Palindrome

Objectives

This lab continues practice with MIPS. The important concepts for this lab are load data from data section of memory into registers, organizing code into multiple files to delineate between functionalities, and callee/caller conventions. The deliverable is a program that is able to discern if user input is a palindrome. User input is limited to one word with variable length letters.

Deliverables

The zip file contains 3 asm files: main.asm, paliChecker.asm, and stringLen.asm. Once you unzip the folder, place Mars MIPS executable in the same folder. When assembling program, make sure the main tab is active. Once you're done, zip and submit the 3 asm files via FSO.

1) In main.asm file, fill out the .data and .text sections.
   a. The .data section contains allocation of space for user input and any messages for the user. For example, in Figure 1, there are two null terminated strings, prompt and output. The strings are null terminated because of the directive ".asciiz". In addition to those two strings, line 3 is an allocation of space of 256 bytes. This is because the .space directive is used. Other ways to allocate space include the directive .byte, .word, and .halfword. For your file, you will need to allocate space for the user input. In addition, you will need to have message(s) alerting the user if their input is a palindrome or if their input is not a palindrome.

```
1           .data
2   prompt: .asciiz "Enter the size of the map. Use format HxW:"
3   buffer: .space 256
4   output: .asciiz "container.bmp"
```

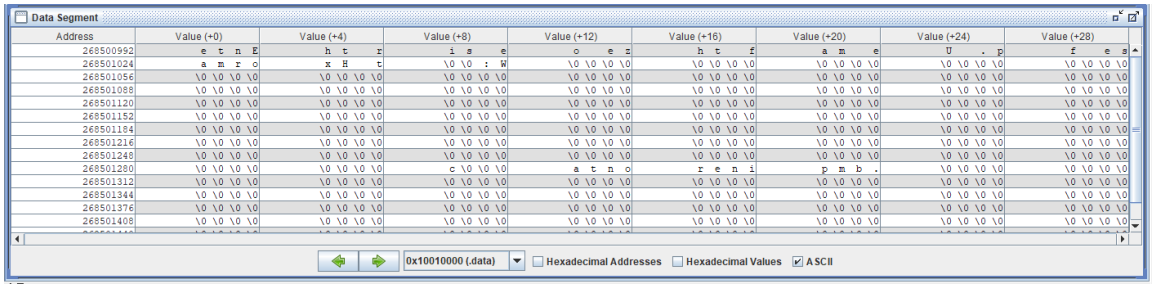

Figure 1

   b. The .text section contains the executed lines during when the program is running. For this lab, ask the user for a string input, call the stringLen file in order to find the length of the user input,

call paliChecker file in order to determine if the input is a palindrome, and then print out, on the console window, the length of the input and if the input is a palindrome. To call stringLen and paliChecker, use the instruction jal. This will save the memory address in the pc counter into register $ra. In addition, caller/callee convention must be followed. For example, when calling stringLen file, pass in the user input address as an argument. The address has to be in one of the $a0-$a3 registers. For paliChecker, you will need two arguments. When the program returns to the main after calling stringLen, any return value(s) should be read from registers $v0-$v1.

2) In the stringLen file, create a label so that main.asm can call the functionality listed in this file. Implement the ability to find the string length of user input. The memory address of user input should have been passed along via one of argument registers ($a0-$a2). Create a loop that has the following (but not necessarily this this order):
    a. With the memory address, use lb instruction to load each byte of the user string.
    b. Use addiu instruction to increment the memory address for each loop.
    c. Use addiu instruction to increment a counter to keep track of the string length.
    d. Use beq instruction to check for conditions that indicate loop must be exited (there are two conditions).
    e. Use j instruction to loop again.

    Once the string length has been found, pass the return value via one of the $v0-$v1 registers. Jump back to where main left off by using the jr instruction.

3) In the PaliChecker file, create a label so that main.asm can call the functionality listed in this file. Implement the ability to compare two elements of array in order to check to see if the entire string is a palindrome. The memory address of user input and the string length should have been passed along via one of argument registers ($a0-$a2). Create a loop that has the following (but not necessarily in this order):
    a. With the memory address and string length arguments, load the first and last byte of the string of characters.
    b. Compare the first and last byte by using the logical operation XOR. See table 1 for XOR logic.
    c. If the currently loaded bytes pass the XOR check, then continue on to the next set by incrementing the memory address that points to the first byte and decrementing the memory address that points the last byte. This ensures the next iteration of the loop loads new values.
    d. Use a branch instruction to check condition for jumping out the loop. For this loop, use stringLen information to determine how many times looping needs to occur.
    e. Use J instruction to loop again.

    Once the check has been complete, return value via one of the $v0-$v1 registers. Jump back to where main left off by using the jr instruction.

RUBRIC (Total 90 Points)

Points	Description
30	Question 1
30	Question 2
30	Question 3

FSO SUBMISSIONS

Your project source code must be zipped up and submitted (turned-in) to FSO. Labs do not require being checked off by the Lab Specialist or Course Director before submission. However, if you complete your lab before the lab period ends, then you may request check-off to leave early.

NOTE: Unless otherwise stated by the course director, labs submitted after due date are considered late and receive a 25 point grade reduction. Each week incurs an additional 25 point penalty.