## RISC-V SIMULATOR

**Design document:**

This document describes the design of the Phase2.py , a RISC-V Simulator.

# **How to execute a program:**

**Use Command:**

$python3 Phase2.py “machine\_file\_name.mc” # $python3 Phase2.py fact.mc

**Input:**

Input to the simulator is .mc file that contains the encoded instruction and the corresponding address at which instruction is supposed to be stored, separated by space. For example:

0x0 0x00500513

0x4 0x008000EF

0x8 0x04000463

**While running the file:**

When you run the file,

1. First it will ask the user whether to use Pipelining or non pipelining (Knob1)
   1. Enter 0 for Non pipelining or 1 for Pipelining
2. Next , it will ask whether to use stalling in pipelining or not.(knob2)
   1. Enter 0 for if pipelining is expected to work with stalling. And 1 for Data forwarding..
3. Next it will ask whether to print values of the registers at the end of each cycle or not.(knob3)
   1. Enter 0 for not printing and 1 for printing values of registers at the end of each file.
4. Next it will ask whether to Enable/disable printing information in the pipeline registers at the end of each cycle(similar to tracing), along with cycle number or not(knob4)
   1. Enter 0 for not printing and 1 for printing.
5. Next Knob is to enable knob 4 for any specific instruction.With this feature we will be able to see the pipeline registers information for a particular instruction of our interest. Here, the instruction can be specified as a number (example, if the instruction we are interested in is the 10th instruction in the input program, 10 will be taken as input).

Then the code will run and give output accordingly.

**How the Simulator works and Output format:**

The simulator reads the instruction from instruction memory, decodes the instruction, reads the register, executes the operation, and writes back to the register file. The instruction set supported is the same as given in the lecture notes.

If the user, selected to print values of register in each stage , output will be as shown below.

* Fetch prints:
  + “FETCH:Fetch instruction 0xE3A0200A from address 0x0”
* Decode
  + “Decode -> operation : addi ,source register 1: 5 ,Immediate: 000000000101 ,destination register : 7”
  + “DECODE: Read registers R2 = 10, R3 = 2”
* Execute
  + “EXECUTE: ADD 10 and 2”
* Memory
  + “MEMORY:No memory operation”
* Writeback
  + “WRITEBACK: write 12 to R1”

# **Simulator flow**

There are two steps:

1. First memory is loaded with an input memory file.

2. Simulator executes instructions one by one.

.

**Test cases**

We tested the simulator with following assembly programs:

1. All test cases for simple ALU operations worked fine.
2. There may be some problem with control instructions.