Assignment on CPUSim

General instruction for the assignment: There are two main parts, (A) and (B). Try to follow the names exactly as given in the problem description wherever possible (this is very important to evaluate your submission). **Major updates are in bold letters. Note that there is no change hardware components.**

Submission Instruction:

Items to be submitted:

1. **PartA.cpu --- cpu for Part A of the assignment.**
2. **partA.a --- code file for Part A to run on PartA.cpu**
3. **PartB.cpu --- cpu for Part B of the assignment.**
4. **partB.a --- code file for Part B to run on PartB.cpu**

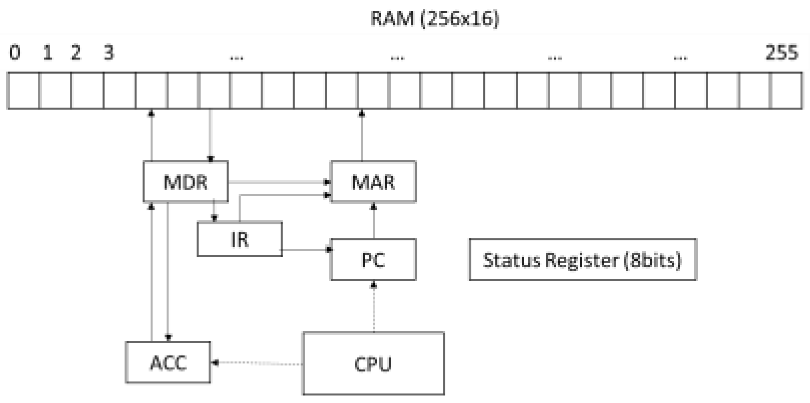
Additionally, follow the naming convention mentioned in the submission link. In short, you need to submit one single .zip file containing a folder (named appropriately and containing details of the group members) and the folder will contain all of these essential 4 files.

Submission link:

https://docs.google.com/forms/d/e/1FAIpQLSeLOsNnqIinkXGiNT9ZfJB095aYak_9MtTtMTMXL88UxZVUXw/viewform

(17 + 13 = 30 marks)

Part (A): Implement a single register CPU in CUPSim, as specified in Figure 1 (without any significant deviation). Broadly, this CPU will support only one type of instruction, and the memory (i.e., RAM) is direct addressable. Status register can be used to define different status bits, like halt, carry, etc.



Note that to answer Part A of the assignment, you cannot add any extra hardware module or any other micro-instruction. If required, you may develop a mechanism to utilize one or more memory cells as buffer or base or index registers.

a) Define hardware modules appropriately along with their specific number of bits in the machine developed by you.

b) Define micro-instructions for the following:

   i. Register transfer
   ii. Arithmetic operations: add and sub
   iii. Test: i. ACC == 0, ACC > 0, ACC < 0, ACC != 0
   iv. Main memory access: read and write
   v. Increment PC
   vi. IO: input and output for integers only (including +ve, -ve, and 0)
   vii. One decode instruction
   viii. Set condition bit: Set-Halt-Bit
   ix. Transfer from MDR to MAR (only the address portion)

c) Define assembly instructions for the following:

Generic instruction format

Opcode (4 bits)		**address**

   i. Unconditional Jump: jmp
   ii. Conditional Jump: all possible jump bases on the microinstruction in the Test section, eg., jmpz -- should indicate a jump if ACC == 0. Similarly, jmpnz, jmpp, jmpn,
   iii. Load: lda
   iv. Store: sta
   v. Read from the keyboard into accumulator: ipa
   vi. Write to terminal from accumulator: opa
   vii. Stop: to stop a program
   viii. Addition: add
   ix. Subtraction: sub
   x. Transfer of content of a memory location to another: m2m
   xi. Transfer of content of a memory address stored in a location provided in the instruction to acc: m2a

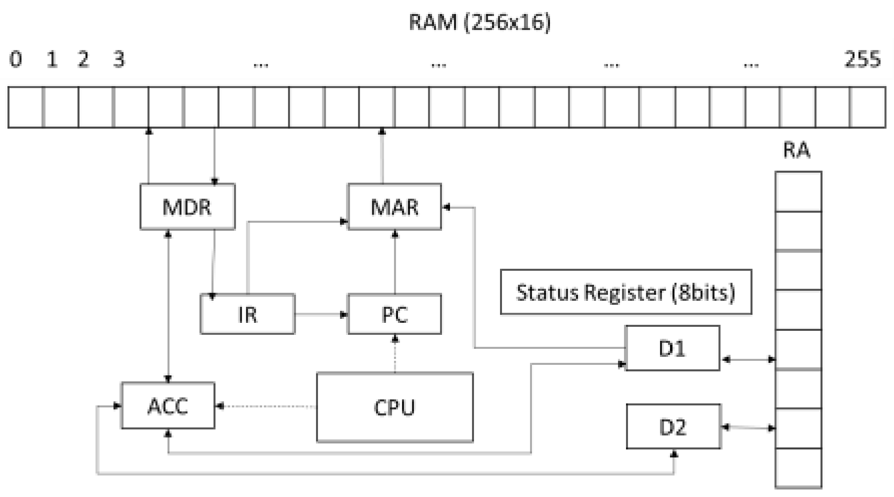
d) Write a program to be executed on the machine developed by you in CPUSim for the following: Store only positive numbers in contiguous memory locations where the user provides the input integers from the keyboard. The user must be allowed to enter any integer and 0 to finish entering numbers. Display all the positive numbers entered by the user in the LIFO order. For example, if user input is 9, -3, 8, 4, -8, 5, 10, -29, 0 (in order), then the output should be 10, 5,4,8,9 (in order).

(Part A) end**.)**

Part B) Update the machine developed in Part A) and write an assembly program.

a) Include hardware modules

   i. Include two registers, namely, ‘d1’ and, ‘d2’.
   ii. Include a register array, namely, ‘ra’ having 8 registers in it where the individual registers in the array should be indicated by r1 to r8 in the assembly instruction.



This figure can be referred to design the updated architecture as mentioned in the Part B. a). Note that this architecture **does not have** a provision for transferring data from MDR to MAR or vice versa.

b) Include microinstructions to support
   a. Data transfer from a register (like acc, mdr, d1 and d2) to register array: e.g., acc -> ra[i]
   b. Data transfer from a register array to a register (like acc, mdr, d1 and d2): e.g., ra[i] -> acc

c) Include assembly instruction
   a. Arithmetic operation:
      i. "add r1 r2" and similar
      ii. "sub r1 r2" and similar

Follow the generic instruction format as shown below, this is mandatory (5bits in opcode).

Generic instruction format

Opcode (5 bits)		**address**

d) Write a program for the same problem as described in Part A). d) with the following modification. Your program should display a cumulative sum up to i-th position from the beginning. For example, if user input is 9, -3, 8, 4, -8, 5, 10, -29, 0 (in order), then memory should store 9, 8, 4, 5, 10 (in order) and the output should be 9, 17, 21, 26, 36 (in order).

(**End of Assignment**)