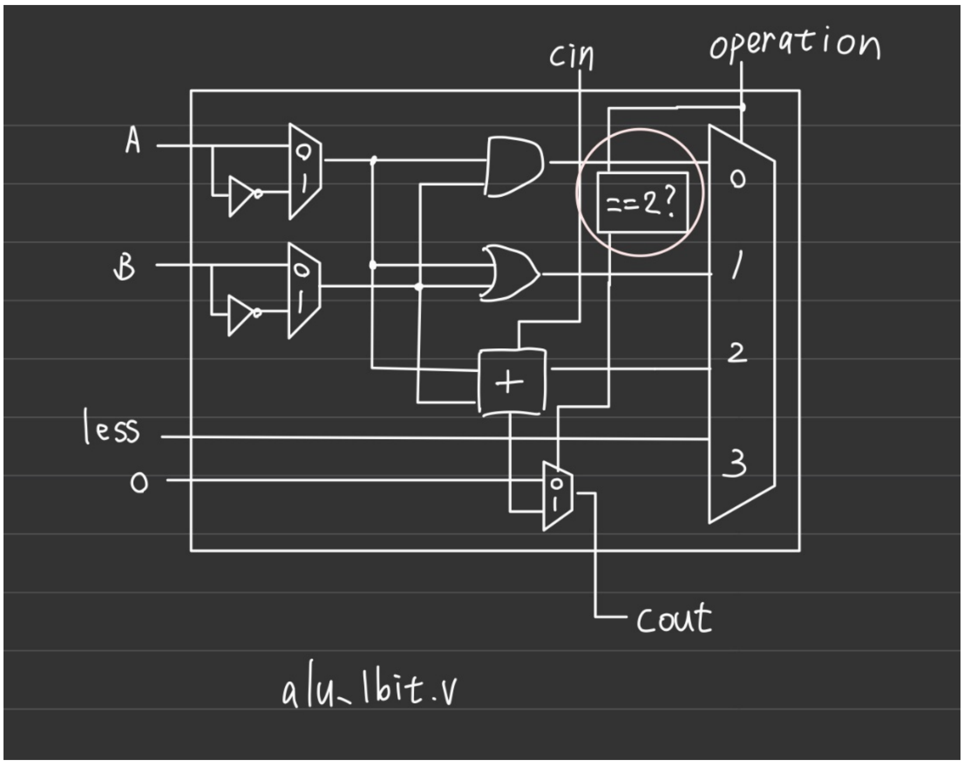
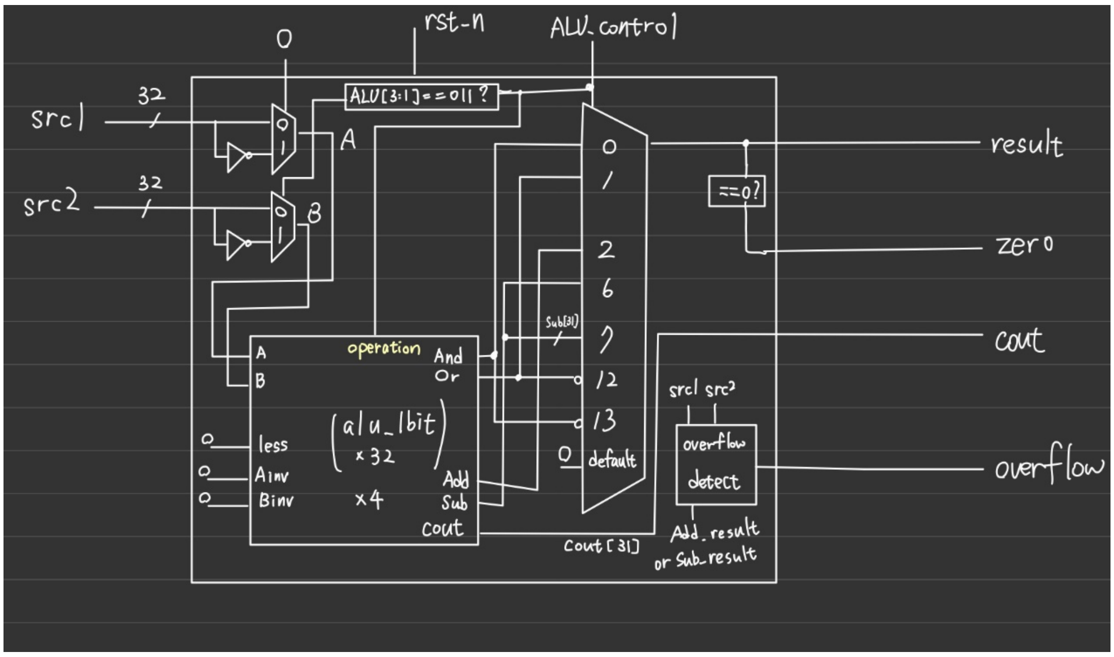
I made some modification about a week after I finished the HW, but I didn't remove the description of my initial verion, I just added a brief explanation for my update on the program.

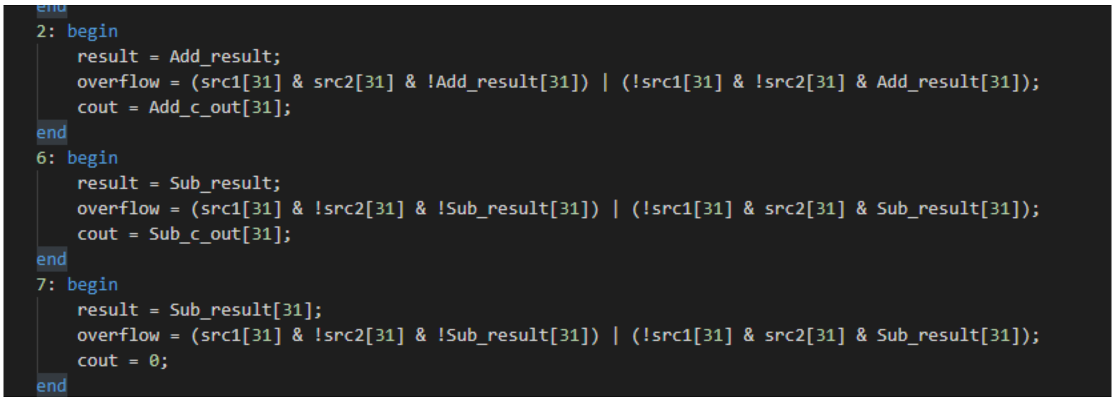
1. My ALU_1bit is designed as in below picture. It is almost the same as that taught in class. Only an additional comparator is added for cout output. I added this because it should output 0 if it was doing and/or/less operation. If it isn't there, our cout might be wrong in some cases.



2. My ALU is designed as in below picture. Some part of it is not explicitly drawn for simplicity. src1 and src2 go through invert gate as in ALU_1bit. And there are four 32bit alu made up of ALU_1bit(128 alu_1bit in total), and all possible calculation are computed when the input changes. After the calculation, a MUX is added to assign the register of the calculation to the result according to the ALU_control

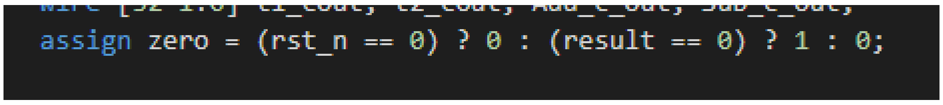


3. Overflow happens in several situations : 1)positive + positive = negative 2)negative + negative = positive 3)positive – negative = negative 4)negative – positive = positive. So the code referring to overflow is :



```
2: begin
    result = Add_result;
    overflow = (src1[31] & src2[31] & !Add_result[31]) | (!src1[31] & !src2[31] & Add_result[31]);
    cout = Add_c_out[31];
end
6: begin
    result = Sub_result;
    overflow = (src1[31] & !src2[31] & !Sub_result[31]) | (!src1[31] & src2[31] & Sub_result[31]);
    cout = Sub_c_out[31];
end
7: begin
    result = Sub_result[31];
    overflow = (src1[31] & !src2[31] & !Sub_result[31]) | (!src1[31] & src2[31] & Sub_result[31]);
    cout = 0;
end
```

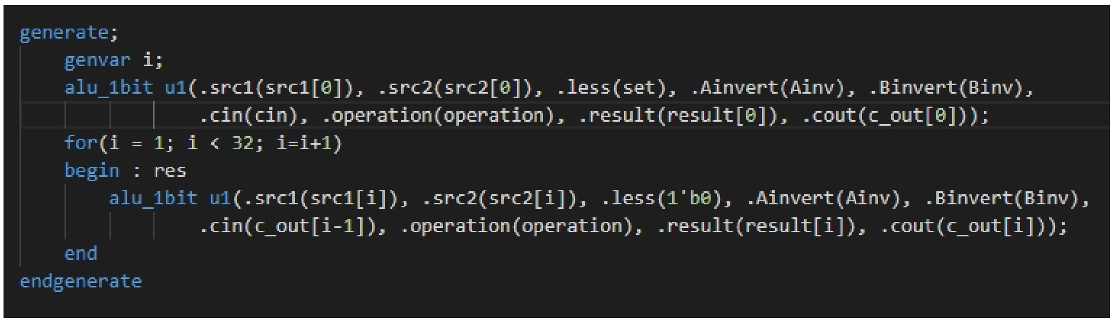
4. cout can be easily determined by the wire I declared in the module c_out. It is simply the 32nd bit of c_out(i.e. c_out[31]);
5. zero is derived by taking nor over every bit in the result, if there is any 1 in result, zero will be 0; otherwise zero will be 1(result = 32'b000..000)



```
assign zero = (rst_n == 0) ? 0 : (result == 0) ? 1 : 0;
```

Second Version :

About a week later, I still wonder why generate for cannot use variables to instantiate, so I tried again by modifying variables according to *ALU_control*. Variables includes 1)operation, 2)Ainv, 3)Binv, 4)cin. They are adjusted to control *alu_1bit* correctly. Therefore, I don't need to store all the results and could just find the answer in one calculation !



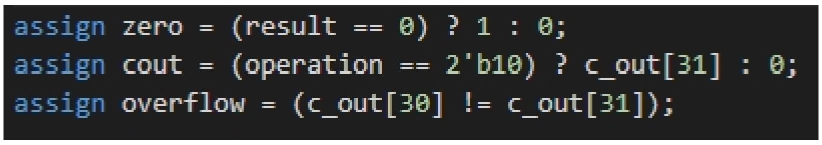
```
generate;
    genvar i;
    alu_1bit u1(.src1(src1[0]), .src2(src2[0]), .less(set), .Ainvert(Ainv), .Binvert(Binv),
                .cin(cin), .operation(operation), .result(result[0]), .cout(c_out[0]));
    for(i = 1; i < 32; i=i+1)
    begin : res
        alu_1bit u1(.src1(src1[i]), .src2(src2[i]), .less(1'b0), .Ainvert(Ainv), .Binvert(Binv),
                .cin(c_out[i-1]), .operation(operation), .result(result[i]), .cout(c_out[i]));
    end
endgenerate
```

The *ZCV* flags are explained below:

Zero : pull high when result == 0;

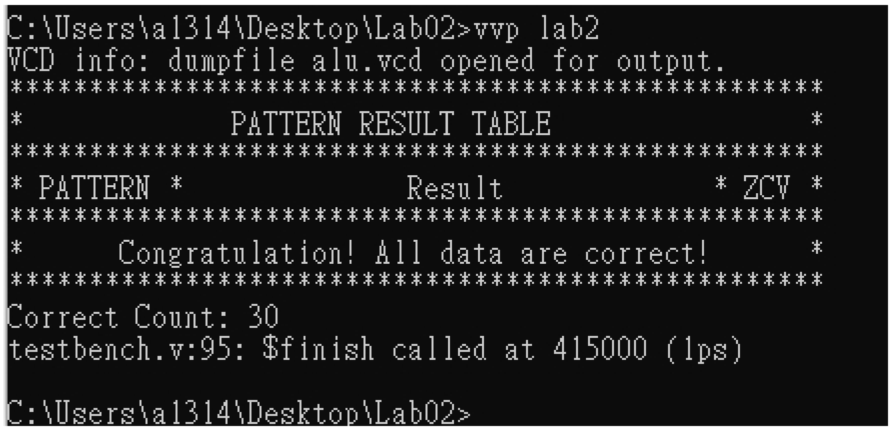
Cout : assign to c_out[31] when doing Addition or Subtraction

oVerflow : pull high when c_out[31] != c_out[30]



```
assign zero = (result == 0) ? 1 : 0;
assign cout = (operation == 2'b10) ? c_out[31] : 0;
assign overflow = (c_out[30] != c_out[31]);
```

Implementation Results:



```
C:\Users\a1314\Desktop\Lab02>vvp lab2
VCD info: dumpfile alu.vcd opened for output.
*****************************************************
*              PATTERN RESULT TABLE                 *
*****************************************************
* PATTERN *              Result             * ZCV  *
*****************************************************
*      Congratulation! All data are correct!        *
*****************************************************
Correct Count: 30
testbench.v:95: $finish called at 415000 (1ps)

C:\Users\a1314\Desktop\Lab02>
```

Problems encountered and solutions

I found some material for how to use generate for and I saw a description said that variables cannot be used to instantiate generate blocks. I wasn't sure what it means so I just took a look at the example and jump into my HW. However, I couldn't use *ALU_control* in the generate block, so I use a *wire* to store *ALU_control* but still bump into error. That's why the first version of my HW is created.

But I tried to use *reg* to store *ALU_control* and the code works as I thought!