Name:

Problem 1 points earned:

Problem 2 points earned:

Problem 3 points earned:

Problem 4 points earned:

Problem 5 points earned:

Total:
### Instructions from Text and Notes for Exam 1

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add $2, $3, $4</td>
<td>$2 = $3 + $4</td>
</tr>
<tr>
<td>addi</td>
<td>addi $2, $3, 10</td>
<td>$2 = $3 + 10 (signed constant)</td>
</tr>
<tr>
<td>addiu</td>
<td>addiu $2, $3, 10</td>
<td>$2 = $3 + 10 (overflow undetected and signed constant)</td>
</tr>
<tr>
<td>sub</td>
<td>sub $2, $3, $4</td>
<td>$2 = $3 - $4</td>
</tr>
<tr>
<td>subu</td>
<td>subu $2, $3, $4</td>
<td>$2 = $3 - $4 (overflow undetected)</td>
</tr>
<tr>
<td>lw</td>
<td>lw $2,8 ($3)</td>
<td>load word at address $3 + 8 into $2</td>
</tr>
<tr>
<td>lb</td>
<td>lb $2,8 ($3)</td>
<td>load byte at address $3 + 8 into $2</td>
</tr>
<tr>
<td>lui</td>
<td>lui $3, 2</td>
<td>load 2 into upper 16 bits of $3</td>
</tr>
<tr>
<td>sw</td>
<td>sw $2,8 ($3)</td>
<td>store word in $2 to word at address $3 + 8</td>
</tr>
<tr>
<td>sb</td>
<td>sb $2,8 ($3)</td>
<td>store byte in $2 to byte at address $3 + 8</td>
</tr>
<tr>
<td>sll</td>
<td>sll $2, $3, 10</td>
<td>$2 = $3 $&lt;&lt; 10</td>
</tr>
<tr>
<td>sllv</td>
<td>sllv $2, $3, 10</td>
<td>$2 = $3 $&lt;&lt; 10</td>
</tr>
<tr>
<td>srl</td>
<td>srl $2, $3, 10</td>
<td>$2 = $3 $&gt;&gt; 10</td>
</tr>
<tr>
<td>srlv</td>
<td>srlv $2, $3, 10</td>
<td>$2 = $3 $&gt;&gt; 10</td>
</tr>
<tr>
<td>and</td>
<td>and $2, $3, $4</td>
<td>$2 = $3 &amp; $4</td>
</tr>
<tr>
<td>andi</td>
<td>andi $2, $3, 15</td>
<td>$2 = $3 &amp; 15 (sign extended constant)</td>
</tr>
<tr>
<td>or</td>
<td>or $2, $3, $4</td>
<td>$2 = $3</td>
</tr>
<tr>
<td>ori</td>
<td>ori $2, $3, 15</td>
<td>$2 = $3</td>
</tr>
<tr>
<td>slt</td>
<td>slt $2, $3, $4</td>
<td>if($3 &lt; $4) $2 = 1 else $2 = 0</td>
</tr>
<tr>
<td>slti</td>
<td>slti $2, $3, 5</td>
<td>if($3 &lt; 5) $2 = 1 else $2 = 0 (signed constant)</td>
</tr>
<tr>
<td>sltiu</td>
<td>sltiu $2, $3, 5</td>
<td>if($3 &lt; 5) $2 = 1 else $2 = 0 (unsigned constant)</td>
</tr>
<tr>
<td>beq</td>
<td>beq $3, $4, Label</td>
<td>if($3 = $4) go to Label</td>
</tr>
<tr>
<td>bne</td>
<td>bne $3, $4, Label</td>
<td>if($3 != $4) go to Label</td>
</tr>
<tr>
<td>j</td>
<td>j Label</td>
<td>go to Label</td>
</tr>
<tr>
<td>jr</td>
<td>jr $31</td>
<td>go to instruction at address in $31 (return from function)</td>
</tr>
<tr>
<td>jal</td>
<td>jal Label</td>
<td>save address of next instruction and go to Label</td>
</tr>
</tbody>
</table>

### Decimal, Hexadecimal and Binary Basic Encoding

<table>
<thead>
<tr>
<th>Dec.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Bin.</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>Dec.</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>HEX</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>Bin.</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
<td>1110</td>
<td>1111</td>
</tr>
</tbody>
</table>
Problem 1: Addressing (10 points)

Recall the MIPS ISA has 5 different addressing modes: register addressing, base or displacement addressing, immediate addressing, PC-relative addressing, and pseudodirect addressing.

For any 3 out of the 5 give examples of MIPS instructions that use the modes and explain the addressing mode briefly.

**Solutions:**

- **Immediate addressing:** the value of the operand is encoded directly in the instruction. Example – addi $2,$3,10
- **Register addressing:** the location of the operand is given by the register number which is encoded in the instruction, the contents of the register is the value of the operand. Example – add $2,$3,$4
- **Base or displacement addressing:** The address of the operand in memory is given by the sum of the contents of a register (whose number is encoded in the instruction) and an immediate operand (whose value is encoded in the instruction). Example – lw $2,8 ($3)
- **PC-relative addressing:** The address of the next instruction to be executed next is given by the sum of the address of the next instruction in memory (PC +4) and an immediate operand (whose value is encoded in the instruction). Example – beq $3,$4, Label
- **Pseudodirect addressing:** The address of the next instruction to be executed next is given an immediate operand (whose value is encoded in the instruction). Example – j Label
Problem 2: Performance (25 points)

(a) 15 points Assume we have the following time, performance and architecture parameters in the specified units.

\[ E_c = \text{execution time in cycles} \]  
\[ E_s = \text{execution time in secs} \]  
\[ P_s = \text{performance rate in instructions/second} \]  
\[ CT = \text{cycle time in sec/cycle} \]  
\[ CR = \text{clock rate in cycle/sec} \]  
\[ I = \text{instructions in program} \]  
\[ CPI = \text{average cycles per instruction in cycle/instruction} \]

Complete the following formulas with the appropriate parameter or show that it is not possible using the supplied parameters:

**Solutions:**

Each of the expressions below have terms with specific units. By examining those units you can identify the units of the parameters that must appear in the blanks. On the third equation, it was also acceptable to say it was not possible by simple substitution of the parameters above, i.e., if you assumed that you could not take a reciprocal and then substitute, as long as a reasonable argument was given based on the units.

\[ E_s = \frac{I}{cycle} \times CPI \times CT \]

\[ E_c = I \times CPI \]
\[ P_s = \frac{CPI^{-1}}{CR} \]

(b) 10 points

Suppose the workload on your machine spends 90% of the time in floating point arithmetic. A vendor proposes that you upgrade your machine to perform the floating point arithmetic 1000 times faster. Since you are such a good customer the vendor suggests a price of only 100 times the price of your current system. Do you buy the upgrade? Justify your answer.

**Solutions:** From Amdahl’s law it can be shown that the maximum speedup is \( S_{\text{max}} = 10 \). This is due to the fact that the smallest time possible, i.e., if floating point arithmetic was free, is \( T_{\text{enhanced}} = 0.1 \times T_{\text{original}} \). A price increase by a factor of 100 is therefore probably not a good buy.
Problem 3: (20 points)

Recall the three MIPS instruction 32-bit encoding formats (number of bits in each field in parentheses):

<table>
<thead>
<tr>
<th>Format</th>
<th>op (6)</th>
<th>rs (5)</th>
<th>rt (5)</th>
<th>rd (5)</th>
<th>shamt (5)</th>
<th>funct (6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solutions:
(a) 10 points Given that the op code for *lw* is 35 (decimal) encode the following instruction:

\[ lw \$2, 4(\$3) \]

- Give the decimal values for each field. 35 3 2 4
- Give the binary values for each field.
  \[ 100011 \ 00011 \ 00010 \ 00000000000000100 \]
- Give the hexadecimal representation of the entire word. \[ 8C620004 \]

(b) 10 points What is the maximum number of operations (i.e., add, sub, and, or, ...) requiring three register operands that can be encoded using the 32-bit MIPS encoding scheme? Justify your answer.

**Solutions:** Since 3 register operands is the R type and the opcode for the R type is by definition all 0, the number must be determined by the field that distinguishes between R type instructions, i.e., the *funct* field which has 6 bits. Therefore the maximum distinct R type operations is \( 2^6 \).
Problem 4: (20 points)

Consider the following MIPS code. Describe what the code does. In particular, include in the description what is computed and written to the variable s. Your description may include C code if you wish.

```
# When entering the function
# you may assume that
# a positive integer m is in $7
# a positive integer n is in $8
# a word aligned address of z[0]
# is in $9 (the dimension of array z is k > n)
# and a word aligned address of s
# is in $10

funct:
    addi $2, $0, 0  # initialize s to 0 (s in $2)
    addi $3, $0, 0  # initialize i to 0 (i in $3)
    add $4, $0, $9  # byte address of z[0] into $4
again:
    slt $7, $3, $8  # true if i < n
    beq $7, $0, endit # branch to endit on false (i >= n)
        # so we iterate for all 0 <= i < n
    lw $5, ($4)  # load z[i] into $5
    slt $11, $5, $7  # check if z[i] < m
    addu $2, $2, $11  # increment $s (s) by 1 if test satisfied
    addiu $4, $4, 4  # address of z[i+1] into $4p
    addi $3, $3, 1  # i++ ($3 = i + 1)
    j again
endit:      jr $31  # return to caller
```

# When entering the function
# you may assume that
# a positive integer m is in $13
# a positive integer n is in $8
# a word aligned address of z[0]
# is in $9 (the dimension of array z is k > n)
# and a word aligned address of s
# is in $10
funct:
    addi $2,$0,0       # initialize s to 0 (s in $2)
    addi $3,$0,0       # initialize i to 0 (i in $3)
    add $4,$0,$9       # byte address of z[0] into $4
again:  slt $7,$3,$8  # true if i < n
    beq $7,$0,wha     # branch to end it on false (i >= n)
    # so we iterate for all 0 <= i < n
    lw $5,($4)        # load z[i] into $5
    slt $11,$5,$13     # check if z[i] < m
    addu $2,$2,$11     # increment $2 (s) by 1 if test satisfied
    addiu $4,$4,4      # address of z[i+1] into $4
    addi $3,$3,1       # i++ ($3 = i + 1)
    j again
wha:   sw $2,($10)   # write result to s
    jr $31            # return to caller

Solutions:
This code scans the array z from z[0] to z[n-1] and counts the number of elements found that are less than the value of m. The number is written to the variable s.
Problem 5: (25 points)

For this problem assume that the 32 bits in a word or register are denoted \((b_{31} b_{30} \cdots b_1 b_0)\) from most significant to least significant.

Recall that MIPS provides pseudoinstructions that are replaced by the assembler/linker with a sequence of the basic MIPS instructions. One example we have seen is the load address (la) pseudoinstruction used to load the address of a label.

For the pseudoinstruction below, use the MIPS instructions we have seen so far (see the table at the beginning of the exam) to define the sequences that replace the pseudoinstructions. You may not use any registers other that those listed in the pseudoinstruction. After the pseudoinstruction executes the source registers must be unchanged from their original contents and the destination registers must have the results specified below.

The pseudoinstruction

\[ \text{evenu } \$2, \$3 \]

sets \$2 to 1 if the contents of \$3 is an even number when interpreted as an unsigned integer, i.e., a non-negative integer, and sets \$2 to 0 otherwise. The contents of \$3 should be unchanged.

Solutions:

There are many ways to do this of course. The most efficient seems to be two instructions:

\[ \begin{align*}
\text{andi } \$2, \$3, 1 \\
\text{slti } \$2, \$2, 1
\end{align*} \]

The first instruction places 0 in all bits of \$2 except \(b_0\). The second will only set \$2 to 1 if the current contents of \$2 is 0, i.e., if \(b_0 = 0\). This is the functionality desired and it works only in the two registers listed in the pseudoinstruction.

Other two instruction sequences that make use of shifts are also possible.