Design Document: Functional Simulator for Subset of RISCV instruction set

The document describes the design aspect of myRISCVSim, a functional simulator for a subset of RISCV instruction sets.

Input/Output

Input

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

```
0x0 0xE3A0200A

0x4 0xE3A03002

0x8 0xE0821003
```

Functional Behavior and output

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.

The execution of instruction continues till it reaches instruction "swi 0x11". In other words as soon as instruction reads "0xEF000011", the simulator stops and writes the updated memory contents onto a memory text file.

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

- 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 output for each stage of instruction execution are declared as global static. Being static, the variables are not visible outside the file, thus making the data encapsulated in the myRISCVSim.py.

Simulator flow:

There are two steps:

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

For the second step, there is an infinite loop, which simulates all the instructions till the instruction sequence reads "SWI 0x11".

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

Test plan

We test the simulator with following assembly programs:

- Fibonacci Program
- Factorial Program

- Simple addition Program