[과제5] 32bit Arithmetic Logic Unit 설계

|  |
| --- |
| //과제5  //이름: 이창민  //학번:2019043890  //융합전자공학부  //코드가 너무 길어서 사진, 설명 먼저 첨부할게요 |
|  |
|  |
| [설명]  기본 1bit ALU 모듈과 overflow를 계산해주는 1 bit ALU 모듈을 만들고, 이를32번 불러서 32bit ALU를 만들었습니다.  testbench에서는 가독성을 위해 localparam 을 정의하였고 각각의 상황에 대해 계산을 돌렸습니다. 계산 결과 overflow\_out와 zero\_out도 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\_in[2]), .carry\_in(op\_in[2]), .less\_in(w\_carry[31]), .op\_in(op\_in[1:0]), .result\_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\_in[2]), .carry\_in(w\_carry[0]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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\_in[2]), .carry\_in(w\_carry[1]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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\_in[2]), .carry\_in(w\_carry[2]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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\_in[2]), .carry\_in(w\_carry[3]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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\_in[2]), .carry\_in(w\_carry[4]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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\_in[2]), .carry\_in(w\_carry[5]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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\_in[2]), .carry\_in(w\_carry[6]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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\_in[2]), .carry\_in(w\_carry[7]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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\_in[2]), .carry\_in(w\_carry[8]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_out(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(op\_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(op\_in[2]), .carry\_in(w\_carry[10]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[11]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[12]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[13]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[14]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[15]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[16]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[17]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[18]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[19]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[20]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[21]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[22]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[23]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[24]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[25]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[26]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[27]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[28]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[29]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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(op\_in[2]), .carry\_in(w\_carry[30]), .less\_in(1'b0), .op\_in(op\_in[1:0]), .result\_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 |