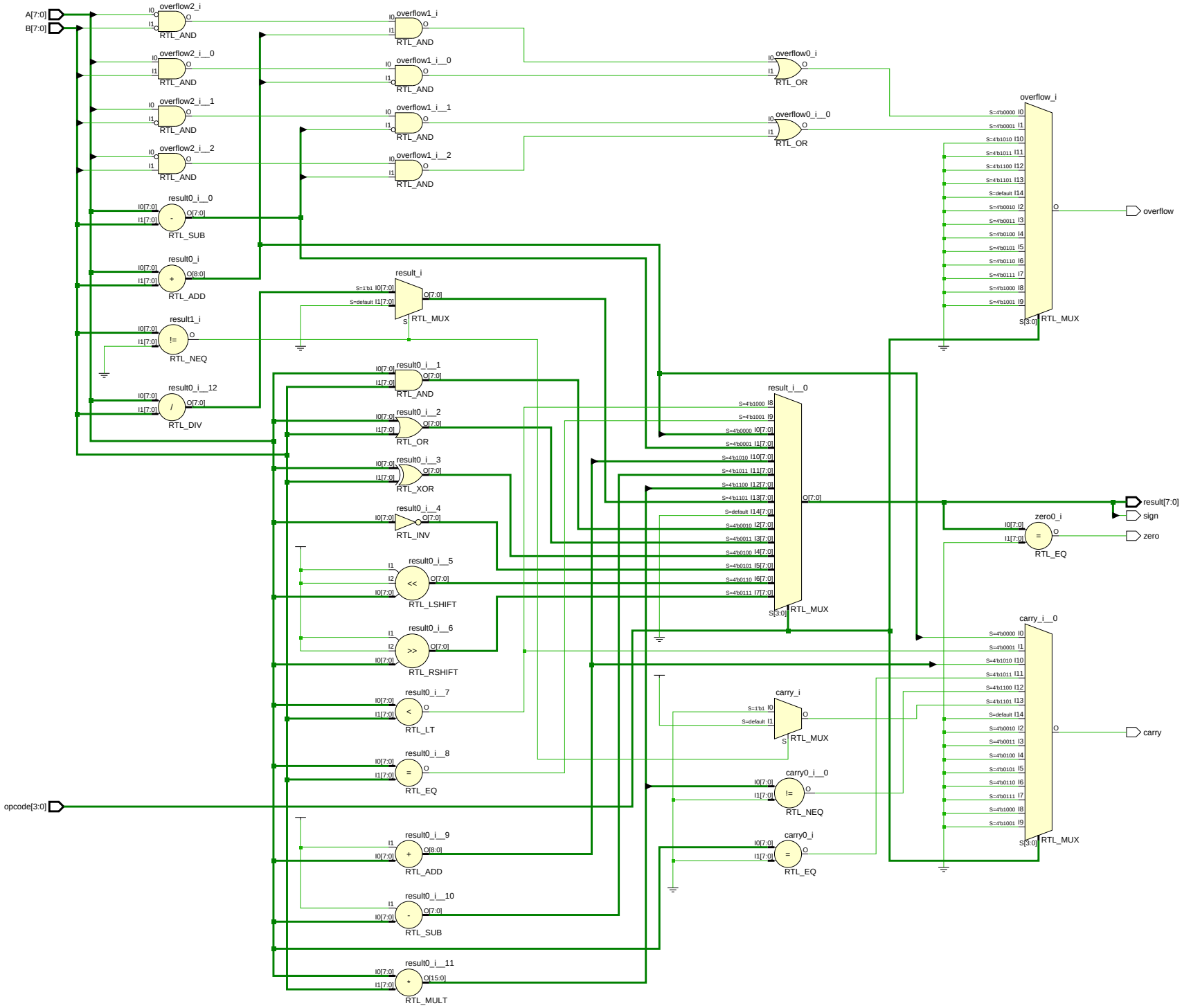
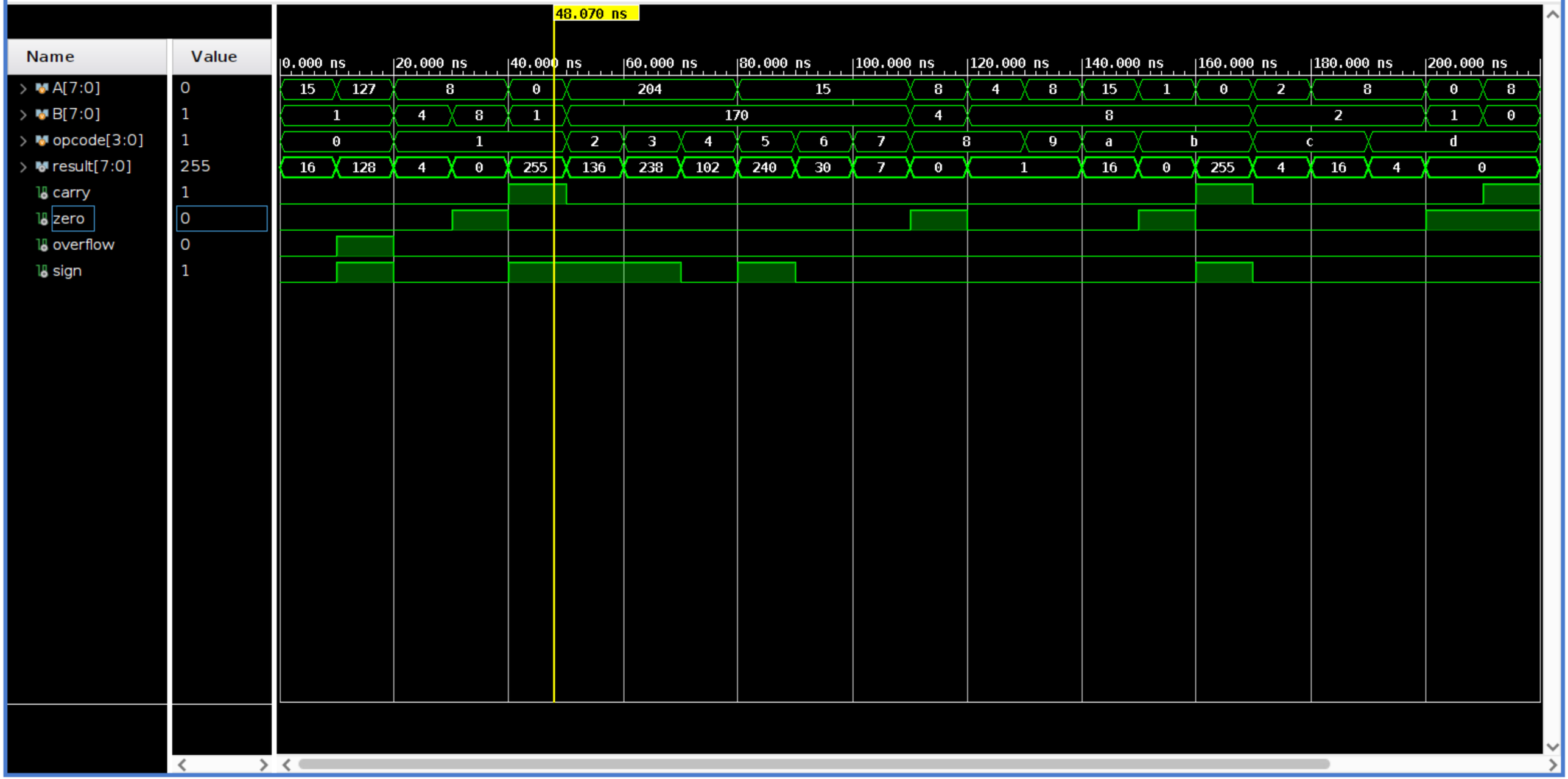

A[7:0]
B[7:0]
opcode[3:0]
overflow2_i
RTL_AND
overflow2_i__0
RTL_AND
overflow2_i__1
RTL_AND
overflow2_i__2
RTL_AND
overflow1_i
RTL_AND
overflow1_i__0
RTL_AND
overflow1_i__1
RTL_AND
overflow1_i__2
RTL_AND
overflow0_i
RTL_OR
overflow0_i__0
RTL_OR
overflow_i
RTL_MUX
overflow
result0_i__0
RTL_SUB
result0_i
RTL_ADD
result1_i
RTL_NEQ
result0_i__12
RTL_DIV
result_i
RTL_MUX
result0_i__1
RTL_AND
result0_i__2
RTL_OR
result0_i__3
RTL_XOR
result0_i__4
RTL_INV
result0_i__5
RTL_LSHIFT
result0_i__6
RTL_RSHIFT
result0_i__7
RTL_LT
result0_i__8
RTL_EQ
result0_i__9
RTL_ADD
result0_i__10
RTL_SUB
result0_i__11
RTL_MULT
result_i__0
RTL_MUX
result[7:0]
sign
zero0_i
RTL_EQ
zero
carry_i__0
RTL_MUX
carry
carry_i
RTL_MUX
carry0_i__0
RTL_NEQ
carry0_i
RTL_EQ

alu_8bit.v | alu_8bit_tb.v | Untitled 5*



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Anjan Prasad
// Create Date: 11/10/2024 05:20:17 AM
// Module Name: alu_8bit
//////////////////////////////////////////////////////////////////////////////////

module alu_8bit (

//    input wire Clk,
    input wire [7:0] A,                                 // 8-bit input A
    input wire [7:0] B,                                 // 8-bit input B
    input wire [3:0] opcode,                            // 4-bit opcode for operation select
    output reg [7:0] result,                            // 8-bit ALU result
    output reg carry,                                   // Carry flag
    output reg zero,                                    // Zero flag
    output reg overflow,                                // Overflow flag
    output reg sign                                     // Sign flag
);

    reg [15:0] temp;                                    // Temporary register to store result for wide operations

    // Opcode Definitions (4-bit values)
    localparam ADD    = 4'b0000;                        // Addition
    localparam SUB    = 4'b0001;                        // Subtraction
    localparam AND    = 4'b0010;                        // Bitwise AND
    localparam OR     = 4'b0011;                        // Bitwise OR
    localparam XOR    = 4'b0100;                        // Bitwise XOR
    localparam NOT    = 4'b0101;                        // Bitwise NOT (A only)
    localparam LSHIFT = 4'b0110;                        // Logical Shift Left
    localparam RSHIFT = 4'b0111;                        // Logical Shift Right
    localparam LT     = 4'b1000;                        // Less Than (A < B)
    localparam EQ     = 4'b1001;                        // Equal (A == B)
    localparam INC    = 4'b1010;                        // Increment A
    localparam DEC    = 4'b1011;                        // Decrement A
    localparam MUL    = 4'b1100;                        // Multiplication (A * B)
    localparam DIV    = 4'b1101;                        // Division (A / B)

    // ALU Operations based on Opcode
    always @(*) begin

                                                        // Initialize flags
        carry = 0;
        overflow = 0;
        zero = 0;
        sign = 0;

        case (opcode)
            ADD: begin                                  // Addition
                temp = A + B;
                result = temp[7:0];
                carry = temp[8];  // Capture carry
                overflow = (~A[7] & ~B[7] & result[7]) | (A[7] & B[7] & ~result[7]);  // Overflow check
            end
            SUB: begin                                  // Subtraction
                temp = A - B;
                result = temp[7:0];
                carry = (A < B);                        // Capture borrow
                overflow = (A[7] & ~B[7] & ~result[7]) | (~A[7] & B[7] & result[7]);  // Overflow check
            end
            AND: result = A & B;                        // Bitwise AND
            OR:  result = A | B;                        // Bitwise OR
            XOR: result = A ^ B;                        // Bitwise XOR
            NOT: result = ~A;                           // Bitwise NOT
            LSHIFT: result = A << 1;                    // Logical Shift Left
            RSHIFT: result = A >> 1;                    // Logical Shift Right
            LT: result = (A < B);                       // A < B (Comparison)
            EQ: result = (A == B);                      // A == B (Comparison)
            INC: begin
                temp = A + 1;
                result = temp[7:0];
                carry = temp[8];                        // Capture carry for overflow
            end
            DEC: begin
                temp = A - 1;
                result = temp[7:0];
                carry = (A == 0);                       // Carry if decrement causes underflow
            end
            MUL: begin                                  // Multiplication
                temp = A * B;
                result = temp[7:0];                     // Only lower 8 bits used
                carry = temp[15:8] != 0;                // Set carry if multiplication result exceeds 8 bits
            end
            DIV: begin                                  // Division
                if (B != 0) begin
                    result = A / B;                     // Division (integer result)
                    carry = 0;                          // No carry in division
                end else begin
                    result = 8'b00000000;               // Division by zero case
                    carry = 1;                          // Set carry flag for error
                end
            end
            default: result = 8'b00000000;              // Default to 0 if no match
        endcase

                                                        // Set Zero flag if result is 0
        zero = (result == 8'b00000000);

                                                        // Set Sign flag (1 for negative result)
        sign = result[7];
    end
endmodule
```

```
1
2  //////////////////////////////////////////////////////////////////////////////////
3  // Engineer:
4  // Create Date: 11/10/2024 05:24:45 AM
5  // Module Name: alu_8bit_tb
6  //////////////////////////////////////////////////////////////////////////////////
7
8  `timescale 1ns / 1ps
9
10 module tb_alu_8bit;
11
12                                                          // Inputs
13     reg [7:0] A;
14     reg [7:0] B;
15     reg [3:0] opcode;
16
17                                                          // Outputs
18     wire [7:0] result;
19     wire carry;
20     wire zero;
21     wire overflow;
22     wire sign;
23
24                                                          // Instantiate the ALU
25     alu_8bit uut (
26         .A(A),.B(B),.opcode(opcode),.result(result),.carry(carry),.zero(zero),.overflow(overflow),.sign(sign)
27     );
28
29                                                          // Task to display results
30     task display_results;
31         input [7:0] A;
32         input [7:0] B;
33         input [3:0] opcode;
34         input [7:0] expected_result;
35         input expected_carry;
36         input expected_zero;
37         input expected_overflow;
38         input expected_sign;
39         begin
40                                                          // Display the results
41             $display("A = %b, B = %b, Opcode = %b | Result = %b, Carry = %b, Zero = %b, Overflow = %b, Sign = %b",
42                      A, B, opcode, result, carry, zero, overflow, sign);
43                                                          // Check the results
44             if (result !== expected_result)
45                 $display("Error: Expected Result = %b", expected_result);
46             if (carry !== expected_carry)
47                 $display("Error: Expected Carry = %b", expected_carry);
48             if (zero !== expected_zero)
49                 $display("Error: Expected Zero = %b", expected_zero);
```

```
10  module tb_alu_8bit;
30      task display_results;
39          begin
48              if (zero !== expected_zero)
50              if (overflow !== expected_overflow)
51                  $display("Error: Expected Overflow = %b", expected_overflow);
52              if (sign !== expected_sign)
53                  $display("Error: Expected Sign = %b", expected_sign);
54          end
55      endtask
56
57      initial begin
58                                                                                  // Test Addition
59          A = 8'b00001111; B = 8'b00000001; opcode = 4'b0000;                     // 15 + 1
60          #10 display_results(A, B, opcode, 8'b00010000, 1'b0, 1'b0, 1'b0, 1'b0); // Expected: 16, no carry, no zero, no overflow, positive
61
62          A = 8'b01111111; B = 8'b00000001; opcode = 4'b0000;                     // 127 + 1
63          #10 display_results(A, B, opcode, 8'b10000000, 1'b0, 1'b0, 1'b1, 1'b1); // Expected: 128, no carry, no zero, overflow, negative
64
65                                                                                  // Test Subtraction
66          A = 8'b00001000; B = 8'b00000100; opcode = 4'b0001;                     // 8 - 4
67          #10 display_results(A, B, opcode, 8'b00000100, 1'b0, 1'b0, 1'b0, 1'b0); // Expected: 4, no carry, no zero, no overflow, positive
68
69          A = 8'b00001000; B = 8'b00001000; opcode = 4'b0001;                     // 8 - 8
70          #10 display_results(A, B, opcode, 8'b00000000, 1'b0, 1'b1, 1'b0, 1'b0); // Expected: 0, no carry, zero, no overflow, positive
71
72          A = 8'b00000000; B = 8'b00000001; opcode = 4'b0001;                     // 0 - 1
73          #10 display_results(A, B, opcode, 8'b11111111, 1'b1, 1'b0, 1'b0, 1'b1); // Expected: 255, carry (borrow), no zero, no overflow, negative
74
75                                                                                  // Test Logical Operations
76          A = 8'b11001100; B = 8'b10101010; opcode = 4'b0010;                     // A AND B
77          #10 display_results(A, B, opcode, 8'b10001000, 1'b0, 1'b0, 1'b0, 1'b0); // Expected: 136, no carry, no zero, no overflow, positive
78
79          A = 8'b11001100; B = 8'b10101010; opcode = 4'b0011;                     // A OR B
80          #10 display_results(A, B, opcode, 8'b11101110, 1'b0, 1'b0, 1'b0, 1'b0); // Expected: 238, no carry, no zero, no overflow, positive
81
82          A = 8'b11001100; B = 8'b10101010; opcode = 4'b0100;                     // A XOR B
83          #10 display_results(A, B, opcode, 8'b01100110, 1'b0, 1'b0, 1'b0, 1'b0); // Expected: 102, no carry, no zero, no overflow, positive
84
85          A = 8'b00001111; opcode = 4'b0101;                                      // NOT A
86          #10 display_results(A, B, opcode, 8'b11110000, 1'b0, 1'b0, 1'b0, 1'b1); // Expected: 240, no carry, no zero, no overflow, negative
87
88                                                                                  // Test Shift Operations
89          A = 8'b00001111; opcode = 4'b0110;                                      // A LSHIFT
90          #10 display_results(A, B, opcode, 8'b00011110, 1'b0, 1'b0, 1'b0, 1'b0); // Expected: 30, no carry, no zero, no overflow, positive
91
92          A = 8'b00001111; opcode = 4'b0111;                                      // A RSHIFT
93          #10 display_results(A, B, opcode, 8'b00000111, 1'b0, 1'b0, 1'b0, 1'b0); // Expected: 7, no carry, no zero, no overflow, positive
49              $display("Error: Expected Zero = %b", expected_zero);
```

```
10  module tb_alu_8bit;
57      initial begin
93          #10 display_results(A, B, opcode, 8'b00000111, 1'b0, 1'b0, 1'b0, 1'b0);
94
95
96          A = 8'b00001000; B = 8'b00000100; opcode = 4'b1000;
97          #10 display_results(A, B, opcode, 8'b00000000, 1'b0, 1'b0, 1'b0, 1'b0);
98
99          A = 8'b00000100; B = 8'b00001000; opcode = 4'b1000;
100         #10 display_results(A, B, opcode, 8'b00000001, 1'b0, 1'b0, 1'b0, 1'b0);
101
102         A = 8'b00001000; B = 8'b00001000; opcode = 4'b1001;
103         #10 display_results(A, B, opcode, 8'b00000001, 1'b0, 1'b0, 1'b0, 1'b0);
104
105
106         A = 8'b00001111; opcode = 4'b1010;
107         #10 display_results(A, B, opcode, 8'b00010000, 1'b0, 1'b0, 1'b0, 1'b0);
108
109         A = 8'b00000001; opcode = 4'b1011;
110         #10 display_results(A, B, opcode, 8'b00000000, 1'b0, 1'b1, 1'b0, 1'b0);
111
112         A = 8'b00000000; opcode = 4'b1011;
113         #10 display_results(A, B, opcode, 8'b11111111, 1'b1, 1'b0, 1'b0, 1'b1);
114
115
116         A = 8'b00000010; B = 8'b00000010; opcode = 4'b1100;
117         #10 display_results(A, B, opcode, 8'b00000100, 1'b0, 1'b0, 1'b0, 1'b0);
118
119         A = 8'b00001000; B = 8'b00000010; opcode = 4'b1100;
120         #10 display_results(A, B, opcode, 8'b00010000, 1'b0, 1'b0, 1'b0, 1'b0);
121
122
123         A = 8'b00001000; B = 8'b00000010; opcode = 4'b1101;
124         #10 display_results(A, B, opcode, 8'b00000100, 1'b0, 1'b0, 1'b0, 1'b0);
125
126         A = 8'b00000000; B = 8'b00000001; opcode = 4'b1101;
127         #10 display_results(A, B, opcode, 8'b00000000, 1'b0, 1'b1, 1'b0, 1'b0);
128
129         A = 8'b00001000; B = 8'b00000000; opcode = 4'b1101;
130         #10 display_results(A, B, opcode, 8'b00000000, 1'b1, 1'b0, 1'b0, 1'b0);
131
132
133         $finish;
134     end
135 endmodule
136
```

```
// A RSHIFT
// Expected: 7, no carry, no zero, no overflow, positive

// Test Comparison Operations
// A < B
// Expected: 0, no carry, no zero, no overflow, positive

// A < B
// Expected: 1, no carry, no zero, no overflow, positive

// A == B
// Expected: 1, no carry, no zero, no overflow, positive

// Test Increment and Decrement
// Increment A
// Expected: 16, no carry, no zero, no overflow, positive

// Decrement A
// Expected: 0, no carry, zero, no overflow, positive

// Decrement A
// Expected: 255, carry, no zero, no overflow, negative

// Test Multiplication
// 2 * 2
// Expected: 4, no carry, no zero, no overflow, positive

// 8 * 2
// Expected: 16, no carry, no zero, no overflow, positive

// Test Division
// 8 / 2
// Expected: 4, no carry, no zero, no overflow, positive

// 0 / 1
// Expected: 0, no carry, zero, no overflow, positive

// Division by zero
// Expected: 0, carry for division by zero

// End of simulation
```