ICS 51: Introductory Computer Organization **[Winter 2019]**

Lab#1 (lab1.s)

Soft deadline: January 27th, 11:59PM (1 make up point)

Hard deadline: January 29th, 11:59PM

NOTE: UCIID is your student number

The primary goal of this lab is to learn how to use low-level MIPS assembly instructions to implement *logical* and *conditional statements*. This lab comes in three parts. There are 10 points total.

Your tasks:

1) Complete the count_bit function where you are going to return the number of 1's inside an integer (32 bit number). For example 0xFFFFFFFF should return 32. You will need to use loops and branching for this one. (3 points)
2) Complete swap_bits function which swaps bits at odd and even positions of an integer (32 bits). In other words, every even position bit is swapped with adjacent bit on right side, and every odd position bit is swapped with adjacent on left side. For instance, bits 0 and 1 are swapped, bits 2 and 3 are swapped, and so on. **It is not recommended to use loops or conditional statements for this part. Usage of loops will result in reduced credit.**

   Here's an example: if the given number is 23 (00010111), it should be converted to 43 (00101011). Note that we only use the lower 8 bits of a number, but your program will work on 32-bit numbers.

   You can expect the input to your function in register $t0. The result must be stored inside $t0 as well. (4 points)

3) Complete double_range functions. This function takes three parameters, it finds the smallest and the largest numbers and multiplies their sum by two and returns the result.

Example: if the given numbers are 30, -999 and 999. -999 would be the smallest, and 999 the largest. The result then would be 2*(999-999) or simply 0.

You can expect the inputs to your function to be stored inside registers $t0, $t1, and $t2. Once finished, the result must be stored inside $t0. (3 points)

Important Things to Consider

1) Don't forget to update student name and id variables. These are used for grading purposes. FAILURE TO ADD NAME AND ID RESULTS IN NO SUBMISSION

```
student_name: .asciiz "Your Name"
student_id: .asciiz "Your Student ID"
```

2) Since we use a computer program to test your code, you are **ONLY** allowed to modify the highlighted area in the source code.

For example:

```
double_range:
move $t0, $a0
move $t1, $a1
move $t2, $a2
############## Part 2: your code begins here #################
Your stuff goes here :D)
############## Part 2: your code ends here  ##################
move $v0, $t0
jr $ra
```

3) Once completed, submit change the lab1.s file to include your UCIID and place it in the associated dropbox folder.

Testing

Lab1.s comes with a tester. It basically calls your function with proper parameters and returns the output value. We have provided three test cases for each part that you can use to verify your code functionality. Make sure the printed output matches the expected data.

Inputs that are used to test swap_bits and count_bits are the same.

Double_range function takes three integer parameters. To test this function, we iterate the following liest and use three consecutive items. I.e. [80000, 111, 0], [111, 0, -111], and [0, -111, 11] will be the numbers that are passed to your function. (Numbers are different in the actual file)

Useful information to consider for attempting this lab.

MIPS assembly instructions to shift bits

You can use the following instructions to shift binary values "n" bits to the left or right. They take 3 parameters, source register, destination register and amount of bits to shift (i.e., n)

```
sll $dest_reg, $src_reg, imm_value # shift to the left
srl $dest_reg, $src_reg, imm_value # shift to the right
```

How to load 32 bit numbers (masks) into registers:

MIPS instructions cannot encode immediate values that need more than 16 bits. You'll learn about this limitation in the upcoming lectures. To load 32 bit numbers you need to use two assembly instructions: with the first one (lui) we load the upper 16 bits. The second one will set the lower 16 bits.

```
lui $reg, 16_bit_upper_imm_value  # load upper immediate
ori $reg, $reg, 16_bit_lower_imm_value  # add the immediate
```

Examples:

```
lui $t1, 0x200
ori $t1, $t1, 0x1000
# t1 = 0x02001000

lui $t1, 200
ori $t1, $t1, 1000
# t1 = 1000 + 200*(2**16) # why?
```

To differentiate between hex and decimal (1000 could be decimal or hex) we use "0x" prefix for hex numbers.

Alternatively, you can use “li (load immediate)” pseudo-instruction.

```
li  $t1, 0x02001000
```

Assembler program will implement li with two real MIPS instructions described above.