CS2208b Lab No. 3
Introduction to Computer Organization and Architecture

Monday February 29, 2016 (section 3 @ ***SSC-1032*** from 2:30 pm to 3:30 pm)
Tuesday March 1, 2016 (section 4 @ ***SSC-1032*** from 1:30 pm to 2:30 pm)
Tuesday March 1, 2016 (section 8 @ ***HSB-14*** from 3:30 pm to 4:30 pm)
Thursday March 3, 2016 (section 5 @ ***SSC-1032*** from 2:30 pm to 3:30 pm)
Friday March 4, 2016 (section 6 @ ***SSC-1032*** from 1:30 pm to 2:30 pm)
Friday March 4, 2016 (section 7 @ ***SSC-1032*** from 3:30 pm to 4:30 pm)

The objective of this lab is:

- To practice ARM assembly programming

If you would like to leave, and at least 30 minutes have passed, raise your hand and wait for the TA.
Show the TA what you did. If, and only if, you did a reasonable effort during the lab, he/she will give you the lab mark.

==========

REVIEW

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 the location whose address is in register `r1`". The various ways of expressing the location of data are called collectively *addressing modes*.

Suppose someone said, "Here is 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 room 30 and you will find the money on the table", they are actually telling you *where* the money is (i.e., its address is room 30). This is called an **absolute address** because it expresses absolutely exactly where the money is. This addressing mode is also called **direct addressing**. ARM processors do ***not*** support this addressing mode.

Now here is where the fun starts. Suppose someone says, "Go to room 40 and you will find something to your advantage on the table". You arrive at room 40 and see a message on the table saying, "The money is in room 60". In this case we have an **indirect address** because room 40 does not have the money, but a pointer to where it is. We have to go to a second room to get the money. Indirect addressing is also called **pointer-based** addressing, because you can think of the note in room 40 as pointing to the actual data.

In real life we cannot 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.

Immediate (literal) addressing is indicated by a '`#`' symbol in front of the operand. Thus, `#5` in an instruction means the actual value 5. 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 on from the location pointed at by the contents of register `r1`. It can also has 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 (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
```

PROBLEM SET

Before you start practicing this lab, you need to review and fully understand how to use *the Keil ARM simulator*, as well as the various addressing modes.

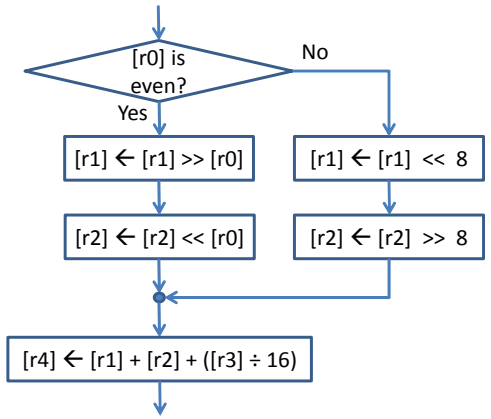
1. The following assembly program adds together a LIST of five numbers stored in memory.

```
        AREA Pointers, CODE, READONLY
        ENTRY
Start   ADR  r0,List    ;register r0 points to List
        MOV  r1,#5      ;initialize loop counter in r1 to 5
        MOV  r2,#0      ;clear the sum in r2
Loop    LDR  r3,[r0]    ;copy the element pointed at by r0 to r3
        ADD  r0,r0,#4   ;point to the next element in the series
        ADD  r2,r2,r3   ;add the element to the running total
        SUBS r1,r1,#1   ;decrement to the loop counter
        BNE  Loop       ;repeat until all elements added
Endless B    Endless    ;infinite loop
List    DCD  3,4,3,6,7 ;the numbers to be added together
                        ;each one is 4 bytes (20 bytes in total)
        END
```

   (a) Modify the original program to utilize:
   - ○ Autoindexing pre-indexed addressing mode
   - ○ Autoindexing post-indexed addressing mode

   (b) Modify the original program by eliminating the instruction "`ADD r0,r0,#4`", while keeping the functionality of the program the same, without using any autoindexing mode.

2. Using only 7 ARM assembly instructions, execute the following flowchart. Test your code by assigning various values to `r0`, `r1`, `r2`, and `r3`.



3. Without using any of the ARM multiplication instructions, write only one ARM instruction that executes [R4] ← 16385 × [R4].
   Manually generate the machine language code for this instruction and verify it using the simulator.

4. What is the machine language of `AND r2,r3,#0x1080`
   Verify it using the simulator.

5. What is the reverse assembly of the following machine language instruction:
   (a) `0x00000000`
   (b) `0x50076808`
   (c) `0xE3A01D22`
   (d) `0xE3A01E88`
   Verify it using the simulator. Hint: you may want to consult Table 3.2 and figure 3.26 in the textbook.

6. What is the reverse assembly of the following machine language instruction: `0xE6DCA408`?
   Verify it using the simulator.
   Hint: you may want to consider Figure 3.39 in the textbook.