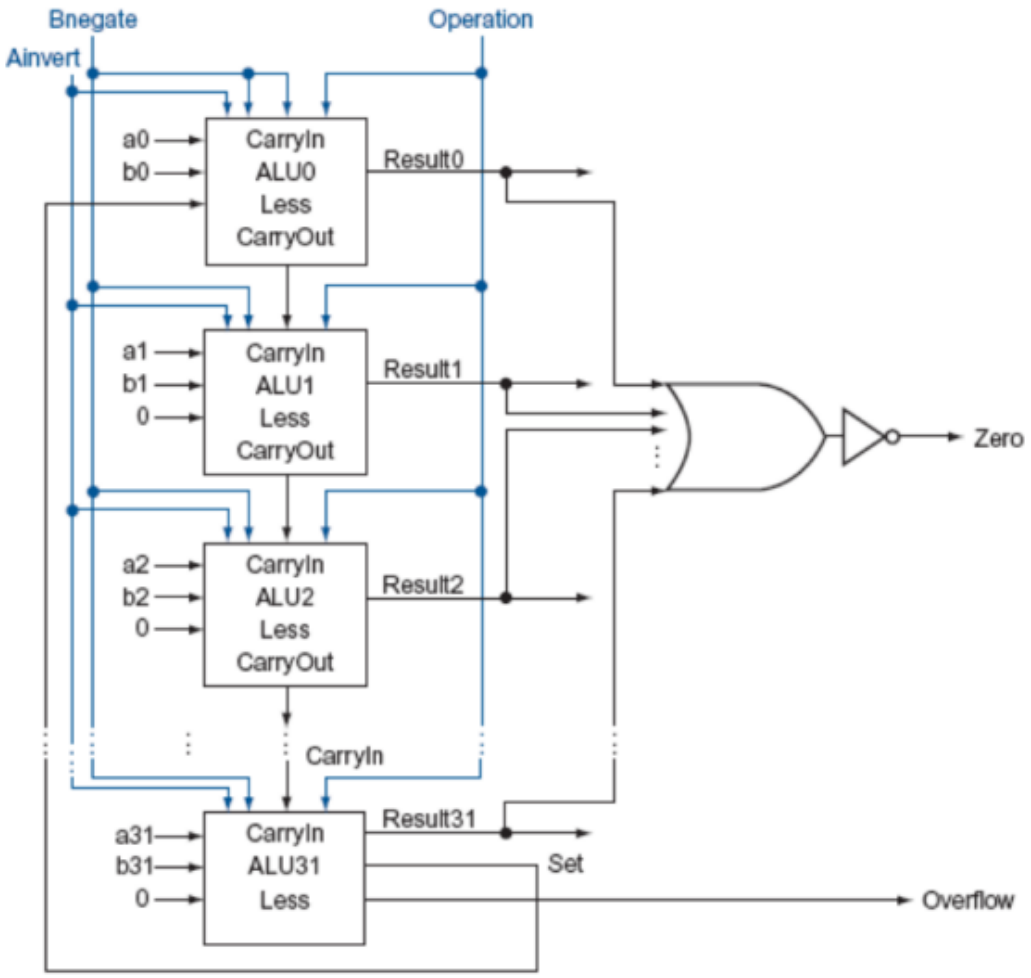
# Computer Organization

# Lab 1: 32-bit ALU

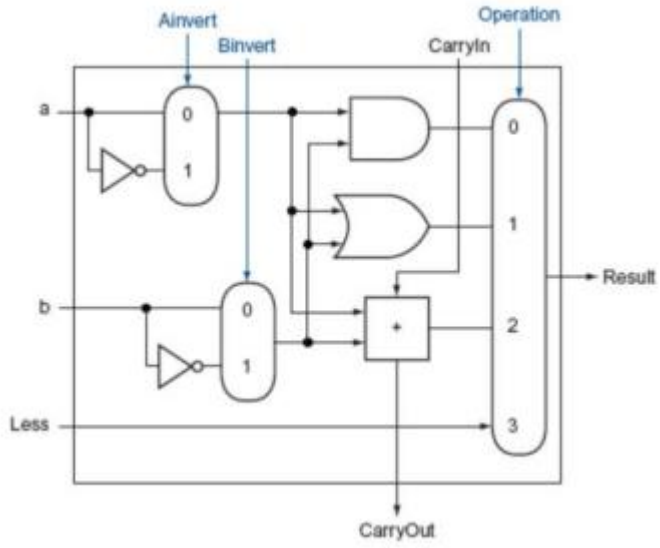
Student ID: 110550063　Name: 張博凱

## 1. Architecture diagrams

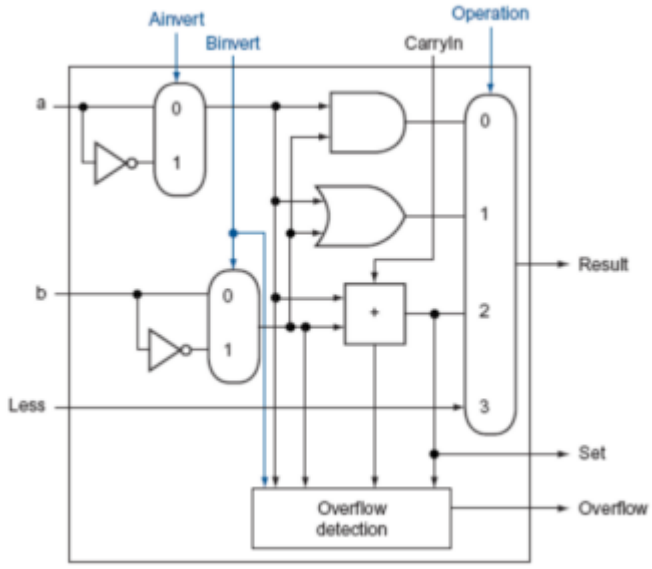
32bit ALU



1bit ALU(bit0 ~ bit30)




1bit ALU(bit31)



## 2. Hardware module analysis

在 alu.v 檔裡，主要的工作就是在 rst_n = 0 時，將所有 output 歸零，在 rst_n = 1 時，將答案輸出到 output，並且寫 31 個 submodule。

```
always@(posedge clk or negedge rst_n)
begin
    if(!rst_n) begin
        result = 0;
        zero = 0;
        cout = 0;
        overflow = 0;
    end
    else begin
        zero = ~(|ALU_result);
        result = ALU_result;
        cout = ALU_cout[31];
        overflow = ALU_overflow;
    end
```

```
alu_top ALU00(.src1(src1[0]), .src2(src2[0]), .less(ALU_set), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]),
alu_top ALU01(.src1(src1[1]), .src2(src2[1]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU02(.src1(src1[2]), .src2(src2[2]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU03(.src1(src1[3]), .src2(src2[3]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU04(.src1(src1[4]), .src2(src2[4]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU05(.src1(src1[5]), .src2(src2[5]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU06(.src1(src1[6]), .src2(src2[6]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU07(.src1(src1[7]), .src2(src2[7]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU08(.src1(src1[8]), .src2(src2[8]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU09(.src1(src1[9]), .src2(src2[9]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .ci
alu_top ALU10(.src1(src1[10]), .src2(src2[10]), .less(1'b0), .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .
```

alu_top.v 檔裡，是 bit0 到 bit30，有 4 個功能，分別是 and、or、add、slt，用 case 實作，剩下的 sub 和 nor 分別可以用 src1 + (-src2)和(~src1) and (~src2) 實作，做運算之所以是輸入^invert，是因為他的真值表結果剛好和 xor 一樣，add 的部分是做 1bit full adder，slt 則是因為要做減法，所以也要計算 cout。

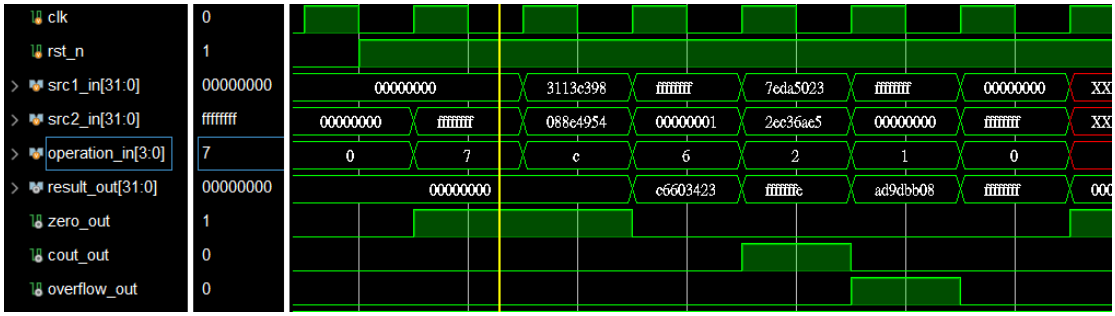
```
2'b00: begin
            result = (src1^A_invert) & (src2^B_invert);
            cout = 0;
        end
2'b01: begin
            result = (src1^A_invert) | (src2^B_invert);
            cout = 0;
        end
2'b10: begin
            result = ((src1^A_invert) ^ (src2^B_invert)) ^ cin;
            cout = (((src1^A_invert) ^ (src2^B_invert)) & cin) + ((src1^A_invert) & (src2^B_invert));
        end
2'b11: begin
            result = less;
            cout = (((src1^A_invert) ^ (src2^B_invert)) & cin) + ((src1^A_invert) & (src2^B_invert));
```

alu_top31 檔，是 bit31，和前 31 個 bit 一樣，只是多了 2 個 output，分別是 set 和 overflow，set 是計算第 31bit 的加法，確定是否為負數，再連到 bit0 的 less 裡做輸出，overflow 則是看第 31bit 的 cin 和 cout 是否相同，並且只需要在 add 的功能裡確認就行，其他時候都是為 0。

```
            set = 0;
            overflow = 0;
        end
2'b10: begin
            result = ((src1^A_invert) ^ (src2^B_invert)) ^ cin;
            cout = (((src1^A_invert) ^ (src2^B_invert)) & cin) + ((src1^A_invert) & (src2^B_invert));
            set = 0;
            overflow = cin ^ cout;
        end
2'b11: begin
            result = less;
            cout = 0;
            set = ((src1^A_invert) ^ (src2^B_invert)) ^ cin;
            overflow = 0;
        end
```

## 3. Experimental result

這裡用新的測資來觀察結果，因為在 alu 裡面是在 positive edge 時才會改變 output，因此 result_out 的部分都要觀察下一個 clock 才是正確的結果。
operation = 7，也就是 slt，可以看出 src1 < src2，因此輸出為 0，zcv = 100。
operation = c，也就是 nor，可以發現答案是正確的，且 zcv = 000。
operation = 6，也就是 sub，ffffffff – 00000001 = ffffffff + fffffff = (1)fffffffe，可以發現答案是正確的，並且有 carry out，zcv = 010。
operation = 2，也就是 add，可以發現這組測資會 overflow，因此 zcv = 001。
operation = 1，也就是 or，all 1 or all 0 的答案是 all 1，且 zcv = 000。
operation = 0，也就是 and，all 0 and all 1 的答案是 0，且 zcv = 100。




## 4. Problems you met and solutions

在實作 slt 的功能時，我一直在思考第 31bit 是否有辦法使用和前面其他 1bit alu 一樣的 module，但是一直找不到方法實作，因為前面的 alu 都只有兩個輸

出，但是我想的 slt 都沒辦法只用兩個 output 做出來，所以最後還是妥協多寫一個 1bit alu 給第 31bit 使用。

## 5. Summary

這次的 lab 讓我好好地複習 verilog 的語法，雖然上個學期有使用，但到了這學期都忘得差不多了，不過這次的 lab 寫了使用了很多基礎的語法，也算是把記憶都差不多找回來了，非常的實用。