Policy Reminders

- Include your CS username (a.k.a. NetID) on your page. You will lose a few points from your score if you do not include it.

- You are allowed to work with other students on this homework, as we will not be grading it for correctness. However, each student must turn in their own copy of the homework.

- Show your work for all problems. While we won’t be grading for correctness, you will not receive full credit unless you show your work. After all, showing your work is required on the test - and homeworks are intended to help you practice for the test!

Required Problems:

1(i-j), 2(e), 3(c)

Allowable Instructions

When writing MIPS assembly, the only instructions that you are allowed to use (so far) are:

- add, addi, sub
- beq, bne, j
- slt, slti
- and, andi, or, ori, nor, xor, xori
- sll, srl, sra
- lw, lh, lb, sw, sh, sb
- la
- syscall

While MIPS has many other useful instructions (and the assembler recognizes many pseudo-instructions), do not use them! We want you to learn the fundamentals of how assembly language works - you can use fancy tricks after this class is over.
Problem 1 - More MIPS

This question assumes the following MIPS code, which sets up memory locations `atsf`, `dlw`, `bnsf`, `epsw`, `cbq`, `erie`, `csx`, and `kcs`. The code then loads the values of some of these variables into the indicated MIPS registers. In answering these questions, you can assume this code has already been executed, and that the value of some of the variables are already in the indicated registers.

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

Do not modify any `sX` register, unless specifically instructed.

```
.data
atsf: .word xxx # hidden so you can’t hard-code values!
dlw: .word xxx
bnsf: .word xxx
epsw: .word xxx
cbq: .word xxx
erie: .word xxx
csx: .word xxx
kcs .word xxx

.text
main:
  # set $s0 = kcs
  la $s0, kcs
  lw $s0, 0($s0)

  # set $s1 = csx
  la $s1, csx
  lw $s1, 0($s1)

  # set $s2 = atsf
  la $s2, atsf
  lw $s2, 0($s2)

  # set $s3 = address of dlw
  la $s3, dlw

1(a)
Put csx - kcs - atsf in register $s4

1(b)
Put kcs + csx - atsf in register $s4

1(c)
Put erie in register $s4

1(d)
Put dlw - cbq in register $s4
```
1(e)  
If ( kcs == csx ), put kcs + atsf in register $s4

1(f)  
Put kcs - erie in memory location $csx

1(g)  
If ( csx+kcs < erie ), put csx+kcs in register $s4

1(h)  
If ( atsf-dlw <= kcs ), decrement dlw by one (update the value in memory)!

1(i) - Turn in this one  
If ( bnsf-atsf > csx-kcs ), store (bnsf-atsf) into cbq.

1(j) - Turn in this one  
If ( kcs < erie and erie < epsw ), then store the value of kcs into epsw.
Problem 2 - Masking

Masking is a technique that allows certain bits within a word to remain while other bits are set to zero. The idea is to create a mask that has 1's in the positions that you wish to remain, and 0's elsewhere. For example, if we want to keep bits 31 to 24 within a word but set all other bits to zero, we can use:

```
1111 1111 0000 0000 0000 0000 0000 0000
```

Sometimes, we store the mask as a variable, and load it from memory when we want to use it. This is useful when the mask is complex:

```
.data
mask: .word 0xF0F0F0F0

.text
la $t0, mask
lw $t0, 0($t0)
and $s1, $s0, $t0
```

However, it is often easier and more efficient to generate the mask from simple instructions. In each problem below, first show the 32-bit mask necessary to mask the bits required. Then give a sequence of instructions which takes a value in $s0, masks off the bits required, and stores the result in $s1. In all cases, you may only modify the destination register; no other registers should be changed.

The only instructions you are allowed to use are: and, andi, addi, sll. Note that, because you are not allowed to use la and lw, you cannot read from a mask stored in memory; you must construct it using immediate values. (Don’t use the lui instruction, either.)

HINT: The assembler allows you to use hex values as your immediate values. Don’t waste time converting long bit fields to decimal!

2(a)
Keep only bits 0 through 13. Do this in one instruction.

2(b)
Keep only bits 31 and 28. Do this in three instructions. 

2(c)
Keep bits 12 through 23. Do this in three instructions.

2(d)
Keep bits 0 through 3, and also bits 16 through 19. Do this in four instructions.

2(e) - Turn in this one
Keep all of the even bits. Do this in four instructions.

---

1If I allowed you another instruction - lui - it could be done in two instructions. But that instruction is not allowed, yet.

2This can also be done with shifts in three instructions, but remember that this problem requires that you use a mask instead!
Problem 3 - Truth Tables and Sum of Products

For each part, convert the truth table into a sum-of-products expression for each of the outputs (W, X, Y, Z).

Then draw a logic network which calculates only the Z output from the inputs.

3(a)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3(b)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
3(c) - Turn in this one

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
EXAMPLES

Example: Problem 1(a)
Put csx - kcs - atsf in register $s4

sub $s4, $s1, $s0  # s4 = csx - kcs
sub $s4, $s4, $s2  # s4 = csx - kcs - atsf

Example: Problem 1(b)
Put kcs + csx - atsf in register $s4

add $s4, $s0, $s1  # s4 = kcs + csx
sub $s4, $s4, $s2  # s4 = kcs + csx - atsf

Example: Problem 1(c)
Put erie in register $s4

la $s4, erie  # s4 = &erie
lw $s4, 0($s4)  # s4 = erie

Example: Problem 1(d)
Put dlw - cbq in register $s4

lw $s4, 0($s3)  # s4 = dlw
la $t0, cbq  # t0 = &cbq
lw $t0, 0($t0)  # t0 = cbq
sub $s4, $s4, $t0  # s4 = dlw - cbq

Example: Problem 1(e)
If ( kcs == csx ), put kcs + atsf in register $s4

bne $s0, $s1, AFTER_IF  # if (kcs != csx) skip ahead
add $s4, $s0, $s2  # if (kcs == csx) s4 = kcs + csx

AFTER_IF:

Example: Problem 1(f)
Put kcs - erie in memory location $csx

la $t0, erie  # t0 = &erie
lw $t0, 0($t0)  # t0 = erie
sub $t0, $s0, $t0  # t0 = kcs - erie
la $t1, csx  # t1 = &csx
sw $t0, 0($t1)  # csx = kcs - erie
Example: Problem 1(g)
If \((\text{csx+kcs} < \text{erie})\), put \text{csx+kcs} in register \$s4

\[
\begin{align*}
\text{add} & & \$t0, \$s1, \$s0 & & \# t0 = \text{csx+kcs} \\
\text{la} & & \$t1, \text{erie} & & \# t1 = \&\text{erie} \\
\text{lw} & & \$t1, 0(\$t1) & & \# t1 = \text{erie} \\
\text{slt} & & \$t1, \$t0, \$t1 & & \# t1 = (\text{csx+kcs}) < \text{erie} \\
\text{beq} & & \$t1, \$zero, \text{AFTER\_IF} & & \# \text{if (csx+kcs} >= \text{erie) jump ahead} \\
\text{add} & & \$s4, \$t0, \$zero \\
\end{align*}
\]

AFTER\_IF:

Example: Problem 1(h)
If \((\text{atsf-dlw} \leq \text{kcs})\), decrement \text{dlw} by one (update the value in memory)!

\[
\begin{align*}
\text{lw} & & \$t0, 0(\$s3) & & \# t0 = \text{dlw} \\
\text{sub} & & \$t1, \$s2, \$t0 & & \# t1 = \text{atsf-dlw} \\
\text{slt} & & \$t2, \$s0, \$t1 & & \# t2 = \text{kcs} < (\text{atsf-dlw}) \\
\text{bne} & & \$t2, \$zero, \text{AFTER\_IF} & & \# \text{if (atsf-dlw} > \text{kcs) skip over} \\
\text{addi} & & \$t0, \$t0, -1 & & \# t0 = \text{dlw}\_1 \\
\text{sw} & & \$t0, 0(\$s3) & & \# \text{dlw}-- \\
\end{align*}
\]

AFTER\_IF:

Example: Problem 2(a)
Mask: 0000 0000 0000 0000 0011 1111 1111 1111

\[
\begin{align*}
\text{andi} & & \$s1, \$s0, 0x3fff \\
\end{align*}
\]

Example: Problem 2(b)
Mask: 1001 0000 0000 0000 0000 0000 0000 0000

\[
\begin{align*}
\text{addi} & & \$s1, \$zero, 0x9 & & \# s1 = 1001 \\
\text{sll} & & \$s1, \$s1, 28 & & \# s1 = 1001 0000 0000 0000 0000 0000 0000 0000 \\
\text{and} & & \$s1, \$s0, \$s1 \\
\end{align*}
\]

Example: Problem 2(c)
Mask: 0000 0000 1111 1111 1111 0000 0000 0000

\[
\begin{align*}
\text{addi} & & \$s1, \$zero, 0x0fff \\
\text{sll} & & \$s1, \$s1, 12 & & \# \text{s1} = 1001 \\
\text{and} & & \$s1, \$s0, \$s1 \\
\end{align*}
\]
Example: Problem 2(d)
Keep bits 0 through 3, and also bits 16 through 19. Do this in four instructions.
Mask: 0x000F 000F

```
addi $s1, $zero, 0x000F
sll $s1, $s1, 16
addi $s1, $s1, 0x000F
and $s1, $s0, $s1
```

Example: Problem 3(a)

\[
W = AB\overline{C} + ABC \\
X = \overline{A}B\overline{C} + A\overline{B}C + AB\overline{C} \\
Y = \overline{A}\overline{B}\overline{C} + ABC \\
Z = \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + ABC
\]
Example: Problem 3(b)

\[ W = A B C + \overline{A} B C + A B \overline{C} \]
\[ X = \overline{A} B \overline{C} + A B C + A B C \]
\[ Y = \overline{A} B \overline{C} + \overline{A} B C + A B C + A B C \]
\[ Z = \overline{A} B \overline{C} + A B C + A B \overline{C} + A B C \]