Each question assumes that some MIPS code has run (you will find it on the last page of this exam). It declares a set of memory locations (you’ll see their names and types, but not their contents). It also includes a .text section, which loads some of the variables into registers.

Each question is independent of the other questions - that is, assume that the program has started over from scratch each time.

Save the s Registers
Assume that the ‘s’ registers ($s0, $s1, $s2, …) contain important variables which you should not be destroy. Thus, do not modify any ‘s’ register unless explicitly told to do so.

However, you may modify any ‘t’ register.

Limited MIPS Instructions
Use only the MIPS instructions provided in the table on the next page. Remember, you don’t have a multiply instruction!

Comments Optional
While comments are a good idea (and give us more opportunity to give you partial credit), they are not required on the test.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
Allowed MIPS Instructions
You may use the following MIPS instructions and no others:

- add, addi, sub
- beq, bne, j
- slt, slti
- lw, lh, lb, sw, sh, sb
- la
- syscall

1. (20 points) Put alpha - bravo in register $s4.

Solution:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Field</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>la</td>
<td>$t0, alpha</td>
<td># $t0 = &amp;alpha</td>
</tr>
<tr>
<td>lw</td>
<td>$t0, 0($t0)</td>
<td># $t0 = alpha</td>
</tr>
<tr>
<td>sub</td>
<td>$s4, $t0, $s0</td>
<td># $s4 = alpha - bravo</td>
</tr>
</tbody>
</table>

2. (20 points) Put 3 * bravo - delta into register $s4.

Solution:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Field</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>$t0, $s0, $s0</td>
<td># $t0 = 2 * bravo</td>
</tr>
<tr>
<td>add</td>
<td>$t0, $t0, $s0</td>
<td># $t0 = 3 * bravo</td>
</tr>
<tr>
<td>sub</td>
<td>$s4, $t0, $s2</td>
<td># $s4 = 3 * bravo - delta</td>
</tr>
</tbody>
</table>
3. (30 points) If $\text{delta} + \text{echo}$ is less than $\text{foxtrot}$, then put $\text{delta} + \text{echo}$ into memory location $\text{charlie}$; if not, then store $\text{foxtrot}$ into memory location $\text{charlie}$.

Solution:

```
la $t0, echo          # t0 = &echo
lw $t0, 0($t0)       # t0 = echo
add $t0, $s2, $t0    # t0 = delta + echo
la $t3, charlie      # t3 = &charlie
slt $t1, $t0, $s1    # t1 = (delta+echo < foxtrot)
beq $t1, $zero, ELSE_BLOCK  # if (delta+echo >= foxtrot) goto ELSE

# this is the true block
sw $t0, 0($t3)       # charlie = delta + echo
j AFTER_IF

ELSE:
sw $s1, 0($t3)       # charlie = foxtrot

AFTER_IF:
```

Page 3
4. (30 points) Convert the following C code to assembly. Note that \texttt{count} is a variable already declared by the MIPS code provided; \texttt{i} is not, and thus should be kept in a register.

\begin{verbatim}
for (int i=0; i<count; i++)
    printf("%d\n", i);
\end{verbatim}

\textbf{REMEMBER:} \texttt{print\_int} is syscall 1; \texttt{print\_str} is syscall 4.

\begin{verbatim}
Solution:
la  $t0, count   # t0 = &count
lw  $t0, 0($t0)  # t0 = count
add $t1, $zero, 0 # i = 0

LOOP_BEGIN:
    slt $t2, $t1, $t0 # t2 = (i < count)
    beq $t2, $zero, LOOP_END # if (i >= count) break

    add $a0, $t1, $zero  # print\_int(i)
    addi $v0, $zero, 1  
syscall

    la  $a0, newline      # print\_str("\n")
    addi $v0, $zero, 4   
syscall  
    # NOTE: print\_char(\'\n\') would have also worked here

    add $t1, $t1, 1       # i++
    j LOOP_BEGIN

LOOP_END:
\end{verbatim}
This is the MIPS code which you should assume runs before each of the questions. Note that the values of the variables are not given!

.data

alpha: .word ???
bravo: .word ???
charlie: .word ???
delta: .word ???
echo: .word ???
foxtrot: .word ???

count: .word ???

newline: .asciiz "\n"

str: .asciiz ???

.text

main:
    la $s0, bravo # s0 = &bravo
    lw $s0, 0($s0) # s0 = bravo

    la $s1, foxtrot # s1 = &foxtrot
    lw $s1, 0($s1) # s1 = foxtrot

    la $s2, delta # s2 = &delta
    lw $s2, 0($s2) # s2 = delta

    la $s3, str # s3 = &str[0]