Design Document: Functional Simulator for 32bit RISC-V ISA

The document describes the design aspect of the functional simulator made by us as a part of CS204 project.

# -----Input/Output-----

## Input

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

0x0 0xeaa0200a

0x4 0xe3a03002

0x8 0x00821003

## 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.

The execution of instruction continues till it reaches end of the “machinecode.txt” file. 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.

* Fetch prints:
  + “FETCH:Fetch instruction 0xe3a0200a from address 0x0”
* Decode
  + “DECODE: Operation is ADD, first operand x2, Second operand x3, destination register x1”
  + “DECODE: Read registers x2 = 10, x3 = 2”
* Execute
  + “EXECUTE: ADD 10 and 2”
* Memory
  + “MEMORY:No memory operation”
* Writeback
  + “WRITEBACK: write 12 to x1”

# -----Design of Simulator-----

## Data structure

Python Lists – register, stack\_array , memory\_array

## Simulator flow:

There are two steps:

1. First memory is loaded with input memory file.
2. Simulator executes instruction one by one.

Next we implemented fetch, decode, execute, memory, and write-back functions.

# Testing

We tested the simulator with following assembly programs:

* Rec\_Fibonacci Program
* Rec\_Factorial Program
* Rec\_BubbleSort Program
* Simple\_add Program
* Loop testing

|| KINDLY REFER “README” FILE FOR MORE DETAILED DESCRIPTION OF THE SIMULATOR. ||