Lab Report
CSE4010 Computer Architecture

Name: Zachary A. Hampton **Score:** /10
Student ID: 008339494 **Due:** 02-23-2025
Lab: Lab 3 - Full Adder and Full Subtractor Implementation

Report

- How can addition and subtraction of large numbers be performed by computers if they are only aware of numbers 0 and 1?
  - Computers perform arithmetic operations on large numbers by breaking them down into binary digits and processing them one bit at a time. For multi-digit numbers:
    - Addition is performed using full adders in cascade, where each full adder processes one bit position and handles the carry from the previous stage
    - Subtraction is similarly performed using full subtractors in cascade, where each subtractor handles one bit position and processes the borrow from the previous stage
    - The carry/borrow propagation between stages allows for handling numbers of any size
  - This binary arithmetic is implemented using combinational circuits that:
    - Process bits from least significant to most significant position
    - Use carry/borrow propagation for managing digit transitions
    - Can be extended to any number of bits by cascading multiple stages

Source Code

fullAdder.v

```
module fullAdder(
    input A,
    input B,
    input Cin,
    output sum,
    output Cout
);

    wire sum1, carry1, carry2;

    // First half adder
    assign sum1 = A ^ B;
    assign carry1 = A & B;

    // Second half adder
    assign sum = sum1 ^ Cin;
    assign carry2 = sum1 & Cin;

    // Final carry output
    assign Cout = carry1 | carry2;

endmodule
```

fullAdder_tb.v

```
`timescale 1ns/1ns
`include "fullAdder.v"

module fullAdder_tb;
    reg A, B, Cin;
    wire sum, Cout;

    fullAdder uut(
        .A(A),
        .B(B),
        .Cin(Cin),
        .sum(sum),
        .Cout(Cout)
    );

    initial begin
        $dumpfile("fullAdder_tb.vcd");
        $dumpvars(0, fullAdder_tb);

        // Test all input combinations
        A = 0; B = 0; Cin = 0; #20;
        A = 0; B = 0; Cin = 1; #20;
        A = 0; B = 1; Cin = 0; #20;
        A = 0; B = 1; Cin = 1; #20;
        A = 1; B = 0; Cin = 0; #20;
        A = 1; B = 0; Cin = 1; #20;
        A = 1; B = 1; Cin = 0; #20;
        A = 1; B = 1; Cin = 1; #20;

        $display("Test Complete!");
        $finish;
    end

    initial begin
        $monitor("Time=%0d A=%b B=%b Cin=%b sum=%b Cout=%b",
                 $time, A, B, Cin, sum, Cout);
    end
endmodule
```

fullSubtractor.v

```
module fullSubtractor(
    input A,
    input B,
    input Bin,
    output diff,
    output Bout
);

    wire diff1, borrow1, borrow2;

    // First half subtractor
    assign diff1 = A ^ B;
    assign borrow1 = !A & B;

    // Second half subtractor
    assign diff = diff1 ^ Bin;
    assign borrow2 = !diff1 & Bin;

    // Final borrow output
    assign Bout = borrow1 | borrow2;

endmodule
```

fullSubtractor_tb.v

```
`timescale 1ns/1ns
`include "fullSubtractor.v"

module fullSubtractor_tb;
    reg A, B, Bin;
    wire diff, Bout;

    fullSubtractor uut(
        .A(A),
        .B(B),
        .Bin(Bin),
        .diff(diff),
        .Bout(Bout)
    );

    initial begin
        $dumpfile("fullSubtractor_tb.vcd");
        $dumpvars(0, fullSubtractor_tb);

        // Test all input combinations
        A = 0; B = 0; Bin = 0; #20;
        A = 0; B = 0; Bin = 1; #20;
        A = 0; B = 1; Bin = 0; #20;
        A = 0; B = 1; Bin = 1; #20;
        A = 1; B = 0; Bin = 0; #20;
        A = 1; B = 0; Bin = 1; #20;
        A = 1; B = 1; Bin = 0; #20;
        A = 1; B = 1; Bin = 1; #20;

        $display("Test Complete!");
        $finish;
    end

    initial begin
        $monitor("Time=%0d A=%b B=%b Bin=%b diff=%b Bout=%b",
                 $time, A, B, Bin, diff, Bout);
    end
endmodule
```

Screenshots

Part A - Full Adder Simulation

[Insert screenshot of full adder simulation waveform and results here]

Part B - Full Subtractor Simulation

[Insert screenshot of full subtractor simulation waveform and results here]