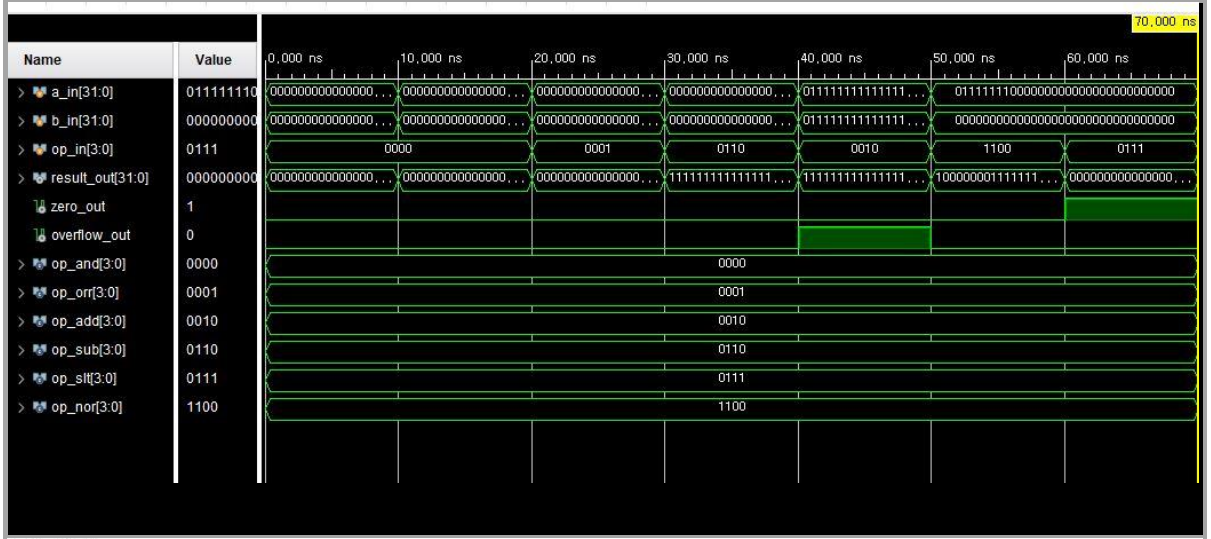
[과제 5] 32bit Arithmetic Logic Unit 설계

```
//과제 5
//이름: 이창민
//학번:2019043890
//융합전자공학부
//코드가 너무 길어서 사진, 설명 먼저 첨부할게요
```



```
a_in[31:0]       011111110
b_in[31:0]       000000000
op_in[3:0]       0111
result_out[31:0] 000000000
zero_out         1
overflow_out     0
op_and[3:0]      0000
op_orr[3:0]      0001
op_add[3:0]      0010
op_sub[3:0]      0110
op_slt[3:0]      0111
op_nor[3:0]      1100
```

```
# run 1000ns
a_in = 00000000000000000000000000001100 b_in 00000000000000000000000000001101 op_in 0000 result_out 00000000000000000000000000001100 zero_out 0 overflow_out 0
a_in = 00000000000000000000000000100001 b_in 00000000000000000000000000111111 op_in 0000 result_out 00000000000000000000000000100001 zero_out 0 overflow_out 0
a_in = 00000000000000000000000000010101 b_in 00000000000000000000000000001100 op_in 0001 result_out 00000000000000000000000000011101 zero_out 0 overflow_out 0
a_in = 00000000000000000000000000001101 b_in 00000000000000000000000000101011 op_in 0110 result_out 11111111111111111111111111100010 zero_out 0 overflow_out 0
a_in = 01111111111111111111111111111111 b_in 01111111111111111111111111111111 op_in 0010 result_out 11111111111111111111111111111110 zero_out 0 overflow_out 1
a_in = 01111110000000000000000000000000 b_in 00000000000000000000000000000000 op_in 1100 result_out 10000001111111111111111111111111 zero_out 0 overflow_out 0
a_in = 01111110000000000000000000000000 b_in 00000000000000000000000000000000 op_in 0111 result_out 00000000000000000000000000000000 zero_out 1 overflow_out 0
$finish called at time : 70 ns : File "C:/Users/BARAMI/project_3/project_3.srcs/sim_1/new/tb.v" Line 35
```

[설명]

기본 1bit ALU 모듈과 overflow 를 계산해주는 1 bit ALU 모듈을 만들고, 이를 32 번 불러서 32bit ALU 를 만들었습니다.

testbench 에서는 가독성을 위해 localparam 을 정의하였고 각각의 상황에 대해 계산을 돌렸습니다. 계산 결과 overflow_out 와 zero_out 도 잘 나오는 걸 확인하였습니다.

```
module ALU_32bit(a_in, b_in, result_out, op_in, zero_out, overflow_out);
    input [31:0] a_in, b_in;
    input [3:0] op_in;
    output [31:0] result_out;
    output zero_out;
    output overflow_out;

    wire [31:0] w_carry;

    ALU_1bit
ALU0(.a_in(a_in[0]), .b_in(b_in[0]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(op_in[2]), .less_in(w_carry[31]), .op_in(op_in[1:0]), .resu
lt_out(result_out[0]), .carry_out(w_carry[0]));
    ALU_1bit
ALU1(.a_in(a_in[1]), .b_in(b_in[1]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[0]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[1]), .carry_out(w_carry[1]));
    ALU_1bit
ALU2(.a_in(a_in[2]), .b_in(b_in[2]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[1]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[2]), .carry_out(w_carry[2]));
    ALU_1bit
ALU3(.a_in(a_in[3]), .b_in(b_in[3]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[2]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[3]), .carry_out(w_carry[3]));
    ALU_1bit
ALU4(.a_in(a_in[4]), .b_in(b_in[4]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[3]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[4]), .carry_out(w_carry[4]));
    ALU_1bit
ALU5(.a_in(a_in[5]), .b_in(b_in[5]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[4]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[5]), .carry_out(w_carry[5]));
    ALU_1bit
ALU6(.a_in(a_in[6]), .b_in(b_in[6]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[5]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[6]), .carry_out(w_carry[6]));
    ALU_1bit
ALU7(.a_in(a_in[7]), .b_in(b_in[7]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[6]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[7]), .carry_out(w_carry[7]));
    ALU_1bit
ALU8(.a_in(a_in[8]), .b_in(b_in[8]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[7]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[8]), .carry_out(w_carry[8]));
    ALU_1bit
ALU9(.a_in(a_in[9]), .b_in(b_in[9]), .Ainvert_in(op_in[3]), .Binvert_in(op_i
n[2]), .carry_in(w_carry[8]), .less_in(1'b0), .op_in(op_in[1:0]), .result_ou
t(result_out[9]), .carry_out(w_carry[9]));
    ALU_1bit
ALU10(.a_in(a_in[10]), .b_in(b_in[10]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[9]), .less_in(1'b0), .op_in(op_in[1:0]), .result
_out(result_out[10]), .carry_out(w_carry[10]));
    ALU_1bit
ALU11(.a_in(a_in[11]), .b_in(b_in[11]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[10]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[11]), .carry_out(w_carry[11]));
```

```
    ALU_1bit
ALU12(.a_in(a_in[12]), .b_in(b_in[12]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[11]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[12]), .carry_out(w_carry[12]));
    ALU_1bit
ALU13(.a_in(a_in[13]), .b_in(b_in[13]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[12]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[13]), .carry_out(w_carry[13]));
    ALU_1bit
ALU14(.a_in(a_in[14]), .b_in(b_in[14]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[13]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[14]), .carry_out(w_carry[14]));
    ALU_1bit
ALU15(.a_in(a_in[15]), .b_in(b_in[15]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[14]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[15]), .carry_out(w_carry[15]));
    ALU_1bit
ALU16(.a_in(a_in[16]), .b_in(b_in[16]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[15]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[16]), .carry_out(w_carry[16]));
    ALU_1bit
ALU17(.a_in(a_in[17]), .b_in(b_in[17]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[16]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[17]), .carry_out(w_carry[17]));
    ALU_1bit
ALU18(.a_in(a_in[18]), .b_in(b_in[18]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[17]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[18]), .carry_out(w_carry[18]));
    ALU_1bit
ALU19(.a_in(a_in[19]), .b_in(b_in[19]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[18]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[19]), .carry_out(w_carry[19]));
    ALU_1bit
ALU20(.a_in(a_in[20]), .b_in(b_in[20]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[19]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[20]), .carry_out(w_carry[20]));
    ALU_1bit
ALU21(.a_in(a_in[21]), .b_in(b_in[21]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[20]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[21]), .carry_out(w_carry[21]));
    ALU_1bit
ALU22(.a_in(a_in[22]), .b_in(b_in[22]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[21]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[22]), .carry_out(w_carry[22]));
    ALU_1bit
ALU23(.a_in(a_in[23]), .b_in(b_in[23]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[22]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[23]), .carry_out(w_carry[23]));
    ALU_1bit
ALU24(.a_in(a_in[24]), .b_in(b_in[24]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[23]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[24]), .carry_out(w_carry[24]));
    ALU_1bit
ALU25(.a_in(a_in[25]), .b_in(b_in[25]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[24]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[25]), .carry_out(w_carry[25]));
    ALU_1bit
ALU26(.a_in(a_in[26]), .b_in(b_in[26]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[25]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[26]), .carry_out(w_carry[26]));
    ALU_1bit
ALU27(.a_in(a_in[27]), .b_in(b_in[27]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[26]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[27]), .carry_out(w_carry[27]));
    ALU_1bit
ALU28(.a_in(a_in[28]), .b_in(b_in[28]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[27]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[28]), .carry_out(w_carry[28]));
    ALU_1bit
ALU29(.a_in(a_in[29]), .b_in(b_in[29]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[28]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[29]), .carry_out(w_carry[29]));
    ALU_1bit
ALU30(.a_in(a_in[30]), .b_in(b_in[30]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[29]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[30]), .carry_out(w_carry[30]));
    ALU_1bit_overflow
ALU31(.a_in(a_in[31]), .b_in(b_in[31]), .Ainvert_in(op_in[3]), .Binvert_in(o
p_in[2]), .carry_in(w_carry[30]), .less_in(1'b0), .op_in(op_in[1:0]), .resul
t_out(result_out[31]), .set_out(w_carry[31]), .overflow_out(overflow_out));

    assign zero_out = (result_out == 0) ? 1 : 0;

endmodule
```

```
module ALU_1bit(a_in, b_in, Ainvert_in, Binvert_in, carry_in, less_in,
op_in, result_out, carry_out);
    input a_in, b_in, Ainvert_in, Binvert_in, carry_in, less_in;
    input [1:0] op_in;
    output carry_out;
    output reg result_out;
    wire w_a;
    wire w_b;

    assign w_a = Ainvert_in ? ~a_in : a_in;
    assign w_b = Binvert_in ? ~b_in : b_in;

    always @(*)
    begin
        case(op_in)
            2'b00 : result_out = w_a & w_b;
            2'b01 : result_out = w_a | w_b;
            2'b10 : result_out = w_a + w_b + carry_in;
            2'b11 : result_out = less_in;
        endcase
    end

    assign carry_out = (((w_a ^ w_b) & carry_in) | (w_a & w_b));

endmodule
```

```
module ALU_1bit_overflow(a_in, b_in, Ainvert_in, Binvert_in, carry_in,
less_in, op_in, result_out, set_out, overflow_out);
    input a_in, b_in, Ainvert_in, Binvert_in, carry_in, less_in;
    input [1:0] op_in;
    output reg result_out;
    output set_out;
    output overflow_out;

    wire w_a;
    wire w_b;
    assign w_a = Ainvert_in ? ~a_in : a_in;
    assign w_b = Binvert_in ? ~b_in : b_in;

    wire carry_out;
    assign carry_out = (((w_a ^ w_b) & carry_in) | (w_a & w_b));

    always @(*) begin
        case(op_in)
            2'b00 : result_out = w_a & w_b;
            2'b01 : result_out = w_a | w_b;
            2'b10 : result_out = w_a + w_b + carry_in;
            2'b11 : result_out = less_in;
        endcase
    end

    assign set_out = w_a + w_b + carry_in;
    assign overflow_out = carry_in ^ carry_out;
```

```
endmodule
```

```
module tb();
    reg [31:0] a_in;
    reg [31:0] b_in;
    reg [3:0] op_in;
    wire [31:0] result_out;
    wire zero_out;
    wire overflow_out;


localparam op_and = 4'b0000,
op_orr=4'b0001,
op_add=4'b0010,
op_sub=4'b0110,
op_slt=4'b0111,
op_nor=4'b1100;
    ALU_32bit ALU32(
        .a_in(a_in),
        .b_in(b_in),
        .op_in(op_in),
        .result_out(result_out),
        .zero_out(zero_out),
        .overflow_out(overflow_out)
    );

    initial begin
        a_in = 12; b_in = 13; op_in = op_and;
        #10 a_in = 33; b_in = 63;
        #10 a_in = 21; b_in = 12; op_in= op_orr;
        #10 a_in = 13; b_in= 43; op_in=op_sub;
        #10 a_in = 32'h7FFFFFFF; b_in = 32'h7FFFFFFF; op_in = op_add;
        #10 a_in = 32'h7F000000; b_in = 0; op_in = op_nor;
        #10 op_in=op_slt;
        #10 $finish;
    end

    initial begin
        $monitor("a_in = %b b_in %b op_in %b result_out %b zero_out %b
overflow_out %b"
        , a_in, b_in, op_in, result_out, zero_out, overflow_out);
    end
```

```
endmodule
```