CS2208A Lab 3

Due by: **11:55 pm on Thursday, November 10, 2022**

Wednesday November 9, 2022 (section 3 @ *HSB-13* from 10:30 am to 11:30 am)
Wednesday November 9, 2022 (section 5 @ *HSB-13* from 11:30 am to 12:30 pm)
Wednesday November 9, 2022 (section 4 @ *HSB-14* from 12:30 pm to 1:30 pm)
Wednesday November 9, 2022 (section 6 @ *HSB-14* from 2:30 pm to 3:30 pm)
Wednesday November 9, 2022 (section 8 @ *HSB-14* from 3:30 pm to 4:30 pm)
Wednesday November 9, 2022 (section 7 @ *HSB-14* from 4:30 pm to 5:30 pm)
Wednesday November 9, 2022 (section 9 @ *HSB-14* from 5:30 pm to 6:30 pm)

After attending the lab session, you must submit a lab report answering the lab questions.

The objective of this lab is:

- ARM shift operations
- ARM addressing modes

Before practicing this lab, you need to review and fully understand tutorials 10 and 11 *(Tutorial_10_ARM_Shift_Instructions.pdf and Tutorial_11_ARM_Addressing_Modes.pdf).*

Note that this lab will be automatically marked.
The automatic marking will not consider any partial correct answer.
It is either 100% correct or not.

===

ADDRESSING MODES

An addressing mode is simply a means of *expressing the location of an operand*. An address can be a register such as `r3` or `PC` (*Program Counter*). An address can be a location in memory, such as address `0x12345678`. You can even express an address *indirectly* by saying, for example, "the address is loaded in register `r1`". The various ways of expressing the location of data are called collectively *addressing modes*.

Suppose someone said, "Here are ten dollars". They are giving you the actual item. This is called a **literal** or **immediate** value because it is what you actually get. Unlike all other addressing modes, you do not have to retrieve addresses from a register or memory location.

If someone says, "Go to this full address in the world, and you will find the money on the table", they are actually telling you *where* the money is (i.e., its address or its actual location). This is called an **absolute address** because it expresses precisely where the money is in absolute terms. This addressing mode is also called **direct addressing**. Unfortunately, ARM processors do ***not*** support this addressing mode.

Now here is where the fun starts. Suppose someone says, "Go to room 12 *in this building,* and you will find something to your advantage on the table". You arrive in room 12 and see a message on the table saying, "The money is in *this full address in the world"*. In this case, we have an **indirect address** because room 12 does not have the money but a pointer to where it is. So we have to go to a second address to get the money. Indirect addressing is also called **pointer-based** addressing because you can think of the note in room 12 as pointing to the actual data.

In real life, we do not confuse a room number or an address with a sum of money. However, in a computer, all data is stored in binary form and the programmer has to remember whether a variable (or constant) is an address or a data value.

A '#' symbol indicates immediate (literal) addressing in front of the operand. Thus, `#5` in an instruction means the actual value 5. For example, a typical ARM instruction is `MOV r0,#5,` which means move the value 5 into register `r0`.

Indirect addressing is indicated by ARM processors by placing the pointer in square parentheses; for example, `[r1]`. All ARM indirect addresses are of the basic form `LDR` **`r0,`**`[r1]` or `STR r3`**`,[r6]`**.

There are variations on this addressing mode; for example, `LDR` **`r0,`**`[r1,#4]` specifies an address that is four bytes from the location pointed at by the contents of register `r1`. It can also have a side effect, such as *autoindexing pre-indexed addressing mode*, e.g., `LDR` **`r0,`**`[r1,#4]!` or *autoindexing post-indexed addressing mode*, e.g., `LDR` **`r0,`**`[r1],#4`

In all these indirect addressing modes, the offset can be a constant, i.e., static (as indicated in the above examples), or *dynamic*, by putting the value of the offset in a register, e.g.,

```
LDR r0,[r1,r2]

LDR r0,[r1,r2]!

LDR r0,[r1],r2
```

All offsets associated with indirect addressing (regardless of constant or dynamic) can be positive or negative.