Tutorial 2
CSE 112 Computer Organization

Q1. What is ISA? Explain RISC and CISC ISA.

Q2. Below is an imaginary ISA called ***virtual ISA***. Based on the information given on it in the table below, answer the questions below:

Physical memory - **RAM**
addr - address

Opcode	**Meaning**	**Semantic**	**Memory /Reg Type**
000	Load data from physical memory to register	ld addr(mem) reg	mem
001	Store data from register to physical memory	st addr(mem) reg	mem
010	Add content of the reg1 and reg2 and overwrite result in reg1	add reg1 reg2	Reg
011	Subtract content of the reg2 from reg1 and overwrite result in reg1	sub reg1 reg2	Reg
100	Branch to addr if register value (reg) is = 0	breq addr reg	mem
101	Branch to addr if register value (reg) is != 0	brne addr reg	mem
110	Branch to addr if register value (reg) is > 0	brgt addr reg	mem
111	Branch to addr if register value (reg) is < 0	brlt addr reg	mem

The memory space supported by the ISA is at max 4kb in size; addressable in the granularity of double bytes (i.e. we can access two bytes simultaneously from a given location).
Example: At address 1, 16 bits are stored; at address 2, the next 16 bits are stored; and so on.
Moreover, we also have 4 registers in the machine: A, B, C, D; each 2 bytes wide. The binary representation to reference these registers is as follows:

Registers	Binary representation
A	0001
B	0010
C	0100
D	1000

FYI: This representation is called one-hot encoding.

The syntax for binary representation of instruction is as follows:

1. Memory type (an instruction which access memory):
   <opcode> <memory operand> <register operand>
2. Register type (an instruction which can only access registers):
   <opcode> <register operand> <register operand>

Answer the following questions:

a. How many bits are needed to uniquely represent a register?

b. How many bits are required to uniquely represent a memory address?

c. How many bits are required to represent the type (opcode) of instruction?

d. What is the minimum number of bits required to write a memory type instruction?

e. What is the minimum number of bits required to write a register type instruction?

f. Can we reduce the number of bits required to uniquely represent each register without decreasing the number of registers? If so, how?

g. Which type of instruction (memory or register) requires more bits for representation?

h. Suppose both memory and register type instructions are required to be of equal length. To accomplish this, the shorter instruction type is allowed to have unused bits in the representation just after the opcode for the instruction type.
   For example, if memory type is shorter, the syntax becomes:
   <opcode> <unused bits> <memory operand> <register operand>
   Else, if the register type is shorter, then the syntax becomes:
   <opcode> <unused bits> <register operand> <register operand>
   Under the aforementioned assumptions, write down the syntax for both instruction types. Clearly mentioned the number of bits required for each part of the instruction representation.

i. After the changes made (if any) in all the previous parts, rewrite the instruction syntax for both instruction types, clearly mentioning the number of bits for each part of the representation.

j. What should be the required size of the Program Counter (aka Instruction Pointer, aka Instruction Counter) in bits?

Q3. Using the modified ISA of Q2, please write the assembly code for the following problems:

For example: Suppose we have to write **a+b** in assembly as defined by the virtual ISA
Since we have both operands in physical memory, we will first load them in registers
Ld X A; // load value **a** from physical memory location X into register A
Ld Y B; // load value **b** into physical memory location Y into register B
Add A B // Perform A = A+B
St Z A; // store the result back into physical memory location Z

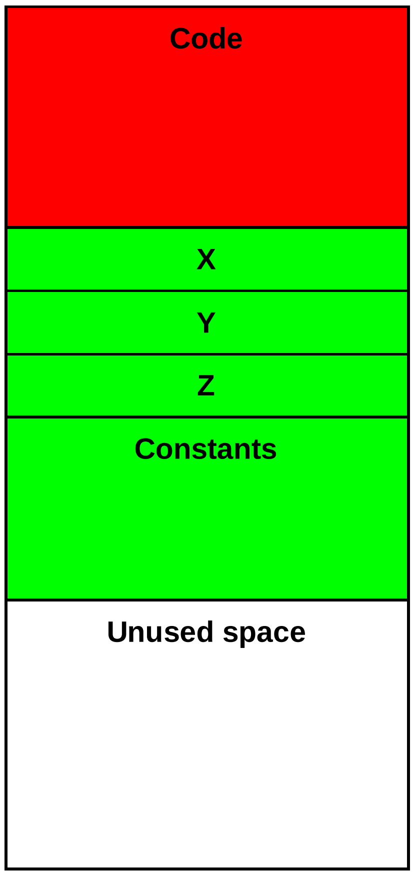
(Note: X, Y and Z are memory locations (aka memory addresses) those contain some fixed values. Moreover, temp0 and temp1 are separate memory locations from where we can access 0 and 1 respectively.)

a. Read two numbers from address X and Y respectively. Compute whether they are equal or not. If they are, write 1 to location Z, else write 0 to location Z.
b. Read two numbers from address X and Y respectively. Multiply these numbers. Write the product to location Z.
c. Read a number from location X. Find the sum of first *val(X)* natural numbers, where val(X) is the value stored at location X. It is given that *val(X) >= 0*. Store the result of the computation back to location Z.

You may use more memory locations if you need to store constants.
You may also use labels in your code to refer to instruction locations.
[Hint: First try to write the code in a high level language (such as C), instead of directly writing in assembly]

Q4. Convert the assembly code for the last part of the previous question (sum of first *n* natural numbers) into binary. You may assume that the code is stored in the memory at address 0. The locations X, Y, and Z begin after the code (highlighted in red). Following that you can store constants in the memory. Please refer to the diagram below for more clarity.

Lower Addresses (0)



Higher Addresses (4kb)

Q5. Suppose the data stored at location X is 2. Trace the execution of the sum of n natural numbers program. Assume every instruction can be completed in one cycle each.

a. Write down the state of all the registers (A, B, C, D and the program counter) *after* each instruction.
b. Mention at each cycle, what all memory accesses are being made (specify the address and the data for the access). (Write the status of memory before instruction execution)
c. Mention at each cycle what all register accesses are being made (specify the register and the data for the access). (Write the status of registers before instruction execution)
d. Mention the data with the memory and register accesses.