Risk-V Simulator Report

Saif Ahmed 900225535
Mohamed Mansour 900222990
Merna Hebishy 900221976

Description

Our implementation consists of a class named program that contains the functions:

- **populate()**: This function parses the raw instructions and populates the alloperations vector with parsed instructions, while also handling labels and updating the labels map with label addresses.
- **dooperation()**: This function executes the instructions stored in alloperations one by one, updating the program counter (PC) accordingly until it encounters an "ebreak", "ecall", or "fence" instruction.
- **getoperation():** This function reads operations from the instruction and data files.
- **dooo():** This function directs the program to the appropriate instruction function.
- **run()**: This function displays the current state of registers and memory, including their values and addresses, along with the program counter.
- **HaveR():** This function returns the value inside the register.
- **DoR():** This function sets the register value.
- **checkRegister():** This function checks if a string represents a valid register
- **Instruction Execution Functions**: These functions, such as ADD(), BEQ(), LW(), etc., execute specific types of instructions by manipulating registers and memory according to the RISC-V instruction set architecture.
- **isValidImmediate(const string&, int&)**: Validates immediate values used in instructions.
- **decimalToBinary(int)**: Converts decimal integers to binary strings.
- **decimalToHexa(int)**: Converts decimal integers to hexadecimal strings.
- **main()**: The entry point of the program where it initializes the program class, specifies input files, and starts the simulation.

We added two **bonus** features:

1. Added support for integer multiplication and division (including remainder) to effectively support the full RV32IM instruction set
2. Outputted all values in decimal, binary, and hexadecimal

Design Decisions

1- We decided to read from a text file “instructions.txt” instead of inputting instructions one by one.
2- Constructor: Initializes the simulator with the provided instruction and data files, as well as the starting program counter value.
3- Data Structures:
- **map<string, string> RegisterHash**: Maps register names to their corresponding register numbers.
- **vector<string> RegistersN**: Stores register names for display purposes.
- **map<unsigned int, int> Memory**: Maps memory addresses to their stored values.
- **map<string, unsigned int> labels**: Maps labels to their corresponding memory addresses.

Handling Errors

Incorrect Number of Arguments: If the program is executed without providing the correct number of arguments, an error message will be displayed, indicating the correct usage of the program.
Invalid Starting PC Address: If the provided starting PC address is negative or not divisible by 4, the program will prompt you to enter a valid starting PC address until the criteria are met.
Input Validation: The program performs thorough validation of input data, including instruction files and data files. It checks for file availability, correct file format, and proper data formatting to prevent potential issues related to missing or incorrectly formatted input files.
Parsing Errors: During the parsing phase, the program checks for syntax errors and inconsistencies in the input assembly code. It ensures that each instruction is correctly formatted and contains the required components. If any parsing error is detected, the program provides informative error messages to guide the user in identifying and correcting the issue.
Memory Access Violations: When accessing memory locations, the program verifies the validity of memory addresses and ensures that memory operations stay within the bounds of allocated memory. If an attempt is made to access an invalid memory location or if a memory operation violates memory alignment constraints, the program raises appropriate error messages to notify the user about the issue.
Invalid Instructions: The simulator detects and handles attempts to execute invalid or unsupported instructions. If the program encounters an instruction that is not part of the supported RISC-V instruction set, it raises an error to indicate that the instruction cannot be executed.
Register Operations: During register operations, the program validates register identifiers and ensures that register values are modified within the permissible range. It

also prevents attempts to modify reserved registers or registers with restricted usage. Any violations of these constraints result in error messages to alert the user about the attempted operation.

Assumptions

It is presumed that the input programs adhere to properly structured RISC-V assembly code.
Rationale: The simulator prioritizes the execution of legitimate RISC-V instructions. Dealing with incorrect or poorly formatted input is limited to notifying the user of errors in specific lines.
Little-Endian Design:
The simulator defaults to a little-endian memory architecture unless otherwise instructed.
Rationale: Little-endian architecture is widely used, simplifying memory access and byte-ordering operations by assuming this default configuration.

Issues

The program will stop working if it can't find the "Memory.txt" file it needs. This file holds the program's memory. Without it, the program can't get the data it requires, leading to a crash.

User Guide for Running the Program

This user guide outlines the steps to run the program successfully from the terminal. The program is designed to process instruction and data files, executing a simulation based on the provided starting PC address.

System Requirements

Operating System: The program is compatible with Windows, macOS, and Linux operating systems.
Compiler: Ensure that a C++ compiler is installed on your system, such as g++, clang++, or Visual C++ compiler.
Getting Started
Download: Obtain the program source code and necessary files.
Compilation: Compile the source code using your preferred C++ compiler.
Terminal Access: Open the terminal or command prompt on your system.
Navigate: Use the cd command to navigate to the directory where the compiled executable is located.

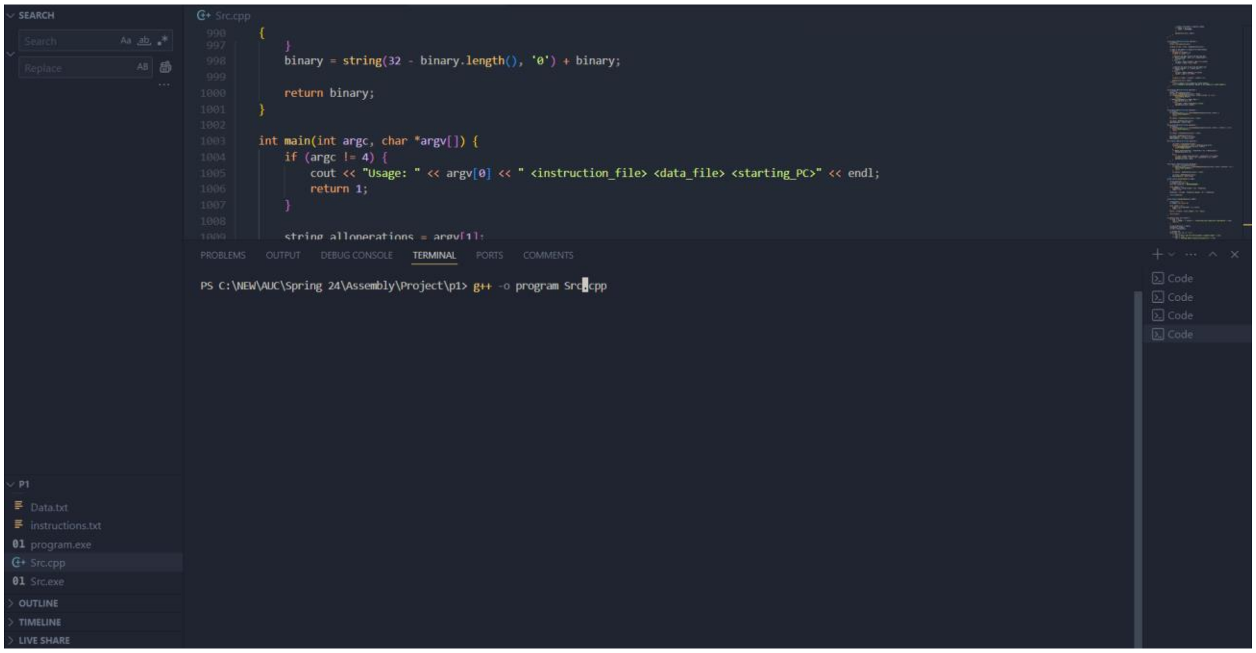
Command Line Arguments

The program requires three command-line arguments:
<instruction_file>: The path to the file containing instructions.
<data_file>: The path to the file containing data.
<starting_PC>: The starting PC address for program execution.

Running the Program
Execute the program by typing the following command in the terminal:
./program <instruction_file> <data_file> <starting_PC>
Replace the program with the name of the compiled executable and provide the paths to the instruction and data files along with the starting PC address.

Simulation Example:

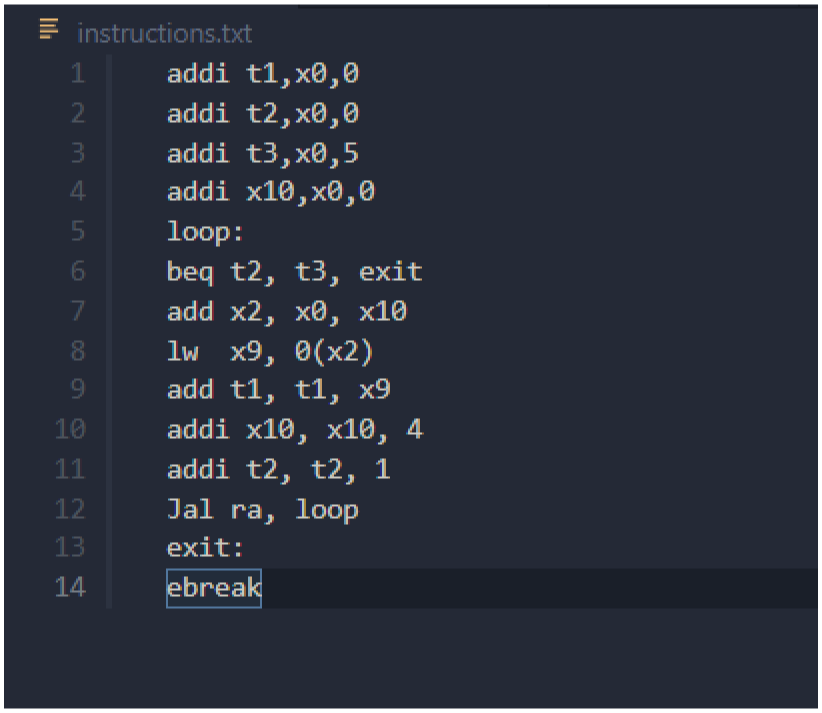
First step
Inside the folder
Write g++ -o program Src.cpp



```cpp
    {
        }
        binary = string(32 - binary.length(), '0') + binary;

        return binary;
    }

    int main(int argc, char *argv[]) {
        if (argc != 4) {
            cout << "Usage: " << argv[0] << " <instruction_file> <data_file> <starting_PC>" << endl;
            return 1;
        }

        string alloperations = argv[1];
```

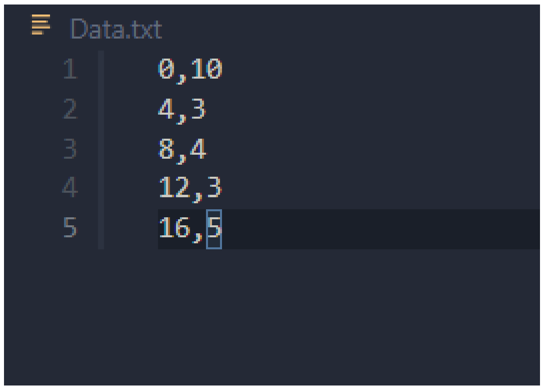
```
PS C:\NEW\AUC\Spring 24\Assembly\Project\p1> g++ -o program Src.cpp
```

Second Step
Write in the terminal
./program instructions.txt Data.txt 0
Change the file names and the initial PC as you want. You will have the output ready for you.
Third Step
Our program fixes any spaces or additional commas. Moreover, our program works perfectly with lower-case or upper-case words. You can just write the instructions as follows: -


instructions.txt
```
addi t1,x0,0
addi t2,x0,0
addi t3,x0,5
addi x10,x0,0
loop:
beq t2, t3, exit
add x2, x0, x10
lw  x9, 0(x2)
add t1, t1, x9
addi x10, x10, 4
addi t2, t2, 1
Jal ra, loop
exit:
ebreak
```

Fourth Step

For writing the data file. Start with the memory location, then the value you want to add to this memory


Data.txt
```
0,10
4,3
8,4
12,3
16,5
```

Test Cases

First test case

1- Sum array elements

Expected: The function should sum the array of elements found in the data file in memory.

The answer would be 25 saved at t1

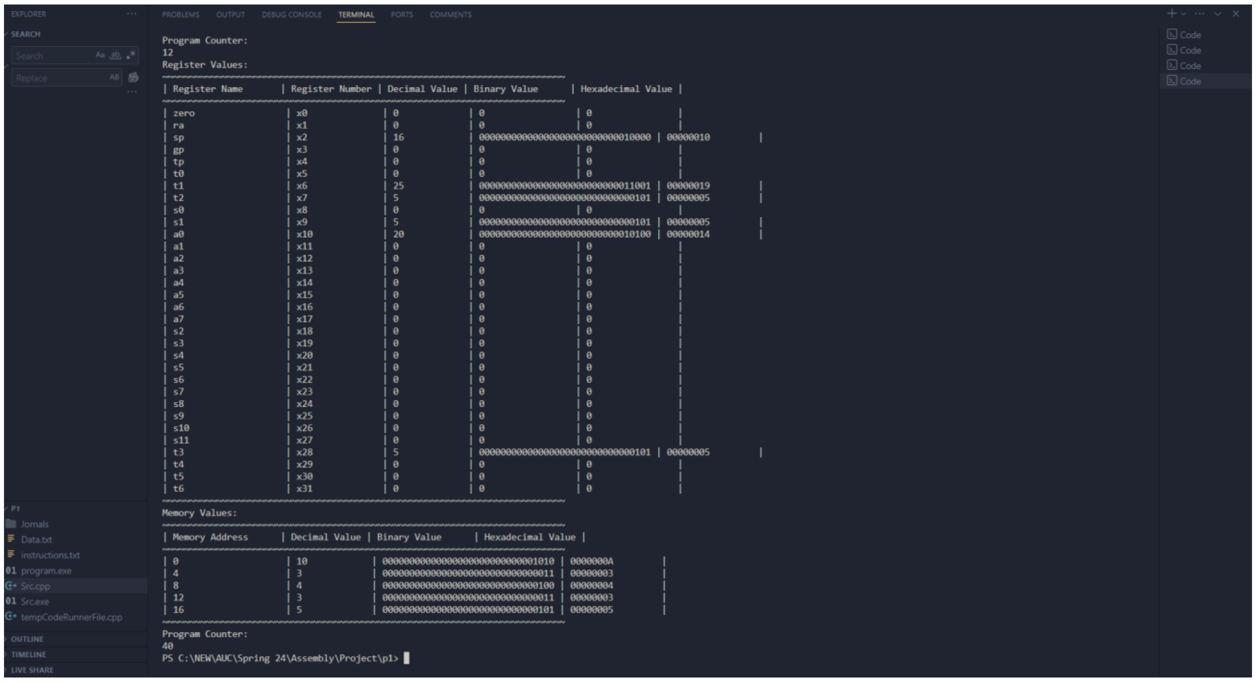
instructions.txt

```
addi t1,x0,0
addi t2,x0,0
addi t3,x0,5
addi x10,x0,0
loop:
beq t2, t3, exit
add x2, x0, x10
lw  x9, 0(x2)
add t1, t1, x9
addi x10, x10, 4
addi t2, t2, 1
jal ra, loop
exit:
ebreak
```

Data.txt

```
0,10
4,3
8,4
12,3
16,5
```

Output simulation



```
Program Counter:
12
Register Values:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| Register Name       | Register Number | Decimal Value | Binary Value      | Hexadecimal Value |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| zero                | x0              | 0             | 0                 | 0                 |
| ra                  | x1              | 0             | 0                 | 0                 |
| sp                  | x2              | 16            | 00000000000000000000000000010000 | 00000010          |
| gp                  | x3              | 0             | 0                 | 0                 |
| tp                  | x4              | 0             | 0                 | 0                 |
| t0                  | x5              | 0             | 0                 | 0                 |
| t1                  | x6              | 25            | 00000000000000000000000000011001 | 00000019          |
| t2                  | x7              | 5             | 00000000000000000000000000000101 | 00000005          |
| s0                  | x8              | 0             | 0                 | 0                 |
| s1                  | x9              | 5             | 00000000000000000000000000000101 | 00000005          |
| a0                  | x10             | 20            | 00000000000000000000000000010100 | 00000014          |
| a1                  | x11             | 0             | 0                 | 0                 |
| a2                  | x12             | 0             | 0                 | 0                 |
| a3                  | x13             | 0             | 0                 | 0                 |
| a4                  | x14             | 0             | 0                 | 0                 |
| a5                  | x15             | 0             | 0                 | 0                 |
| a6                  | x16             | 0             | 0                 | 0                 |
| a7                  | x17             | 0             | 0                 | 0                 |
| s2                  | x18             | 0             | 0                 | 0                 |
| s3                  | x19             | 0             | 0                 | 0                 |
| s4                  | x20             | 0             | 0                 | 0                 |
| s5                  | x21             | 0             | 0                 | 0                 |
| s6                  | x22             | 0             | 0                 | 0                 |
| s7                  | x23             | 0             | 0                 | 0                 |
| s8                  | x24             | 0             | 0                 | 0                 |
| s9                  | x25             | 0             | 0                 | 0                 |
| s10                 | x26             | 0             | 0                 | 0                 |
| s11                 | x27             | 0             | 0                 | 0                 |
| t3                  | x28             | 5             | 00000000000000000000000000000101 | 00000005          |
| t4                  | x29             | 0             | 0                 | 0                 |
| t5                  | x30             | 0             | 0                 | 0                 |
| t6                  | x31             | 0             | 0                 | 0                 |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Memory Values:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| Memory Address      | Decimal Value | Binary Value      | Hexadecimal Value |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| 0                   | 10            | 00000000000000000000000000001010 | 0000000A          |
| 4                   | 3             | 00000000000000000000000000000011 | 00000003          |
| 8                   | 4             | 00000000000000000000000000000100 | 00000004          |
| 12                  | 3             | 00000000000000000000000000000011 | 00000003          |
| 16                  | 5             | 00000000000000000000000000000101 | 00000005          |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Program Counter:
40
PS C:\NEW\AUC\Spring 24\Assembly\Project\p1>
```

As you can see t1 = 25 **(Successful)**

Second Test case

Expected: After tracing the instructions, x1 is expected to be 6.

instructions.txt

```
addi x2, x2, 25
addi x3, x0, 20
addi x4, x0, 30
bge x2, x3, bge_label
addi x1, x1, 1
Fence
bge_label:
addi x1, x1, 1
bgeu x2, x3, bgeu_label
addi x1, x1, 1
Fence
bgeu_label:
addi x1, x1, 1
blt x3, x2, blt_label
addi x1, x1, 1
Fence
blt_label:
```

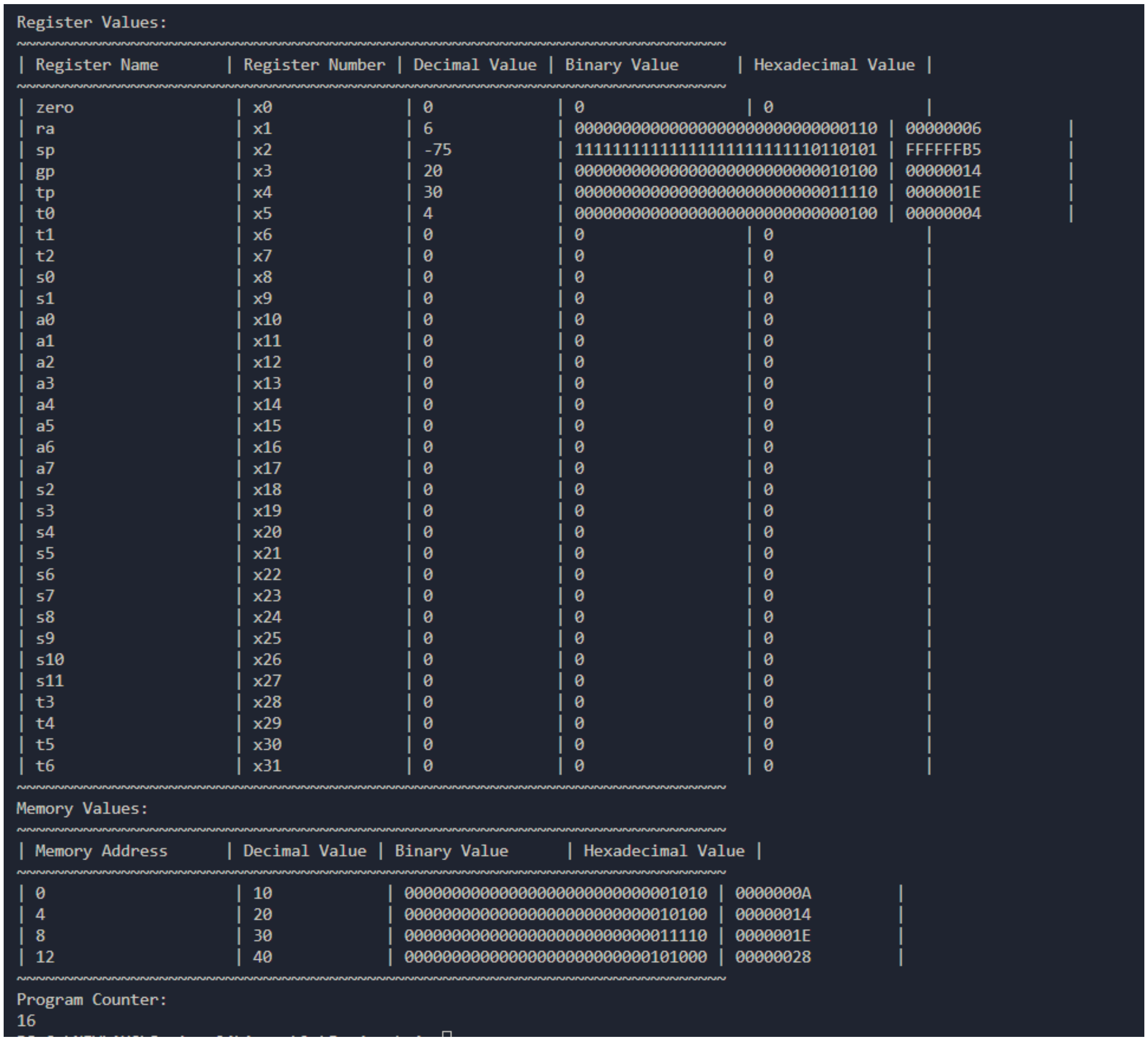
```
addi x1, x1, 1
bltu x3, x2, bltu_label
addi x1, x1, 1
Fence
bltu_label:
addi x1, x1, 1
bne x2, x3, bne_label
addi x1, x1, 1
Fence
bne_label:
addi x1, x1, 1
addi x2, x0, -100
jalr x5, x0,0
```

Data.txt

```
0,10
4,20
8,30
12,40
```

Output simulation

(x1 equals 6; Successful)



```
Register Values:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| Register Name      | Register Number | Decimal Value | Binary Value       | Hexadecimal Value |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| zero               | x0              | 0             | 0                  | 0                 |
| ra                 | x1              | 6             | 00000000000000000000000000000110 | 00000006          |
| sp                 | x2              | -75           | 11111111111111111111111110110101 | FFFFFFB5          |
| gp                 | x3              | 20            | 00000000000000000000000000010100 | 00000014          |
| tp                 | x4              | 30            | 00000000000000000000000000011110 | 0000001E          |
| t0                 | x5              | 4             | 00000000000000000000000000000100 | 00000004          |
| t1                 | x6              | 0             | 0                  | 0                 |
| t2                 | x7              | 0             | 0                  | 0                 |
| s0                 | x8              | 0             | 0                  | 0                 |
| s1                 | x9              | 0             | 0                  | 0                 |
| a0                 | x10             | 0             | 0                  | 0                 |
| a1                 | x11             | 0             | 0                  | 0                 |
| a2                 | x12             | 0             | 0                  | 0                 |
| a3                 | x13             | 0             | 0                  | 0                 |
| a4                 | x14             | 0             | 0                  | 0                 |
| a5                 | x15             | 0             | 0                  | 0                 |
| a6                 | x16             | 0             | 0                  | 0                 |
| a7                 | x17             | 0             | 0                  | 0                 |
| s2                 | x18             | 0             | 0                  | 0                 |
| s3                 | x19             | 0             | 0                  | 0                 |
| s4                 | x20             | 0             | 0                  | 0                 |
| s5                 | x21             | 0             | 0                  | 0                 |
| s6                 | x22             | 0             | 0                  | 0                 |
| s7                 | x23             | 0             | 0                  | 0                 |
| s8                 | x24             | 0             | 0                  | 0                 |
| s9                 | x25             | 0             | 0                  | 0                 |
| s10                | x26             | 0             | 0                  | 0                 |
| s11                | x27             | 0             | 0                  | 0                 |
| t3                 | x28             | 0             | 0                  | 0                 |
| t4                 | x29             | 0             | 0                  | 0                 |
| t5                 | x30             | 0             | 0                  | 0                 |
| t6                 | x31             | 0             | 0                  | 0                 |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Memory Values:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| Memory Address     | Decimal Value | Binary Value       | Hexadecimal Value |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| 0                  | 10            | 00000000000000000000000000001010 | 0000000A          |
| 4                  | 20            | 00000000000000000000000000010100 | 00000014          |
| 8                  | 30            | 00000000000000000000000000011110 | 0000001E          |
| 12                 | 40            | 00000000000000000000000000101000 | 00000028          |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Program Counter:
16
```

Third Test case

instructions.txt

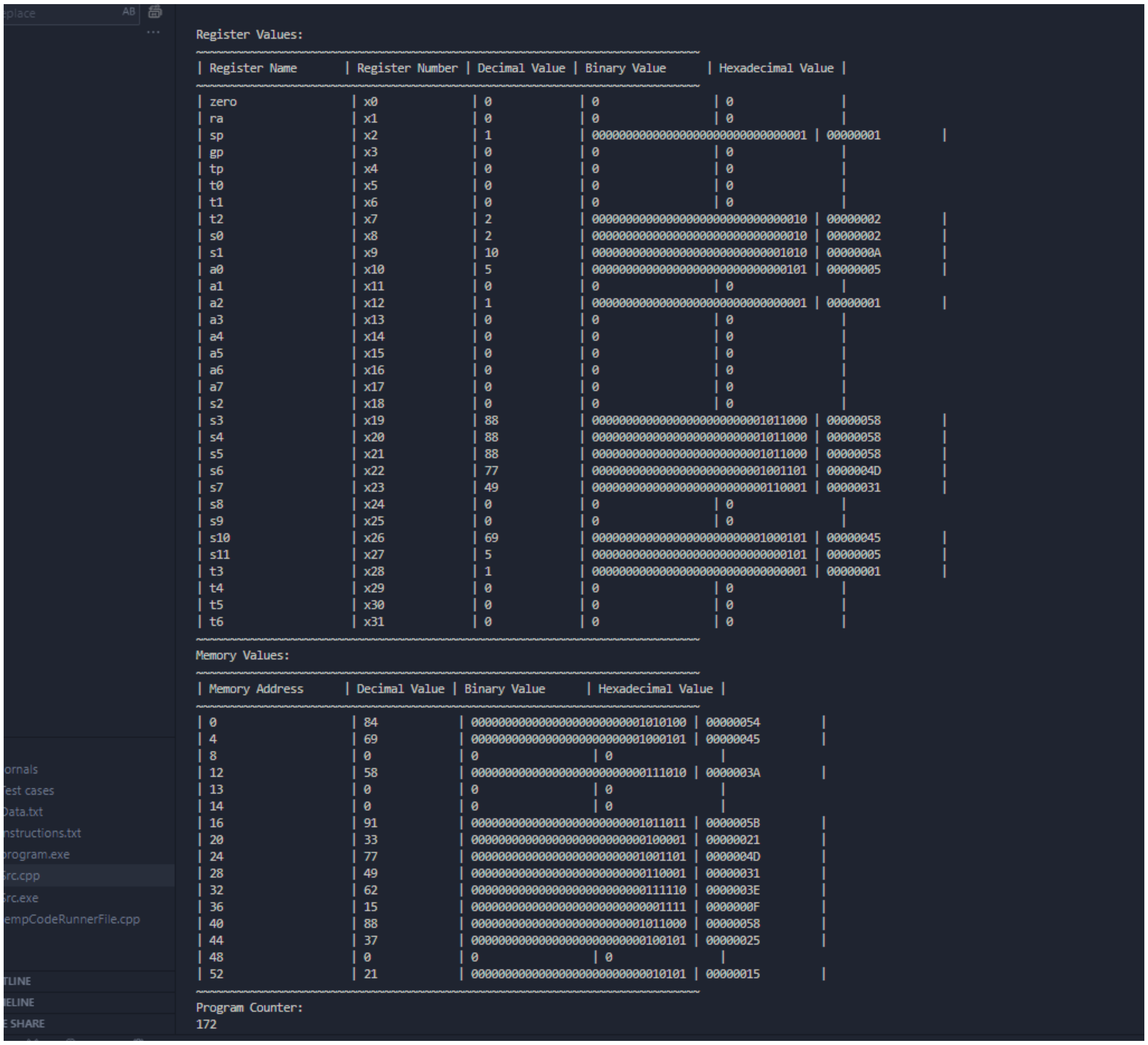
```
addi x1,x0,0
addi x2,x0,0
addi x3,x0,5
add x4, x0, x10
lw x5, 0(x4)
add x6, x1, x5
andi x7, x6, 20
andi x8, x7, 255
addi x9, x8, 5
addi x10, x9, 20
addi x11, x10, -10
addi x12, x11, -5
auipc x13, 20
lb x21, 040(x4)
lbu x22, 24(x4)
lh x23, 28(x4)
lui x25, 0
lw x26, 4(x4)
or x27, x2, x3
ori x28, x2, 1
mul x3, x4, x5
sub x17, x18, x19
sw x18, 8(x4)
xor x19, x20, x21
xori x20, x21, 0
sra x13, x14, x10
srai x14, x15, 2
srl x15, x16, x10
srli x16, x17, 2
rem x9, x10, x11
remu x10, x11, x12
sltu x11, x12, x13
sltiu x12, x13, 5
div x7, x8, x9
divu x8, x9, x10
mulhsu x6, x7, x8
mulh x4, x5, x6
mulhu x5, x6, x7
```

```
sll x31, x2, x3
slli x31, x2, 2
slt x1, x2, x3
slti x2, x3, 5
sh x30, 12(x12)
sb x29, 48(x4)
exit:
ebreak
```

Data.txt

```
0, 84
4, 69
8, 72
12, 58
16, 91
20, 33
24, 77
28, 49
32, 62
36, 15
40, 88
44, 37
48, 94
52, 21
```

Output simulation



```
Register Values:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| Register Name       | Register Number | Decimal Value | Binary Value      | Hexadecimal Value |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| zero                | x0              | 0             | 0                 | 0                 |
| ra                  | x1              | 0             | 0                 | 0                 |
| sp                  | x2              | 1             | 00000000000000000000000000000001 | 00000001          |
| gp                  | x3              | 0             | 0                 | 0                 |
| tp                  | x4              | 0             | 0                 | 0                 |
| t0                  | x5              | 0             | 0                 | 0                 |
| t1                  | x6              | 0             | 0                 | 0                 |
| t2                  | x7              | 2             | 00000000000000000000000000000010 | 00000002          |
| s0                  | x8              | 2             | 00000000000000000000000000000010 | 00000002          |
| s1                  | x9              | 10            | 00000000000000000000000000001010 | 0000000A          |
| a0                  | x10             | 5             | 00000000000000000000000000000101 | 00000005          |
| a1                  | x11             | 0             | 0                 | 0                 |
| a2                  | x12             | 1             | 00000000000000000000000000000001 | 00000001          |
| a3                  | x13             | 0             | 0                 | 0                 |
| a4                  | x14             | 0             | 0                 | 0                 |
| a5                  | x15             | 0             | 0                 | 0                 |
| a6                  | x16             | 0             | 0                 | 0                 |
| a7                  | x17             | 0             | 0                 | 0                 |
| s2                  | x18             | 0             | 0                 | 0                 |
| s3                  | x19             | 88            | 00000000000000000000000001011000 | 00000058          |
| s4                  | x20             | 88            | 00000000000000000000000001011000 | 00000058          |
| s5                  | x21             | 88            | 00000000000000000000000001011000 | 00000058          |
| s6                  | x22             | 77            | 00000000000000000000000001001101 | 0000004D          |
| s7                  | x23             | 49            | 00000000000000000000000000110001 | 00000031          |
| s8                  | x24             | 0             | 0                 | 0                 |
| s9                  | x25             | 0             | 0                 | 0                 |
| s10                 | x26             | 69            | 00000000000000000000000001000101 | 00000045          |
| s11                 | x27             | 5             | 00000000000000000000000000000101 | 00000005          |
| t3                  | x28             | 1             | 00000000000000000000000000000001 | 00000001          |
| t4                  | x29             | 0             | 0                 | 0                 |
| t5                  | x30             | 0             | 0                 | 0                 |
| t6                  | x31             | 0             | 0                 | 0                 |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Memory Values:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| Memory Address      | Decimal Value | Binary Value      | Hexadecimal Value |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
| 0                   | 84            | 00000000000000000000000001010100 | 00000054          |
| 4                   | 69            | 00000000000000000000000001000101 | 00000045          |
| 8                   | 0             | 0                 | 0                 |
| 12                  | 58            | 00000000000000000000000000111010 | 0000003A          |
| 13                  | 0             | 0                 | 0                 |
| 14                  | 0             | 0                 | 0                 |
| 16                  | 91            | 00000000000000000000000001011011 | 0000005B          |
| 20                  | 33            | 00000000000000000000000000100001 | 00000021          |
| 24                  | 77            | 00000000000000000000000001001101 | 0000004D          |
| 28                  | 49            | 00000000000000000000000000110001 | 00000031          |
| 32                  | 62            | 00000000000000000000000000111110 | 0000003E          |
| 36                  | 15            | 00000000000000000000000000001111 | 0000000F          |
| 40                  | 88            | 00000000000000000000000001011000 | 00000058          |
| 44                  | 37            | 00000000000000000000000000100101 | 00000025          |
| 48                  | 0             | 0                 | 0                 |
| 52                  | 21            | 00000000000000000000000000010101 | 00000015          |
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Program Counter:
172
```

(Successful)

Conclusion

In conclusion, the RISC-V simulator program presented here offers a robust and versatile platform for simulating the execution of RISC-V assembly code. By leveraging a well-structured class-based design, the program encapsulates the core functionalities required for loading, parsing, and executing instructions, as well as managing register values, memory contents, and program flow.

The implementation includes support for a wide range of RISC-V instructions, covering arithmetic, logical, memory access, control transfer, and bitwise operations. Additionally, bonus instructions such as multiplication, division, and remainder operations further enhance the functionality of the simulator. With comprehensive error handling mechanisms in place, the simulator ensures the integrity of the simulation process, providing informative error messages in case of invalid inputs or unexpected conditions.