Policy Reminders

- Include your CS username 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(g), 1(h), 2(h), 3(d)

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 - Basic MIPS

This question assumes the following MIPS code, which sets up memory locations hermit, kaibab, tanner, clear, creek, ribbon, falls, and tonto. 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 question 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
hermit: .word xxx # hidden so you can’t hard-code values!
kaibab: .word xxx
tanner: .word xxx
clear: .word xxx
creek: .word xxx
ribbon: .word xxx
falls: .word xxx
tonto: .word xxx

.text
main:
  # set $s3 = tonto
  la  $t0, tonto
  lw  $s3, 0($t0)
  # set $s4 = hermit
  la  $t0, hermit
  lw  $s4, 0($t0)
  # set $s5 = clear
  la  $t0, clear
  lw  $s5, 0($t0)
  # set $s6 = creek
  la  $t0, creek
  lw  $s6, 0($t0)

(a)
Put clear + creek in register $t9

(b)
Put hermit - creek - clear + tonto in register $t2

(c)
Put hermit + falls in register $t1

(d)
Put tonto - clear + hermit in memory location ribbon
(e) If ( \text{tonto} \neq \text{hermit} ), put \text{tonto} + \text{clear} \text{ in register } s2.

(f) If ( \text{creek} \geq \text{clear} ), put \text{ribbon} + \text{clear} \text{ in register } s2.

(g) - Turn in this one

If \text{kaibab} - \text{tanner} = \text{falls} - \text{tonto}, put \text{tonto} \text{ in register } s7.

Solution:

\begin{verbatim}
la $t0, kaibab       # t0 = &kaibab
lw $t0, 0($t0)      # t0 = kaibab
la $t1, tanner      # t1 = &tanner
lw $t1, 0($t1)      # t1 = tanner
sub $t2, $t0, $t1   # t2 = kaibab - tanner

la $t3, falls       # t3 = &falls
lw $t3, 0($t3)      # t3 = falls
sub $t4, $t3, $s3   # t4 = falls - tonto

bne $t2, $t4, AFTER # if (kaibab - tanner != falls - tonto) jump
add $s7, $s3, $zero # s7 = tonto
\end{verbatim}

AFTER:

(h) - Turn in this one

Add 10 to \text{ribbon}, and store the updated value back into the variable.

Solution:

\begin{verbatim}
la $t0, ribbon       # t0 = &ribbon
lw $t1, 0($t0)      # t1 = ribbon
addi $t1, $t1, 10   # t1 = ribbon+10
sw $t1, 0($t0)      # ribbon = ribbon+10
\end{verbatim}
Problem 2 - 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

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

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

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

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

Solution:

```
lw $t0, 0($s3)       # t0 = dlw
sub $t1, $s2, $t0    # t1 = atsf-dlw

slt $t2, $s0, $t1    # t2 = kcs < (atsf-dlw)
bne $t2, $zero, AFTER_IF:  # if (atsf-dlw > kcs) skip over

addi $t0, $t0, -1    # t0 = dlw-1
sw $t0, 0($s3)       # dlw--
```

AFTER_IF:
Problem 3 - 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!

(a) Keep only bits 0 through 13. Do this in one instruction.
```
addi $s1, $zero, 0x000F
sll $s1, $s1, 16
addi $s1, $s1, 0x000F
and $s1, $s0, $s1
```

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

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

(d) - Turn in this one
Keep bits 0 through 3, and **also** bits 16 through 19. Do this in four instructions.

<table>
<thead>
<tr>
<th>Solution: Mask: 0x000F 000F</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.addi $s1, $zero, 0x000F</code></td>
</tr>
<tr>
<td><code>.sll $s1, $s1, 16</code></td>
</tr>
<tr>
<td><code>.addi $s1, $s1, 0x000F</code></td>
</tr>
<tr>
<td><code>.and $s1, $s0, $s1</code></td>
</tr>
</tbody>
</table>

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!
EXAMPLES

Example: Problem 1(a)
Problem: Put clear + creek in register $t9

    add $t9, $s5, $s6  # t9 = clear + creek

Example: Problem 1(b)
Problem: Put hermit - creek - clear + tonto in register $t2

    sub $t2, $s4, $s6  # t2 = hermit - creek
    sub $t2, $t2, $s5  # t2 = hermit - creek - clear
    add $t2, $t2, $s3  # t2 = hermit - creek - clear + tonto

Example: Problem 1(c)
Problem: Put hermit + falls in register $t1

    la $t1, falls      # t1 = &falls
    lw $t1, 0($t1)    # t1 = falls
    add $t1, $s4, $t1  # t1 = hermit + falls

Example: Problem 1(d)
Problem: Put tonto - clear + hermit in memory location ribbon

    sub $t0, $s3, $s5  # t0 = tonto - clear
    add $t0, $t0, $s4  # t0 = tonto - clear + hermit
    la $t1, ribbon     # t1 = &ribbon
    sw $t0, 0($t1)     # ribbon = tonto - clear + hermit

Example: Problem 1(e)
Problem: If (tonto != hermit), put tonto + clear in register $s2

    beq $s3, $s4, AFTER_IF  # if (tonto == hermit) skip ahead
    add $s2, $s3, $s5  # if (tonto != hermit) s2 = tonto + clear

AFTER_IF:

Example: Problem 1(f)
Problem: If (creek >= clear), put ribbon + clear in register $s2

    slt $t0, $s6, $s5     # t0 = (creek < clear)
    bne $t0, $zero, AFTER_IF  # if (creek < clear) skip ahead
    la $t0, ribbon       # t0 = &ribbon
    lw $t0, 0($t0)       # t0 = ribbon
    add $s2, $t0, $s5    # s2 = ribbon + clear

AFTER_IF:
Example: Problem 2(a)
Put $csx - kcs - atsf$ in register $s4$

\[
\begin{align*}
\text{sub} & \quad s4, s1, s0 \quad \# s4 = csx - kcs \\
\text{sub} & \quad s4, s4, s2 \quad \# s4 = csx - kcs - atsf
\end{align*}
\]

Example: Problem 2(b)
Put $kcs + csx - atsf$ in register $s4$

\[
\begin{align*}
\text{add} & \quad s4, s0, s1 \quad \# s4 = kcs + csx \\
\text{sub} & \quad s4, s4, s2 \quad \# s4 = kcs + csx - atsf
\end{align*}
\]

Example: Problem 2(c)
Put $erie$ in register $s4$

\[
\begin{align*}
\text{la} & \quad s4, erie \quad \# s4 = &erie \\
\text{lw} & \quad s4, 0(s4) \quad \# s4 = erie
\end{align*}
\]

Example: Problem 2(d)
Put $dlw - cbq$ in register $s4$

\[
\begin{align*}
\text{lw} & \quad s4, 0(s3) \quad \# s4 = dlw \\
\text{la} & \quad t0, cbq \quad \# t0 = &cbq \\
\text{lw} & \quad t0, 0(t0) \quad \# t0 = cbq \\
\text{sub} & \quad s4, s4, t0 \quad \# s4 = dlw - cbq
\end{align*}
\]

Example: Problem 2(e)
If ($kcs == csx$), put $kcs + atsf$ in register $s4$

\[
\begin{align*}
\text{bne} & \quad s0, s1, AFTER_IF \quad \# \text{if } (kcs != csx) \text{ skip ahead} \\
\text{add} & \quad s4, s0, s2 \quad \# \text{if } (kcs == csx) \ s4 = kcs + csx
\end{align*}
\]

AFTER_IF:

Example: Problem 2(f)
Put $kcs - erie$ in memory location $csx$

\[
\begin{align*}
\text{la} & \quad t0, erie \quad \# t0 = &erie \\
\text{lw} & \quad t0, 0(t0) \quad \# t0 = erie \\
\text{sub} & \quad t0, s0, t0 \quad \# t0 = kcs - erie \\
\text{la} & \quad t1, csx \quad \# t1 = &csx \\
\text{sw} & \quad t0, 0(t1) \quad \# csx = kcs - erie
\end{align*}
\]
Example: Problem 2(g)

If \((csx+kcs < \text{erie})\), put \(csx+kcs\) in register \(s4\)

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

AFTER\_IF:

Example: Problem 3(a)

Mask: \(0000 \ 0000 \ 0000 \ 0000 \ 0011 \ 1111 \ 1111 \ 1111\)

\[
\text{andi} \quad s1, s0, 0x3fff
\]

Example: Problem 3(b)

Mask: \(1001 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000\)

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

Example: Problem 3(c)

Mask: \(0000 \ 0000 \ 1111 \ 1111 \ 1111 \ 0000 \ 0000 \ 0000\)

\[
\begin{align*}
\text{addi} & \quad s1, \text{zero}, 0x0fff \\
\text{sll} & \quad s1, s1, 12 \\
\text{and} & \quad s1, s0, s1 \\
\end{align*}
\]