Design Document: Project 1

Ayesha Gagguturi

Introduction

An arithmetic and logic unit (ALU) is a combinational circuit that performs various Boolean and arithmetic operations on a pair of n-bit operands. In this homework, you will create a 32-bit ALU that can perform the operations of most MIPS arithmetic and logical instructions.

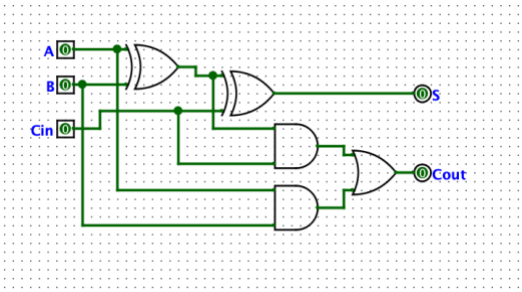
Overview

This project had the purpose of creating a 32-bit Arithmetic Logic Unit (ALU). This is a subset of the RISC32 architecture in Logisim, a software logic simulator. The goal after this is to have another part for the 32-bit pipelined RISC CPU we will make in the future. The ALU in this circuit consists of 12 operands-- OR, AND, XOR, NOR, EQ, LE, GT, NE, ADD, SUB, Shift left logical, shift right logical, and shift right arithmetic. The high level idea is to create a circuit that uses an opcode as a selector in a mux to determine which operation to compute and feed as an output and to determine whether overflow has occurred..

Component Subcomponent Design Documentation

1-bit Adder

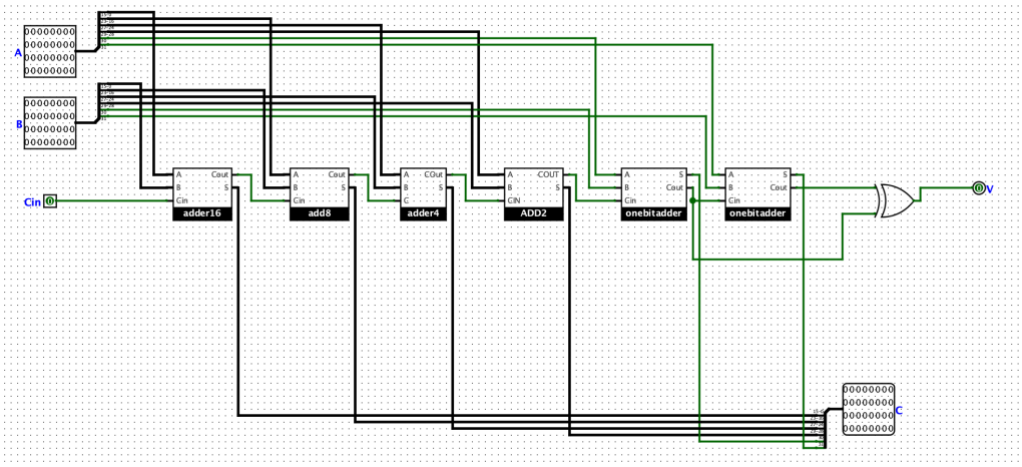
The 1-bit adder takes in 3 1-bit inputs- A, B, and C_{in}, and 2 1-bit outputs- S and C_{out}. The basic functionality of this circuit is to take the inputs and find the summation which is fed into S and then any overflow is given to C. The idea behind the adders is that if A and B are 1 then the S is 1 and the C_{out} is 1, and vice versa for if A and B are 0.



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

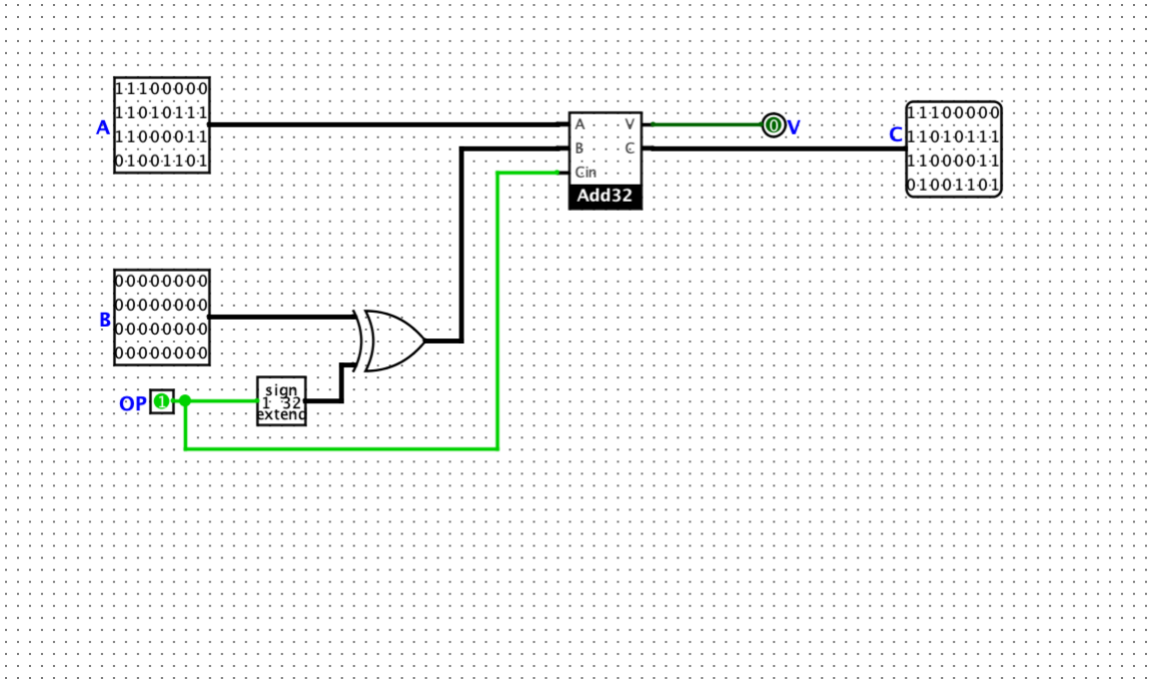
32-bit Adder

The implementation of the 32-bit signed adder consists of a 16-bit adder, 8-bit adder, 4-bit adder, and 2 1-bit adder. This is to help with the logic of the overflow that needs to look at the 2 bits on the right most side of the binary number. The V is only 1 when exactly one of the C_{out} of the last two MSB are 1. So, when V turns 1 when there is an overflow.



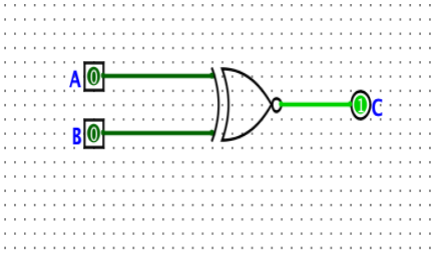
Add and Subtract Subcomponent

In order for the ALU to add and subtract I created a subcomponent that based on a given input from the opcode would add if it equals 0 or subtract if 1. The addition is done by simply using the 32-bit adder made before and connecting the inputs and outputs. The subtraction is done by knowing the fact that $A - B = A + (-B) = A + (\bar{B} + 1)$. So, when the opcode is one the bits of the B are noted and then the carry in takes in the value of the opcode, which is 1, and then feeds that in to represent the addition.



1-bit Equal

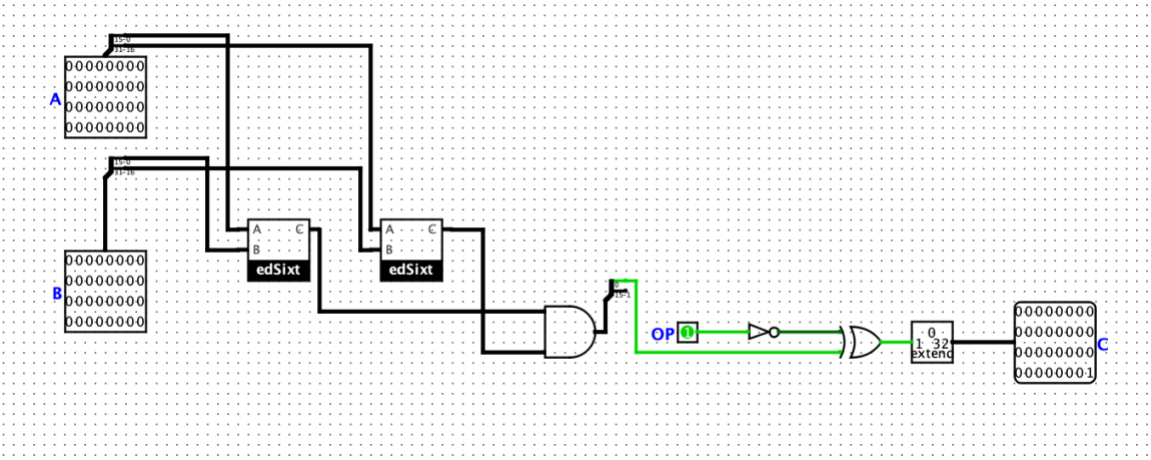
The 1-bit Equality checker is done by passing the inputs of A and B into a XNOR gate to check whether they are equal.



A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

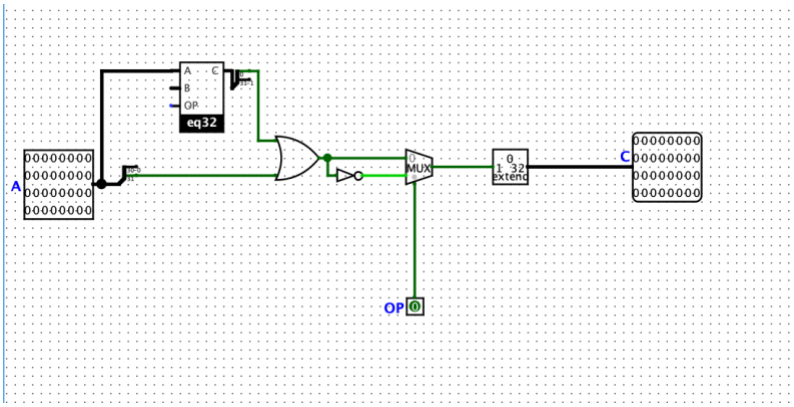
32-bit Equal

The 32-bit equality checker uses two 16-bit equal circuits which both individually use two 8-bit circuits and two 4-bit circuits with 2-bit circuits. This is necessary because we want to check whether the entire 32-bit inputs and versus just checking each individual bits. Another change we had was that we originally had a mux that took in the input and its inverse, so we replaced it with a X-OR



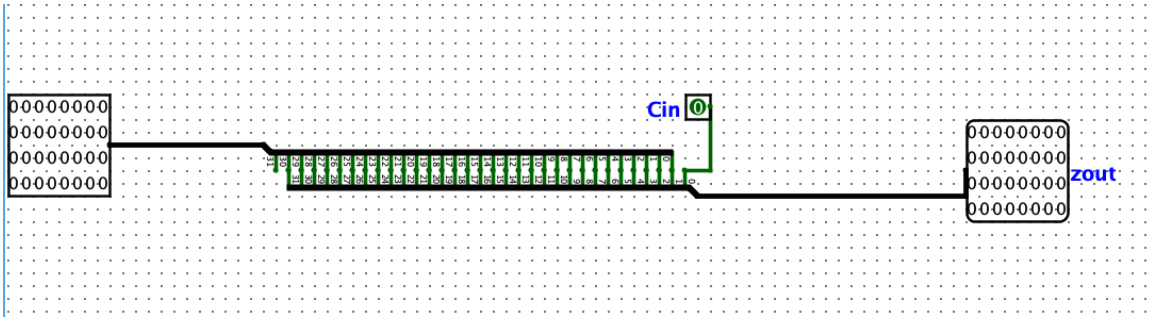
Equal and Not Equal Component

This uses a 32-bit equal circuit to accomplish the goal of checking whether the inputs are equal or not equal based on the opcode. If the opcode is 0 then the circuit will go with the equal operation and if it is 1 then it will return the opposite of equals. This result is then sign extended with 0 to make it a 32-bit.



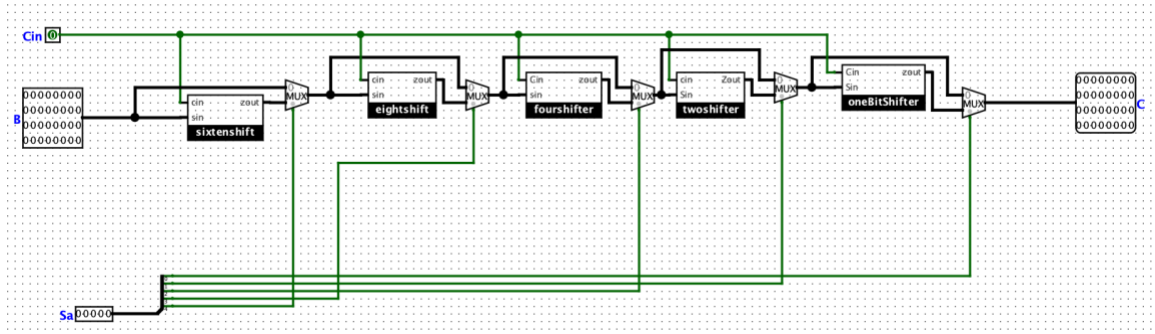
32-bit 1-bit Left Shifter

This sub circuit has purpose of left shifting a 32-bit input by 1-bit and filling in the rest of the unknown bits with the value of $C_{in.}$



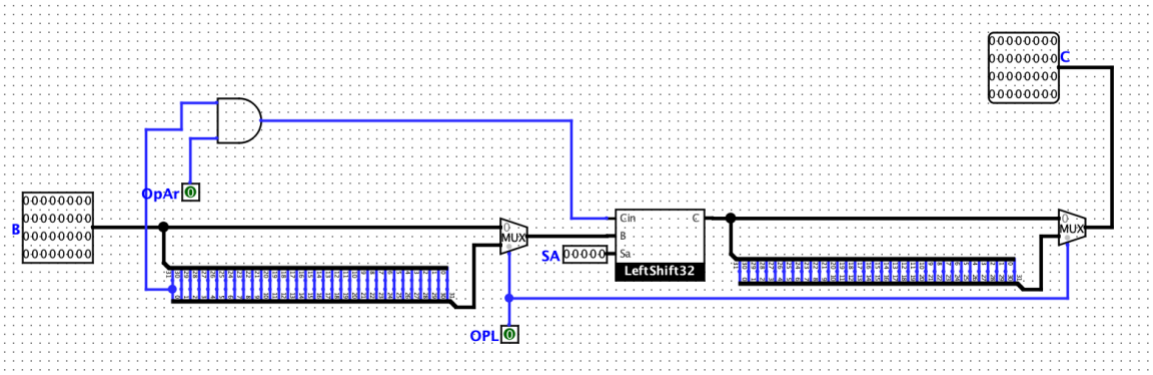
32-bit Left Shifter with a shift amount of 32

This circuit has a 16-bit shifter, 8-bit shifter, 4-bit shifter, 2-bit shifter, and a 1-bit shifter. By enabling different combinations of stages, the circuit can shift any desired amount. The purpose of this to shift the circuit by the amount inputted into Sa.



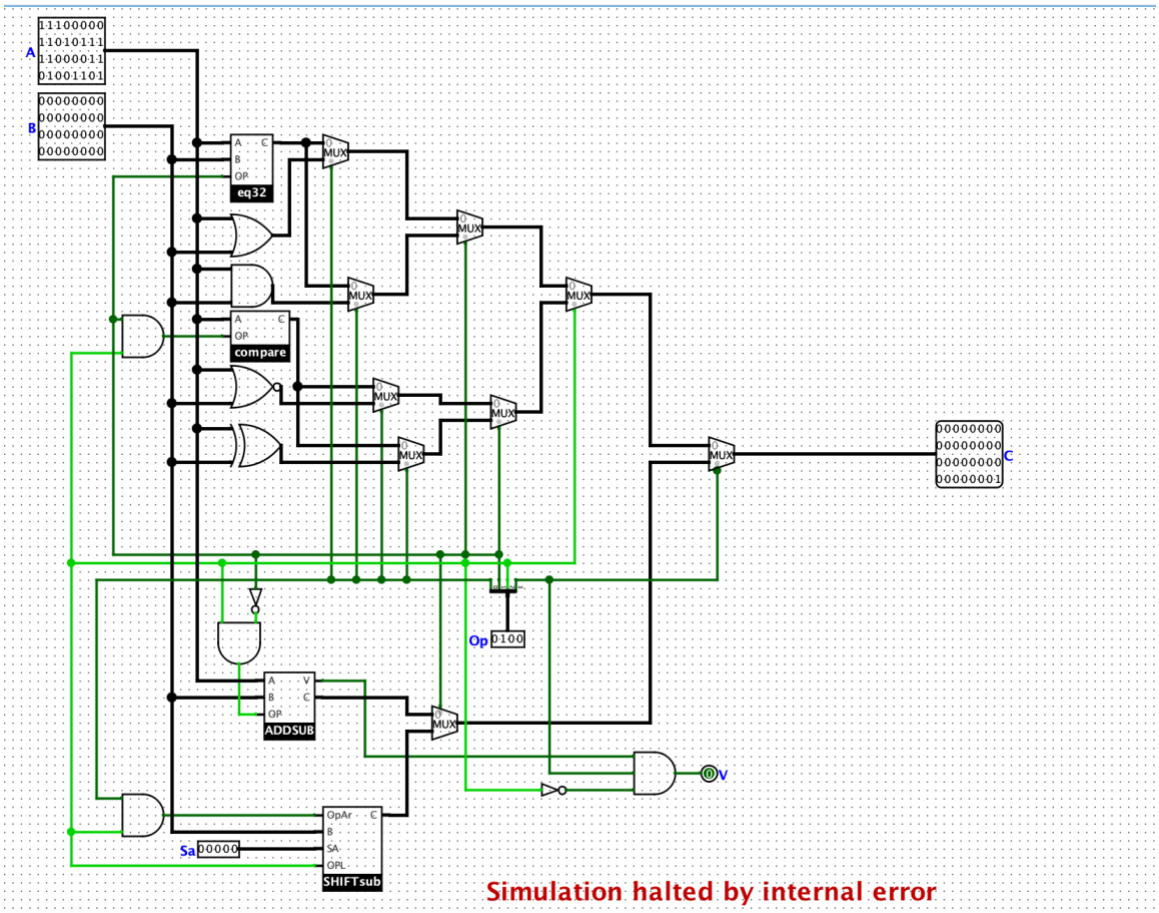
Shifter Subcomponent with Right Shift Logical, Right Shift Arithmetic, and Left Shift Logical

This subcomponent includes Right Shift Logical, Right Shift Arithmetic, and Left Shift Logical. The OPL decides whether it should pick the left or right shift. The OpAr decides which value C_{in} should be. The And gate is added to ensure that when both the MSB of B and the OpAr are 1 then the C_{in} is also 1.



Arithmetic Logic Unit

For the overall, ALU, the logic is done with muxes that choose the operation based on the opcode by going left to right of the opcode. So, the mux on the rightmost side looks at the MSB and then it goes on after the next significant bit. This means we can group our operations so that the mux decides which one to take based on the digit of the n's place in the opcode. I group all the operations with a 0 as the MSB as one group and likewise for 1. Then I proceeded left to right, seeing where the opcodes differed to decide on their grouping. Overall the circuit contains two subsections, with the top being Eq, Or, And, Le, Gt, Nor, Xor, and the bottom being Add, Subtract, Right Shift Logical, Right Shift Arithmetic, and Left Shift Logical. There is additional logic on the Shifter so that C_{in} is only 1 when all the opcode bits are 1 indicating it should be right arithmetic and likewise for add/sub. The a


A
11100000
11010111
11000011
01001101
B
00000000
00000000
00000000
00000000
eq32
compare
MUX
ADDSUB
SHIFTsub
OpAr
SA
OPL
Op
0100
Sa
00000
V
C
00000000
00000000
00000000
00000001
Simulation halted by internal error