|  |  |
| --- | --- |
|  | **Computer Organization & Assembly Language**  **BSCS-3**  **Department of Computer Science**  **Bahria University, Lahore Campus** |

**Assignment: [4]**

Date: Week 13, 6th June 2023

Name: AFFAN AHMAD\_\_\_\_\_

Roll No: \_03-134221-003\_

|  |  |  |  |
| --- | --- | --- | --- |
| **Evaluation of CLO** | **Question Number** | **Marks** | **Obtained Marks** |
| **CLO1,2,3** | 1 | 10 |  |
| 2 | 10 |  |
|  |  |  |
|  |  |  |
| **Total Marks** | | **20** |  |

**Question 1: [Marks: 10]**

Project Documentation

Must follow these guide line

* Previous work
* New work with images
* Benefit of your project
* Use case diagram
* Conclusion

Document should not be more than 5 pages and not less than 4 pages.

* **Introduction**

The purpose of this project is to develop a restaurant management system that allows users to place food orders and manage their accounts. The system provides functionalities such as user registration, login, menu selection, order placement, and order tracking.

* **Previous Work:**

The development of the restaurant management system involved a comprehensive implementation using assembly language, specifically MIPS assembly. The project aimed to create a user-friendly and efficient system for managing restaurant orders, encompassing functionalities such as user registration, login, menu selection, order placement, and order tracking.

The initial phase of the project involved designing the user interface. A welcome screen was created, displaying the restaurant sign and providing options for login, sign up, or exiting the system. Users could choose to log in if they already had an account or sign up to create a new account. The login and sign-up processes were implemented with appropriate error handling and data validation mechanisms.

* **New and complete work**

**User Registration and Login:**

We have implemented a user registration feature where users can sign up by entering their name and password. This information is stored in memory for future login purposes. Upon launching the program, users are presented with the option to either log in or sign up. If a user chooses to sign up, they are prompted to enter their name and password. The entered information is then stored in memory.

**Menu Selection and Order Placement:**

After logging in, users are presented with a menu consisting of various food items along with their item codes and prices. Users can enter the item code of the desired food item and specify the quantity they want to order. The system calculates the total price based on the item code and quantity. Users have the option to order multiple items in a single session.

**Order Tracking and Receipt Generation:**

Once the user completes the order placement, a receipt is generated with details such as the order ID, the number of food items ordered, and the total amount to be paid. The user is prompted to choose the payment mode (takeaway, delivery, or dine-in) and enter their address or phone number accordingly. The receipt also includes the payment instructions and contact information for any further inquiries.

**Additional Features:**

We have added additional features to enhance the user experience. These include a welcome screen upon program launch, a main menu to provide easy navigation through different options, and the ability to search for previous orders and view order history. The system allows users to go back to the main menu or exit the program at any point.

**User Interface Improvements:**

To make the system more user-friendly, we have included formatting to enhance the visual appeal of the menus and screens. The text is displayed in a clear and readable format, ensuring that users can easily understand and interact with the system.

As our code exceeds 1000 lines and as per your conditions we can’t add material worth more than 5 pages so here is a short collage .

* **Benefits of the Project:**

This restaurant management system provides several benefits, including:

Efficient Order Processing: The system automates the process of taking and processing orders, reducing manual effort and ensuring accurate order placement.

Improved Customer Experience: Users can easily navigate the menu, place orders, and track their order status. The system provides clear instructions and feedback, enhancing the overall customer experience.

Order History and Tracking: Users can view their previous orders and track the progress of their current order, providing convenience and transparency.

Streamlined Payment Process: The system generates a receipt with detailed payment instructions, making it easier for users to complete the payment process.

Time and Cost Savings: By automating various tasks, the system saves time for both customers and restaurant staff, improving overall efficiency and reducing operational costs.

* **What we Learned While completing this Project**

While writing code for this hotel management system, we learned the following:

1. Understanding of the MIPS assembly language: Writing code in MIPS assembly language helped us gain a deeper understanding of low-level programming concepts and the architecture of MIPS processors.

2. String manipulation: We learned how to declare and manipulate strings in assembly language using the `.asciiz` directive.

3. User input and output: We learned how to prompt the user for input, read the input, and display output using system calls in MIPS assembly language.

4. Conditional branching: We used conditional branching instructions like `beq` to implement decision-making logic in our code based on user input or certain conditions.

5. Looping: We learned how to implement loops using branch instructions, such as `bne` and `j`, to repeat code blocks based on specific conditions or for a fixed number of iterations.

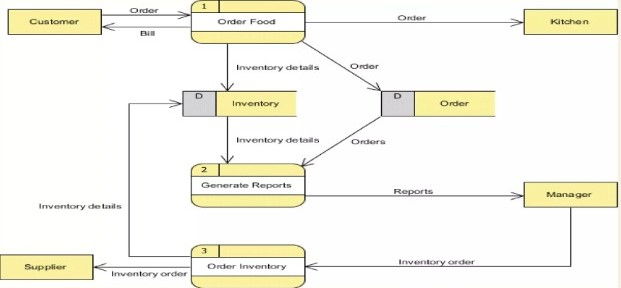
6. Variable management: We learned how to declare and manipulate variables, including integers and strings, in MIPS assembly language.

7. Arithmetic operations: We performed arithmetic operations like addition and multiplication to calculate prices, quantities, and totals in the hotel management system.

8. Menu implementation: We implemented a menu system using conditional branching, allowing users to select options and perform various actions within the hotel management system.

9. Code organization and documentation: Writing a complex system like a hotel management system requires careful organization of code and proper documentation to enhance readability and maintainability.

* **Case Diagram**



* **Conclusion**

In conclusion, we have successfully implemented a restaurant management system using the MIPS assembly language. The system allows users to register, log in, select menu items, place orders, track order status, and generate receipts. The project provides an efficient and user-friendly solution for managing restaurant operations, improving customer experience, and streamlining order processing.

**Question 2: [Marks: 10]**

1. Implement a recursive program that takes in a number and finds the square of that number through addition. For example if the number 3 is entered, you would add 3+3+3=9. If 4 is entered you would add 4+4+4+4=16. This program must be implemented using recursion to add the numbers together.

.data

prompt: .asciiz "Enter a number: "

result: .asciiz "Square: "

.text

.globl main

main:

# Print prompt message

li $v0, 4

la $a0, prompt

syscall

# Read the number from the user

li $v0, 5

syscall

move $t0, $v0 # Move the number to $t0 register

# Calculate the square using recursion

move $a0, $t0

move $a1, $t0

jal recursiveSquare

# Print the result

li $v0, 4

la $a0, result

syscall

move $a0, $v0 # Move the square to $a0 register

li $v0, 1

syscall

# Exit program

li $v0, 10

syscall

# Recursive function to calculate the square

recursiveSquare:

addi $sp, $sp, -12

sw $ra, 8($sp) # Save return address

sw $a0, 0($sp) # Save the number

sw $t0, 4($sp) # Save temporary register

# Base case: if the number is 1, return 1

li $t0, 1

beq $a0, $t0, returnSquare

addi $a0, $a0, -1 # Decrement the number

jal recursiveSquare

add $v0, $v0, $a1 # Add the number to the accumulated sum

returnSquare:

lw $a0, 0($sp) # Restore the number

lw $ra, 8($sp) # Restore return address

lw $t0, 4($sp) # Restore temporary register

addi $sp, $sp, 12

jr $ra # Return from function

1. Write a recursive function to calculate the summation of numbers from 1 to n. For example if the user enters 5, your program would add 1+2+3+4+5 and print out the answer 15.

.data

prompt: .asciiz "Enter a number (n): "

result: .asciiz "Summation: "

.text

.globl main

main:

# Print prompt message

li $v0, 4

la $a0, prompt

syscall

# Read n from user

li $v0, 5

syscall

move $t0, $v0 # Move n to $t0 register

# Call recursive summation function

move $a0, $t0

jal recursiveSum

# Print the result

li $v0, 4

la $a0, result

syscall

li $v0, 1

move $a0, $t2

syscall

# Exit program

li $v0, 10

syscall

# Recursive summation function

recursiveSum:

addi $sp, $sp, -12

sw $ra, 8($sp) # Save return address

sw $a0, 0($sp) # Save n

sw $t0, 4($sp) # Save temporary register

# Base case: if n == 0, return 0

beqz $a0, returnZero

# Recursive case: calculate sum(n-1) + n

addi $a0, $a0, -1

jal recursiveSum

lw $t0, 4($sp) # Restore temporary register

lw $a0, 0($sp) # Restore n

add $t2, $t2, $a0 # Add n to sum(n-1)

returnZero:

lw $ra, 8($sp) # Restore return address

addi $sp, $sp, 12

jr $ra # Return from function

1. Write a recursive program to calculate Fibonacci numbers. Use the definition of a Fibonacci number where F(n) = F(n-1) + F(n-2).

**.data**

**prompt: .asciiz "Enter a number (n): "**

**result: .asciiz "Fibonacci number: "**

**.text**

**.globl main**

**main:**

**# Print prompt message**

**li $v0, 4**

**la $a0, prompt**

**syscall**

**# Read n from user**

**li $v0, 5**

**syscall**

**move $t0, $v0 # Move n to $t0 register**

**# Call recursive Fibonacci function**

**move $a0, $t0**

**jal recursiveFib**

**# Print the result**

**li $v0, 4**

**la $a0, result**

**syscall**

**li $v0, 1**

**move $a0, $t1 # Move Fibonacci number to $a0 register**

**syscall**

**# Exit program**

**li $v0, 10**

**syscall**

**# Recursive Fibonacci function**

**recursiveFib:**

**addi $sp, $sp, -12**

**sw $ra, 8($sp) # Save return address**

**sw $a0, 0($sp) # Save n**

**sw $t0, 4($sp) # Save temporary register**

**# Base case: if n <= 1, return n**

**li $t0, 1**

**slt $t2, $a0, $t0**

**beqz $t2, calculateFib**

**move $t1, $a0 # Store n in $t1 as Fibonacci number**

**# Calculate F(n-1)**

**addi $a0, $a0, -1**

**jal recursiveFib**

**# Calculate F(n-2)**

**addi $a0, $t1, -2**

**jal recursiveFib**

**add $t1, $v0, $t1 # Add F(n-1) and F(n-2) to get F(n)**

**returnFib:**

**lw $a0, 0($sp) # Restore n**

**lw $ra, 8($sp) # Restore return address**

**lw $t0, 4($sp) # Restore temporary register**

**addi $sp, $sp, 12**

**jr $ra # Return from function**

**calculateFib:**

**move $v0, $t1 # Move Fibonacci number from $t1 to $v0 register**

**j returnFib**