Design Document: Functional Simulator for RISCV instruction set

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

# Input/Output

## Input

Input to the simulator is .mc file that contains the encoded instructions. Instructions will be stored from address 0x0 For example:

0xE3A0200A

0xE3A03002

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 instruction “swi 0x11”. In other words as soon as instruction reads “0xEF000011”, 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 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.

## 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 “SWI 0x11”.

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

**Fetch**

instruction is fetched from the instruction memory using the value of PC using read\_word() method. The instruction is stored in a global variable instruction\_word

**Decode**

If instruction\_word = 0xEF000011, then swi\_exit() method is called. Which write the data memory to data\_out.mem.

Otherwise instruction is decoded by looking at opcode, fun3, fun7, etc. Most of the control variables are modified at this stage depending on type of instruction. The selection of operand1, operand2, rd, rs1, rs2, imm, ALUOp, etc also done here.

**Execute**

According to the value of ALUop, ALU operation is performed using operand1 and operand2 using an ALU unit. Result is stored in another global variable ALUResult. The branch target address is computed and selected at this stage. Also isBranch

**Memory Access**

According to the value of MemOp(decides read, write or no memory operation) and Mem\_b\_h\_w (decides byte, half\_word, or word) data is read from or write to the memory. Memory is implemented by two dictionaries (instruction memory and data memory). Memory address will be the ALUResult

**WriteBack**

According to the value of RFWrite the result is written into rd. Result is determined by resultSelect.

# 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 Arr[N].
* Bubble sort program