CSULB

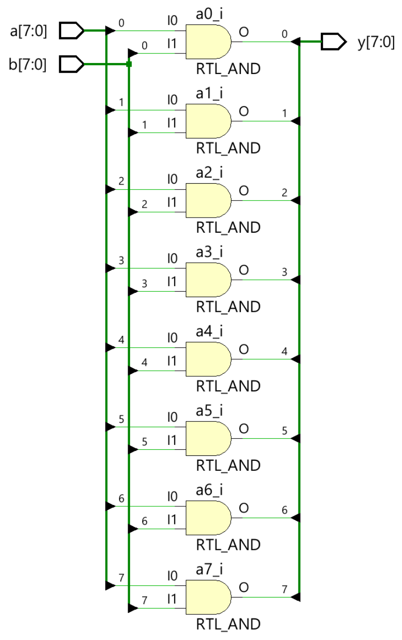
CECS225

Lab ALU

OBJECTIVE: Create an 8-bit ALU (Arithmetic Logic Unit).

PROCEDURE: First make sure you have the halfadder, fulladder, ripplecarry4, ripplecarry8, and ripplesubtracter8 module source files from the previous labs.
*Be sure to use the **exact same file names** that were specified in the previous two labs.*

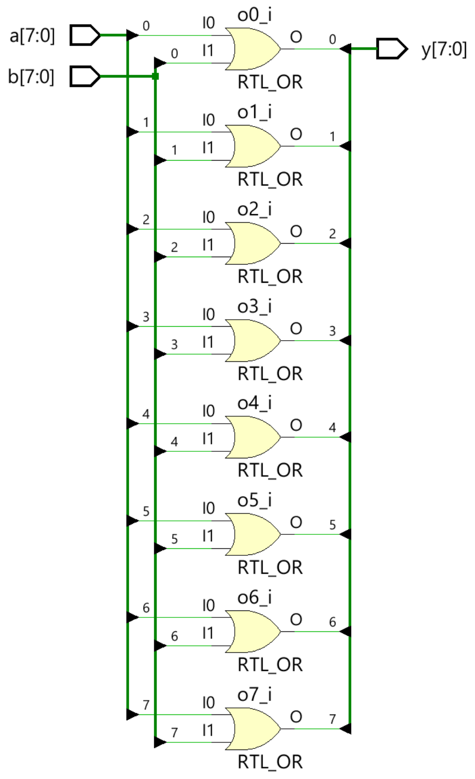
Now create an 8-bit AND gate according to the block diagram below:



Name the file **and8.v** and use the module skeleton below to get started:

```
module and8(
   input [7:0] a, b,
   output [7:0] y
   );
   and
      a0(y[0], a[0],       ),
      a1(y[1], a[1],       ),
      a2(                  ),





      a7(y[7],       , b[7]);

endmodule
```

Then create an 8-bit OR gate according to the block diagram below:

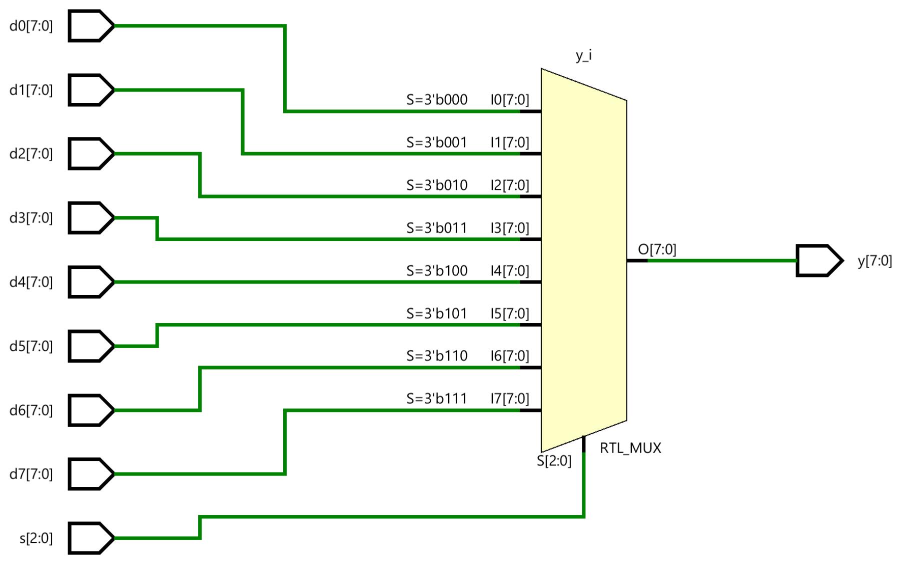


The following module will be given for you to use:

Create **slt8.v** given the Verilog source code below:

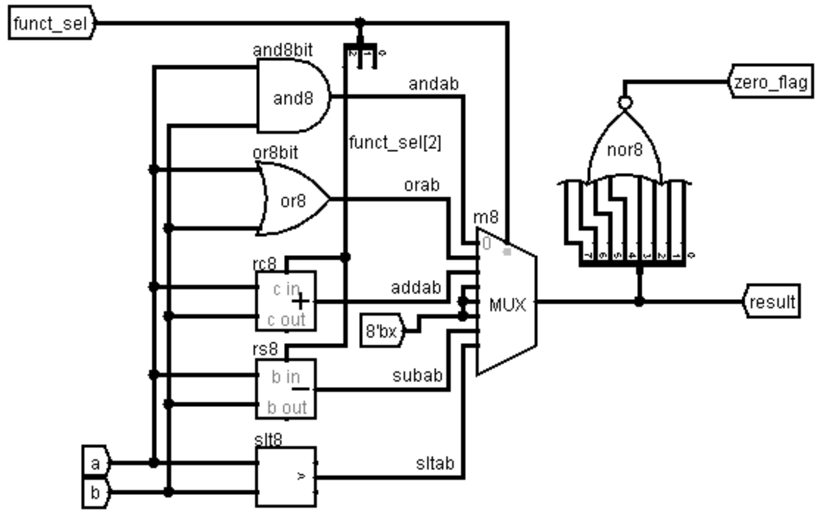
```
module slt8(
    input [7:0] a, b,
    output reg [7:0] y
    );
    always @(*)
    begin
      if (a < b)
        y = 8'b0000_0001;
      else
        y = 8'b0000_0000;
    end
endmodule
```

Create **mux8to1_8bit.v** given the Verilog schematic below:

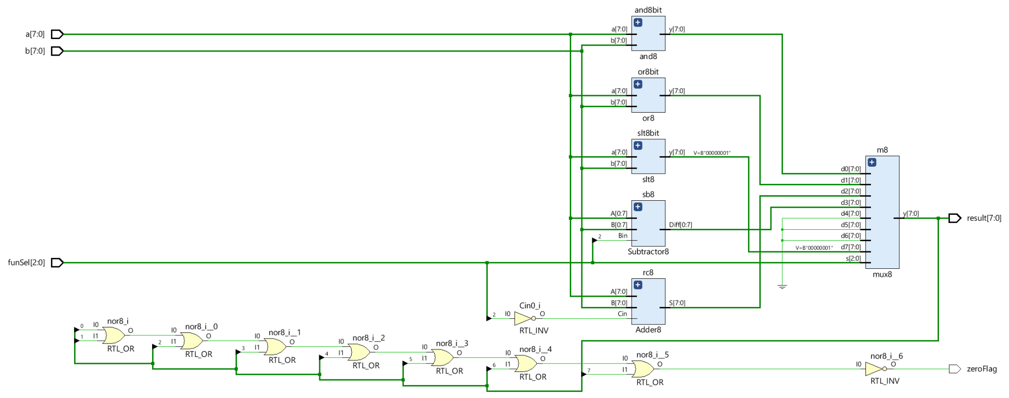


```
module mux8(
   input [7:0] d0, d1, d2, d3, d4, d5, d6, d7,
   input [2:0] s,
   output reg [7:0] y
   );
   always @(*) begin
     case (s)      //begin
       3'b000: y =d0;




       3'b111: y =d7;
       default: y = 8'bx; //invalid output by default
     endcase
   end
endmodule
```

Create the Arithmetic Logic Unit Block diagram below:



Which should give you the following schematic:



Name your Verilog source file as **alu8.v**, a skeleton is shown below:

```
module alu8(
  input [2:0] funSel,
  input [7:0] a, b,
  output zeroFlag,
  output [7:0] result
  );

  wire [7:0] andab, orab, addab, subab, slthb;

  and8 and8bit (a, b,          );
  or8 or8bit (a,         , orab);
  Adder8 rc8(1'b0, a, b, addab,);       //leave the last space after the, empty
  Subtractor8 sb8(          , a, b, subab,);           //leave the last space after the, empty
  slt8 slt8bit (      ,         , slthb);
  mux8 m8(andab,        ,            , 8'bx, 8'bx,8'bx,          ,         , funSel, result);

  nor nor8(zeroFlag, result[0], result[     ],             ,              ,          , result[5],
                                   result[    ],result[    ]);


endmodule
```

Source code for testbench.sv is provided below:

```
module TestBench();
    reg [7:0] a, b;
    reg [2:0] funSel;
    wire [7:0] result;
    wire zero;

    integer i;
    alu8 uut(funSel,a, b, zero, result);
    initial begin
        b = 8'b00001111;
        a = 8'b01011010;
        for(i=0; i< 8; i = i + 1)
        begin
           funSel = i;
           test_case;
        end //for
        b = 8'b00001111;
        a = 8'b01011010;
        test_case;    //one more test case
        $finish;
     end  //initial

    task test_case;
    begin
    #1;
    case (i)
        3'b000: $display("\nTesting the AND function");
        3'b001: $display("\nTesting the OR function");
        3'b010: $display("\nTesting the Addition function");
        3'b110: $display("\nTesting the Subtraction function");
        3'b111: $display("\nTesting the Set Less Than function");
        default: $display("\nInvalid function Selection Code");
    endcase
    if (i== 2)
        i = 5;
    $display("a = %b, b =%b, result =%b, zero = %b", a, b, result, zero);
    end   //test_case
    endtask
endmodule
```

If everything works correctly then the following console output will be produced:

```
Testing the AND function
a = 01011010, b = 00001111, result = 00001010, zero=0

Testing the OR function
a = 01011010, b = 00001111, result = 01011111, zero=0

Testing Addition
a = 01011010, b = 00001111, result = 01101001, zero=0

Testing Subtraction
a = 01011010, b = 00001111, result = 01001011, zero=0

Testing Set Less Than Function
a = 01011010, b = 00001111, result = 00000000, zero=1

Testing Set Less Than Function
a = 00001111, b = 01011010, result = 00000001, zero=0
```

If your module did not work correctly CHECK your code again then ask for help

WHAT TO TURN IN: Once the modules in this lab are working correctly:

- Copy the contents of your modules to a word document, a screenshot of the timing diagram and output in the console, and a link to video showing a screen recording of your while explaining it and showing a simulation running. Save as a pdf and upload to the beachboard dropbox.

NOTE: alaway@(*) is just a shortcut for listing all of the wires that the always block depends on.
An incomplete event_expression list of an event control is a common source of bugs in register transfer level (RTL) simulations. The implicit event_expression, @*, is a convenient shorthand that eliminates these problems by adding all nets and variables that are read by the statement (which can be a statement group) of a procedural_timing_ control_statement to the event_expression.

Extra Credit:
As you can tell we still have room for 3 more operations in this ALU. Add one extra operation of your choice using any of the non-used selection, with all the needed files and testbench modification for an extra 5 points.
If you add three more instructions correctly with all the needed modifications, correct outputs simulation and demonstration, this is 10 more points.