The Report of Lab2

109550025 謝翔丞

- Detailed description of the implementation
  1. mux2to1.v and mux4to1.v:
     a. 這部份我是在一個always裡面使用”case”,case要放的東西很簡單,就是輸入的select,根據select是1`b0或是1`b1選擇要將src1還是src2給result。
     同理mux4to1也是一樣的操作。

```
always @(*) begin
    // $display("mux2 ",src1,src2,select);
    case(select)
        1'b0:
            result <=src1;
        1'b1:
            result <=src2;
    endcase
    // $display("res %d",result);
end
```

     b. 同理mux4to1也是一樣的操作。

```
always @(*) begin
    // $display("mux4 ",src1,src2,src3,src4,select);

    case(select)
        2'b00:
            result<=src1;
        2'b01:
            result<=src2;
        2'b10:
            result<=src3;
        2'b11:
            result<=src4;
    endcase
end
```

  2. alu_1bit.v
     a. 這部份我先設立一些wire，因為wire可以assign 對我之後的一些操作比較方便。然後將src1和~src1以及Ainvert丟入m1(mux2to1)去做第一次選擇，並同樣將src2放入m2(mux2to1)取得第二次選擇的結果，而這兩個選擇器的結果別給到a_res和b_res。
     b. 接著，我先算出四種可能運算的值，分別給到一個對應的wire，其中，add因為有可能有carry out 所以我宣告{carry,add_res} = a_res+b_res+cin;這樣他就會將第0bit丟給add_res,然後將第1bit丟給carry。
     c. 接著把4個可能結果丟入mux4to1去根據opearation選出最後result。
     d. 最後就是在always裡面根據opearation，利用case來選擇我的cout要直接給0還是add和less的carry運算結果。
     e. 圖片在下頁

```
wire a_res,b_res,and_res,or_res,mux_end;
wire not_a,not_b,carry,add_res;

assign not_a = ~src1;
assign not_b = ~src2;

MUX2to1 m1(.src1(src1),.src2(not_a),.select(Ainvert),.result(a_res));
MUX2to1 m2(.src1(src2),.src2(not_b),.select(Binvert),.result(b_res));
assign and_res = (a_res & b_res);
assign or_res = (a_res | b_res);
assign {carry,add_res} = a_res + b_res+cin;


MUX4to1 m3(.src1(and_res),.src2(or_res),.src3(add_res),.src4(less),.select(operation),.result(mux_end));

always @(*) begin
    case(operation)
        2'b00:
            cout<=0;
        2'b01:
            cout<=0;
        2'b10:
            cout<=carry;
        2'b11:
            cout<=carry;
    endcase
    result <= mux_end;

end


endmodule
```

3. alu.v
    a. 這部份很有趣，為了不要直接複製32行，我選擇用一個for迴圈來執行，每次丟進去第i個bit，並且將答案輸出到一個同樣為32bit的暫存器（mux_end），cout的部份就是存到另一個32bit的暫存器（tmp_cout_reg），而每一次的cin都是第i-1個tmp_cout_reg，因為下一次的cout就是前一次的cin。
    b. 講到這裡可能會友人好奇，那第一個alu不是要給set嗎？是怎麼算的？其實道理很簡單，我assign {non , set} = (src1[31] + ~src2[31] + tmp_cout_reg[32-2]) 這樣就可以得到最高位的結果，也就是set的定義。
    c. 最後在always裡面，我是設定如果（~rst_n）就將一切值規零，若否則不斷更新result值為mux_end，就是我前面一直用來得到每個bit答案的暫存器，zero就是將全部答案bit or起來再not 所以我直接用bitwise or，~| 來處理;cout則是如果在add運算下 則給tmp_cout_reg[31];而overflow的判定就是cout的最高兩位做xor（當然也是在add用到下才需要算）。
    d. 以下是code
    e.

```
assign  {non , set}  = (src1[31] + ~src2[31] + tmp_cout_reg[32-2]) ;


// alu_1bit set_init(.src1(src1[32-1]) ,.src2(src2[32-1]), .less(1'b0) ,.Ainvert(Ainvert) , .Binvert(Binvert)

generate
    for(i=0;i<32;i=i+1)begin
        if( i == 0)begin
            alu_1bit a1(.src1(src1[i]) ,.src2(src2[i]), .less(set) ,.Ainvert(Ainvert) , .Binvert(Binvert), .c
        end
        else begin
            alu_1bit a1(.src1(src1[i]) ,.src2(src2[i]), .less(1'b0) ,.Ainvert(Ainvert) , .Binvert(Binvert), .
        end
    end

endgenerate


always @(*) begin
    if(~rst_n)begin
        result<=0;
        zero<=0;
        cout<=0;
    end
    else begin

        result <= mux_end;
        zero <= ~(|result);
        cout <= tmp_cout_reg[32-1] && operation == 2'b10;
        overflow <= ^tmp_cout_reg[32-1:32-2] && operation == 2'b10;
    end
end
```

f.
```
.Ainvert(Ainvert) , .Binvert(Binvert), .cin(cin) ,.operation(operation) , .result(mux_end[i]) ,.cout(tmp_cout_reg[i]

,.Ainvert(Ainvert) , .Binvert(Binvert), .cin(tmp_cout_reg[i-1]) ,.operation(operation) , .result(mux_end[i]) ,.cout(
```

g.
```
wire set;
wire [32-1:0] mux_end;
wire cin,non;
wire [32-1:0] tmp_cout_reg;
reg set_reg;
wire Ainvert,Binvert;
wire [2-1:0] operation;
assign operation = ALU_control[1:0];
assign cin = ALU_control[2];
assign Ainvert = ALU_control[3];
assign Binvert = ALU_control[2];

genvar  i,j;
```

- Implementation results
  1. alu_1bit.v

a.
```
hsianchengfun@hsianchengfun-Swift:~/110second/ComputerOrg/Lab2/Lab02$ vvp 1bit
VCD info: dumpfile alu_1bit.vcd opened for output.
a 1 b 1 ope 0
sum 1
carry 0
===============
a 1 b 1 ope 1
sum 1
carry 0
===============
a 1 b 1 ope 3
sum 0
carry 0
===============
```

  2. alu.v

a.
```
hsianchengfun@hsianchengfun-Swift:~/110second/ComputerOrg/Lab2/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
```

- Problems encountered and solutions
  1. 這次的lab算好處理，因為助教有提供alu的電路圖，這樣在思考上會更加清晰、有很大的幫助。再加上之前有修dlab，所以沒遇到甚麼大問題，除了一開始打算多用一個alu_1bit來算出set時，因為operation給錯所以一直有三筆測資跑不過，幸好後來發現要給add，答案就對了。（不過最後還是改用直接assign 來得到set ）