Number : Monday - 13

Group Members :

ID	Name	Surname
040130051	Halil İbrahim	Onuş
150130002	Ahmet Seha	Çelenk
150130032	Baran	Kaya

Laboratory Assistant : Gökhan Seçinti

1 INTRODUCTION

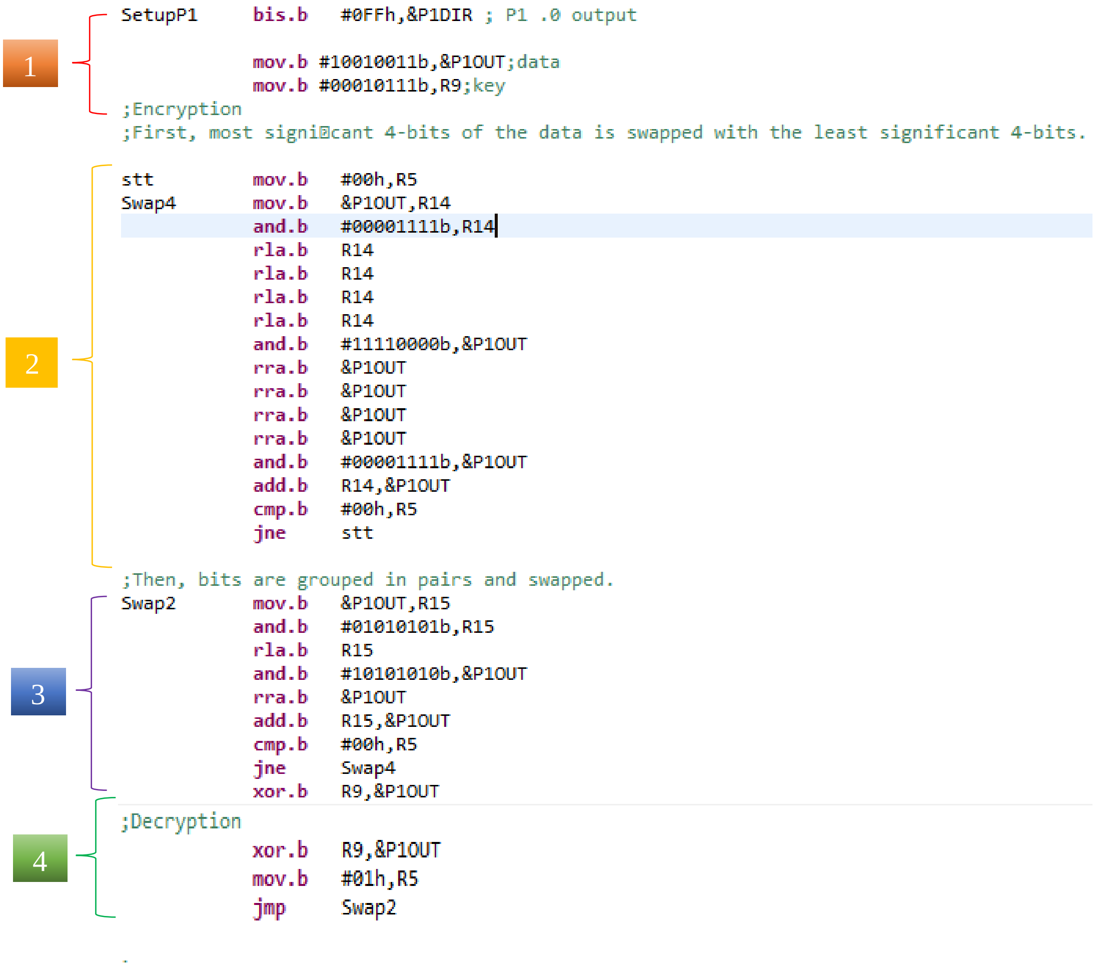
In this experiment we have learn how to define an array and usage of register as pointer, also we have been more familiar to branching. We implemented these algorithms in assembly: Bit-wise Encryption and Bubble Sort.

2 EXPERIMENT

Experiment 3 consists of two parts. First part is implementing Bit-wise Encryption algorithm in assembly language and also show all steps in P1 ports LEDs. In the second part, we have learned how to define array in assembly while implementation of Bubble Sort algorithm.

2.1 BIT-WISE ENCRYPTION

Assembly code of Bit-wise Encryption algorithm:

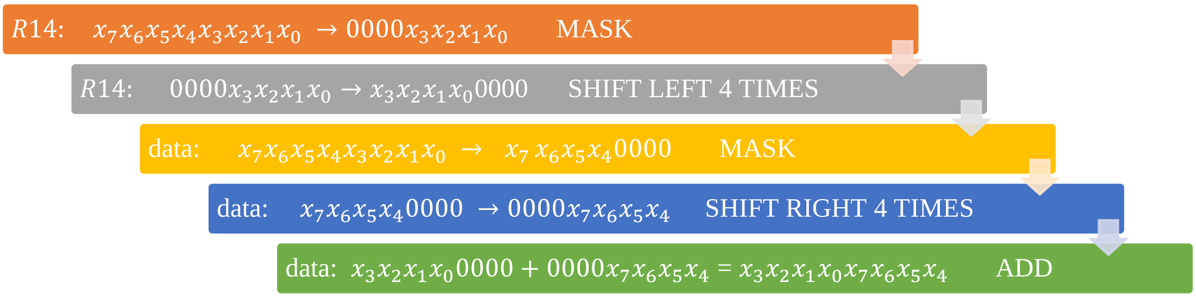


```
SetupP1     bis.b   #0FFh,&P1DIR ; P1 .0 output

            mov.b #10010011b,&P1OUT;data
            mov.b #00010111b,R9;key
;Encryption
;First, most significant 4-bits of the data is swapped with the least significant 4-bits.

stt         mov.b   #00h,R5
Swap4       mov.b   &P1OUT,R14
            and.b   #00001111b,R14
            rla.b   R14
            rla.b   R14
            rla.b   R14
            rla.b   R14
            and.b   #11110000b,&P1OUT
            rra.b   &P1OUT
            rra.b   &P1OUT
            rra.b   &P1OUT
            rra.b   &P1OUT
            and.b   #00001111b,&P1OUT
            add.b   R14,&P1OUT
            cmp.b   #00h,R5
            jne     stt

;Then, bits are grouped in pairs and swapped.
Swap2       mov.b   &P1OUT,R15
            and.b   #01010101b,R15
            rla.b   R15
            and.b   #10101010b,&P1OUT
            rra.b   &P1OUT
            add.b   R15,&P1OUT
            cmp.b   #00h,R5
            jne     Swap4
            xor.b   R9,&P1OUT
;Decryption
            xor.b   R9,&P1OUT
            mov.b   #01h,R5
            jmp     Swap2
```

1 In this part of code, we give a direction to all bits of P1 as using output (LEDs). After that we assign data value #93h ($data \stackrel{\text{def}}{=} P1OUT$) and key value #17h.

2 R5 is our flag that shows encryption (R5 = 0), decryption (R5 = 1). At “stt” label we assign 0 to R5 for doing encryption. After that we assign our *data* to R14 for swapping most and least significant 4-bits. By “and” operation we mask least significant 4-bits of *data* and shift left arithmetically four times.

Same masking process is done to the first 4bits part of $data$. Then rotate right arithmetically four times. After that operations we add changed R14 and changed $data$ and obtain wanted swapped $data$.



$R14$: $x_7x_6x_5x_4x_3x_2x_1x_0 \rightarrow 0000x_3x_2x_1x_0$ MASK

$R14$: $0000x_3x_2x_1x_0 \rightarrow x_3x_2x_1x_00000$ SHIFT LEFT 4 TIMES

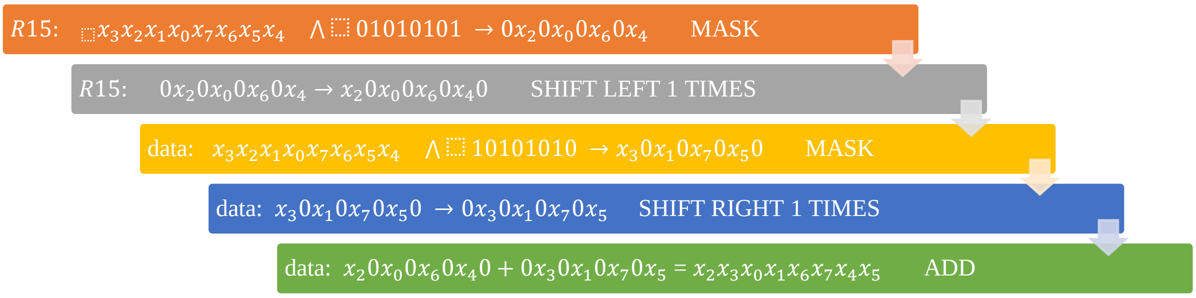
data: $x_7x_6x_5x_4x_3x_2x_1x_0 \rightarrow x_7\, x_6x_5x_40000$ MASK

data: $x_7x_6x_5x_40000 \rightarrow 0000x_7x_6x_5x_4$ SHIFT RIGHT 4 TIMES

data: $x_3x_2x_1x_00000 + 0000x_7x_6x_5x_4 = x_3x_2x_1x_0x_7x_6x_5x_4$ ADD

After that control which operation (encryption or decryption) by comparing value of flag (R5), then continue to “swap2” in encryption or “stt” end of decryption case.

3 R14 is our temp register for swapping grouped pairs. In this part of code, we swap grouped pairs for having more complex encryption algorithm. Even bits are selected by “and” operation of R14 with #01010101b, then rotate left for switching location of even bits. Also same process is done for odd bits. Odd bits are selected by “and” operation of $data$ with #10101010b, then rotate right for switching location of odd bits. By adding R15 and $data$ we obtain swapped data.



$R15$: $x_3x_2x_1x_0x_7x_6x_5x_4 \wedge 01010101 \rightarrow 0x_20x_00x_60x_4$ MASK

$R15$: $0x_20x_00x_60x_4 \rightarrow x_20x_00x_60x_40$ SHIFT LEFT 1 TIMES

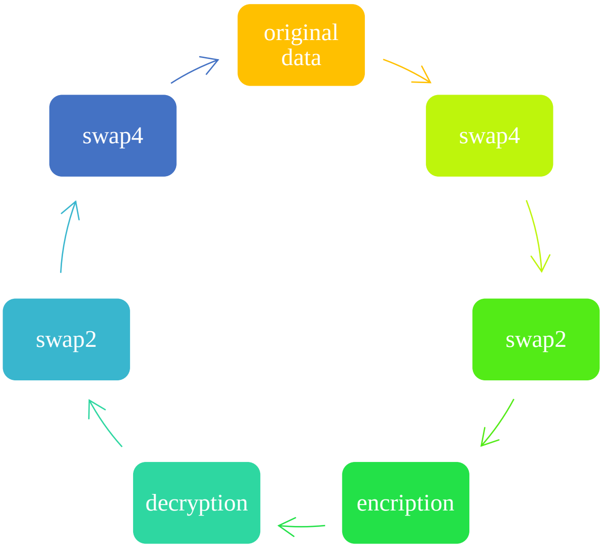
data: $x_3x_2x_1x_0x_7x_6x_5x_4 \wedge 10101010 \rightarrow x_30x_10x_70x_50$ MASK

data: $x_30x_10x_70x_50 \rightarrow 0x_30x_10x_70x_5$ SHIFT RIGHT 1 TIMES

data: $x_20x_00x_60x_40 + 0x_30x_10x_70x_5 = x_2x_3x_0x_1x_6x_7x_4x_5$ ADD

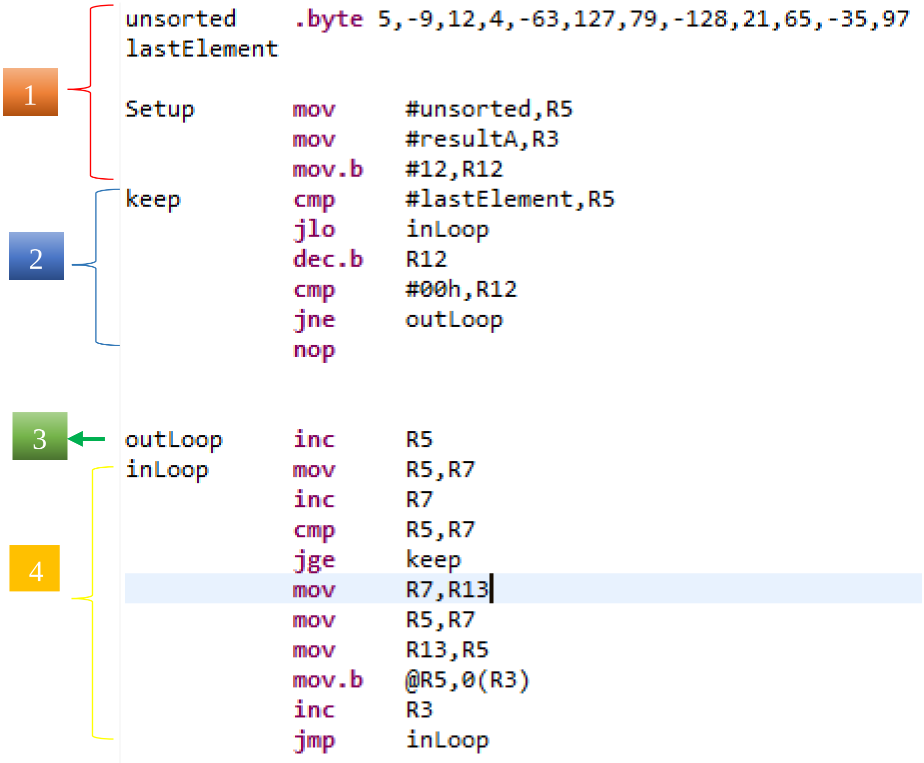
After that control which operation (encryption or decryption) by comparing value of flag (R5), then continue to “xor” operation of $data$ with key if encryption else jump to “swap4” doing reversely same processes of decryption.

4 Decryption process start by “xor” operation of $data$ with key. Then assign flag (R5) as 1, jump to “swap2”. After “swap2” we obtain least and most significant 4-bits swapped original data. After “swap4” we obtain original data and start process of encryption again.



2.2 Bubble Sort

Bubble Sort is a sorting algorithm that organizes a given sequence in ascending or descending order. Assembly code of Bubble Sort algorithm:



```
unsorted    .byte 5,-9,12,4,-63,127,79,-128,21,65,-35,97
lastElement

Setup       mov     #unsorted,R5
            mov     #resultA,R3
            mov.b   #12,R12
keep        cmp     #lastElement,R5
            jlo     inLoop
            dec.b   R12
            cmp     #00h,R12
            jne     outLoop
            nop


outLoop     inc     R5
inLoop      mov     R5,R7
            inc     R7
            cmp     R5,R7
            jge     keep
            mov     R7,R13
            mov     R5,R7
            mov     R13,R5
            mov.b   @R5,0(R3)
            inc     R3
            jmp     inLoop
```

1 In this part of code, array which is named as “unsorted” is defined and initialized. R5 register is assigned as pointer to unsorted[0] array and R3 is also assigned as pointer to resultA[0] array.

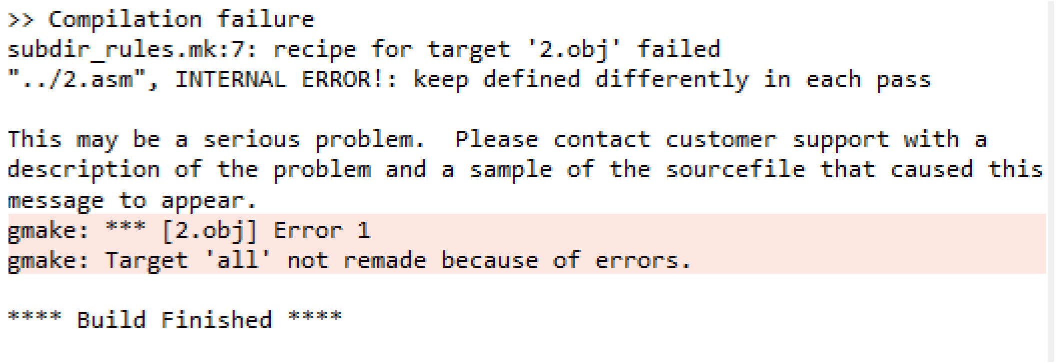
2 If compared numbers are equal the number should be kept and it is realized in“keep” label. In first line, we compare if we reached last element in inner loop or not. If it has not reached last element (jlo is true) jump to inner loop to continue for comparing numbers up to last element. If it (R5 pointer) has reached last element it decrements the R12(outer loop counter). Then control if the R12(outer loop counter) have been reached 0 or not. If it hasn’t been reached jump to “outLoop”. If reached “nop” no operation is done.

3 In this line, it increments pointer(R5) to point next array element. Then inner loop has started to compare at index R5.

4 R7 is temporary register for purpose of pointing next array element. R7 is assigned to R5 then incremented by 1 (shows next element of R5 pointed). Then it compares values, if next array element is equal to element or greater than element (jge is true) then jumps to “keep” label for keeping this element. If it is smaller than element that we compared, swapping operation is done by using R13 temporary register. After that we assign output array then jump to inner loop again.

3 CONCLUSION

We have learned how to implement Bit-wise Encryption algorithm and Bubble Sort algorithm in assembly language. In second part of experiment we encountered internal error that is:



```
>> Compilation failure
subdir_rules.mk:7: recipe for target '2.obj' failed
"../2.asm", INTERNAL ERROR!: keep defined differently in each pass

This may be a serious problem.  Please contact customer support with a
description of the problem and a sample of the sourcefile that caused this
message to appear.
gmake: *** [2.obj] Error 1
gmake: Target 'all' not remade because of errors.

**** Build Finished ****
```

We think our bubble sort algorithm implementation logic is good but we have done some mistakes in array operations: pointing, next element pointing etc.