

RISC-V RV32I Simulator

Mohamed Noureldin

Roa Abulmagd

Sara Mohamed

1. Brief description of the implementation:	3
2. Design implementation:	3
2.i. Memory and registers.....	3
2.ii. Conventional registers and labels maps	3
2.iii. global header file	4
2.iv. working with classes	4
3. User guide:	4
3.i. Running the program	5
.....	8
4. Problems to work on solving in the code	9
5. List of imulated programs	9
5.i. Test 1	9
5.ii. Test 2	9
5.iii. Test 3	10

1. Brief description of the implementation:

This project was mainly implemented to work as a simulator for RISC-V RV32I. So, it simulates the memory, and the 32-bit registers whose methodology of their implementation would be discussed later in detail within the design implementation section. It also represents the types of the instructions (R, S,B,...) as classes. As an input, It takes 1:2 files including both the code and the pre-saved values in memory- if available. Then it parses the code line by line while taking the program counter updating into consideration. It then saves the values of the memory and converts each register name into its index in the registers array, so that the program can start computing the output by calling the functions of each instruction from its class sending the proper parameters from rs1, rs2, imm, rd, labels, and so on. Then, it outputs the final output values of the PC, memory and registers along with the steps of computing them. In addition, we implemented two bonus features within the program. The first is outputting each value in decimal, hexadecimal and binary representation. Then the second is assembling the risc-v code into machine code.

2. Design implementation:

2.i. Memory and registers

The memory was implemented as an ordered map of <int, char> in order to act in accordance with the endianness of the Risc-V (little endian). So, the key holds the address and the value which is of type char (because the memory is byte addressable so each place in the memory should be 8 bits) holds the value saved in the memory.

The registers were implemented as an array of fixed size 32, since the registers size are previously determined.

2.ii. Conventional registers and labels maps

To be able to accept the user-input whether named in the original registers' names (x,x1,x2,...) or the conventional names (t0,t1,s1,...), The original names had to be mapped into their conventional ones in an ordered map to facilitate implementing the function **parse_register** that takes the name of any register and returns its index in the registers array.

For the purpose of handling the addresses of the labels, we decided to save them into an unordered map that maps their name to their addresses. This is to use them later in computing the result of the instructions that have a label as part of it; such as Btype and Jtype.

2.iii. global header file

We decided to create a header file for all the global variables (global.h)that should be accessed within all the program and include it in all other header files, and in the main as well.

2.iv. working with classes

The project was written using object-oriented programming to develop a clean code where each family is represented by a class that included all the functions of its instructions in addition to the print_Btype_machine_code function that both prints the machine code and saves it into the memory map. So that we can parse each line of code by calling the function of the current instruction being executed in the line by sending the values of registers, labels or immediate used in this function as parameters.

3. User guide:

In order for the program to start running, The user has to input one or two; one including the lines of code (a must), and the other including the values that should be pre-saved into the memory before starting to run the program(optional).

(The layout of the code files should be as presented in figure 1 below, and the memory file as in figure 2)

```
1  addi t1, t1, 256
2  addi t2, t2, 1
3  jal ra, jlabel
4  sh t1, 100(t2)
5  jlabel:
6      addi x2, t1, 3
7      fence x2, x3
8  slt x2, t2, t1
```

Figure 1

```
1  1  50
2  2  -10
3  30  30
4  40  40
5  50  50
```

Figure 2

3.i. Running the program

Step one :

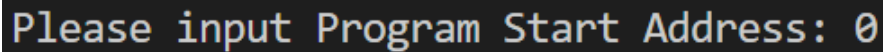
The user should start by writing the path of the first file including the risc-v code, as shown in figure 3 below.

```
Welcome to RISC-V Simulator
Please input code filename with full absolute path: test_code.txt
```

Figure 3

Step two:

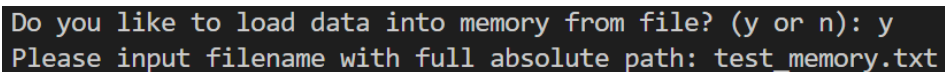
The user should input the value from which the program counter starts counting as shown in figure 4.



```
Please input Program Start Address: 0
```

Figure 4

Step three: if he/she wants to load data from the memory, they should put y and then input the exact path of the other file. Otherwise, they should input n. This is shown in figure 5.



```
Do you like to load data into memory from file? (y or n): y  
Please input filename with full absolute path: test_memory.txt
```

Figure 5

Output :

Then the program is expected to print the final output of the code along with all intermediate steps. In other words. After executing each line of code, the program prints the current values saved in the memory, in the 32 registers, the value of the program counter and the machine code representation of this line of code. All of these values are displayed in decimal, hexadecimal and binary. As shown below in figure 6,7,8 and 9.

```

Running: addl t1, t1, 256
t1, 256
t1
256
Machine code: 00001111111111000001100010011
Registers:
PC = 4
x0 = 0, x1 = 0, x2 = 2147483647, x3 = 0, x4 = 0, x5 = 0, x6 = 256, x7 = 0, x8 = 0, x9 = 0, x10 = 0, x11 = 0, x12 = 0, x13 = 0, x14 = 0, x15 = 0, x16 = 0, x17 = 0, x18 = 0, x19 = 0, x20 = 0, x21 = 0, x22 = 0, x23 = 0, x24 = 0, x25 = 0, x26 = 0,
x27 = 0, x28 = 0, x29 = 0, x30 = 0, x31 = 0
Pstatus:
4 = 15,
5 = -129,
6 = -1,
7 = 15,
1000 = 5,
1004 = 5,
1008 = 5,
1012 = 5,
1016 = 5,
1020 = 5,
1024 = 9,
Binary:
Registers:
PC = 00000000000000000000000000000000, x1 = 00000000000000000000000000000000, x2 = 01111111111111111111111111111111, x3 = 00000000000000000000000000000000, x4 = 00000000000000000000000000000000, x5 = 00000000000000000000000000000000, x6 = 00000000000000000000000000000000, x7 = 00000000000000000000000000000000, x8 = 00000000000000000000000000000000, x9 = 00000000000000000000000000000000, x10 = 00000000000000000000000000000000, x11 = 00000000000000000000000000000000, x12 = 00000000000000000000000000000000, x13 = 00000000000000000000000000000000, x14 = 00000000000000000000000000000000, x15 = 00000000000000000000000000000000, x16 = 00000000000000000000000000000000, x17 = 00000000000000000000000000000000, x18 = 00000000000000000000000000000000, x19 = 00000000000000000000000000000000, x20 = 00000000000000000000000000000000, x21 = 00000000000000000000000000000000, x22 = 00000000000000000000000000000000, x23 = 00000000000000000000000000000000, x24 = 00000000000000000000000000000000, x25 = 00000000000000000000000000000000, x26 = 00000000000000000000000000000000, x27 = 00000000000000000000000000000000, x28 = 00000000000000000000000000000000, x29 = 00000000000000000000000000000000, x30 = 00000000000000000000000000000000, x31 = 00000000000000000000000000000000
Pstatus:
4 = 00010011,
5 = 10000111,
6 = 11111111,
7 = 00001111,
1000 = 00000101,
1004 = 00000101,
1008 = 00000101,
1012 = 00000101,
1016 = 00000101,
1020 = 00000101,
1024 = 00001001,
Hex:
Registers:
PC = 0000000004
x1 = 00000000, x2 = 007ffffff, x3 = 00000000, x4 = 00000000, x5 = 00000000, x6 = 00000000, x7 = 00000000, x8 = 00000000, x9 = 00000000, x10 = 00000000, x11 = 00000000, x12 = 00000000, x13 = 00000000, x14 = 00000000, x15 = 00000000, x16 = 00000000, x17 = 00000000, x18 = 00000000, x19 = 00000000, x20 = 00000000, x21 = 00000000, x22 = 00000000, x23 = 00000000, x24 = 00000000, x25 = 00000000, x26 = 00000000, x27 = 00000000, x28 = 00000000, x29 = 00000000, x30 = 00000000, x31 = 00000000
Pstatus:
4 = 00000001,
5 = 00000000,
6 = 00000000,
7 = 00000000,
1000 = 00000000,
1004 = 00000000,
1008 = 00000000,
1012 = 00000000,
1016 = 00000000,
1020 = 00000000,
1024 = 00000000,

```

Figure 6

```

1004 = 00000000,
1008 = 00000000,
1012 = 00000000,
1016 = 00000000,
1020 = 00000000,
1024 = 00000000,
.....
Running: addl t2, t2, 1
t2, 1
t2
1
Machine code: 00000000001111000001100010011
Registers:
PC = 8
x0 = 0, x1 = 0, x2 = 2147483647, x3 = 0, x4 = 0, x5 = 0, x6 = 256, x7 = 1, x8 = 0, x9 = 0, x10 = 0, x11 = 0, x12 = 0, x13 = 0, x14 = 0, x15 = 0, x16 = 0, x17 = 0, x18 = 0, x19 = 0, x20 = 0, x21 = 0, x22 = 0, x23 = 0, x24 = 0, x25 = 0, x26 = 0,
x27 = 0, x28 = 0, x29 = 0, x30 = 0, x31 = 0
Pstatus:
4 = 15,
5 = -129,
6 = -1,
7 = 15,
8 = -109,
9 = -129,
10 = 15,
11 = 0,
1000 = 5,
1004 = 5,
1008 = 5,
1012 = 5,
1016 = 5,
1020 = 5,
1024 = 9,
Binary:
Registers:
PC = 00000000000000000000000000000000, x1 = 00000000000000000000000000000000, x2 = 01111111111111111111111111111111, x3 = 00000000000000000000000000000000, x4 = 00000000000000000000000000000000, x5 = 00000000000000000000000000000000, x6 = 00000000000000000000000000000000, x7 = 00000000000000000000000000000000, x8 = 00000000000000000000000000000000, x9 = 00000000000000000000000000000000, x10 = 00000000000000000000000000000000, x11 = 00000000000000000000000000000000, x12 = 00000000000000000000000000000000, x13 = 00000000000000000000000000000000, x14 = 00000000000000000000000000000000, x15 = 00000000000000000000000000000000, x16 = 00000000000000000000000000000000, x17 = 00000000000000000000000000000000, x18 = 00000000000000000000000000000000, x19 = 00000000000000000000000000000000, x20 = 00000000000000000000000000000000, x21 = 00000000000000000000000000000000, x22 = 00000000000000000000000000000000, x23 = 00000000000000000000000000000000, x24 = 00000000000000000000000000000000, x25 = 00000000000000000000000000000000, x26 = 00000000000000000000000000000000, x27 = 00000000000000000000000000000000, x28 = 00000000000000000000000000000000, x29 = 00000000000000000000000000000000, x30 = 00000000000000000000000000000000, x31 = 00000000000000000000000000000000
Pstatus:
4 = 00010011,
5 = 10000111,
6 = 11111111,
7 = 00001111,
8 = 10010011,
9 = 10000111,
10 = 00001111,
11 = 00000000,
1000 = 00000101,
1004 = 00000101,
1008 = 00000101,
1012 = 00000101,
1016 = 00000101,
1020 = 00000101,
1024 = 00001001,

```

Figure 7

4. Problems to work on solving in the code

There is only one issue that the memory map we did to simulate the memory only takes decimal(integer) values, and cannot store any other type.

5. List of simulated programs

5.i. Test 1

While loop for I<10. Followed by testing of all load types

```
addi x3,x0,0  # i = 0
```

```
addi x4,x0,10 # const 10
```

```
loop:
```

```
bge x3,x4, exit
```

```
addi x3,x3,1
```

```
jal x0,loop
```

```
exit: xor t2, x4, x0
```

```
xori t3, t2, 1
```

```
and x0, x3, x4
```

5.ii. Test 2

Program to multiply 3 by 3 to give 9 in a0.

```
ADDI x12,x0,3
```

```
ADDI x13,x0,3
```

JAL x0, MUL

MUL:

ADDI x5,x0,0

L1:

BGE x5,x13,EXIT

ADD x10,x10,x12

ADDI x5,x5,1

JAL x0, L1

EXIT:

ECALL

5.iii. Test 3

Program to clear array after providing the size of array in a1 and array starting address in a0.

addi a1, zero, 5

addi a0, zero, 1000

addi t0,zero,0

addi t5, zero, 10

L1:

slli t1,t0,2

add t2,a0,t1

```
sw zero, 0(t2)
```

```
addi t0,t0,1
```

```
blt t0,t5,L1
```

N.B: test 3 also takes values to be overwrite in the memory, so it requires the other following file

```
1000 10
```

```
1004 11
```

```
1008 12
```

```
1012 13
```

```
1016 14
```