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

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

# Input/Output

## Input

Input to the simulator is .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 0x00528293

0x4 0x00530333

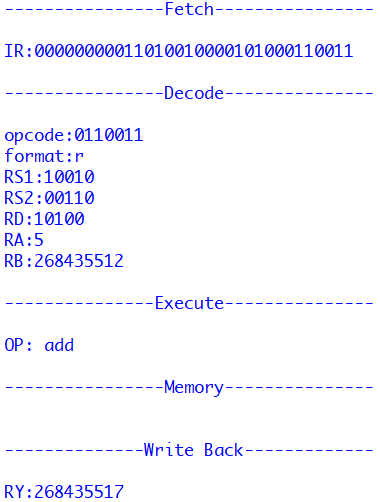
0x8 0x00738393

0xc 0x00740433

## 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 the last instruction. In other words as soon as the whole .mc file is the read, the simulator stops and writes the updated memory contents onto a memory list.

The simulator also prints messages for each instruction, For example:

# Design of Simulator

## Data structure

Objects of registers, memories are created every time a new .mc file is processed and the same object is updated after every instruction is processed. Values of memory and register are printed before and after all the instructions are executed. As classes are used hence data is not visible outside the file, thus, make the data encapsulated in the code file.

## Simulator flow:

There are two steps:

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

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

Fetch: It fetches instructions one by one starting with PC 0x0 and converts it to binary format and sends it to decoder also it increases PC by 4 and thus picking the next instruction at 0x4 and so on.

Decode: It decodes the instructions basically it figures out the format of instruction by getting opcode and then decodes further based on format. This decode function is divided in different decode function based on format, further in every decode function it figures out what type of function is called in that format using func3 & func7 and then sends RA, RB, R imm, PC, etc value to execute function to perform the action relative to the funct3 & funct7.

Execute: It contains the part where main alu functions take part depending on funct3 and funct7 received from decode. It calculates values as per the instruction needs.

Memory: It basically takes care of instruction where memory access is needed like lw, sw, etc. And updates the memory object created in the starting.

Write-Back: Once all the calculation is done, finally data needs to be written back onto the register. This function basically does that. And updates the register object created in the starting.

# Test plan

We test the simulator with following assembly programs:

* Simple Add Program
* Fibonacci Program
* Sum of the array of N elements. Initialize an array in the first loop with each element equal to its index. In the second loop find the sum of this array, and store the result at Arr[N].