

Assembly Project 1 report: RISC-V RV32I Simulator

Mohamed Abdelmohsen, Carol Michail, Sara Naseef

900201290, 900194282, 900202369

Department of Computer Science and Engineering

CSCE 2303 - Computer Organization and Assembly Programming

Professor: Nourhan Zayed

Spring 2023

# Table of contents

**1. Introduction 2**

**2. Design implementation and assumptions made 2**

**3. A step-by step user guide on how to run the simulator 3**

**4. Any known bugs or issues in our simulator 5**

**5. A list of programs we simulated 5**

**6. Conclusion 8**

**Introduction:**

In this project, we have been asked to code a RISC-V RV32I simulator. In the report, we will be discussing our design implementation and assumptions made, a step by step user guide on how to run our simulator, any known bugs or issues, and lastly we will be showing a list of programs we simulated ourselves. Our simulator is implemented in C++, and it reads RISC-V instructions from an assigned instructions file. It also reads all 32 registers from another file. The simulator then executes the given set of instructions and returns the results to the user. The simulator outputs the values of the registers that have been used while keeping unused registers with a constant value of zero. This simulator is designed to be very user friendly, the user would only need to update the instruction file, the registers file is a constant. After updating the instructions file, the code would do the rest of the work and output the simulation.

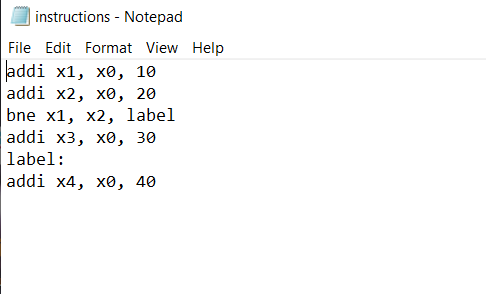
**Design implementation and assumptions made:**

* The use of input files: these make it a bit easier for the user to directly use the simulator. There is no need to use the console to enter the instructions. In case the user needed to edit one of the instructions in the file, they would only go back to the file and edit that one instruction instead of rewriting every instruction all over.
* Our design approach:

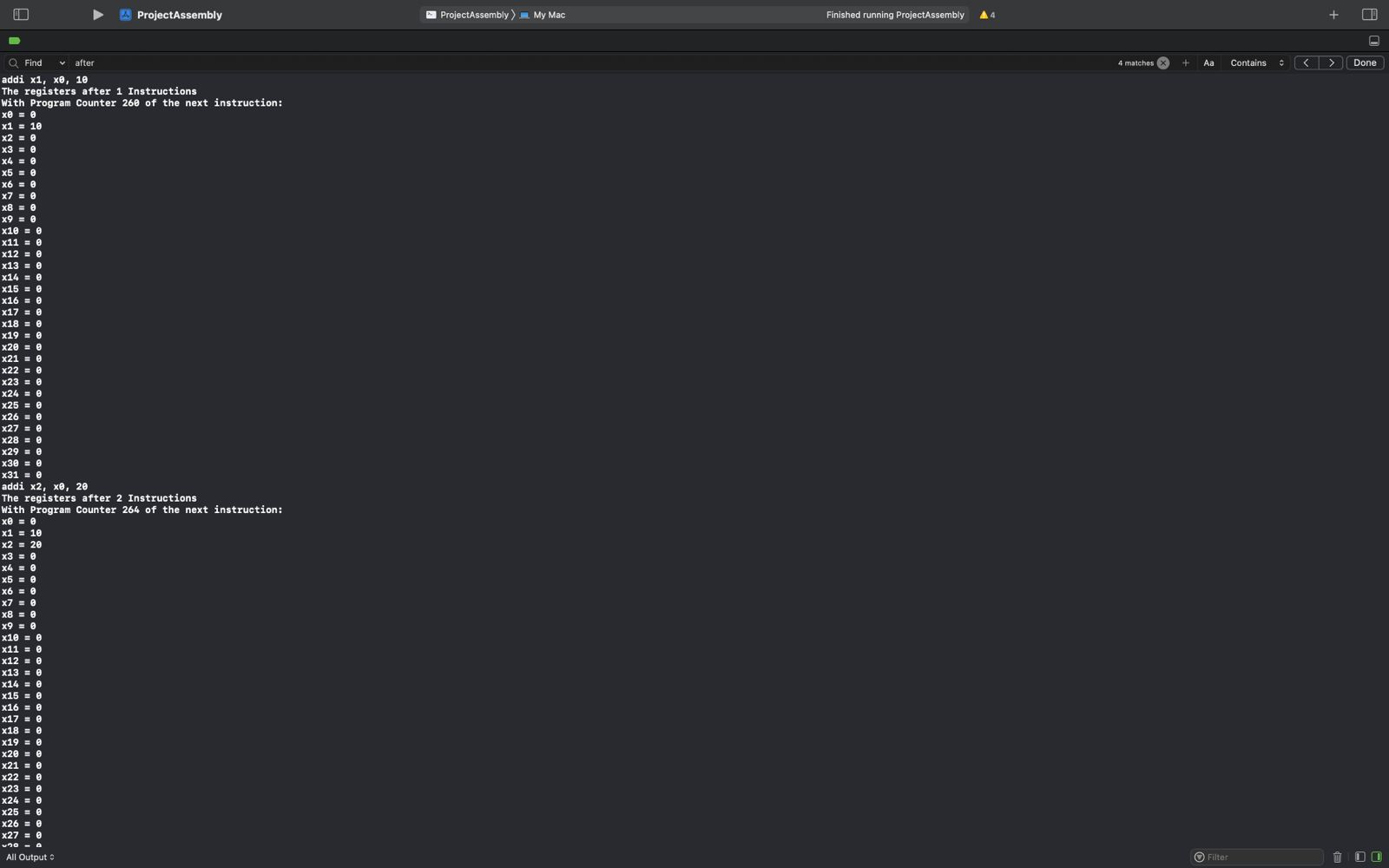
For the simulation of registers, we assigned them in a map. A map is a type of data structure that allows for key-value pairs to be stored and accessed efficiently. By using a map to represent registers, the program can quickly access values stored in specific registers using their corresponding keys. The values are stored in a way that includes both names for each register and contains its value. For the Instructions, we used a vector of pairs. By using a vector of pairs to represent instructions, the program can easily store and retrieve sets of commands along with their corresponding operands. For the memory, it is implemented using an array. A fixed size for the array was given to the memory beforehand. Also, we implemented each instruction as a function, so that these instructions are easily modified and the use of one doesn’t affect the use of another.

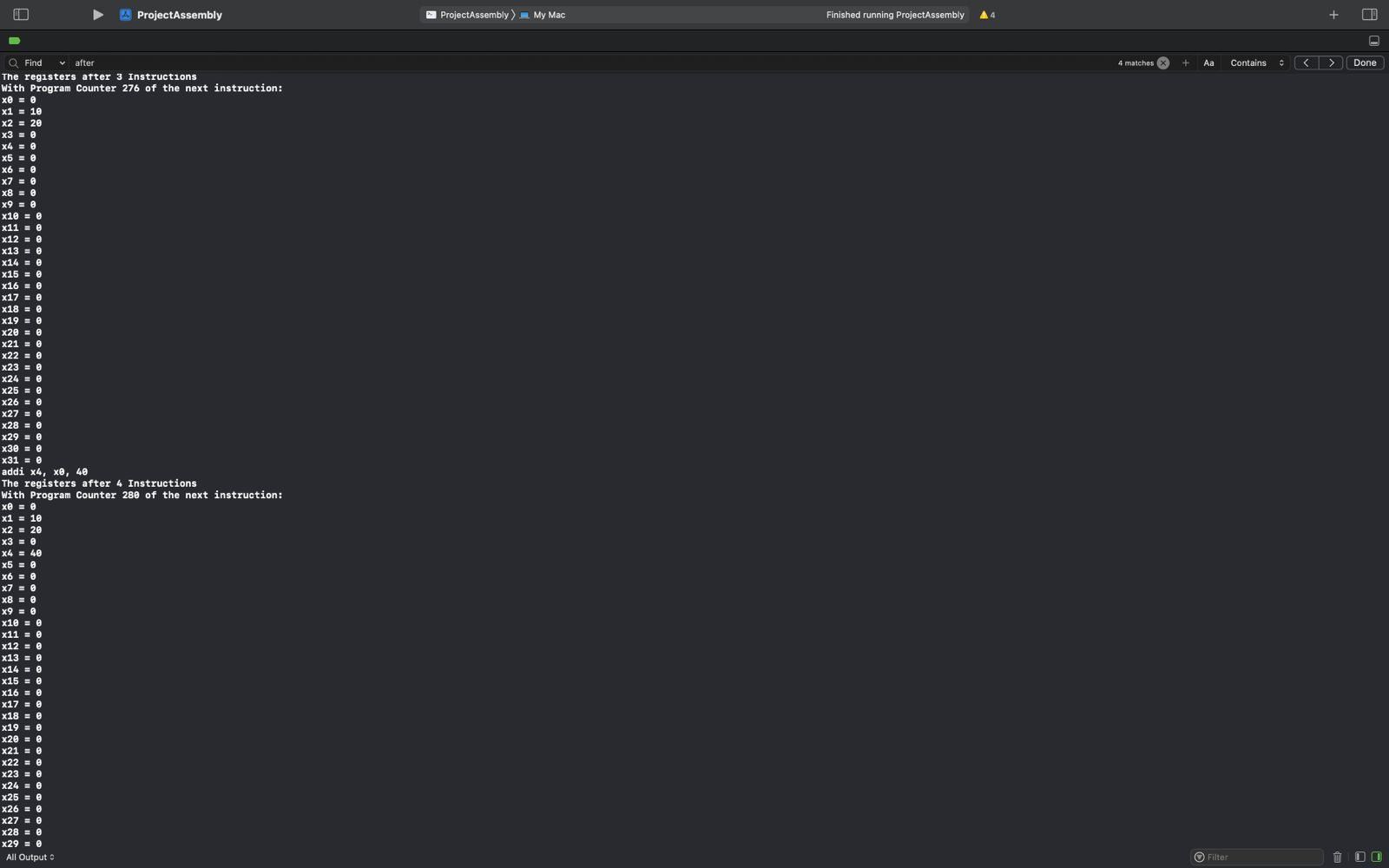
**Step by step user guide on how to run the simulator:**

* Write the instructions you need to simulate in the instruction file and save leave the instruction and registers file in the same file as the main.cpp file for the compiler to understand.



* Run the C++ code: Just press run on any compiler.
* Output will show all 32 registers with their values after each instruction use. Unused registers would remain zero.





**Any known bugs or issues in the simulator:**

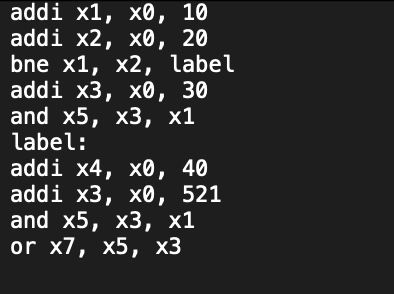
There has been issues with choosing the way of implementing the memory, lw and lb instructions have been very tricky to implement. The running of the simulator worked using the test cases we tried.

**A list of programs we simulated (with running code screenshots):**

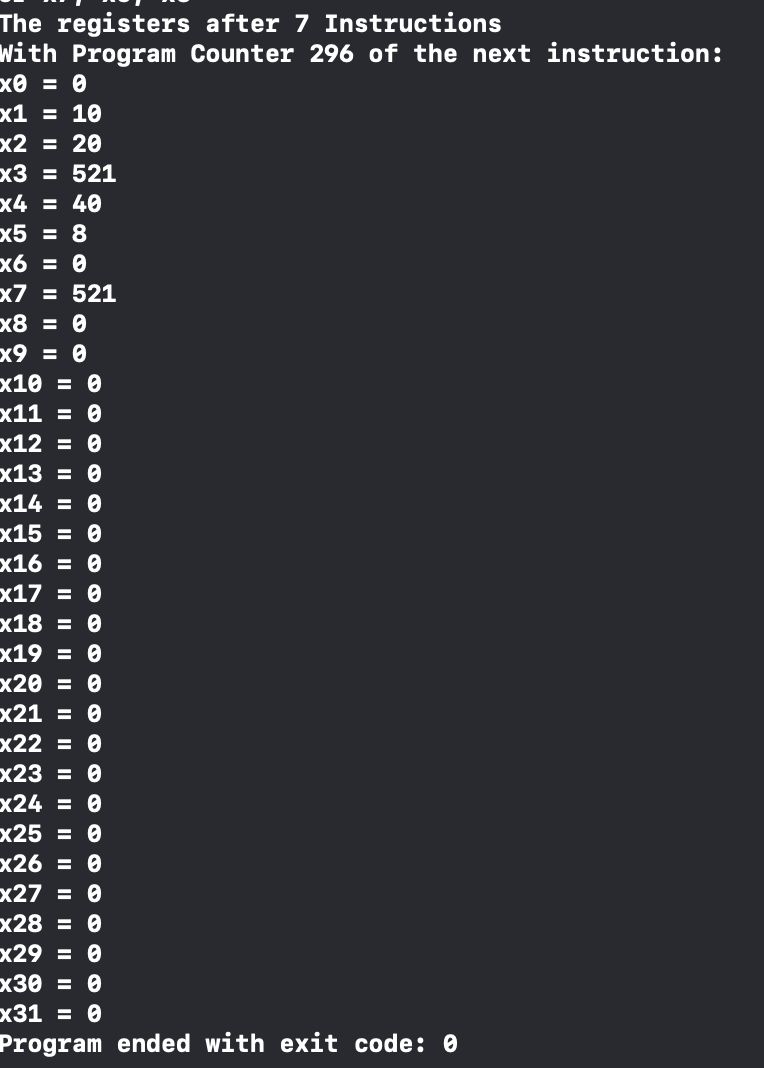
Here are 3 programs we simulated. We tired to make the programs all together cover most of the instructions we support in the simulator. Each program on its own doesn’t include all instructions.

Program 1:

Instruction file:

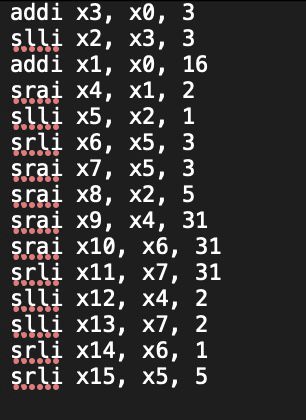


Output:

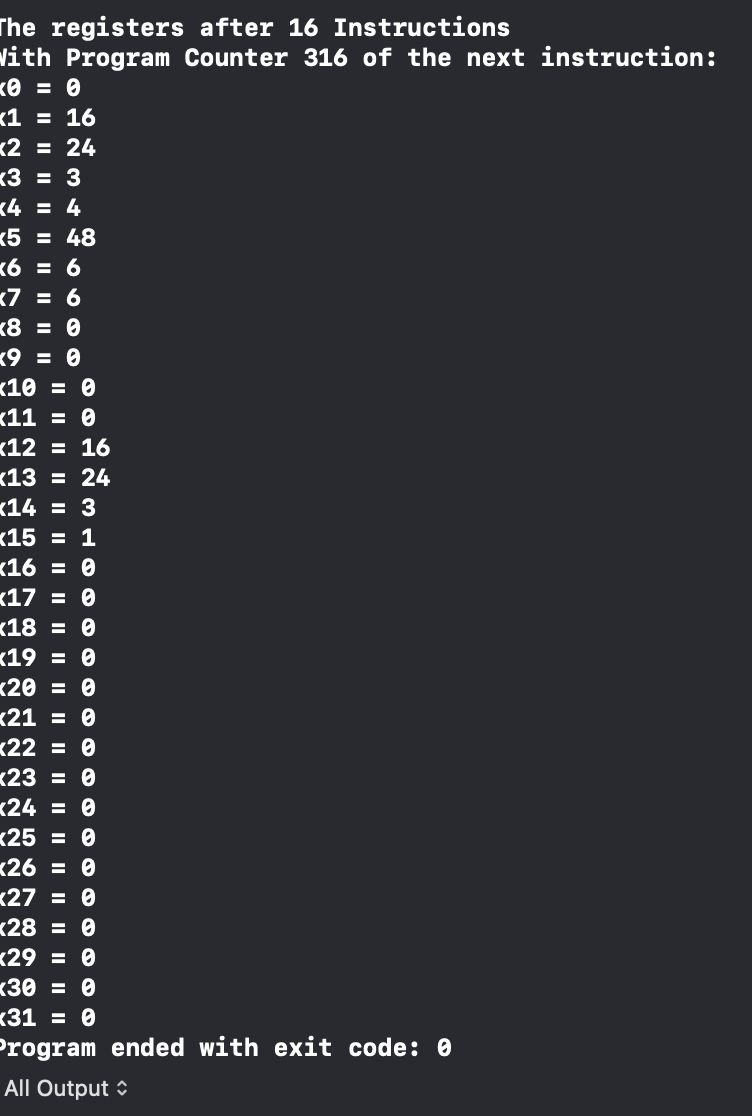


Program 2:

Instruction file:

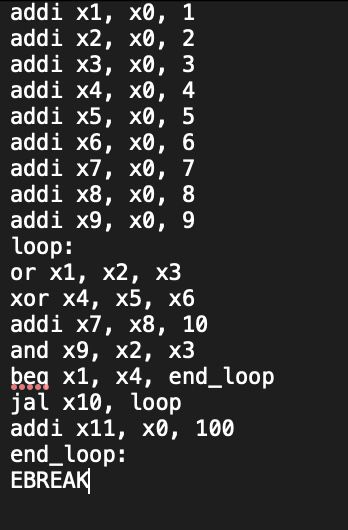


Output:

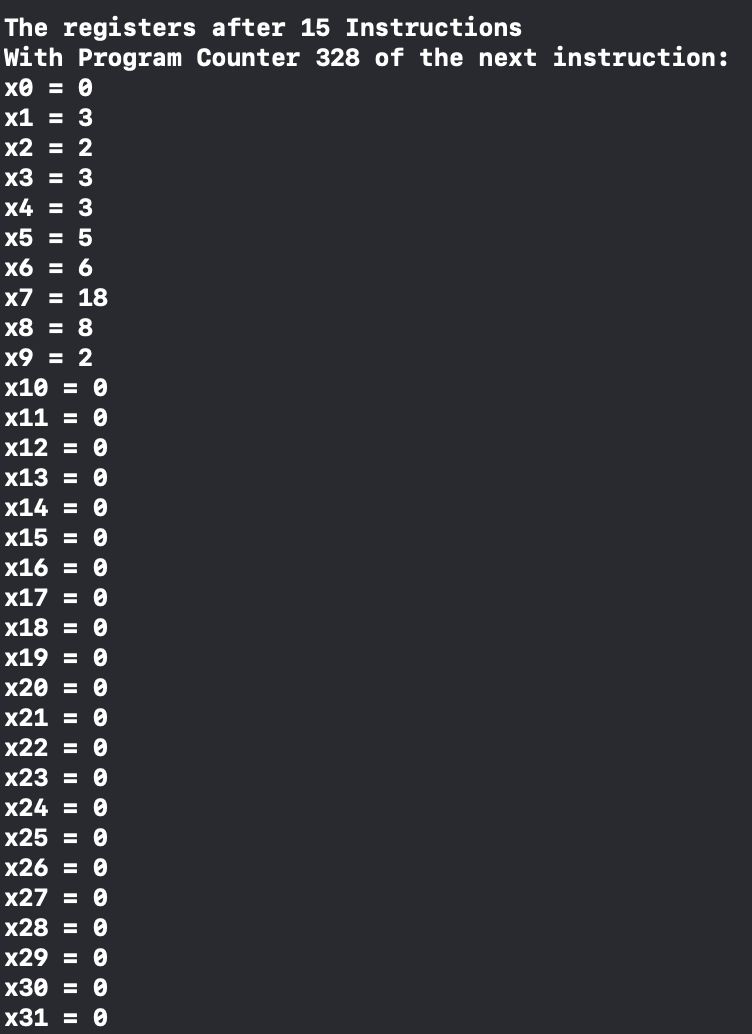
\

Program 3:

Instruction file:



Output:



**Conclusion:**

We truly learned so much when it comes to RISC-V instructions, C++ code, and the way to indirectly connect our understanding of both languages all together. We had to use both our knoowladge of C++ and RISC-V to contribute to this project.Tt been a great learning experience even if we faced some difficulties in picking between multiple approaches that this project can be implemented with. Afterall, this is what makes a learning experience truly beneficial: trial and error. We hope this report was useful and all the sections we provided were clear and straigh to the point.