CSE 230 Project 3: UART Palindrome Checker

Learning Objectives:

- Create modular code and interface with unfamiliar modularized code

The Task

In this project, you will be writing a program that receives a string of characters via the UART, checks if this string is a palindrome, and then uses a print function to print either "Yes" or "No". A palindrome is a sequence of characters (typically a word or phrase) that is the same both forwards and backwards. For this project, strings will be terminated using a period ('.'). You may assume that a string will contain at least one letter in addition to a period (e.g., the input, "b.", should be considered a palindrome). You will not need to handle empty strings, strings containing only a period, or stings containing characters other than letters, spaces, and periods. Your program should be able to handle multiple strings sent one after another or concatenated together. For example, the string: "abba. data." should print "Yes" followed by "No" on the next line. Spaces should be ignored when checking for a palindrome and the palindrome should not be case sensitive. For example, "A nut for a jar of Tuna." would be considered a palindrome.

Print Function

A template PLP project file is available to download on Canvas. The PLP project includes a second *ASM* file titled, *project3_lib.asm*. This *ASM* file contains the print function used in this project. PLPTool concatenates all *ASM* files within a PLP project into a single location in memory (unless additional *.org* statements have been added to specify different location for code). No changes to *project3_lib.asm* should be made.

When called, depending on the value in register *$a0*, the following string will be displayed on the simulated UART device's output. If *$a0* contains a zero then "No" will be displayed and if *$a0* contains a non-zero value (e.g. one) then "Yes" will be displayed. The print function is called using the following instruction:

```
call project3_print
```

To use the print function, your PLP program needs to initialize the stack pointer (*$sp*) before performing the function call (or any other operations involving the stack pointer). For this reason, the template project file includes an initialization that sets the stack pointer to `0x10ffffc` (the last address of RAM).

Template Structure

The template project file contains six function stubs that need to be implemented. Five are called from the *main* loop and the sixth is called from "*period_check*". The template file contains comments with descriptions of what each function needs to do and how it should be implemented. The table below provides a brief description of the functions.

Function Name	Function Description
poll_UART	Polling loop for UART's status register, reading new character, and indicating character has been read
period_check	Checks if new character is a period and makes a nested function call to *palindrome_check* if it is
space_check	Skips saving space characters to array
case_check	Converts new character to uppercase if it is lowercase
array_push	Saves new character to array
palindrome_check	Moves inwards from front and back of array and compares characters at each step to determine if the string is a palindrome. If at any point mirroring characters are not equal, then it should use the print function to print "No". If the comparison reaches or passes the midpoint then the print function should be used to print "Yes".

Debugging Tools

To show that you properly tested your program, please save your program while it is in simulation mode (the blue *simulation mode* button has been toggled on) such that a **CPU Memory Visualization window** is open and shows **at least the first 6 words of memory in the array**. Please keep in mind that the starting address of the array is relative to the number of initializations placed before the assembler directive allocating space for the array. If you change your initializations you may need to adjust your CPU Memory Visualization base address. The *Array Example* video in this project's module demonstrates how to set up a CPU Memory Visualization

window. It is acceptable to also see program contents in memory (these cells will be colored blue). For example, it would be acceptable to see the following in the visualizer after the string, “abcd.”, has been sent by the user.

	$sp	Address	Contents
4 bytes / div 11 entries		0x1000003c	0x08000075
		0x10000040	0x00000000
		0x10000044	0x00000061
		0x10000048	0x00000062
		0x1000004c	0x00000063
		0x10000050	0x00000064
		0x10000054	0x00000000
		0x10000058	0x00000000

Additionally, the **watcher window** should contain, at minimum, registers $v0, $s1, and $s2. You may also include any other registers in the watcher window that you find useful for debugging.

Points will be deducted if a CPU Memory Visualization window configured to show the start of the array does not open when simulation mode is toggled on or the watcher window does not include the minimum set of registers (the watcher window does not need to open when simulation mode is toggled on).

Deliverables:

1. Take the Project 3 Pre Quiz (6 points)
2. Submit your program on Canvas with the format: *Firstname_Lastname_project3.plp* (21 points)
3. Take the Project 3 Post Quiz (3 point)