Design Document: RISC-V SIMULATOR

The document describes the design aspect of our RISC-V simulator made using Python as a programming language.

# Input/Output

## Input

Input to the simulator is data.mc file that contains the encoded instruction and the corresponding address at which instruction is supposed to be stored, separated by space as a seperater.

For example:

0x0 0x07B00293

0x4 0x01300213

0x8 0x0480006F

0xc 0xFF010113

0x10 0x00112023

## Functional Behavior and output

The simulator reads the instruction from instruction memory, decodes the instruction, read the register, execute the operation, and write back to the register file. The instruction set supported is same as given in the lecture notes of CS-204.

The execution of instruction and fetching continues till it reaches instruction “0xffffc” in the input file. In other words as soon as instruction reads “0xffffc”, simulator stops and writes the updated memory contents on to a memory text file.

The simulator also prints messages for each stage, for example for the third instruction above following messages are printed.

We have added **GUI** for this simulator :

A window will appear

| 2. Firstly click on Assemble

| If you want to run the code step by step:-

| >>>>Click on run button on the gui

| -In output log, will get the output after each step

| -Like the value stored in the register, opcode,immediate, Instruction Type etc.

| If you want to run the whole code together:-

|

|

| >>>>Click on run button

| -And the whole code will get executed

We have mentioned further details in the readme file.

* Fetch prints:
  + “FETCH:Fetch instruction 0xE3A0200A from address 0x0”
* Decode
  + “DECODE: Operation is ADD, first operand R2, Second operand R3, destination register R1”
  + “DECODE: Read registers R2 = 10, R3 = 2”
* Execute
  + “EXECUTE: ADD 10 and 2”
* Memory
  + “MEMORY:No memory operation”
* Writeback
  + “WRITEBACK: write 12 to R1”

# Design of Simulator

## Data structure

Registers, memories, intermediate, PC, RZ , Instruction register , clock,muxy, used for each stage of instruction execution are declared as global variables.

For the implementation of registers and memories two separate dictionaries are used, while for storing the instructions instruction\_dict and for storing data in memory data\_dict is used. Our memory is Byte Addressable.

## Simulator flow:

There are two steps:

1. First memory is loaded with input memory dictionary and each instruction of the input file is stored in instruction\_dict and memory data in data\_dict.
2. Simulator executes instruction one by one.

For the second step, there is infinite loop, which simulates all the instruction till the instruction sequence reads “0xffffc”.

Next we describe the implementation of fetch, decode, execute, memory, and write-back function.

### FETCH:

In this step the input file data.mc is passed to the function fetch\_file when called and two dictionaries are returned by this function which is stored in instruction\_dict and data\_dict for further stages.

### DECODE:

As for each instruction stored in instruction register each instruction is decoded according to the opcode it has. Further according to each of the R,I,S,SB,U,UJ type instruction the rs1, rs2, immediate value , the type of operation , and other necessary details are stored and returned in the decoded\_info dictionary which is further used in the Execution step .All sign extensions required are taken care of for the execution step in this stage.

### EXECUTE:

The decoded\_info and the pc\_temp (temporary program counter are passed to the execute function to execute the necessary operations according to the field. In this step according to the type of instruction information is stored in the decode\_info the function to execute that appropriate function is called off and the executed operation value is returned along with a pointer to the execution which we call as PC immediate which stores value of PC if any change occurs in it.These returned values are further stored in RZ and pc\_final variable for the next step.

### MEMORY:

In this step the memory function is called off and according to the operation which requires the use of memory such as lb,lh,ld,jal,jalr the values in the memory dictionary are read / updated and the updated memory dictionary along with other necessary parameters are returned along with updated rz if necessary. The value the memory step returns in this step is stored in muxy.

### WRITEBACK:

If any operation requires the updating of value given in the destination register(rd) is updated by calling the write back function with the passing of necessary parameters muxy and register\_dictionary.

At last pc is updated to pc\_final value and this process repeats until 0xffffc is encountered.

TODO

# Test plan

We test the simulator with following assembly programs:

* Fibonacci Program
* Factorial Program
* Bubble sort Program