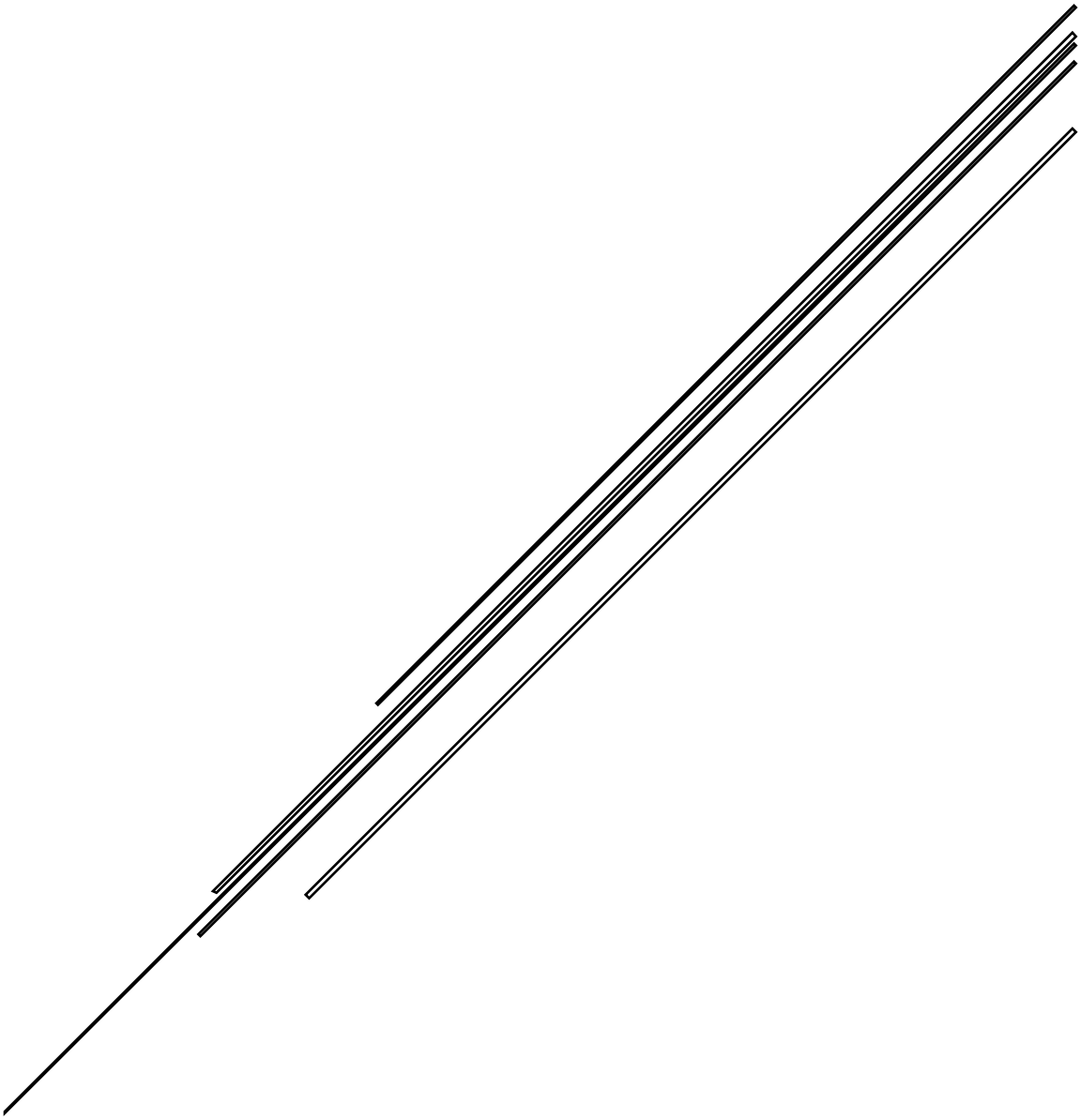
FARAH KABESH – 900191706
KARIM SHERIF – 900191474
YASEEN AHMED – 900201235

COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE PROGRAMMING

RISC-V RV32I SIMULATOR

PROJECT REPORT

SPRING 2022



Project Summary

The aim of this project is to create a functional RISC-V simulator which supports the RV32I base integer instruction set. The 6 RISC-V instruction formats will be implemented; R-type, I-type, S-type, B-type, U-type, and J-type. The simulator is designed to execute 40 user-level instructions, which can be found below.

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

Implementation Language:

This simulator is programmed using C++, which is a general-purpose programming language.

Description of Implementation:

The simulator starts by initializing the registers to zero, which is done using the function *void init_registers()*. After the initialization of registers, the function *void run_program()* is called and the user is required to input a file that contains the assembly program they wish to run on the simulator. The function *bool validate_file(string path)* is used to validate that the file exists. If the file path is valid, then the simulator will proceed with reading the program from the inputted file through the function *void read_file(string file_path)* which is used to find labels and store instructions. Moreover, the user will be required to input which memory address they would like to start from. This memory address will then be stored in the program counter. The user will also be asked whether they have any variables to add. If yes, then the user will be prompted to enter the number of variables, the address of each variable and the value stored inside. Map data structures are used to store the memory which contains the instructions, labels and their respective addresses as well as the 32 RISC-V registers.

When the data has been inputted, read and stored, the simulator will proceed to the function *void assemble_code* which is used to process the instruction and send it to its corresponding operation. This function also checks if any halting instructions, such as ECALL, EBREAK, and FENCE are found in the instruction.

The simulator consists of 5 functions which are responsible for the different operations of the instructions. These 5 functions are:

1. *void assemble_AL(string)*
   Includes arithmetic and logic operations.
   R-type instructions: add, sub, sll, slt, sltu, xor, srl, sra, or, and
   I-type instructions: addi, slti, sltiu, xori, ori, andi, slli, srli, srai

2. *void assemble_branching(string)*
   Includes branching operations.
   B-type instructions: beq, bge, blt, bltu, bne, bgeu

3. *void assemble_LS(string)*
   Includes load and store operations.
   S-type instructions: sb, sh, sw
   I-type instructions: lb, lbu, lh, lhu, lw

4. *void assemble_J(string)*:
   Includes jump operations.
   J-type instructions: jal, jalr

5. *void assemble_U(string)*
   Includes operations with upper immediates.
   U-type instructions: lui, auipc

These 5 functions identify the operations and execute the instructions using if and else-if statements. These functions also keep track of the program counter value, the register file contents and the memory contents. The program counter is incremented by 4 for each instruction.

Subsequently, the input program's execution is simulated and the function *void get_main_info()* is called and repeatedly outputs the values of the program counter, the contents of the memory and the contents of the registers.

Bonus feature #3 has been implemented, so that the output values are displayed in decimal, binary and hexadecimal; instead of just decimal representation which is considered the default. The function *string int_to_hex(int i)* converts the output from decimal to hexadecimal form, whilst the function *string int_to_bin(int n)* converts the output from decimal to binary form.

Design Decisions/Assumptions:

The assumptions made in order for the simulator to be functional is as follows:

- There are 32 registers in the simulator and they are referred to as x0 – x31. They should not be referred to using their symbolic names, such as t0, a0, or s0.
- Programs inputted by the user should be contained in text files (.txt).
- The user needs to input the values stored in the memory and their addresses.
- Instructions in the program inputted by the user should be written in the format 'add x5, x6, x7' where registers should be separated by a comma and a single space.
- In the case of a label, there should be no spaces between the label, the colon and the instruction. For example: LOOP:sub x5, x6, x7

Bugs/Issues:

The simulator does not have any known issues or bugs.

User Guide:

To begin with, the user is required to enter the path of the file that contains their program. The program should be saved as a text file and should contain the RISC-V assembly instructions.

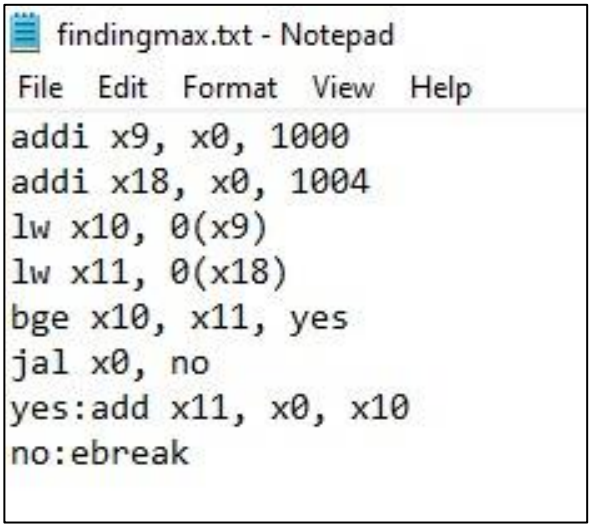


Fig. 1 – Example of a RISC-V program

The user will then be asked to enter the memory address they wish to start from. This memory address will then be stored in the program counter.

Afterwards, the user will be asked if they have any variables they wish to add. The user should then input either ‘y’ for yes or ‘n’ for no. If the user inputs ‘y’, they will be prompted to enter the number of variables they want to add, the address of each variable and the value to be stored inside this variable.

The simulator will then execute the instructions and will repeatedly output the contents of the memory, contents of the registers and the program counter until the program ends. The final results will be displayed in decimal, hexadecimal and binary representations.

Fig. 2 – Input & Output of FindingMax Program
(Screenshot 1)

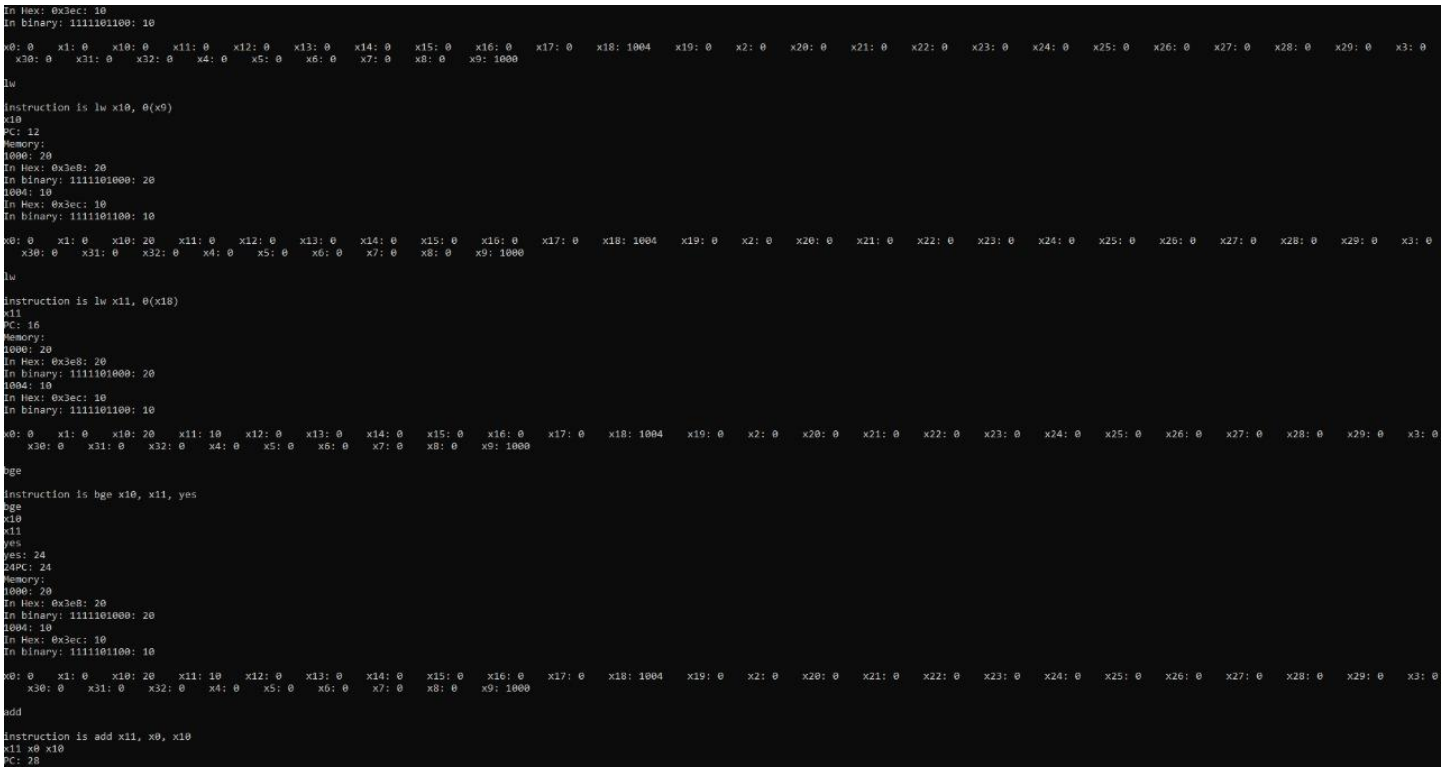


Fig. 3 – Input & Output of FindingMax Program
(Screenshot 2)

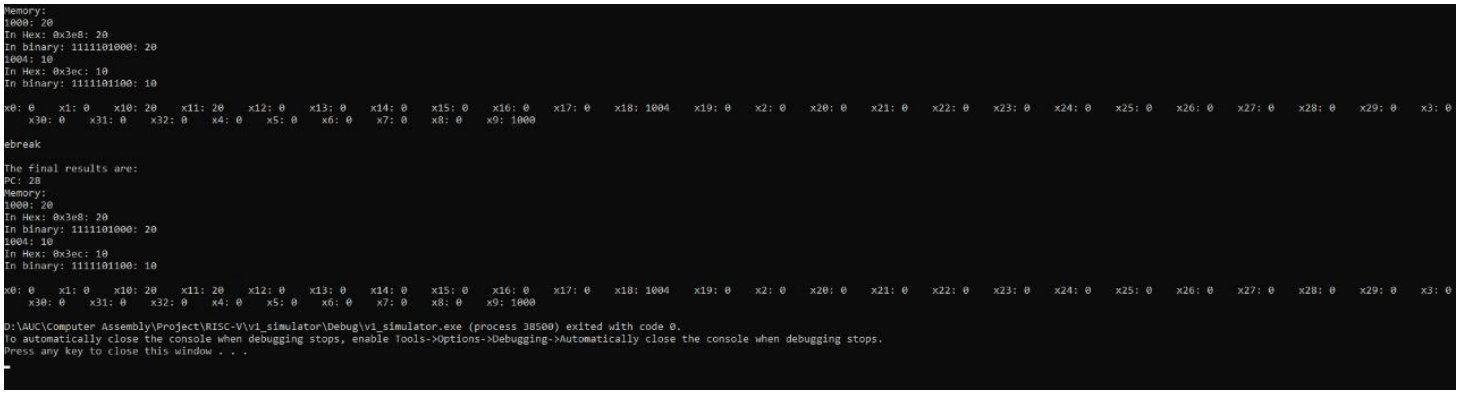


Fig. 4 – Input & Output of FindingMax Program
(Screenshot 3)

List of Programs Simulated:

1. **Finding Max**

   Function: finds the maximum value between 2 numbers.

   File Name: findingmax.txt

2. **Finding Min**

   Finds the minimum value between 2 numbers.

   Function: findingmin.txt

3. **Swap**

   Function: swaps the locations of 2 elements.

   File Name: swap.txt