Assignment 3

Objectives:

- Memory addressing modes
- Assembly instructions
- Reading object code (machine level instruction) expressed in hexadecimal and understanding how these instructions are stored in memory
- Writing a C program that corresponds to given assembly program

---

Submission:

- Submit your document called **Assignment_3.pdf**, which must include the number of the question you are answering (e.g., Question 1) followed by your answer, keeping the questions in their original numerical order. Formatting your assignment document this way makes it a lot easier to mark. ☺
    - Add your full name and student number at the top of the first page of your document.
- **If you write your answers by hand (as opposed to using a computer application to write them),** when putting your assignment document together, do not take photos of your assignment sheets! Scan them instead! Better quality -> easier to read -> easier to mark! ☺

Due:

- Friday Oct. 8 at 4pm on CourSys
- Late assignments will receive a grade of 0, but they will be marked (if they are submitted before the solutions are posted on Monday) in order to provide feedback to the student.

Marking scheme:

- This assignment will be marked as follows:
    - Questions 1, 2 and 3 will be marked for correctness.
- The amount of marks for each question is indicated as part of the question.

- A solution will be posted on Monday after the due date.

1. [10 marks] Memory addressing modes

Assume the following values are stored at the indicated memory addresses and registers:

Memory Address	Value
`0x230`	`0x23`
`0x234`	`0x00`
`0x235`	`0x01`
`0x23A`	`0xed`
`0x240`	`0xff`

Register	Value
`%rdi`	`0x230`
`%rsi`	`0x234`
`%rcx`	`0x4`
`%rax`	`0x1`

Imagine that the operands in the table below are the **Src** (source) operands for some unspecified assembly instructions (any instruction except `lea*`), fill in the following table with the appropriate answers.

Note: We do not need to know what these assembly instructions are in order to fill the table.

Operand	**Operand Value** (expressed in hexadecimal)	**Operand Form** (Choices are: Immediate, Register or one of the 9 Memory Addressing Modes)
`%rsi`		Register
`(%rdi)`		Indirect memory addressing mode
`$0x23A`		
`0x240`	`0xff`	
`10(%rdi)`		“Base + displacement” memory addressing mode
`560(%rcx,%rax)`		
`-550(, %rdi, 2)`		

0x6(%rdi, %rax, 4)		

Still using the first table listed above displaying the values stored at various memory addresses and registers, fill in the following table with three different **Src** (source) operands for some unspecified assembly instructions (any instruction except `lea*`). For each row, this operand must result in the operand **Value** listed and must satisfy the **Operand Form** listed.

Operand	**Value**	**Operand Form** (Choices are: Immediate, Register or one of the 9 Memory Addressing Modes)
	0x00	Absolute memory addressing mode
	0x00	Scaled indexed memory addressing mode
	0x00	Indexed memory addressing mode

2. [2 marks] Machine level instructions and their memory location

Consider a function called **`arith`**, defined in a file called **`arith.c`** and called from the main function found in the file called **`main.c`.**

This function **`arith`** performs some arithmetic manipulation on its **three parameters**.

Compiling **`main.c`** and **`arith.c`** files, we created an executable called **`ar`**, then we executed the command:

```
objdump -d ar > arith.objdump
```

We display the partial content of **`arith.objdump`** below. The file **`arith.objdump`** is the disassembled version of the executable file **`ar`**.

Your task is to fill in its missing parts, which have been underlined:

```
0000000000400527 <arith>:
  400527:    48 8d 04 37         lea   (%rdi,%rsi,1),%rax
  ______:    48 01 d0            add   %rdx,%rax
  40052e:    48 8d 0c 76         lea   (%rsi,%rsi,2),%rcx
  ______:    48 c1 e1 04         shl   $0x4,%rcx
  400536:    48 8d 54 0f 04      lea   0x4(%rdi,%rcx,1),%rdx
  ______:    48 0f af c2         imul  %rdx,%rax
```

```
______:    c3                    retq
```

3. [8 marks] C program versus assembly program

Do the Homework Problem 3.58 at the end of Chapter 3. Make sure you satisfy the following requirements:

- Your code must be commented and well spaced such that others (i.e., TA's) can read your code and understand what each instruction does.
- About comments:
  - Comment of Type 1: Here is an example of a useful comment:

    ```
    cmpl  %edx, %r8d    # loop while j < N
    ```

  - Comment of Type 2: Here is an example of a **not** so useful comment:

    ```
    cmpl  %edx, %r8d    # compare %edx with %r8d
    ```

    Do you see the difference? Make sure you write comments of Type 1.
- You cannot use the **goto** statement.
- You must write your program using C (not C++) and your program must compile on our *target machine*.

Once you have created your program and saved it in a file called **decode2.c**, generate its assembly code version using the optimization level "g" (`-Og`) and save it in a file called **decode2.s**.

Include the content of both files **decode2.c** and **decode2.s** in your assignment **Assignment_3.pdf** document. Label them well.

You do not have to electronically submit your files **decode2.c** and **decode2.s** on CourSys. However, your program must be functionally correct (i.e., it must compile, execute properly and solve this problem).