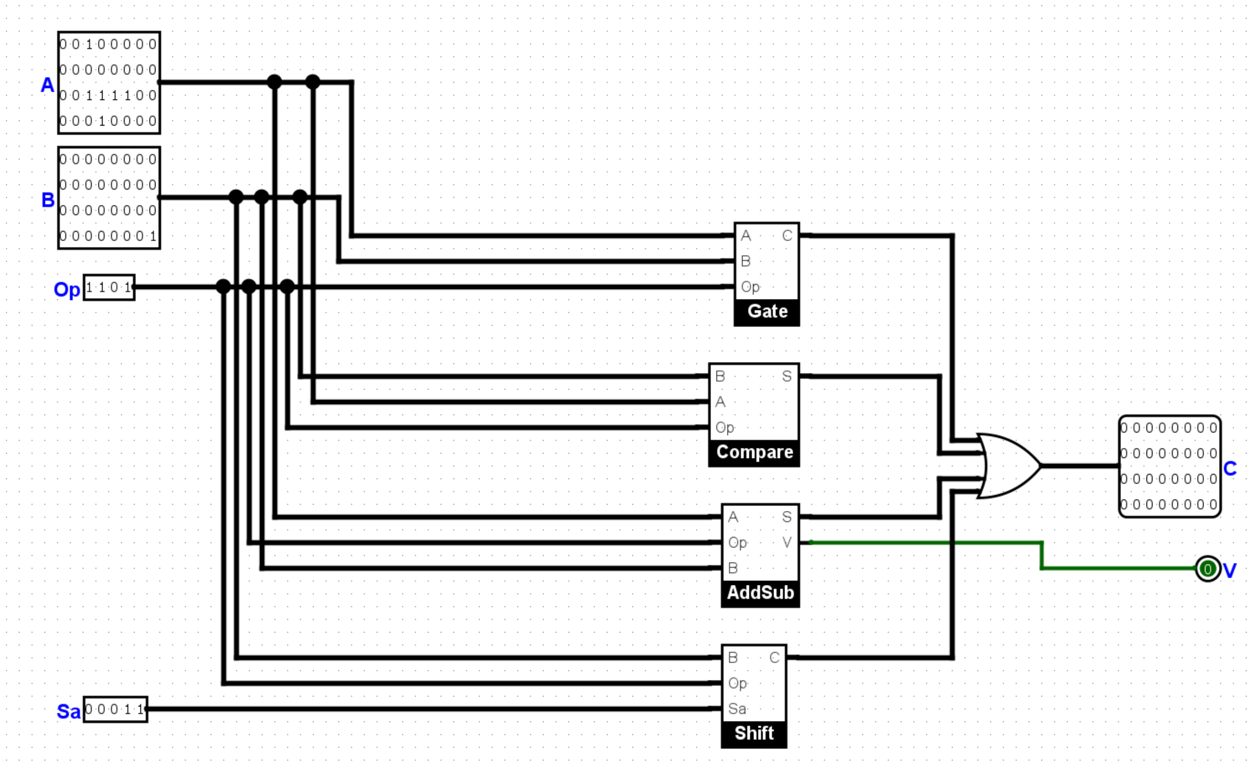
Project 1 - ALU Design

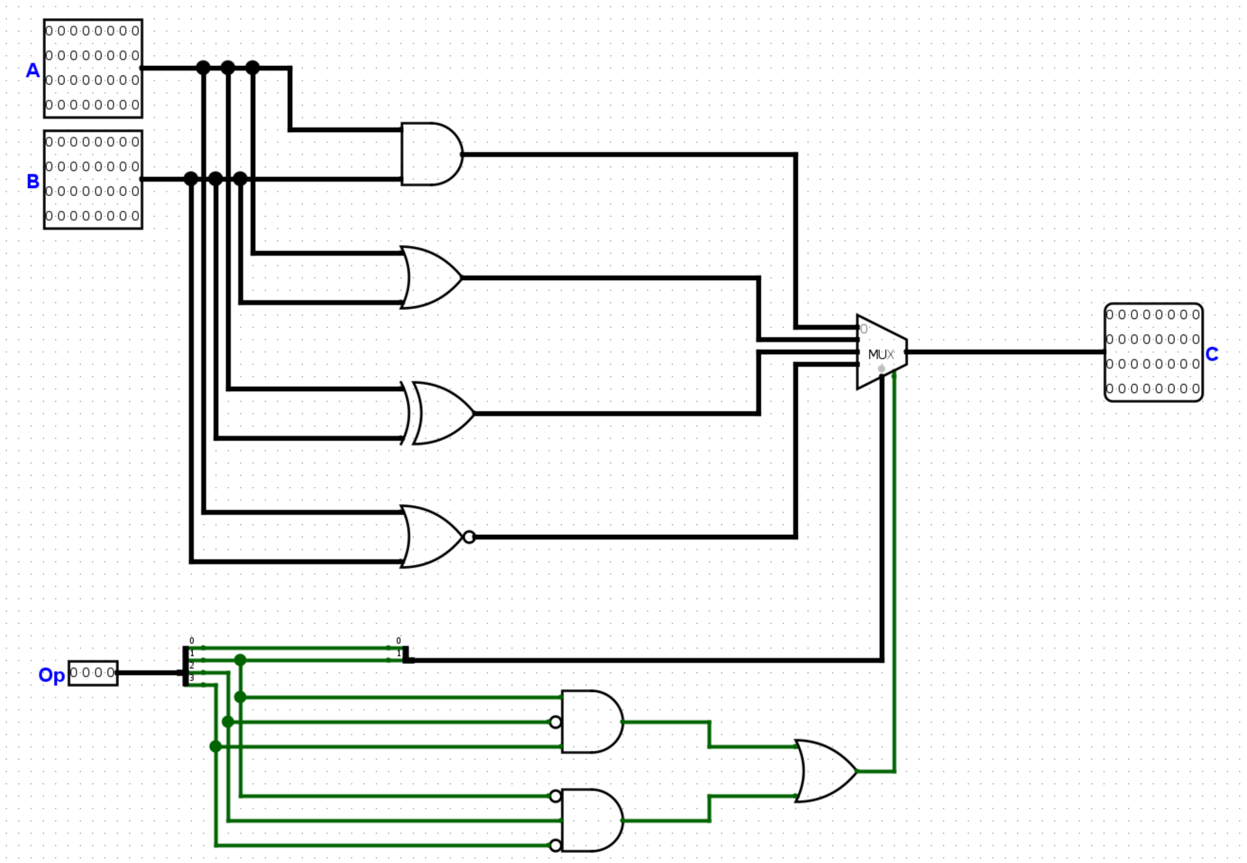
Dau Vu Dang Khoi – V202000116

1. Overview:

This is a 32-bit ALU capable of performing core functions on one or two 32-bit numbers. The logic operation, compare operation, add or subtract operation, and shift operation are all included in this ALU.

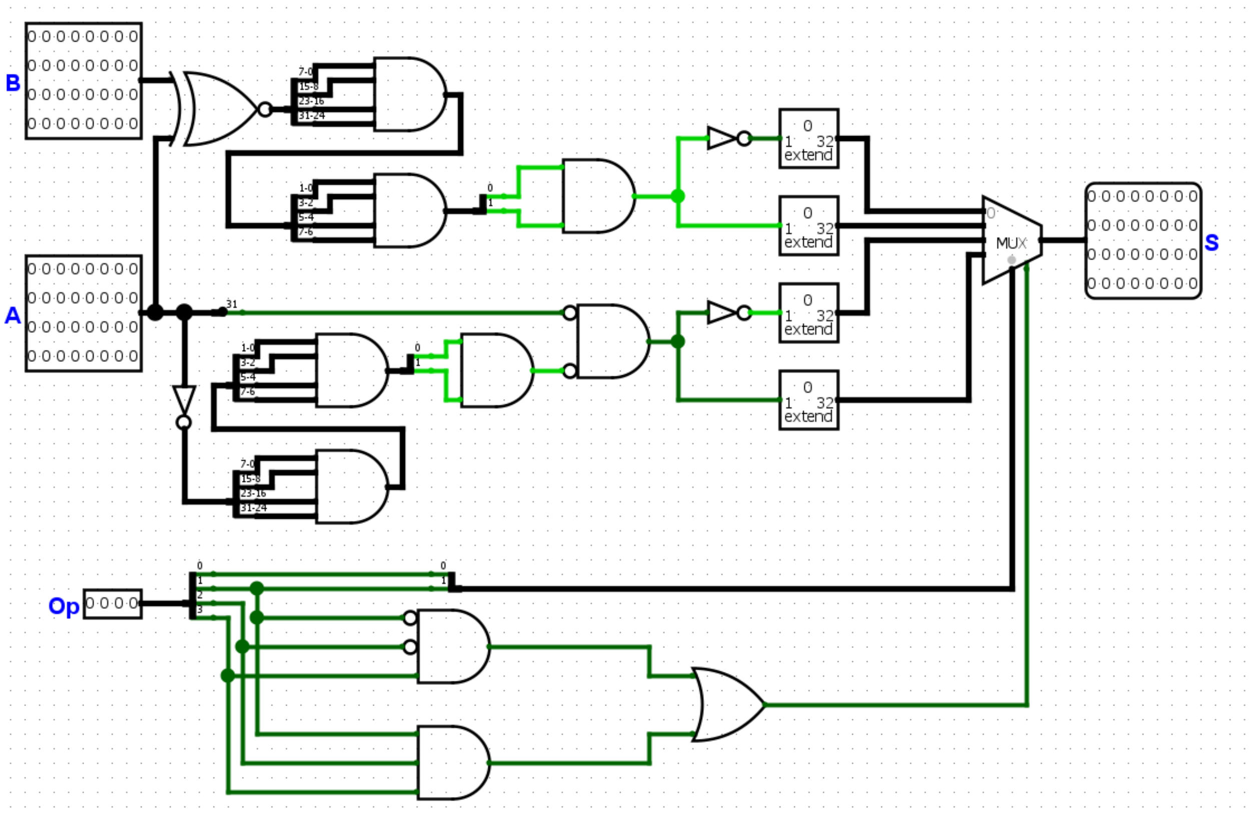


2. **Logic Operation:**
   - The 4 logical operators that the ALU perform are AND, OR, XOR, NOR.
   - We utilize a mux to pick what to output as a result that matches to the Op. The input signal is routed into a 2x4 multiplexer by combining Op bit 0 and 1 onto a single wire.



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

3. Compare Operation:

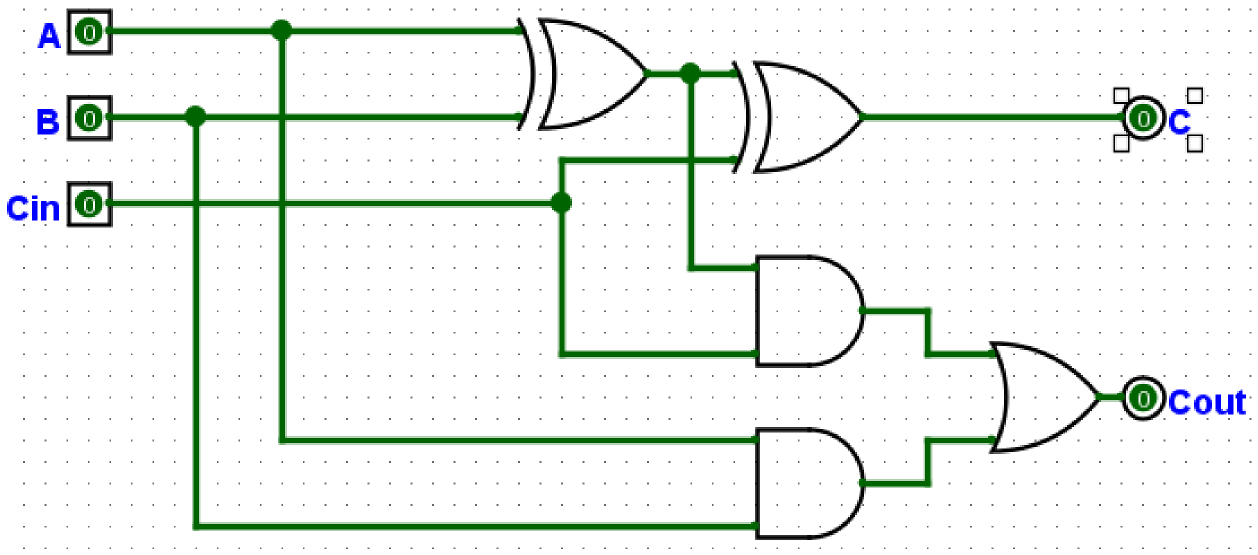


- A and B are compared. A^B equals a 32-bit zero if A equals B. To see if A^B is 0, use AND gates. If the answer is yes, A = B; if the answer is no, A !=B.
- We verify whether A >0. If A > 0, its 31st bit will be 0, while at least one of the remaining bits will not be zero. A > 0 if it meets this criterion; otherwise, A ≤ 0.
- The input signal is routed into a 2x4 multiplexer by combining Op bit 0 and 1 onto a single wire.

Op bit 0	Op bit 1	Output
0	0	(A != B) ?
0	1	(A == B) ?
1	0	(A ≤ 0) ?
1	1	(A > 0) ?

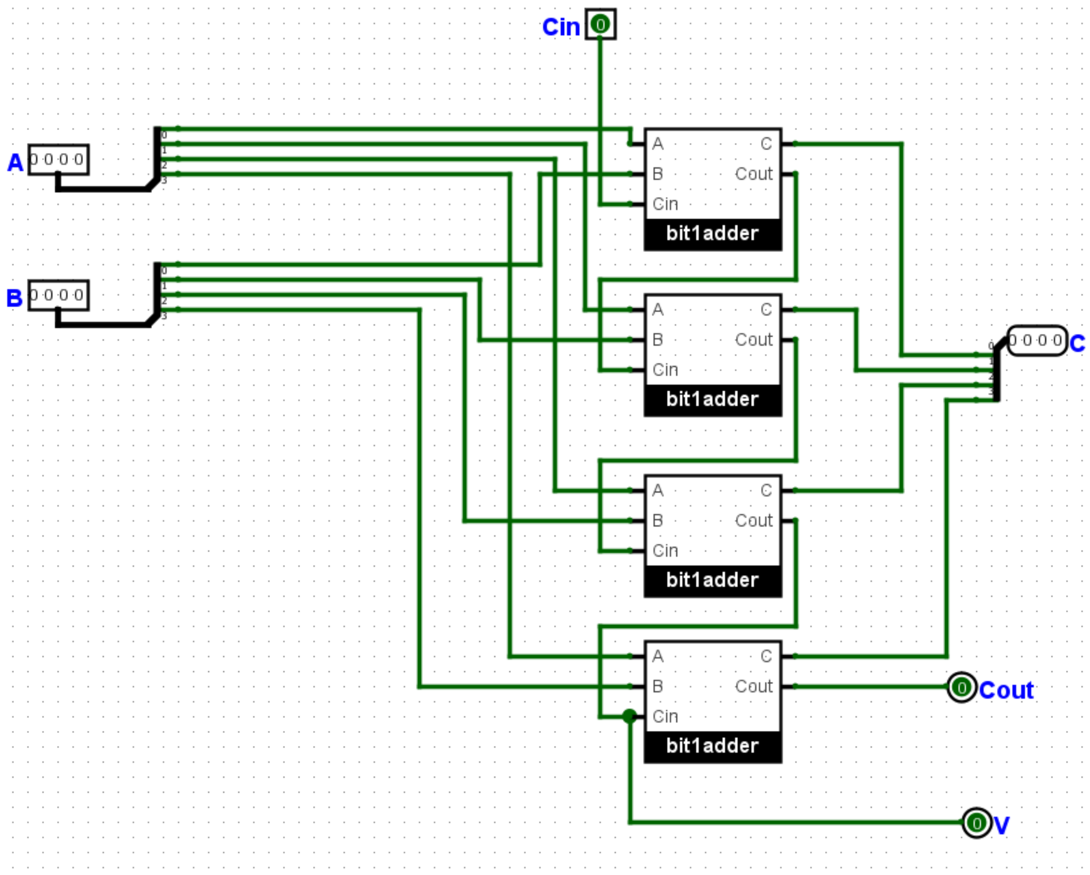
4. Add/Subtract Operation:

a. 1-bit full adder:



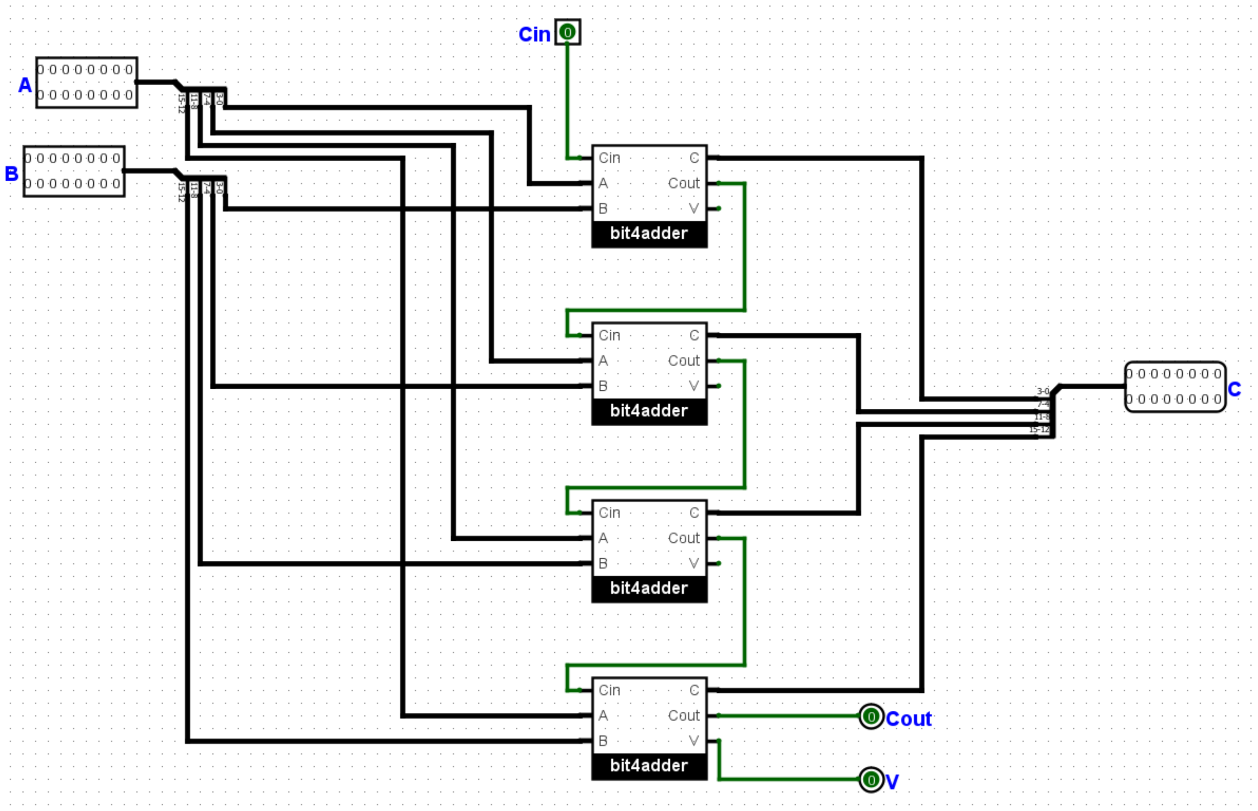
A and B are 1-bit inputs, Cin is the carry-in bit, S is the output bit, and Cout is the carry-out bit.

b. 4-bit full adder:



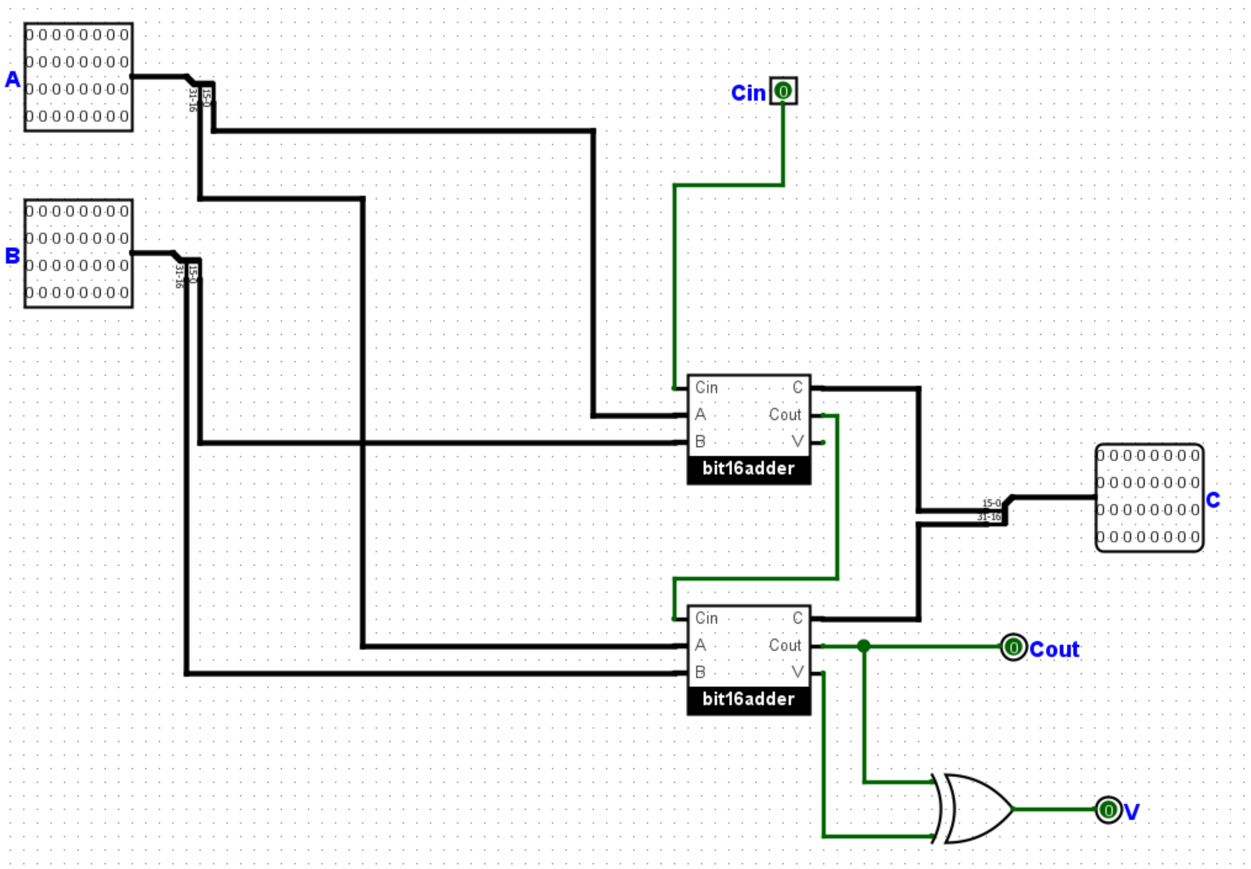
We make a 4-bit full adder by merging four 1-bit full adders. V is an output that allows us to see if it's an overflow.

c. 16-bit full adder:



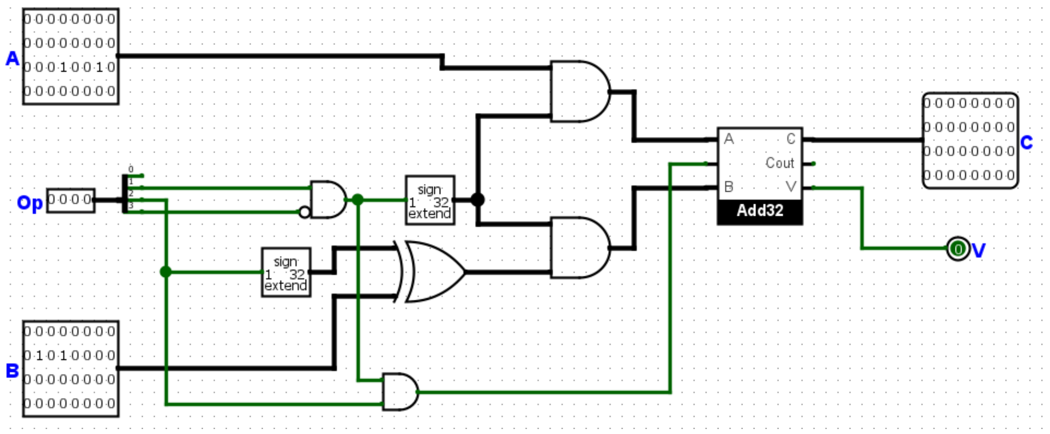
We make a 16-bit full adder by merging four 4-bit full adders. V is an output that allows us to see if it's an overflow.

d. 32-bit full adder:



We make a 32-bit full adder by merging two 16-bit full adders. V is an output that allows us to see if it's an overflow.

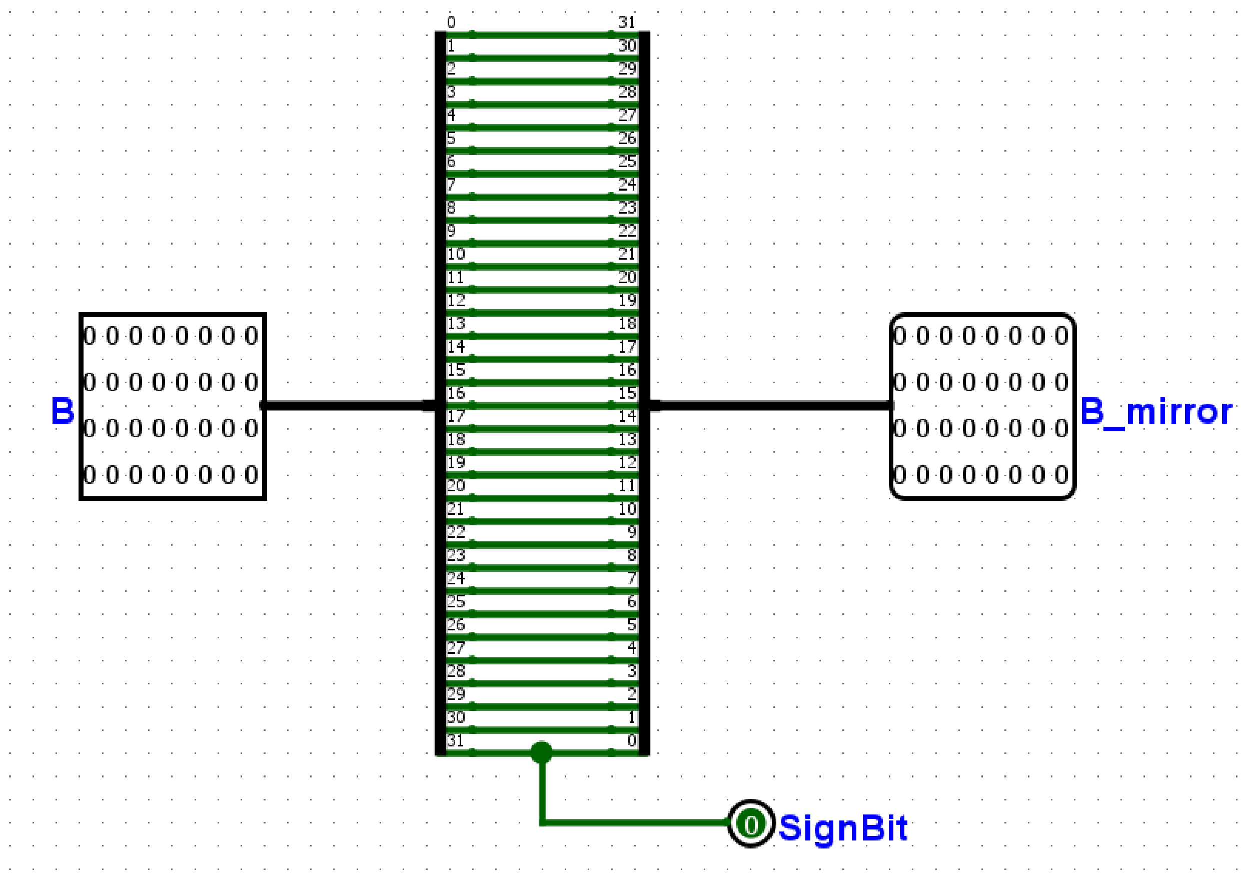
e. Operation part:



- Bit 2 of the Opcode input is in charge of handling the Add/Sub operation. If Op Bit 2 is 0, A+B is assessed, and if Op Bit is 0, A-B is accessed. In case of taking the inverse of B, a bit extender and a XOR gate are implemented.
- For managing Opcode bits, using an AND gate with two inputs: a bit extender and a negated input.

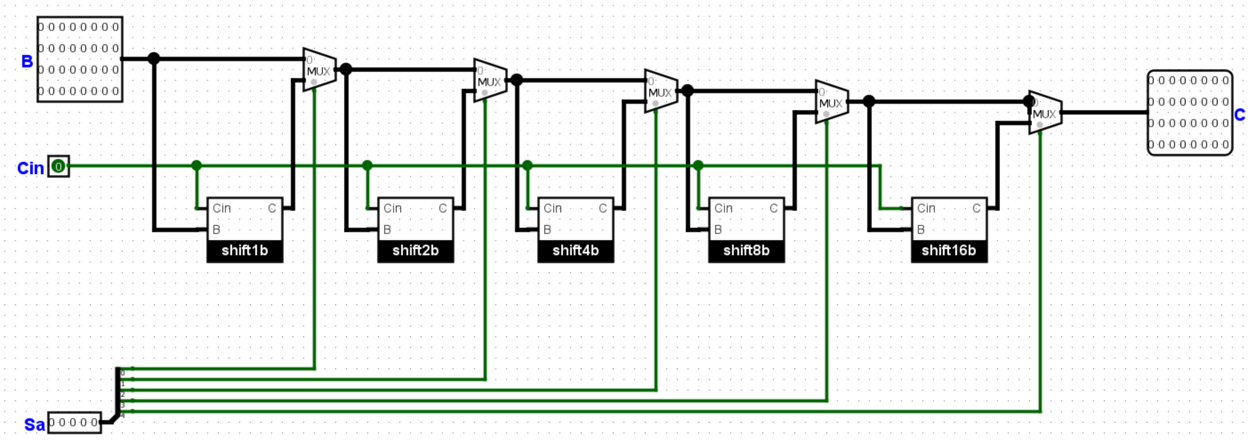
5. Shift Operation:

a. Mirror:

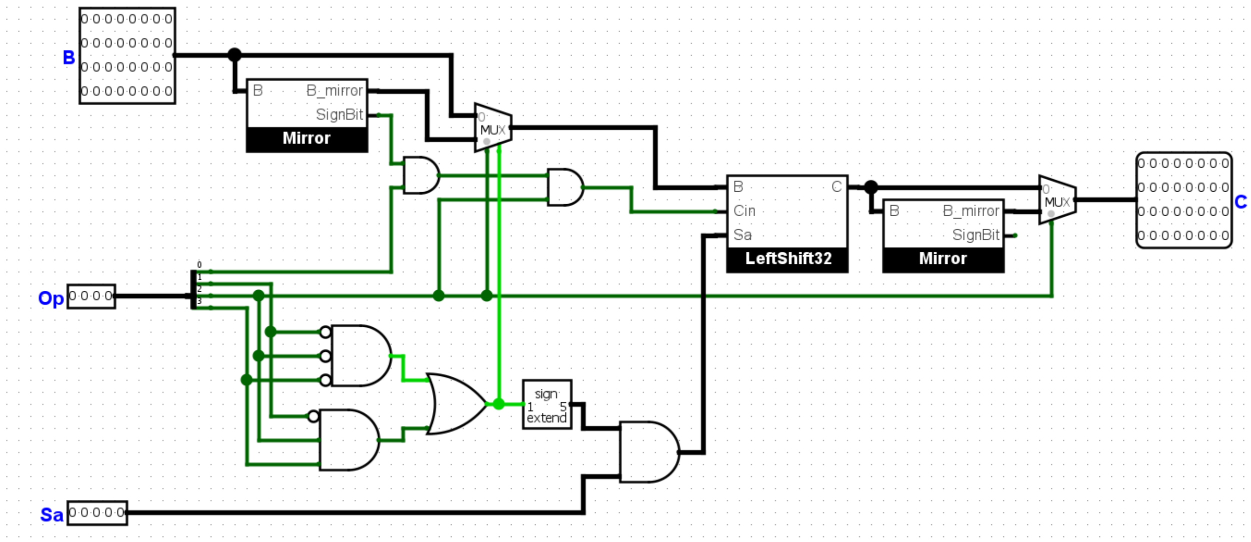


The mirror circuit mirror the input and output the input's Sign bit. This sign bit can be utilized to send Cin into the LeftShift32 circuit when executing an arithmetic right shift.

b. LeftShift32:



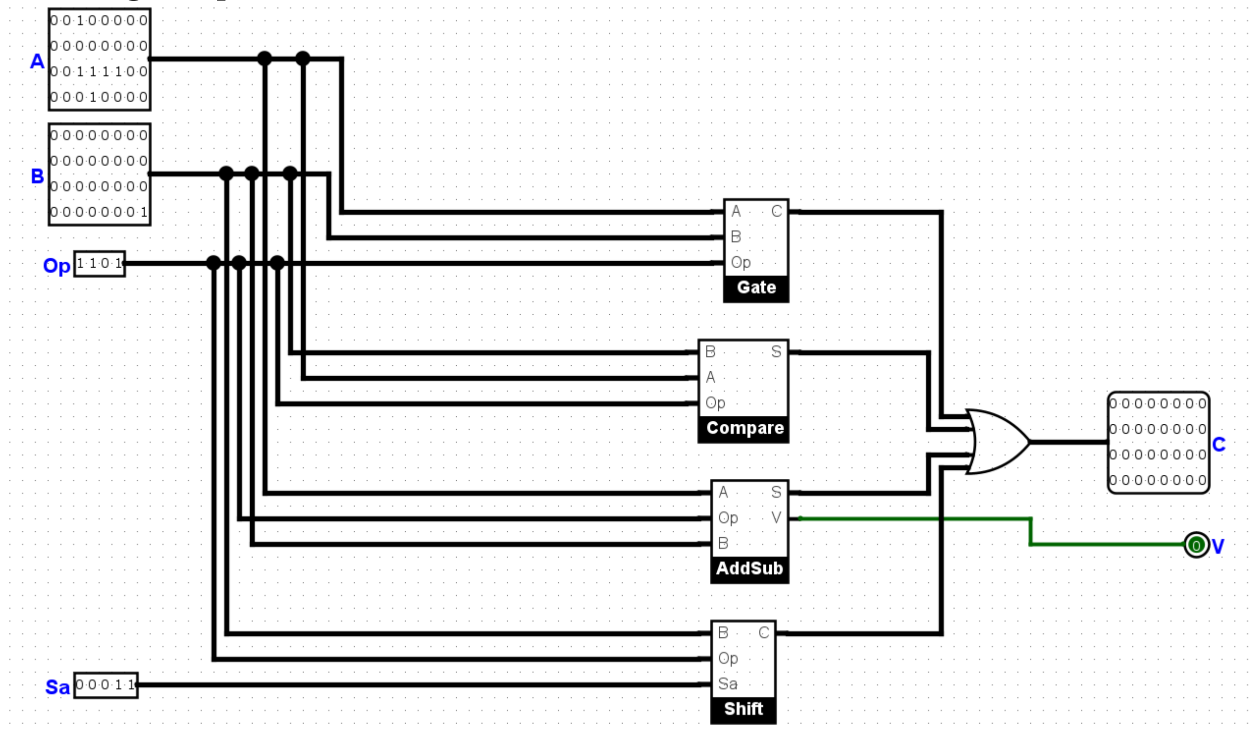
c. Shift Operation:



- The rationale for determining the shift type is based upon two mux. The first multiplexer takes both B and the inverse of B.
- Utilizing Op bit 2 as the control bit. If the shift is to the left, B is remains unaltered, and if the shift is to the right, B is inverted.
- Regarding the second multiplexer, if right shift was performed, this multiplexer will re-reverse B.

Op bit 2	Output
0	A << B
1	A >> B

6. Gathering four parts:



To output the only non-zero result into C, we implement an OR gate.