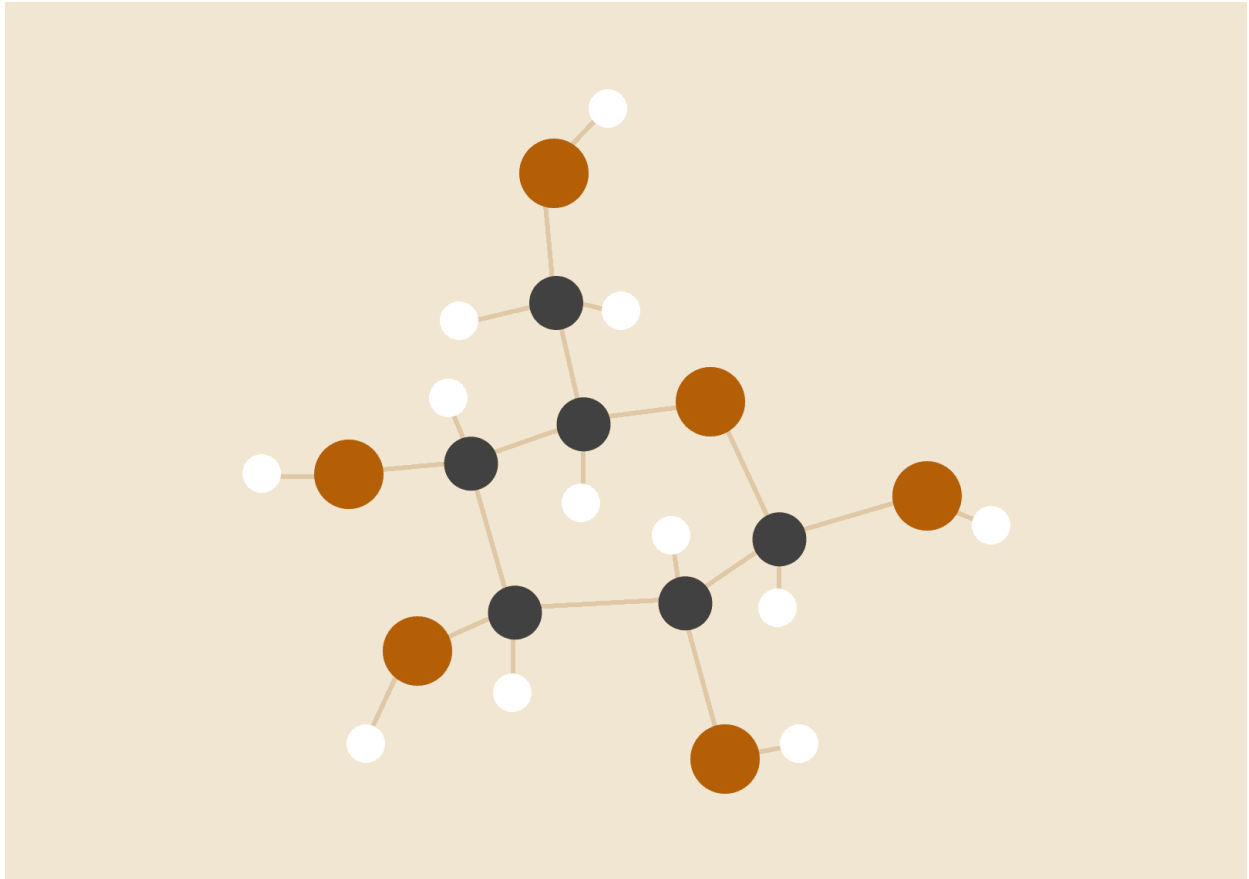


ASSIGNMENT 3

COMPUTER ARCHITECTURE



Yogita Mundankar

B22CS068

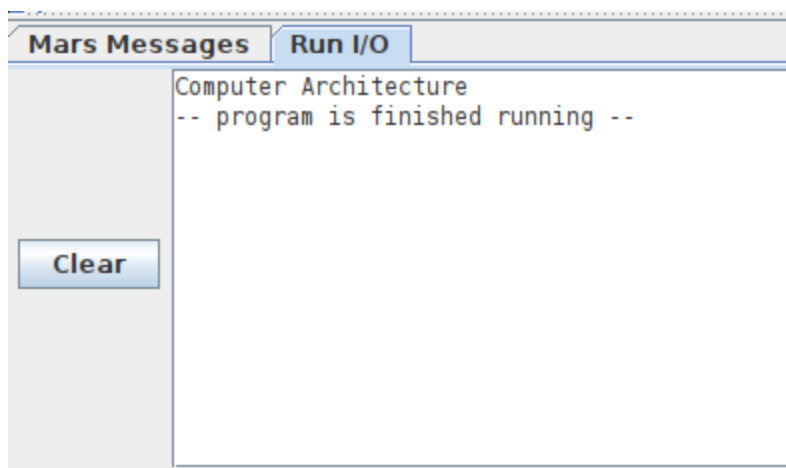
OBJECTIVE:

- String handling in MIPS
- Concatenating two strings
- Finding length of string
- Reversing a string
- Checking if a string is palindrome or not

SUBTASK 1: Concatenate two strings

Summary

1. **Data Section:**
 - Defines two null-terminated strings (`str1=" Computer "` and `str2="Architecture"`) and allocates space for the concatenated result.
2. **Text Section:**
 - **Copy `str1` to `result`:** Loads each byte from `str1` and copies it to `result` until the null terminator is encountered.
 - **Copy `str2` to `result`:** Continues copying bytes from `str2` to `result`, following the end of `str1`.
 - **Terminate `result`:** Appends a null terminator to the end of the concatenated string.
 - **Print `result`:** Outputs the concatenated string to the console.
 - **Exit Program:** Ends the program.



SUBTASK 2: Check If Palindrome

1. **Objective:** Verify if the input string is a palindrome by comparing it to its reversed version.
2. **Procedure:**
 - a. **Initialization:** Set up registers to point to the start of the original string (\$t0), the reversed string (\$t1), and the length of the string (\$t2). Initialize a counter (\$t5) to zero.
 - b. **Loop:** Compare each character of the original string with the corresponding character in the reversed string. If they do not match, jump to the not_Palindrome label.
 - c. **Check Completion:** Increment the counter and move to the next characters in both strings. Repeat until the end of the string or until all characters have been checked.
 - d. **Output Result:** If all characters match, print a message indicating the string is a palindrome; otherwise, print a message indicating it is not. End the program after printing the result.

SUBTASK 3: Reverse a string

Objective: Verify if the input string is a palindrome by comparing it to its reversed version.

Procedure:

1. **Initialization:** Set up registers to point to the start of the original string (\$t0), the reversed string (\$t1), and the length of the string (\$t2). Initialize a counter (\$t5) to zero.
2. **Loop:** Compare each character of the original string with the corresponding character in the reversed string. If they do not match, jump to the not_Palindrome label.
3. **Check Completion:** Increment the counter and move to the next characters in both strings. Repeat until the end of the string or until all characters have been checked.

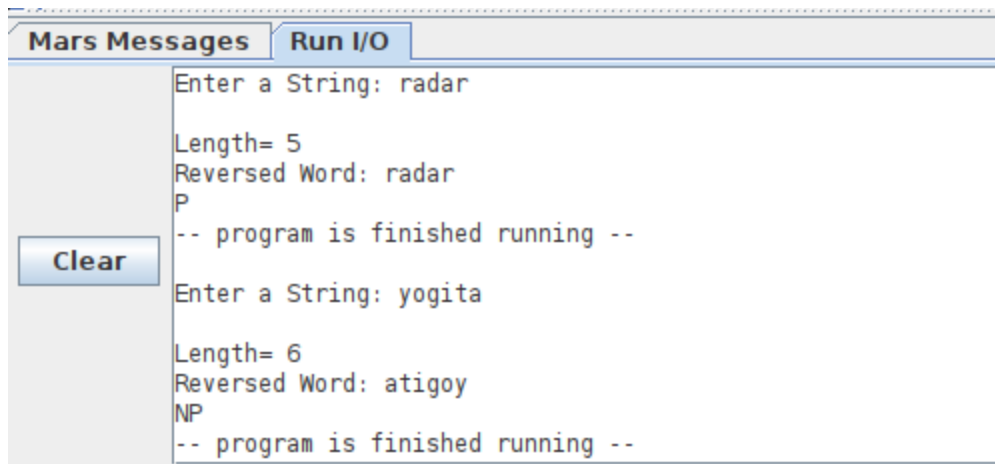
4. **Output Result:** If all characters match, print a message indicating the string is a palindrome; otherwise, print a message indicating it is not. End the program after printing the result.

-

SUBTASK 4: Finding length of the string

- **Objective:** Calculate the length of the input string (excluding the null terminator).
- **Procedure:**
 1. **Initialization:** Set the starting address of the input string in register \$t0 and initialize a counter (\$t1) to zero.
 2. **Loop:** Read each byte from the string using lb instruction. Check if the byte is a null terminator (\$zero).
 3. **Update Counter:** If not null, increment the counter and move to the next byte in the string.
 4. **End Loop:** When a null terminator is found, store the length (excluding the null terminator) in the length variable by subtracting 1 from the counter.
 5. **Output Length:** Print the length of the string and a newline character for clarity.

OUTPUT:



The screenshot shows a window titled "Mars Messages" with a "Run I/O" button. The output text is as follows:

```
Enter a String: radar
Length= 5
Reversed Word: radar
P
-- program is finished running --

Clear

Enter a String: yogita
Length= 6
Reversed Word: atigoy
NP
-- program is finished running --
```

CHALLENGES FACED:

1. Finding Length of String

Challenges:

- **Counting Null Terminator:** Initially, the length includes the null terminator, so you need to subtract one.
- **Handling End of String:** Ensuring the loop correctly identifies the end of the string.

Solutions:

- **Subtraction of One:** Subtract one from the length after the loop to exclude the null terminator.
- **Accurate Loop Conditions:** Ensure that the loop condition (`beq $t2, $zero, end_length`) correctly detects the null terminator.

2. Reversing the String

Challenges:

- **Pointer Arithmetic:** Correctly computing the end address of the string and managing pointer offsets.
- **Buffer Management:** Ensuring the reversed string is correctly null-terminated and stored in the reverse buffer.

Solutions:

- **Compute End Address:** Calculate the end address of the string by adding the length to the base address and adjusting for the null terminator.
- **Null-Terminate Properly:** Explicitly add a null terminator to the end of the reversed string to ensure proper string termination.

3. Checking If Palindrome

Challenges:

- **Character Comparison:** Accurately comparing characters from the original and reversed strings.
- **Loop Control:** Ensuring the loop terminates correctly after checking all characters.

Solutions:

- **Correct Comparison Logic:** Use `bne` to check if characters are not equal and branch to the `not_Palindrome` label if they differ.
- **Control Counter and Length:** Maintain a counter to ensure all characters are checked, and compare the counter with the string length to decide when to end the loop.

General Challenges:

- **Register Management:** Properly using and managing registers to store intermediate values and pointers.
- **Debugging:** Carefully tracing the code to ensure each subtask works as intended, especially handling edge cases and validating outputs.

Overcoming Challenges:

- **Testing and Validation:** Incrementally test each subtask to ensure correctness before integrating into the final program.
- **Using Comments:** Clearly document the code to understand the purpose of each instruction and subtask, aiding in debugging and future modifications.

Technical Learnings:

1. **Understanding MIPS Assembly Syntax and Semantics:**
 - Gained experience with MIPS instructions, registers, and `syscall` operations.
 - Learned how to use MIPS instructions for basic operations like loading, storing, and comparing data.
2. **String Manipulation in Assembly:**
 - Implemented string handling techniques, including reading, reversing, and concatenating strings.
 - Managed buffers effectively and ensured correct null-termination of

strings.

3. Loop Control and Conditional Branching:

- Developed a deeper understanding of loop constructs and conditional branching in assembly.
- Improved skills in controlling loop execution and managing program flow based on conditions.

4. System Calls and I/O Operations:

- Practiced using system calls for input and output operations, such as reading from the user and printing to the console.
- Learned how to handle and print integer values and strings using MIPS syscalls.

Learnings from the assignment:

Conceptual Learnings:

1. Low-Level Programming:

- Gained an appreciation for low-level programming and the challenges of managing data at the byte level.
- Understood the importance of precise memory management and pointer arithmetic.

2. Algorithm Implementation:

- Implemented algorithms for common string operations, such as finding the length and reversing strings.
- Applied algorithms to check for palindromes, reinforcing the importance of algorithm design and validation.

3. Debugging and Testing:

- Developed strategies for debugging and validating assembly code, including step-by-step execution and testing individual components.
- Recognized the need for careful error checking and handling in low-level programming.

4. Performance Considerations:

- Consider the efficiency of various operations and how low-level details impact performance.
- Learned to optimize code by minimizing unnecessary operations and improving loop efficiency.

Personal Growth:

1. Problem-Solving Skills:

- Enhanced problem-solving abilities by breaking down complex tasks into smaller, manageable subtasks.
- Gained experience in troubleshooting and resolving issues that arise during development.

2. Attention to Detail:

- Improved attention to detail by ensuring correct implementation of each step and handling edge cases.
- Appreciated the precision required in assembly language programming.

3. Adaptability:

- Adapted to the challenges of working with assembly language and embraced the learning process.

PROGRAM OUTPUT SUMMARY:

Find Length of String:

- Prints the length of the user-input string.

Reverse String:

- Displays the reversed version of the input string.

Check if Palindrome:

- Shows "P" if the string is a palindrome or "NP" if it is not.

Concatenate Two Strings:

- Outputs the concatenation of two predefined strings.

