COE 147 Spring 2014
Lab 3: Endianness, Bit Manipulation, Strings, Loops

Each of you should submit your own solution, according to the instructions at your TA's website. Each person must turn in their own copies of the lab. If you choose to work with a neighbor/partner, put your partner's name on your submitted copy of the lab. This lab is due before recitation the week after it is assigned. Submission timestamps will be checked. Late submissions will not be accepted.

For each of the three parts in this lab, you must name your files in a specific format. Each part will tell you the name to use. Follow instructions so that your program produces output in the correct format. You will have to zip your files into a `.zip` archive and submit it via email as directed on the lab website.

Part 1: Endianness

The following MIPS code defines space in the data segment for 4 bytes and initializes those 4 bytes to the following values. We can refer to the start of this byte sequence, by using the label "a."

```
.data
a:   .byte    0x53, 0x19, 0xA4, 0x4D
```

Copy the code to the simulator and assemble it.

Optional Explore: Examine the data in MARS's data segment window. What do you see? Is this what you expected to see?

Please make a text file called **lab3part1a.txt** to include your answers for questions 1, 2 and 3.

Question 1: What is the address of the byte with value 0xA4?

Tip: Try writing a simple program to load a byte from memory. Ensure that the byte loaded is the 0xA4 byte. Then, consider what address you loaded the byte from.

If our purpose is to use these 4 bytes as a word, we could have defined the previous label as follows:

```
a:   .word    0x5319A44D
```

Replace a's definition in the simulator and assemble the code again.

Question 2: What is the address of the byte with value 0xA4 now?

Question 3: Is the simulator little endian or big endian? How can you tell?

Submit your "lab3part1a.txt". No specific output format is required.

Question 4: Add code to your program to accomplish the following:

Read an integer `A` from user. You can either store it in a register or in a memory word. 32 bits in a word are counted from bit 0 (LSB: Least Significant Bit) to bit 31 (MSB: Most Significant Bit). Now, set register `$a0` to contain bits 19, 20 and 21 of A's, e.g., the least significant bit of `$a0` should contain the 19th bit of `A`, the second least significant bit of `$a0` should contain the 20th bit of `A`, etc.

For example, say your input integer is 57412476 which is 0x36C0B7C. If you take out bits 19, 20 and 21 and store them in `$a0` as the least significant three bits, you will get 5 in decimal in `$a0`.

Tip: This can be done using some sort of shift operation and a bitwise operation

Here is a sample output your program should produce:

```
Please enter your integer:
57412476 <--- this is the input
Here is the output: 5
```

Please make sure your output line contains the string "`Here is the output: `" as shown in the sample output.

Submit your program "**lab3part1b.asm**". Output format will be strictly checked.

Part 2: Strings (Modifying in Place)

Consider the following data segment that defines a null-terminated string:

```
.data
some_string: .asciiz "PittsBurgh TransPorTation"
```

Each byte of memory stores the ASCII code of the corresponding character in the string. For example, the first byte (address 0x10010000, right part of the box in MARS) contains the value 0x50 (80 in decimal), which corresponds to the letter "P" in ASCII code (you can find a list of the ASCII codes in 2nd page of the green MIPS reference sheet of your textbook or online at http://www.asciitable.com). The last byte allocated to the string contains the value 0x00, which identifies the end of the string. Mars automatically adds this 0x00 byte to the end of a string when you use `.asciiz`.

Write a MIPS program that transforms the lowercase characters in an input string to its corresponding uppercase ones and converts the uppercase letters in the input string into lowercase ones. Then print the string to standard output (using a Mars syscall).

Each character is a byte. To declare memory space for a 64 byte string, you can use the following:

```
.data
some_string: .space 64
```

Hint: All uppercase letters are adjacent to each other in the ASCII table. Similarly, all lowercase letters are adjacent to each other. So, you can simply check whether the value of a given byte falls within a specific range and you'll know whether its upper case or lower case (depending on the range you are considering).

To figure out how to quickly convert an uppercase character to a lowercase one (or convert a lowercase character to an uppercase one), consider the ASCII value of the lowercase letter and its uppercase version. For example, lowercase "a" is 0x61 and uppercase "A" is 0x41, a difference of 0x20. What is the difference between "b" and "B"? "z" and "Z"?

Here is a sample output your program should produce:

```
Please enter your string:
PittsBurgh TransPorTation <--- this is the input
Here is the output: pITTSbURGH tRANSpORtATION
```

Please make sure your output line contains the string "`Here is the output: `" as shown in the sample output.

Question 5: Submit your program "**lab3part2.asm**". Output format will be strictly checked.

Part 3: Strings (Modifying a Copy)

We will write a MIPS program that copies a string from one buffer to another in a reverse direction word by word, ignoring spaces.

The following data segment defines 2 different 64-byte long regions of data, which we will use as buffers to store strings:

```
.data
buf1: .space 64
buf2: .space 64
```

Initially, these buffers will contain null bytes (0x00).

First, write MIPS code that prompts the user for a string and stores it in buf1. Use the MIPS read string syscall. Then, write MIPS code to move that string from one buffer to the other in a reverse direction word by word, ignoring spaces (ASCII value 0x20). So if `buf1` contains "`Assembly Lan-guage`", then `buf2` will contain "`ylbmessAegaugnaL`" at the end of your program.

You will have to use two registers to keep two separate pointers to each buffer, because one might be moving faster than the other one. Do this by writing a loop that examines each character in `buf1` in reverse within a word. You have to stop processing when you reach the beginning of each word in `buf1`. One way to do this is to use the length of the word and keep track of how many characters within the word you have already processed. If the character is a space, do not write it to `buf2`. When you have finished processing, do not forget to store a null character at the end of `buf2`.

Finally, write MIPS code that prints the contents of `buf2` using the print string syscall.

Here is a sample output your program should produce:

```
Please enter your string:
Assembly Language <--- this is the input
Here is the output: ylbmessAegaugnaL
```

Please make sure your output line contains the string "`Here is the output: `" as shown in the sample output.

Question 6: Submit your program "**lab3part3.asm".** Output format will be strictly checked.