This exam is open book and open notes. You have 50 minutes. Credit for problems requiring calculation will be given only if you show your work.

1. Performance 20 Points

Two processor designs M1 and M2 implement the same architecture but with different performance characteristics, as shown below:

<table>
<thead>
<tr>
<th>Machine</th>
<th>CPI</th>
<th>Clock Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>3.2</td>
<td>1.0 GHz</td>
</tr>
<tr>
<td>M2</td>
<td>4</td>
<td>2.0 GHz</td>
</tr>
</tbody>
</table>

When we run benchmark program P it executes 20 Million instructions.

(a) How many clock cycles will it take for program P to execute on machine M1?

(b) How many clock cycles will it take for program P to execute on machine M1?

(c) What will be the execution time of program P be on machine M1?

(d) What will the execution time of program P be on machine M2?

(e) Which machine is faster, and by how much?

2. MIPS Instructions 20 Points

Consider the sequence of three instructions shown below.

```
loop1:  lbu  $8, 0($4)
        add  $4, $4, $5
        bne  $8, $zero, loop1
```

In the space provided on the next page, show the binary representation of each of the three instructions in this sequence. Clearly mark the instruction fields and show their decimal equivalent. (Information about the lbu instruction can be found on page A-66 of the book).
3. Instructions – Assembly Language 40 Points

The assembly language program shown below operates on two strings stored using the C convention discussed in class. The argument registers $a0 and $a1 initially contain the base addresses of these two strings, i.e. $a0=0x10010000 and $a1=0x10010009. Note that the “sb” instruction stores a single byte from a register into memory and is otherwise similar to the “sw” instruction discussed in class.

```
loop1:  lbu $t0, 0($a0)
        addi $a0, $a0, 1
        bne $t0, $zero, loop1
        sub $a0, $a0, 1

loop2:  lbu $t0, 0($a1)
        sb  $t0, 0($a0)
        addi $a0, $a0, 1
        addi $a1, $a1, 1
        bne $t0, $zero, loop2

done:   # end of program
```

The following diagram shows the contents of memory (since this program processes strings, memory is shown in a “byte” organization). Memory contents are shown as their printable equivalent in ASCII, with ‘\0’ denoting the NULL (all-zero) byte.
(a) Fill in the diagram above to show the contents of the memory after the loop labeled “Loop1” has completed (just before the sub instruction executes). Also show where registers $a0 and $a1 point in memory. How many times does this loop execute?

(b) Fill in the diagram above to show the contents of the memory after the loop labeled “Loop1” has completed. Also show where registers $a0 and $a1 point in memory. How many times does this loop execute?

(c) What string function does this code perform?

(d) Is this code vulnerable to buffer overflow? Why or why not?
4. **Verilog Coding**  

Consider the Verilog code shown below:

```verilog
module arith(a, b, c, d, y);
    input  [15:0] a, b, c, d;
    output [15:0] y;

    wire   [15:0] t1, t2;
    assign t1 = (a + b) >> 1;
    assign t2 = (c + d) >> 1;
    assign y = (t1 + t2) >> 1;
endmodule
```

(a) Draw a block diagram showing the inputs and outputs of this circuit.

(b) What common arithmetic function is being calculated using inputs a, b, c, and d?

5. **Verilog Coding – EXTRA CREDIT**  

The following is a different version of the same circuit described in problem 4.

```verilog
module arith(a, b, c, d, y);
    input  [15:0] a, b, c, d;
    output [15:0] y;

    assign y = (a + b + c + d) >> 2;
endmodule
```

Which version is better, and why?
6. **Carry Lookahead**  

In a carry-lookahead adder, each bit-slice of the adder must use inputs \( a_i \) and \( b_i \) and \( c_i \) to produce generate function \( g_i \), propagate function \( p_i \), and sum output \( s_i \).

A proposal has been made to replace the standard definition of the propagate function:

\[
p_i = a_i + b_i
\]

With a version that uses the **exclusive or** function:

\[
p_i = a_i \oplus b_i
\]

The carry function is now implemented in the carry-lookahead unit as before:

\[
c_{i+1} = g_i + p_i \cdot c_i
\]

(a) Will the modified propagate function work properly in the calculation of carry signals by the carry lookahead unit? Are any additional modifications needed? Why or why not?

(b) What advantage would there be in using the exclusive-or version of the propagate signal?