Name: Nick Petro **CoE 0147**

Lab 3, Spring 2016

Lab 3: Endianness, Bit Manipulation, Strings, Loops

Samuel J. Dickerson

Important:

- While collaborations are allowed, names of all the collaborators must be listed in the assignment. Each student should submit individual assignments for grading, despite collaboration.
- Solutions (***.asm files and text files if any**) for this assignment should be submitted at `Courseweb/CourseDocuments/Recitations/Lab3` before the next recitation.
- Each program of the assignment must be in a separate file. You must write your name in the first line of every file in the following format.
  #Name: First_name Last_name
  #rest of the file...

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     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?

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

1.1 Questions:

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

   Tip: 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:

   ```
   a: .word 0x93FEBC3E
   ```

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

2. What is the address of the byte with value 0xBC now?

1) 0x10010002
2) 0x10010001
3) Little endian, because it stores the least significant bit in the lowest memory address and the most significant bit in the greatest memory address.

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

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

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.

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 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`.

2.1 Questions:

5. 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.

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

3.1 Questions:

6. 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 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:
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.

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

3.2 Practice problems:

1. (2.22) For the following C statement, write a minimal sequence of MIPS assembly instructions that does the identical operation. Assume $t1 = A, $t2 = B, and $s1 is the base address of C.

   A = C[0] << 4;

2. (2.23) Assume $t0 holds the value 0x00101000. What is the $t2 after the following instructions?

```
slt $t2, $0, $t0
bne $t2, $0, ELSE
j DONE
ELSE: addi $t2, $t2, 2
DONE:
```

3. (2.27) Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of a, b, i, and j are in registers $s0, $s1, $t0, and $t1, respectively. Also, assume that register $s2 holds the base address of the array D.

```
for (i = 0; i < a; i++)
for (j = 0; j < b; j++)
D[4*j] = i + j;
```

4. (2.29) Translate the following loop into C. Assume that the C-level integer i is held in register $t1, $s2 holds the C-level integer called result, and $s0 holds the base address of the integer `MemArray`.

```
addi $t1, $0, $0
LOOP: lw $s1, 0($s0)
add $s2, $s2, $s1
addi $s0, $s0, 4
addi $t1, $t1, 1
slti $t2, $t1, 100
bne $t2, $s0, LOOP
```

Practice Problems:

1)
```
lw   $t1, 0($s1)
sll  $t1, $t1, 4
```
2) $t2 = 3

3)
```
add $t0, $0, $0          # i = 0
L1:
slt  $t2, $t0, $s0       # i < a
beq $t2, $0, Exit        # $t2 == 0, go to Exit
addi $t0, $t0, 1         # i = i+1
add $t1, $0, $0          # j = 0
L2:
slt  $t2, $t1, $s1       # j < b
beq $t2, $0, L1          # if $t2 == 0, go to L1
add $t2, $t0, $t1        # i+j
sll  $t4, $t1, 4         # $t4 = 4*j    (sll was written instead of mul.)
add $t3, $t4, $s2        # $t3 = &D[4*j]
sw  $t2, 0($t3)          # D[4*j] = i+j
addi $t1, $t1, 1         # j = j+1
j L2
Exit:
```

4)
```
i = 0;
do {
    result += MemArray[i];
    i++;
} while((i<100) != MemArray[i]);
```