This exam has five regular questions and one extra-credit question. Complete questions that are easiest for you first, then complete what you can of the difficult questions. There is no penalty for guessing. However, on questions involving calculation, you must show your work. If you do not show your work, you risk getting only partial credit for any answer.

Q1. (20 pts) Define these terms using 1-3 sentences, a formula, or diagram (5 pts each):

(a) MIPS I-format Instruction

![MIPS I-format Instruction diagram](http://www.cs.gmu.edu/~setia/cs365-S02/class3.pdf)

(b) Funct Field in MIPS R-format Instruction

Six-bit field at right-hand end of instruction that modifies the functionality of the op-code.

(c) Jump Target Address

Target address for MIPS Jump (j) instruction (diagram on next page)

(d) **Sign Extension** (in MIPS)

The immediate operand is 16 bits (as are all MIPS immediate operands). However, when extended to a 32-bit operand by the ALU it is **sign extended**: The value of the left-most bit of the immediate operand (bit 15) is copied to all bits to the left (into the high-order bits). So if the 16-bit immediate operand is a 16-bit two's complement negative integer, the 32-bit ALU operand is a 32-bit version of the same negative integer.


(e) **PC-Relative Addressing** (in MIPS)

PC-relative addressing occurs in branch instructions, **beq** and **bne** (and other variations of branch instructions):

\[
PC \leftarrow PC + \text{sign-ext}_{32}(\text{IR}_{15-0}::00)
\]

You take the 16 bit immediate value, add two zeroes to the end (which is the same as shifting it logical left 2 bits). This creates a value that's divisible by 4. Then, you sign-extend it to 32 bits, and add it to the PC.

Q2. (30 pts) Convert the following numbers to the indicated format (SHOW WORK):

2.1) \(-117_{10}\) to two’s complement using a 16-bit representation:

\[
100 = 64 + 32 + 4\quad \text{and}\quad 17 = 16 + 1.
\]

So \(117 = 64 + 32 + 16 + 4 + 1 = 0000\ 0000\ 0111\ 0101\) \(2\)-signedmagnitude

and the one’s complement is \(1111\ 1111\ 1000\ 1010\) \(1s\)-complement

thus the 2’s complement is \(1111\ 1111\ 1000\ 1011\) \(2s\)-complement

2.2) \(A3D4_{16}\) (which is in signed-magnitude) to one’s complement binary format:

\(A3D4_{16} = 1010\ 0011\ 1101\ 0100\) \(SM\) (obtained by converting \(A_{16} = 1010_2\), etc.)

Invert:

\(0101\ 1100\ 0010\ 1011\) \(1s\) complement

2.3) \(-1317_8\) to two’s complement using a 20-bit representation:

Given:

\[x = 1111\ 1111\ 1101\ 0011\ 0000\] \(1s\) complement + 1

Invert & Add 1:

\[1111\ 1111\ 1000\ 1000\ 0001\] \(2s\) complement

Q3. (40 pts) A MIPS program has an integer array \(A[n]\) with base address in register \(s0\). The register \(s1\) is used for a loop index and the loop limit \(n\) is stored in register \(s2\).

YOU CAN ONLY USE THE BNE AND BEQ BRANCH INSTRUCTIONS.

3.1) Code this in MIPS:

\[\text{for } i = 0 \text{ to } 39: \{ A[i] = 2 \ast i - A[i] ; \} . \]

DO NOT USE PSEUDOINSTRUCTIONS – COMMENT YOUR CODE

<table>
<thead>
<tr>
<th>Equivalent Simple Code</th>
<th>MIPS Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i = -1)；</td>
<td>addi $s1, $zero, -1</td>
<td># (i \leftarrow -1)</td>
</tr>
<tr>
<td>Loop: (i = i + 1)；</td>
<td>Loop: addi $s1, $s1, 1</td>
<td># (i++)</td>
</tr>
<tr>
<td>(\text{if } i &gt; 39, \text{then goto Exit ;})</td>
<td>slt $s3, $s2, $s1</td>
<td># (n &lt; i)?</td>
</tr>
<tr>
<td>(\text{temp1} = 2 \ast i)；</td>
<td>bne $s3, $zero, Exit</td>
<td># Yes =&gt; Exit</td>
</tr>
<tr>
<td>(\text{offset} = \text{temp1} + \text{temp1} ;)</td>
<td>add $t2, $t1, $t1</td>
<td># (\Delta \leftarrow 4 \ast i)</td>
</tr>
<tr>
<td>(\text{addr} = \text{baseaddr} + \text{offset} ;)</td>
<td>add $t4, $s0, $t2</td>
<td># (\text{addr} \leftarrow \text{base}+\Delta)</td>
</tr>
<tr>
<td>(\text{load temp3 from } M(\text{addr}) ;)</td>
<td>lw $t3, 0($s4)</td>
<td># (t3 \leftarrow M[\text{addr}])</td>
</tr>
<tr>
<td>(\text{temp4} = \text{temp1} - \text{temp3} ;)</td>
<td>sub $t4, $t1, $t3</td>
<td># (t4 \leftarrow t1 - t3)</td>
</tr>
<tr>
<td>(\text{store temp4 at } M(\text{addr}) ;)</td>
<td>sw $t4, 0($s4)</td>
<td># (M[\text{addr}] \leftarrow t4)</td>
</tr>
<tr>
<td>(\text{goto Loop ;})</td>
<td>j Loop</td>
<td># Loopback</td>
</tr>
<tr>
<td>(\text{Exit: } \ldots)</td>
<td>Exit &lt;code&gt;</td>
<td></td>
</tr>
</tbody>
</table>
3.2) Code this in MIPS: i = 0; while i < 47: \{A[i] = i + 33; i++\}.

**DO NOT USE PSEUDOINSTRUCTIONS – COMMENT YOUR CODE**

<table>
<thead>
<tr>
<th>Equivalent Simple Code</th>
<th>MIPS Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 0;</td>
<td>addi $s1, $zero, $zero</td>
<td># i \leftarrow 0</td>
</tr>
<tr>
<td>Loop: if i \geq 47, then goto Exit;</td>
<td>sli $s3, $s1, 47</td>
<td># is i &lt; 47?</td>
</tr>
<tr>
<td>offset = 4 * i;</td>
<td>beq $s3, $zero, Exit</td>
<td># No \Rightarrow Exit</td>
</tr>
<tr>
<td>addr = baseaddr + offset;</td>
<td>sll $t2, $s1, 4</td>
<td># \Delta \leftarrow 4 * s1</td>
</tr>
<tr>
<td>temp1 = i + 33;</td>
<td>add $s4, $s0, $t2</td>
<td># of \leftarrow base+\Delta</td>
</tr>
<tr>
<td>store temp1 at M(addr);</td>
<td>addi $t1, $s1, 33</td>
<td># t1 \leftarrow i + 33</td>
</tr>
<tr>
<td>i++;</td>
<td>sw $t1, 0($s4)</td>
<td># A[i] \leftarrow t1</td>
</tr>
<tr>
<td>goto Loop;</td>
<td>addi $s1, $s1, 1</td>
<td># s1 \leftarrow s1 + 1</td>
</tr>
<tr>
<td>Exit: …</td>
<td>j Loop</td>
<td># Loopback</td>
</tr>
</tbody>
</table>

PRINT YOUR NAME: _______ KEY _________  YOUR CDA 3101 NUMBER: _______

Q4. (30 pts) Tell what each MIPS addressing mode (R, I, J) is used for, and give one example of each mode (10 pts each):

**Grading notes for this question:** addressing modes are techniques for storing the location (address) of data operands in instructions. To get full credit for each part, students needed describe how an effective address is produced from the data stored in the instruction using the addressing mode specified. (As shown on Slide 13 from the Sep. 13 lecture, which actually includes a yellow hint that states this will be on the exam.) Students who only gave an example of each addressing mode received a maximum of 3 points per part. Students who only copied the diagrams from Slide 13, with no accompanying explanation, also received partial credit.

4.1) Direct or Register Addressing:

**Use (7 pts):** Register addressing is used to specify the address (number) of a register directly in an instruction. The data in that register is used as an operand to the instruction. In MIPS, there are 32 registers, which require 5 bits to unambiguously address. Since the entire 5 bit register number is stored in the instruction, Register Addressing is sometimes called Register Direct Addressing.

**Example (3pts):** *(Many examples can be used here, including just “add $s0 $s0 $s0”. Students who only gave an example received 3 pts for their score. The following example of register addressing was the most commonly given.)*

Register addressing is used in the \texttt{jr} instruction. Because a register stores 32 bits, and because an address in a MIPS CPU is also 32 bits, you can specify any address in memory. The typical call is:

\[ \texttt{jr } \$rs \]

where \$rs is replaced by any register. The semantics of this is:
PC $\leftarrow R[s]

This means the PC (program counter) is updated with the contents of register $s$. Recall that a jump or branch is updated by modifying the contents of the program counter.

4.2) Pseudo-Direct Addressing:

Use (7 pts): Direct addressing specifies a complete address in the instruction itself. However, since MIPS instructions are 32 bits, we can't do that. In theory, you only need 30 bits to specify the address of an instruction in memory. However, MIPS uses 6 bits for the opcode, so there's still not enough bits to do true direct addressing.

Instead, we can do pseudo-direct addressing. This occurs in [jump] $j$ instructions.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{31:26}$</td>
<td>$B_{25:0}$</td>
</tr>
</tbody>
</table>

26 bits are used for the target. This is how the address for pseudo-direct addressing is computed.

PC $\leftarrow$ PC$_{31:28}$::IR$_{25:0}$::00

Take the top 4 bits of the PC, concatenate that with the 26 bits that make up the target, and concatenate that with 00. This produces a 32 bit address. This is the new address of the PC.

Example (3pts): $j$ TOP

4.3) Base Addressing:

Use (7pts): Base addressing is used to represent a memory address in an instruction that occurs at the contents of a register + some offset, which is included in the instruction. In MIPS, the register value occupies 5 bits of the instruction, and is added to a 16 bit immediate offset (instruction bits 0 .. 15) to produce the address.

Example (3pts): (Full credit given for writing a complete MIPS load or store instruction. Explanation not required.)

lw $rt$, offset($rs$)

where $rs$ and $rt$ are any two registers. The offset is stored in 16 bits 2C. Thus, lw and sw are I-type instructions. The address computed is:

addr $\leftarrow R[s] + \text{sign-ext}_{32}(\text{offset})$

$rs$ is the base register, which is where the name base addressing comes from.

The offset is the 16 bit immediate value from the instruction. Unlike branch instructions, we don't add a 00 to the end of the immediate value, even though we can only load and store are word-aligned addresses.

The reason is because there are other load/store instructions that load/store halfwords and bytes, and it makes sense to compute the addresses the same way, regardless of what you're loading.

(Some content from http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Mips/addr.html)

Q5. (20 pts) This question has two parts: 5.1 and 5.2. Show all work for full credit.

5.1) Express $-0.012345_{10}$ in IEEE 754 single precision format:
a) Decimal normalization: \(-0.012345_{10} = -1.2345 \times 10^{-2}\)

b) Exponent transformation: \(10^{-2} = 2^{(2/\log_{10}(2))} \approx 2^{0.6438} \approx 2^{-7} \times 1.28\)

c) Mantissa transformation: \(-1.2345 \times 1.28 \approx -1.5802\)

d) Mixed-radix expression: \(-1.2345 \times 10^{-2} \approx -1.5802_{10} \times 2^{-7}\)

e) Determining significand:

\[
\begin{align*}
0.5802 \times 2 &= 1.1604 \\
0.1604 \times 2 &= 0.3208 \\
0.3208 \times 2 &= 0.6416 \text{ and so forth…}
\end{align*}
\]

Answer: \(-1.5802_{10} \times 2^{-7} = -1.100101001000010101110_2 \times 2^{-7}\)

So:

- Sign bit = 1 (negative number)
- Significand = 100101001000010101110
- Exponent = 0111000 = 120 (-127) \rightarrow -7

IEEE 754: \(1 \ 0111000 \ 100101001000010101110\)

5.2) Express IEEE 754 single precision number \(1001 \ 0101 \ 1110 \ 1101 \ 0001 \ 0000 \ 0010 \ 0111\) in decimal scientific notation (e.g., \(6.02 \times 10^{23}\)):

a) Sign bit: \(1 \rightarrow \text{negative mantissa}\)

b) Exponent: \(0010101 = 43_{10} \rightarrow 2^{-84}\) since \(43 - 127 = -84\)

c) Significand: Expand by powers of 2: \(1 \times 2^{-1} + 1 \times 2^{-2} + \ldots + 1 \times 2^{-23}\)

After renormalizing the exponent and compensating the mantissa, we get:

\(\text{Decimal number} = -9.574902 \times 10^{-26}\)

PRINT YOUR NAME: _______  KEY _________  YOUR CDA 3101 NUMBER: _______

Extra Credit Problem: (20 pts) A histogram is a plot of frequency-of-occurrence. For example a set \(\{1,2,1,3,4,8,8\}\) has the histogram \(h = \{(1,2), (2,1), (3,1), (4,1), (8,2)\}\), where \((x,y)\) denotes a number \(x\) and its frequency \(y\).

Write recursive MIPS code to compute and print a histogram of integers in \(N\)-element array \(A\) whose base address is in register $s0$ and length \(N\) in register $s1$. Assume a library routine \(HPrint(N, h(N))\) prints each entry in the histogram (don’t worry how it works). FIRST write a short explanation of how your program works (2-3 sentences), describing your base case and recursive case (pseudocode is ok). SECOND, state all register assignments, THIRD, COMMENT MIPS CODE TO GET FULL CREDIT.