Tutorial 5
CSE 112 Computer Organization

ISA description:

The instructions supported by an ISA are mentioned in the table below. The ISA has 16 general purpose registers: r0 to r15

Name	Semantics	Syntax
Add	Performs reg1 = reg2 + reg3	add reg1, reg2, reg3
Sub	Performs reg1 = reg2 - reg3	sub reg1, reg2, reg3
Branch if zero	Branch to addr if register value (reg) is = 0	bz reg, addr
Branch if not zero	Branch to addr if register value (reg) is != 0	bnz reg, addr
Mov	Moves immediate, a constant value, into reg	mov reg, #imm

Programing example: suppose there is a need to compute 1 + 2 using the ISA described above, then, this might be done as follows:

```
mov r0, #1
mov r1, #2
add r2, r1, r0
```

Q1. Using the ISA described above, write the assembly code for the following problems:

a. Store 10 and 5 in registers r0 and r1. Check whether they are equal or not. If they are equal, then write 1 to register r2, else write 0 to register r2.

b. Store 10 and 5 in registers r0 and r1. Multiply them. Write the product to register r3.

c. Store **X** to register r0. Find the sum of all whole numbers till the value stored in register r0. Store the sum in register r1. Given that X is a constant positive integer, X = 2, then r0 = 2, r1 = 1+2 = 3

You may also use labels in your code to refer to instruction locations.
[Hint: First try to write the code as an algorithm instead of directly writing in assembly]

Q2.

A. Trace the execution of the following assembly program, which has been written using the assembly language described in Q1. Assume every instruction can be completed in

one cycle each. Write down the state of **all the registers** being used, *after* the execution of each instruction. Also, trace the current **program counter** accordingly. Answer by filling the table [given at the end of the question]. If the program ends, fill all the cells in the next row with END [PC increases by 1].

```
0 mov r0, #2
1 mov r1, #0
2 mov r2, #1
3 mov r3, #1
4 bz r0, loop_exit
5 loop: add r1, r2, r1
6 add r2, r3,r2
7 sub r0, r0, r3
8 bnz r0, loop
 loop_exit:
```

B. Assuming all instructions are executed in 1 cycle and the frequency of the processor is 1GHz, what is the total execution time of the program?
C. Suppose add, sub instructions take 1 cycle; bz, bnz take 2 cycles and mov takes 3 cycles to execute, but the processor frequency is reduced to 2 GHz. What is the execution time now? Is it a gain or a loss in execution time, when compared to part B?

Cycle	PC	r0	r1	r2	r3
0					
1					
2					
...	...	...	...	...	...

Q3. Suppose we decide to come up with a new assembly syntax for the same ISA. In the new syntax, the order of operands is different. The change is *only* done to the assembly syntax. The ISA is the same, i.e. **the machine code syntax is the same**. An example of add instruction in both the syntaxes is given below:

Old syntax assembly :add r1,r2, r3
(semantically r1 = r2+r3)
Old syntax machine code :000 01 10 11
New syntax assembly :add r3, r2, r1
(semantically r1 = r2+r3)

New syntax machine code :000 01 10 11

a. Would such a change in assembly syntax require a change in the processor hardware? Why, why not?
b. Would such a change in assembly syntax require a change in the assembler? Why, why not?
c. If an assembled program (machine code) is given to you, can you tell which syntax the programmer used to write the code? If yes, how? If not, why?