Neeharika Kotimreddy
CS 3410
Prof. Bracy
9/13/22

# CS 3410: PROJECT 1 DESIGN DOCUMENTATION

## I. ALU32

This is an overview of the RISC-V ALU displayed in Fig. 1 that can perform numerous computations on 32-bit numbers. It is made of five sub-function circuits along with two helper circuits that utilize the build it and box it principle, which allows for a cleaner ALU. The computational subcircuits consist of ne_eq, add_subtract, le_gt, logicgates, and subshift circuits, with LeftShift32, Add32, and Shift32 as further subcircuits in Shift32, add_subtract, and subshift respectively. The helper circuits consist of overflow, opselect, and a reverse subcircuit not displayed in this overall circuit but used in the Shift32 circuit.

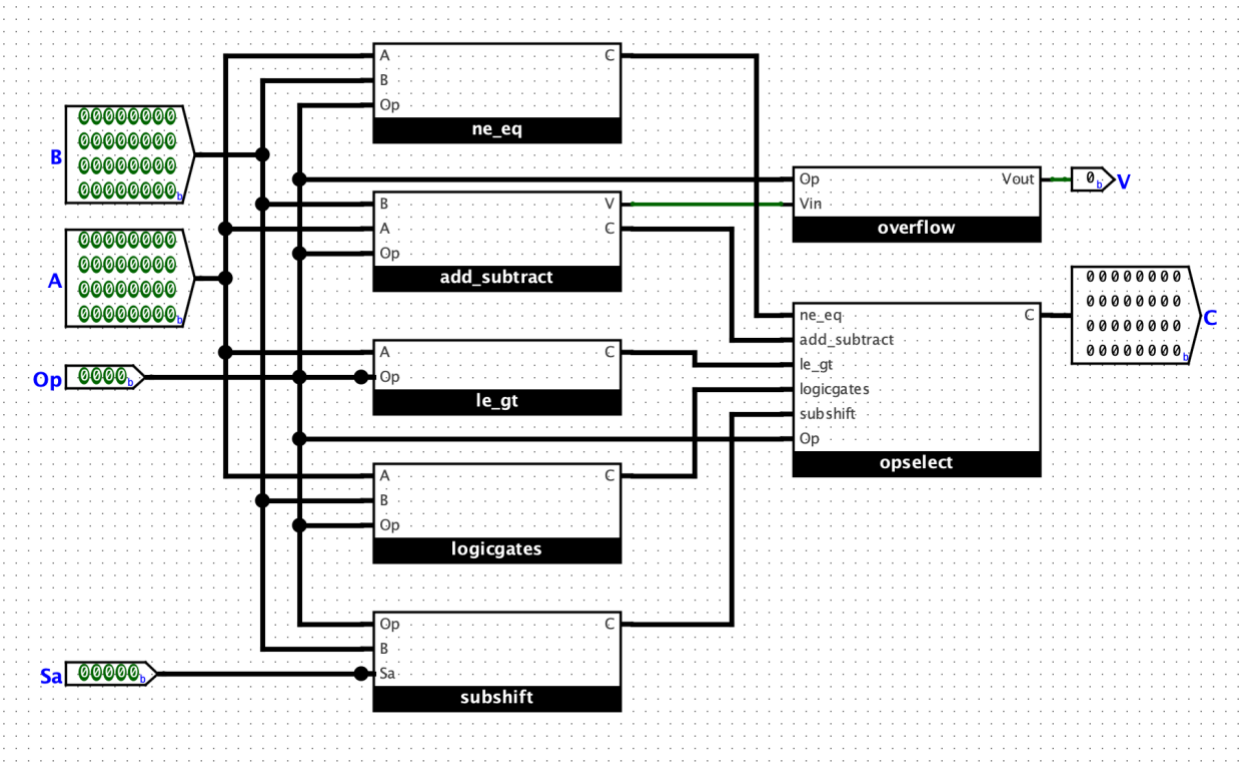


Fig. 1

## II. LeftShift32

This is an overview of LeftShift32 displayed in Fig. 2, a left shifter that shifts the input B to the left by Sa number of bits while filling the right with a Cin bit value, outputting a final answer of C. This circuit is made up of multiple sub-bit shift circuits, e.g., bitshift2 is made of two bitshift1 circuits, which is displayed in Fig. 3 below, bitshift4 is made of two bitshift2 circuits, bitshift 8 is made of two bitshift4 circuits, so on and so forth. Starting off with bitshift16, using a splitter, the most significant bit (MSB) of Sa is passed into a mux to determine whether we shift B by 16 bits or not. The mux process continues for each bit shift subcircuit, passing each consecutive bit to the right of the MSB of Sa for each respective bit shift until C is outputted.

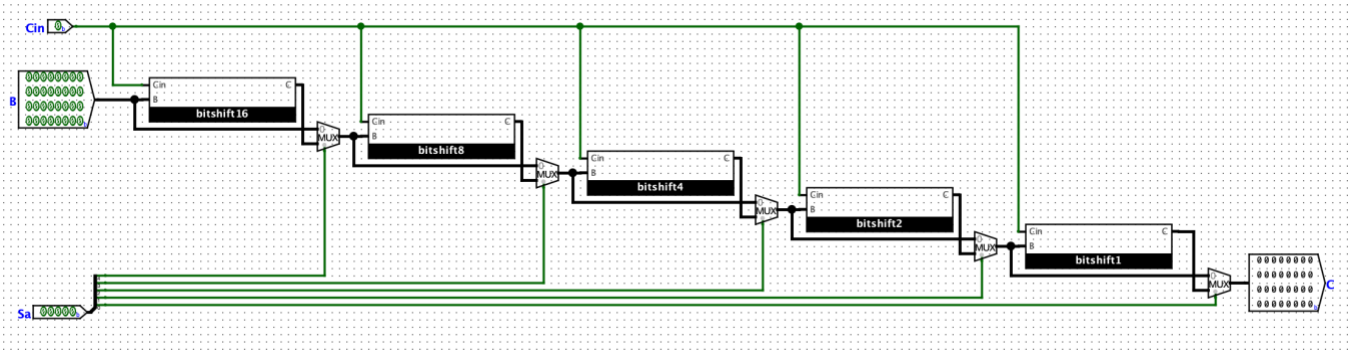


Fig. 2

### A. bitshift1

This one-bit shifter, displayed in Fig. 3, shifts one bit from 32-input B to the left. This shifter also takes a Cin value and fills the shifted bits with either 0 or 1 depending on the Cin value.

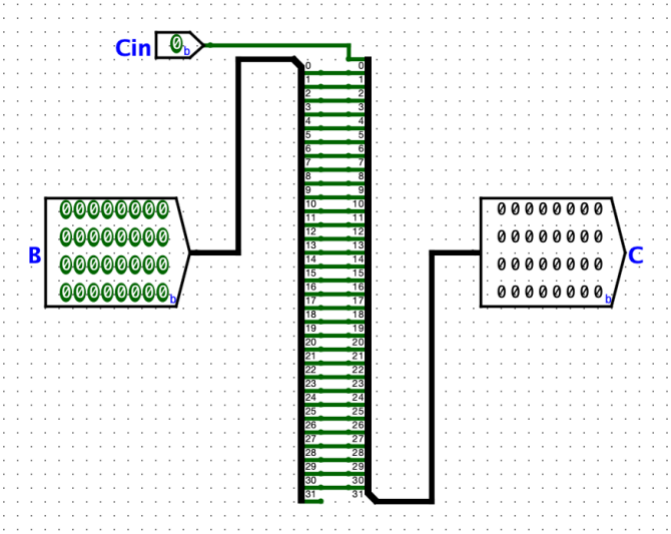


Fig. 3

## III. Add32

This is an overview of Add32 displayed in Fig. 4, a 32-bit adder that takes in input A and B, along with a Cin value, and outputs final sum value C along with an overflow value V. This circuit is made of multiple sub-add circuits, adding the 16 least significant bits, then the next 8, then the next 4, and so on and so forth until the most significant bits. In this case, I used four add1 subcircuits, which is displayed in Fig. 5, to construct the add4unsigned and add4 sub-circuit, which are displayed in Fig. 6 and Fig. 7 respectively. Each adder circuit passes a Cin value into the next add sub-circuit, and passes in the specific added bits to the final C output, which is done using a splitter.

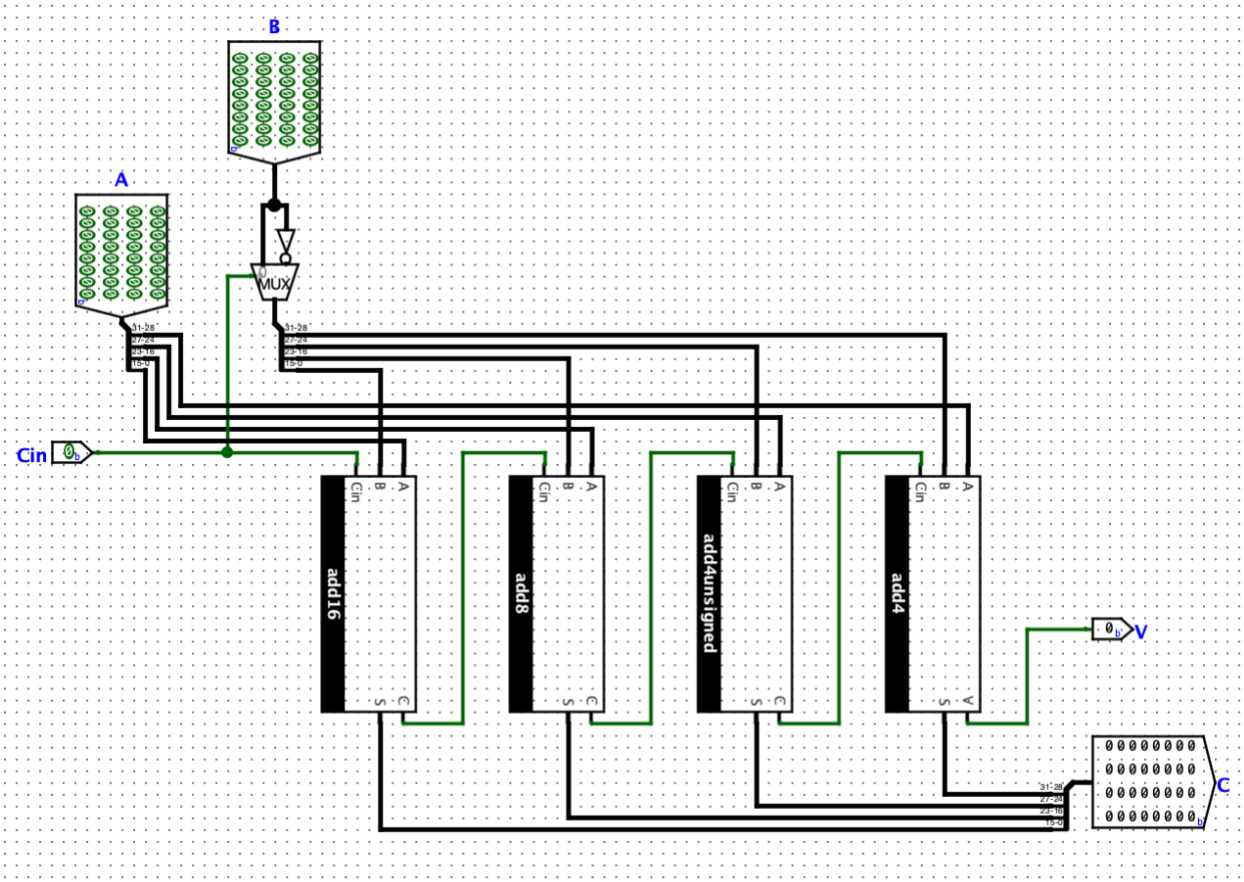


Fig. 4

## A. add1

The one-bit adder displayed in Fig. 5 was used to construct the many other sub-add circuits for Add32. AND, OR, and XOR gates are used to find the S and C values.

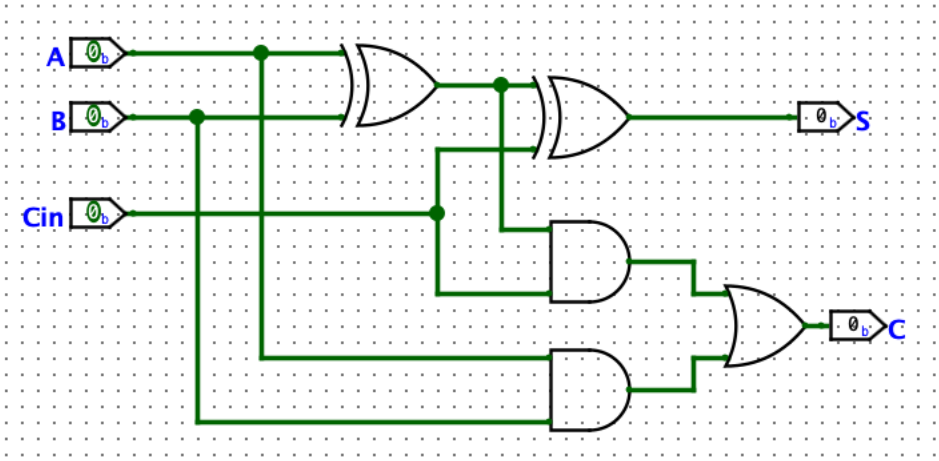


Fig. 5

## B. add4unsigned

The four-bit unsigned adder displayed in Fig. 6 was used to add bits 24-27 in the 32-bit inputs in Add32. Only the four least significant bits need to be passed through a signed adder so that overflow can be calculated, so this unsigned adder can be used for bits 24-27. This adder uses four add1 circuits with a Cin value to add two four-bit inputs.

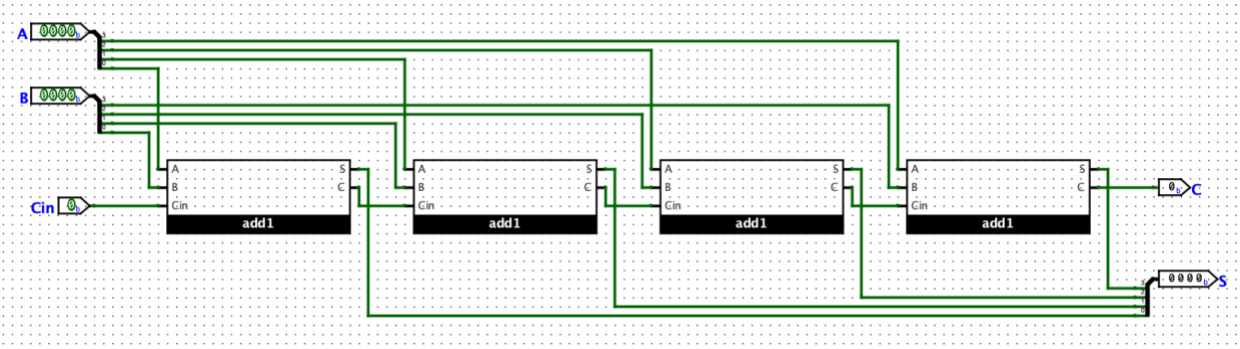


Fig. 6

### C. add4

The four-bit adder displayed in Fig. 7 was used to add bits 28-31, the least significant bits (LSB) in the 32-bit inputs in Add32. These four bits need to be passed through this signed adder so that overflow can be calculated. This adder also uses four add1 circuits with a Cin value to add two four-bit inputs, but also uses an XOR gate to calculate overflow V.

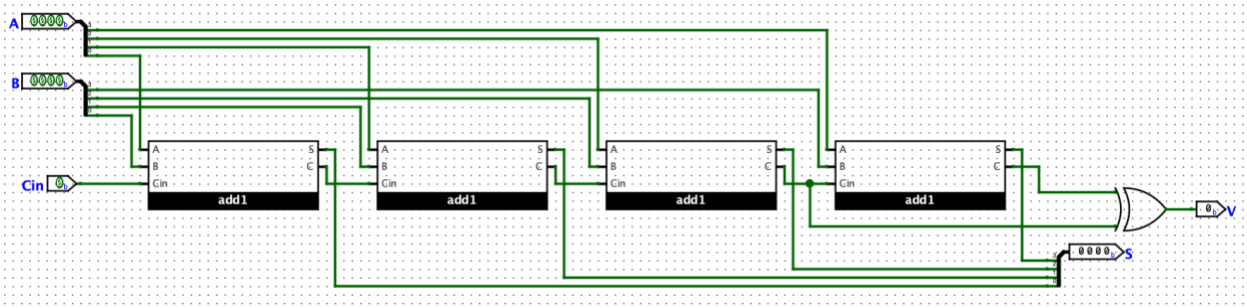


Fig. 7

## IV. Shift32

This is an overview of Shift32, displayed in Fig. 8, which shifts 32-bit input B to the left or right Sa number of bits and outputs C. The circuit uses an SL value, which indicates whether you shift the bits right or left. If the SL value is 0, the circuit shifts left, and if the value is 1, the circuit shifts right.

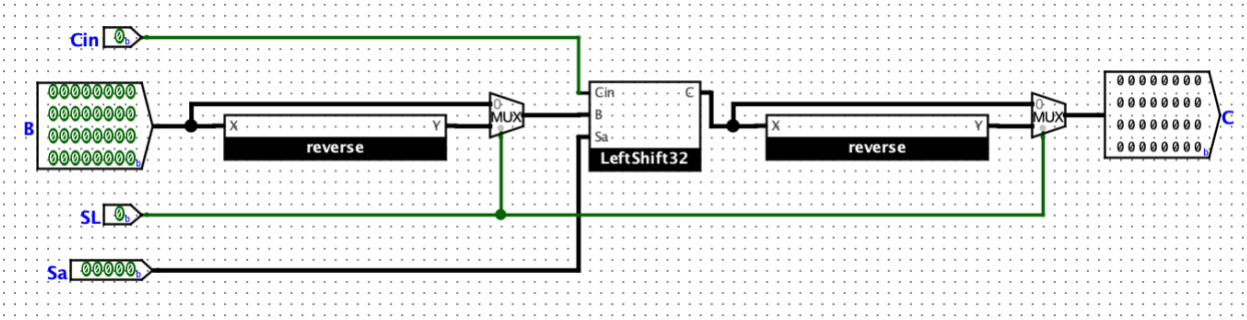


Fig. 8

## V. ne_eq

The ne_eq subcircuit, displayed in Fig. 9, performs the not equal and equal to operations. An XNOR gate is used to compare inputs A and B. Then, the 32 bits of the output are compared using an AND gate, which is once again passed into an XNOR gate. The $0^{th}$ bit of the opcode is also passed into the XNOR gate to determine whether we perform the ne or eq operation. The output is then zero extended and outputted as output C.

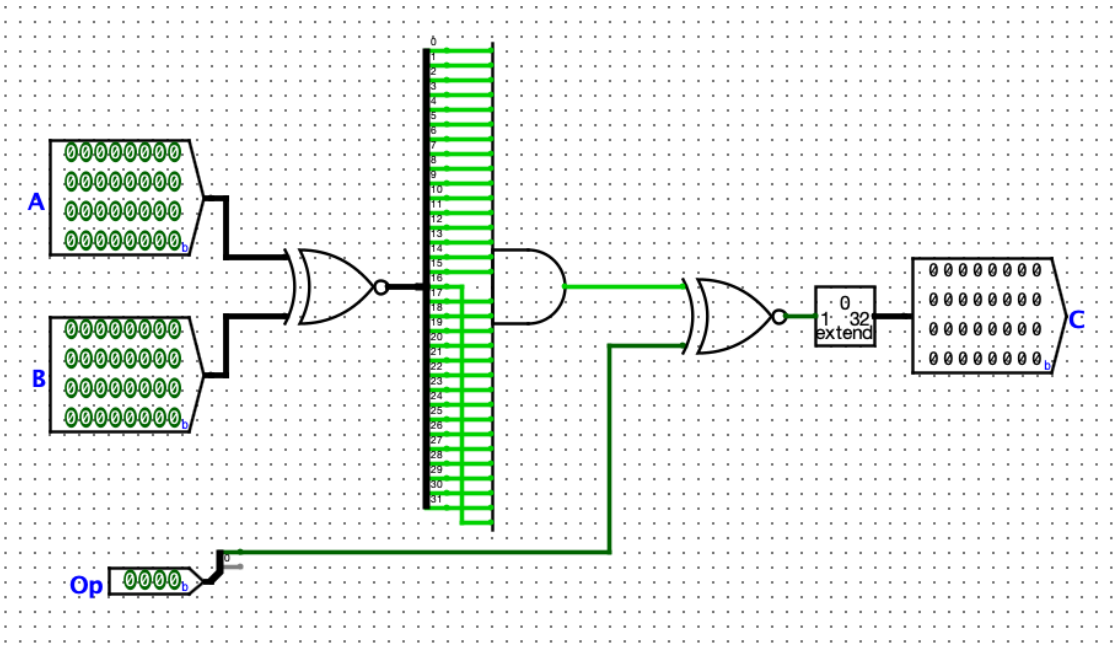


Fig. 9

## VI. le_gt

The le_gt subcircuit, displayed in Fig. 10, performs the greater than and less than operations. Using an NOR gate, all 32 bits of input A are passed through the gate, and this is then compared to the most significant bit of A through an OR gate. This is then passed into an XOR gate that compares that input with the $0^{th}$ bit of the opcode, and this determines whether the operation is less than or greater than. The output is then zero extended and outputted as output C.

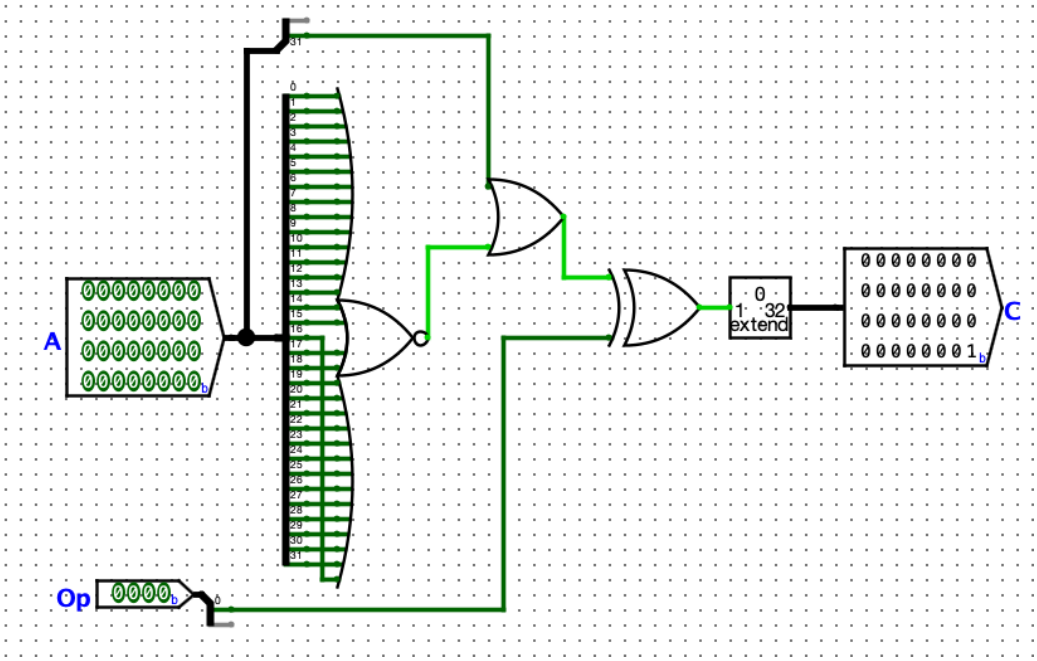


Fig. 10

## VII. subshift

The subshift circuit, displayed in Fig. 11 below, uses Shift32 as a subcircuit to perform the left shift logical, right shift logical, and right shift arithmetic operations. An opcode is passed into the circuit and the 3rd, 2nd, and 1st bit are passed through various AND gates, which determines the type of shift that is performed. To perform the left shift, the SL value needs to be 0. To perform the right shift logical, the SL value needs to be 1, and to specifically perform the right shift arithmetic, the opcode must fully match and the MSB of the B input must be 1. If so, a Cin value of 1 will be passed into the Shift32 circuit, which will input a value of 1 into every left bit from the shifted bits.

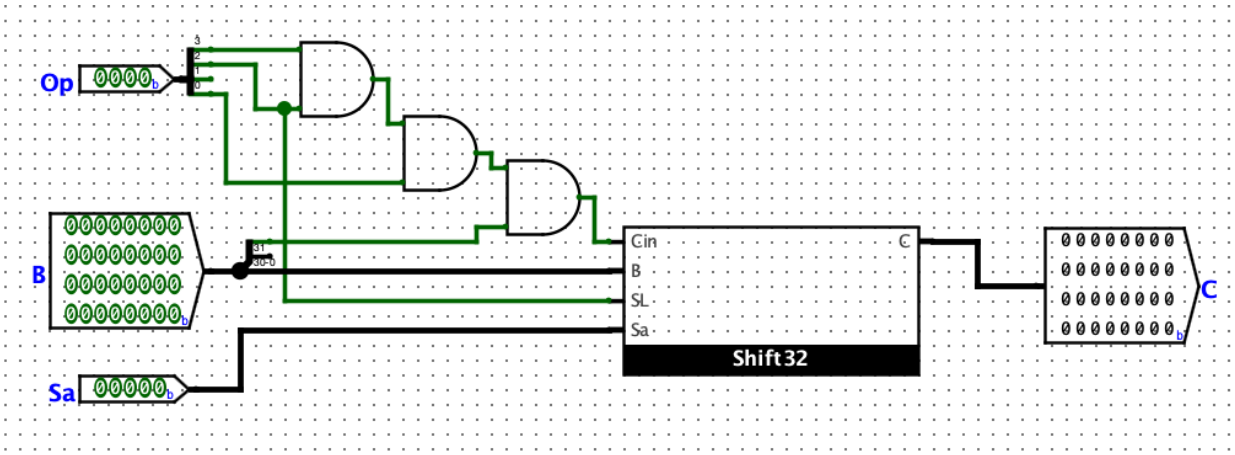


Fig. 11

## VIII. logicgates

The logicgates circuit, displayed in Fig. 12 below, performs the basic OR, AND, XOR, AND NOR operations. The circuit takes in the opcode and 32-bit inputs A and B. Using the 0th and 1st bits to differentiate within the opcodes, the circuit muxes these two bits to determine which logic gate to use, and outputs value C.

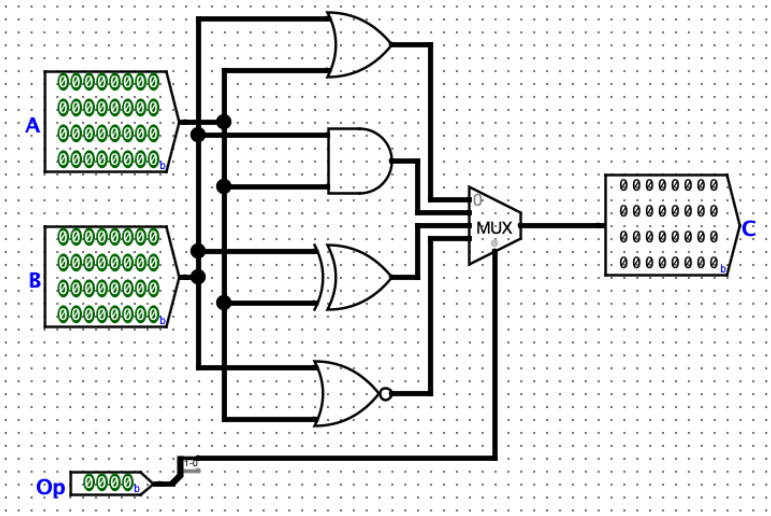


Fig. 12

## IX. add_subtract

The add_subtract circuit, displayed below in Fig. 13, performs both the add and subtract operations by using Add32. It takes in an opcode and 32-bit inputs A and B and passes in the 3rd bit of the opcode as the Cin value. Overflow V and output C are outputted.

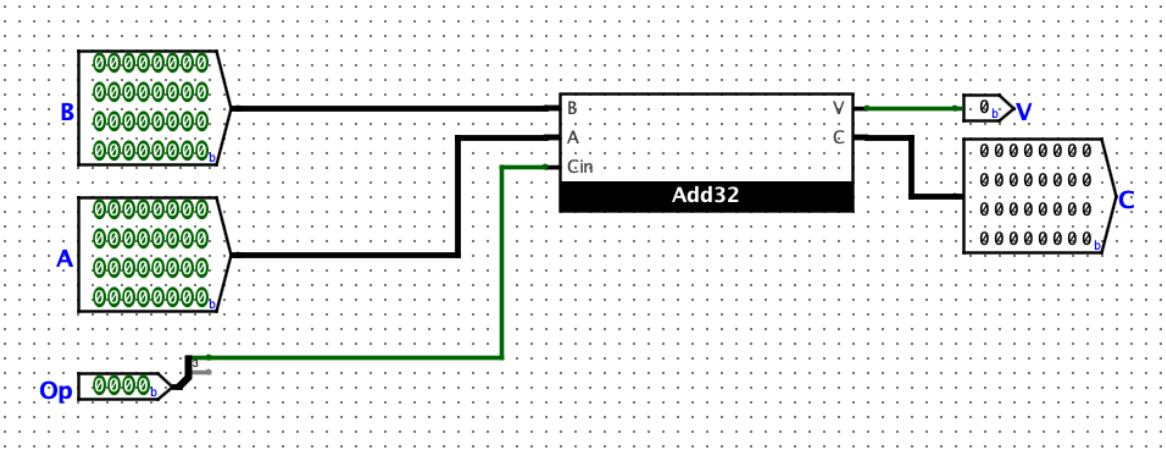


Fig. 13

## X. reverse

The reverse subcircuit, displayed in Fig. 14, is a helper circuit used in the Shift32 circuit. It takes in an input of X and reverses the bits using a splitter, and then outputs Y.

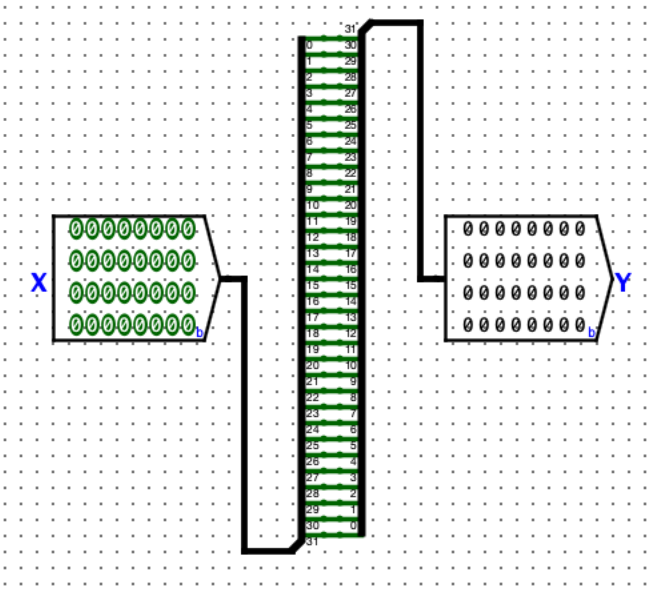


Fig. 14

## XI. overflow

The overflow circuit, displayed in Fig. 15 below, is a helper circuit used in ALU32 to determine whether the operation performed in add_subtract yields an overflow value Vout that needs to be outputted in the main ALU as V. It takes a Vin value of either 0 or 1 and passes the 1st, 2nd, and 3rd bits of the opcode through AND and XOR gates and then passes the value into a mux to determine whether Vin should be outputted as Vout in ALU32.

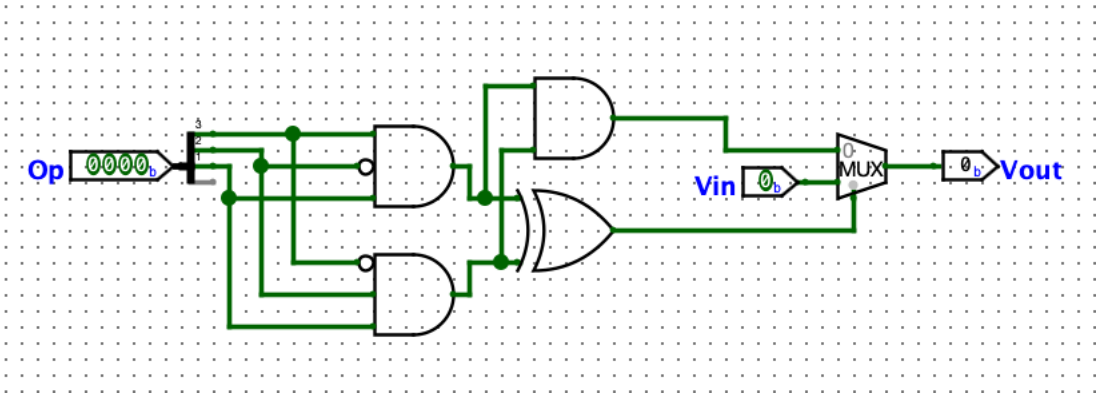


Fig. 15

## XII. overflow

The opselect circuit, displayed in Fig. 16, is a helper circuit used in ALU32 to help determine which of the sub-circuit outputs from the operations should finally be outputted as the C value. This circuit utilizes the build it and box it method, which allows for a clean ALU. I used numerous gates to differentiate between opcode bits. First, I checked for the ne_eq opcode and passed the value into the mux. If the value of the opcode is true, then the ne_eq output would pass, or else the add_subtract output would pass. For subsequent muxes, I checked for le_gt, logicgates, and subshift. If any of the opcode values were true after the gates comparisons, the output for those respective circuits would pass. If none of them are true, then the output would default to the add_subtract value, which would be outputted as C.