



Project I

RISC V32I Simulator

Youssef Salah Balboul 900203051

Ismail Sabry 900222002

Hamza Abouheiba 900222153

Introduction:

Implementation:

Known issues / Limitations:

Guide:

Test Cases:

Test Case 1:

Test Case 2(loop):

Test Case 3: Lengthy loop, and test for ECALL, BLT, LB and SB

Bonus Features:

Output in Binary and Hexadecimal:

GUI:

Introduction:

The following report showcases our RISC-V32I simulator using C++. The list of functions implemented can be found [here](#). All 40 functions are implemented except for ECALL, EBREAK, and FENCE, which act as exit instructions for the program. The user is tasked with providing a starting memory address and a .txt file containing their code to be processed. This is discussed further in the Guide section of this report. You can find a link to our GitHub Repo [here](#).

Implementation:

We have made the following design choices for our program, in order to ensure convenience for both the user and us. First, we have assumed that the user is not going to require a memory space greater than the number of lines in their program. What this means is that for every instruction in the user's personal program, a word is preserved in memory that the user can manipulate using store and load instructions. Building on this, if the user will utilize branch instructions, they will have to be aware of the address of their Labels and personally add them to registers. The user will be able to see the address of each label during runtime for convenience's sake.

Known issues / Limitations:

As the user's memory space is limited, we have opted to initialize the Stack Pointer register to ensure smooth operation. The stack pointer will initialize during runtime to an increment of 12 of the entered starting address. If this is not done, the user might store elements in a memory space that does not exist. Thus, the stack is limited to 3 words.

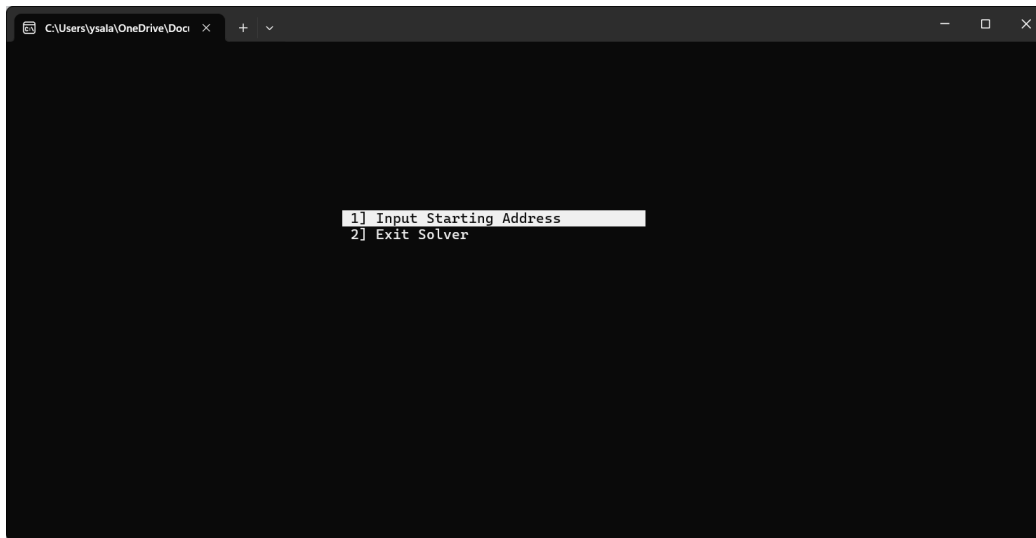
Spacing issue, for the program to work it should have only one space between the instruction name and its components. For example, for it to work *LUI a5,8*. Any space after commas will also result in an error. This also happens in branching Label it has to be only *Li:* , where i is any integer or letter, with no space after the colon. The registers have to use specific names (a0, s0, zero), and not X0, X1, and X2. Register names were pulled from the online BRISC simulator. Note that this code is unable to handle floating-point operations.

For the instructions SH, SB, LH and LB. The sign extension will work on the 16th or 8th bit respectively. So, if a positive value has a 1 in its utmost bit, a negative number will be loaded. The user should use LBU and LHU if they are not dealing with negative numbers.

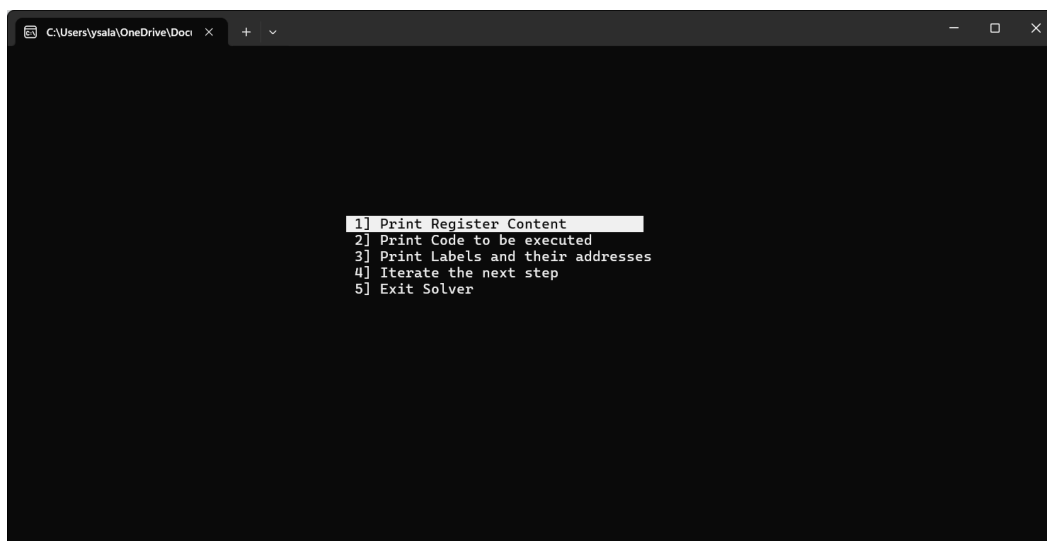
Guide:

Using the GUI the user can follow these steps to run the program:

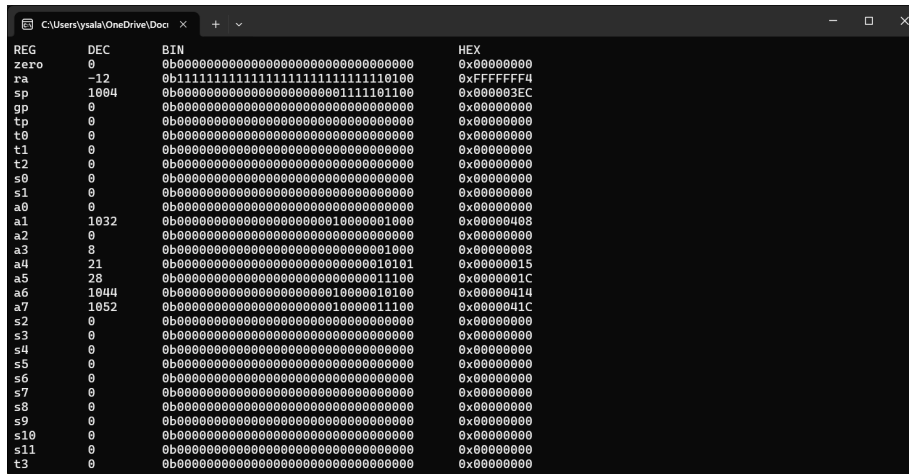
1. Store the instructions to be executed in a .txt file and name it “riscvcode.txt”
2. Ensure that the .txt file “registerstate.txt” is present. It should contain the 32 registers along with their initialized value of zero.
3. Launch RISCsim.exe. You’ll be greeted with the following menu:



4. Press Enter on the first option to enter the starting address
5. The following window displays 5 options.

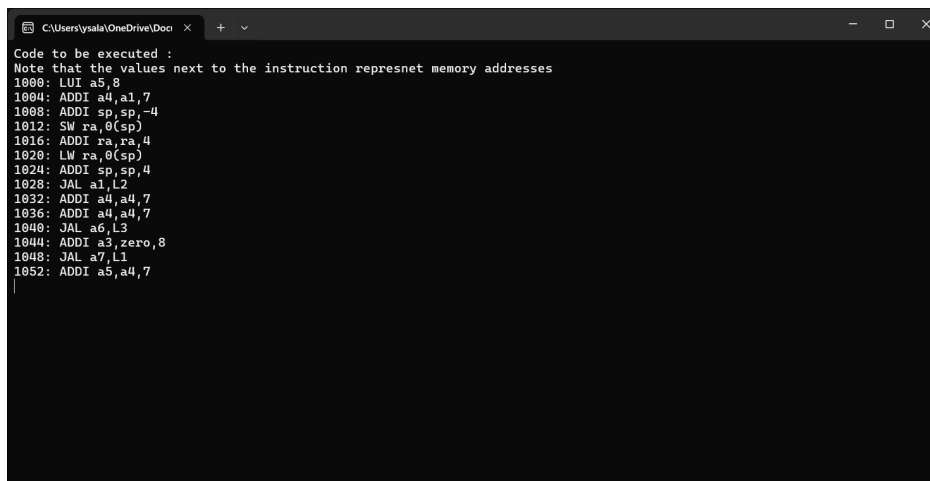


- The first option will display the register content in Decimal, Binary, and Hex after all the instructions have been executed.



REG	DEC	BIN	HEX
zero	0	0b00000000000000000000000000000000	0x00000000
ra	-12	0b11111111111111111111111111110100	0xFFFFF4
sp	1004	0b00000000000000000000000000000000	0x00003EC
gp	0	0b00000000000000000000000000000000	0x00000000
tp	0	0b00000000000000000000000000000000	0x00000000
t0	0	0b00000000000000000000000000000000	0x00000000
t1	0	0b00000000000000000000000000000000	0x00000000
t2	0	0b00000000000000000000000000000000	0x00000000
s0	0	0b00000000000000000000000000000000	0x00000000
s1	0	0b00000000000000000000000000000000	0x00000000
a0	0	0b00000000000000000000000000000000	0x00000000
a1	1032	0b00000000000000000000000000000000	0x0000408
a2	0	0b00000000000000000000000000000000	0x00000000
a3	8	0b00000000000000000000000000000000	0x0000008
a4	21	0b00000000000000000000000000000000	0x0000015
a5	28	0b00000000000000000000000000000000	0x000001C
a6	1004	0b00000000000000000000000000000000	0x0000014
a7	1052	0b00000000000000000000000000000000	0x000001C
s2	0	0b00000000000000000000000000000000	0x00000000
s3	0	0b00000000000000000000000000000000	0x00000000
s4	0	0b00000000000000000000000000000000	0x00000000
s5	0	0b00000000000000000000000000000000	0x00000000
s6	0	0b00000000000000000000000000000000	0x00000000
s7	0	0b00000000000000000000000000000000	0x00000000
s8	0	0b00000000000000000000000000000000	0x00000000
s9	0	0b00000000000000000000000000000000	0x00000000
s10	0	0b00000000000000000000000000000000	0x00000000
s11	0	0b00000000000000000000000000000000	0x00000000
t3	0	0b00000000000000000000000000000000	0x00000000

- The second option will display the instructions inside “riscvcode.txt”



```
Code to be executed :
Note that the values next to the instruction represnet memory addresses
1000: LUI a5,8
1004: ADDI a4,a1,7
1008: ADDI sp,sp,-4
1012: SW ra,0(sp)
1016: ADDI ra,ra,4
1020: LW ra,0(sp)
1024: ADDI sp,sp,4
1028: JAL a1,L2
1032: ADDI a4,a4,7
1036: ADDI a4,a4,7
1040: JAL a6,L3
1044: ADDI a3,zero,8
1048: JAL a7,L1
1052: ADDI a5,a4,7
```

- The third option will print the Labels inside “riscvcode.txt” and their allocated addresses.
The user should utilize this option when using branch instructions.
- The fourth option will execute the next line in the user program.
- Error messages will appear on the top left corner of the screen. NOTE: some error messages do not cause an immediate exit.

Test Cases:

Test Case 1:

Riscv Code with starting address 44:

```
≡ riscvcode.txt M ×  ≡ RISCsim.cpp M  ≡ RISCsim.exe M
≡ riscvcode.txt
1  main:
2      LUI a5,8
3      ADDI a3,zero,7
4      SUB a4,a3,zero
5      AND a2,a3,zero
6      XOR a4,a1,zero
7      ADDI sp,sp,-4
8      SW ra,0(sp)
9      ADDI ra,ra,4
10     LW ra,0(sp)
11     ADDI sp,sp,4
12     JAL a1,L2
13     BEQ zero,a3,L2
14 L1:
15     ADDI a4,a4,7
16     ADDI a4,a4,7
17     JAL a7,main
18 L2:
19     ADDI a6,a6,20
20     JALR a6,s2,32
21 L3:
22     ADDI a5,a4,7
```

REG	DEC	BIN	HEX
zero	0	0b00000000000000000000000000000000	0x00000000
ra	0	0b00000000000000000000000000000000	0x00000000
sp	56	0b00000000000000000000000000000000111000	0x00000038
gp	0	0b00000000000000000000000000000000	0x00000000
tp	0	0b00000000000000000000000000000000	0x00000000
t0	0	0b00000000000000000000000000000000	0x00000000
t1	0	0b00000000000000000000000000000000	0x00000000
t2	0	0b00000000000000000000000000000000	0x00000000
s0	0	0b00000000000000000000000000000000	0x00000000
s1	0	0b00000000000000000000000000000000	0x00000000
a0	0	0b00000000000000000000000000000000	0x00000000
a1	88	0b000000000000000000000000000000001011000	0x00000058
a2	0	0b00000000000000000000000000000000	0x00000000
a3	7	0b000000000000000000000000000000000111	0x00000007
a4	0	0b00000000000000000000000000000000	0x00000000
a5	7	0b000000000000000000000000000000000111	0x00000007
a6	112	0b000000000000000000000000000000001110000	0x00000070
a7	0	0b00000000000000000000000000000000	0x00000000
s2	0	0b00000000000000000000000000000000	0x00000000
s3	0	0b00000000000000000000000000000000	0x00000000
s4	0	0b00000000000000000000000000000000	0x00000000
s5	0	0b00000000000000000000000000000000	0x00000000
s6	0	0b00000000000000000000000000000000	0x00000000
s7	0	0b00000000000000000000000000000000	0x00000000
s8	0	0b00000000000000000000000000000000	0x00000000
s9	0	0b00000000000000000000000000000000	0x00000000
s10	0	0b00000000000000000000000000000000	0x00000000
s11	0	0b00000000000000000000000000000000	0x00000000
t3	0	0b00000000000000000000000000000000	0x00000000
t4	0	0b00000000000000000000000000000000	0x00000000
t5	0	0b00000000000000000000000000000000	0x00000000
t6	0	0b00000000000000000000000000000000	0x00000000

PC counter For the Above cycle:116

Memory addresses:

44: 0

48: 0

52: 0

56: 0

60: 0

64: 0

PC counter For the Above cycle:116

Memory addresses:

44: 0

48: 0

52: 0

56: 0

60: 0

64: 0

68: 0

72: 0

76: 0

80: 0

84: 0

88: 0

92: 0

96: 0

100: 0

104: 0

108: 0

112: 0

Test Case 2(loop):

Riscv with loop starting address 1000.

```
1  main:
2      auipc a6,5
3      addi a5,zero,7
4      addi a4,zero,6
5  L1:
6      slti a1,a0,10
7      sltiu a2,a0,10
8      beq a1,zero,END
9      ADDI a3,a3,6
10     ori a1,a1,0
11     or a1,a1,a1
12     andi a5,a5,7
13     addi a0,a0,1
14     xori a4,a4,3
15     bne a0,zero,L1
16  END:
17     addi a7,zero,1
```


REG	DEC	BIN	HEX
zero	0	0b00000000000000000000000000000000	0x00000000
ra	0	0b00000000000000000000000000000000	0x00000000
sp	1012	0b0000000000000000000000000111110100	0x000003F4
gp	0	0b00000000000000000000000000000000	0x00000000
tp	0	0b00000000000000000000000000000000	0x00000000
t0	0	0b00000000000000000000000000000000	0x00000000
t1	0	0b00000000000000000000000000000000	0x00000000
t2	0	0b00000000000000000000000000000000	0x00000000
s0	0	0b00000000000000000000000000000000	0x00000000
s1	0	0b00000000000000000000000000000000	0x00000000
a0	10	0b0000000000000000000000000000001010	0x0000000A
a1	0	0b00000000000000000000000000000000	0x00000000
a2	0	0b00000000000000000000000000000000	0x00000000
a3	60	0b000000000000000000000000000000111100	0x0000003C
a4	6	0b0000000000000000000000000000000110	0x00000006
a5	7	0b0000000000000000000000000000000111	0x00000007
a6	4116480	0b000000000111101101000000000000	0x003ED000
a7	1	0b0000000000000000000000000000000001	0x00000001
s2	0	0b00000000000000000000000000000000	0x00000000
s3	0	0b00000000000000000000000000000000	0x00000000
s4	0	0b00000000000000000000000000000000	0x00000000
s5	0	0b00000000000000000000000000000000	0x00000000
s6	0	0b00000000000000000000000000000000	0x00000000
s7	0	0b00000000000000000000000000000000	0x00000000
s8	0	0b00000000000000000000000000000000	0x00000000
s9	0	0b00000000000000000000000000000000	0x00000000
s10	0	0b00000000000000000000000000000000	0x00000000
s11	0	0b00000000000000000000000000000000	0x00000000
t3	0	0b00000000000000000000000000000000	0x00000000
t4	0	0b00000000000000000000000000000000	0x00000000
t5	0	0b00000000000000000000000000000000	0x00000000
t6	0	0b00000000000000000000000000000000	0x00000000

PC counter For the Above cycle:1056

Memory addresses:

1000: 0

1004: 0

1008: 0

1012: 0

1016: 0

1020: 0

Test Case 3: Lengthy loop, and test for ECALL, BLT, LB and SB

```

≡ riscvcode.txt
1  main:
2      ADDI a3,zero,20
3      ADDI a2,zero,12
4      SB a3,0(a2)
5      ADDI a4,zero,3
6      ADDI a3,a2,5
7      LB a3,0(a2)
8      ADDI a1,a1,1
9      BLT zero,a3,L1
10     ECALL a3,zero
11     SUB a3,a3,zero
12  L1:
13     ADDI a5,a1,1
14     BNE a4,a1,main

```

REG	DEC	BIN	HEX
zero	0	0b000000000000000000000000000000	0x00000000
ra	0	0b000000000000000000000000000000	0x00000000
sp	12	0b0000000000000000000000000000001100	0x0000000C
gp	0	0b000000000000000000000000000000	0x00000000
tp	0	0b000000000000000000000000000000	0x00000000
t0	0	0b000000000000000000000000000000	0x00000000
t1	0	0b000000000000000000000000000000	0x00000000
t2	0	0b000000000000000000000000000000	0x00000000
s0	0	0b000000000000000000000000000000	0x00000000
s1	0	0b000000000000000000000000000000	0x00000000
a0	0	0b000000000000000000000000000000	0x00000000
a1	2	0b0000000000000000000000000000000010	0x00000002
a2	12	0b0000000000000000000000000000001100	0x0000000C
a3	20	0b00000000000000000000000000000010100	0x00000014
a4	3	0b00000000000000000000000000000000011	0x00000003
a5	3	0b00000000000000000000000000000000011	0x00000003
a6	0	0b000000000000000000000000000000	0x00000000
a7	0	0b000000000000000000000000000000	0x00000000
s2	0	0b000000000000000000000000000000	0x00000000
s3	0	0b000000000000000000000000000000	0x00000000
s4	0	0b000000000000000000000000000000	0x00000000
s5	0	0b000000000000000000000000000000	0x00000000
s6	0	0b000000000000000000000000000000	0x00000000
s7	0	0b000000000000000000000000000000	0x00000000
s8	0	0b000000000000000000000000000000	0x00000000
s9	0	0b000000000000000000000000000000	0x00000000
s10	0	0b000000000000000000000000000000	0x00000000
s11	0	0b000000000000000000000000000000	0x00000000
t3	0	0b000000000000000000000000000000	0x00000000
t4	0	0b000000000000000000000000000000	0x00000000
t5	0	0b000000000000000000000000000000	0x00000000
t6	0	0b000000000000000000000000000000	0x00000000

PC counter For the Above cycle:40

Memory addresses:

0: 0
4: 0
8: 0
12: 20
16: 0
20: 0
24: 0
28: 0
32: 0

Bonus Features:

Output in Binary and Hexadecimal:

We created a function that receives the decimal value and converts it to binary and hexadecimal through typical conversion methods and it also deals with negative numbers. Negative numbers are dealt with using the two's complement method in the binary case and in the hexadecimal case, two's complement is applied, and then after that it is converted to hexadecimal through normal hexadecimal conversion.

GUI:

We added a GUI that creates a menu where the user can choose what to do and the options are: Input Address, Print Register with memory, print the user's program, print labels with their address, and exit. This menu contains the whole functionality of the code as each option is dealt with as a case, so each case contains the corresponding functions/code that match the chosen option. This menu was derived from a YouTube channel called Troubleshooter Youssef Shawky. It utilizes header files windows.h and conio.h and you traverse the menu using the arrow keys and choose an option through the enter button. The GUI is implemented mainly with 2 functions, ShowMenu and ShowResult. The integration of the menu with our simulator code and the options of the menu are our own original creations.