VinUniversity

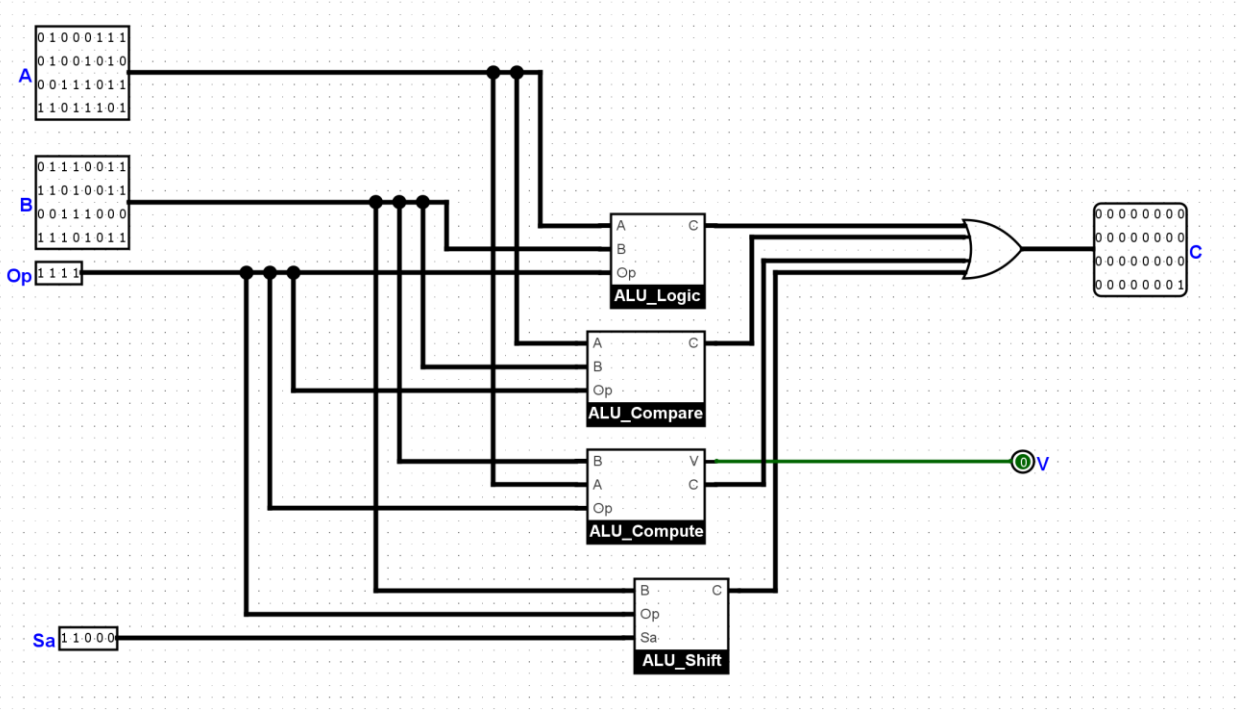
RISC-V ALU Design

Project 1

Tran Tuan Viet (S.CECS)
10-19-2021

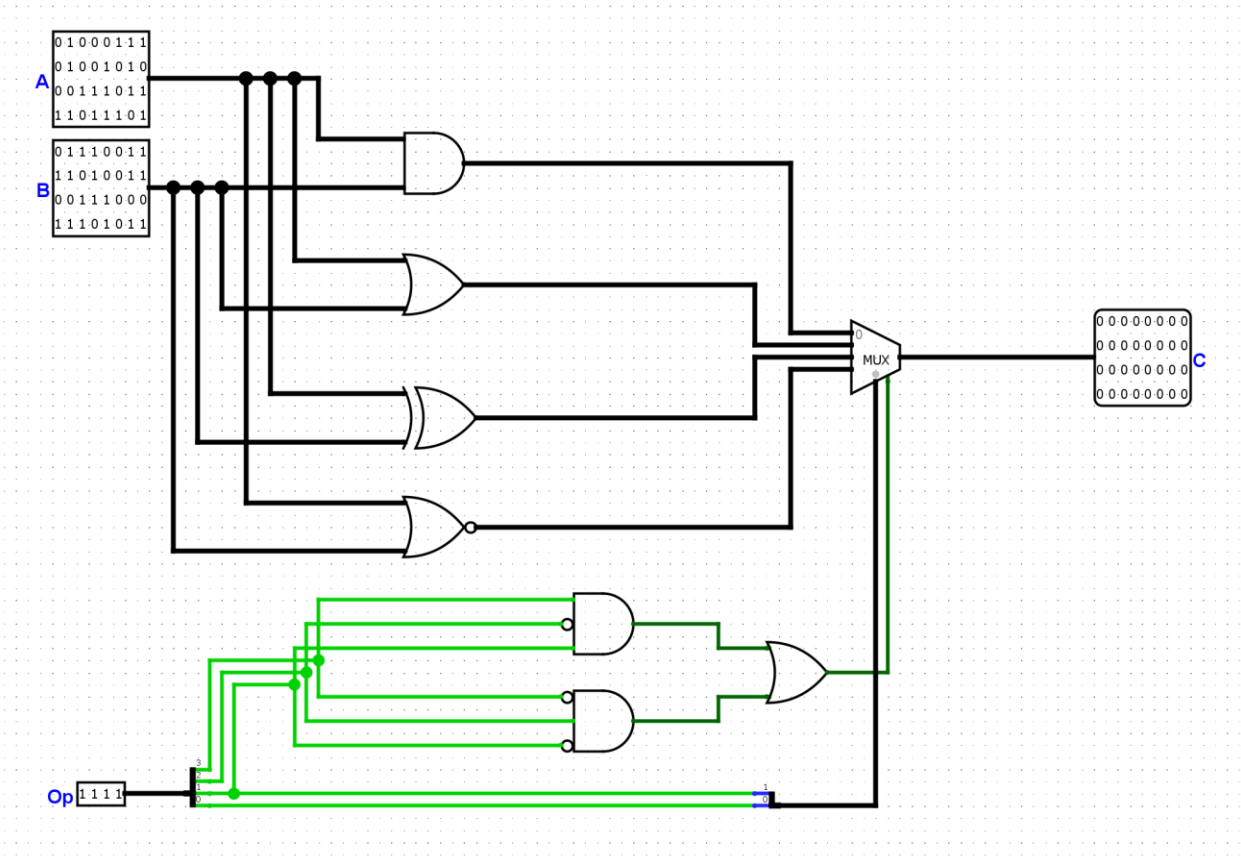
1 Introduction

This is a basic 32-bit ALU which can implement basic function for one or two 32 bits number. The circuit is divided into 4 sub circuits including Logic operation, Compare operation, Add/Subtract operation and Shift operation



The circuit is designed so that for each Op code, there will only be one operation function which means, the Op code of comparison cannot activate the shift function, or the AND op code cannot activate OR operation, and so on. The OR gate in the diagram will choose the only the result that is not 0 to the output.

2 Logic Operation

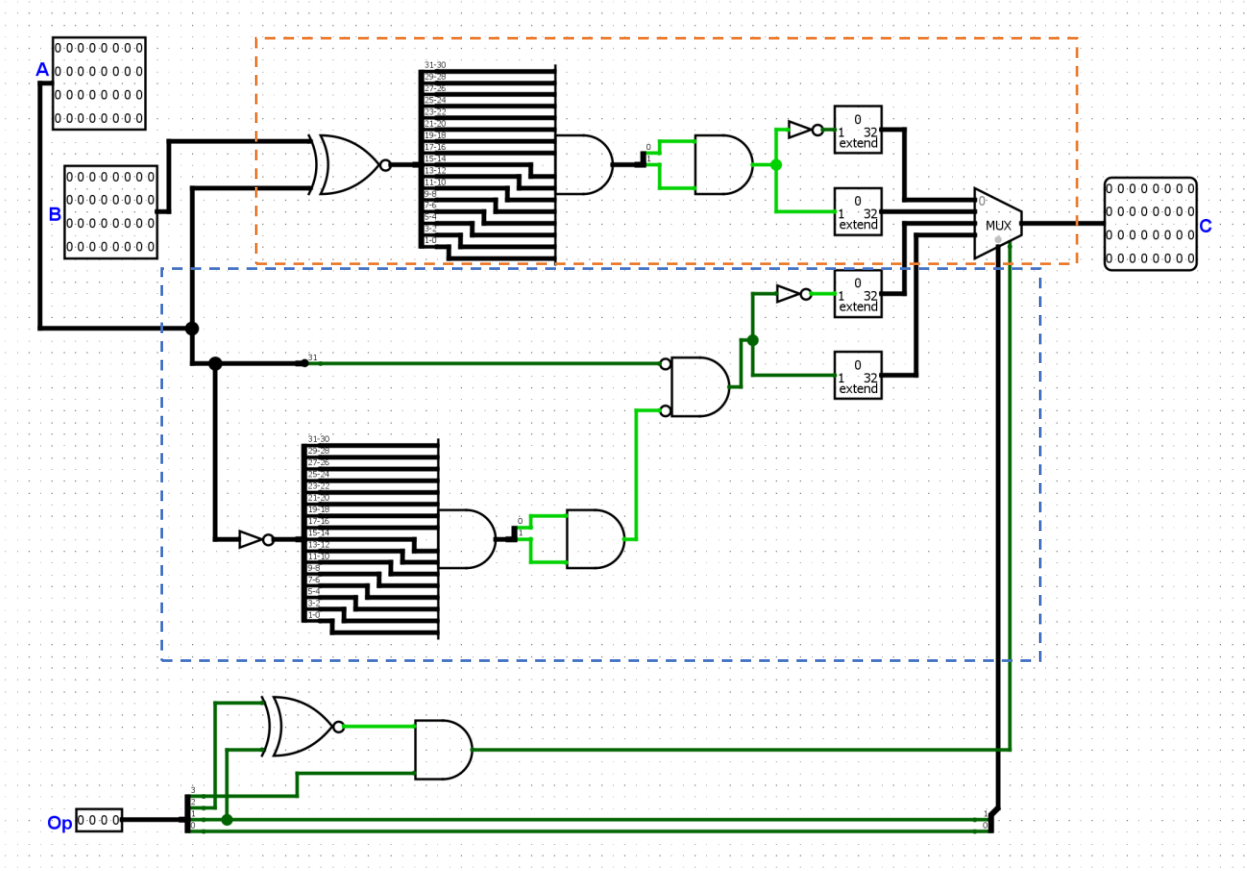


Logic operations include 4 32-bit gates to implement the AND, OR, NOR, XOR functions

To choose which functions to operate, I use the MUX to separate the output based on the operation code (Op). If the first 3 digits of is not 101x or 010x the output will be 0. If Op satisfy the condition, the MUX will choose the output based on the operation code.

Op	Input MUX	MUX	Operation
0100	00	0	AND
0101	01	1	OR
1010	10	2	XOR
1011	11	3	NOR
Else	00	Disabled	None

3 Compare Operation



Divide the circuit to two parts:

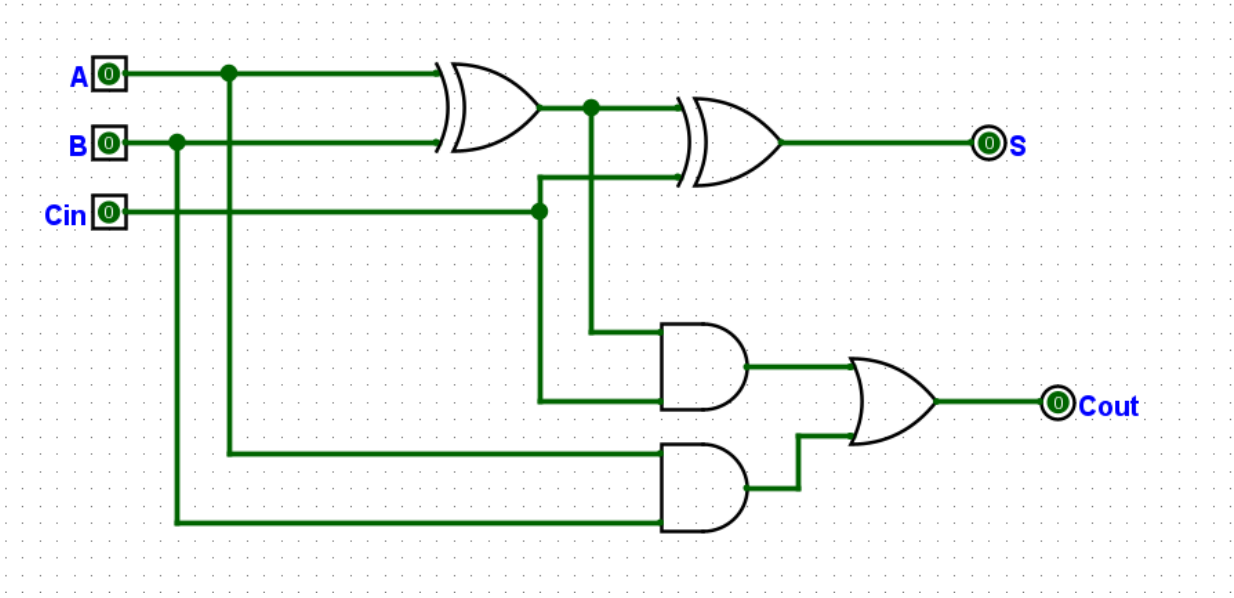
- The part inside orange square is used to compare A and B, if A is equal to B than A AND B equals to a 32-bit 0. Otherwise, A is not equal to B,
- The part inside blue square is used to check if A > 0 or A <= 0. If A > 0 the 31th bit will be zero and there are at least another bit that is not 0. Otherwise, A <=0.

Same as Logic, we use MUX to choose which operation we are going to use based on the operation code (Op). The first digit of the Op should be 1 and the second & third digit should be either 00 or 11 so the code should be 100x or 111x. If not disable the MUX and output would be 0

Op	Input MUX	MUX	Operation
1000	00	0	ne
1001	01	1	eq
1110	10	2	le
1111	11	3	gt
Else	00	Disabled	None

4 Add/Subtract Operation

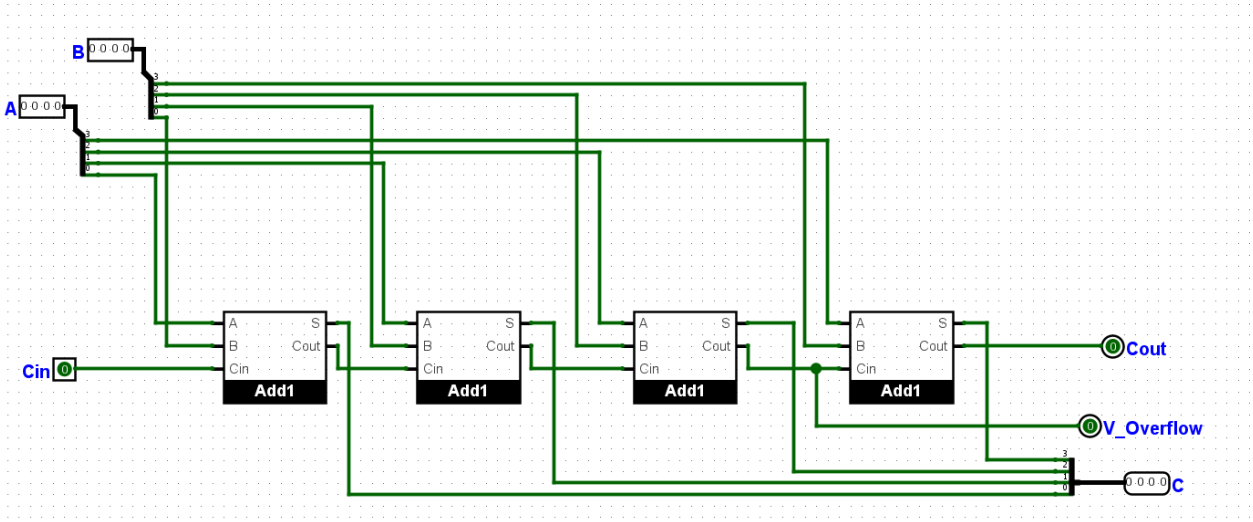
A, 1-bit full adder



This is a 1 bit full adder, A and B are the 1-bit input, while C_{in} is the carry-in bit. S is the output bit, while C_{out} is the carry out bit. This is based on the truth table.

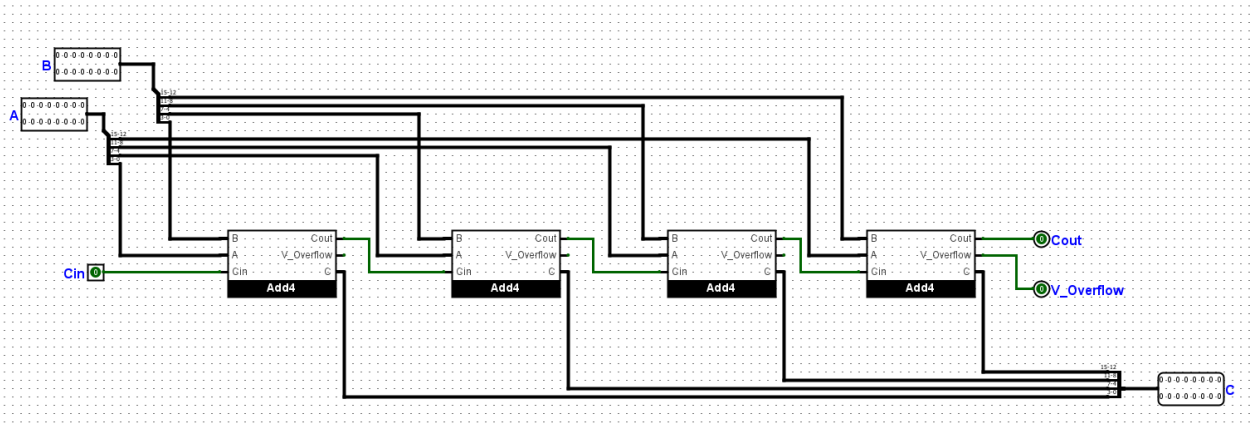
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1

B, 4-bit adder



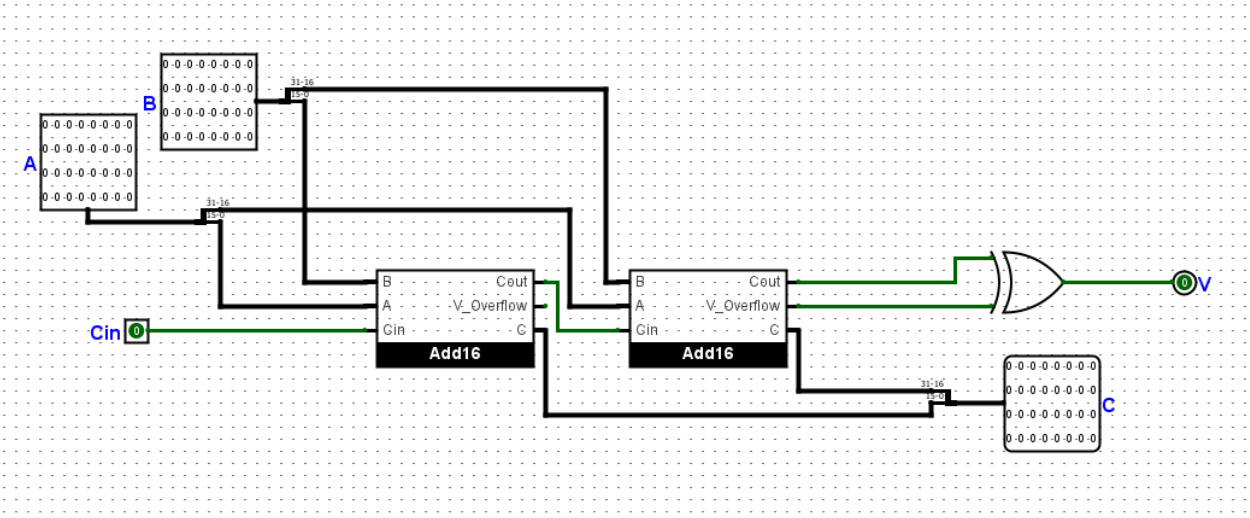
4-bit adder consist of 4 1-bit adders. V_Overflow is used to indicate if over happens

C, 16-bit adder



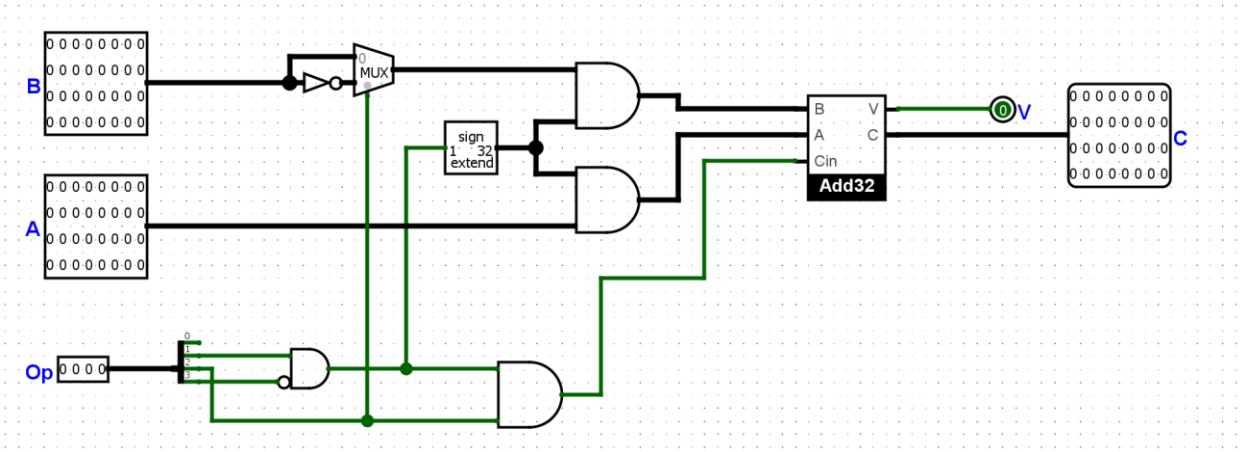
16-bit adder consist of 4 4-bit adders. V_Overflow is used to indicate if over happens

D, 32-bit adder



32-bit adder consist of 2 16-bit adder, if C out is not equal to Overflow, V will be indicated overflow

E, Add/Subtract Operation



First, the Op code should be in the form 0x1x. Otherwise, 0 will be put in Add32. And Cin will only be on if Op is 011x. Therefore, the output will only be 0 if Op code is wrong.

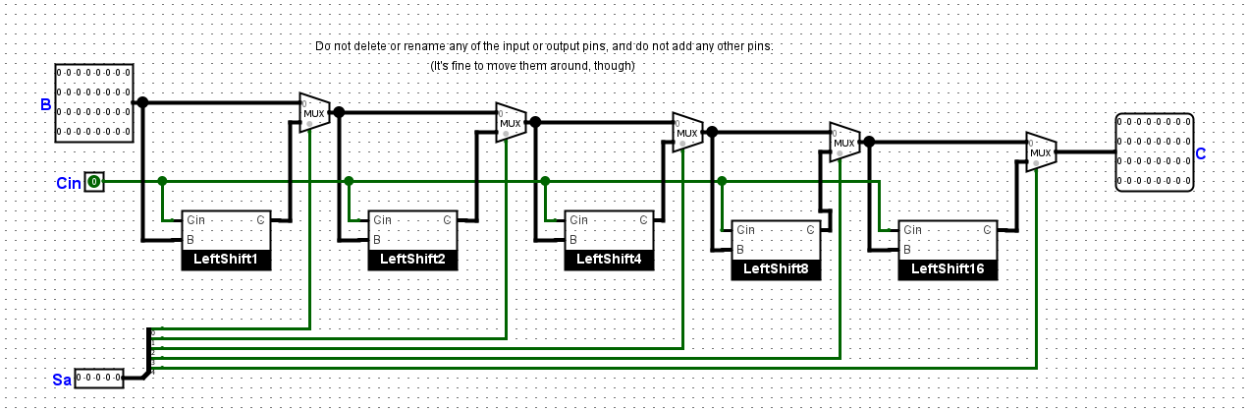
Instead of creating a subtract circuit, we will put -B in the adder which make it become A+(-B) = A-B.

Here a MUX is used to switch between B & -B. If Op is 001x, the MUX will choose B to push into Add32. If Op is 011x and Cin is 0, if Op is 011x, the MUX will choose -B to push into Add32 and Cin will be 1.

Op	Cin	MUX	B value	Operation
001x	0	0	B	add
011x	1	1	-B	subtract
Else	0	Disabled	0	None

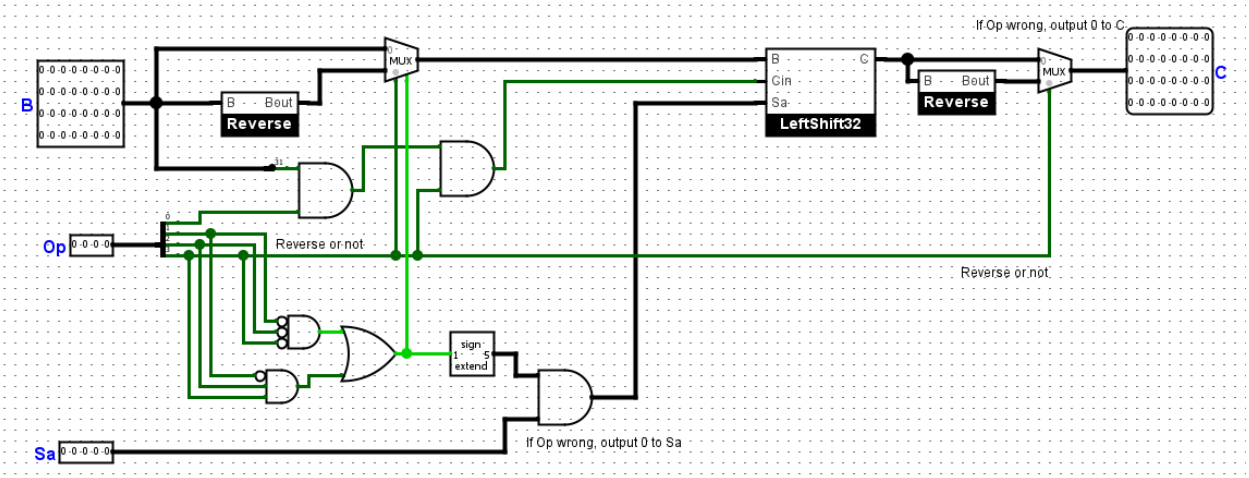
5 Shift Operation

A, Leftshift 32



As built in Previous Lab, Leftshift 32 is separate into 5 stage, Leftshift 1, Leftshift 2, Leftshift 4, Leftshift 8 and Leftshift 16 and is controlled via the Sa code.

B, Shift operation



The Op code here should be 000x or 110x, else both Sa and the first MUX will be sent into Leftshift32 the output will be 0

If Op is 000x both Reverse will not be used, input B will go straight to Leftshift 32 and output to C

If Op is 1100 or 1101, B will B reverse -> shift left -> reverse, which is Rightshift ultimately. Therefore, this operation can do both Leftshift and Rightshift using Leftshift32.

Op	Left MUX	Right MUX	Cin	B value	Sa
000x	0	0	0	B	Sa
1100	1	1	0	Reverse B	Sa
1101	1	1	0/1	Reverse B	Sa
Otherwise	Disable	0/1	0	0	0