Name ______________________________

Computer Organization

EE 3755

Practice Midterm Examination

23 October 2001

Problem 1	______	(10 pts)
Problem 2	______	(30 pts)
Problem 3	______	(10 pts)
Problem 4	______	(10 pts)
Problem 5	______	(10 pts)
Problem 6	______	(10 pts)
Problem 7	______	(10 pts)
Problem 8	______	(10 pts)
Exam Total	______	(100 pts)

Alias ______________________________

Good Luck!

Problem 1: Add Verilog code to the module below for the `carry` signals and `sum[3]` using the generate and propagate signals. *Hint: This is straight from the notes.* (10 pts)

```
module cla_3(sum,a,b);
   input [2:0] a, b;
   output [3:0] sum;

   wire [2:0]   g, p, carry;

   assign       carry[0] =



   assign       carry[1] =



   assign       carry[2] =



   assign       sum[3] =




   cla_slice s0(sum[0],g[0],p[0],a[0],b[0],carry[0]);
   cla_slice s1(sum[1],g[1],p[1],a[1],b[1],carry[1]);
   cla_slice s2(sum[2],g[2],p[2],a[2],b[2],carry[2]);

endmodule
```

Problem 2: Complete the module below so that it determines whether its input, a floating point number in IEEE 754 single format, is positive, zero, negative, and whether it is an integer. Output `pos` is `1` if the input is positive, `neg` is `1` if it's negative, etc. The solution can ignore special values ($\pm\infty$, NaN, subnormals, etc.) (30 pts)

```
module fp_flags(pos,zero,neg,int,single);
   input [31:0] single;
   output       pos, zero, neg, int;
```

```
endmodule
```

Problem 3: The `for` loop in the code below looks harmless but is actually an infinite loop. Why? *Hint: It has to do with the way* `i` *is declared.* (10 pts)

```
module iloop(z,a);
   input [31:0] a;
   output       z;

   reg [4:0] i;
   reg       s, z;

   initial begin
      s = 0;
      for(i=0; i<32; i=i+1) s = s | a[i];
      z = !s;
   end

endmodule
```

Problem 4: Consider the adder modules below.(10 pts)

(a) What kind of adders are these?

(b) How do the speed of the two adders compare?

(c) Compare the amount of hardware that the adders will synthesize into. How is the second adder penny wise and £ foolish?

```
module add_1(sum,a,b,clk);
   input [31:0] a,b;   input        clk;   output       sum;
   reg [31:0]   sum;   integer      i;     reg          carry;

   always @( posedge clk )
     begin
        carry = 0;

        for(i=0; i<31; i=i+1) begin

           sum[i] = ~a[i] & ~b[i] &  carry |
                    ~a[i] &  b[i] & ~carry |
                     a[i] & ~b[i] & ~carry |
                     a[i] &  b[i] &  carry;

           carry = a[i] & b[i] | b[i] & carry | a[i] & carry;

        end
   end

endmodule

module add_2(sum,a,b,clk);
   input [31:0] a,b;   input        clk;   output       sum;
   reg [31:0]   sum;   integer      i;     reg          carry;

   always @( posedge clk )
     begin
        i = i + 1;
        if( i == 32 ) begin carry = 0; i = 0; end

        sum[i] = ~a[i] & ~b[i] &  carry |
                 ~a[i] &  b[i] & ~carry |
                  a[i] & ~b[i] & ~carry |
                  a[i] &  b[i] &  carry;

        carry = a[i] & b[i] | b[i] & carry | a[i] & carry;
     end

endmodule
```

Problem 5: Consider the module below. (10 pts)

```
module prefix_xor_4(x,a);
   input [3:0] a;
   output [3:0] x;

   assign      x[0] = a[0];

   xor x1(x[1],a[0],a[1]);
   xor x2(x[2],x[1],a[2]);
   xor x3(x[3],x[2],a[3]);

endmodule
```

(a) Suppose that each gate has a delay of one unit. How long would it take to compute the result?

(b) Suppose during a run of the simulator on the code above new inputs arrived at $t = 100$. At what simulated time would the results be available? *Hint: The first part is intentionally misleading.*

(c) How would timing obtained after synthesis relate to the times used to solve the first two parts?

Problem 6: In the module below fill in the values for c, whether the corresponding addition overflowed, and fix the last assignment. (10 pts)

```
module sums();

   reg [3:0] a, b, c;
   reg [5:0] d;

   initial begin

      a = 4'b0101;  b = 4'b0001;  c = a + b;

      // Unsigned decimal: c =             Overflow?

      // Signed decimal:   c =             Overflow?


      a = -6; b = 4'b0001;  c = a + b;

      // Unsigned decimal: c =             Overflow?

      // Signed decimal:   c =             Overflow?


      a = -6; b = 4'b0001;  c = a + b;

      // Unsigned decimal: c =             Overflow?

      // Signed decimal:   c =             Overflow?


      a = 4'b1101; b = 4'b1100;  c = a + b;

      // Unsigned decimal: c =             Overflow?

      // Signed decimal:   c =             Overflow?


      // Suppose c and d are used for signed quantities.
      // Fix the assignment below.
      d = c;

   end

endmodule
```

Problem 7: Convert the module below to an explicit structural form. (10 pts)

```
module to_str(x,s,a,b);
   input [1:0] s;
   input       a, b;
   output      x;

   assign x = s == 2 ? a : b;

endmodule
```

Problem 8: Show the longhand steps needed to multiply $00100111_2 \times 00100111_2$ using radix-4 Booth recoding. (10 pts)