Lab02-32 Bit ALU REPORT

109550164 徐聖哲

1.Code explain

MUX2to1

- • In MUX2to1, I use a case to choose which src to put in the result.
- When select is equal to 0, select src1
- When select is equal to 1, select src2
- I set default as result

```
/* Write your code HERE */
always @(src1 or src2 or select) begin
    case (select)
        1'b0:
            result = src1;
        1'b1:
            result = src2;
        default:
            result = result;
    endcase
end

endmodule
```

MUX4to1

- When select is equal to 00, select src1
  When select is equal to 01, select src2
  When select is equal to 10, select src3
  When select is equal to 11, select src4

```
/* Write your code HERE */
always @(*) begin
    case (select)
        2'b00:
            result = src1;
        2'b01:
            result = src2;
        2'b10:
            result = src3;
        2'b11:
            result = src4;
        default:
            result = result;
    endcase
end

endmodule
```

ALU_1bit

- In ALU_1bit, I use two MUX2to1 and one MUX4to1 to implement a one bit alu
- The two MUX2to1 is used to select a and b using Ainvert and Binvert to select.
- Then set wire “and_o”, “or_o”, “add_o” as MUX4to1’s src1, src2, src3
  and use boolean expression in slide to illustrate
- MUX4to1 use operation code to select right output for result
- In always block, I let set = sum_o for furthur use and calulate cout

```
/* Write your code HERE */
wire and_o, or_o, sum_o;
MUX2to1 MUX1 (.result(result1), .src1(src1), .src2(~src1), .select(Ainvert));
MUX2to1 MUX2 (.result(result2), .src1(src2), .src2(~src2), .select(Binvert));
MUX4to1 MUX3 (.result(result3), .src1(and_o), .src2(or_o), .src3(sum_o), .src4(less), .select(operation));
assign and_o = result1 & result2;
assign or_o = result1 | result2;
assign sum_o = result1 ^ result2 ^ cin;

always @(*) begin
    set = sum_o;
    cout = (result1 & result2) | (result1 & cin) | (result2 & cin);
    result = result3;
end

endmodule
```

ALU

- In ALU, I use generate with for loop to implement 32 bit alu
- In generate, since the top alu’s less need to be set of the bottom alu, so I use if to deal with it
  other 31 alu use for to construct
- since opecode is list in labw soec, so I set alu_control[3] as A_invert and alu_control[2] as B_invert, which can represent sub, slt, nor and nand
- I setup wire carry_out for cout, carry_in for cin , set for set and res for result
- In always block, I set result as res, because result is register
  zero is 1 if res is equal to 32’b0
  cout is 1 if cout of the bottom alu is 1 when function code is 0010(add) or

0110(subtract)
overflow is 1 if the MSB of cout is not equal to MSB of cin when function code is 0010(add) or 0110(subtract)

- the comment code below is used to display value in the console to check the error

```
/* write your code HERE */
wire [32-1: 0] carry_out;
wire [33-1: 0] carry_in;
wire [32-1: 0] set;
wire [32-1: 0] res;
assign carry_in[0] = ALU_control[2];
assign carry_in[31:1] = carry_out[30:0];

genvar  i;
generate
    for(i = 0; i < 32; i=i+1) begin : label
        if (i == 0)
            alu_1bit alu(.src1(src1[i]), .src2(src2[i]), .less(set[31]), .Ainvert(ALU_control[3]), .Binvert(ALU_control[2]), .cin(carry_in[i]), .operation(ALU_control[1:0]), .result(res[i]), .cout(carry_out[i]), .set(set[i]));
        else
            alu_1bit alu(.src1(src1[i]), .src2(src2[i]), .less(1'b0), .Ainvert(ALU_control[3]), .Binvert(ALU_control[2]), .cin(carry_in[i]), .operation(ALU_control[1:0]), .result(res[i]), .cout(carry_out[i]), .set(set[i]));
    end
endgenerate

always @(*) begin
    result = res;
    zero = res == 32'b0;
    cout = carry_out[31] & ((ALU_control == 4'b0010) | (ALU_control == 4'b0110));
    overflow = (carry_out[31] ^ carry_in[31]) & ((ALU_control == 4'b0010) | (ALU_control == 4'b0110));
end

// always @(*) begin
//  if (ALU_control[3:0]==7|ALU_control[3:0]==7) begin
//      #1;
//      $display(" fuck %h, %h, %h, %h, %h, %h", src1[31:0], src2[31:0], carry_in[31:0], carry_out[31:0], res[31:0], set[31:0]);
//  end
// end

endmodule
```

2.Implementation results

```
PS C:\Users\danzel\徐聖哲\課程\大二下\計算機組織\Lab02_test> iverilog -o lab2.vvp alu.v alu_1bit.v testbench.v MUX*
PS C:\Users\danzel\徐聖哲\課程\大二下\計算機組織\Lab02_test> vvp lab2.vvp
VCD info: dumpfile alu.vcd opened for output.
**************************************************
*             PATTERN RESULT TABLE               *
**************************************************
* PATTERN *              Result             * ZCV *
**************************************************
*      Congratulation! All data are correct!     *
**************************************************
Correct Count: 30
```

3.Problems encountered and solution

Compared to lab1, I think lab2 is not that hard, but still needs some hard work to finish it.

First, I used to study digital experiments last semester, so I am used to clk. But this time, I don't have clk. I am not in over my hand initially. But, I figure out after read the slide again.

Secondly, the mux is quite easy but when it comes to merge 32 alu, things become complicated. It takes me some time to find out that I have to use wire to store

temporary value. For example carry_in and carry_out, I can easily use them to calculate cout and overflow.

Third, the hint in the last page of the slide is very useful. I will create 32 modules separately if I dont search genvar first. I think generating is quite powerful. I can drill with a repeat module in a block, but it seems that the display doesn't work. So I use it on another always block, to check my error

Finally, I spent about two hour to solve the error of subtracting, but I found the error in alu_1bit.v. I missed writing result1 and result2 as src1 and src2 in L26. I am so stupid haha.