CS2208b Lab No. 4
Introduction to Computer Organization and Architecture

Tuesday March 10, 2020 (section 8 @ HSB-13 from 3:30 pm to 4:30 pm)
Wednesday March 11, 2020 (section 6 @ HSB-16 from 1:30 pm to 2:30 pm)
Wednesday March 11, 2020 (section 10 @ HSB-16 from 3:30 pm to 4:30 pm)
Thursday March 12, 2020 (section 11 @ HSB-14 from 11:30 am to 12:30 pm)
Thursday March 12, 2020 (section 4 @ HSB-14 from 1:30 pm to 2:30 pm)
Thursday March 12, 2020 (section 5 @ HSB-14 from 3:30 pm to 4:30 pm)
Thursday March 12, 2020 (section 7 @ HSB-13 from 4:30 pm to 5:30 pm)

The objective of this lab is:
- To practice ARM assemble programing

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, the TA will give you the lab mark.
===

PROBLEM SET

1. Translate the following tasks into a single ARM instruction:

   ADDPL R2, R1, R0, LSL #5 0x50812280 0101 0000 1000 0001 0010 0010 1000 0000

   a. Add 32 times of the content of registers r0 and the content of r1 only if N is clear. Store the result in register r2

   RSBHI R3, R0, #0x990 0xB2603E99 1011 0010 0110 0000 0011 1110 1001 1001

   b. Subtract the content of register r0 from 0x990 and put the results in register r3 only if C is set and Z is clear.

   BICS R4, R1, #0x1100 0xE3D14C11 1110 0011 1101 0001 0100 1100 0001 0001

   c. Clear the 2nd least significant byte of the content of register r1, i.e., store $(00000000)_2$ in it, and put the results in register r4. The result of the instruction must affect the value of the *Current Program Status Register* (CPSR).

   Test your answers by putting them in a program, which starts by assigning values to r0 and r1 and then comparing them together to set/clear the flags in such a way to test both cases of (a), (b), and (c). Note that you will need two sets of r0 and r1 values for each case. Hint: you may want to consider Table 3.2 in the textbook.

   Encode *by hand* the instructions that you suggested for (a), (b), and (c), i.e., generate the 4-byte machine language for each ARM instruction. Verify your answers using Keil's simulator.

2. Convert the GCD algorithm given in this flowchart into

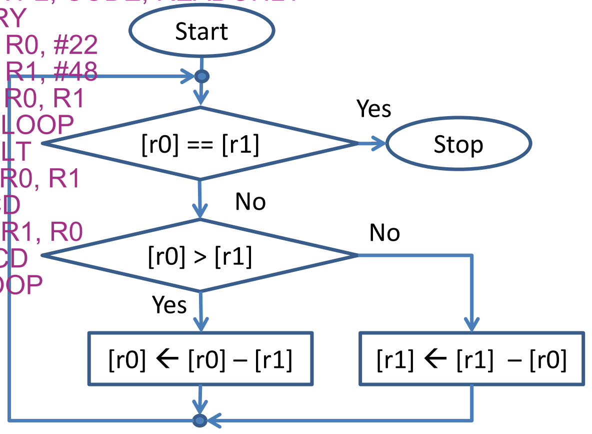
   a. ***Traditional*** assembly, where only branches can be conditional, i.e., do ***not*** utilize the ARM conditional execution feature.

   b. ARM assembly, where any instructions can be conditional, thus improving code density.

   PS: The only instructions you need are `CMP`, `B`, and `SUB`.

   Test your code by assigning various values to `r0` and `r1` using `MOV` instruction.

```
    AREA P2, CODE, READONLY
    ENTRY
    MOV R0, #22
    MOV R1, #48
GCD CMP R0, R1
    BEQ LOOP
    BLT LT
    SUB R0, R1
    B GCD
LT  SUB R1, R0
    B GCD
LOOP B LOOP
    END
```



3. Our ARM7 does not have any division instruction. Yet, you still can implement division by utilizing ASR. Basically, if you want to divide the content of a register by 10, you multiply it by 6554, and then you shift the result using ASR#16. The effective result is $6554 \div 2^{16} = 6554 \div 65536 = 1/10$, which is division by 10.

   Write an ARM assembly program to divide the content of `r0` by 10 and store the result in `r1`.
   Test your code by assigning various values to `r0`
   Hints: Numbers are internally represented in Hexadecimal.

```
     AREA P3, CODE, READONLY
     ENTRY
     MOV R0, #0x64
     LDR R1, =0x199A
     MUL R1, R0, R1
     MOV R1, R1, ASR #16
LOOP B LOOP
     END
```

4. 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.

1110 0110 1101 1100 1010 0100 0000 1000

Register Argument	Base: r12
Post-index pointer	Trans: r10
Increment	Op2: r8
Byte-access	LSL #8
Load	

LDRB R10, [R12], R8, LSL #8