CSCE 2303 – Computer Organization and Assembly Language Programming
Summer 2024

Project 1: RV32IC Simulator

1. Background

An ISA simulator (ISS) is a program that accepts the machine code of a compiled/assembled application and mimics what real CPU hardware would do to execute the application instructions.

RISC-V (pronounced "risk-five") is an open-source instruction set architecture (ISA) based on established reduced instruction set computing (RISC) principles. The project was started in 2010 by researchers in the CS Department at the University of California, Berkeley.

2. Requirements

You are required to develop ISS for RISC-V RV32IC Base Integer Instruction Set with support for instruction compression (bonus). You may use any programming language of your choice (C/C++ is preferred and a C++ skeleton is given). Your program must:

- Assume that the program memory (used for the text section) starts at 0x00000000 and has a size of 64kbyts.
- Assume that the data memory (used for the data section) starts at 0x00010000 and it has a size of 64kbytes.
- Read RV32I machine code file. The file is just a binary file that contains the machine code of a program instructions. The first 4 bytes (32-bit word) in the file belong to the first instruction; the 2nd 4 bytes belong to the 2nd instruction, etc.
- The first instruction is assumed to be at location 0x00000000 in the main memory.
- Optionally, read the application data section file.
- Decode each instruction machine code word, then translate it into a string that represents a true RV32I instruction to print it on the terminal.
- Execute the decoded instruction. The execution involves: modifying a register, modifying a memory location or performing an I/O operation (ecall).
- Print out the decoded instruction on the screen.
- Be invoked from the command line using the syntax:

  `rvsim <machine_code_file_name> [data_section_file_name]`
- Where:

  `<machine_code_file_name>` is the file name of the input machine code
- If you don't know how to pass parameters using the command line to a C/C++ program, read this.
- The simulator should be able to decode/execute all RV32i instructions covered in the lecture.
- The simulator should be able to execute the following ecall services:
  - Print an integer (a7=1): a0= integer to print,
  - Print a string (a7=4): a0= address of a null-terminated string to print,
  - Terminate execution (a7=10)

3. Bonus (10%)

Support all the RV32IC compressed instructions. Typically, every compressed instruction can be expanded into a non-compressed RV32I instruction. Hence, to support compressed instructions, just add code to recognize the compressed (16-bit) instruction word and then expand it into a non-compressed (32-bit) instruction word.

4. Guidelines

- Work in a group of 3 students
- The given skeleton is in C++. You may use any other programming language for your implementation like Python. If you are planning to do so, please contact Dr. Shalan first to get his approval.
- You need to construct test cases to ensure the correctness of your implementation.
- You have to use GitHub for development and you need to follow a good development flow. I will check the history and the contributions of each group member. Make sure that you follow a good workflow. Uploading files or sharing files using WhatsApp is not a part of a good workflow.
- Your GitHub repo. “readme.md” file will be used as your report; it should outline:
  - How to build the simulator
  - How to use the simulator
  - The simulator design.
  - The challenges you faced as well as any limitations your simulator has.
- Grade breakdown:
  - -10% for not using GitHub for development
  - 10%: The readme.md file
  - 30%: Your test cases. Use RARS or oak.js to create some. Every instruction must be tested.
  - 30%: Code organization including comments.
  - 30%: Offline test cases’ results.