



THE AMERICAN UNIVERSITY IN CAIRO الجامعة الأمريكية بالقاهرة

School of Sciences and Engineering
Department of Computer Science and Engineering
Spring 2024

CSCE 231: Computer Organization and Assembly
Language Programming

Project 01: RISC-V Simulator

Dr. Cherif Salama

Submitted By:

Farida Bey - 900212071

Hassan Mohammed - 900226023

Mark Kyrollos - 900211436

Peter Aziz - 90212368

Date: Apr 27, 2024

TABLE OF CONTENTS

GitHub repo link: <https://github.com/FaridaBey/rv32i-simulator.git>

1. Implementation Description:.....	3
2. Design Decisions and Assumptions:.....	4
3. Known Bugs or Issues:.....	4
4. User Guide:.....	5
5. List of Simulated Programs:.....	9
Conclusion:.....	11

1. Implementation Description:

Our RISC-V simulator is coded using our preferred local coding IDE, Apache NetBeans. This IDE provides robust support for Java development and was instrumental in implementing our GUI feature seamlessly. Leveraging the capabilities of Apache NetBeans, we were able to create an intuitive graphical interface that enhances the user experience. We integrated the test programs in our GUI for better access by the user. There are several features that set our RISC-V simulator apart from others in the field. In addition to the core functionality of simulating RV32I instructions, we have incorporated several bonus features to enhance the functionality of the simulator:

- **GUI Application:** Users can interact with the simulator through an intuitive graphical interface, simplifying the simulation process and making it accessible to users of all levels of expertise.
- **Output in Decimal, Binary, and Hexadecimal:** All values are displayed in multiple formats, including decimal, binary, and hexadecimal. This multi-format output provides users with flexibility and aids in better understanding and analysis of program execution.
- **Support for Pseudo Instructions and Directives:** We have extended the simulator's capabilities by adding support for five pseudo instructions and three directives. This expansion of supported instructions allows for a wider range of programs to be simulated, increasing the versatility of the simulator.
- **Support for Compressed Instructions (RV32IC):** The simulator effectively supports the full RV32IC instruction set, including compressed instructions. This comprehensive support enables users to simulate programs that utilize compressed instructions, enhancing the realism and accuracy of the simulation.
- **Integer Multiplication/Division (RV32IM):** Our simulator fully supports the RV32IM instruction set, including integer multiplication and division. This addition enables users to simulate programs that involve complex arithmetic operations, facilitating more comprehensive program execution and analysis.

2. Design Decisions and Assumptions:

The RISC-V simulator is built using the Swing GUI framework in Java, providing a user-friendly interface for interaction. Users can step through instructions, exit the simulator, or restart the simulation using dedicated buttons. The GUI allows users to select different formats (Decimal, Binary, Hexadecimal) for displaying register values, enhancing customization. Instructions are processed sequentially, with error handling mechanisms in place for unsupported instructions or attempts to use register x0 as a destination. Separate displays for data memory and instruction memory facilitate visualization of memory contents. The simulator supports a range of RISC-V instructions, including arithmetic, logical, memory access, and control transfer instructions, along with compressed instructions such as "c.addi". Simulation completion is determined by comparing the program counter to the maximum program counter value; if exceeded, a completion message is displayed. Initialization of the simulator includes setting initial values for assembly code, program counter, and data memory, ensuring a smooth start to the simulation process.

3. Known Bugs or Issues:

There are no bugs or issues that showed up while we conducted the testing using the test cases shown below.

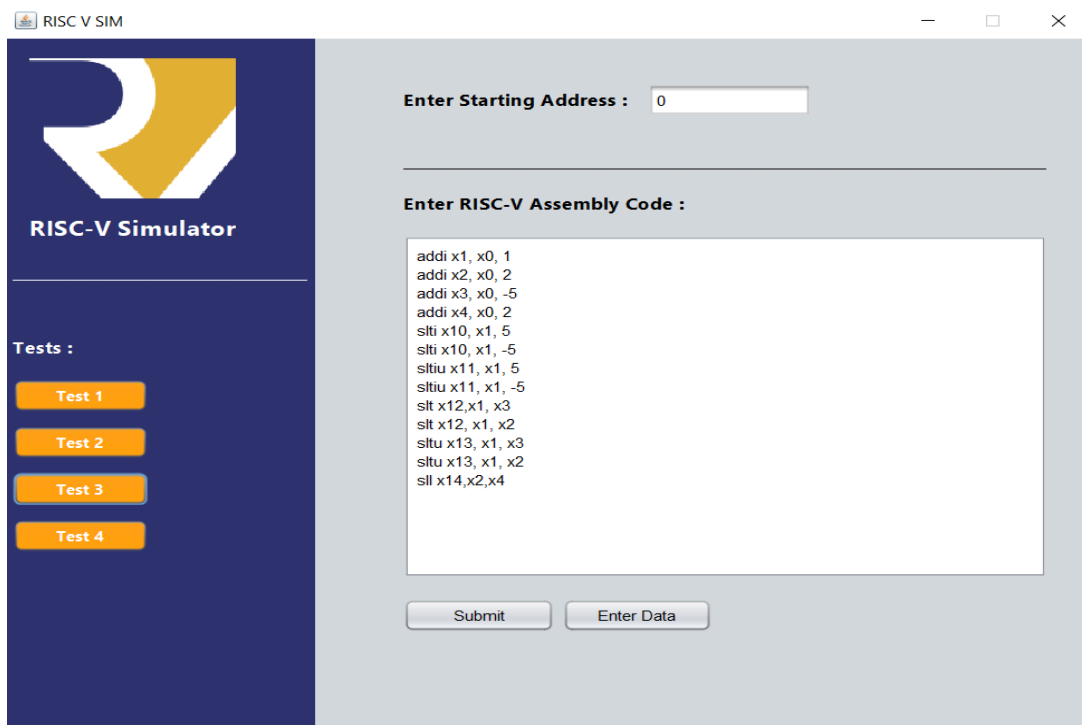
4. User Guide:

Compilation and Execution:

1. Download the project files from GitHub.
2. Open the project in an IDE or compile using command-line tools.
3. Run the main Java class to launch the GUI application.
4. Follow the on-screen instructions to input the assembly program, program data, and starting address.
5. Start the simulation, and the GUI will display the program counter, register file contents, and memory contents after each instruction execution.

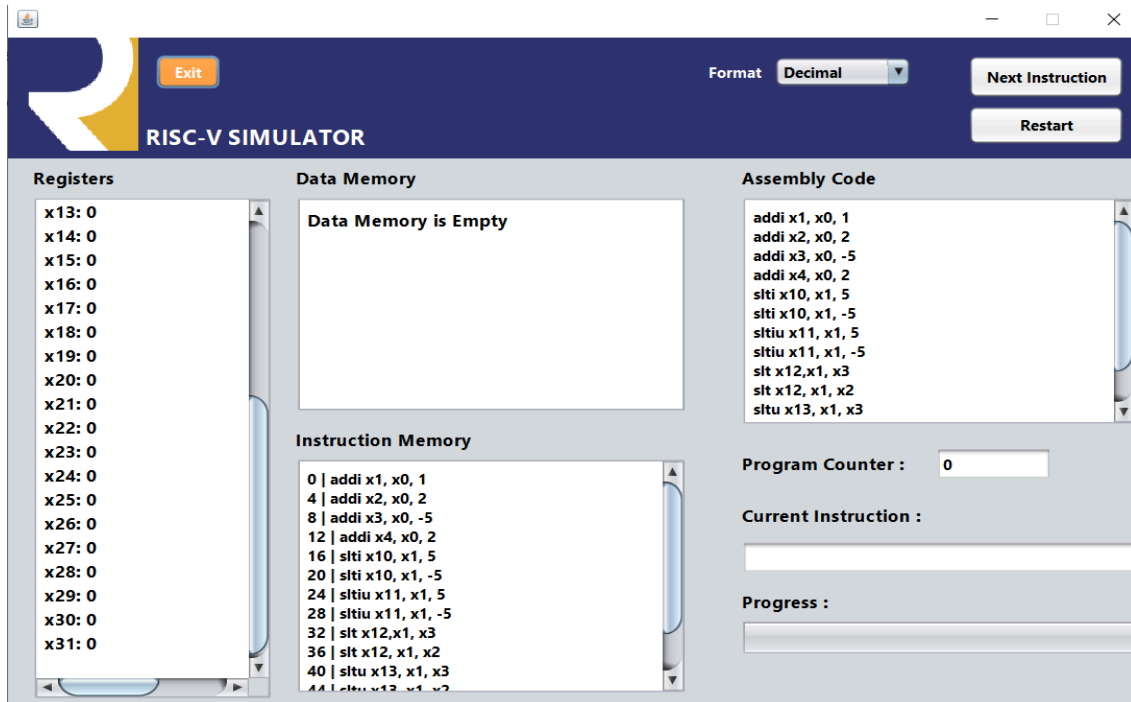
Here is a step-by-step guide to simulate a simple program:

1. Just after running our program, the user is given several options such as a textbox to enter their assembly code as well as for the starting address. There are also predefined test case programs as seen below:

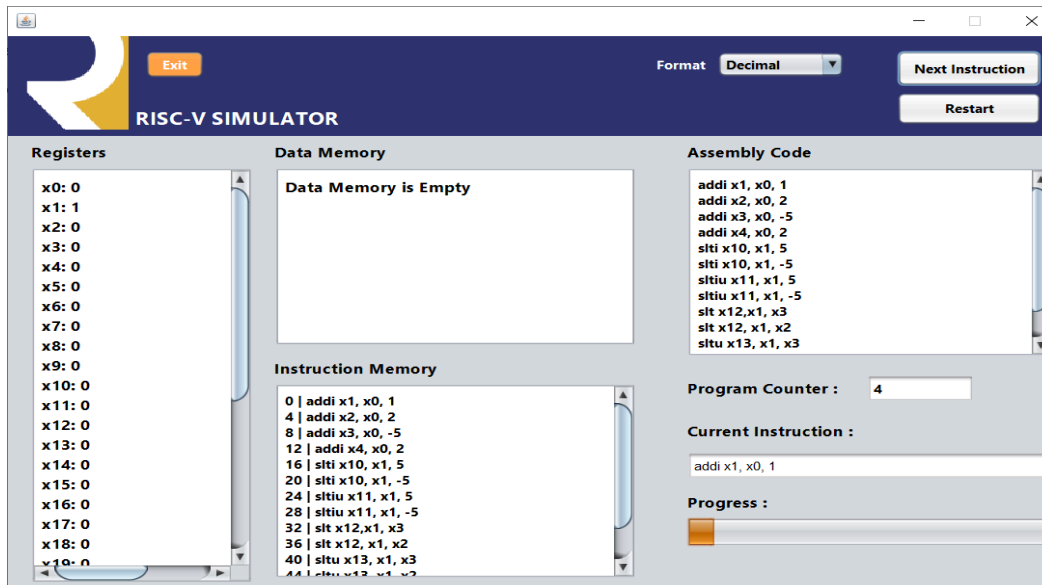


2. After pressing the 'Submit' button, the program displays the window below. This window showcases many things including all registers used, the data memory of each register at every instance in our assembly code, the instruction memory display which shows the memory of the data at every line of code, and the assembly code itself, a program

counter, a current instruction display, and finally a progress bar which increments as soon as the 'Next Instruction' button is clicked. The 'Restart' button clears the current running program and runs the program again. We implemented a bonus feature to include a button called 'Format' at the top right of the window for the user to change the data display to Decimal, binary, or hexadecimal as they desire

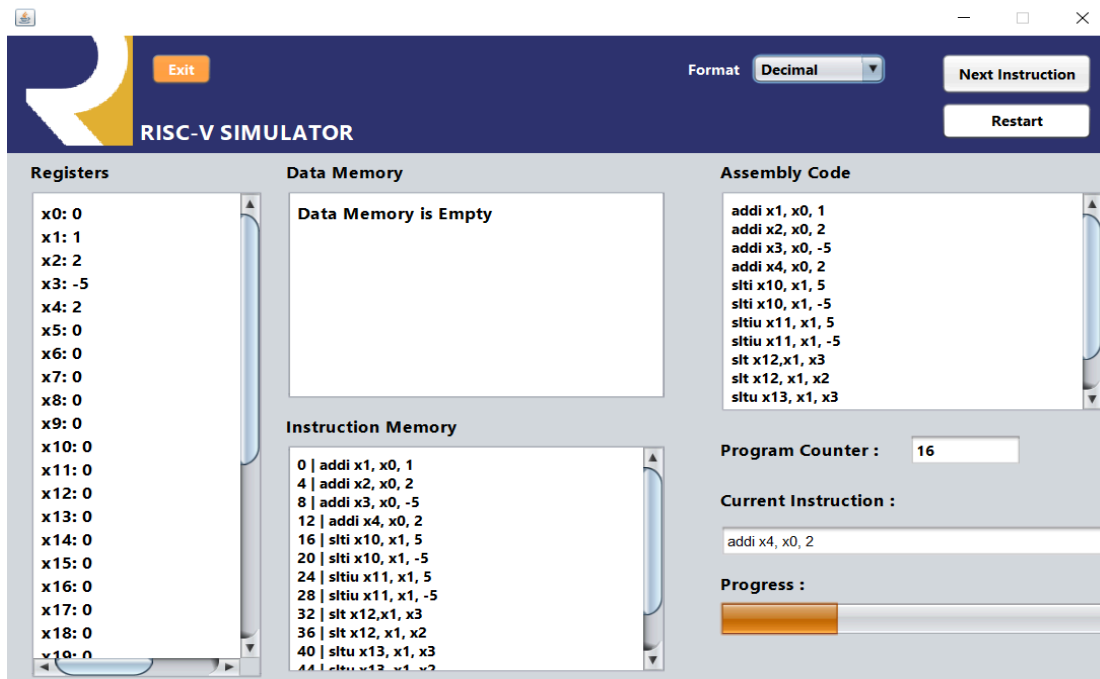


- After pressing the 'Next Instruction' button for the first time, the user can see where in the memory is each instruction stored at. Also, the 'Current Instruction' helps the user keep track of the line of code they are currently at. The 'Progress' bar increases as shown after each line of code. As we can see below, the register x1 is automatically updated to have a value of 1, displayed in the 'Registers' column on the left of the window

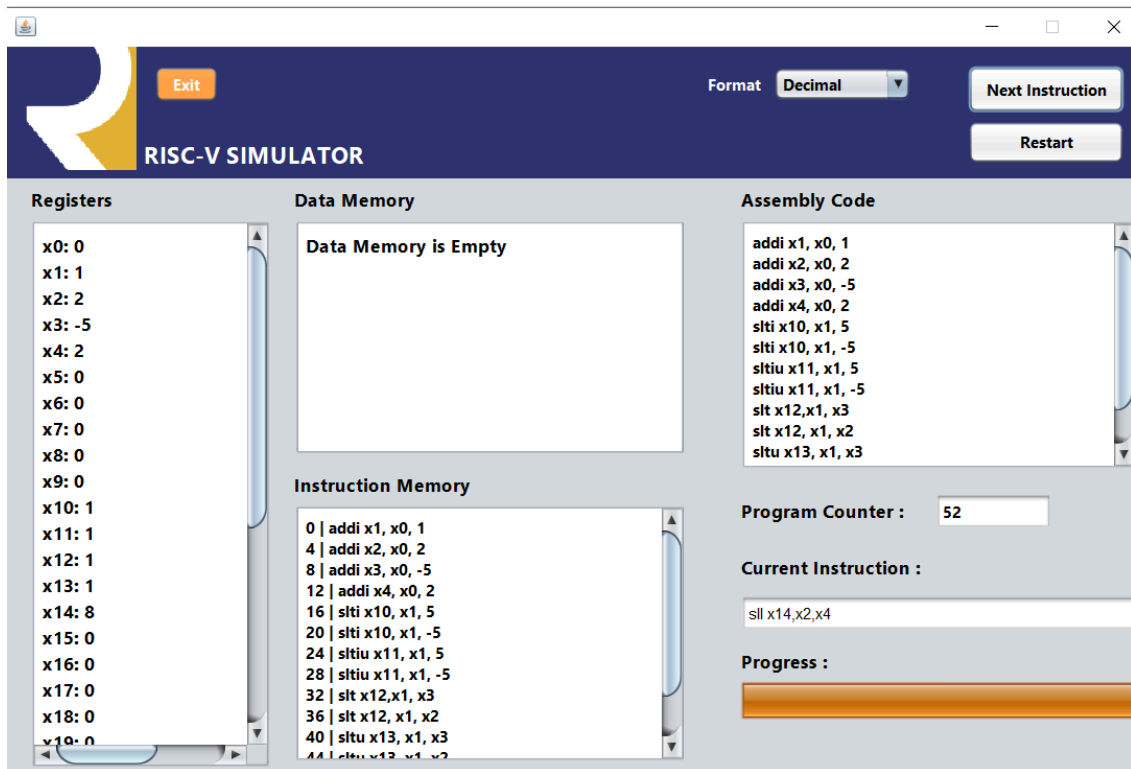


4.

5. Whenever we are dealing with instructions that deal with the memory (such as lw or sw for example) the 'Data Memory' column is updated to show the stack pointer memory allocation. There are no load/save functions used in this specific program, however there the user would notice a change in the 'Data Memory' if any memory allocation is encountered. The program counter is updated accordingly



6. After going through multiple instructions, all necessary updates are displayed as shown below.



The screenshot shows the RISC-V Simulator interface. The top bar includes an 'Exit' button, a 'Format' dropdown set to 'Decimal', and 'Next Instruction' and 'Restart' buttons. The main area is divided into four panels: Registers, Data Memory, Assembly Code, and Instruction Memory. The Registers panel shows values for x0 through x19. The Data Memory panel is empty. The Assembly Code panel shows a list of instructions. The Instruction Memory panel shows the current instruction at address 52, which is 'sll x14, x2, x4'. The Program Counter is 52, and the Current Instruction is 'sll x14, x2, x4'. A progress bar is at the bottom.

Registers

Register	Value
x0	0
x1	1
x2	2
x3	-5
x4	2
x5	0
x6	0
x7	0
x8	0
x9	0
x10	1
x11	1
x12	1
x13	1
x14	8
x15	0
x16	0
x17	0
x18	0
x19	0

Data Memory

Data Memory is Empty

Assembly Code

```
addi x1, x0, 1
addi x2, x0, 2
addi x3, x0, -5
addi x4, x0, 2
slli x10, x1, 5
slli x10, x1, -5
slli x11, x1, 5
slli x11, x1, -5
slli x12, x1, x3
slli x12, x1, x2
slli x13, x1, x3
```

Instruction Memory

Address	Instruction
0	addi x1, x0, 1
4	addi x2, x0, 2
8	addi x3, x0, -5
12	addi x4, x0, 2
16	slli x10, x1, 5
20	slli x10, x1, -5
24	slli x11, x1, 5
28	slli x11, x1, -5
32	slli x12, x1, x3
36	slli x12, x1, x2
40	slli x13, x1, x3
44	slli x13, x1, x2

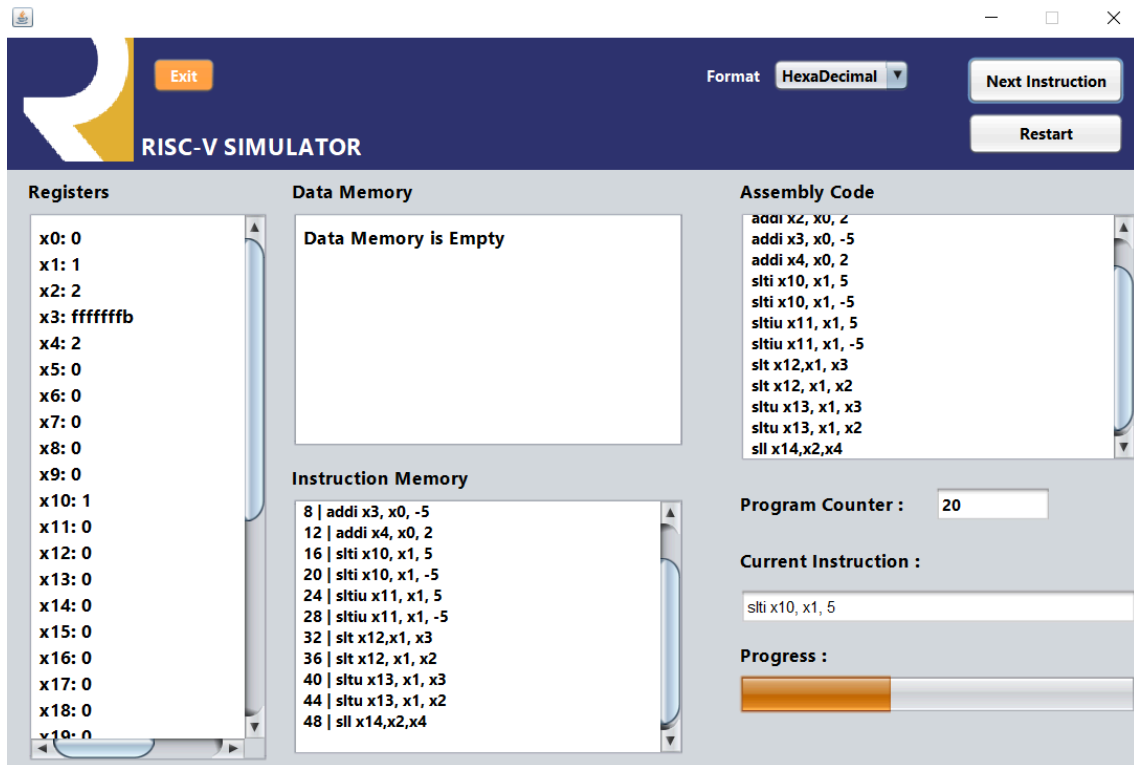
Program Counter : 52

Current Instruction :

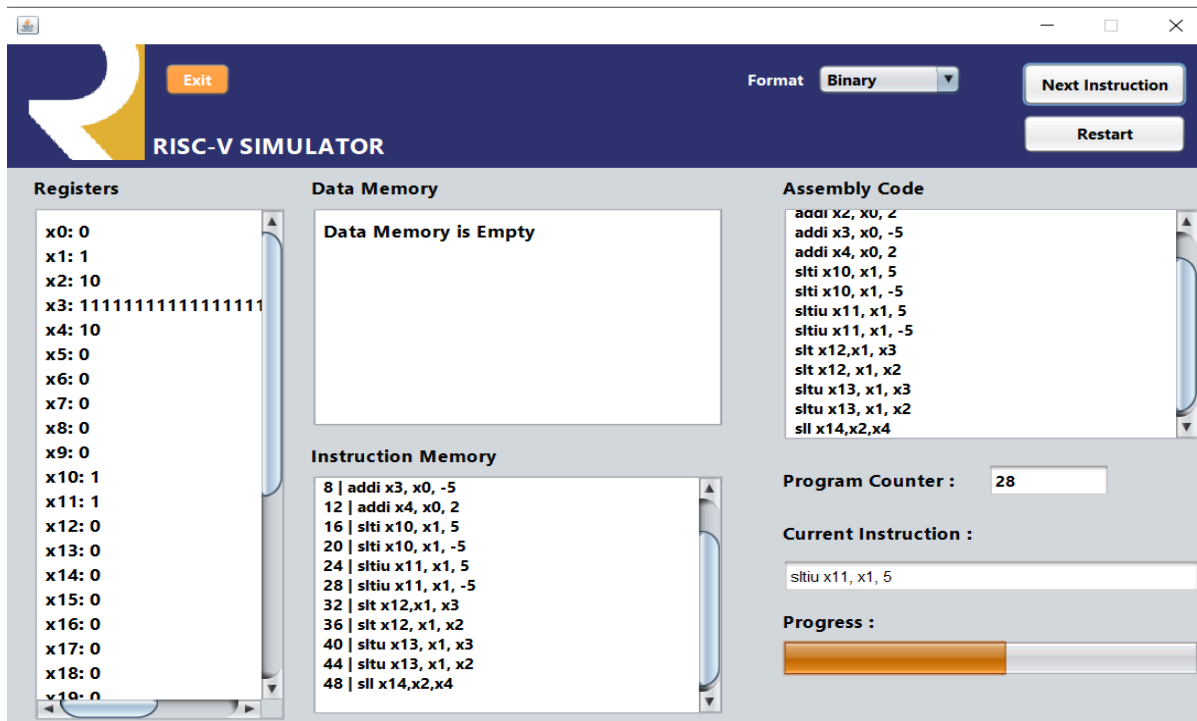
sll x14, x2, x4

Progress :

7. In addition, here is how it would look if the 'Format' at the top of the window is set to Hexadecimal and Binary:



Binary format:



5. List of Simulated Programs:

In the simulation, We've included programs covering all supported instructions, including loops. Each program is integrated into the GUI as buttons in the test section for easy access and evaluation by the user. This setup allows for comprehensive testing and demonstration of the simulator's capabilities within the graphical interface.

1. **1st test program:** The first program supports the given instructions: **addi, add, sub, sb, sh, sw, lb, lh, lbu, lhu, slli, srli, srai**. Here is the full assembly program with its starting address:

Starting Address = 0

```
addi x1, x0, 10
addi x20, x0, 2000
addi x2, x0, 20
add x3, x2, x1
sub x4, x1, x2
sb x1, 0 (x20)
sh x2, 4 (x20)
sw x4, 8 (x20)
lb x10, 8 (x20)
lh x10, 8 (x20)
lw x10, 8 (x20)
lbu x10, 8 (x20)
lhu x10, 8 (x20)
slli x11, x1, 2
srli x12, x1, 2
srai x13, x1, 2
```

2. **2nd test program:** The second program supports the given instructions: **lui, auipc, addi, bge, bgeu, jal, blt, beq, bne, bltu, ECALL** . Here is the full assembly program with its starting address:

Starting Address = 100

```
lui x1,20
auipc x2,20
addi x3,x3,10
beq x4,x3,124
addi x4,x4,1
jal x5,112
addi x6,x6,5
addi x7,x7,2
blt x6,x7,144
addi x6,x6,-1
jal x8,132
addi x9,x9,10
addi x10,x10,7
bne x9,x10,164
addi x10,x10,1
jal x11,152
addi x12,x12,1
addi x13,x13,3
bge x12,x13,184
addi x12,x12,1
jal x14,172
addi x15,x15,5
addi x16,x16,2
bltu x15,x16,204
addi x15,x15,-1
jal x18,192
addi x19,x19,1
addi x20,x20,3
bgeu x19,x20,224
addi x19,x19,1
jal x21,212
ECALL
```

3. **3rd test program:** The third program supports the given instructions: **addi, xori, ori, andi, xor, or, and, srl, sra- EBREAK**. Here is the full assembly program:

```
addi x1, x0, 20
addi x2, x0, 30
addi x3, x0, 2
xori x10, x1, 12
ori x11, x1, 12
andi x12, x1, 12
xor x13, x1, x2
or x14, x1, x2
and x15, x1, x2
srl x16, x2, x3
sra x17, x2, x3
```

EBREAK

4. **4th Test Program:** the fourth program supports the given instructions: **slti-sltiu-slt-sltu-sll-FENCE**

```
addi x1, x0, 1
addi x2, x0, 2
addi x3, x0, -5
addi x4, x0, 2
slti x10, x1, 5
slti x10, x1, -5
sltiu x11, x1, 5
sltiu x11, x1, -5
slt x12, x1, x3
slt x12, x1, x2
sltu x13, x1, x3
sltu x13, x1, x2
sll x14, x2, x4
FENCE
```

Conclusion:

In conclusion, our RISC-V RV32I simulator offers comprehensive support for simulating assembly programs, with additional features to enhance usability and functionality. We are confident that our simulator meets the requirements outlined and provides a valuable tool for understanding and experimenting with RISC-V architecture.