COL732: Virtualization and Cloud Computing
Semester I, 2022-2023

Lab-2: SIMD Processor
25 August 2022
Deadline: 5 September 2022

Submission Instructions

1. You will submit the source code in **zip** format to Moodle (Lab 2). The naming convention of the zip file should be <Entry_Number>_<First_Name>.zip. The starter code is available here.
2. The Lab would be **auto-graded**. Therefore, **follow** the same naming conventions described in the Deliverables section. Failing to adhere to these conventions will lead to zero marks in the Lab.
3. You should write the code **without** taking help from your peers or referring to online resources except for documentation. Not doing any of these will be considered a breach of the honor code, and the consequences would range from zero marks in the Lab to a disciplinary committee action.
4. You can use **Piazza** for any queries related to the Lab and **avoid** asking queries on the last day. Do not make any **assumptions** about the assignment based on similar assignments available on the internet in case.

Problem Statement

Physicist Divya got access to a few particle simulation programs that she wishes to run. But, these programs were written for an old computer. Unfortunately, she does not have access to the programs' source code, so she can not recompile them for her computer. She also does not have access to any computer that supports this old instruction set. The instruction set has been deprecated. No manufacturer is producing computers supporting this instruction set anymore.

Fortunately, she has a rough schematic of the old architecture and has a data sheet for the instruction set. She has asked you to design an interpreter in Rust that can run these programs.

Architecture:

The processor consists of the following components: control unit (CU), arithmetic and logic unit (ALU), memory, six general purpose registers (R_0 through R_5), accumulator register (AC), a flag register (NF), and the program counter (PC). The ALU unit can perform addition and subtraction operations. The PC register holds the next instruction that has to be executed by the processor. The PC register is auto-incremented by 1 after the execution of an instruction. Branch and loop structures can update the PC register non-sequentially.

The memory component consists of 100 memory cells, and each can store either 4-digit instruction or data (ranging from 0000 to 9999). In the case of data underflow, store 0 into the AC and 1 into NF.

ISA:

Numeric Code	Description
19xx	Add the value currently stored in the accumulator to the value stored in the memory cell xx, store the result in the accumulator. AC = AC + mem[xx]. Set NF = 0.
1N00	Add all the values currently stored in the data registers (R_0 through R_N) to the value in the accumulator and store the result in the accumulator. AC = AC + R_0 + R_1 ... + R_N. N can be 0 to 5. Set NF = 0.
29xx	Subtract the value currently stored in the accumulator from the value stored in the memory cell xx. Store the result in the accumulator. AC = mem[xx] - AC. Set NF = 1 and AC = 0, if the subtraction underflows.
2N00	Subtract the sum of values currently stored in the data registers (R_0 through R_N) from the accumulator and store the result in the accumulator. AC = AC - (R_0 + R_1 ... + R_N). N can be 0 to 5. Set NF = 1 and AC = 0, if the subtraction underflows.
30xx	Store the contents of the accumulator to the memory cell xx. mem[xx] = AC
59xx	Load the value stored in the memory cell xx into the accumulator. AC = mem[xx]

5Nxx	Load the value stored in the memory cell xx through the memory cell xx+N into the data registers (R_0 through R_N). R_0 = mem[xx], R_1 = mem[xx+1], ... N can be 0 to 5.
60xx	Set the PC register to memory cell XX.
70xx	Set the program counter to the value xx if the accumulator (calculator) holds the value 0. AC = 0 => PC = xx.
71xx	Set the PC register value to xx if the accumulator is 0 or positive. This instruction entirely depends on the negative flag set by an underflow on SUBTRACT. NF = 0 => PC = xx.
9001	Take input, store in accumulator.
9002	Output the value in the accumulator.
9003	Dump the state. Call println!("{}", state). Useful for debugging.
0000	Stop working/end the program.

Execution:

To execute a program, perform the following steps:

1. Check the Program Counter for the memory cell that contains a program instruction.
2. Fetch the instruction from the memory cell with that number.
3. Decode the instruction based on the table above.
4. If the instruction utilises data stored in another memory cell, then use the address field to find the address of the memory cell for the data it will work on
5. Fetch the data (from the input, accumulator, or memory cell with the address determined in step 4)
6. Execute the instruction based on the opcode given
7. Yield the CPU resources and Branch or store the result (in the output, accumulator, or memory cell with the address determined in step 4)
8. Update the program counter and fetch the next instruction to repeat the cycle or halt.

Evaluation:

A sample test program (prog.lmc) is provided to you in the starter code to test your code. We will test your implementation on other similar programs.