**PROJECT 1: ALU DESIGN**

**Name: Le Dinh Nam**

**Student ID: V202000210**

**Date: 17/10/2021**

1. **Overview**

The project requires every person to build a RISC-V ALU. RISC-V is an instruction set architecture (ISA) that defines the functions a computer can carry out through assembly instructions. An ALU, or arithmetic and logic unit, performs many of the core computations dictated by those instructions.

I will Logisim, a free hardware-design and circuit-simulation tool, to build this circuit. However, I am required to use some of the tools in the program, including:

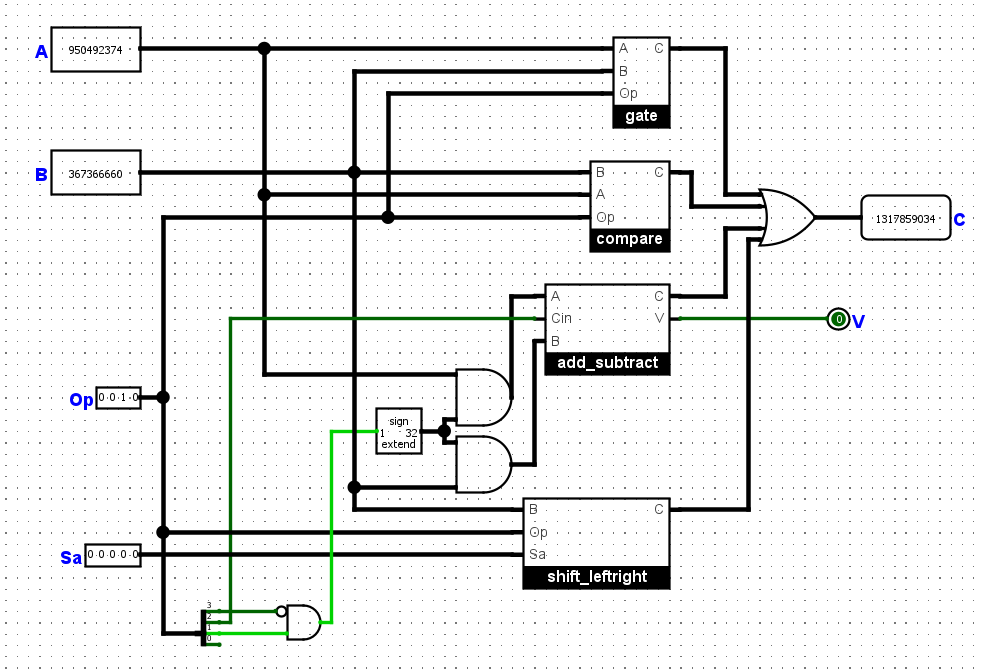
* Anything in the Wiring folder except constants, and anything smaller/less abstract than a gate including but not limited to the resistor, power, ground and transistor elements.
* Anything in the Base folder (wires, text, etc.)
* Anything in the Gates folder except the even parity, odd parity, and controlled buffer elements
* Anything in the Plexers folder

Besides, I need to write three test vectors to test the three circuits, a read me file to introduce the circuit, and this documentation to explain the use of each implementation that I put in the diagram.

After a time thinking, designing, fixing mistakes, watching videos as well as reading online sources, and even asking the lab instructor, I finally finished the circuit which is believed to work correctly.

This is a 32-bit ALU which can realize basic function for one or two 32 bits number. This

ALU has 4 parts, including logic operation, compare operation, add/subtract operation and shift operation.



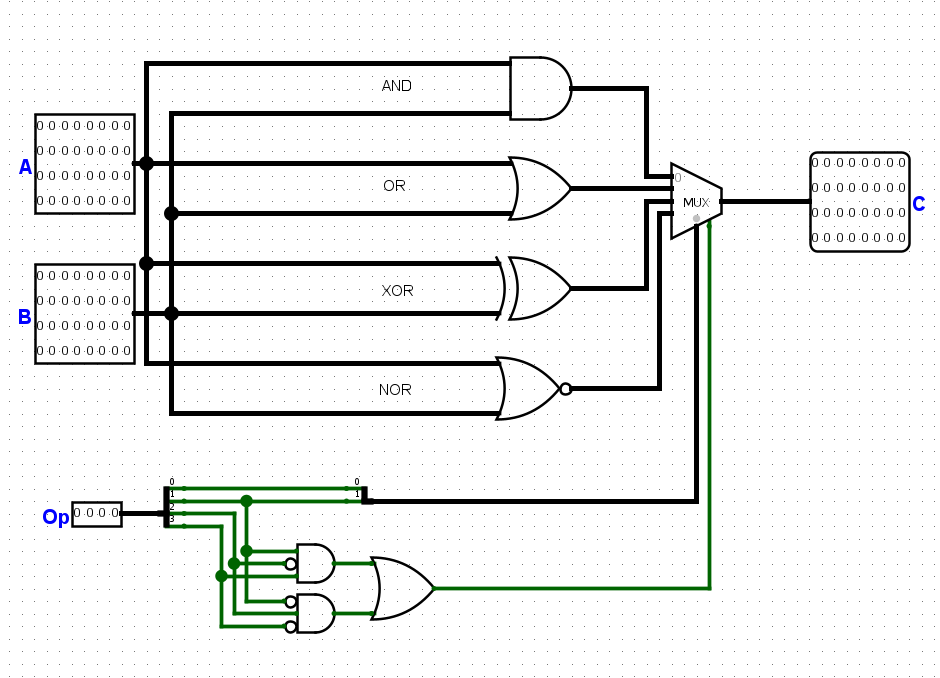
In this diagram, I will explain each gate from above to below, respectively. Each operation has a unique property, and it may be tricky at some points when designing or choosing the logic gates. Therefore, in the next parts, I will explain clearly how I designed these circuits.

1. **Logic Operation**

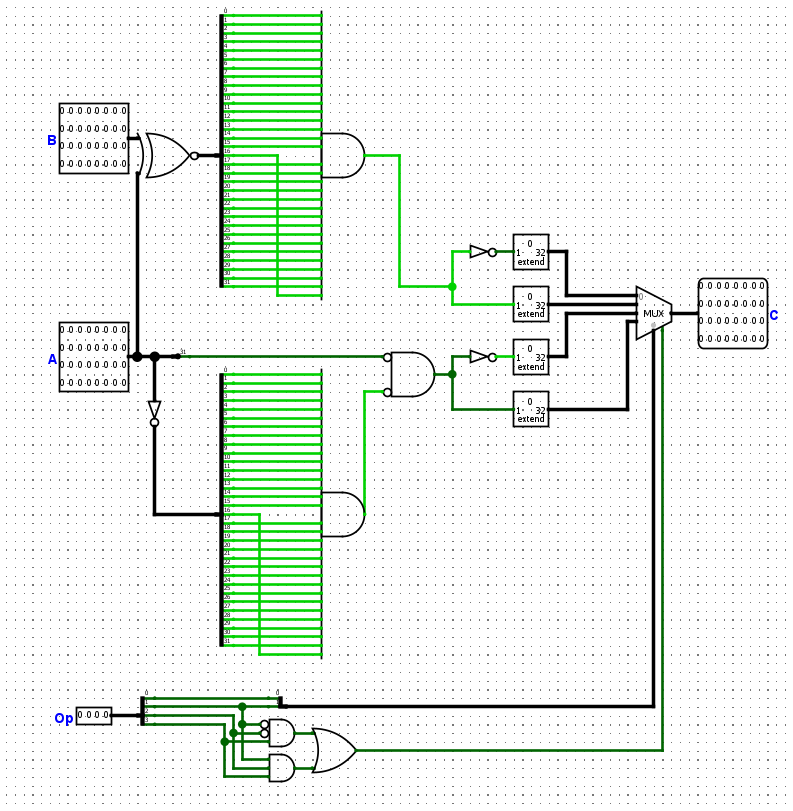
We use four 32-bit gates to realize AND, OR, NOR and XOR functions. The table of Op with corresponding operation is shown below:

|  |  |
| --- | --- |
| OP | Mux |
| 0100 | 0 |
| 0101 | 1 |
| 1010 | 2 |
| 1011 | 3 |

With other kinds of Op, the Mux will be disabled.

****Logic operation is the easiest part in the ALU circuit, as one has to only use 4 logic gates (AND, OR, XOR, NOR) and check whether the Op code is connected with that gate. Comparing with the given Opcode, I designed this circuit that would fit the function of each gate.

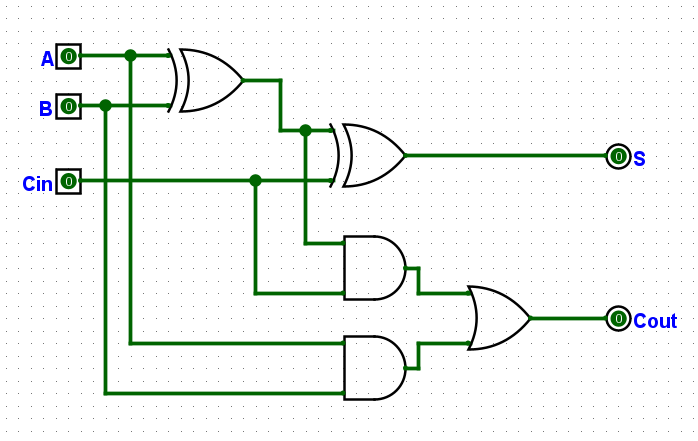
1. **Compare Operation**

****

The compare operation is the most difficult circuit, personally judging. Here is some of my explanation: At the XNOR gate above, I compare A and B. If A is equal to B, A XOR B equals to a 32-bit zero. Then, I will use AND gates to check if A XOR B is zero. If yes, A is equal to B, otherwise A is not equal to B. The idea of the circuit is to divide the inputs into 1-bit so that I can compare and create corresponding outputs.  
Below the circuit, I check whether A is bigger than zero. If A is positive, its 31st bit will be zero, while there exits at least one of all other bits is not zero (in order for A to be larger than 0). If number A satisfies this condition, A is positive, otherwise, A is less or equal to zero. For now, I compare A and B and check if A is positive. In the right of the diagram, in order to get the right answer in four answers for comparation part, I will use a mux to choose the answer correspond to our operation (Op). If Op is not 0xx0, the mux will be disable and only 0 will be output. Otherwise, this mux will choose the output that correspond to our operation

|  |  |
| --- | --- |
| Op | Mux |
| 1000 | 0 |
| 1001 | 1 |
| 1110 | 2 |
| 1111 | 3 |

1. **Add/Subtract Operation**
2. **Adder\_1bit**

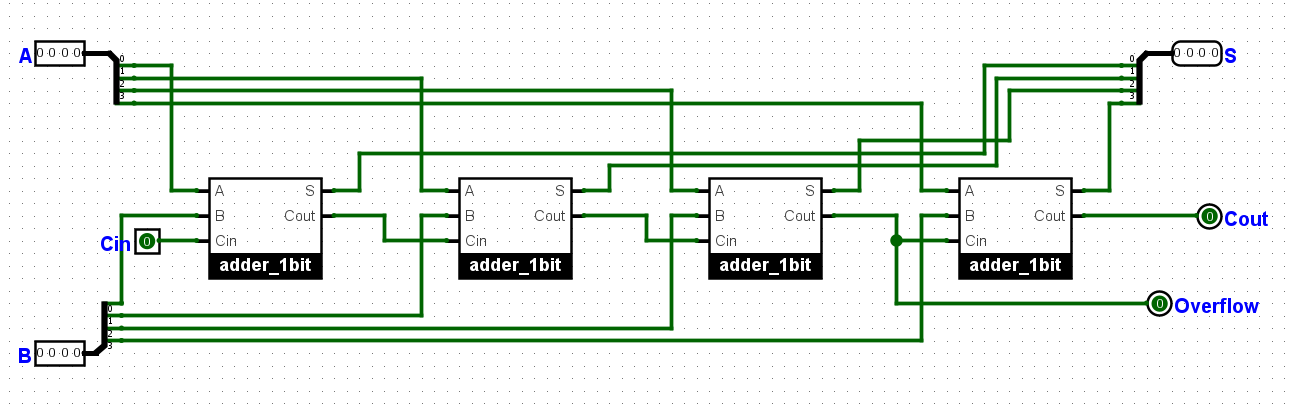
****Follow the instruction of the instruction in the lab session, I understand the concept behind this diagram and further develop other circuits.

This is the truth table that relates to the diagram above.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Cin | S | Cout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

1. **Adder\_4bit and adder\_16bit**

Thanks to the instruction, I learned how to implement the 4-bit adder and did the 16-bit adder myself. In addition, I noticed that it the circuit needs to have the overflow, and the overflow button is only affected by the last bit, so I used the XOR gate and connected with the Cin/Cout of the last bit.



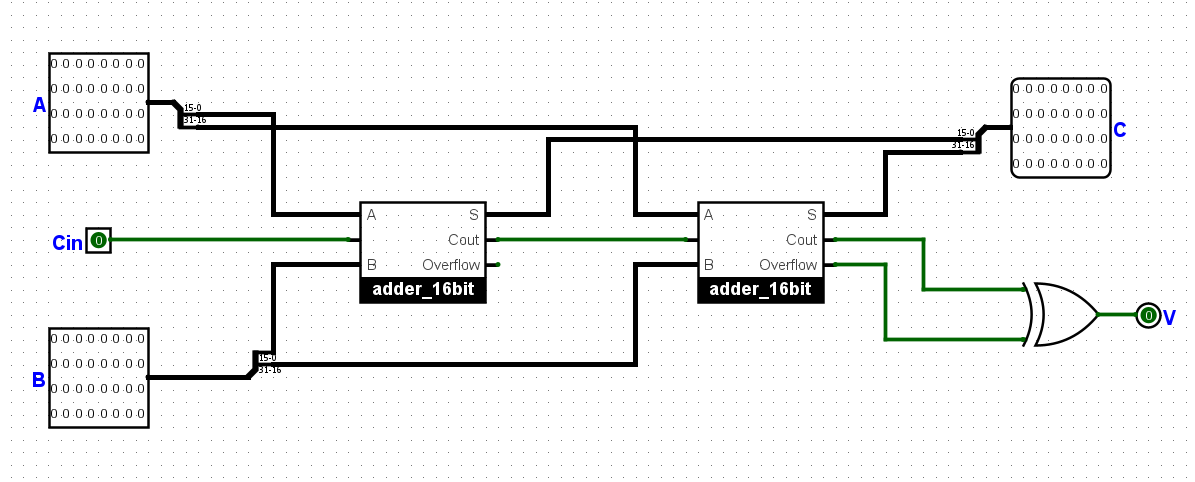
This is the 4-bit adder

Ảnh có chứa văn bản, ảnh chụp màn hình, thiết bị

Mô tả được tạo tự động

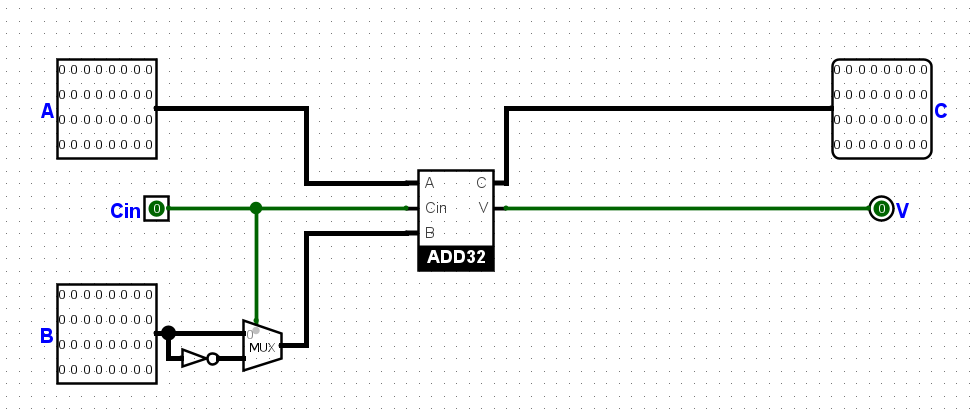
This is the 16-bit adder. Here, we can see that the overflow is attached with the last bit of the sum of two numbers. Using this overflow, we can apply it in the 32-bit adder circuit, and even some larger circuits.

1. **Adder\_32bit**

****

Combining two 16-bit adders, I will have the full 32 bit adder which will be used for addition only. Finally, to add and subtract, I created another file name ADD32.

1. **Add32 (add and subtract)**



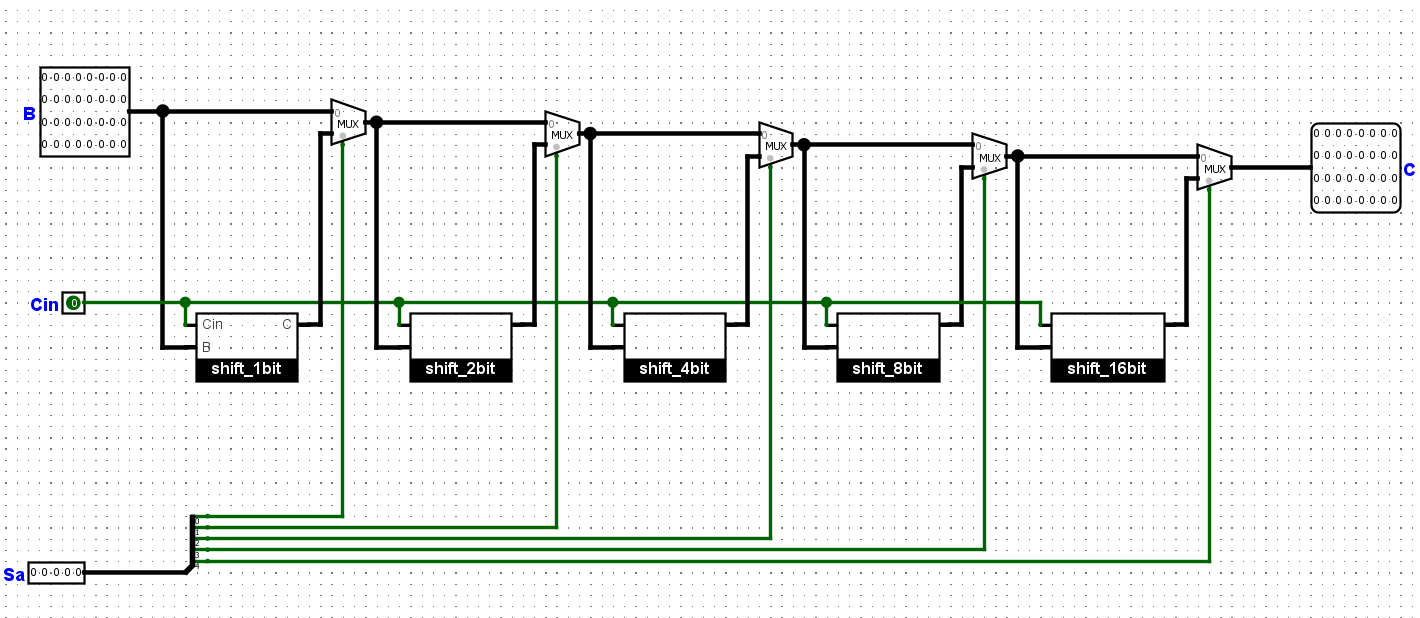
In this diagram, I put the Mux into B to check whether the sign of B is + or -. Therefore, I care about the bit Cin, which will determine this sign. If Cin is 1, it will trigger the Mux, and therefore the sign of B will be changed.

Last but not least, I need to consider this circuit with the given Op code (when the Op is 011x or 001x). Therefore, the last bit in the Op code plays no roles in determining the plus or minus sign of the number B, and the Op table is shown below:

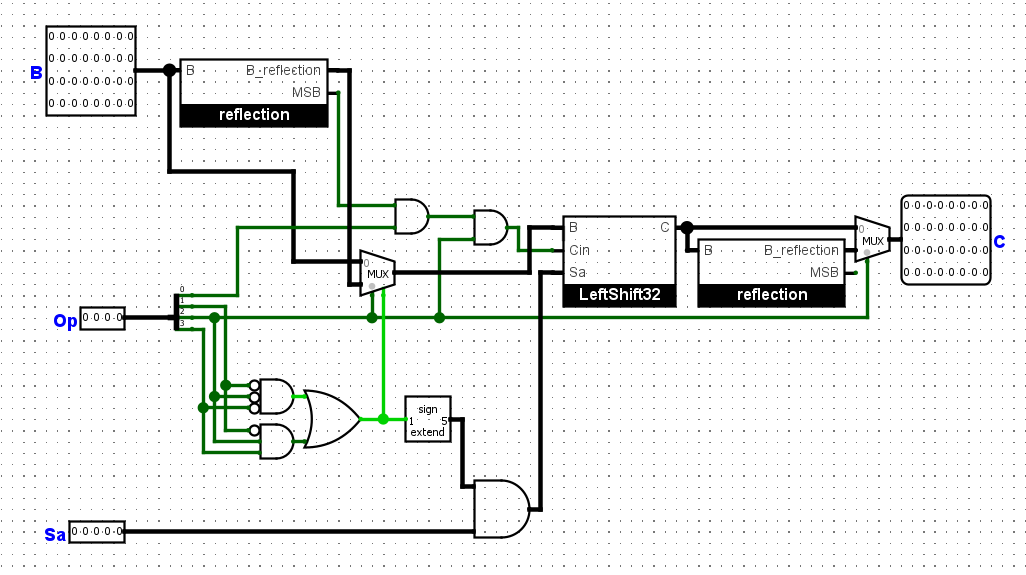
|  |  |
| --- | --- |
| Op | Operation |
| 001x | + |
| 011x | - |

Otherwise, we will not activate this operation, and need to check other functions in the circuit.

1. **Shift Operation**
2. **Leftshift32**

****In order to make 32-bit shift left circuit, I needed to design the left shift from 1 to 16 bits numbers. Then, combing with the Mux gates, the implementation of the circuit is shown as above.

1. **Shift 32**

****

In this circuit, I perform both left and right shift. The idea behind this circuit is the function reflection which would “reflect” the bit of a number. The implementation of the reflection circuit is shown below. Next, after reflecting, I will do the shift left, and finally reflect one more time to the first number. However, I need to care about the most significant bit of the reflection circuit, as I may change that bit when we do the shift left, and I also need to distinguish SRL as well as SRA.

Ảnh có chứa bàn

Mô tả được tạo tự động

The Op code for each shift is in the table. If it is different, the circuit will be disable.

|  |  |
| --- | --- |
| Op | Operation |
| 000x | Shift left logical |
| 1100 | Shift right logical |
| 1101 | Shift right Arithmetic |

1. **Acknowledgement**

I am very grateful to receive comments and feedbacks from our lab instructor Vu as well as my beloved Professor Nam. Thanks to the instructions of the labs and project 1, I have learned and practiced a lot with

1. **References**

In order to understand more how to build the gates, I checked these sources below:  
<https://www.youtube.com/watch?v=lvYCchzQTyE>

<https://www.youtube.com/watch?v=mAl809U9aB4>

<https://www.researchgate.net/publication/341055790_Design_and_Development_of_a_32-bit_ALU>