Design Document: Functional Simulator for RISCV instruction set

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

# Input/Output

## Input

Input to the simulator is a .mc 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, 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 the machine code “0x00000020”. simulator stops and writes the updated memory contents on to a data\_out.mc file.

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

* Fetch prints:
  + “FETCH: instruction\_register : 2325063”
  + PC : 20
* Decode
  + Opcode :99
  + Operand1: 6
  + Operand2: 6
* Execute
  + “ALUOP : 32”
  + “ALUResult: 0”
* Memory
  + “MemOp: 0”
* Writeback
  + “RFWrite : 0”

# 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, make the data encapsulated in the myRISCVSim.cpp.

For Instruction memory we used a char array and for the all-other memory we used map.

## Simulator flow:

There are two steps:

1. First memory is loaded with input memory file.
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 the machine code “0x00000020”.

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

**Implementation of fetch**:

In the fetch function the instruction is fetched from the memory .The program counter is used to retrieve the instruction from the memory location pointed by the PC the fetched instruction is stored in the instruction register.

**Implementation of Decode**:

In the decode function first we extract bits we need from the instruction register and assign them as per the format of the type of instruction. By using the opcode and func3 obtained we determine the type of the instruction and we generate all the necessary control signals needed for the data path.

We also determine operands and immediate required for the instruction. Values of the operands are retrieved from the register file.

**Implementation of Execute**:

In the execute function all the arithmetic and logical operations are executed. By the control signal for ALU, generated from the decode function the required ALU result will be selected.

**Implementation of Memory Access**:

In the memory access function, the memory is accessed if the instruction requires it. Mainly used to load or store the data from the memory which is decided by the control signals obtained from the decode stage.

**Implementation of Write Back**:

By considering the control signals RFwrite the function writes backs the required result in the destination register. It also updates the pc value according the isBranch control signal.

# Test plan

We test the simulator with following assembly programs:

* Fibonacci Program.
* 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 A[N].
* Bubble sort program.