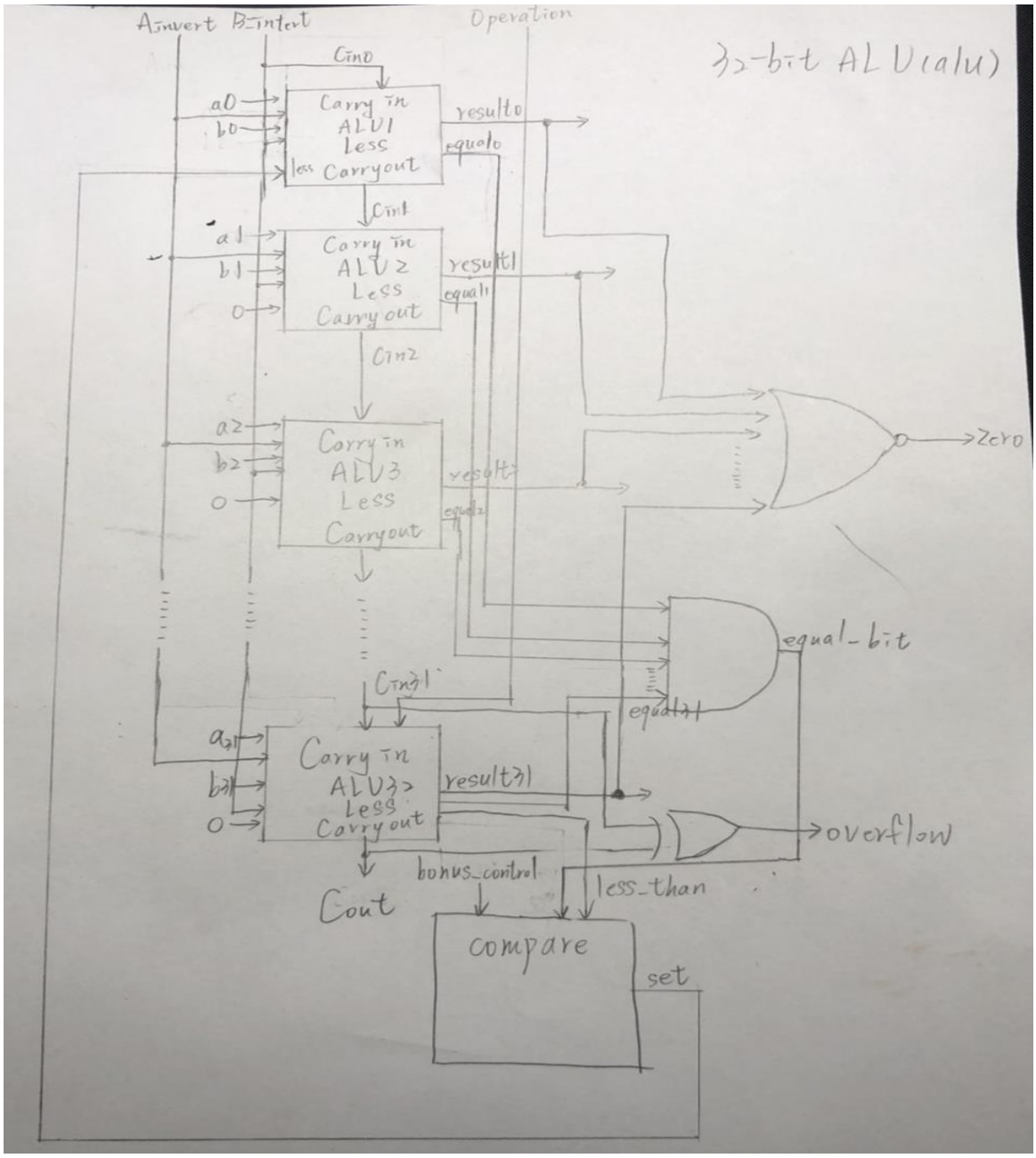
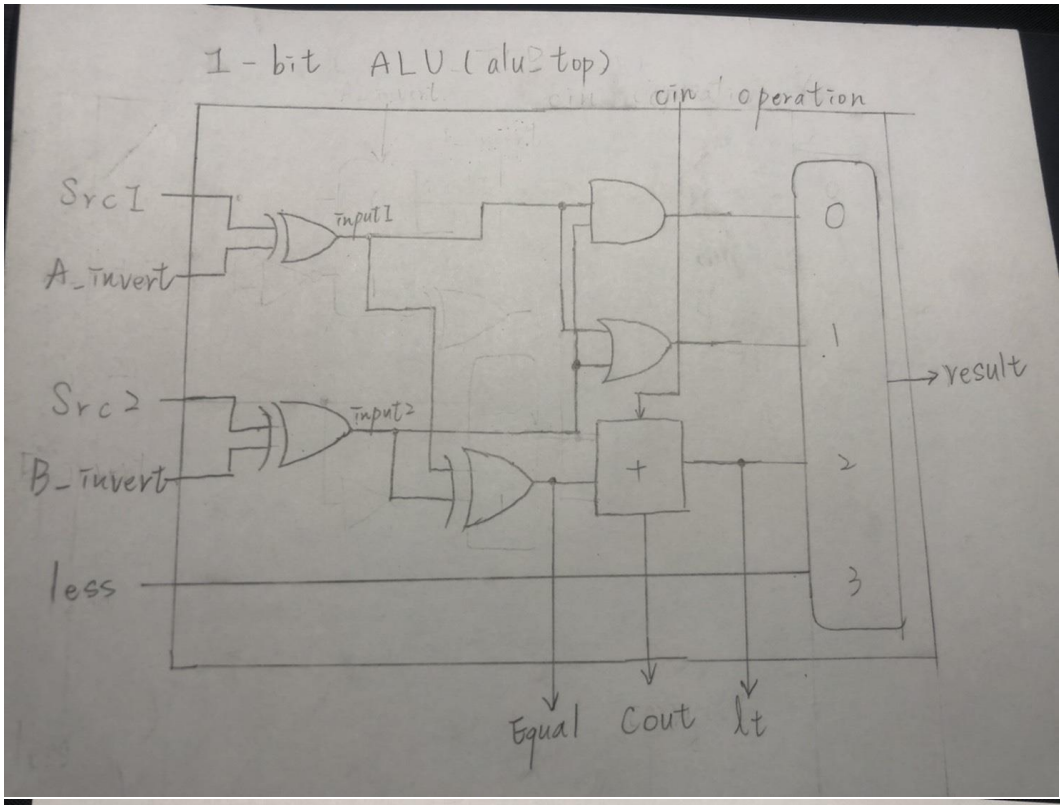
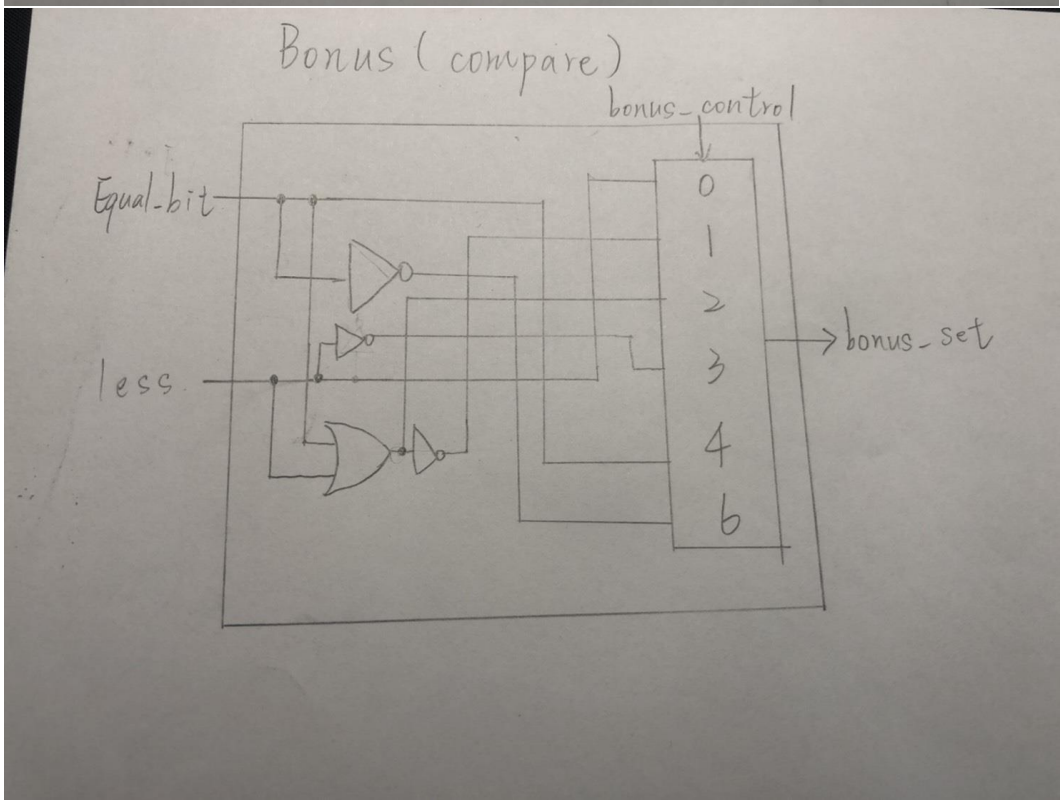
Computer Organization Lab 1: 32bit ALU

0716033 周俊毅

1. Architecture diagram，以下是我設計的 32bit ALU 的 Architecture diagram 分別是 32bit ALU, 1bit ALU, Bonus







2. 先從最基本的 1bit ALU 開始 input 有 src1 的一個 bit, src2 的一個 bit, less, A_invert:ALU_control[3], B_invert: ALU_control[2], cin, operation , output 有 1bit result cout, equal: 確認每個 bit 是否一樣, lt:將數字相減結果輸出(只會用於第 32 個 ALU)

下面是 code 的說明

```
wire input1, input2, And, Or, Add;

//將src1和src2分別跟a_invert和b_invert做XOR可以知道直要不要做反向
//像如果要做NAND, NOR, SUB, 都需要加個not所以可以運用abinvert來直接讓src1跟src2做反向
//e.g:A nand B =~A or ~B
assign input1=src1^A_invert;
assign input2=src2^B_invert;

//equal 是要檢查兩個bit是否一樣如果一樣的話就output→1, 因為A_invert→0, B_invert→1
//, 所以當scr1跟src2的bit一樣的話要用xor來得到 1, 因為src2會xorB_invert來得到相反的值
//所以最後input1跟input2要不一樣才是原本一樣, 所以用xor
assign equal=input1^input2;

//A and B
//A nor B = ~A and ~B 因為A nor B可以這樣表示所以就寫在一起
assign And=input1&input2;

//A or B
//A nand B =~A or ~B 因為A nand B可以這樣表示所以就寫在一起
assign Or=input1|input2;

//A add B= A xor B xor cin (因為A xor B以經算過了所以替換成算過的equal)
//cout如同底下的code一樣只要三個變數中有其中兩個是1, cout就會是1
assign Add=cin^equal;
assign cout=(input1&input2)|(input1&cin)|(input2&cin);

//最後因為要算slt, 所以我需要將最後一個bit鄉檢結果來看是正數還是負數
assign lt=Add;

//最後就是決定result 的輸出結果, 根據operation的輸入結果來決定
//00:and/nor 01:or/nand 10:add/sub 11:less
always@( * )
    begin
        case(operation)
            2'b00:result=And;
            2'b01:result=Or;
            2'b10:result=Add;
            2'b11:result=less;
        endcase
end
```

接下來是 32bit ALU, 主要架構是 32 個 1 bit ALU 和一個 compare，來做出 bonus，比較特別是第一個 ALU 的 less 值是經過 compare 出來的值，其他都是 0 因為最後出 result=00……..001 這樣，所以其他 ALU less 都是 0，因為要檢查每個 bit 是否一樣所以要多輸出一個 equal

```
//ALU_control 可以拆解成 A_invert B_invert 跟最後的operation
//因為ALU_control 的第一二個值，就是決定src1跟src2要不要not
//而決定slt的話需要將src1-src2的值來看大小, src1是否小於src2

output [32-1:0] result;
output          zero;
output          cout;
output          overflow;

reg    [32-1:0] result;
reg             zero;
reg             cout;
reg             overflow;

//ov: overflow, rs: 32 bit result , c: 32 bit carry
//equal: 檢查32個位置有沒有一樣, equal_bit: 檢查所有equal32bit是不是都一樣
//less_than: src1是否小於src2, set: 經過compare出來的less值, lt: 第32bit相減後的結果
wire ov;
wire [32-1:0] rs;
wire [33-1:0] c;
wire [32-1:0] equal;
wire equal_bit;
wire less_than, set, lt;

//因為要把減法變成加法，所以原本是A-B, 會變成A+(-B)
//所做的方式就是將B變成他的2's component
//也就是將B xor B_invert + 1, B_invert就是alu_control[2]
assign c[0]=ALU_control[2];

//down is alu 32, made up of 32 alu_top
//將32bit src1和src2 放入32個Alu_top
//那less只有第一個bit是將第32個ALU所計算出來lt來知道以及後來的compare來確定值是多少
//A_invert B_invert就是ALU_control的[3][2]來決定src1跟2要不要invert
//cin就是個別放入上一個ALU所計算出來的cout, 最後一個ALU的cout就是ZCV的C值
//equal就是來確認每個bit是否一樣, 一樣是1, 不一樣是0
alu_top a1(
    .src1(src1[0]),
    .src2(src2[0]),
    .less(set),
    .A_invert(ALU_control[3]),
    .B_invert(ALU_control[2]),
    .cin(c[0]),
    .operation(ALU_control[1:0]),
    .result(rs[0]),
    .cout(c[1]),
    .equal(equal[0])
);
```

下一個是後面的關於輸出以及 bonus 的部分

```
//overflow的計算方式就是檢查最後一個bit的cin跟cout是不是一樣如果不一樣的話就是overflow
//最後一個bit代表這個數字的sign
//會overflow只有兩種方式一個是正數加正數但結果是負數也就是0+0+1=1, cin=1, cout=0
//, 這樣最後一個bit就是1 overflow
//負數加負數 cin=0, 1+1+0=0, cout=1 這樣cin≠cout, overflow
assign ov=c[31]^c[32];

//因為lt只是最後一個bit+cin相加之後的結果所以有可能會因為overflow出來結果是錯的
//解決方法就是看兩個數是不是同號，因為相減會overflow只有可能是正減負以及負減正
//如果同號只要看被減數(src1)的sign就知道結果，而同號就還是要看lt的結果來判斷
assign less_than=equal[31] ? lt : src1[31];

//equal_bit將所有bit and起來只要有一個不一樣也就是0, equal_bit就會是0
and and1(   equal_bit, equal[0], equal[1], equal[2], equal[3], equal[4],
            equal[5], equal[6], equal[7], equal[8], equal[9], equal[10],
            equal[11], equal[12], equal[13], equal[14], equal[15], equal[16]
            , equal[17], equal[18], equal[19], equal[20], equal[21], equal[22]
            , equal[23], equal[24], equal[25], equal[26], equal[27], equal[28]
            , equal[29], equal[30], equal[31]
        );

//因為要比較大小以及等於所以要將equal_bit 跟 less_than放入 compare 裡
//而bonus_control是決定要做出哪種比較的，所以也要放入
//output是set，最後set就是會接到第一個ALU的less來表示結果
compare c1(
    .bonus_control(bonus_control),
    .equal_bit(equal_bit),
    .less_than(less_than),
    .bonus_set(set)
);
//最後就是輸出的部分
//result就是rs
//zero則是把每一個bit nor在一起來知道是不是全部都是0
//cout則是最後一個bit的cout值
//overflow就是ov
always @(*) begin
    if (rst_n) begin
        result <= rs;
        zero <= ~|rs;
        cout <= c[32];
        overflow <= ov;
    end
end
```

再來是 compare 的部分，也就是 bonus 的部分 input equal_bit: 每個 bit 是否都依樣
Less_than :有沒有小於

```
input [3-1:0]   bonus_control;
input           equal_bit;
input           less_than;

output reg      bonus_set;

wire slt, sgt, sle, sge, seq, sne;
//因為所有的結果都可以透過less_than 跟 equal來表示
//想是set_greater_than 就是要大於，也就是要檢查不能小於也不能等於
//其他也都可以用這樣來表示
assign slt=less_than;
assign sgt=~(less_than|equal_bit);
assign sle=less_than|equal_bit;
assign sge=~less_than;
assign seq=equal_bit;
assign sne=~equal_bit;

//最後就是依照bonus_control來決定是輸出何種結果
//我是用case來寫
always @(*)
begin
    case (bonus_control)
        3'b000: bonus_set <= slt;
        3'b001: bonus_set <= sgt;
        3'b010: bonus_set <= sle;
        3'b011: bonus_set <= sge;
        3'b110: bonus_set <= seq;
        3'b100: bonus_set <= sne;
    endcase
end
```

3. Commands for executing your source codes:
   iverilog -o bonus.vvp testbench.v alu.v alu_top.v compare.v
   vvp bonus.vvp
4. Problems encountered and solutions:
   我遇到最大的問題在於搞懂 verilog 這個語言其它到是還好，還有在用 test 的時候用了沒有把註解刪掉，其它都沒什麼問題。