This quiz is based on material covered in the first two classes.
This quiz should take you 30 minutes.
A copy of the MIPS instruction set sheet is enclosed at the end for your reference.
This is a closed book, closed notes quiz.
Please answer all questions in the space provided and attach additional sheets if necessary.
All questions need to have enough intermediate detail to let us understand how you got to the answer. Absence of such detail will lead to a loss of points even if the answer is correct.
Write your name in the space provided. Absence of identification information will lead to a loss of points.
Requests for re-grading within a week of getting the homeworks, in person and to either the TAs or me during office hours. Written description of “why you think we should change your grade” accompanying the quiz solutions is required.

<table>
<thead>
<tr>
<th>Question</th>
<th>Max</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.a</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1.b</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1.c</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1.d</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2.a</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2.b</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2.c</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2.d</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 1:
You are the CEO of a small web-server development company. During a web-transaction, your web-server uses about 10,000 instructions. Your engineering group tells you these instructions can be categorized as 20% integer arithmetic instructions (INT), 40% disk access instructions(DIO) and 40% network access instructions (NIO). The machine you run this server on has a clock speed of 200MHz and takes 1 clock cycle for INT, 2 for DIO and 3 for NIO instruction types.

(1.a)[4 pts] What is the typical transaction rate (in units of transactions/second) for this configuration of the server?

\[
\text{CPI} = 1 \times 20\% + 2 \times 40\% + 3 \times 40\% = 2.2 \\
\text{transaction rate} = \frac{200 \times 10^6}{2.2 / 10,000} = 9091
\]

Your marketing group suggests that you really need about 25K TPS before anyone will buy your server. Your CIO says he can manage this by simply improving the networking inside the machine and reducing the NIO time to 5 ns.

(1.b) [3] Do you believe your CIO? explain your answer.

No. Even NIO is zero, CPI = 1
\[
\text{transaction rate} = \frac{200 \times 10^6}{1 / 10,000} = 20K
\]

(1.c) [3] Even if your CIO is right, what would the new network time (NIO) have to achieve a TPS of 25K?

\[
\text{CPI should be} = \frac{200 \times 10^6}{10,000/25,000} = 0.8 \\
\text{CPI for NIO X should be} 0.8 = 1 \times 20\% + 2 \times 40\% + X \times 40\% \\
X = -0.5 \\
\text{New network time is negative.}
\]

(1.d) [5] A year from now, you expect clock rate of your machines to improve by 20%. Further, due to the work your group is putting into compiler improvements, you expect the total number of instructions to fall by about 30%. Additionally, you have heard that the hardware vendors have disk and networking enhancements, reducing the number of cycles needed for DIO to 1, and for NIO to 2 (without any NIO changes that your CIO recommended!). How much faster do you think the web-server will be in a year, compared to your previous one?

\[
\text{New CPI} = 1 \times 20\% + 1 \times 40\% + 2 \times 40\% = 1.4 \\
\text{For one transaction,} \\
\text{New Execution time} = \left( \text{Old IC} \times 0.7 \right) \times \left( \text{Old CPI} \times 1.4 / 2.2 \right) \times \left( \text{Old Cycle time} / 1.2 \right) \\
= 0.7 \times 1.4 / 2.2 / 1.2 = 0.37 \\
\text{Speed up} = 1 / 0.37 = 2.7
\]
**Question 2:**

Consider the routine:

```c
char* chrFind(char *s, char c, size_t n)
```

which find the first occurrence of the character c in the string s, where the string s is of length n. If the string s is found, a pointer to the location of ‘c’ is returned. If ‘c’ is not found in string ‘s’, the routine return a 0.

A “C” implementation could look like

```c
char * chrFind(char *s, char c, size_t n) {
    char *rv=0; // Return value
    For (int i=0; i<n ; i++) {
        If ( s[i] == c ) { rv = s+i; break; }
    }
    return rv;
}
```

and a “higher routine” program calling chrFind() could look like

```c
// Prototype for chrFind()
char *chrFind(char *, char, size_t );
topRoutine()
{
    // <do you thing>
    r = (s,c,n);
    // <more do your thing>
}
```

(2.a): [10] Write the equivalent assembly language version for the MIPS ISA for chrFind()

(assume that s is in $a0 and c in $a1 and n in $a2 and the return needs to be in $v0)

Make sure you explain what happens to the return address and all the values carefully.

(Note: MIPS assembly language cheat sheet is enclosed at the end).

Make sure you explain what you are doing in the “code” by appropriate comments on the side, to increase chances for partial marks.

chrFind :

```assembly
Sub $sp, $sp, 4   # adjust stack for one more item
Sw $s0, 0($sp)   # save $ S0
Add $v0, $zero, $zero # initialize return value to 0
Add $s0, $zero, $zero # start of string , I
```

loop:

```assembly
add $t1, $a0, $s0 # the address of ith character in $t1
Lb $t2, 0($t1) # load the character into $t2
Beq $s0 $a2, finish # if end of string, get out of loop
Beq $t2, $a1, found # if $t2 = $a1, character is found
Addi $s0, $s0, 1 # not found, increase index by 1
j loop
```

finish:

```assembly
ret
```

found:

```assembly
add $s1, $s0, $zero
```

chrFind:
found:
  add $v0, $s0, $a0  # character found, return the address

finish:
  lw $s0, 0($sp)    # load the saved $s0
  Addi $sp, $sp, 4  # pop up one item from stack
  Jr $ra           # return

(2.b) [5] Write the relevant sections of the assembly code in topRoutine() to call chrFind. In particular, note that topRoutine itself has a return address, since it was called by someone else (say, main()). Make sure you manage this return address correctly.

topRoutine :
  Sub $sp, $sp, 4   # adjust stack for one more item
  Sw $ra, 0($sp)   # save $ra
  Jal chrFind
  lw $ra, 0($sp)   # load the saved $ra
  Addi $sp, $sp, 4  # pop up one item from stack
  Jr $ra           # return

(2.c) [7] Explain how you could do the same thing using a stack based approach for passing the arguments, (in and out) instead of using $a0 and other registers, for both routines topRoutine() as well as chrFind().
Write pseudo code. Explaining what the code does is more important than getting the details of the code right.

Instead of using $a0, $a1, $a2, and $v0 to pass parameters, we can also use stack as another way of doing it. In topRoutine, we should push all parameters onto stack, also we need to allocate one space for out parameters. In chrFind, we can get the parameters from the stack, and at the end the routine, write the result back to the stack. The following code assumes that the three parameters are in $a0, $a1, $a2. The result in chrFind is in $v0.

topRoutine :
  Sub $sp, $sp, 20  # adjust stack for one more item
  Sw $ra, 16($sp)   # save $Ra
  Sw $a0, 12($sp)   # save first para
  Sw $a1, 8($sp)    # save second para
  Sw $a2, 4($sp)
  Jal chrFind
  lw $ra, 16($sp)   # load the saved $ra
  Addi $sp, $sp, 20  # pop up one item from stack
  Jr $ra            # return
chrFind:
  Lw $a0, 12($sp)  # get first parameter
  Lw $a1, 8($sp)
  Lw $a2, 4($sp)
  ... # do the calculation
  Sw $v0, 0($sp)   # result
  Jr $ra          # return

(2.d) [8] Explain what happens if the prototype in main (by programmer mistake), is
   // Prototype for chrFind()
   char chrFind(char *, char, size_t );
In particular explain what this does to the stack. Once again, explaining the issue and
demonstrating that you understand it, is more important than getting the intricate details of the
code right.

If the prototype is wrong and we continue to use stack to pass parameters, then
we would only allocate one byte for the out parameter because it is of type char.
In chrFind, the program will do the same as last question, calculate the result and
write to the head of stack a 4 bytes address. These four bytes will partially
overwrite input parameter number 3. When the program returns to topRoutine, it
will not get the correct result.
MIPS operands

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 registers</td>
<td>$s0$-$s7$, $t0$-$t9$, $zzero$, $s8$-$s9$, $sv0$-$sv1$, $gp$, $fp$, $sp$, $ra$, $at$</td>
<td>Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zzero$ always equals 0. Register $at$ is reserved for the assembler to handle large constants.</td>
</tr>
<tr>
<td>Memory[0]</td>
<td></td>
<td>Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.</td>
</tr>
<tr>
<td>Memory[4],...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory[4294967292]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MIPS assembly language

<table>
<thead>
<tr>
<th>Category</th>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>add</td>
<td>add $s1$, $s2$, $s3$</td>
<td>$s1 = s2 + s3$</td>
<td>Three operands; data in registers</td>
</tr>
<tr>
<td></td>
<td>subtract</td>
<td>sub $s1$, $s2$, $s3$</td>
<td>$s1 = s2 - s3$</td>
<td>Three operands; data in registers</td>
</tr>
<tr>
<td></td>
<td>add immediate</td>
<td>addi $s1$, $s2$, 100</td>
<td>$s1 = s2 + 100$</td>
<td>Used to add constants</td>
</tr>
<tr>
<td>Data transfer</td>
<td>load word</td>
<td>lw $s1$, 100($s2$)</td>
<td>$s1 = Memory[s2 + 10] + 1$</td>
<td>Word from memory to register</td>
</tr>
<tr>
<td></td>
<td>store word</td>
<td>sw $s1$, 100($s2$)</td>
<td>Memory[$s2 + 10] = $s1$</td>
<td>Word from register to memory</td>
</tr>
<tr>
<td></td>
<td>store byte</td>
<td>sb $s1$, 100($s2$)</td>
<td>Memory[$s2 + 10] = $s1$</td>
<td>Byte from register to memory</td>
</tr>
<tr>
<td></td>
<td>load upper</td>
<td>lui $s1$, 100</td>
<td>$s1 = 100 * 2^{16}$</td>
<td>Loads constant in upper 16 bits</td>
</tr>
<tr>
<td>Immediate branch</td>
<td>branch on equal</td>
<td>beq $s1$, $s2$, 25</td>
<td>If ($s1 == s2$) go to PC + 4 + 100</td>
<td>Equal test; PC-relative branch</td>
</tr>
<tr>
<td></td>
<td>branch on not equal</td>
<td>bne $s1$, $s2$, 25</td>
<td>If ($s1 != s2$) go to PC + 4 + 100</td>
<td>Not equal test; PC-relative branch</td>
</tr>
<tr>
<td>Conditional branch</td>
<td>set on less than</td>
<td>slt $s1$, $s2$, $s3$</td>
<td>If ($s2 &lt; s3$) $s1 = 1$; else $s1 = 0$</td>
<td>Compare less than; for beq, bne</td>
</tr>
<tr>
<td></td>
<td>set less than immediate</td>
<td>sll $s1$, $s2$, 100</td>
<td>If ($s2 &lt; 100$) $s1 = 1$; else $s1 = 0$</td>
<td>Compare less than constant</td>
</tr>
<tr>
<td></td>
<td>jump</td>
<td>j 2500</td>
<td>go to 10000</td>
<td>Jump to target address</td>
</tr>
<tr>
<td></td>
<td>jump register</td>
<td>jr $ra$</td>
<td>go to $ra$</td>
<td>For sw itch, procedure return</td>
</tr>
<tr>
<td></td>
<td>jump and link</td>
<td>jal 2500</td>
<td>$ra = PC + 4; go to 1000$</td>
<td>For procedure call</td>
</tr>
</tbody>
</table>