Lab 5-1

```
module lab5 (
input a,
input b,
output reg sum, //add reg for output for the behavioral
output reg carry
);

/*structural
xor(sum, a, b);
and (carry, a, b);*/

/*dataflow
assign sum = a ^ b;
assign carry = a & b;*/

/* behavioral description */
always @(*)
begin
sum = a ^ b;
carry = a & b;
end


endmodule
```

Lab 5-2

```
module test (
input a, b, c_in,
output sum, c_out
);

assign sum = (a^b)^c_in;
assign c_out = (a&b)|((a^b)&c_in);

endmodule
```

Lab 6-1

```
module LAB6(
input [3:0] A,
input [3:0] B,
input select,
output [3:0] Result,
output overflow,
output carry
);

wire [4:0] sum_result;
wire [4:0] sub_result;
reg [4:0] operation_result;

assign sum_result = {1'b0,A}+{1'b0,B};
assign sub_result = {1'b0,A}-{1'b0,B};

always @(*) begin
if (select)
operation_result = sub_result;
else
operation_result= sum_result;
end

assign Result= operation_result[3:0];
assign overflow= operation_result[4];
assign carry= (select==0)? sum_result[4]: (sub_result[4] != 0); //condition ? if_value_true :
if_value_false

endmodule
```

Lab6-2

```
module lab6_2 (
   input [7:0] in,       // 8 input switches
   output reg [2:0] Out, // 3-bit encoded output
   output reg Valid      // Valid output
);
   always @(*) begin
      case (in)
         8'b00000001: begin
            Out = 3'b000; // Input 0
            Valid = 1'b1; // Valid
         end
         8'b00000010: begin
            Out = 3'b001; // Input 1
            Valid = 1'b1; // Valid
         end
```

```
            8'b00000100: begin
                Out = 3'b010; // Input 2
                Valid = 1'b1; // Valid
            end
            8'b00001000: begin
                Out = 3'b011; // Input 3
                Valid = 1'b1; // Valid
            end
            8'b00010000: begin
                Out = 3'b100; // Input 4
                Valid = 1'b1; // Valid
            end
            8'b00100000: begin
                Out = 3'b101; // Input 5
                Valid = 1'b1; // Valid
            end
            8'b01000000: begin
                Out = 3'b110; // Input 6
                Valid = 1'b1; // Valid
            end
            8'b10000000: begin
                Out = 3'b111; // Input 7
                Valid = 1'b1; // Valid
            end
            default: begin
                Out = 3'b000; // Default output
                Valid = 1'b0; // Invalid
            end
        endcase
    end
endmodule
```

Lab 7

```
module Lab_7_correct (
input [3:0] SW,
output [6:0] HEX0
);

reg [6:0] result;

always @(*) begin
case (SW)
```

```
4'b0000: result = 7'b1000000;//0
4'b0001: result = 7'b1111001;//1
4'b0010: result = 7'b0100100;//2
4'b0011: result = 7'b1111000;//7
4'b0100: result = 7'b0010000;//9
4'b0101: result = 7'b0000011;//11
4'b0110: result = 7'b0000011;//11
4'b0111: result = 7'b0100001;//13
4'b1000: result = 7'b0001110;//15
default: result = 7'b1000000;
endcase

end
assign HEX0 = result;

endmodule


/*list of numbers in display
7'b1000000; // 0
7'b1111001; // 1
7'b0100100; // 2
7'b0110000; // 3
7'b0011001; // 4
7'b0010010; // 5
7'b0000010; // 6
7'b1111000; // 7
7'b0000000; // 8
7'b0010000; // 9
7'b0001000; // A
7'b0000011; // B
7'b1000110; // C
7'b0100001; // D
7'b0000110; // E
7'b0001110; // F
*/
```

Lab 8

```
module Lab8 (
   input clock,      // 50 MHz clock input
   input reset,      // Active-low reset
   input Up_Down,    // Count direction (1: up, 0: down)
```

```
    output [6:0] out  // Seven-segment display output
);
    // Internal signals
    reg [25:0] freq_counter; // For frequency division
    reg clk_div;             // 1 Hz clock signal
    reg [3:0] count;         // 4-bit counter
    reg [6:0] display_out;   // Seven-segment display signal

    // Frequency Divider: Generate 1 Hz clock
    always @(posedge clock or negedge reset) begin
        if (!reset) begin
            freq_counter <= 0;
            clk_div <= 0;
        end else if (freq_counter == 25_000_000 - 1) begin
            freq_counter <= 0;
            clk_div <= ~clk_div;
        end else begin
            freq_counter <= freq_counter + 1;
        end
    end

    // Counter: Count up or down based on Up_Down
    always @(posedge clk_div or negedge reset) begin
        if (!reset) begin
            count <= 4'b0000; // Reset to 0
        end else begin
            if (Up_Down) begin
                count <= count + 1; // Count up
            end else begin
                count <= count - 1; // Count down
            end
        end
    end

    // Seven-Segment Display: Map count value to output
    always @(count) begin
        case (count)
            4'd0:  display_out = 7'b1000000;
            4'd1:  display_out = 7'b1111001;
            4'd2:  display_out = 7'b0100100;
            4'd3:  display_out = 7'b0110000;
            4'd4:  display_out = 7'b0011001;
            4'd5:  display_out = 7'b0010010;
            4'd6:  display_out = 7'b0000010;
            4'd7:  display_out = 7'b1111000;
            4'd8:  display_out = 7'b0000000;
            4'd9:  display_out = 7'b0010000;
            4'd10: display_out = 7'b0001000;
            4'd11: display_out = 7'b0000011;
            4'd12: display_out = 7'b1000110;
            4'd13: display_out = 7'b0100001;
            4'd14: display_out = 7'b0000110;
            4'd15: display_out = 7'b0001110;
            default: display_out = 7'b1111111; // Blank display
        endcase
    end

    // Output assignment
    assign out = display_out;

endmodule
```