**PROECT REPORT**



Project Name: Palindrome Checker

Author: Vedad Kruho & Mustafa Sinanović

Date: 22.05.2024

Course: Computer Architecture (CS304)

Project: MIPS Project

Contents

[1.0 Project Description 3](#_Toc167608064)

[2.0 Program Design 3](#_Toc167608065)

[3.0 Symbol Table 6](#_Toc167608066)

[4.0 Prototype in C-language 6](#_Toc167608067)

[5.0 Test Plan 8](#_Toc167608068)

[Test results 10](#_Toc167608069)

# 1.0 Project Description

**Project Name:** Palindrome Checker

**Description:** The Palindrome Checker is a MIPS assembly program designed to determine whether a given string is a palindrome. A palindrome is a word, phrase, number, or other sequence of characters that reads the same forward and backward (ignoring spaces, punctuation, and capitalization). This program processes a hardcoded string, checking its characters from both ends towards the center to verify if they match.

**Inputs and Outputs:**

* **Input:** A predefined null-terminated string (e.g., "kayak").
* **Output:** The program outputs **1** if the string is a palindrome and **0** if it is not.

The program uses MIPS assembly instructions to iterate through the string from both ends, comparing characters until the middle of the string is reached. If all corresponding characters match, the string is declared a palindrome; otherwise, it is not.

# 2.0 Program Design

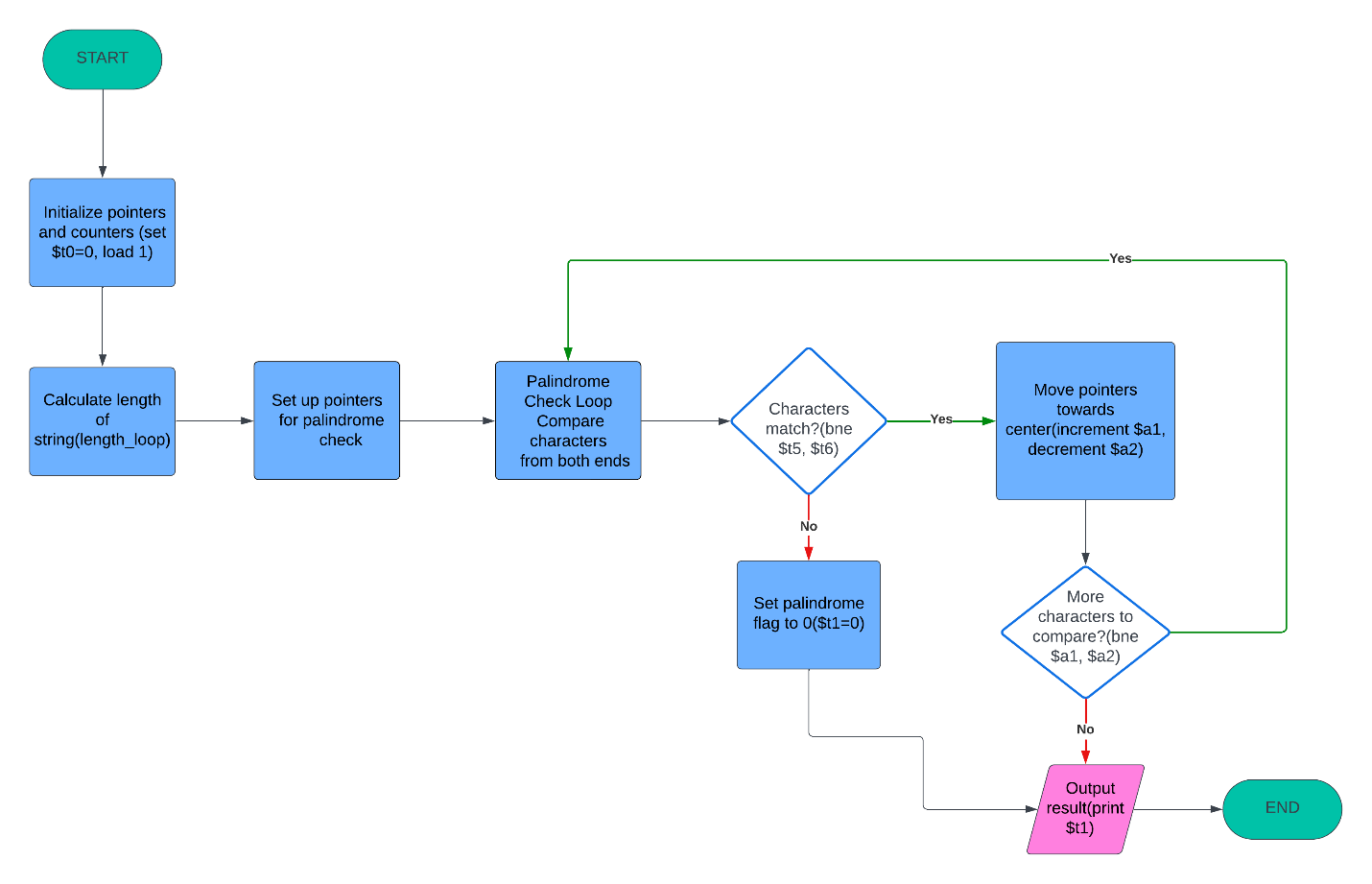
**Narrative Description:**

The Palindrome Checker operates by first calculating the length of the input string. It then uses two pointers: one starting at the beginning of the string and the other at the end. The characters pointed to by these pointers are compared. If all characters match, the string is a palindrome; otherwise, it is not.

**Detailed Steps:**

1. **Initialization:**
   * The program starts by setting up initial values for counters and pointers. It initializes the length counter ($t0) to 0 and loads the base address of the input string into $a1.
2. **Calculate String Length:**
   * A loop is used to iterate through the string until the null terminator is encountered. For each character, the counter ($t0) is incremented. This loop terminates when the null terminator (0) is found, indicating the end of the string.
3. **Setup for Palindrome Check:**
   * After calculating the string length, pointers are set up for checking the palindrome. $a1 is reset to the start of the string, and $a2 is set to the last character of the string. The palindrome flag ($t1) is initialized to 1, assuming the string is a palindrome unless proven otherwise.
4. **Palindrome Check Loop:**
   * In this loop, characters from the start ($a1) and end ($a2) of the string are compared. If any characters do not match, the palindrome flag ($t1) is set to 0, indicating the string is not a palindrome. Pointers are then moved towards the center: $a1 is incremented, and $a2 is decremented. This process repeats until the pointers meet in the middle or a mismatch is found.
5. **Output the Result:**
   * The final value of the palindrome flag ($t1) is printed: 1 if the string is a palindrome, or 0 if it is not. The program then terminates.

**Flowchart:**

****

**Registers behavior:**

The main perspective gives exactly registers which values are best representation of what is really happening behind the scene, so this report is going to include that part.

‘kayak’ -> string that is going to be checked

addi $t0, $zero, 0x1000 -> $s5=0x10000000

ori $a1, $s5, 0x0000 -> $a1=0x10000000

length\_loop\_

1st iteration:

lb $t5, 0($a1) -> $t5=107

beq $t5, $zero, length\_end -> $t5 is not 0 so continue

addi $a1, $a1, 1 -> $a1=0x10000001

addi $t0, $t0, 1 -> $t0=1

subsequent iterations: character by character $a1 will pont to the null terminator, and $t0 will contain the length of the string

final iteration:

lb $t5, 0($a1) -> $t5=0

beq $t5, $zero, length\_end -> goes to length\_end

length\_end:

lui $s5, 0x1000 and ori $a1, $s5, 0x0000 -> $a1=0x10000000

addi $a2, $a1, 0 -> $a2=0x10000000

add $a2, $a2, $t0 -> $a2=0x10000005

addi $a2, $a2, -1 -> $a2=0x10000004 (last char)

addi $t1, $zero, 1 -> $t1=1 (palindrome)

palindrome\_check\_loop:

1st iteration:

lb $t5, 0($a1) -> $t5=107

lb $t6, 0($a2) -> $t6=107

bne $t5, $t6, not\_palindrome -> $t5=$t6 so continue

addi $a1, $a1, 1 -> $a1=0x10000001

addi $a2, $a2, -1 -> $a2=0x10000003

sub $at, $a1, $a2 -> $at=0x10000001-0x10000003=-2

bne #at, $zero, palindrome\_check\_loop -> $at is not so continue

…

Until $a1=$a2

not\_palindrome:

ori $t1, $zero, 0 -> $t1=0

# 3.0 Symbol Table

Registers used in application:

|  |  |
| --- | --- |
| Register | Purpose |
| $t0 | Length counter: Stores the length of the string |
| $t1 | Palindrome flag: Stores 1 if the string is a palindrome, else 0 |
| $t5 | Temporary register: Holds the current character from the start of the string |
| $t6 | Temporary register: Holds the current character from the end of the string |
| $s5 | Base address holder: Used to load the upper half of the string's address |
| $a0 | Argument register: Used for system calls (print result) |
| $a1 | Address register: Pointer to the current character in the string |
| $a2 | Address register: Pointer to the current character in the reverse string |
| $v0 | System call number register |
| $zero | Constant zero register |
| $at | Assembler temporary register |

Labels used in program:

|  |  |
| --- | --- |
| Label | Purpose |
| \_\_start | Entry point of the program |
| length\_loop | Loop to calculate the length of the input string |
| length\_end | Marks the end of the length calculation and the start of the palindrome check setup |
| palindrome\_check\_loop | Loop to compare characters from both ends of the string |
| not\_palindrome | Sets the palindrome flag to 0 if any characters do not match |
| print\_result | Determines whether to print 1 or 0 based on the palindrome flag |
| program\_end | Ends the program |

# 4.0 Prototype in C-language

Code in C language:

#include <stdio.h>

#include <string.h>

int main() {

    char test[] = "rotator"; // Input for testing if it is palindrome

    int length\_test = 0;

    // Calculating the length of our input

    while (test[length\_test] != '\0') { // It will calculate (be inside loop) until 'null character' is reached - no more characters)

        length\_test++; // Increasing length each time by one

    }

    // We need to initialize start and end point

    char \*starting\_point = test; // Pointing to the start of a string

    char \*ending\_point = test + length\_test - 1; // Pointing to the last character of a string

    int check\_palindrome = 1; // We assume string is palindrome - set to 1

    // Now, go inside the loop to check if string is palindrome

    while (starting\_point < ending\_point) { // Basically, goes until last character is reached

        if (\*starting\_point != \*ending\_point) {

            check\_palindrome = 0; // In palindrome, first and last character needs to match, so if they are not equal, it is 0

            break; // Loop - exit because it is not plaindrome

        }

        starting\_point++; // Move to the next character from the start

        ending\_point--; // Move to the previous character from the end

    }

    // Length of a string

    printf("Length: %d\n", length\_test);

    // Result

    if (check\_palindrome) {

        printf("PALINDROME!\n");

    } else {

        printf("NOT PALINDROME!\n");

    }

    return 0; // Exit call

}

# 5.0 Test Plan

**First Input: "kayak"**

In the first input, we selected the string "kayak," which has seven characters and is a palindrome. This input was chosen to verify the basic functionality of our program.

1. **Length Calculation:**
   * The program calculates the length of "rotator" as 5 characters.
   * This length is stored in the register R8 (t0) as **00000005** in hexadecimal.
2. **Palindrome Check:**
   * The program then checks each character from both ends of the string.
   * Since "kayak" reads the same backward and forward, it is identified as a palindrome.
   * The result is stored in the register R9 (t1) as **00000001**, indicating the string is a palindrome.

**Second Input: "diegomaradona"**

For the second input, we used the string "diegomaradona" to examine the program's handling of string length and hexadecimal conversion. This input also helps to test the program's ability to identify non-palindromes.

1. **Length Calculation:**
   * The program calculates the length of "diegomaradona" as 13 characters.
   * This length is stored in the register R8 (t0) as **0000000d** in hexadecimal (where "d" represents 13).
2. **Palindrome Check:**
   * The program compares characters from both ends of the string.
   * Since "diegomaradona" does not read the same backward and forward, it is not a palindrome.
   * The result is stored in the register R9 (t1) as **00000000**, indicating the string is not a palindrome.

**Third Input: "rooooooooooooooooooor"**

In the third input, "rooooooooooooooooooor," we chose a longer string to further test the program's handling of string length and hexadecimal conversion.

1. **Length Calculation:**
   * The program calculates the length of "rooooooooooooooooooor" as 21 characters.
   * This length is stored in the register R8 (t0) as **00000015** in hexadecimal (where 15 represents 21 in decimal).
2. **Palindrome Check:**
   * The program compares characters from both ends of the string.
   * Since "rooooooooooooooooooor" reads the same backward and forward, it is a palindrome.
   * The result is stored in the register R9 (t1) as **00000001**, indicating the string is a palindrome.

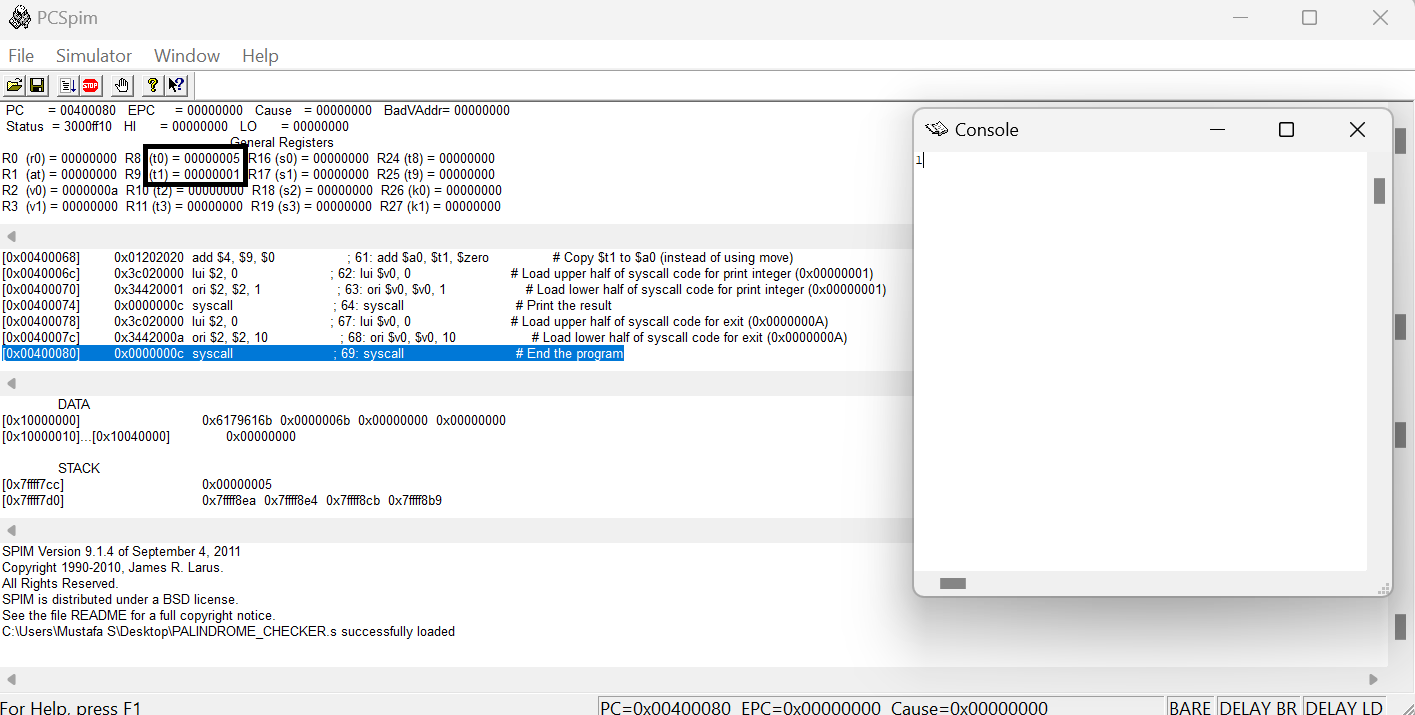
**Fourth Input: "818"**

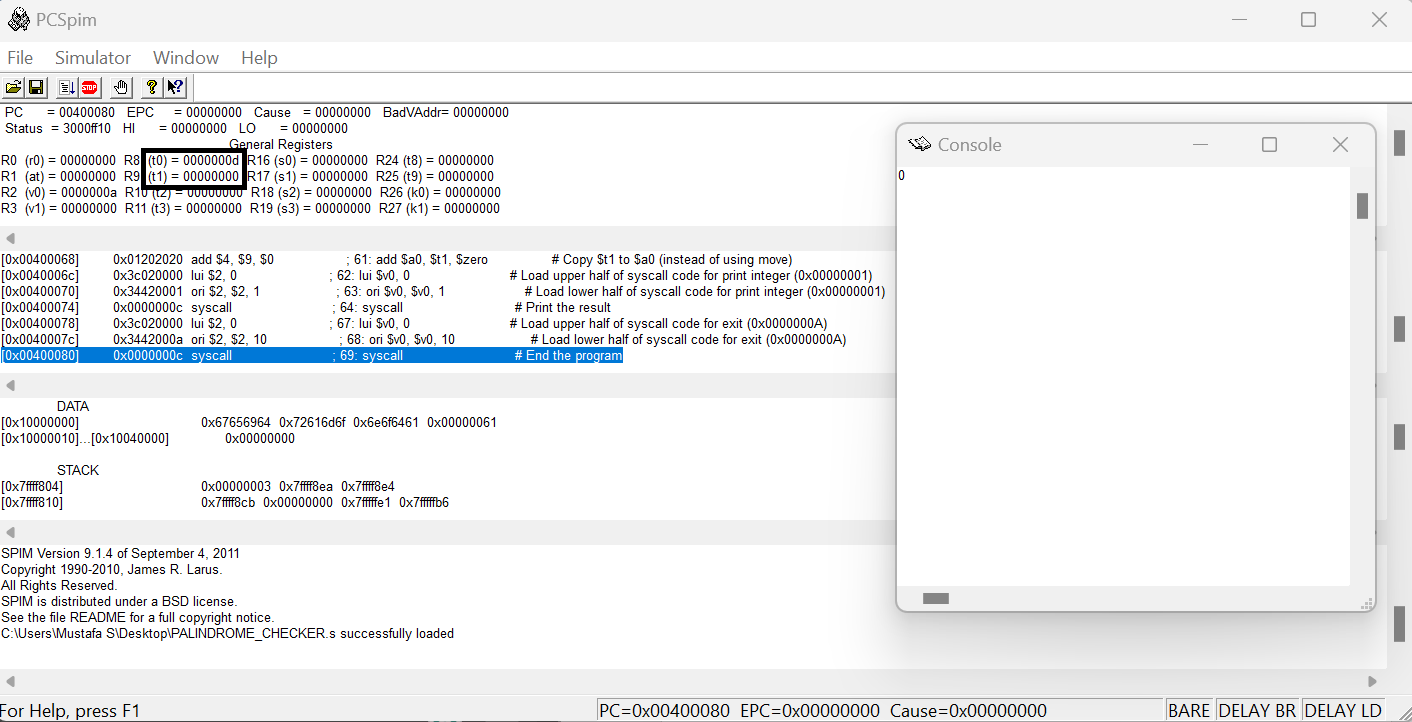
In the fourth input, "818," we used a string of numbers to observe the program's behavior with numerical characters treated as strings.

1. **Length Calculation:**
   * The program calculates the length of "818" as 3 characters.
   * This length is stored in the register R8 (t0) as **00000003** in hexadecimal.
2. **Palindrome Check:**
   * The program compares the numerical characters from both ends of the string.
   * Since "818" reads the same backward and forward, it is identified as a palindrome.
   * The result is stored in the register R9 (t1) as **00000001**, indicating the string is a palindrome.

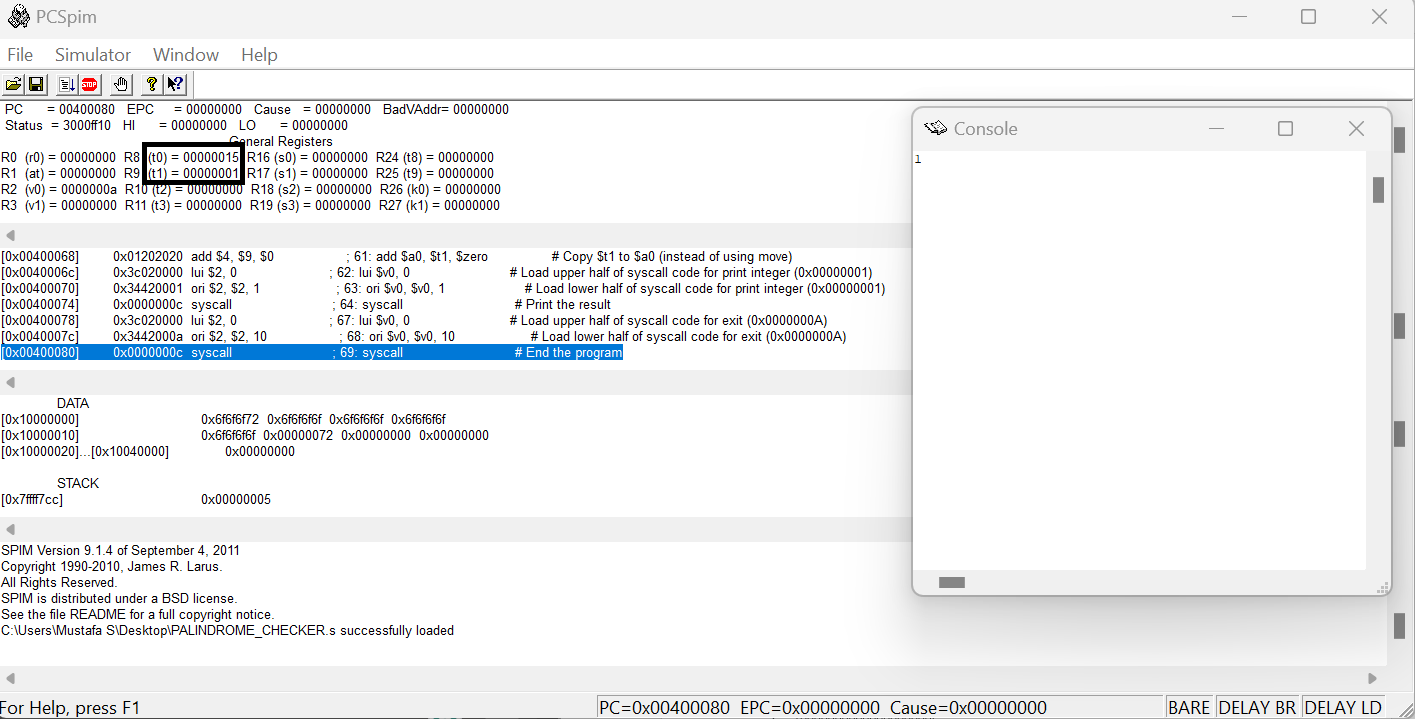
# 6.0 Test results

This section is going to provide screenshots of our MIPS code:

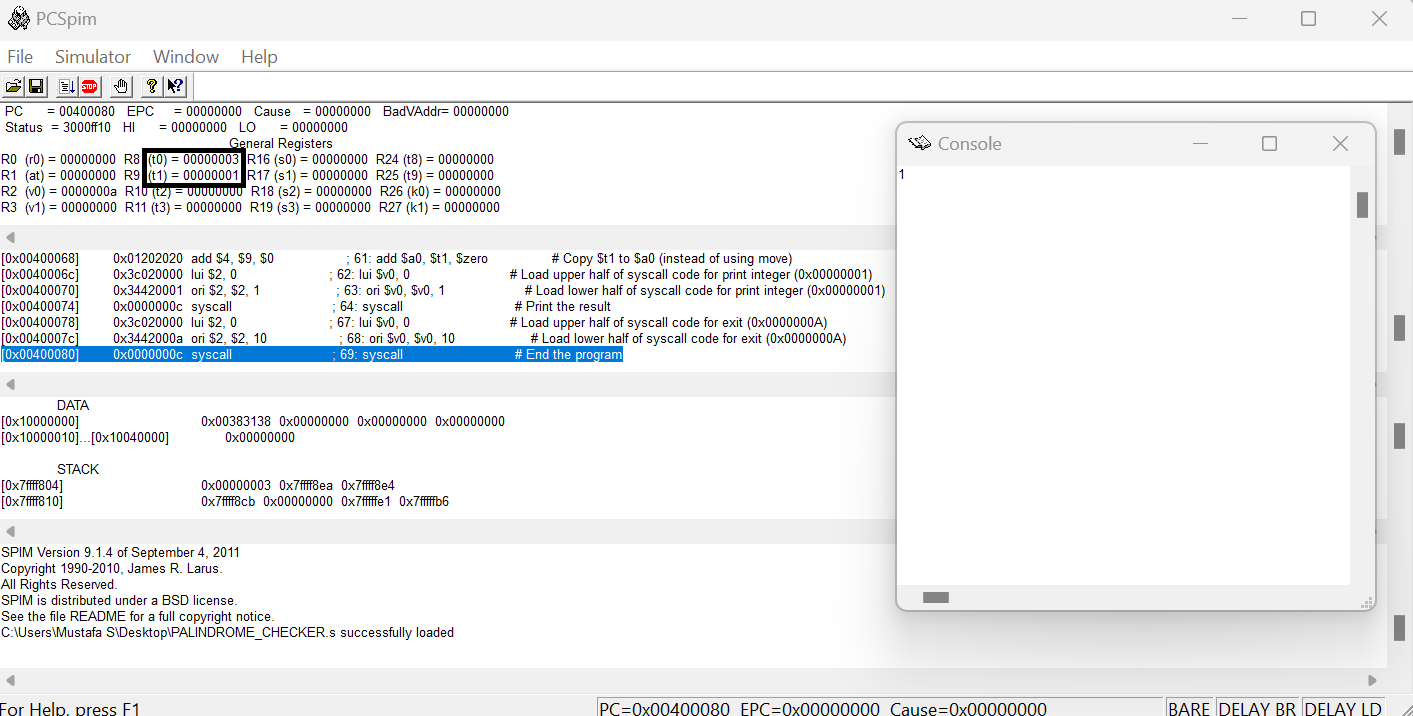
1. “kayak”
2. “diegomaradona”



1. “rooooooooooooooooooor”



1. “818”



# Summary

The Palindrome Checker is a MIPS assembly program designed to verify whether a given string is a palindrome. A palindrome is a sequence of characters that reads the same forward and backward. The program processes a predefined string, calculates its length, and compares characters from both ends towards the center to determine if the string is a palindrome. The result is then printed: **1** if the string is a palindrome and **0** if it is not.

The program starts by initializing a length counter and loading the base address of the string. It then iterates through the string to calculate its length, incrementing the counter for each character until the null terminator is encountered. After determining the length, the program sets up pointers for the palindrome check by resetting the starting pointer and setting an end pointer based on the string length. The palindrome flag is initially set to 1, assuming the string is a palindrome unless proven otherwise.

In the palindrome check loop, characters from the beginning and the end of the string are compared. If any characters do not match, the palindrome flag is set to 0, indicating the string is not a palindrome. The pointers are then moved towards the center, and the comparison continues until all characters are checked or a mismatch is found.

The final value of the palindrome flag is printed: **1** if the string is a palindrome or **0** if it is not. The program then terminates.