CS204 Project Phase 1

**Functional simulator for a subset of RISC-V instruction set**

This document describes the design aspect of a GUI-based functional simulator for a subset of RISC-V instruction set.

**Input/Output Mechanism**

Input to the simulator is a “data.mc” file that contains the encoded instructions and their corresponding addresses separated by a space along with data values and their corresponding addresses separated by a space.

*Example:*

0x0 0xE3A0200A

0x4 0xE3A03002

0x8 0xE0821003

.

.

.

0x10000000 0x2

0x10000004 0x8

The instructions segment starts at 0x00000000 and data segment starts at 0x10000000, stack segment at 0x7FFFFFFC and heap segment at 0x10007FE8 in memory.

Our simulator then reads instructions from the instruction memory, decodes the instruction, reads from register file, executes the operation, reads/writes data to the memory and then writes back to the register file.

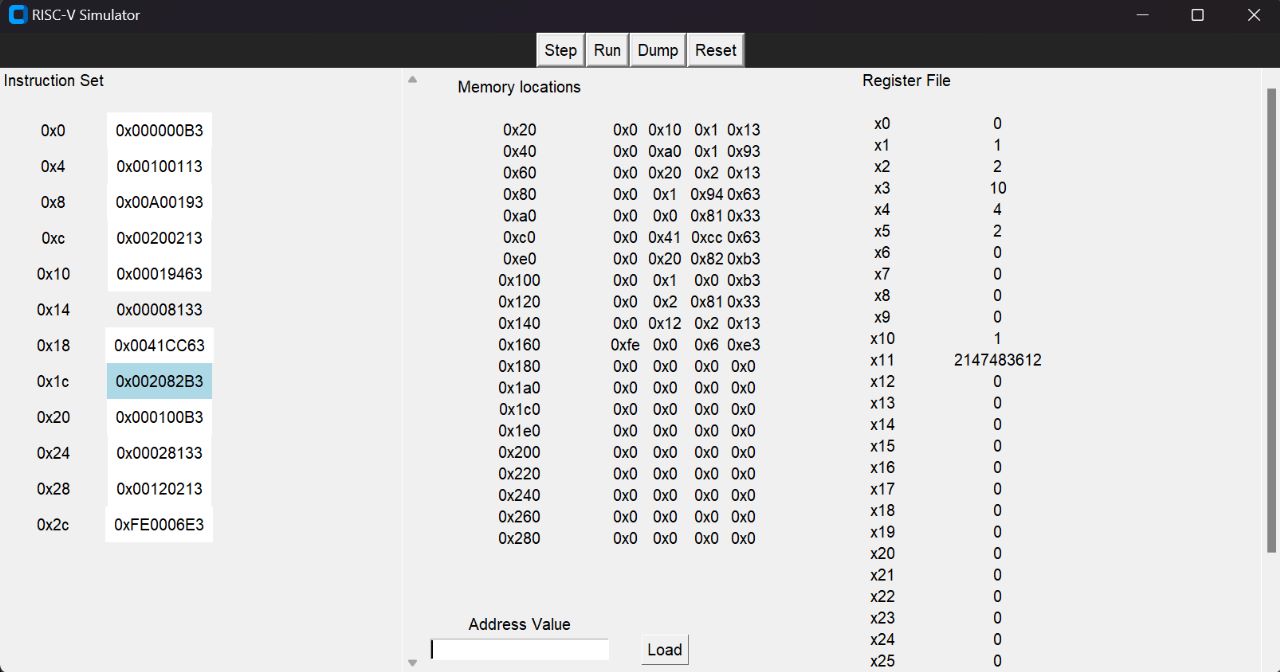
Our GUI has three sections, namely Instruction, Memory and Register.

Instruction Section shows all the instructions stored in the instruction memory along with the address of the instruction.

Memory Section shows the data values stored in the memory along with the search functionality to search value at a particular address.

Register File shows values stored in all 32 registers.

The GUI updates after every instruction.



**Design Of Simulator**

**Data Structure**

We have implemented the Simulator in Python. Entire Memory is declared as bitarray of size 2^32 using bitarray library. Register File is a list of size 32 with initial values set as the same as in venus simulator. Controls Signals are variables used as integers storing value 1,2,3.. for different controls possible.

**Simulator Flow**

We have the following files in our simulator:-

* main.py:- Main file that calls methods in all other files to
* gui.py:- Supports GUI
* data.mc :- Given as input to the simulator
* processor.py:- Handles the working of Processor. All the stages of processor are defined as methods in Processor class.
* control.py:- Handles all the control signals.
* Hardware:-
  + ALU.py:- Handles the working of ALU and gives ALU Result
  + memory.py:- Handles the memory
  + register.py:- Register File

Following are the steps in the execution of the simulator using above files.

1. main.py creates an instance window of the Window class declared in gui.py

and then its methods are called in main.

1. Then, Processor object is created and its run method is called.
2. In processor.py instructions are then read from data.mc and loaded in desired location in memory.
3. For each instruction the methods fetch(), decode(), execute(), memory\_access() and write\_back() are called until we reach the end of the instructions.
4. Environment class supports all attributes of current instruction e.g rs1, rs2, opcode which are used during entire program.
5. PC is updated during the execution of each instruction.
6. The Processor object uses ALU class for ALU operations, Memory class for reading and writing data into the memory and RF Class for updating Register file. Control attribute of processor is an object of Control class and generates control signals Rs2SelectOp, ALUOp, isBranch, ResultSelect\_gen etc.
7. These control signals are used in processor class to get the output of muxes in the implementation of Simulator.
8. In the GUI, we have 4 buttons:-
   1. Step:-On pressing Step next instructions is run and GUI window is updated.
   2. Run:-On pressing Run, all the instructions in instruction memory are run and GUI window is updated as per the result.
   3. Reset:-On pressing Reset, everything is reset and all values are set to initial.
   4. Dump:- On pressing Dump, our memory is written in memout.txt.

**Implementation of Fetch, Decode, Execute, Memory Access, Write Back**

1. **Fetch:** Current instruction is read from the memory for address given by PC attribute of processor object.
2. **Decode:** The instruction obtained from fetch() is used and opcode, rs1, rs2, rd, imm, immB, immJ, immS, immU, funct3, funct7 are calculated using the reference card. Using funct3, funct7 and opcode we determine the type of instruction which is attribute of control attribute of processor object.
3. **Execute:** Firstly, immediate are sign extended. Then ALUOp control signal is calculated and operands are sent to ALU and result is stored in ALUResult.

Also BranchSelect control signal is generated.

1. **Memory Access:** Depending on ALUOp data is read from memory/written into memory
2. **Write Back:** Data is updated in registers depending on type of instruction. Also the PC is updated for next instruction using isBranch control signal.

**Testing Of Simulator**

We test the simulator using the following assembly programs :-

1. Fibonacci Number Program
2. Sum of the array of N elements. Initialize an array in first loop with each element equal to its index. In second loop find the sum of this array, and store the result at Arr[N].
3. Bubble Sort Program