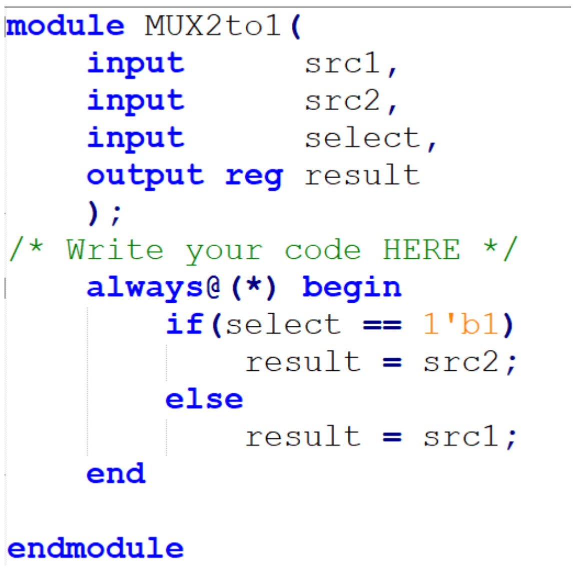
Computer Organization LAB2 Report

109550168 林慧旻

1. Description of the Implementation

(1)MUX2to1.v

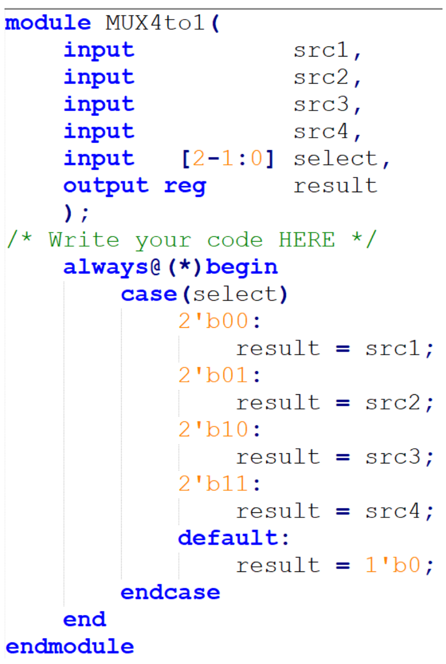


```
module MUX2to1(
    input       src1,
    input       src2,
    input       select,
    output reg  result
    );
/* Write your code HERE */
    always@(*) begin
        if(select == 1'b1)
            result = src2;
        else
            result = src1;
    end

endmodule
```

If select equals 1, then let result be src2; else, let result be src1.

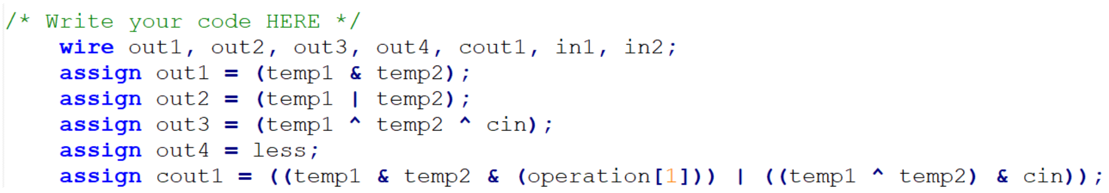
(2)MUX4to1.v



```
module MUX4to1(
    input           src1,
    input           src2,
    input           src3,
    input           src4,
    input   [2-1:0] select,
    output reg      result
    );
/* 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 = 1'b0;
        endcase
    end
endmodule
```

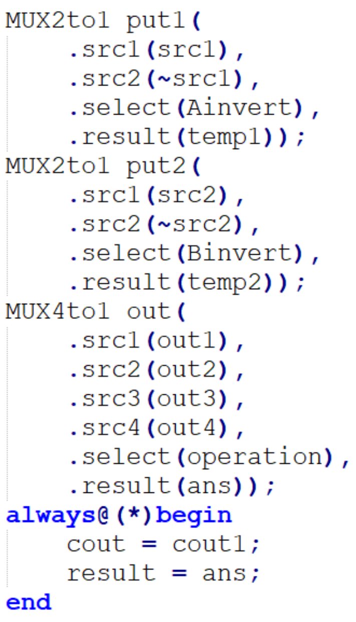
In fact, there is no need to set up a default value for the MUX, but I still set one to be one here. Let result be src1 if select is 0, be src2 if select is 1, be src3 if select is 2, and be src3 if select is 3.

(3)alu_1bit.v



```
/* Write your code HERE */
    wire out1, out2, out3, out4, cout1, in1, in2;
    assign out1 = (temp1 & temp2);
    assign out2 = (temp1 | temp2);
    assign out3 = (temp1 ^ temp2 ^ cin);
    assign out4 = less;
    assign cout1 = ((temp1 & temp2 & (operation[1])) | ((temp1 ^ temp2) & cin));
```

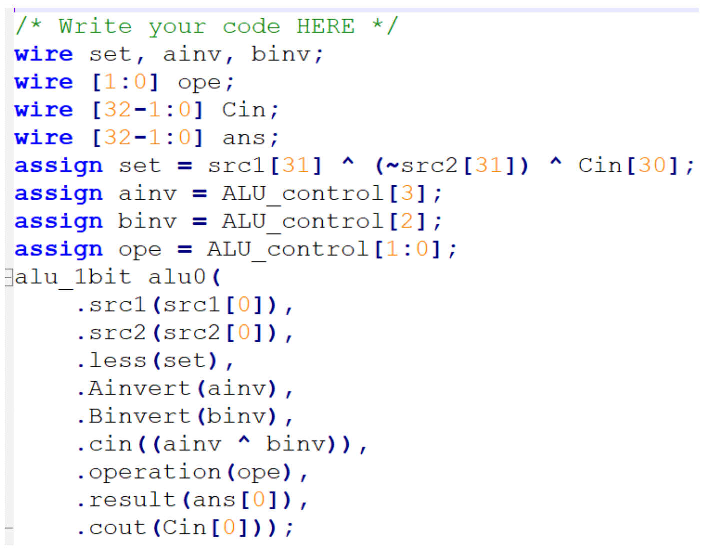
I firstly set up some wires that will be used in the following code. Later, let out1, out2, out3, out4 be the results of and, or, add and subtract, and less. As for the wire cout1, I set it to be as the upper code, which will be discussed in the next paragraph.



```
MUX2to1 put1(
    .src1(src1),
    .src2(~src1),
    .select(Ainvert),
    .result(temp1));
MUX2to1 put2(
    .src1(src2),
    .src2(~src2),
    .select(Binvert),
    .result(temp2));
MUX4to1 out(
    .src1(out1),
    .src2(out2),
    .src3(out3),
    .src4(out4),
    .select(operation),
    .result(ans));
always@(*)begin
    cout = cout1;
    result = ans;
end
```

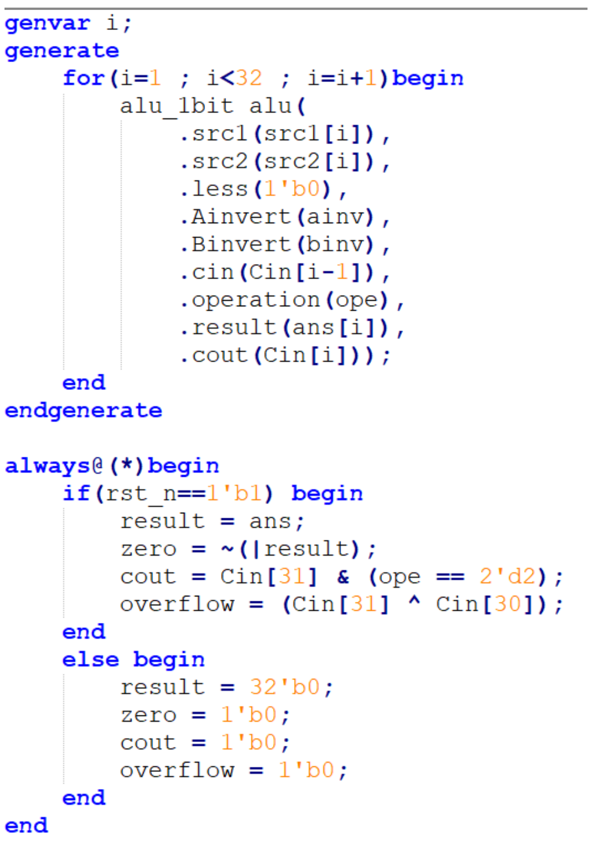
There are 3 multiplexers here. Two are two to one, which are used to choose a negative value of a and b or not, giving the final values temp1 and temp2. The other is four to one MUX. It is used for choosing the final result for the 1 bit alu. Finally, in the always block, cout register will be connected with cout1 wire, which is set up to be: when temp1 and temp2 are both one, and the operation is subtract, add, or slt, then cout is one; or, if temp1 or temp2 is one, and the cin input is one too, then set cout to be one.

(4)alu.v



```
/* Write your code HERE */
wire set, ainv, binv;
wire [1:0] ope;
wire [32-1:0] Cin;
wire [32-1:0] ans;
assign set = src1[31] ^ (~src2[31]) ^ Cin[30];
assign ainv = ALU_control[3];
assign binv = ALU_control[2];
assign ope = ALU_control[1:0];
alu_1bit alu0(
    .src1(src1[0]),
    .src2(src2[0]),
    .less(set),
    .Ainvert(ainv),
    .Binvert(binv),
    .cin((ainv ^ binv)),
    .operation(ope),
    .result(ans[0]),
    .cout(Cin[0]));
```

The first is the wire set used for the alu0's input less. Wire ainv and binv are used for all 1-bit alus' input Ainvert and Binvert. Wire op is for all alus' input operation. The alu0 is as the upper code.



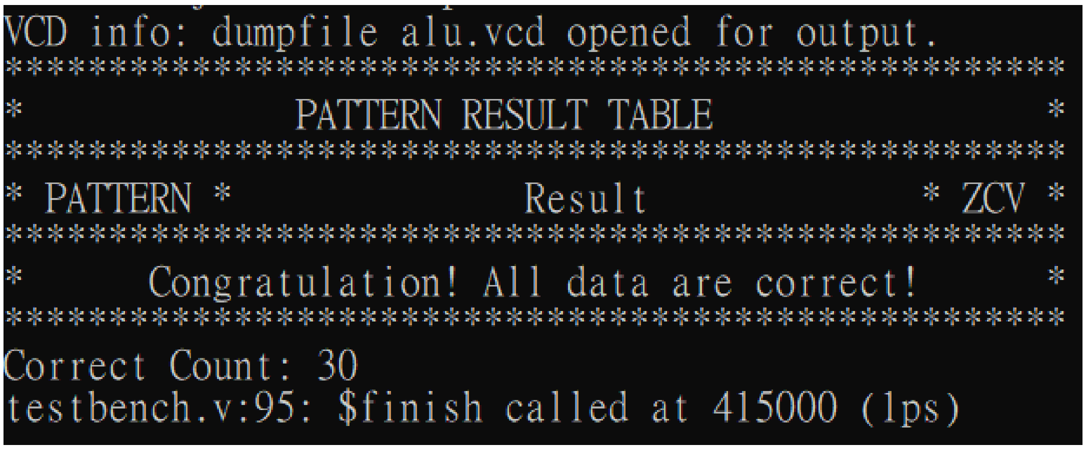
```
genvar i;
generate
    for(i=1 ; i<32 ; i=i+1)begin
        alu_1bit alu(
            .src1(src1[i]),
            .src2(src2[i]),
            .less(1'b0),
            .Ainvert(ainv),
            .Binvert(binv),
            .cin(Cin[i-1]),
            .operation(ope),
            .result(ans[i]),
            .cout(Cin[i]));
    end
endgenerate

always@(*)begin
    if(rst_n==1'b1) begin
        result = ans;
        zero = ~(|result);
        cout = Cin[31] & (ope == 2'd2);
        overflow = (Cin[31] ^ Cin[30]);
    end
    else begin
        result = 32'b0;
        zero = 1'b0;
        cout = 1'b0;
        overflow = 1'b0;
    end
end
```

The generate block contains a for loop that will simplify the code that I have to write for each alu except for alu0. Other alus' structures are similar to alu0, only the less input and the cin input are different. The less input will always be zero, and the cin input is connected to the other wire called Cin, where comes the last 1 bit alu's cout.

When rst_n is 0, then all output are set to be 0. Otherwise, the result register is connected to the ans wire. Zero register will be one if all bits in result are 0. Cout register is set to be the value of cin[31] when the ope wire is 2. Overflow register is the value of xor(Cin[30], Cin[31]).

2. The Implementation Result



```
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)
```

3. Problems encountered and the solutions

(1) I use module in the form as :

Ex. MUX2to1 put1(src1, ~src1, Ainvert, temp1);

Which is not that correct(but still useful). I've even found out that if I want the code still work in this form, then I cannot connect the input wire to the module's input directly, but I need to use another wire to connect to it.

Solution: I changed it to the correct form as the follows in the end:

```
MUX2to1 put1(
        .src1(src1),
        .src2(~src1),
        .select(Ainvert),
        .result(temp1));
```

(2) The cout part needs to be set as zero when doing operations except for add, subtract, and slt. After asking TA on Hackmd, I corrected the output answer.