Design Document: Functional Simulator for Subset of RISC-V instruction set

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

Input/Output

Input

Input to the simulator is .asm 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 the instruction memory, decodes the instruction, read the register, execute the operation, accesses memory if needed and write back to the register file. The instruction set supported is the same as given in the problem statement of Phase1.

The execution of instruction continues until it reaches the last instruction. When the user clicks 'EXIT', simulator stops and writes the updated memory contents on to output.mc.

Along with the execution of an instruction in steps, the simulator also provides messages what it is doing in each stage as an update in all the structures that would change.

```
Ater fetch: PC: 4, IR: 00000000000000000000000000110011  RA: None  RB: None RZ: None RY: None RM: None

Ater decode: PC: 4, IR: 00000000000000000000000000110011  RA: 0  RB: 0 RZ: None RY: None RM: None

Ater alu: PC: 4, IR: 00000000000000000000000000110011  RA: 0  RB: 0 RZ: 0 RY: None RM: None

Ater memory_access: PC: 4 , IR: 00000000000000000000000000110011  RA: 0  RB: 0 RZ: 0 RY: 0 RM: None

Ater write_back: PC: 4 , IR: 00000000000000000000000000110011  RA: 0  RB: 0 RZ: 0 RY: 0 RM: None
```

Cycle: 1

Design of Simulator

Data structure

Registers, memories, intermediate structure's output for each stage of instruction execution is declared as global static. Being static, the variables are not visible outside the file, thus, make the data encapsulated in the Phase2.py.

Simulator flow:

There are two steps:

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

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

Fetch:- Word is read from memory at the address of PC and PC is incremented.

Decode:- Instruction's format is identified first based on opcode,funct3,funct7, then register addresses are extracted from IR, then register values are read from the register file. And the operation is set which is going to be executed in ALU.

Alu:- Arithmetics is performed

Memory:- Effective address was generated in the alu will be substituted in MAR, and memory is accessed at MAR.

Write-Back:- Register is updated now if needed.

Test

The simulator is tested on Assemble code of

- Bubble Sort Program

- Factorial Program

- Fibonacci Series Program