**PHASE 1 DOCUMENTATION**

**SINGLE CYLCE RISC-V INSTRUCTION SET**

This document describes the design aspect of RISC-V Simulator, a GUI-based functional simulator for a Single cycle RISC-V Instruction set.

**Input Mechanism**

Input to the RISC-V Simulator is a .mc file that contains the machine code from the venus simulator Dump end line character is a must in the .mc file.

Corresponding address to a machine code is not to be given in .mc file it will be mapped in code automatically.

**Example :**

**00a00193**

**00001437**

**00000293**

**00000313**

**0032de63**

**00640433**

**00542023**

**40640433**

**00128293**

**00430313**

**fe0004e3**

**00000293**

**00000393**

**0032d863**

**005383b3**

**00128293**

**fe000ae3**

**00640433**

**00742023**

**40640433**

**Note:** end line character at the end of the .mc file represents termination in running the RISC-V simulator.

**Output Mechanism**

output.txt – This file shows the updated value of Registers, Memory, Instruction name, and PC after each instruction execution..

registers.txt – This shows the value of all Registers after the code is executed.

memory.txt – This shows the value stored in memory after the code is executed.

**Simulator Design**

**DATA STRUCTURE**

There are black boxes defined to simulate all the components of RISC-V process in components.cpp and there is functions.cpp in this code where the required functions are defined namely fetch, decode, execute, memory\_access, and write\_back are declared. This functions just uses components defined in components.h file.

**SIMULATOR FLOW**

There are two broad steps:

1. First, the load\_program\_memory function will read instruction from .mc file and update a map which will map PC to hex string of instruction.
2. Secondly, the Simulator executes the instructions one by one.

The second step is executed by using a while loop. While the PC points to a valid instruction, the body of the while loop is executed otherwise the while loop is terminated.

The body of the while loop lines up the execution of the five stages of Instruction Execution i.e. **Fetch**, **Decode**, **Execute**, **Memory Access** and **WriteBack**, these stages are written in functions.cpp.

**FETCH**

In this stage, we are converting hex string which is mapped by PC value to a vector of int which is binary form of our string. We will also update adder that will increment PC by 4.

**DECODE**

In this stage we will decode opcode, func3, func7, immediate values, rs1, rs2, rd according to instruction fetched. We will update few global variables in decode like alu.operation, mux\_op2select.selectline, mux\_brancTargetSel.selectline, mux\_isbranch.selectline, etc.

We will also update mux\_branctargetsel, mux\_op2select, branch adder, wb adder.

**EXECUTE**

This is very easy we will first run the ALU black box after running this just alu.output() will give us output of ALU unit.

Then we will update Branch control unit and then populate isbranch mux

**MEMORY ACCESS**

In this stage we update which address will be accessed and if we have to write in memory then just call mem.data\_write function.

Then we will update resultselect\_mux.

**WRITE BACK**

In this stage we will just update rd register if regs.rfwrite is true.

And finally we will update PC by mux\_isbranch.output();

**Credits :**

1. Fetch : Vipul, Darshan, Keshav, Vikalp
2. Decode : Vipul, Darshan, Keshav, Vikalp
3. Execute : Vipul, Darshan, Keshav, Vikalp
4. Memory Access : Vipul, Darshan, Keshav, Vikalp
5. WriteBack : Vipul, Darshan, Keshav, Vikalp
6. GUI : Vipul, Darshan, Keshav, Vikal