3

Advanced Assembly Coding

This experiment aims to enhance the practical experience on advanced assembly coding. Therefore, investigate definitions in the upcoming sections and implement given algorithms. At the end of the experiment, students will gain the ability to implement more complex algorithms on MSP430.

Students are required to implement the following algorithms:

- Bit-wise Encryption
- Bubble Sort

3.1 Background Information

In this section, Bit-wise Encryption and Bubble Sort algorithms are explained.

3.1.1 Bit-wise Encryption

Bit-wise Encryption is achieved by manipulating the bits of data. A most commonly used method is applying XOR operation to the data with a predefined key. In order to decrypt the encrypted data, XOR operation is re-applied to the encrypted data with the same key.

An illustration of the described 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.

3.1.2 Bubble Sort

Bubble Sort is a sorting algorithm that organizes a given sequence in ascending or descending order. The pseudo code of this algorithm is given below:

Algorithm 1 Bubble Sort

```
1: procedure BubbleSort(A[])
2:     for i ← 1 to length[A] do
3:         for j ← length[A] downto i+1 do
4:             if A[j] < A[j − 1] then
5:                 exchange A[j] and A[j-1];
6:             end if
7:         end for
8:     end for
9: end procedure
```

3.2 Experiment

3.2.1 Part 1 - Encryption

Implement the encryption method described above. Test your program with different inputs. An example input is given below[1] (Hint:check masking operations in MSP430_introduction document.).

```
1 ;Example input data and key
2 exampleData      .byte   10010011b
3 exampleKey       .byte   00010111b
```

3.2.2 Part 2 - Bubble Sort

Implement Bubble Sort algorithm and test your implementation using input arrays with different sizes. An example input array to be sorted is provided below.

```
1 ;Integer array
2 unsorted     .byte   5,-9,12,4,-63,127,79,-128,21,65,-35,97
3 lastElement
```

#unsorted gives the pointer to the first element of the array, which is the memory address of 5 in the example. If 5 is stored in the memory location 0x300, then $\#unsorted \Rightarrow 0x300$.

```
1   mov #unsorted, R5 ;address of 5 (in the example) is
        loaded to R5
```

#lastElement gives the pointer to the memory location that follows the memory cell containing 97 in the example. If 97 is stored in the memory location 0x300, then $\#lastElement \Rightarrow 0x301$. By means of the appropriate addressing modes, you can reach the elements of the array and perform the sorting operation.

```
1   mov #lastElement, R5 ;address of the memory location
        following 97 (in the example) is loaded to R5
```

[1] Do not forget to include the given code which defines initialized data arrays under the **.data** section of your main.asm file.

3.3 Report

Prepare a report by considering the items listed below.

- **Content of the experiment** Briefly explain what is studied during the experiments.
- **Experiment:** Include assembly codes and draw flow charts for the algorithms that you implemented with sufficient explanations.

Explain the anomalies (if countered) with their possible reasons while testing your program.