CSCE2303 – Computer Organization and Assembly Language Programming
Spring 2024

Project 1: RISC-V RV32I Simulator

Project Overview: The goal of this project is to implement a functional RISC-V simulator capable of tracing the execution of RV32I instructions. This document details the requirements of the simulator.

Implementation language: Any general-purpose programming language. The resulting application can either be a console application or a graphical user interface (GUI) application (desktop, web-based, or mobile app) as a bonus feature.

Team Size: 2 to 3 students

Instruction Set Architecture (ISA): The simulator must support the RV32I base integer instruction set according to the specifications found here: https://riscv.org/technical/specifications. All forty user-level instructions (listed in page 130 of the RISC-V Instruction Set Manual – Volume I: Unprivileged ISA and explained in Chapter2 of the same manual) must be implemented as described in the specifications except ECALL, EBREAK, and FENCE instructions. Instead, your implementation should interpret any of these 3 instructions as a halting instruction that ends the execution of any program (by preventing the program counter from being updated anymore).

Simulator inputs:

1. *Assembly program:* The user should be able to input a program to be simulated. All input programs should be terminated by one of the 3 instructions interpreted as a halting instruction. The user should also specify the program starting address (where the program's first instruction should be loaded in the memory). The input program can be provided by the user in any format. If you find it easier, you can ask the user to enter the instructions of the program one by one. He can even specify the operands of each instruction one by one. Alternatively, you might allow the use to provide an input program by specifying a text file that contains the program's instructions.
2. *Program data:* The user should also specify any data required by the program to be initially loaded in the memory. For each data item both its value and memory address should be specified. This information can also be provided one by one or through a text file.

Simulation and simulation outputs: The simulator should start by reading the inputs provided by the user and then it should simulate the input program's execution by keeping track of the **program counter value**, the **register file contents**, and the **memory contents**. The program should repeatedly **output all these** values (after each instruction's execution) until the program ends.

Please note the following:

1. To keep track of program counter value, the register file contents, and the memory contents you need to **initialize** them. The program counter should be initialized to the program's starting address as specified by the user. All registers should be initialized to zeros. The memory should also be assumed to be empty except for the locations containing the instructions (which can be ignored unless you intend to implement the assembling to machine code bonus described below) and the locations containing the data values provided by the user.
2. Given that the memory address space is large (4 GBs), you will not be able to keep track of the memory contents by creating an array that can hold 4GBs of data. Instead, you will need to use a data structure that allows to record the contents of **relevant memory locations only** (along with the addresses of these locations of course). Relevant memory locations include all memory locations initialized through the user's input and all the memory locations modified by the program through store instructions.
3. Register 0 always contains the value 0. You will need to make sure that no instructions can modify it.

***Bonus* features:**

To get bonus marks, you can implement up to 2 bonus features from the list below:

1. Building the application as a GUI application (either desktop, web-based, mobile app).

2. Implementing and integrating an assembler that will convert the input program into machine code and use these values to properly initialize the corresponding memory locations (which will be displayed as part of the relevant memory locations output).
3. Outputting all values in decimal, binary, and hexadecimal (instead of just decimal which is assumed to be the default)
4. Add support for at least 5 pseudoinstructions (out of the many pseudoinstructions supported by RARS), and at least 3 directives (like `.data`, `.word`, and `.text`).
5. Add support for compressed instructions to effectively support the full RV32IC instruction set except for compressed instructions that do not map to RV32I instructions. The compressed instructions are also described in the official specifications mentioned above.
6. Add support for integer multiplication and division to effectively support the full RV32IM instruction set. The integer multiplication and division specifications can also be found in the document above.
7. Including a larger set of test programs (at least 6 meaningful programs) and their equivalent C programs.

Project Report: In addition to your **team member names and IDs**, the report should include:

1. A brief description of your implementation including any bonus features included.
2. Any design decisions and/or assumptions you made.
3. Any known bugs or issues in your simulator.
4. A user guide showing how to compile and run your simulator including a full simulation example step-by-step with screenshots.
5. A list of programs (and associated data if any) you simulated. You should at least provide 3 programs. The programs must cover all instructions supported and one of them at least must have a loop.
6. Optionally, you can include a section about your experience working on this project. This section will NOT affect your grade in any way.

General Guidelines

- Every group member must log all her/his activities in a journal (a text file). A journal entry may look like the following:

  ```
  April 17, 1:55PM: added support for sw, sh, sb instructions. Fixed issue
          in program counter update in case of conditional branching.
  ```

- Only one member of each group should submit on Canvas. The submission should consist of a single compressed folder (**zip** or **rar**) which must include the following:
  - A journal folder that contains the journal file of each team member.
  - A source code folder that contains the code you wrote using your chosen programming language.
  - A test folder that contains all input files (assembly and data files) used for testing.
  - A **PDF** report that contains the information described above.

Important Plagiarism notice: You must write your own code from scratch. Submissions based on others code will receive a grade of **zero** in the entire project and will be reported as an academic integrity violation (even if you understand every code line and even if the code is heavily re-factored/modified). Examples of such sources include (but is not limited to) code coming from the following sources: other teams, previous course offerings, open-source software, tutors, Internet, etc.

Deadline: You are required to submit the full project on **Monday April 22, 2024 (11:59 pm)**. You will also be required to make a short demo of your project. **After submission, a Google sheet with several 15-minute slots will be shared with you to allow you to reserve one slot for the demo**.

Grading Criteria

- This project is first of 2 equally weighted projects worth 30% of the course marks. Therefore, each project will account for **15%** of the course marks.
- **Bonuses**: Each bonus feature will count for 5% with a maximum of 2 bonuses worth 10%. Please do not be tempted to implement more than 2 bonus features, as this will cost you too much time.
- Deductions:
  - **-5%** for not following the required submission directory structure.
  - **-5%** per day for late submission (one day maximum).
  - **-100%** for plagiarized submissions.
- Group members may receive different grades based on their contribution to the project as determined by the submitted journals and the discussion during the demo.