Name _____

Computer Organization
EE 3755
Midterm Examination

Wednesday, 24 October 2012, 9:30–10:20 CDT

Problem 1	_____	(15 pts)
Problem 2	_____	(14 pts)
Problem 3	_____	(14 pts)
Problem 4	_____	(14 pts)
Problem 5	_____	(18 pts)
Problem 6	_____	(11 pts)
Problem 7	_____	(14 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: [15 pts] Consider the module below:

```
module module1(x,y,a,b,c);
   input a, b, c;    output x, y;
   wire   t1, an;

   not n1(an, a);
   and a1(t1, an, b);
   or  o1(x, t1, c);
   and a2(y, t1, c);
endmodule
```

(*a*) Draw a logic diagram corresponding to the module. Don't optimize.

☐ Logic diagram of `module1`.

(*b*) Complete the module below so that it performs the same operation as `module1` but is in implicit structural form.

☐ Implicit structural form of `module1`.

```
module module1(x,y,a,b,c);
   input a, b, c;    output x, y;




endmodule
```

Problem 2: [14 pts] Appearing below is a Verilog description of a two-level adder used in Homework 2.

```
module adder_r4_c3(sum,a,b);
   input [11:0]  a, b;                    output [12:0] sum;
   wire [2:0]    P, G, carry, CO;

   ripple_4_block ad0(sum[3:0],  CO[0], a[3:0],  b[3:0],  carry[0]);
   ripple_4_block ad1(sum[7:4],  CO[1], a[7:4],  b[7:4],  carry[1]);
   ripple_4_block ad2(sum[11:8], CO[2], a[11:8], b[11:8], carry[2]);

   gen_prop_4 gp0(G[0], P[0], a[3:0],  b[3:0]);
   gen_prop_4 gp1(G[1], P[1], a[7:4],  b[7:4]);
   gen_prop_4 gp2(G[2], P[2], a[11:8], b[11:8]);

   assign      carry[0] = 1'b0;
   assign      carry[1] = G[0];
   assign      carry[2] = G[0] & P[1] | G[1];
   assign      sum[12]  = G[0] & P[1] & P[2] | G[1] & P[2] | G[2];
endmodule
```

(*a*) Suppose that the input to the adder were `a=0x123` and `b=0xabc`. Show the values on the wires indicated below: *Hint: If you don't remember the formulas for* `P` *and* `G`*, try to remember what they are supposed to do.*

☐ Wire `a[3:0]`

☐ Wire `b[7:4]`

☐ Wire `P[2]`, `P[1]`, and `P[0]`

☐ Wire `G[2]`, `G[1]`, and `G[0]`

Problem 3: [14 pts] Appearing below is Verilog code for a carry lookahead adder which computes the carry signals two ways. First is the conventional Old way, the second computes the same values for the carry signals but the Verilog expression is much shorter. (Code for computing `carry0` to `carry3` is not shown.) An actual module would use one of these ways, not both.

```
// Old Way
assign carry4 = g0 & p1 & p2 & p3  |  g1 & p2 & p3  |  g2 & p3  |  g3;
assign carry5 = g0 & p1 & p2 & p3 & p4  |  g1 & p2 & p3 & p4
                | g2 & p3 & p4  |  g3 & p4  |  g4;

// "New" Way:
assign carry4 = carry3 & p3 | g3;
assign carry5 = carry4 & p4 | g4;
```

(*a*) Compute the amount of time it will take to compute `carry5` using the Old and "New" ways. The `p` and `g` signals are available at $t = 0$. All gates have a delay of one time unit, regardless of the number of inputs. Don't forget to account for `carry0`, `carry1`, ..., `carry3` when necessary, they **are not** available at $t = 0$. Show work or explain your answers.

☐ Time for `carry5` using Old way:

☐ Time for `carry5` using "New" way:

(*b*) There is an advantage to "New" way (and it's not performance). However, *"New" way* is not the best name.

☐ Explain the advantage of "New" way.

☐ Calling it the *"New"* way of building a carry lookahead adder is misleading because...

Problem 4: [14 pts] Appearing below is the streamlined multiplier used in Verilog note set 7. Suppose that the multiplier is used with `multiplicand=0x8a` and `multiplier=0x85`. In the table below show the values in registers `bit` and `product` when execution reaches the indicated places in the code. The table is already filled in for the `INIT` block row, complete the four `PP` rows. Note that it takes 16 iterations to compute the product, so don't expect `product` to be $8a_{16} \times 85_{16} = 47b2_{16}$ in the fourth `PP` row.

```
module streamlined_mult(product,ready,multiplicand,multiplier,start,clk);
   input [15:0]  multiplicand, multiplier;
   input         start, clk;
   output        product, ready;
   reg [31:0]    product;            reg [4:0]     bit;
   wire          ready = !bit;
   initial bit = 0;

   always @( posedge clk )
     if ( ready && start ) begin:INIT   //   <-  THE INIT BLOCK
        bit     = 16;
        product = { 16'd0, multiplier };
        //  VALUES SHOWN WHEN EXECUTION REACHES HERE, IN THE INIT BLOCK
     end else if ( bit ) begin:PP      //   <-  THE PP BLOCK
        reg lsb;
        lsb     = product[0];
        product = product >> 1;
        bit     = bit - 1;
        if ( lsb ) product[31:15] = product[30:15] + multiplicand;
        //  SHOW VALUES WHEN EXECUTION REACHES HERE, IN THE PP BLOCK
     end
endmodule
```

☐ Show values for `bit` and `product` in the four `PP` rows table below.

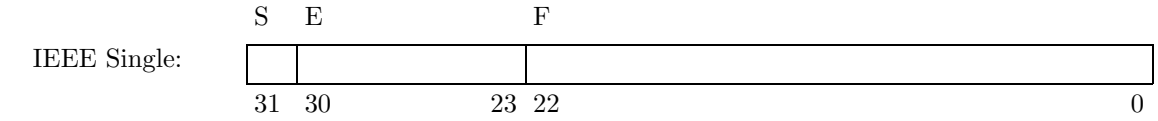
location	bit	product
INIT	16	0x 0000 0085
PP	15	
PP		
PP		
PP		

Problem 5: [18 pts] Answer the computer arithmetic questions below.

(*a*) Show the longhand steps needed to multiply $6c_{16} \times 39_{16}$ using a radix-4 (two bit) multiplication algorithm, but do not add together the partial products. Show the work in binary or hexadecimal. This is **without** Booth recoding. (The product is $180c_{16}$, but remember there is no need to add the partial products.)

☐ Longhand steps for radix-4 (2 bit) $6c_{16} \times 39_{16}$:

(*b*) Show the value of IEEE 754 single-precision floating point number `0x41810000`. For your convenience the layout of an IEEE 754 single is shown below. *Note: In the original version of this exam, the encoded number was* `0x41810000`*, which was much smaller than intended.*



☐ Value of number (in decimal or as a formula to compute value) is:

Problem 6: [11 pts] Show the values in register `$s0` after the execution of each instruction below in the spaces indicated.

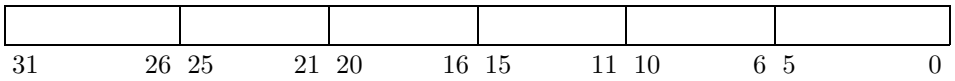
☐ Fill in the `s0 =` blanks below.

```
 # Initial register values:  $s1 = 10,  $s2 = 20

add $s0, $s1, $s2
                    # $s0 =

and $s0, $s1, $s2
                    # $s0 =

sll $s0, $s1, 2
                    # $s0 =

slt $s0, $s1, $s2
                    # $s0 =

addi $s0, $s1, 30
                    # $s0 =

lui $s0, 0x4321
                    # $s0 = ;
```

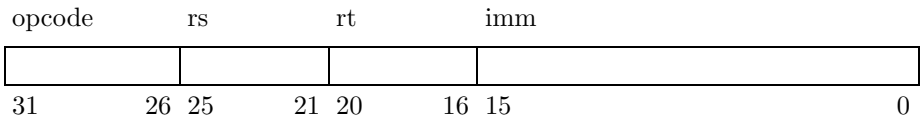
Problem 7: [14 pts] Answer each MIPS encoding question below.

(*a*) Show the encoding of each assembly language instruction below. Some field values would have to be looked up in a table, for those write "look up" instead of the value. For your convenience the layout of the MIPS R and I formats are shown, answers can be written in these or they can be copied.

Format R:



Format I:



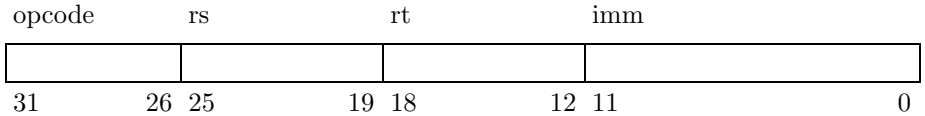
☐ Encoding for `add $5, $6, $7`

☐ Encoding for `addi $8, $9, -1`

☐ Encoding for `sll $10, $11, 12`

(*b*) Shown below is instruction format XI, a format for XIPS, an instruction set similar to MIPS, but with significant differences. Describe two differences of XIPS from MIPS in terms directly relevant to an assembly language programmer. That is, don't just answer that the `rs` and `rt` fields are two bits larger than in MIPS.

Format XI:



☐ Difference of XIPS relevant to a assembly language programmer: