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:

n/a

Allowable Instructions

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

- add, addi, sub, addu, addiu, subu
- beq, bne, j, jal, jr
- slt, slti
- and, andi, or, ori, nor, nori, 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 - Loops in MIPS

Convert the following loops from C to assembly. Some variables have already been assigned to registers; for others, you will need to assign them registers yourself. Use sX registers for all variables that have names in the C program; use tX registers for any other temporaries that you create.

1(a)
C code:

```c
int sum = 0; // s0
for (int i=-10; i<10; i++)
{
    sum += i;
}
```

1(b)
C code:

```c
int val = ... ; // s2 - assume that some previous code has set it!
while (val > 0)
{
    // HINT: '0' is equal to 0x30 http://www.asciitable.com
    printf("%c", '0'+(val & 0x1));
    // REMINDER: 'div' is not allowed. Use a shift.
    val /= 2;
}
```

Bonus Question (not required): What does this program do? Why does it have a bug if $s2 < 0$?

1(c)
C code:

```c
int len = ... ; // s1 - this is set by previous code
int min = ... ; // s3 - this is set by previous code
for (int i=0; i<len; i++)
{
    if (i >= min)
        printf("%d", i);
}
```

1(d) - Turn in this one
C code:

```c
int pow = ... ; // s0 - this is set by previous code
int prod = 1; // allocate a register for this, of your choosing
for (int i=0; i<pow; i++)
{
    prod = prod*2;
}
```
Solution:

# remember: s0 contains pow
#
# t0 will contain prod
# t1 will contain i
# t2 will be used for various temporaries

addi $t0, $zero, 1  # prod = 1
addi $t1, $zero, 0  # i = 0

LOOP:
    slt $t2, $t1, $s0  # t2 = (i < pow)
    beq $t2, $zero, AFTER  # if (i >= pow) break
    add $t0, $t0,$t0  # prod *= 2
    addi $t1, $t1, 1  # i++
    j LOOP

AFTER:
EXAMPLES

Example: Problem 1(a)

# Remember: sum is in s0.
#
# We’ll decide to store i in t0, and other temporaries in t1.

addi $s0, $zero, 0      # sum = 0
addi $t0, $zero, -10    # i = -10

LOOP:
   slti $t1, $t0, 10      # t1 = (i < 10)
   beq $t1, $zero, AFTER  # if (i >= 10) break
   add $s0, $s0, $t0      # sum += i
   addi $t0, $t0, 1       # i++
   j LOOP

AFTER:

Example: Problem 1(b)

# Remember: val is in s2. It has been initialized to *some* value before
# this code runs.
#
# We’ll decide to use t1 for all temporaries.

LOOP:
   slt $t1, $zero, $s2     # t1 = (0 < val)
   beq $t1, $zero, AFTER   # if (val <= 0) break
  andi $t1, $s2, 0x1      # t1 = (val & 0x1)
   add $a0, $t1, $t130     # t1 = 0x30 + (val & 0x1)
   add $v0, $zero, 11      # print_char('0' + (val & 0x1))
   syscall
   sra $s2, $s2, 1         # s2 /= 2
   j LOOP

AFTER:

Bonus Question: This program prints out the bits of $s2, in reverse order (that is, from LSB to MSB). It has a bug if $s2 < 0 because it will never print out anything; the loop ends immediately.
Example: Problem 1(c)

```assembly
# remember: s1 contains len, s3 contains min
# s0 will contain i
# t0 will be used for various temporaries

addi $s0, $zero, 0  # i = 0

LOOP:
    slt $t0, $s0, $s1  # t0 = (i < len)
    beq $t0, $zero, AFTER  # if (i >= len) break
    slt $t0, $s0, $s3  # t0 = (i < min)
    bne $t0, $zero, SKIP  # if (i < min) skip over
    addi $v0, $zero, 1  # print_int(i)
    add $a0, $s0, $zero
    syscall

SKIP:
    addi $s0, $s0, 1  # i++
    j LOOP

AFTER:
```