This prerequisite quiz will be used in determining class admissions. The use of is not allowed during this quiz. Good Luck!

<table>
<thead>
<tr>
<th>Your Name:</th>
<th>SOLUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID Number:</td>
<td></td>
</tr>
<tr>
<td>Discussion Section:</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
(1) Here is a translation of the above program into MIPS assembly language.

```mips
fib:  addi $v0, $0, 0
     slti $t0, $a0, 1
     bne $t0, $0, end

fib1: addi $sp, $sp, -12
      sw $a0, 4($sp)
      addi $a0, $a0, -1
      j fib

      add $s0, $v0, $0

      lw $a0, 4($sp)
      addi $a0, $a0, -2
      jr fib1

      add $v0, $s0, $v0

end:  lw $ra, 0($sp)
      addi $sp, $sp, 12
      jr $ra
```

There are 8 mistakes in this translation. Fix them in the above listing. Assume that MIPS register conventions must be maintained throughout the execution.

```
Fib:  addi $v0, $0, 1  # 1, return 1
      slti $t0, $a0, 2  # 2, <= 1
      bne $t0, $0, end

Fib1: addi $sp, $sp, -12
      sw $ra, 0($sp)    # 3, must save return address
      sw $a0, 4($sp)
      sw $s0, 8($sp)    # 4, must save “saved” registers before usage
      addi $a0, $a0, -1
      jal fib           # 5, subprocedure calls are jal

      add $s0, $v0, $0

      lw $a0, 4($sp)
      addi $a0, $a0, -2
      jal fib           # 6, another subprocedure call

      add $v0, $s0, $v0
      lw $ra, 0($sp)
      lw $s0, 8($sp)    # 7, must restore “saved” registers
      addi $sp, $sp, 12
      jr $ra            # 8, bne was before stack push so must move label to after the stack pop
```

2) Assume that we have a 16 bit system that uses signed, two’s-complement integers. Perform the following conversions:

-13\textsubscript{10} to base 2: \hspace{1cm} 1111 1111 1111 0011\textsubscript{2}

EFA\textsubscript{16} to base 8: \hspace{1cm} 167642\textsubscript{8}

34\textsubscript{10} to base 16: \hspace{1cm} 22\textsubscript{16}

(111111110011100)\textsubscript{2} to base 10: \hspace{1cm} -196

234\textsubscript{8} to base 10: \hspace{1cm} 156
3)  

a) Complete the Following State Transition Diagram for the VersaCounter’s count output. Ignore the “ZPULSE” signal. Include “HOLD”, and “TYPE” as inputs, and ignore RESET for the moment:

```
H T CS0 CS1 NS0 NS1
0 X 0 0    0  1
1 X 0 0    0  0
0 0 0 1    1  0
0 1 0 1    1  1
1 X 0 1    0  1
0 0 1 0    1  1
0 1 1 0    0  0
1 X 1 0    1  0
0 0 1 1    0  0
0 1 1 1    1  0
1 X 1 1    1  1
```
c) Derive Next-State Logic Equations given the state transition table. Include the RESET signal in your equations.

\[
\begin{align*}
NS0 &= H C0 + \overline{H} T \overline{C0} + \overline{H} T \overline{C1} \\
NS1 &= H C1 + \overline{H} T C0 + \overline{H} C1 C0 + \overline{T} C1 \overline{C0}
\end{align*}
\]

Construct a Karnaugh map(s) to minimize the next-state logic equations. Derive the resulting simplified equations.

\[
\begin{array}{c|cccc}
\text{C0} & \text{C1} \\
\hline
\text{HT} & 00 & 01 & 11 & 10 \\
00 & 1 & 0 & 0 & 1 \\
01 & 1 & 0 & 0 & 0 \\
11 & 0 & 1 & 1 & 0 \\
10 & 0 & 1 & 1 & 0
\end{array}
\]

\[
\begin{array}{c|cccc}
\text{CS0} & \text{CS1} \\
\hline
\text{NS1} & 00 & 01 & 11 & 10 \\
00 & 0 & 1 & 0 & 1 \\
01 & 0 & 1 & 1 & 0 \\
11 & 0 & 0 & 1 & 1 \\
10 & 0 & 0 & 1 & 1
\end{array}
\]

d) Draw a small circuit that will take a clock and the 2-bit count value as input and will output the “ZPULSE” signal. ZPULSE is high for the very first cycle that the 2-bit count is zero, and deasserted otherwise. *Make this as simple as possible.* You may use falling-edge triggered flip-flops, inverters, and 2, 3, or 4-input NAND gates. Don’t forget to include the RESET signal. ZPULSE may be asserted during reset, but should definitely be asserted for the first cycle after reset. (This should *not* be hard to do).

The trick is to capture the state change to 0 only once which means the 0 1 transition of C0 AND C1.

![Circuit Diagram]

C0

C1

CLK

D

Q

\(
\overline{Q}
\)

ZPULSE
e) Implement the complete VersaCounter using falling-edge triggered flip flops, inverters, and 2, 3, or 4-input NAND gates. Minimize the number of gates that you use. Clearly label input and output signals. It should have as input: CLOCK, HOLD, TYPE, and RESET. It should have as output a 2-bit count value and a 1-bit “ZPULSE” value.

Simple translation of AND/OR stages into NAND/NAND stages
4) Assume that we have a 32-bit processor with 32-bit words, and that this processor is byte-addressed (i.e. addresses specify bytes). Now, suppose that this processor has a 32-word, direct-mapped cache with 2-word cache lines.

a) Split the 32-bit address into “tag”, “index”, and “cache-line offset” pieces. We start you off with one of these.

   tag:  31-7
   index: 6-3
   cache-line offset:  bits 2 – 0

b) Assume that the processor makes a series of byte accesses to memory (shown to the left below). Label each memory reference address as a Hit (H) or a Miss (M). Also, show the final contents of the cache in the table at the right. Fill in values for the TAG and words of the cache. Use the notation "M[ij]" to mean the word in memory at address i (keep these word aligned!). Include no more than one value per word (words can be left blank).

<table>
<thead>
<tr>
<th>Reference Address</th>
<th>Hit or Miss?</th>
</tr>
</thead>
<tbody>
<tr>
<td>138₁₀ (08A₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>60₁₀ (03C₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>56₁₀ (038₁₆)</td>
<td>H</td>
</tr>
<tr>
<td>140₁₀ (08C₁₆)</td>
<td>H</td>
</tr>
<tr>
<td>419₁₀ (1A3₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>484₁₀ (1E4₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>142₁₀ (08E₁₆)</td>
<td>H</td>
</tr>
<tr>
<td>392₁₀ (188₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>175₁₀ (0AF₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>140₁₀ (08C₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>344₁₀ (158₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>487₁₀ (1E7₁₆)</td>
<td>H</td>
</tr>
<tr>
<td>396₁₀ (18C₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>376₁₀ (178₁₆)</td>
<td>M</td>
</tr>
<tr>
<td>380₁₀ (17C₁₆)</td>
<td>H</td>
</tr>
<tr>
<td>160₁₀ (0A0₁₆)</td>
<td>M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>TAG Value</th>
<th>Word 0</th>
<th>Word 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x3</td>
<td>M[0x188]</td>
<td>M[0x18C]</td>
</tr>
<tr>
<td>2</td>
<td>0x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x1</td>
<td>M[0x0A0]</td>
<td>M[0x0A4]</td>
</tr>
<tr>
<td>4</td>
<td>0x1</td>
<td>M[0x0A8]</td>
<td>M[0x0AC]</td>
</tr>
<tr>
<td>5</td>
<td>0x0</td>
<td>M[0x038]</td>
<td>M[0x03C]</td>
</tr>
<tr>
<td>6</td>
<td>0x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0x0</td>
<td>M[0x158]</td>
<td>M[0x15C]</td>
</tr>
<tr>
<td>8</td>
<td>0x3</td>
<td>M[0x1E0]</td>
<td>M[0x1E4]</td>
</tr>
<tr>
<td>9</td>
<td>0x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0x2</td>
<td>M[0x178]</td>
<td>M[0x17C]</td>
</tr>
<tr>
<td>C</td>
<td>0x3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0x2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


c) Calculate the cache hit rate (you can leave as a fraction).