COMP2020 Project 1: ALU, Design Document

Nguyen Dinh Cuong, V202000025

1 Overview

This is a 32-bit ALU with the following features: equal, not equal, and, or, nor, xor, less than or equal to, greater than, add, subtract, right shift arithmetic, and left/right shift logical. The ALU is part of a RISC V processor, which is the latter project that we have to do.

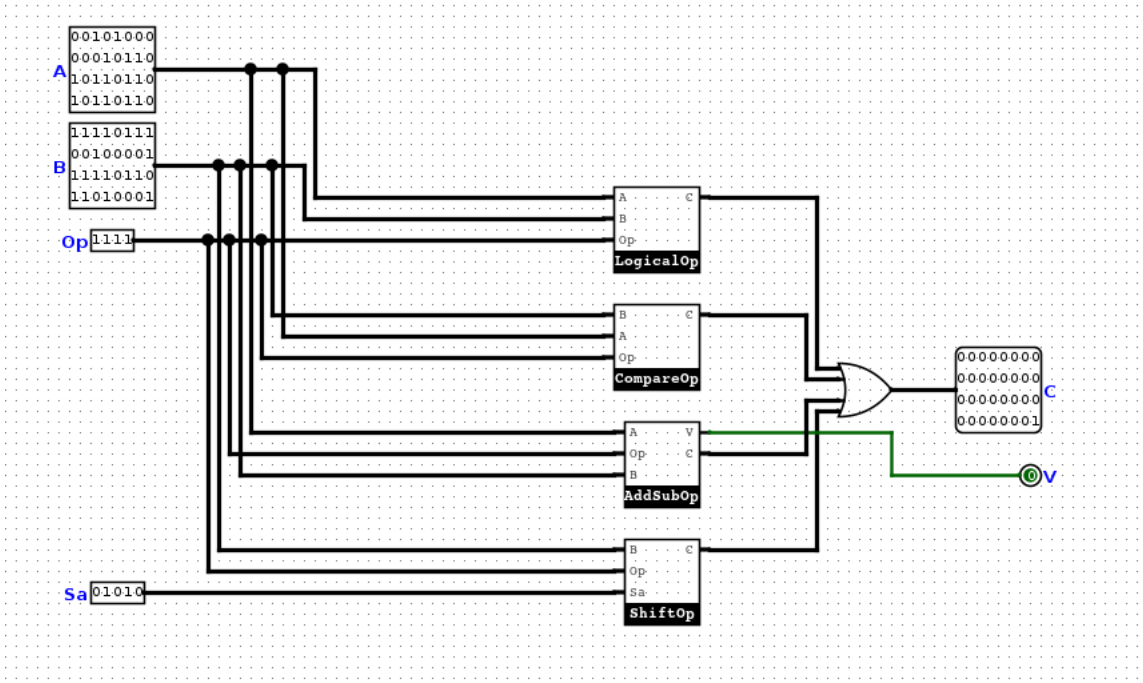

Figure 1: Final Product - The 32-bit ALU

2 Logical Operators

2.1 Implementation Details

AND, OR, XOR, and NOR are the four logical operators that the ALU must execute. Since NOR could be replaced using the NOT of OR, we can use three gates only. Op bits 0 and 1 represent the

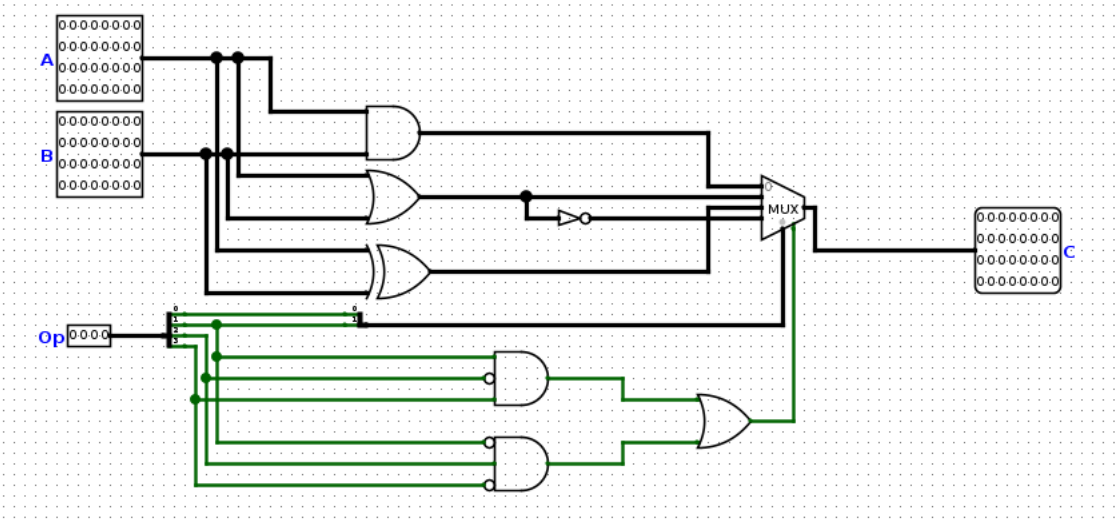

Figure 2: Logical Operators Implementation - LogicOp

control for these operations. The two Op bits are combined into a single wire and fed into a 2x4 multiplexer as the input signal.

Op Bits Handling Truth Table - Logical Operators		
Op Bit 0	Op Bit 1	Output
0	0	A AND B
0	1	A XOR B
1	0	A OR B
1	1	A NOR B

We utilize the minterms and sum of products approach that we learned in Lecture 2 to create the detection circuit for Op bits handling and logic implementation to identify that the user wishes to use logical operators. Other detection circuits will be built using the same method, with the exception of the Adder/Subtractor, which will be built using a different technique.

2.2 Evaluation

The most significant design decision was to divide the wire after OR rather than implementing a separate NOR gate. While adding another NOR gate to the circuit may improve its clarity, employing a NOT gate is more efficient.

3 Value Comparisons

3.1 Implementation Details

This circuit is divided into two sections, the upper half comparing A and B. A XOR B equals a 32-bit zero if A is equal to B, thus, A XNOR B equals a 32-bit one if A equals B. Check if A XOR B equals zero using AND gates. If the answer is yes, A equals B; if the answer is no, A does not equal B. We verify that A is greater than zero in the lower section. If A is positive, the 31st bit will be zero, and at least one of the remaining bits will not be zero. A is positive if it meets this criterion; else, it's less than or equal to zero.

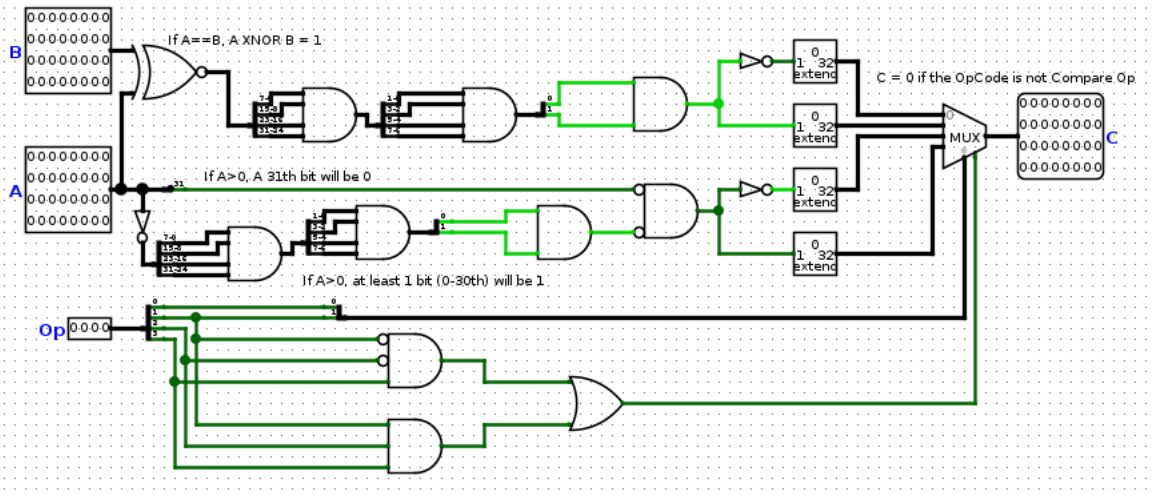

Figure 3: Comparing Operation Implementation - CompareOp

The control for these operations is likewise represented by Op bits 0 and 1. As the input signal, the two Op bits are merged into a single wire and sent through a 2x4 multiplexer.

Op Bits Handling Truth Table - Comparing Operation		
Op Bit 0	Op Bit 1	Output
0	0	(A != B) ?
0	1	(A == B) ?
1	0	(A $\leq$ 0) ?
1	1	(A $>$ 0) ?

3.2 Evaluation

The function of this circuit gets apparent and the implementation becomes simpler by splitting it into two smaller components. Using those bit extenders for calculating and gates for Op bits handling are effectively-sufficient methods in this case.

4 Adder/Subtractor

The sum of two 1-bit inputs with a carry bit is supported by a one-bit complete adder. This circuit is required for more complex full adders to be built.

4.1 Implementation Details

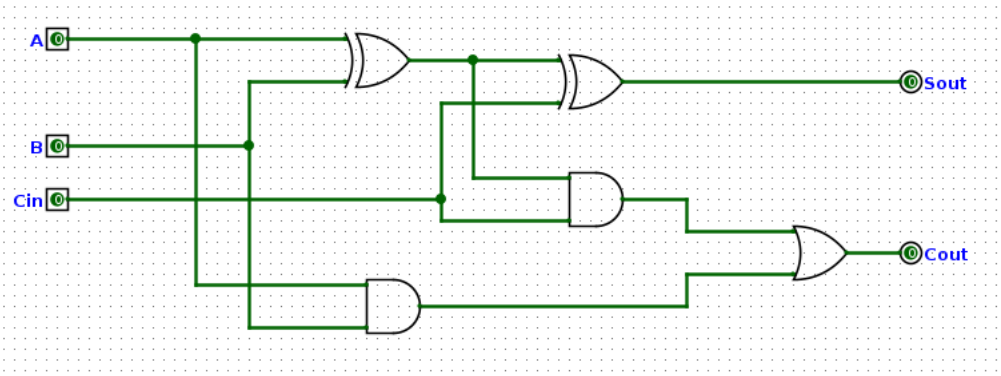

Figure 4: One-Bit Complete Adder Implementation

A and B are the 1-bit inputs in the figure above, whereas Cin is the carry-in bit. The output bit is S, and the carry-out bit is Cout. It is the most optimized circuit for the table below:

Truth Table of One-Bit Adder Circuit				
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

All additional higher-order adders are built using 1-bit adder. As this 32-bit adder requires an output V to indicate overflow, a circuit which is Add2V was created as a 2-bit adder that outputs V instead than Cout. This process is repeated for all adders up to and including 16-bit adders.

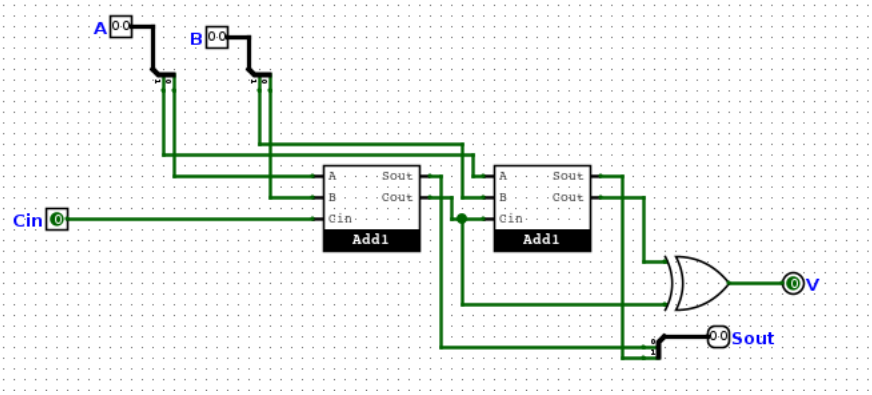

Figure 5: Two-Bit Adder Implementation With Overflow Detection - Add2V

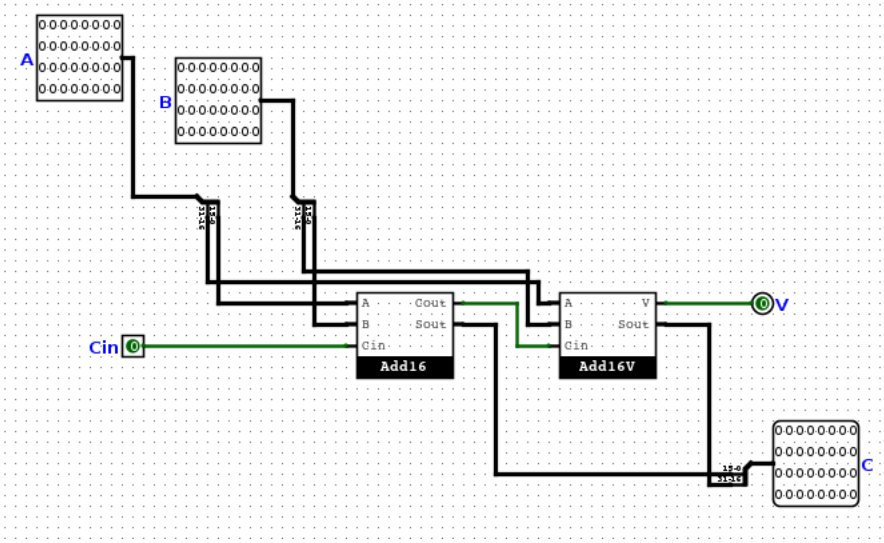

Figure 6: Thirty-Two-Bit Adder Implementation With Overflow Detection - Add32

The control for the Add/Sub operation comes from Bit 2 of the Op input. A-B is evaluated if Op Bit 2 is 1, and A+B if Op Bit 2 is 0. In case we need to use B is a two's complement number, the negative B is obtained by taking the inverse and adding one in subtraction. This may be done in circuit form using a NOT gate on B and a multiplexer, as well as 1 in Cin. I, on the other hand, have chosen a different approach:

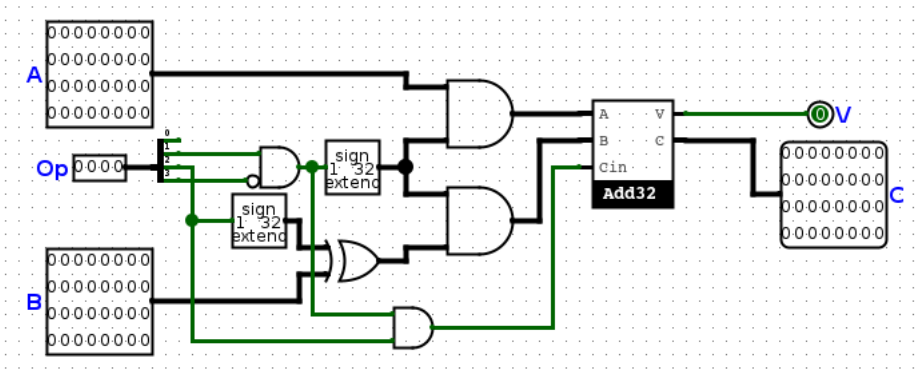

Figure 7: Adder/Subtractor Implementation - AddSubOp

Instead of using a multiplexer with inverses as inputs, which would decrease the efficiency, we just need to use a bit extender and a XOR gate. They are logically equivalent since their truth tables are the same. For the sake of Op bits handling, things are also a bit different. Because the Op code is unique for this operation (011x for subtracting and 001x for adding), we don't need the use of the minterms and sum of products technique. All we need is an AND gate with one negated input and a bit extender.

4.2 Evaluation

For both the resultant bit and the carry-out bit, the circuit implements the simplest and most effective implementation. However,the fact that each sub-circuit for the adder required two distinct types of circuits would have been a weakness in this design. While this has no bearing on performance, a more simple architecture for circuits in general would be preferable. This circuit also employs a ripple carry adder, which is too inefficient for an actual microprocessor, as stated in the project description. The use of XOR gates and bit extenders instead of multiplexers for Op bits handling may be the most prominent feature of this design.

5 Shifter

5.1 Implementation Details

5.2 Evaluation

6 Output C

Any portion of this ALU will only produce a non-zero result if and only if the operation is corresponding to this component, as shown in the figure below. As a consequence, the only non-zero result is output into C through an OR gate.

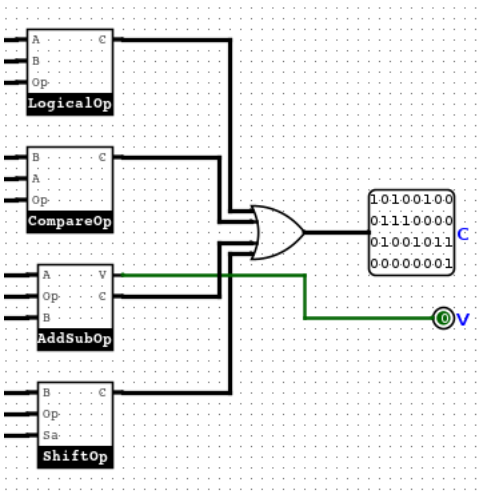

Figure 8: Comparing Operation Implementation