Computer Organization and Assembly Language
Project 1- RISC-V Simulator
Dana Alkhouri- 900214106
Mhamad Khalil- 900214275

The program was done in C++ language. The program's implementation includes 40 instructions in RISC-V that were divided between us (Dana and Mhamed). The implemented bonus parts were no.3 and no.7 in the bonus part.

How to run the code:

To simulate the assembler correctly, the RISC V program should be written in a txt file named (program.txt). The user should ensure that the opcodes of the instruction are all written in capital letters, with no empty lines between instructions. The space after the opcode and the commas between the parameters are mandatory (spaces between parameters are unimportant and will not affect the program execution). Labels in the assembly code are not supported, and the instructions must end with FENCE, ECALL, or EBREAK. Otherwise, the simulation will get into an infinite loop. The only instructions allowed are the 40 instructions found in the table (page 148) in the PDF.

Mhamad's Part:

My part included working on extracting the output from the txt file and parsing it. I created a struct (instruction) for the instructions that supports all the parameters an instruction may need (rd, r1, r2, immediate, and opcode). I filled these attributes with their values through parsing. I also figured out a way to call the needed function dynamically and more efficiently using unordered_maps with Function Pointers as values to string keys. The parsing functions were divided into 7 groups according to the table in the link attached with the project prompt. I also worked on creating a vector of instructions to serve as an iterable data structure with the PC index. After parsing, each instruction is pushed to this vector. In addition, I worked on the program counter and ensured our program was byte-addressable. Furthermore, I implemented a loop that fills the memory in a way that makes it in a little-endian format, and worked on taking initial memory input from the user. Finally, I worked on the execution flow of the RISC V

instructions and implemented 20 instructions as functions in C++ (the first 20 functions in the table).

My experience in the project:

Working on this project was overwhelming overall, as delaying it many times was inconvenient. Moreover, there was a huge lack of communication between my teammate and me, so we ended up having an unfair work distribution and, in some cases, one of us worked on some functionalities that were not used in the program or by the other teammate. However, I learned a lot at the end since I discovered some features in C++ that are very efficient and convenient, and I revised all 40 instructions covered and the memory format.

Dana's Part

In the project, I contributed to implementing the memory in which we use the function to write and read from and to the memory. I used the unordered map for the memory of an unsigned int for the address of 32 bits and a signed integer for the value of 8 bits; in addition to that, I had to implement the write to the memory from a text file that read from the file an address and a value and write it to the memory using the WriteNBytes function. Implementing the WriteNBytes takes the value, then the value & 0xFF (which is '11111111'), and stores it in the memory address. Then, we shift the value by 8 bits to write the other 8 bits (1 byte) to the other addresses of the memory. [supposed to be used in the Load Word instruction]. The other part I implemented is the readNBytes, which is the opposite of the WriteNBytes. It changes the values of the register based on the values stored in the memory [supposed to be used in Store Word instruction]. In addition, I implemented 18 instructions in RISC-V with the last three instructions, FENCE, EBREAK, and ECALL, that have similar implementations. In addition, one of the essential things I implemented is the registers used in almost every function (from instructions to memory and others). The registration was implemented using a class, and in this class, I overridden an operator to ensure that the value of X0 wouldn't change.

I used this link to help me understand how each instruction works to be able to convert it into a C++ code (https://msyksphinz-self.github.io/riscv-isadoc/html/rvi.html#sll).
The memory outputting was done in PrintMemory(), then it was called in the output(), which contains the output of the program counter and registers and the values of the registers in 3 ways: decimal, hexadecimal, and binary.
The last thing I have contributed to doing is the other bonus part, which is writing C code programs with its assembly language. All these files from C and assembly languages code are in the file. I have implemented 6 C programs: SumOfArray, Maximum number, Swapping, Sum two numbers, GCD, and Fibonacci.

- Please check the files for the C code programs for the bonus part and the Assembly language code included in the files.

I choose straightforward algorithms to help me implement the memory, from writing to reading. Also, writing a C code and converting it into an Assembly language code helped me improve my assembly language skills as I was finding it difficult to do so. The main difficulty is understanding how memory works at first sight. I knew the idea of it and how it works, but I transferred it into a C code.