```
// Eugene Ngo
// 4/7/2023
// CSE 469
// Lab 1 Task 3

// This is the top level module
// alu is the module used to implement a 32 bit ARM-based ALU which
// contains the central logic for calling on submodules that make up the ALU.
module alu(input logic [31:0] a, b,
           input logic [1:0] ALUControl,
           output logic [31:0] Result,
           output logic [3:0] ALUFlags);
   // 00 = add
   // 01 = subtract
   // 10 = AND
   // 11 = OR

   logic [31:0] carries;

   singleALU setCarries (.a(a[0]), .b(b[0]), .carryIn(ALUControl[0]),
                         .ALUControl(ALUControl), .Result(Result[0]),
                         .carryOut(carries[0]));

   genvar i;
   generate
      for (i = 1; i < 32; i++) begin: ALUPipeline
         singleALU results (.a(a[i]), .b(b[i]), .carryIn(carries[i - 1]),
                            .ALUControl(ALUControl), .Result(Result[i]),
                            .carryOut(carries[i]));
      end // end loop
   endgenerate // end generate

   // Setting flags:
   xor overFlowCheck (ALUFlags[0], carries[31], carries[30]);
   assign ALUFlags[1] = carries[31];

   // Inefficient and bad style. RTL would be better.
   nor zeroChecker
      (ALUFlags[2], Result[31], Result[30], Result[29], Result[28],
       Result[27], Result[26], Result[25], Result[24], Result[23],
       Result[22], Result[21], Result[20], Result[19], Result[18],
       Result[17], Result[16], Result[15], Result[14], Result[13],
       Result[12], Result[11], Result[10], Result[9], Result[8],
       Result[7], Result[6], Result[5], Result[4], Result[3],
       Result[2], Result[1], Result[0]);

   assign ALUFlags[3] = Result[31];

endmodule

// alu_testbench tests all expected, unexpected, and edgecase behaviors
module alu_testbench();
   logic [31:0] a,b;
   logic [1:0] ALUControl;
   logic [31:0] Result;
   logic [3:0] ALUFlags;
   logic clk;
   logic [103:0] testvectors [1000:0];

   alu dut (.a(a), .b(b), .ALUControl(ALUControl), .Result(Result),
            .ALUFlags(ALUFlags));

   parameter CLOCK_PERIOD = 100;

   initial clk = 1;

   always begin
         #(CLOCK_PERIOD/2);
         clk = ~clk;
   end

   initial begin
      $readmemh("alu.tv", testvectors);

      for (int i = 0; i < 20; i = i + 1) begin
         {ALUControl, a, b, Result, ALUFlags} = testvectors[i];
         @(posedge clk);
      end // end loop
   end // end initial
endmodule

```