Design Document: Functional Simulator

The document describes the design aspect of CS204 Phase 3

# How to execute a program:

## Use Command:

Compilation

**# Non-pipelined version**

g++ main.cpp alu\_unit.cpp control\_unit.cpp dec2bin.cpp global\_variables.cpp immediate.cpp memory\_read\_write\_funcs.cpp myRISCVSim.cpp registerfile.cpp -o output.exe

**# Pipelined version (includes forwarding and branch prediction)**

g++ main.cpp alu\_unit.cpp BTB\_operations.cpp control\_unit.cpp forwarding\_unit.cpp dec2bin.cpp global\_variables.cpp immediate.cpp memory\_read\_write\_funcs.cpp myRISCVSim.cpp registerfile.cpp -o output.exe

Execution:

./output.exe <program\_file.mc>

Example: ./output ../test/bubble\_sort.mc

Where in brackets we have to write the name of the file which contain assembly code of our desired program.

# 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 0x00500513

0x4 0x008000EF

0x8 0x04000463

## 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 execution of instruction continues till it reaches instruction “0xFFFFFFFF”, simulator stops and print updated registers and memory as well as updated memory contents on to a memory .mc 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 at address 0x0”
* Decode
  + “DECODE: Operation is ADD, first operand R2, Second operand R3, destination register R1”
* Execute
  + “EXECUTE: ADD 10 and 2”
* Memory
  + “MEMORY: if (memory operation is used) : accessed memory location at ‘memory address’ “

Else : No memory operation

* Writeback
  + “WRITEBACK: if(writeback operation exists): write 12 to x[1]”

else: no writeback

# Design of Simulator

## Data structure

We used array for the registers and map for the memory.

## 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 “0xFFFFFFFF”.

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
* Bubble-sort
* Factorial
* Some other test cases