COE 0147 Lab 3

### Endianness, Bit Manipulation, Strings, Loops

# Endianness and Bit Manipulation

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 0x93, 0xFE, 0xBC, 0x3E

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?

## Questions:

* + 1. **What is the address of the byte with value 0xBC?**

0x10010002

**Hint:** Try writing a simple program to load a byte from memory. Ensure that the byte loaded is the 0xBC 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:

Now replace the definition of a with the following (notice the bytes are the same as above).

a: .word 0x93FEBC3E

* + 1. **What is the address of the byte with value 0xBC now?**

0x10010001

* + 1. **Is the simulator little endian or big endian? How can you tell?**

**A**

Little endian: The first byte inputted in both cases is the least significant byte in memory

* + 1. **Now, write a 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 lab3part1.asm. Output format will be strictly checked.**

# 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 online at http://www.asciitable.com).

The last byte allocated to the string contains the value 0x00, which identifies the end of the string (such strings are called null-terminated strings). Mars automatically adds this 0x00 byte to the end of a string when you use .asciiz. Some programming languages also automatically store strings with a null terminator.

**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).**

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

.data

some\_string: .space 64

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.

**Submit your program lab3part2.asm. Output format will be strictly checked.**

# Strings (Modifying a Copy)

**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). This is a characteristic of MARS simulator, but other simulators may initialize with random data.

First, write code that prompts the user for a string and stores it in buf1. Use the MIPS syscalls. Then, write 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 Language”, then buf2 will contain “ylbmessAegaugnaL” at the end of your program.

Start by using a pointer to buf1 and find the length of the first word. Then using the length, store the word in reverse direction in buf2. Once a word is reversed, update the pointer to point to the next word and repeat the same procedure. 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:

Here is the output: ylbmessAegaugnaL

Assembly Language *←*this is the input

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

**Submit your program lab3part3.asm. Output format will be strictly checked.**