The American University in Cairo
CSCE2303 – Computer Organization and Assembly Language Programming
Spring 2022
Project 1: RISC-V RV32I Simulator

Alaa Alkady - 900191122
Salma Zaghlou - 900183256
Omar Fayed - 900191831

Table of Contents

1. Program Description (+2bouns)

This program is written in c++ programming language. The program uses the **map** data structure to create both the memory and registers. In addition, it includes an ***instructions*** **struct** which includes the (rd, rs1, rs2, the operation, pc, base, immediate, and offset). So, instead of creating 6 structures for the 6 instruction types, we created one general structure that uses certain parameters according to the desired instruction type. The program starts by reading the file containing the RISC-V instructions which then gets divided and stored in a vector to easily call the desired functions with their parameters. There are 40 functions implemented, representing the 40 instructions of the RISC-V. The program, accordingly, calls the desired instruction through the "function_choose" function. After identifying the instruction type, the register with the needed parameters is passed to the desired function. These functions then implement the exact instruction through certain conditions, which differ according to the called instruction. Further, they use the register and memory maps to modify the registers and/or memory (in case of the loading and storing instructions) according to the program instructions. Finally, the function returns the updated program count (PC).

This program implements 2 bonuses which are bonus 3 and 7. For bonus 3, the program includes functions which turn all the out-putted values into binary and hexadecimal instead of only decimals. The "decToBinary" function takes the decimal input and applies the needed calculations to output a 32-bit binary equivalent. Similarly, the "decToHexa" function takes the decimal input and calculates the hexadecimal equivalent. As for bonus 7, the program was used to implement 6 different RISC-V programs, which include a variety of all different instruction types. (Refer to section 5. List of Simulated Programs)

2. Designs and Assumptions

1) The file containing the instruction should be in the following format:

add x1 , x2 , x3

Where a space should separate the operation/instruction from the registers, in addition to spaces separating the commas from the registers on each side.

2) At the start of a new line of instruction, there should not be any spades or tabs:

addi x2 , x3 , 4

sub x9 , x9 , x2

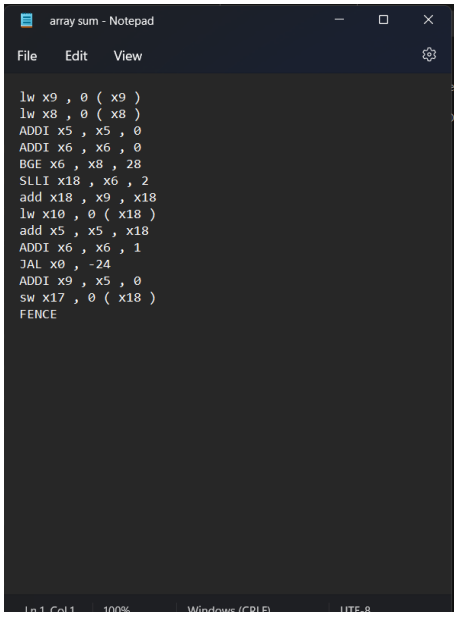
3) The register x0 always holds the value 0, unchangeable.
4) The registers x5-x7 & x28-x31 are temporary registers, while x8-x9 & x18-x27 are saved registers, and x10-x17 hold function arguments.
5) All the registers used are from x0 to x31, whereas users cannot use a0-a31 or t0-t31 for any registers.
6) The instructions should be saved in a .txt file to be executed.
7) We designed our program to handle the labels by reading numbers instead of words.
8) The user need to input the initial program count (PC) address in hexadecimal form (ex: 0x00000004)

3. Bugs or Issues

This program doesn't have any bugs or issues.

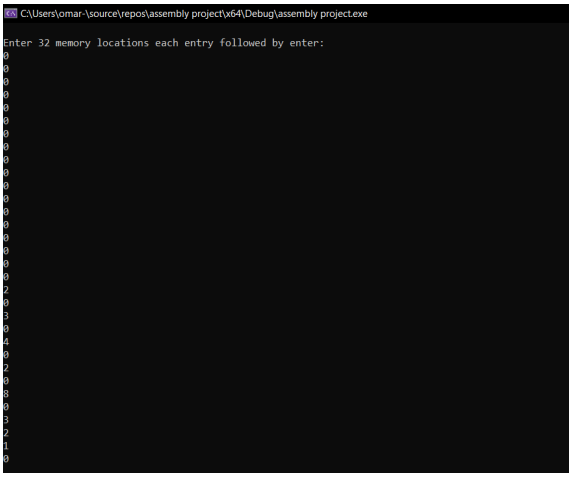
4. User Guide

The implemented RISC-V simulator supports all the 40 assembly instructions. The whole program is contained in one file, named "read.cpp". Before executing the file, the user needs to create a txt file filled with the set of instructions. Then, the user needs to edit the file path in the "read.cpp" file.



A sample RISC-V program in a txt file

After executing the program the user is asked to enter the starting address of the program count (PC) in hexadecimal (ex: 0x00000004). Moreover, the user is asked to input any values needed in the memory file.



User input in the memory

Subsequently, the program starts executing every instruction in the text file. After the execution of each instruction, the content of the register file, the memory, and the program count (PC) are displayed on the screen. This eases the process of tracing the instructions for the user. In addition, all the output values are displayed in decimal, binary and hexadecimals.

5. List of Simulated Programs

1) Swap

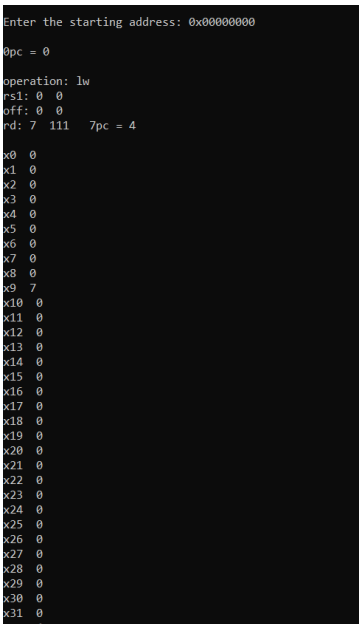
- **File Name:** swap2
- **C++:**

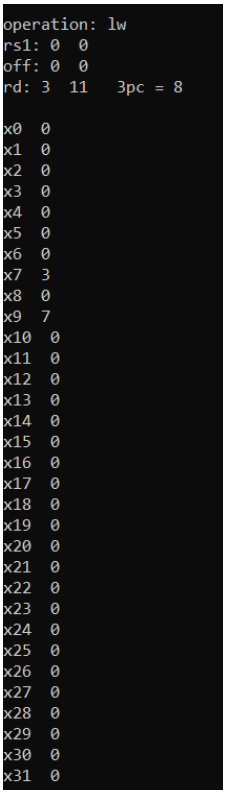
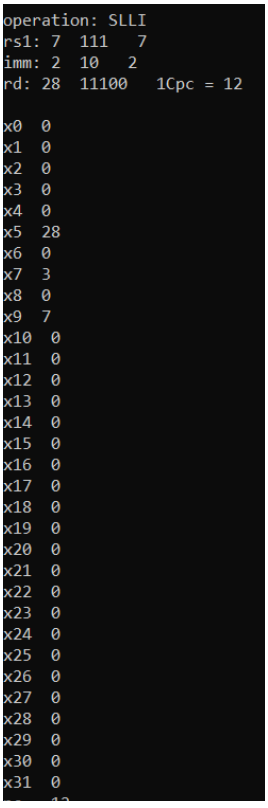
```
void swap(int v[], int k)
{
        int temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
}
```

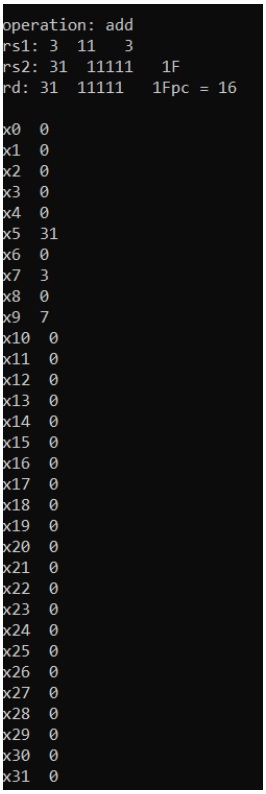
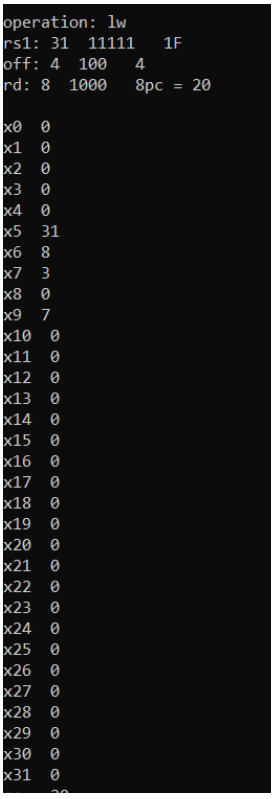
- **RISC-V:**

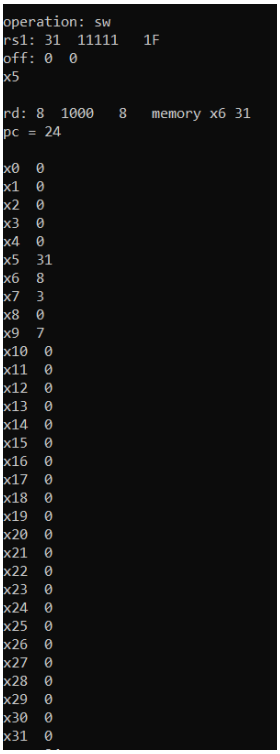
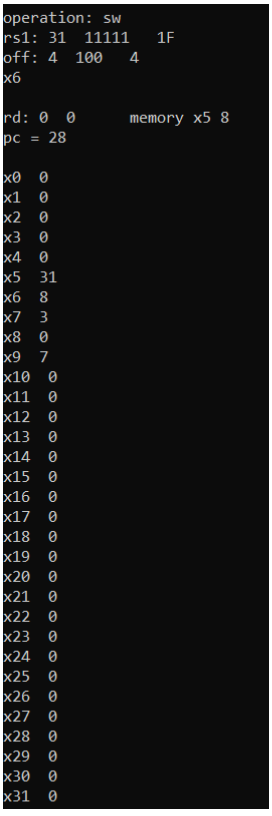
```
lw x9 , 0( x4 )
lw x7 , 0( x7 )
SLLI x5 , x9 , 2
add x5 , x7 , x5
lw x6 , 4 ( x5 )
sw x6 , 0 ( x5 )
sw x5 , 4 ( x5 )
FENCE
```

- **Program simulation screenshots**

1.	lw x9 , 0(x4)	

2.	lw x7 , 0(x7)	
3.	SLLI x5 , x9 , 2	

4.	add x5 , x7 , x5	
5.	lw x6 , 4 (x5)	

6.	sw x6 , 0 (x5)	
7.	sw x5 , 4 (x5)	
8.	FENCE	Terminates the program

2) **Max number finder**

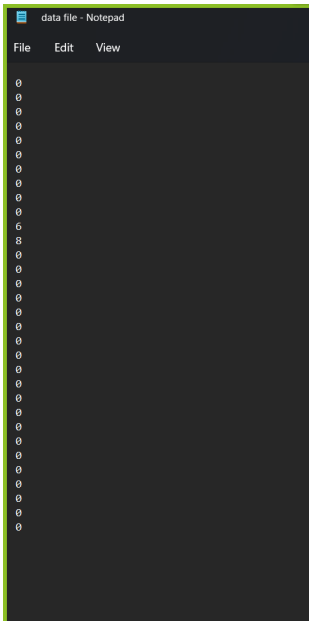
- **File Name:** find max
- **C++:**

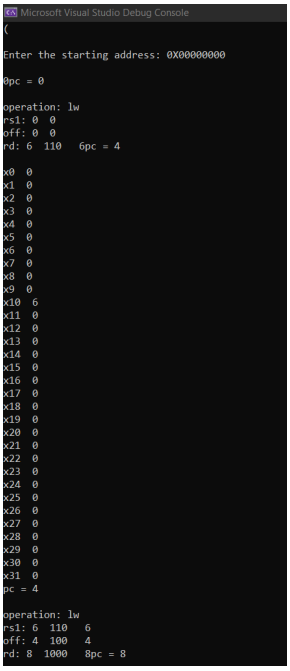
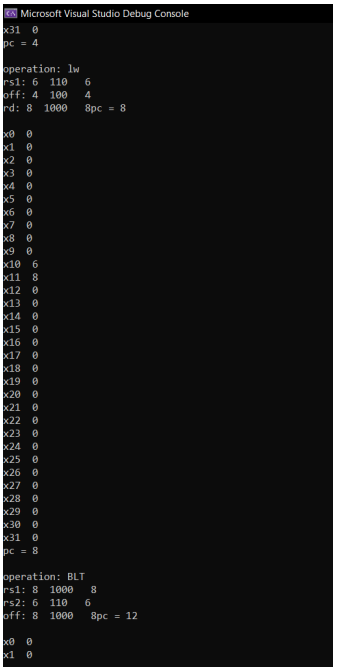
```
int max(a, b)
{
        if (a > b)
                return a;
        else return b;
}
```

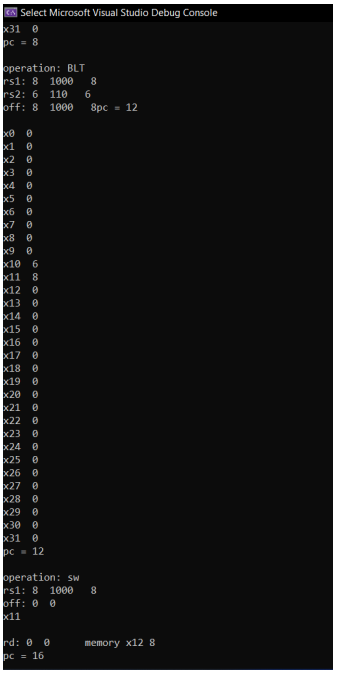
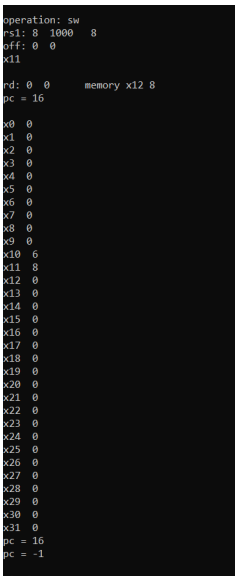
- **RISC-V:**

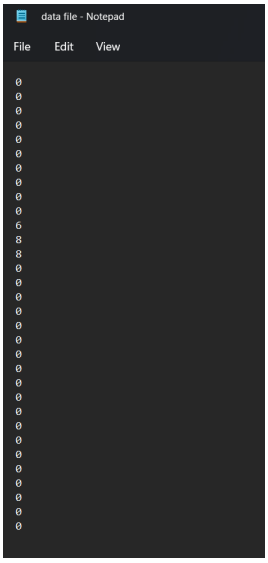
```
lw x10 , 0 ( x10 )
lw x11 , 4 ( x10 )
BLT x11 , x10 , 8
sw x12 , 0 ( x11 )
EBREAK
sw x12 , 0 ( x10 )
EBREAK
```

- **Program simulation screenshots**

	The memory before the execution the program	

1.	lw x10 , 0 (x10)	
2.	lw x11 , 4 (x10)	

3.	BLT x11 , x10 , 8	
4.	sw x12 , 0 (x11)	
5.	EBREAK	Terminates the program

	The memory file after execution	

3) Sum of array integers

- **File Name:** array sum
- **C++:**

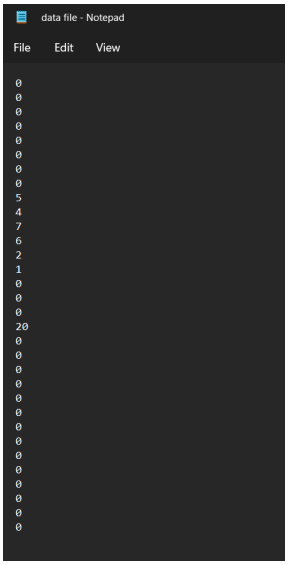
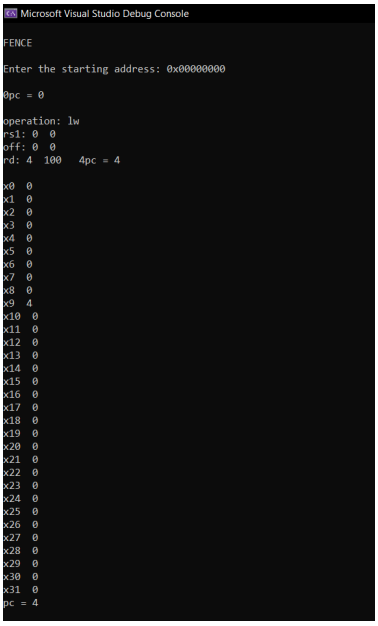
```
int arraysum(int a[], int size)
{
        int ret = 0;
        int i;
   for (i = 0;i < size;i++) {
      ret = ret + a[i];
   }
        return ret;
}
```

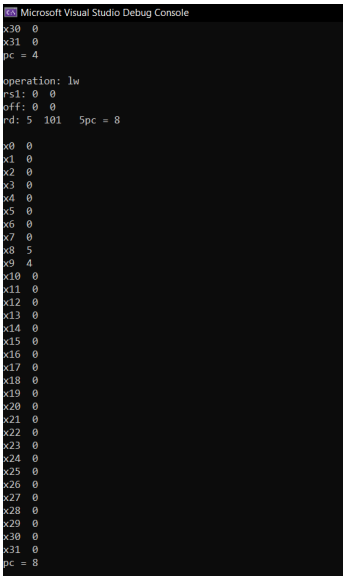
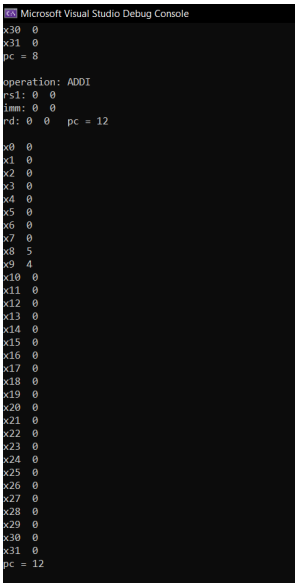
- **RISC-V:**

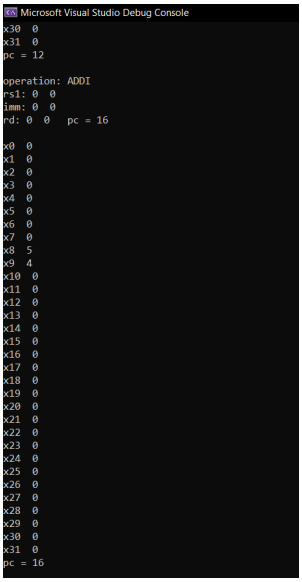
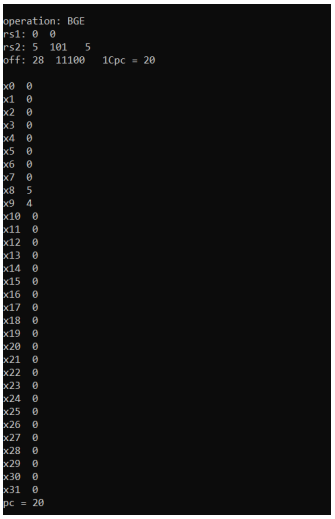
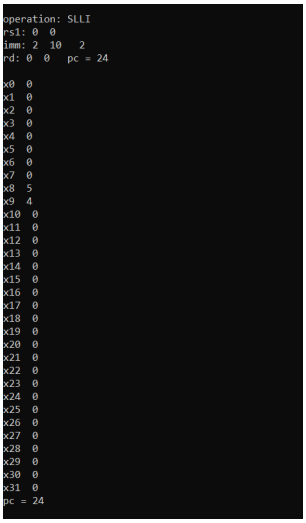
```
lw x9 , 0 ( x9 )
lw x8 , 0 ( x8 )
ADDI x5 , x5 , 0
ADDI x6 , x6 , 0
BGE x6 , x8 , 28
SLLI x18 , x6 , 2
add x18 , x9 , x18
lw x10 , 0 ( x18 )
add x5 , x5 , x18
```

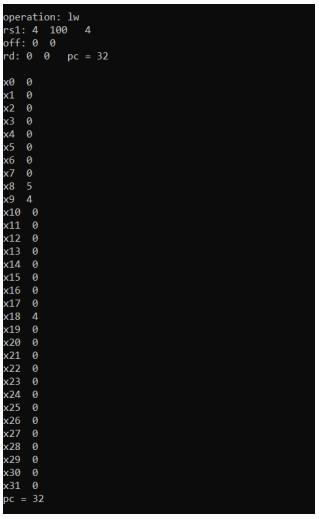
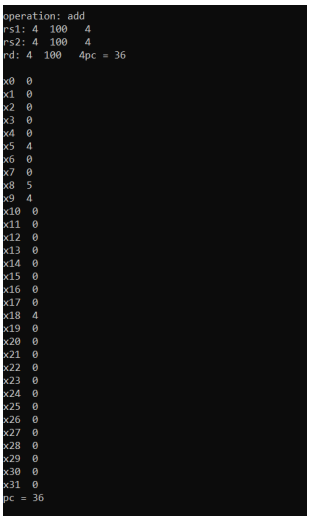
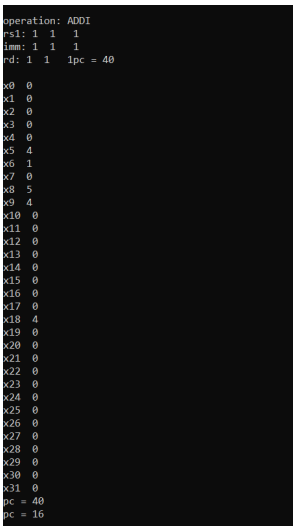
ADDI x6 , x6 , 1
JAL x0 , -24
ADDI x9 , x5 , 0
sw x17 , 0 (x18)
FENCE

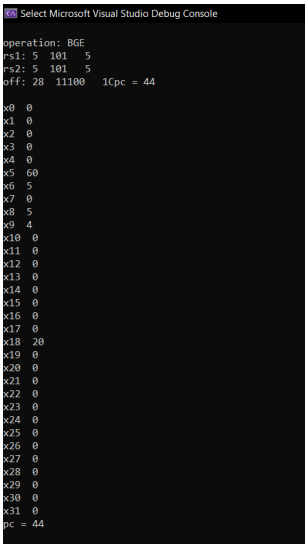
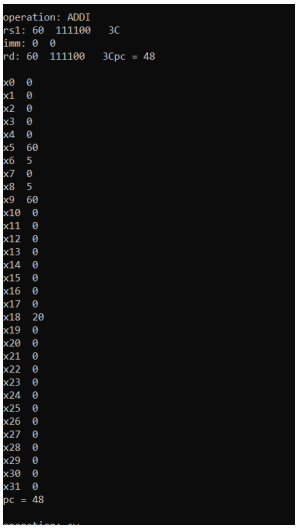
- **Program Simulation screenshots**

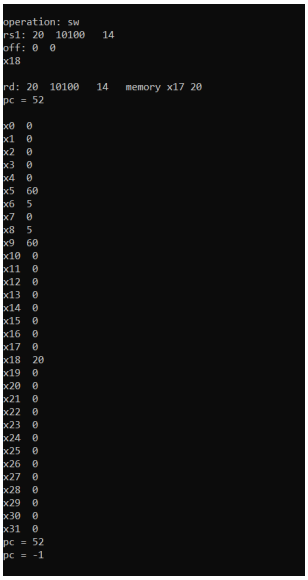
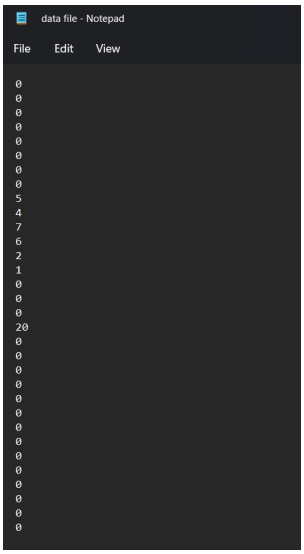
1.	The memory before the execution of the program	
2.	lw x9 , 0 (x9)	

3.	lw x8 , 0 (x8)	
4.	ADDI x5 , x5 , 0	

5.	ADDI x6 , x6 , 0	
6.	BGE x6 , x8 , 28	
7.	SLLI x18 , x6 , 2	

8.	lw x10 , 0 (x18)	
9.	add x5 , x5 , x18	
10.	ADDI x6 , x6 , 1	

11.	JAL x0 , -24	
12.	ADDI x9 , x5 , 0	

13.	sw x17 , 0 (x18)	
14.	FENCE	Terminates the program
	The output file after simulation	

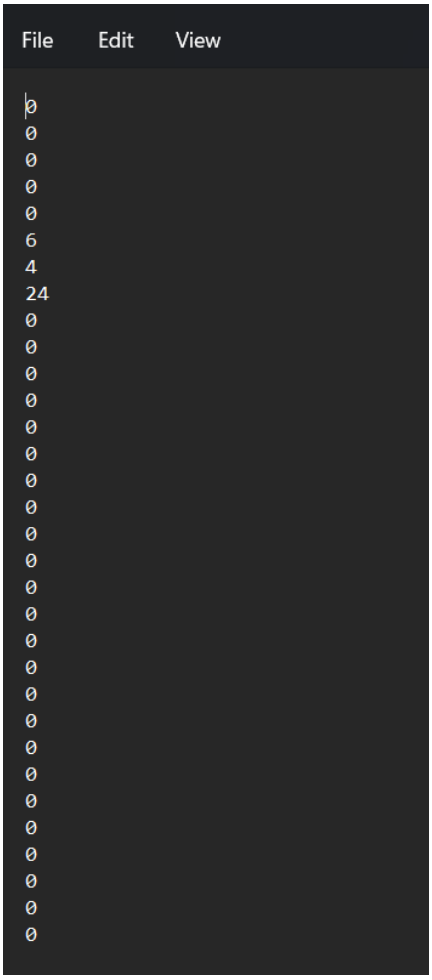
4) **Multiplication of two numbers**
   - **File Name:** multiply
   - **C++:**

```
int multiply (int n, m)
{
        int mult;
        return mult = n * m;
}
```

- **RISC-V:**

```
lw x5 , 0 ( x5 )
lw x6 , 0 ( x6 )
ADDI x9 , x9 , 1
add x7 , x5 , x7
BNE x9 , x6 , -8
sw x7 , 0 ( x7 )
FENCE
```

- **Program simulation screenshots**
  - The output register file after performing the simulation:



5) Logic Gates

- **File Name:** logic gates
- **C++:**

```
int x = 4;
int y = 6;
int z = 5;
int  k = 3;
x = x && y;
z = z || k;
x= x^z;
```

- **RISC-V:**

```
lw x4 , 0 ( x4 )
lw x5 , 0 ( x5 )
lw x7 , 0 ( x7 )
lw x8 , 0 ( x8 )
OR x6 , x4 , x5
AND x9 , x7 , x8
XOR x10 , x6 , x9
sw x6 , 0 ( x6 )
sw x9 , 0 ( x9 )
sw x10 , 0 ( x10 )
FENCE
```

- **Program simulation screenshots**

6) **Powers of 2**
   - **File Name:** powers of 2
   - **C++:**

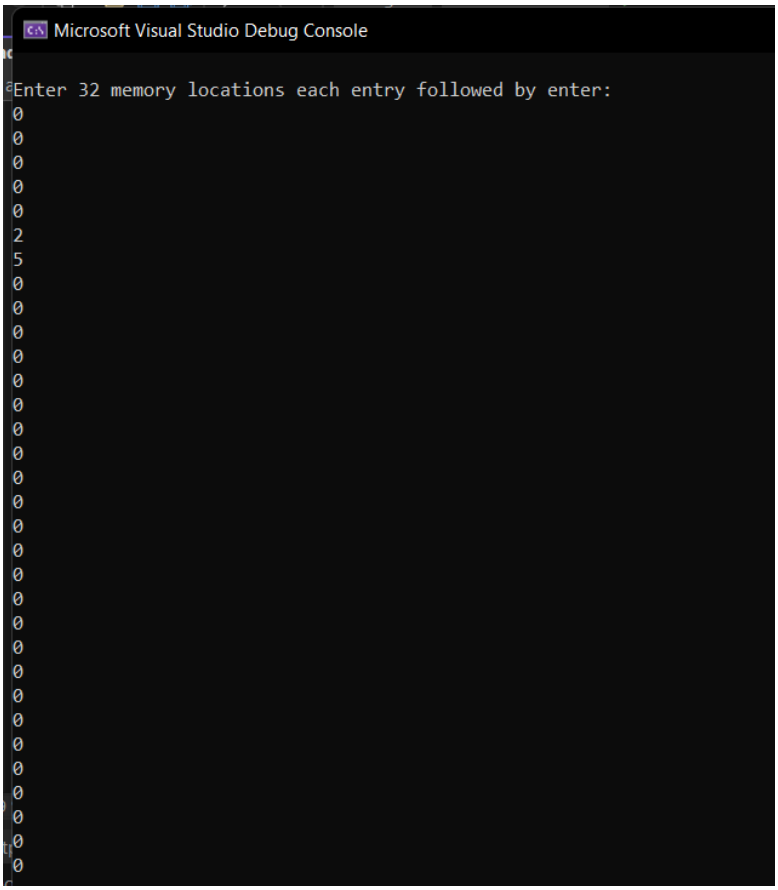
```
int Powers ()
{       int exponent;, base, exponent;
        float base, result = 1;
        cin >> base >> exponent;



        while (exponent != 0) {
                result *= base;
                 --exponent;

        return result;
    }
```

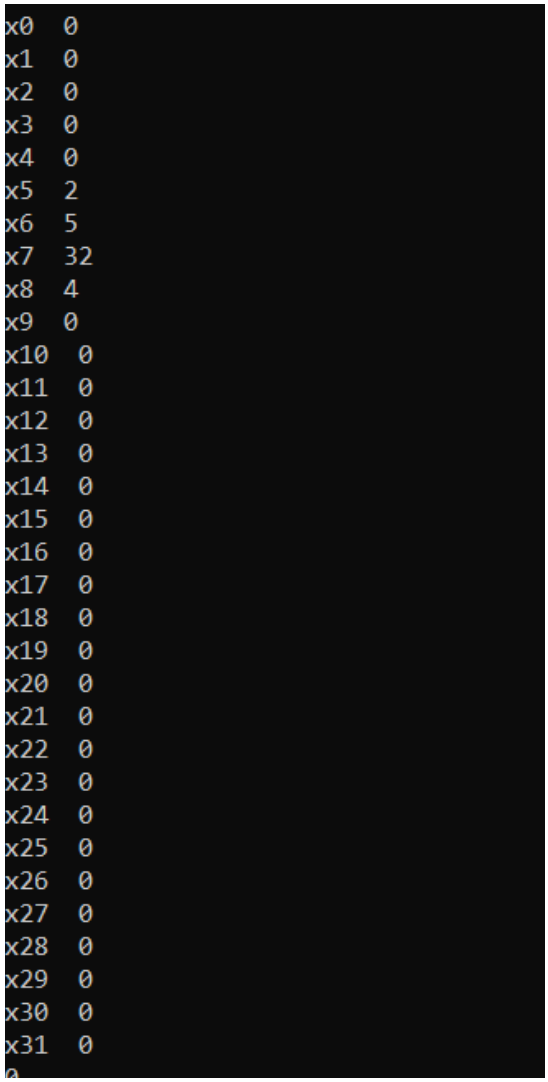
   - **RISC-V:**

```
ADDI x9 , x0 , 0
lw x5 , 0 ( x5 )
lw x6 , 0 ( x6 )
ADDI x8 , x6 , -1
```

SLL x7 , x5 , x8
sw x7 , 0 (x7)
FENCE

- **Program simulation screenshots**




x0 0
x1 0
x2 0
x3 0
x4 0
x5 2
x6 5
x7 32
x8 4
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