Indian Institute of Technology, Ropar

CS204 Course Project - Phase I

Team Members:

1. Aditya Agarwal - 2019CSB1064
2. Aneeket Mangal - 2019CSB1071
3. Fadia Het Rakeshkumar - 2019CSB1084
4. Shikhar Soni - 2019CSB1119
5. Tanmay Aeron - 2019CSB1124

Project Description

The aim of this project is to simulate the machine level execution of RISC V 32-bit instructions using a high level language.

The Project also aims to give updates to the user regarding each step of the execution of the program. It also returns the final status of the memory and registers as output for the user to analyse the working of their programs thoroughly.

The Project currently allows the user to use 29 different instructions and can be extended to allow the use of any number of instructions by editing the .csv files as long as the instructions are supported by 32-bit RISC V ISA.

For each instruction the program gives various updates like IR, PC, decoded instruction, temporary registers like RA, RB, RZ, RY, etc. during each cycle and prints the number of cycles.

The program executes each instruction using five stages as described in the RISC V architecture.

Technologies employed

Back-end - Python3

1. **pandas** for reading **.csv** files.
2. **os** for getting and adding path to certain file locations.
3. **defaultdict** to make a hash map for memory.
4. **sys** for reading and editing files with ease.

Front-end - Python3

1. **PyQt5** for the Graphic User Interface.
2. **qdarkstyle** for dark theme.

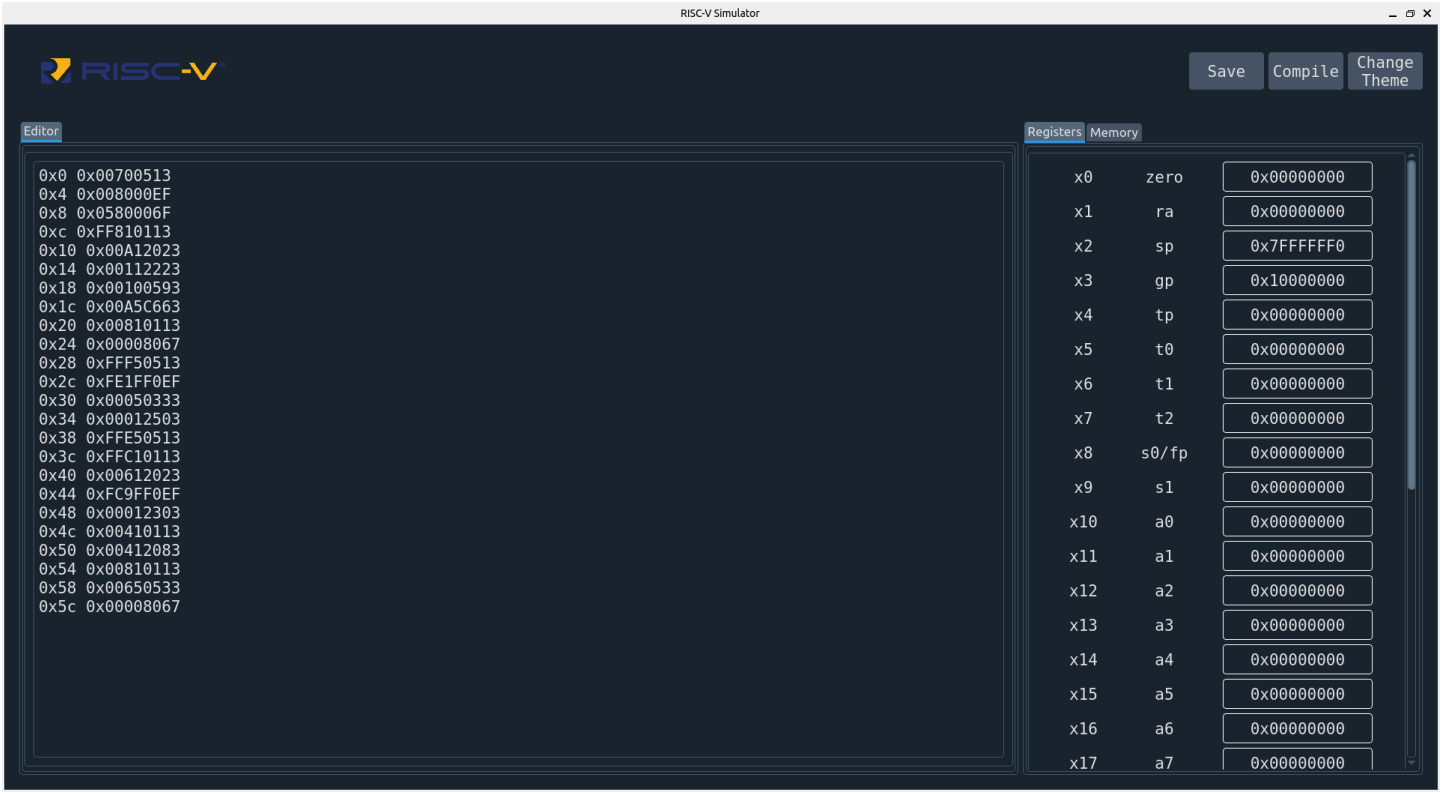
Implementation Details

We have implemented the project using the following method:

First it takes the input from a ***.mc*** file that contains the input in the required format, the content of this file can be modified using the GUI editor.
Then, it stores the ***.data*** part into the data memory and the ***.text*** part in the text part of the memory. Once that is completed, we run the program using the following method:

1. Instruction Fetch: PC is incremented by 4, and the instruction is loaded from the memory and stored in the Instruction Register.
2. Instruction Decode: We are identifying the instruction using the .csv file and **pandas** library and returning all the required fields. The registers RA and RB are also set during this stage.
3. Execute: The ALU executes the instruction by computing the desired output using values stored in registers in RA, RB, imm, etc. The required type of ALU instruction is determined by another .csv file. We are updating the output in the RZ register.
4. Memory Access: Memory is read and written in this stage. This stage is used only for load and store instructions. For the remaining instructions, this stage is redundant. The value of the RY register is also consequently updated.
5. Register Write-back: Here we are storing the value of the temporary register RY in the destination register, resulting in the register files being updated. The register update is extremely fast compared to memory read and update.

Graphical User Interface



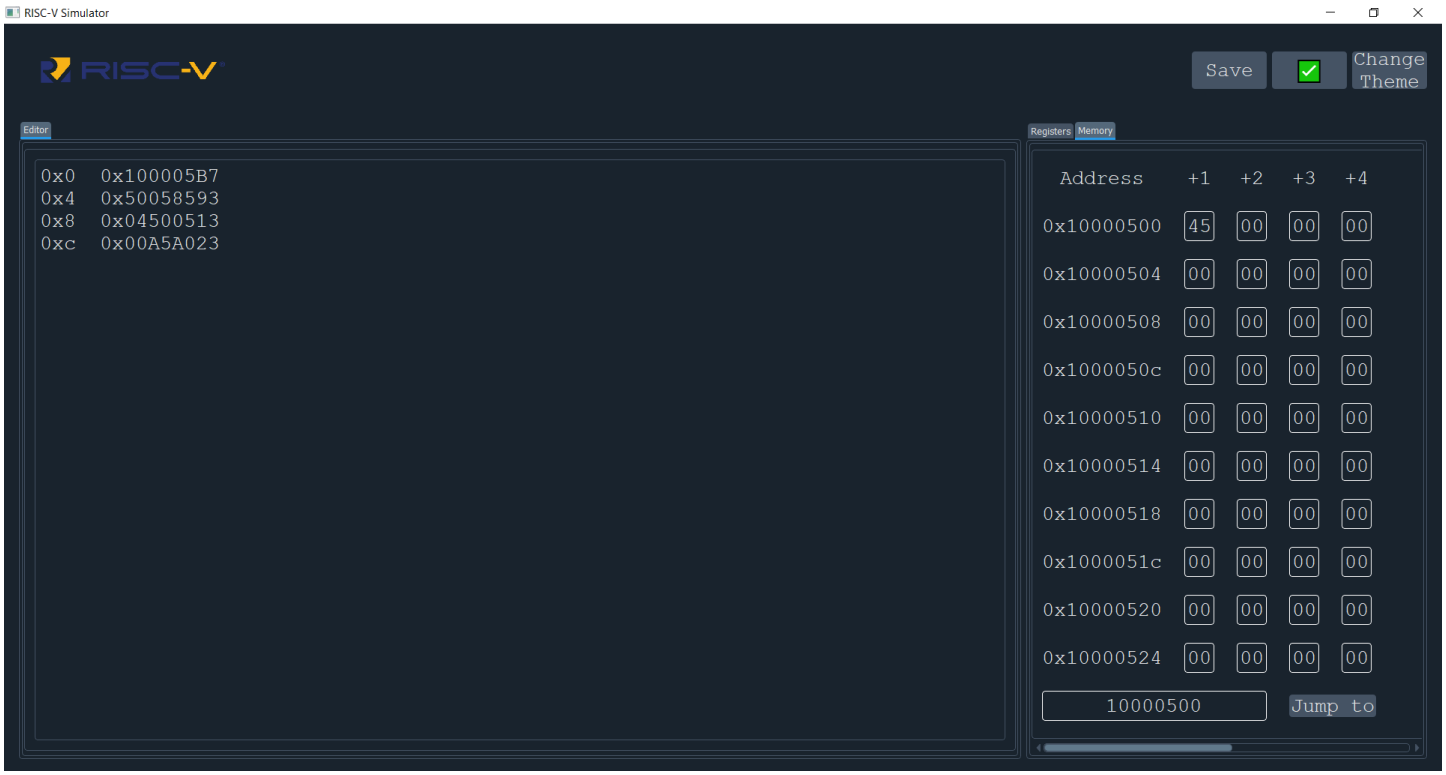
We have made GUI in both light and dark theme. The theme can be changed using the button "Change theme" on the top right. We have shown memory and registers on the right part of the GUI while on the left part there is Code Editor. First of all we will paste the code in the Editor on the left side and then save and compile the code by clicking the respective 'save' and 'compile' button. The code will get updated in main.mc file and then the code will be executed and a ✓ will appear. The memory and registers will be updated simultaneously.

We can jump in the memory using 'Jump to' on the bottom right in memory section.

We can see all the registers in the register file by scrolling in the register file section.

Executing main.py always generates GUI window and you can code in Editor.

We can also execute non GUI code by writing python main.py 2



Here we can see the jump to register functionality, where we have stored a value in the register 0x10000500, and are jumping to it and seeing the value.

Input/Output format details

Input Format:

0x0 0x00500513
0x4 0x008000EF
0x8 0x0440006F
$
0x10000000 0x64

The section given before the '$' sign is the text segment of the code and all the lines after it signify the data segment of the code.

Output is generated in the 'generated' folder. 'registers.txt' contains the register values, the 'memory.txt' contains the memory part and the 'outputLog.txt' contains the registers, memory and other details at each stage.

An example output for OutputLog.txt is given as follows:
Fetch stage:
The value of PC is : 00000000
The instruction in IR is : 10000597
Decode stage:
code : 'neumonic': 'auipc', 'opcode': '0b0010111', 'immediate': '00010000000000000000000000000000', 'rd': '01011', 'format': 'U', 'id': 25
rd is : 11
imm is : 10000000
Execute stage:
operand1 is 00000000 operand2 is: 10000000
RZ is : 10000000
Memory Access stage:
Register Update Stage:
RD: RY: 11 10000000
cycle is : 1