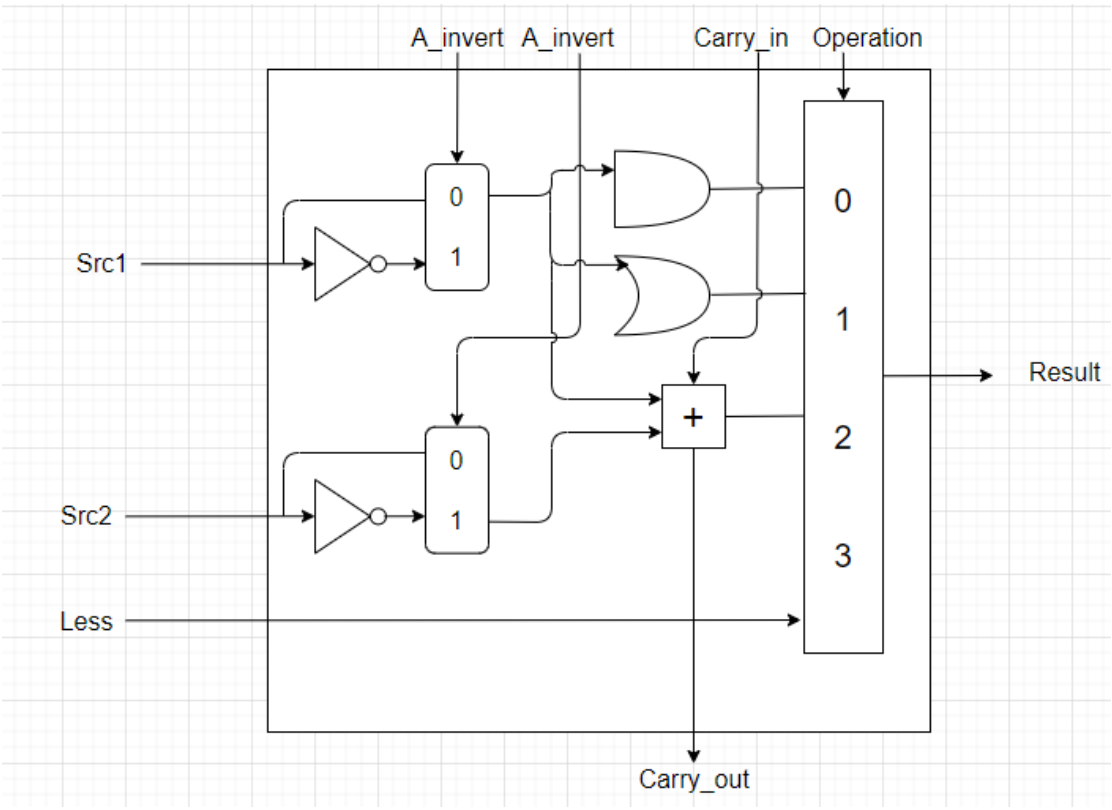
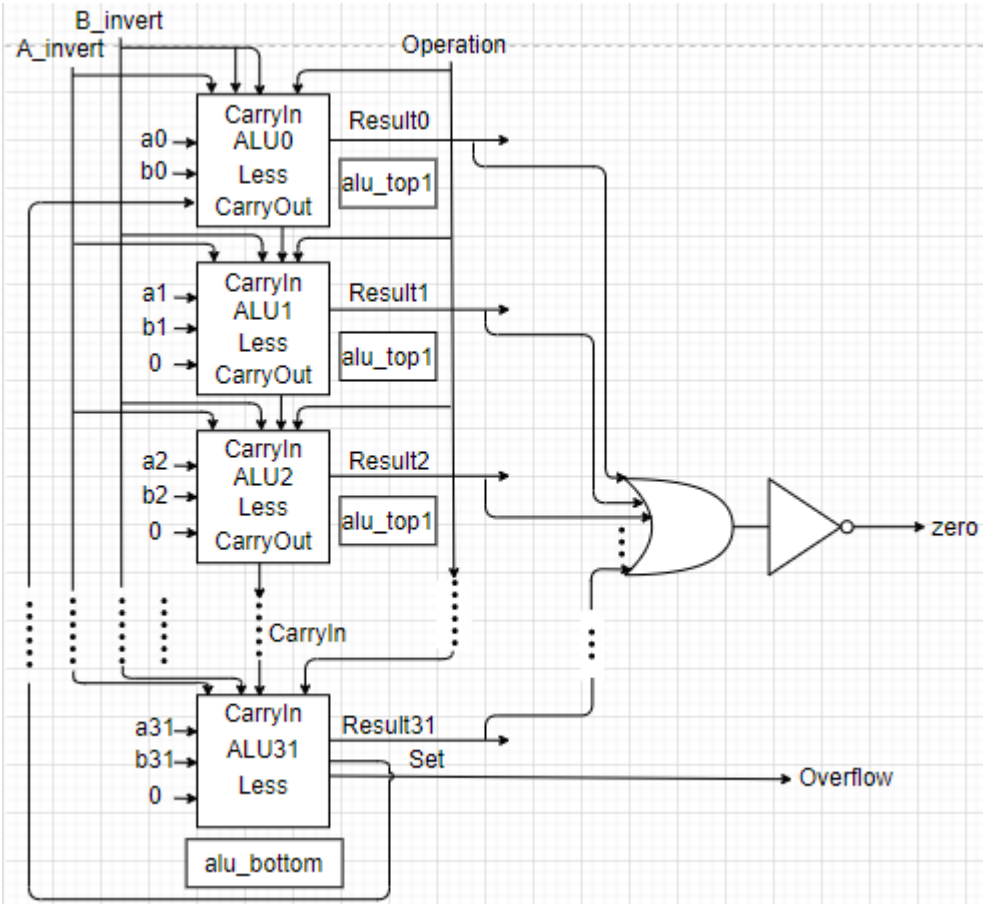
Architecture:





Implementation:

1. AND : src1 and src2 doing bitwise and
2. OR : src1 and src2 doing bitwise or
3. NOR : the invert of src1 and the invert of src2 doing bitwise and
4. NAND : the invert of src1 and the invert of src2 bitwise doing or
5. ADD : add src1 and src2 from the least significant bit(0) to the most significant bit(31), and if the addition of bit have a carry, we let carry be 1 and send it to the next bit addition.
6. SUB : first we invert src2, as we turn a positive number into a negative number, we have to plus one in the invert of src2, so we make the CIN of the first bit be 1, and then apply the same process of the ADD part
7. SLT : to compare src1 and src2, we have to apply subtraction and analyze by the result, if it's negative, we say that src1 is smaller than src2, otherwise it's false. Furthermore, as the carry is more important than the result of every bit, we only send the carry to the next processing bit and ignore the result. At the last bit, we enter a different module called "alu_bottom1", there we use a bit parameter "set" to determine if the result is true or it's only the error of overflow,. After that, we send the value back to the first alu_top to print it in the first bit
8. Zero : if the result is all 0, let the parameter be 1
9. Overflow : At the last bit(31) addition, in ADD part, if src1 and src2 are both positive([31]==0), we let overflow be 1 if result[31] is 1. Meanwhile, if src1 and src2 are both negative([31]==1), we let overflow be 1 if result[31] is 0. While in SUB part, if src1 is positive and src2 is negative, we let overflow be 1 if result[31] is 1, as when src1 is negative and src2 is positive, we let overflow be 1 if result[31] is 0.
10. Cout : if the addition of the last bit has a carry, which want to make result[32] be 1, but in reality result[32] doesn't exist, so we make cout be 1

Commands:

1. iverilog -
   o .\basic.vvp .\testbench.v .\add.v .\alu.v .\alu_top.v .\alu_bottom1.v
2. vvp .\basic.vvp

Problems:

As the lab this time is to build 32 1-bit ALU-top to combine into a fully functioned 32-bit ALU, we need to generate 32 1-bit ALU-top, that is to call the module 32 times, as fool as I am, I totally forgot the existence of the function 'generate', so I just copy the calling module process 32 times, which cause a huge trouble in the latter coding, as every of them have different input and output, I have to revise them one by one, and just a small typo will cause a tragic result. However, I didn't realize there are a function that can safe plenty of my time until I finish the homework.

Moreover, I forgot to put 4'b and 2'b in front of ALU_control and my operation, which almost drive me crazy, causing plenty of odd errors.

Assigning a wire to two different source is also a dumb error that I made, which as well cost me a lot of time.

Lesson learnt:

This time I start doing the homework a little too late, so as the deadline is approaching, it makes my more nervous, causing me unable to find minor bug or simply implant bugs. Although I'm in the former week, I should have started the lab bit by bit to ease the suffering now.