EE 109 Lab 10 – A Man, A Plan, A Canal: Panama

1 Introduction

In this lab you will write a small program to check whether an input string of digits (no text, just numeric digits) form a valid **numeric** palindrome. A palindrome is a string whose contents are the same when read forward or backwards. Examples of text palindromes are "racecar", "a toyota", and the title of our lab. **Numeric palindromes** include "123454321" or "2002". This lab will ONLY consider **numeric palindromes**.

2 What you will learn

This lab is intended to teach you the basics of writing assembly language code that includes various control structures (loops, conditionals, etc.) as well as how to use appropriate syscalls for input & output via the MARS simulator.

3 Background Information and Notes

Syscall's: Syscall's are a mechanism by which user applications can call specific OS routines, usually for I/O purposes. These are accomplished by simply using the instruction 'syscall'. Before executing the 'syscall' instruction you must place an argument in $v0 and often $a0 or $a1. The value in $v0 indicates which syscall should be executed with additional arguments in $a0 and $a1. Table 1 shows several useful syscall's, while the online help in MARS has a list of all available syscalls which you should look over as you do your other labs.

Syscall	$v0	Additional arguments	Notes
'read string'	8	$a0 = address of input buffer $a1 = maximum # of characters to read	Leaves the LF character at the end of the string
'print string'	4	$a0 = address of null terminated ASCII string	
'print character'	11	$a0 = ASCII char. to print	
'print integer'	1	$a0 = integer to print	

Table 1- Useful Syscalls

4 Procedure and Program Requirements

The requirements that your program must meet are as follows:

a. Your program shall prompt a user to enter an ASCII string of digits, maximum 20 digits without spaces plus the termination 'Enter' key / LF character.

b. Receive the input. You can assume the user enters a correct and appropriately formatted input string (you do not have to do any error checking).

c. Determine if the numeric string is a palindrome. Be sure your method works for strings of both even and odd length. There are several approaches you could use. You can setup pointers to the start and end character and walk for length n/2 comparing the start and end character then moving start forward and end backward by one location. A less efficient but acceptable approach is to make a reversed copy of the string in a new buffer of memory and then walk down both strings in parallel checking that the character at each corresponding location is the same.

d. Output a message indicating the result. Your output message should include the original input from the user and then the result. (See the attached output example). Think about how early you'd be ready to say something IS or IS NOT a palindrome and consider breaking out of the loop appropriately.

e. Finally, **regardless of the result above,** perform some statistical analysis. Determine the mode digit (most commonly occurring digit) of the user input string. If there is a tie for the mode digit, select the one with greater value to output. A good approach to do this is to create an array with 10 locations and store the occurrence count of digit i at index i in the array. Do not use 10 separate registers to store the counts. Use an approach that mimics the high-level code: `counts[ buf[i] – '0'] += 1;`

f. You MAY NOT use separate registers for each digit to count their occurrences or for storing each of the 20 characters. You must use the arrays defined and code loops using branches to iterate over the BUF array.

g. Beyond just a working program, you should attempt to minimize the number of instructions executed by your program as it runs. This can be determined by using the Instruction Counter Tool available in MARS. First make sure your program is running correctly (no need to measure instructions if the program does not work). Now assemble it again but this time start the Instruction Counter (from the Tools menu) before starting execution of the program. When the Instruction Counter window appears, click Connect so the tool can begin monitoring the MARS simulation. Then run your program (you can still step through or set breakpoints if desired). Use the total instruction count as a metric to determine how “good” your implementation is. More points will be awarded to those with LOWER total instruction counts. Note: Especially when finding the mode, consider the possible tradeoffs of using more memory to reduce instruction count. A bad count would be greater than 500. A good count would be less than 420 instructions.

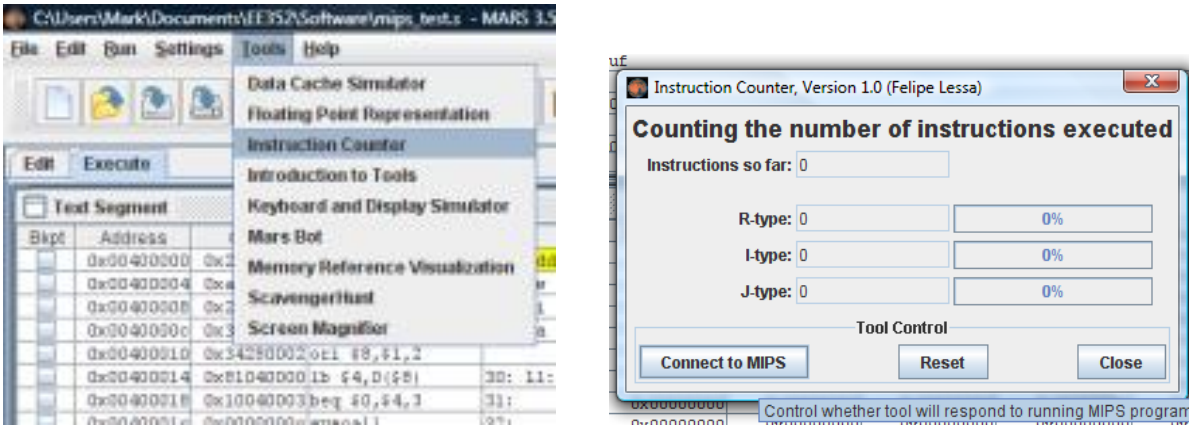

Figure 1- Starting and Connecting the Instruction Counter Tool in MARS

h. Comment your code enough for the TA/Instructor to understand your approach/intent to solving the problem.

i. Try to organize your code in a coherent fashion (no spaghetti code with jumps and branches all over) and make it readable using helpful labels, good formatting/alignment, etc.

5 Programming and Submission Guidelines

1) A program skeleton is provided. Use it as a baseline. You may first want to code up a solution to this problem in C/C++ to help you.

2) You should be careful about a few things. First, the numbers you will be receiving as input will be *ASCII representations* of 0-9. You should think carefully whether it is beneficial to convert them to their actual 0-9 binary representations or leave them as their ASCII representations. You can convert to integer values 0-9 by subtracting the 0x30 from each ASCII digit. Second, be careful to use appropriately sized load and store instructions. ASCII characters are each a byte.

3) You may use any pseudo-instructions you desire. "li – load immediate", "la – load address", and branch pseudo-instructions will be helpful.

4) Once your program works and you have tested it on several input patterns (consider what would be good test cases…maybe something with the max number of characters, something with a minimum number of characters, something of even length, something with odd length, etc.), and something with the maximum number of characters (20) submit your source file by going to the Lab9 assignment on Blackboard (Assignments..Labs..Palindrome Lab) and then attach and submit your ee109_palindrome.s file. Note: You must "Submit" and not just "Save" when you perform the submission.

6 Review

None

See below for sample program execution and program skeleton.

Sample Program Execution:

```
Enter up to 20 digits without spaces: 567898765
Your string: 567898765 IS a palindrome
The mode digit is: 8
```

```
# ee109_palindrome.s
# Name:  _____________________
#

          .data
Buf :    .space  22     # receive original input in this buffer
counts: .byte 0,0,0,0,0,0,0,0,0,0  # array of counts for digits 0-9
                                   # useful for finding the mode digit

# the following are constant strings that you can use
# for your prompts and messages... use syscall with $v0=4
msgin:  .asciiz "Enter up to 20 digits without spaces: "
msg1:   .asciiz "Your string: "
msg2:   .asciiz " IS a palindrome\n"
msg3:   .asciiz " IS NOT a palindrome\n"
msg4:   .asciiz "The mode digit is: "

# print this string for a newline character
nline:  .asciiz "\n"


          .text
main:
          li     $v0, 4        # Print the prompt
          la     $a0, msgin
          syscall
          li     $v0, 8        # Get the string from the user
          la     $a0, buf1
          li     $a1, 22
          syscall

          # this syscall will keep the newline character on the end of the string
          # so we will know we're at the end of a string when we hit the newline
          # newline = '\n' = 0x0a

# a lot more of your code



# exit
          li     $v0,10
          syscall
```

Student Name: _____

Item	Outcome	Score	Max.
Palindrome check • Walks array backwards and forwards • Compares correct characters to each other • Produces correct palindrome determination	 Yes / No Yes / No Yes / No		 2 1 1
Mode • Computes number of occurrences for each digit • Selects max as mode • Selects greater digit if there is a tie for the max • Uses loop rather than loading 10 registers	 Yes / No Yes / No Yes / No Yes / No		 2 2 1 1
Instructor Tests			
• Program works correctly for input set 1	Yes / No		2
• Program works correctly for input set 2	Yes / No		2
• Program works correctly for input set 3	Yes / No		2
• Program works correctly for input set 4	Yes / No		2
Efficient (Good=2 / Avg=1 / Bad=0)			2
SubTotal			20
Late Deductions			
Total			
Open Ended Comments:			