Section 1: HARDCOPY SUBMISSION (100 Points)

Problem 1 (20 Points):
With \( x = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101\ 1011 \) and \( y = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101 \) representing two’s complement signed integers, perform, showing all work:

(a) \( x + y \)
(b) \( x - y \)
(c) \( x \times y \)
(d) \( x/y \)

Problem 2 (20 Points): Given the bit pattern, 1010 1101 0001 0000 0000 0000 0000 0000 0010, what does it represent, assuming that is

(a) a two’s complement integer?
(b) an unsigned integer?
(c) a single precision floating-point number?
(d) a MIPS instruction?

Problem 3 (20 Points): Show the single precision IEEE 754 binary representation for the floating point number \( 20_{\text{ten}} \) and \(-5/6_{\text{ten}}\).

Problem 4 (20 Points): A sequential circuit with 3 FFs is given below. Write the FF input and output equations \((A, B, C)\) and complete the state table of this sequential circuit. Draw the transition table and state diagram. Is this a Mealy Machine or Moore Machine?
Problem 5 (20 Points): Design a sequential circuit that recognizes the occurrence of a particular sequence of bits regardless of where it occurs in a longer sequence. This sequence recognizer has one input X and one output Z. The circuit is to recognize the occurrence of the sequence bits 1101 on X by making Z equal to 1 when the previous three inputs to the circuit were 110 and the current input is a 1. Otherwise Z is 0. Draw the state diagram and state table.

Section 2: VERILOG (100 Points)
(a) (20 Points) Design a 4-bit CLA module. Your module must have the following format. Submit your verilog code. Convince yourself that the code is working correctly with test cases.

module CLA(g0, p0, g1, p1, g2, p2, g3, p3, cin, C1, C2, C3, C4, G, P);
input g0, p0, g1, p1, g2, p2, g3, p3; // Generate and propagate signals corresponding to each bit.
input cin; // Carry-in input
output C1, C2, C3, C4; // Carry bits computed by the CLA.
output G, P; // Block generate and block propagate to be used by CLAs at a higher level.
(b) (20 Points) Modify this one-bit ALU that you designed to generate and propagate signals as shown in Figure below.

```verilog
module OneBitALU(a, b, cin, less, op, result, cout, g, p, set);
input a, b, cin; // Inputs to the one-bit ALU, "cin" is the carry-in bit.
input [2:0] op; // 3-bit operation code. op[2] is the "binv". op[0] is the least significant bit.
input less; // This input will be set as 0 for all ALUs but the one corresponding to the most-significant bit.
output result; // The result of the ALU (depends on the operation that is chosen)
output cout; // Carry out bit of the adder
output g, p; // Generate and propagate signals that are to be used by the CLA unit.
output set; // This is the "sum" output of the full-adder.
```

(c) (10 Points) Design an overflow detection module in verilog. Name the module as "OverflowDetection". Refer to the posting on overflow detection on the webpage.

a. Assuming that this overflow detection will be used in a 32-bit ALU implementation, identify the inputs to the OverflowDetection module.

b. Submit your verilog code and waveforms with comments.

(d) (50 Points) Design a 4-bit ALU by using 4 one-bit ALUs, a CLA, and Overflow detection unit. The four-bit inputs are represented in 2's compliment notation. Submit the verilog model and the waveform showing that your 4-bit ALU is working correctly. The verilog model must have the following definition. Note that this module does not have the "cin" and "less" inputs. The "cin" input is the "binv" (op[2]) input. The most significant "set" bit is fed as input to the least significant "less" bit. (See figure below.)
module FourBitALU(a, b, op, result, cout, G, P, set, overflow);

input [3:0] a, b; // Inputs to the one-bit ALU.
input [2:0] op; // 3-bit operation code. op[2] is the "binv". op[0] is the least significant bit.
output [3:0] result; // The result of the ALU (depends on the operation that is chosen)
output cout; // Carry-out bit of the ALU
output G, P; // Block generate and propagate of the four-bit ALU
output set; // This is the set output of the most significant ALU block
output overflow; // This bit indicates that an overflow has occurred. (Ignores what operation is chosen for ALU)