Department of Computer Engineering

BLG 351E Microcomputer Laboratory Experiment Report

Experiment No : 3

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 knowledge about assembly language and how to apply basic programming level concepts on assembly. Also, we have gained more experience on MSP430 educational board. And we gained more information and experience about bubble sort and basic bit wise encryption

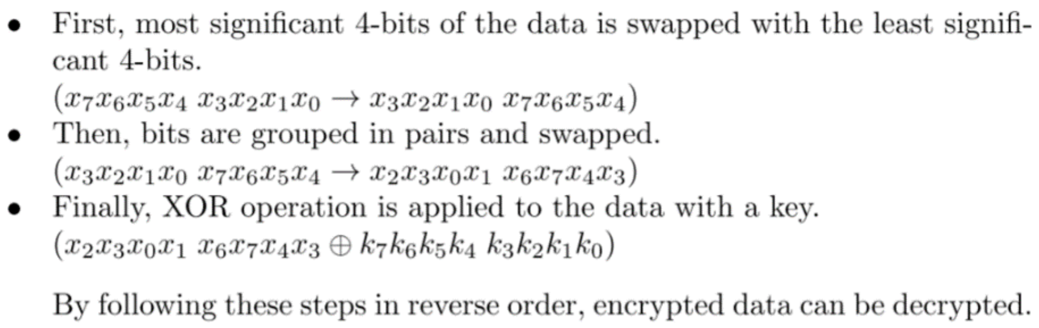
In the first experiment we have applied given pseudocode for bit-wise encryption on MSP430 with assembly language. After that we have encrypted given example. Then we have decrypted to get first example from our encrypted example.

In the second experiment we have applied given pseudocode for bubble sort on MSP430 with assembly language. After that we have applied the algorithm to given example array.

2. EXPERIMENT

2.1. PART 1

In this part, we have applied given algorithm for bit wise encryption. Given algorithm swaps the most significant 4 bits with the least significant 4 bits. In our assembly code this part labeled with swap Then bits swapped in pair. This part labeled as shuffle in our code. A key used for next step. Appling XOR to swapped and shuffled bits with given key gives the final result. Appling first XOR, then shuffle and then swap decrypts given data with known key. Algorithm is given below.



- First, most significant 4-bits of the data is swapped with the least significant 4-bits.
  ($x_7x_6x_5x_4\ x_3x_2x_1x_0 \rightarrow x_3x_2x_1x_0\ x_7x_6x_5x_4$)
- Then, bits are grouped in pairs and swapped.
  ($x_3x_2x_1x_0\ x_7x_6x_5x_4 \rightarrow x_2x_3x_0x_1\ x_6x_7x_4x_3$)
- Finally, XOR operation is applied to the data with a key.
  ($x_2x_3x_0x_1\ x_6x_7x_4x_3 \oplus k_7k_6k_5k_4\ k_3k_2k_1k_0$)

By following these steps in reverse order, encrypted data can be decrypted.

After coding explained algorithm we have encrypted and decrypted given example successfully. Each step’s produced result checked by hand.

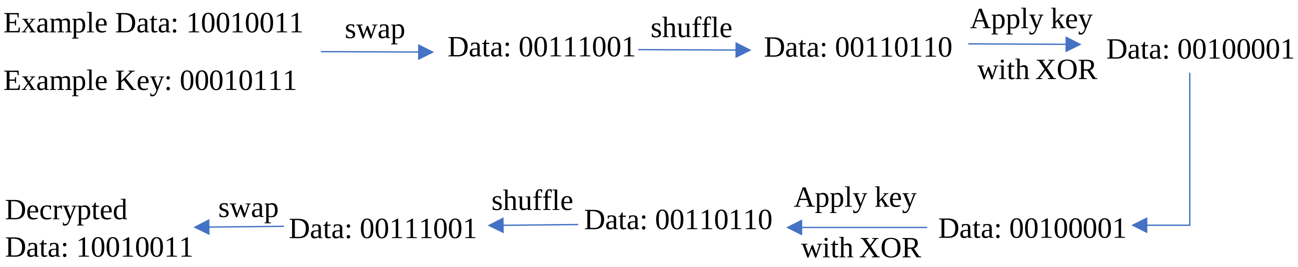
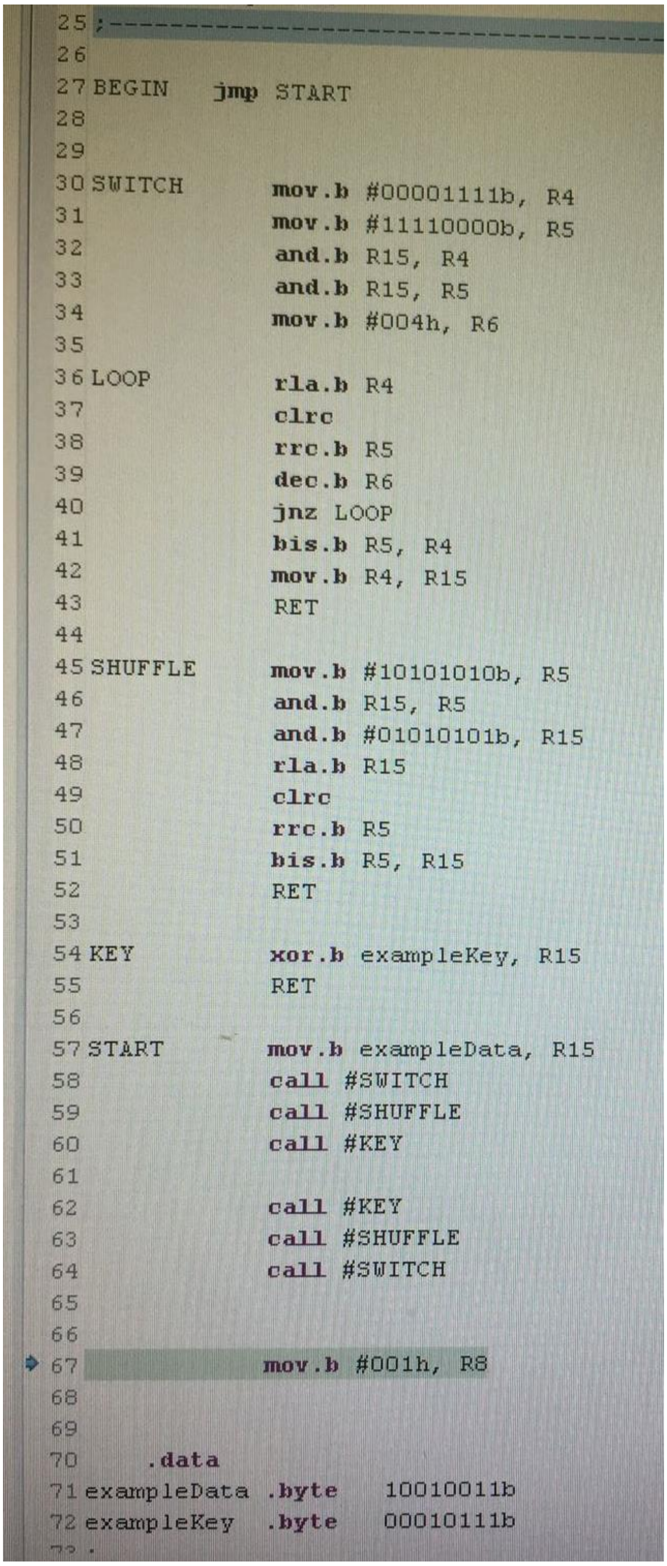


Figure 2.1.1: Example steps for bitwise encryption and decryption



```
;-----------------------------------

BEGIN    jmp START


SWITCH          mov.b #00001111b, R4
                mov.b #11110000b, R5
                and.b R15, R4
                and.b R15, R5
                mov.b #004h, R6

LOOP            rla.b R4
                clrc
                rrc.b R5
                dec.b R6
                jnz LOOP
                bis.b R5, R4
                mov.b R4, R15
                RET

SHUFFLE         mov.b #10101010b, R5
                and.b R15, R5
                and.b #01010101b, R15
                rla.b R15
                clrc
                rrc.b R5
                bis.b R5, R15
                RET

KEY             xor.b exampleKey, R15
                RET

START           mov.b exampleData, R15
                call #SWITCH
                call #SHUFFLE
                call #KEY

                call #KEY
                call #SHUFFLE
                call #SWITCH


                mov.b #001h, R8


      .data
exampleData .byte   10010011b
exampleKey  .byte   00010111b
```

Picture 2.1.2: Code used for bitwise encryption

As seen in the code switch part where most and least significant bits swapped has a loop for shifting bits 4 times to left or to right. Most significant and least significant bits of the given data filtered with and operation and stored in R5, R4 in order. Most significant bits which is stored on R5 shifted right, and least significant bits which is stored in R4 shifted left four times. Then combined with an OR operation.

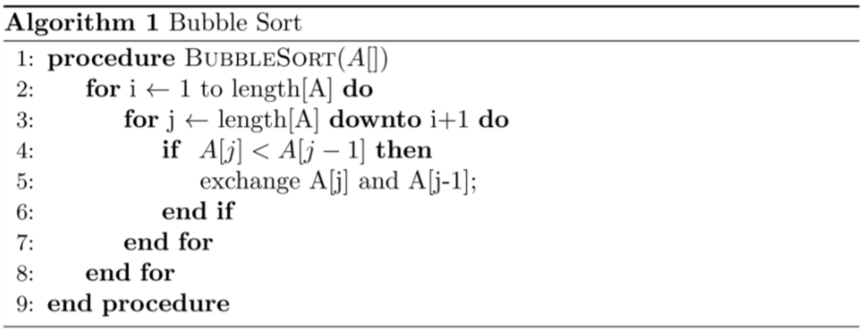
Shuffle step uses filters to get to odd numbered bits and even numbered bits. Since the aim is to swap bit pairs, we can use left shift on bits which is on odd number order and we can use right shift on bits which is on even number order. Combining those result will give us the data with switched bit pairs.

Part that labeled as key in the code uses XOR and a given key to complete encryption or start decryption.

To combine those steps R15 used as a data register. Each step starts with getting its data from R15 and ends with putting back processed data in R15. Using steps in order (switch, shuffle, key) produces encrypted data in R15. Using steps in reverse order (key, shuffle, switch) produces decrypted data.

2.2. PART 2

In this experiment we have applied given pseudocode for bubble sort. Creating functioning for loops in assembly was a challenging task. We have applied for using a variable to increase or decrease and comparing with a constant in each step.

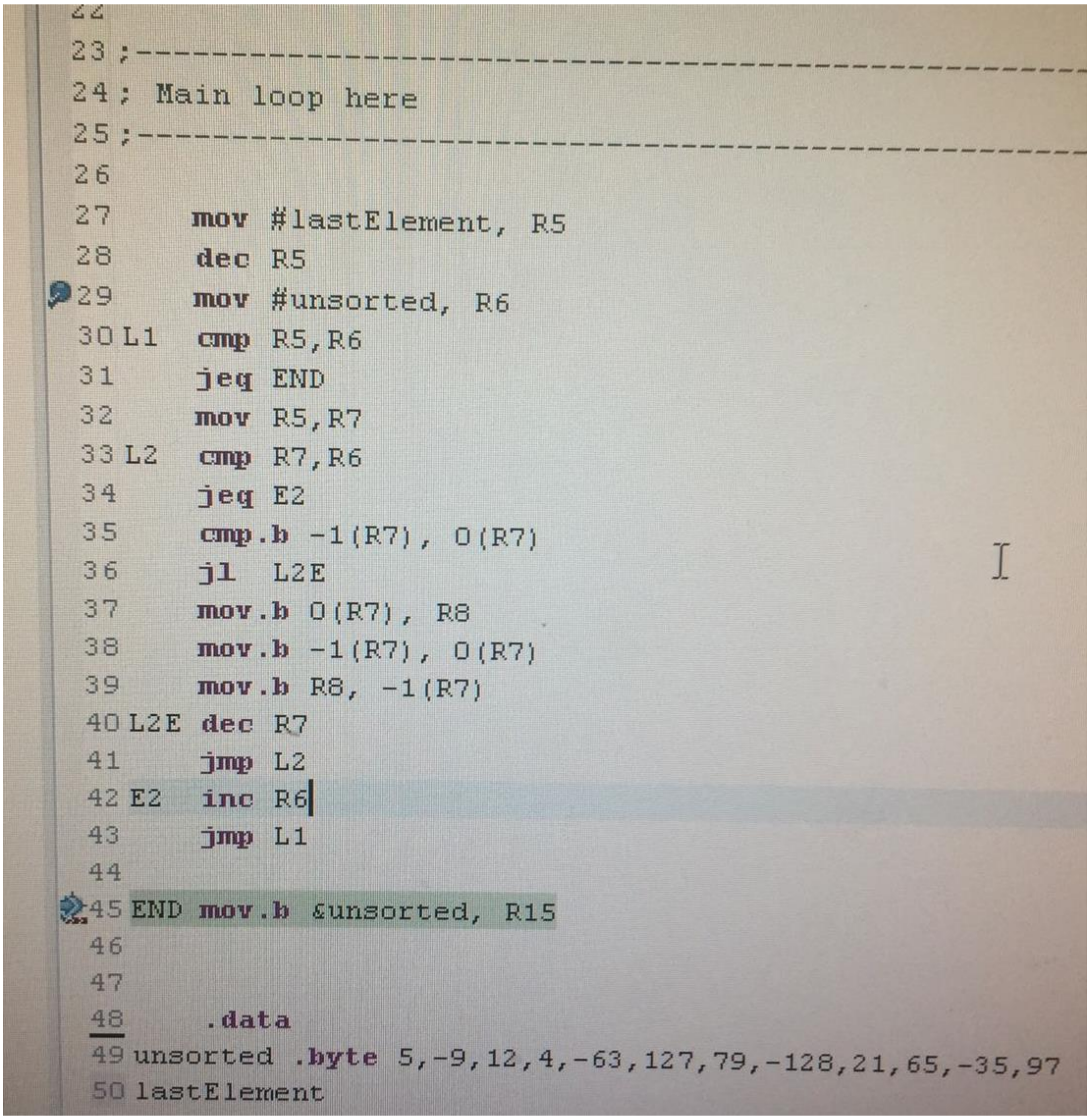


Algorithm 1 Bubble Sort

```
procedure BubbleSort($A[]$)
    for i ← 1 to length[A] do
        for j ← length[A] downto i+1 do
            if $A[j] < A[j-1]$ then
                exchange A[j] and A[j-1];
            end if
        end for
    end for
end procedure
```

Picture 2.2.1: Pseudocode for bubble sort

After we have applied for loops in assembly, we have implemented if statement and exchange part using a temporary register. Then we have realized that we can simplify the solution by using direct addresses in for loops rather than variables.



```
;-----------------------------------------------
; Main loop here
;-----------------------------------------------

        mov  #lastElement, R5
        dec  R5
        mov  #unsorted, R6
L1      cmp  R5,R6
        jeq  END
        mov  R5,R7
L2      cmp  R7,R6
        jeq  E2
        cmp.b -1(R7), 0(R7)
        jl   L2E
        mov.b 0(R7), R8
        mov.b -1(R7), 0(R7)
        mov.b R8, -1(R7)
L2E     dec  R7
        jmp  L2
E2      inc  R6
        jmp  L1

END mov.b &unsorted, R15


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

Picture 2.2.2: Assembly code for bubble sort

In the code, label L1 represents first loop, and L2 represents second loop. R5 used as the last address of the array. R6 and R7 used as variables in loops. And R8 used as temporary register in swap operation. After the implementation, given array sorted by decreasing order.

3. CONCLUSION

In this experiment we learned to implement loops and implement bit operations such as filtering and shifting. Also gained experience on assembly language and on MSP430 microcontroller. Also we have learned to how to debug complicated structures like loops in assembly language. Debugging and checking work flow on loop inside loop was a great challenge. Using breakpoints and memory browser helped us in debugging. We also used memory browser for checking results for part 2.