

**FACULTY OF ENGINEERING TECHNOLOGY**

**ELECTRIC & COMPUTER ENGINEERING DEPARTMENT**

**ENCS2380-Computer Organization and Microprocessor**

**Project: Single Cycle Processor Design**

**Prepared by:**

|  |  |  |
| --- | --- | --- |
| **Student name** | **ID** | **Sec** |
| Waad Ziadeh | 1220423 | 1 |
| Layan salem | 1221026 | 3 |
| Shahd sbaih | 1220060 | 1 |

1. **Date**: 11 June 2024

**Concept:**

# **Phase One**

In this phase we were asked to design and implement two circuits (Register File circuit, ALU circuit) using a tool called Logisim and with the help of the videos sent by the instructor we were able to achieve these circuits, in order to implement this phase two circuits were needed:

1. **Register File Circuit:**

**Defenision:** It is a CPU component that temporarily stores discreate memory and provides storage

**Implementation:**

We were asked to develop a register file containing seven registers (R1-R7) R0 always reads zero with width of 32-bits, with one write port and two read ports, this circuit was very much essential for storing temporary data during instruction execution.

**Mechanism:**

1. The first decoder determines which register is selected through the selection pin. It contains the values of 000 to 111, indicating the register chosen from R1 through R7.
2. Depending on the selected register, its value goes into the corresponding AND gate.
3. The selected register's value passes through an AND gate along with the register write pin.
4. The output from the AND gate goes through a buffer which increases the impedance and ensures that the signal is maintained when taking the input of the register.
5. Only the selected register is written during a write operation, and the other two multiplexers are used to select the registers for reading.
6. The read values are then available on the output buses.

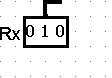
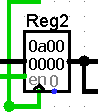
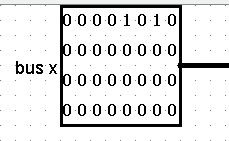
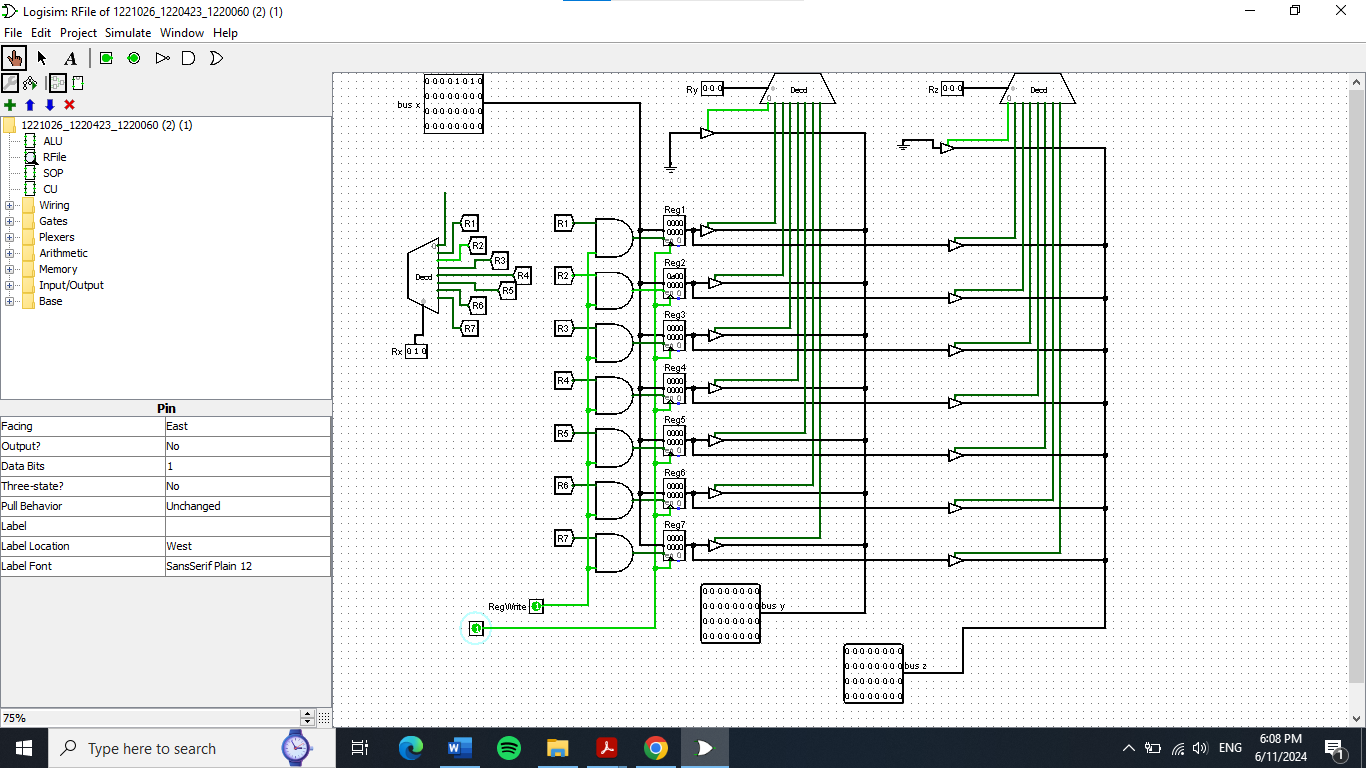
**Components:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Location | Number Of Bits | Purpose of usage | Drawings |
| 3 Decoders | Plexers | 3 selection bits each | Select which register to read from or write to, based on the register address. |  |
| 7 AND gates | Gates | 1 data bit each | Control Write operations to registers, and Ensures data is only written to the selected register when the write enable signal is active. |  |
| 7 Registers | Memory | 32 bit each | Stores data temporarily for processing by the ALU. |  |
| 18 Buffers | Gates | 32 bit each | It helps in matching the impedance of different circuit components, ensuring efficient signal transmission. |  |
| Tunnels | Wiring | 1 bit each | organize the circuit design by logically grouping related signals together when they’re distant in the diagram. |  |
| 3 Busses(pins) | Wiring | 32 bit each | used for transferring data between different components |  |
| Clock | Wiring | 32 bit each | Synchronize the operations of the CPU components. |  |

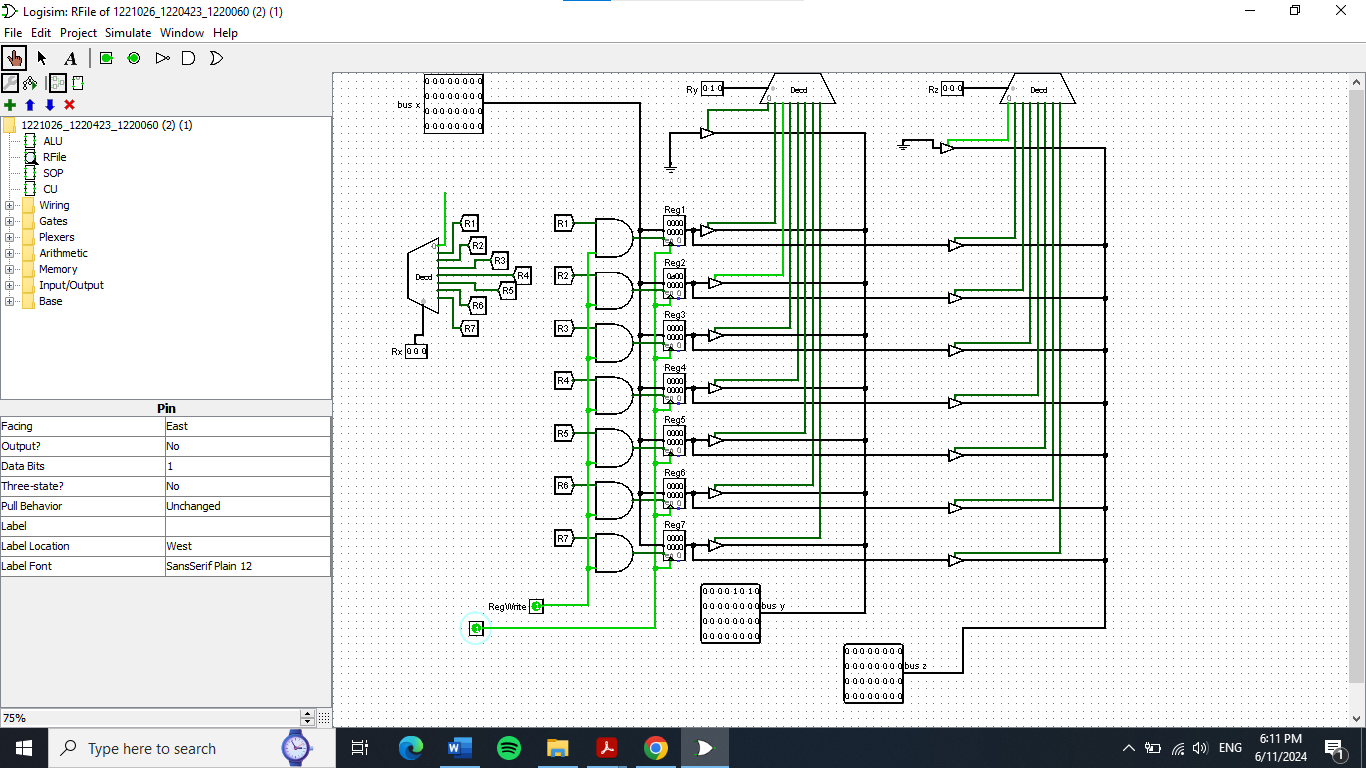
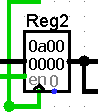
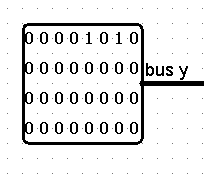
**Simulation and Testing Part:**

These images show the execution of the Register File after Entering different values to it:

1. **Case 1:**

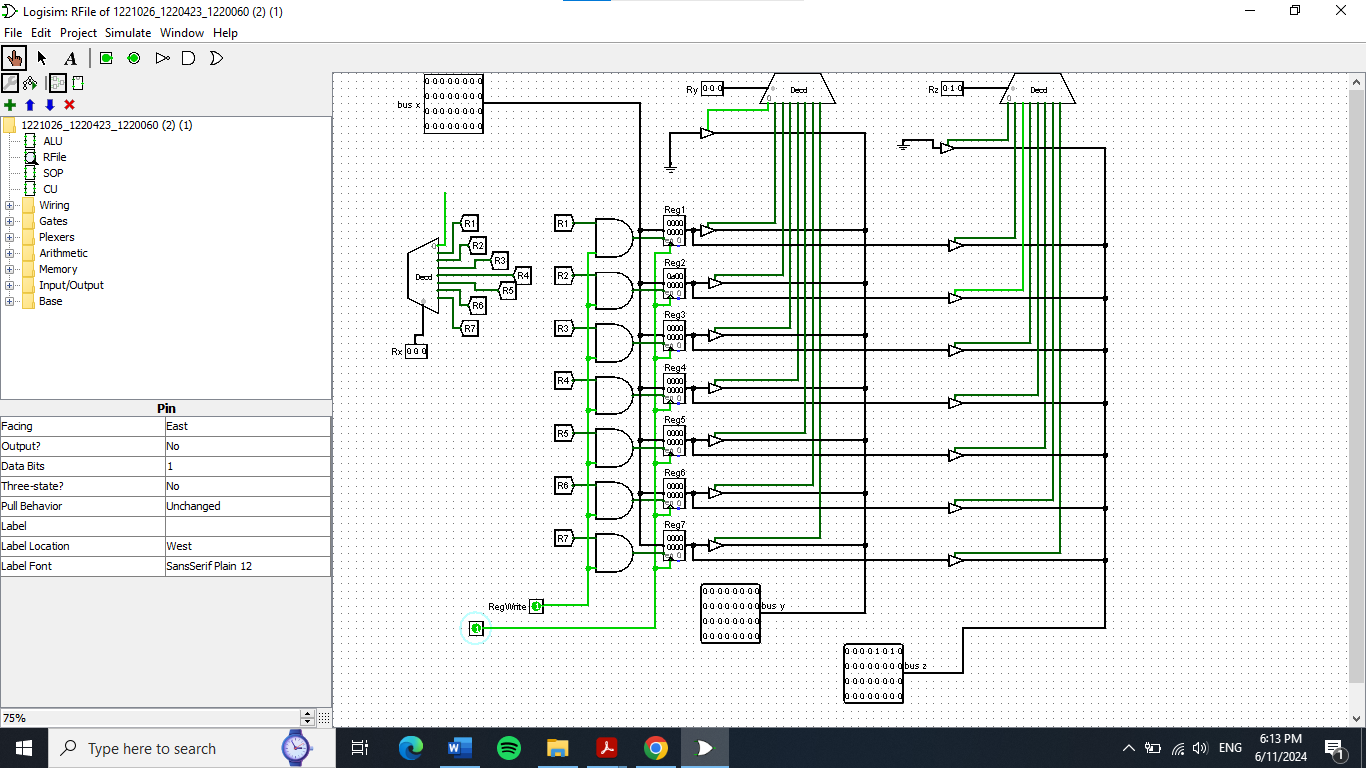
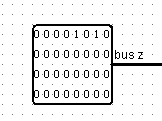
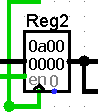
we went with the value “1010“to be written to Reg2, so The Rx input is set to “010”, the data input is set to 1010, and the RegWrite control signal is enabled. This setup ensures that the value 1010 will be stored in Reg2 when the write operation is executed.

1. **Case 2:**

This screenshot demonstrates reading the value from Reg2 to verify the previous write operation. The Ry input is set to “010”, and the output on busY shows “1010”. This confirms that the value “1010” was successfully written to Reg2 in the previous step.

1. **Case 3:**

And here, this screenshot shows when RZ input is set to “010” , and the output on busZ shows “1010”, to also make sure “1010” was successfully written to Reg2 .



1. **Arithmetic and Logic Unit (ALU)** **Circuit:**

**Defenision:** It is the Arithmetic and Logic Unit and it preforms the computer’s data processing function basically almost all the mathematical operation are done in the ALU.

**Implementation:**

We were asked to develop an Arithmetic and Logic Unit (ALU) that is required to do these operations: XOR, AND, OR, CAND, ADD, NADD, SLT, SRA, SRL, SLL, and ROR.

**Mechanism:**

1. The ALU receives two 32-bit input values, labeled as Input 1 and Input 2 , and a 4-bit ALU operation (ALU OP) code determines which operation the ALU will perform from the Operation Gates .
2. A 32-bit wide MUX selects the output from one of the operations based on the ALU OP code. The MUX has multiple inputs corresponding to each operation (XOR, AND, OR, CAND, ADD, NADD, SLT, SRA, SRL, SLL, and ROR) and one output which provides the final result.
3. When the MUX is done with the selection of the operation the operation’s result is output from the MUX as the ALU Result, a 32-bit value representing the outcome of the chosen operation.

**Components:**

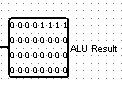
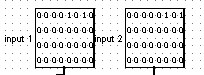
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Location | Number Of Bits | Purpose | Drawings |
| XOR gate | Gates | 32 bits | Performs a bitwise XOR operation between Input 1 and Input 2. |  |
| 2 AND gates | Gates | 32 bit each | Performs a bitwise AND operation between Input 1 and Input 2. |  |
| OR gate | Gates | 32 bits | Performs a bitwise OR operation between Input 1 and Input 2. |  |
| Not gate | Gates | 32 bits | Negates the input. |  |
| Adder | Arithmetic | 32 bits | Adds Input 1 and Input 2. |  |
| Subtractor | Arithmetic | 32 bits | Subtracts Input 2 from Input 1 by adding the two's complement of Input 2 to Input 1. |  |
| 2 comparators | Arithmetic | 32 bit each | Compares Input 1 and Input 2 and outputs a 1-bit result. |  |
| 4 Shifters | Arithmetic | 1. bits | **Shift Right Arithmetic (SRA):** Shifts Input to the right  **Shift Right Logical (SRL):** Shifts Input to the right  **Shift Left Logical (SLL):** Shifts Input 1 to the left  **Rotate Right (ROR):** Rotates the bits of Input to the right. |  |
| MUX | Plexers | 32 bits | Select one of the many inputs and forward it to the output based on the control signals (ALU OP code). |  |
| 3 Pins | Wiring | 32 bit each | Used for transferring data between different components. |  |
| Splitter | Wiring | 32 bits | Signal Distribution as the it is used to take a single input signal and split it into multiple output signals |  |
| 2 Bit Extender | wiring | 1 bit in ,32 bit Out | Bit-width matching or ensuring that the operands involved in an operation have the same number of bits |  |

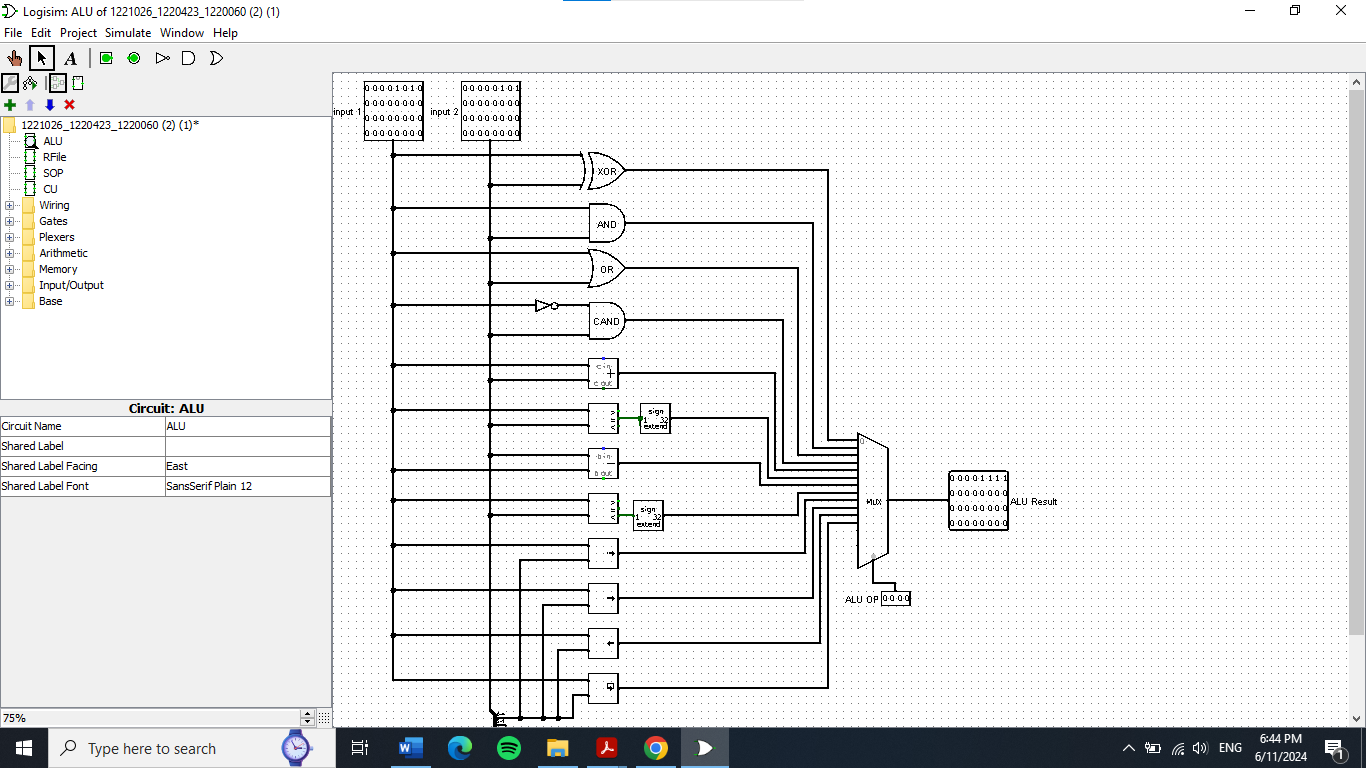
**Simulation and Testing Part:**

These images show the execution of the ALU after Entering different values to it:

1. **Case 1:**

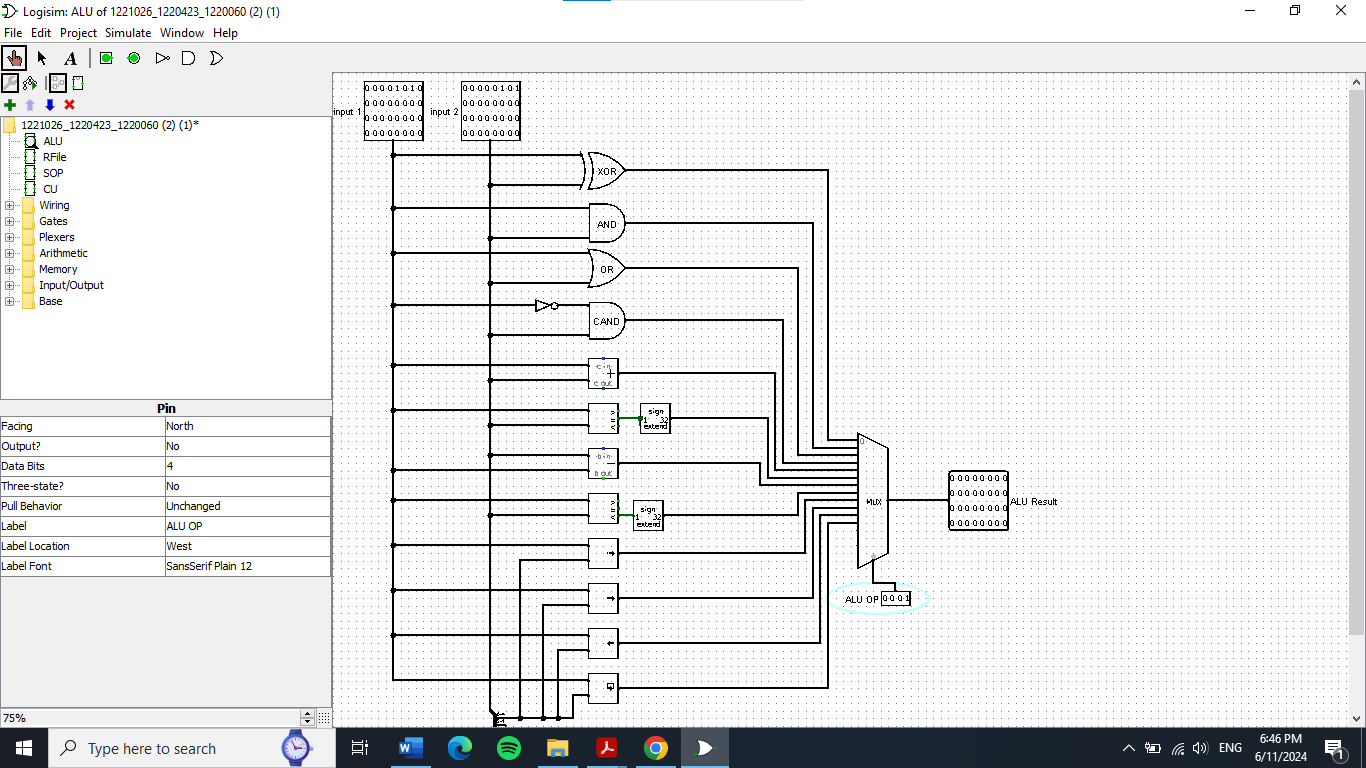
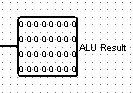
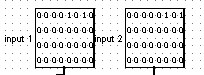
We set the values **input1** **=** **1010** (10 in decimal) and **input2 = 0101** (5 in decimal) to demonstrate the addition operation. The ALU OP control signal is set to 000, which corresponds to the addition operation. This configuration ensures that the ALU will add the two input values, resulting in the output showing the sum of 1010 and 0101 which is

01111 (15 in decimal).



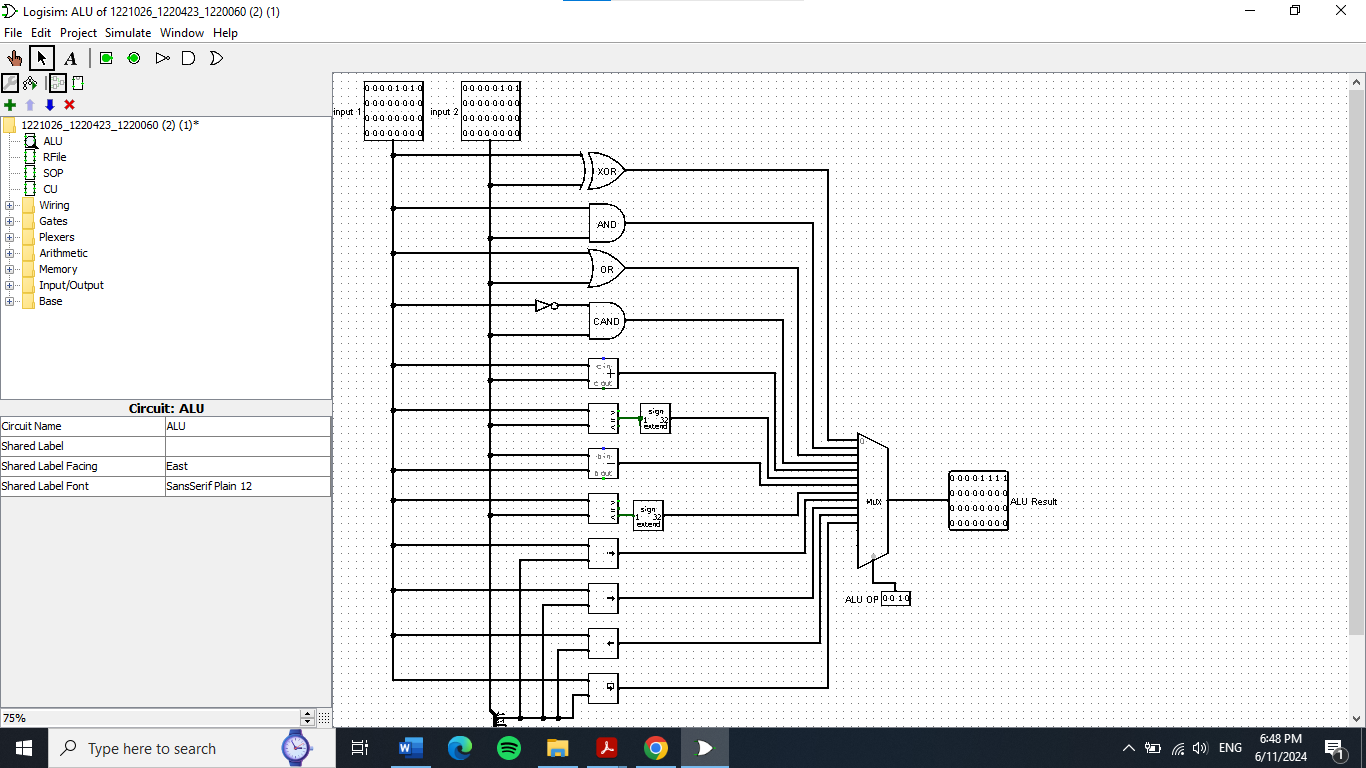
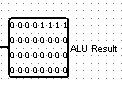
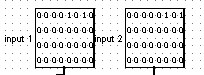
1. **Case 2:**

We set the values **input1** **=** **1010** (10 in decimal) and **input2 = 0101** (5 in decimal) to demonstrate the addition operation. The ALU OP control signal is set to 001, which corresponds to the bitwise AND operation. This configuration ensures that the ALU will perform a bitwise AND on the input values 1010 and 0101, resulting in the output showing the result of this AND operation with output of 0000.



1. **Case 3:**

We set the values **input1** **=** **1010** (10 in decimal) and **input2 = 0101** (5 in decimal), the ALU OP control signal is set to 010 which corresponds to the left shift operation. This configuration ensures that the ALU will shift the value 1010 left by 5 positions, resulting in the output showing the shifted value.



**Teamwork Chart analysis:**

**Details :**

1. **Phase one :**

In this phase , we decided that it would is better if each one of us watched the tutorial videos and tried making it herself , that is because we wanted to get used to the Logisim tool because it is a new tool for us ,and of course if one of us had a problem and didn’t understand something we review her work and explain what she had done wrong or what she could’ve done differently and so on , so all of us Layan Shahd and Waad has implemented the Register file and the ALU .

1. **Phase two :**

This phase needed more work than the previous one so we saw that it would be better to divide the work among the three of us so shahd took the part with implementing the circuit’s structure like connecting the ALU with the register file and PC and the RAM and ROM so basically the structure , while Layan and Waad worked on …………………..

1. **Phase Three:**

For this phase we completed this phase via a zoom meeting and watched the tutorial videos together and implemented the circuit and did the final touches together

1. **The Report:**

We took upon ourselves dividing the project parts so that each one of us reports on a curtain part of the project for example:

**Shahd** was responsible on phase one and reporting the flow of the circuit during the first phase.

**Layan** was responsible on the second phase, and **Waad** worked along with her on this phase.

**Waad** was responsible on the third phase.

And for the other details like the cover page, chart, etc. were made together via another zoom meeting.

# **Phase Two**

**Concept:**

In this phase we were asked to design and implement Data path circuit by using the ALU and Register File circuit blocks from the previous phase, and used separate memory for instruction and data, with pc counter.

**Data Path:**

**Defenision:**  A functional unit that performs data processing operations on input data using the arithmetic logic unit buses, multiplexers, and registers.

**Components:**

Most of component that used in this phase are uses in the previous phase except component like:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **component** | **location** | **Number of input bits** | **Number of output bits** | **Purpose of usage** | **Drawing** |
| PC register | Memory | 20bit | 20bit | Store the address that want to fetch it in the Rom |  |
| Rom memory | Memory | 20bit | 16bit | It used to store instruction data, and you can read from it, and cannot write |  |
| Ram Memory | Memory | 20bit | 32bit | It used to store/write the data and to load/ read the data |  |
| Register file | From previous phase | 32bit | 32bit |  |  |
| ALU circuit block | From previous phase | 32bit | 32bit |  |  |

* **In the Data path circuit has 2 types of memory:**

1. Rom: using for instruction code ‘in our project ‘, and it cannot write on it, it can only read from it the code, the Rom has input Address with size 20bit “bits come from PC reg “, and output Data with size 16 bit “16 bit depends on the length of instruction and ‘as mentioned in the project document ‘“.
2. Ram: using for the data, read and write, make the data interface separate load and store, The Ram has input Address with size 20 bit “it will take the first 20bit from 32bit ALU result, as mentioned in the project document”, and the output Data 32bit, must connect it to clock.

* PC register: as mentioned in the project document the PC must be 20bit, so the size of PC register is 20bit, pc give the address that ant to fetch it in the instruction memory “ROM”.

To Execution the instruction:

1. must take the address that store in PC reg to know which instruction must fetch from the rom.
2. after the address reaches the Rom, the instruction Fetch.
3. Then the instruction decoding stage starts.
4. After that the operand fetch.
5. Now the instruction is execute, compute the result value or state.
6. In the end the result will store, and these steps will repeat.

These steps happened in our program:

# Phase Three