Benha University
Faculty Of Engineering (Shoubra)
Communications And Computer Engineering

Calculator History Using LinkedList

Under the supervision of
Dr. Lamia Alrefaai

Summary

This MIPS assembly program functions as an all-encompassing computational solution, incorporating an interactive menu catering to mathematical operations, history browsing, and result administration. Upon user command, it performs arithmetic computations utilizing floating-point inputs and allows for the storage of results in a linked list. Users retain the ability to review past calculations, remove specific entries, or employ the latest result for subsequent computations. The program's architecture includes initialization, constant declaration, and function implementation to manage arithmetic operations and linked list functionalities. In essence, it amalgamates fundamental arithmetic capabilities with streamlined data management, delivering a flexible computation platform underscored by a robust focus on tracking and managing computation history.

Abstract

The program serves as a versatile tool, offering users a menu-driven interface to perform various statistical operations on floating-point numbers. By presenting a clear and concise menu, users can easily navigate through available options, selecting the desired action effortlessly. Whether they need to perform basic arithmetic operations or manipulate data sets, the program accommodates diverse needs with its flexible functionalities. Additionally, features like result history, deletion capabilities, and result reuse enhance user convenience and productivity, fostering a seamless experience for mathematical tasks and analysis.

Introduction

This MIPS assembly program offers a comprehensive suite of arithmetic functionalities for floating-point numbers. Through carefully crafted algorithms, it executes addition, subtraction, multiplication, and division operations with precision. The program employs a Linked List data structure to efficiently manage the results of these operations, providing users with seamless access to their calculations. Notably, users can not only perform arithmetic but also view and delete previously stored results, enhancing the program's utility and user-friendliness.

I. Code Description

1. Data Definitions and Constants for Linked List Operations

```
.data
    # Prompt messages for user selection
    prompt:           .asciiz "\n1.Addition.\n2.Subtraction.\n3.Multiplication.\n4.Division.\n5.View history.\n6.Delete no..\n7.Use previous result.\n8.Previous result.\n9.Exit.\n"
    # Constants for handling floats
    zeroFloat:        .float 0.0            # zero as a float to be used in moving floats
    # Size of each node in bytes
    node_size:        .word  8
    # Newline character for separating values
    newline:          .asciiz "\n"
    # Error messages
    nodeNotFound:     .asciiz "Number not found"
    noPreviousNumber: .asciiz "There is no previous number.\n"
    msg_empty:        .asciiz "No more values to display.\n"
    # Linked list pointers
    list_head:        .word  0         # Pointer to the head of the list
```

1. Prompt Messages (‘*prompt*’):

- This constant holds the prompt message displayed to the user for selecting different operations. It includes options for addition, subtraction, multiplication, division, viewing history, deleting a number, using the previous result, displaying the previous result, and exiting the program.

2. Constants for Handling Floats (‘*zeroFloat*’):

- It defines a constant ‘***.float 0.0***’, which represents zero as a floating-point number. This value is used for initializing floating-point registers and handling floating-point arithmetic operations.

3. Node Size (‘*node_size*’):

- This constant specifies the size of each node in the linked list, measured in bytes. In this case, each node is defined to be 8 bytes in size.

4. Newline Character (‘*newline*’):

- It defines the newline character (‘***\n***’) used for formatting output by separating values or messages on different lines.

5. Error Messages:

- ‘***nodeNotFound***’: This message indicates that a particular number was not found in the linked list.
- ‘***noPreviousNumber***’: It indicates that there is no previous number available for use in a mathematical operation.
- ‘***msg_empty***’: This message informs the user that there are no more values to display in the linked list.

6. Linked List Pointers (‘*list_head*’):

- This pointer holds the memory address of the head of the linked list. Initially, it is set to zero, indicating an empty list. As nodes are added to the list, this pointer is updated to point to the first node.

2. Main Control Flow and User Interaction

```
17  .text
18  main:
19      # Load zero float value into register $f11
20      lwc1 $f11, zeroFloat
21
22  start:
23      # Print the menu prompt
24      li $v0,4
25      la $a0, prompt
26      syscall
27
28      # Get user's selection
29      li $v0, 5
30      syscall
31
32      # Branch based on user's input
33      beq $v0,1,Addition
34      beq $v0,2,Subtraction
35      beq $v0,3,Multiplication
36      beq $v0,4,Division
37      beq $v0,5,print
38      beq $v0,6,delete_float
39      beq $v0,7,use_previous_result
40      beq $v0,8,printItems
41      beq $v0,9,Exit
42
43  Exit:
44      # Terminate the program
45      li $v0, 10
46      syscall
```

1. Main:

- This is the entry point of the program. It's the main function where the execution begins.

2. Load zero float value:

- This line loads a float value of zero into register '***$f11***'. It's probably used for initialization or comparison purposes later in the code.

3. Start:

- This label marks the beginning of the main loop where the program waits for user input and performs actions based on that input.

4. Print the menu prompt:

- The program prints the menu prompt to the console, displaying options for the user to choose from.

5. Get user's selection:

- It reads the user's selection from the console input.

6. Branch based on user's input:

- Depending on the user's input, the program branches to different sections of the code to perform specific actions. For example, if the user chooses option 1, it branches to the "***Addition***" section of the code.

7. Exit:

- If the user chooses to exit the program (option 9), the program terminates by invoking the '***syscall***' with the exit code.

3. Float Input Handling in Arithmetic Operations

```
48  # Function to get two floats used by mathematical functions
49  GetTwoFloats:
50      # Check if only one float is needed
51      beq $t7, 1, GetOneFloat
52
53      # Get the first float
54      li $v0, 6
55      syscall
56      add.s $f1, $f0, $f11 # Move the input to f1
57
58  GetOneFloat:
59      # Get the second float
60      li $v0, 6
61      syscall
62      add.s $f2, $f0, $f11 # Move the second input to f2
63
64      # Reset flag to avoid continuously using the previous result
65      li $t7, 0
66      jr $ra
```

The '***GetTwoFloats***' function is responsible for obtaining one or two floating-point numbers from the user. It checks a flag ('***$t7***') to determine whether one or two floats are needed.

1. Check Flag:

- The function begins by checking the value of the flag '***$t7***'.
- If '***$t7***' is equal to 1, it means that only one float is needed, so it branches to the label '***GetOneFloat***'.
- If '***$t7***' is not equal to 1, it proceeds to get the first float directly.

2. Get First Float:

- It prompts the user to enter a floating-point number using system call '***li $v0, 6***' (which corresponds to the "read float" system call).
- After obtaining the input, it stores the float value in register '***$f1***'.

- It adds the value of '***$f0***' (the input) to '***$f11***' (a predefined constant representing 0.0) and stores the result in '***$f1***'.

3. Get Second Float (Optional):

- If the flag indicated that only one float is needed, it skips this step.
- Otherwise, it proceeds to obtain the second floating-point number similar to the first one.
- The second float is stored in register '***$f2***'.

4. Reset Flag:

- After obtaining the required float(s), the function resets the flag '***$t7***' to 0.
- This ensures that subsequent operations won't mistakenly use the previous result.

5. Return:

- Finally, the function returns to the caller using the '***jr $ra***' instruction, which jumps back to the address stored in the return address register.

4. Result Display and History Recall

```
68  # Function to print the result of mathematical operations
69  PrintResult:
70      li $v0,2
71      syscall
72      jr $ra
73
74  # Function to use the previous result
75  use_previous_result:
76      # Load the address of the head of the linked list
77      lw $t6, list_head
78      # Check if the list is empty
79      bne $t6, $zero, ResultExists
80      # If list is empty, print error message
81      li $v0,4
82      la $a0,noPreviousNumber
83      syscall
84      j start
85
86  ResultExists:
87      # Set flag to ask for one number only from the user in the next operation
88      li $t7, 1
89      # Load the first value from the list into f1
90      lwc1 $f1, 0($t6)
91      j start
```

1. '*PrintResult*' Function:

- This function is responsible for printing the result of mathematical operations to the console.
- It sets the '***$v0***' register to 2, indicating that the system call should print an integer or a float value.

- Then, it invokes the '***syscall***' to perform the printing operation.
- Finally, it returns control to the caller using the '***$ra***' register.

2. '*use_previous_result*' Function:

- This function is intended to use the previous result stored in the linked list.
- It first loads the address of the head of the linked list into register '***$t6***'.
- Then, it checks if the list is empty by comparing '***$t6***' with zero. If the list is not empty, it branches to the label '***ResultExists***'.
- If the list is empty, it prints an error message indicating that there is no previous number using '***syscall***' with set to 4 (printing string) and loads the appropriate message into '***$a0***'.
- Finally, it jumps back to the '***start***' label, resuming the main loop.

3. '*ResultExists*' Label:

- This section is executed if there is a previous result available in the linked list.
- It sets a flag ('***$t7***') to indicate that only one number is needed from the user in the next operation.
- Then, it loads the value of the first node from the linked list into the floating-point register '***$f1***'.
- Finally, it jumps back to the '***start***' label, continuing with the main loop.

5. Addition Operation and Result Handling

```
93  # Addition function
94  Addition:
95      jal GetTwoFloats
96      add.s $f12, $f1, $f2
97      jal PrintResult # Print the result
98      jal insert_float # Insert the result into the linked list
99      lw $s5, list_head
00      j start
```

1. '*Addition*' Function:

- This function performs the addition operation.
- It first calls the '***GetTwoFloats***' subroutine using '***jal***' (jump and link) instruction. This subroutine is responsible for getting two float numbers from the user.
- After obtaining the two float numbers, it adds them together using the '***add.s***' instruction, which adds two single-precision floating-point numbers.
- Then, it calls the '***PrintResult***' subroutine to print the result of the addition operation to the console.
- After printing the result, it calls the '***insert_float***' subroutine to insert the result into the linked list.
- Finally, it loads the address of the head of the linked list into register '***$s5***' and jumps back to the '***start***' label to resume the main loop.

2. '*j start*':

- This line of code is a jump instruction. It directs the program to jump to the '***start***' label, which marks the beginning of the main loop.
- Essentially, after completing the addition operation and updating the linked list, the program returns to the main loop to wait for further user input and continue with the calculator functionalities.

6. Subtraction Operation and Result Handling

```
102  # Subtraction function
103  Subtraction:
104      jal GetTwoFloats
105      sub.s $f12, $f1, $f2
106      jal PrintResult # Print the result
107      jal insert_float # Insert the result into the linked list
108      lw $s5, list_head
109      j start
```

1. '*Subtraction*' Function:

- This function performs the subtraction operation.
- It starts by calling the '***GetTwoFloats***' subroutine using the '***jal***' (jump and link) instruction. This subroutine retrieves two float numbers from the user.
- Once the two float numbers are obtained, it subtracts the second number from the first number using the '***sub.s***' instruction, which subtracts two single-precision floating-point numbers.
- After calculating the result, it calls the '***PrintResult***' subroutine to print the result of the subtraction operation to the console.
- Following that, it calls the '***insert_float***' subroutine to insert the result into the linked list for future reference.
- Finally, it loads the address of the head of the linked list into register '***$s5***' and prepares to jump back to the '***start***' label to continue the main loop.

2. '*j start*':

- This line of code is an unconditional jump instruction. It directs the program to jump back to the '***start***' label, which marks the beginning of the main loop.
- After completing the subtraction operation, updating the linked list, and preparing the register for the next iteration, the program returns to the main loop to wait for further user input and continue the calculator functionalities.

7. Multiplication Operation and Result Handling

```
111  # Multiplication function
112  Multiplication:
113      jal GetTwoFloats
114      mul.s $f12, $f1, $f2
115      jal PrintResult # Print the result
116      jal insert_float # Insert the result into the linked list
117      lw $s5, list_head
118      j start
```

1. '*Multiplication*' Function:

- This function is responsible for performing the multiplication operation.
- It begins by calling the '***GetTwoFloats***' subroutine using the '***jal***' (jump and link) instruction. This subroutine retrieves two float numbers from the user.
- Once the two float numbers are obtained, it multiplies them using the '***mul.s***' instruction, which multiplies two single-precision floating-point numbers.
- After calculating the result, it calls the '***PrintResult***' subroutine to print the result of the multiplication operation to the console.
- Following that, it calls the '***insert_float***' subroutine to insert the result into the linked list for future reference.
- Finally, it loads the address of the head of the linked list into register '***$s5***' and prepares to jump back to the '***start***' label to continue the main loop.

2. *'j start'*:

- This line of code is an unconditional jump instruction. It directs the program to jump back to the '***star***' label, which marks the beginning of the main loop.
- After completing the multiplication operation, updating the linked list, and preparing the register for the next iteration, the program returns to the main loop to wait for further user input and continue the calculator functionalities.

8. Division Operation and Result Management

```
120  # Division function
121  Division:
122      jal GetTwoFloats
123      div.s $f12, $f1, $f2
124      jal PrintResult # Print the result
125      jal insert_float # Insert the result into the linked list
126      lw $s5, list_head
127      j start
```

1. *'Division'* Function:

- This function is responsible for performing the division operation.

- It begins by calling the '***GetTwoFloats***' subroutine using the '***jal***' (jump and link) instruction. This subroutine retrieves two float numbers from the user.
- Once the two float numbers are obtained, it divides the first number by the second number using the '***div.s***' instruction, which divides two single-precision floating-point numbers.
- After calculating the result, it calls the '***PrintResult***' subroutine to print the result of the division operation to the console.
- Following that, it calls the '***insert_float***' subroutine to insert the result into the linked list for future reference.
- Finally, it loads the address of the head of the linked list into register '***$s5***' and prepares to jump back to the '***start***' label to continue the main loop.

2. *'j start'*:

- This line of code is an unconditional jump instruction. It directs the program to jump back to the '***start***' label, which marks the beginning of the main loop.
- After completing the division operation, updating the linked list, and preparing the register for the next iteration, the program returns to the main loop to wait for further user input and continue the calculator functionalities.

9. Linked List Printing Functionality

```
129  # Function to print the values in the linked list
130  print:
131      # Load the address of the head of the list
132      lw $t0, list_head
133      j print_loop
134
135  print_loop:
136      # If current node is null, end loop
137      beq $t0, $zero, print_end
138      # Load float value of current node
139      lwc1 $f12, 0($t0)
140      # Print float value
141      li $v0, 2
142      syscall
143      # Print newline
144      li $v0, 4
145      la $a0, newline
146      syscall
147      # Move to next node
148      lw $t0, 4($t0)
149      j print_loop
150
151  print_end:
152      # Return to the start
153      lw $t0, list_head
154      j start
```

1. *'Print'* Function:

- This function is responsible for printing the values stored in the linked list.
- It starts by loading the address of the head of the linked list into register '***$to***'.
- Then, it jumps to the '***print_loop***' label to begin iterating through the linked list and printing each value.

2. *'print_loop'*:

- This label marks the beginning of the loop that iterates through the linked list and prints its values.
- It first checks if the current node (pointed to by register '***$to***') is null. If it's null, indicating the end of the linked list, the loop ends, and the program jumps to the '***print_end***' label.
- If the current node is not null, it loads the float value stored in that node into floating-point register '***$f12***'.
- Then, it prints the float value to the console using '***syscall***' with '***$v0***' set to 2, which indicates printing a float value.
- After printing the float value, it prints a newline character to format the output.
- Finally, it moves to the next node in the linked list by loading the address of the next node (stored as the second word in the current node) into register '***$to***' and jumps back to the '***print_loop***' label to continue the iteration.

3. *'print_end'*:

- This label marks the end of the '***print***' function.
- After printing all values in the linked list, the program returns to the start of the main loop by loading the address of the head of the linked list into register '***$to***' and jumping to the '***start***' label.

10. Linked List Float Insertion Function

```
156 # Function to insert a float into the linked list
157 # $f12 = value to insert
158 insert_float:
159     # Allocate memory for new node
160     li $v0, 9
161     lw $a0, node_size
162     syscall
163     move $t0, $v0
164
165     # Store float value in the new node
166     swc1 $f12, 0($t0)
167
168     # Insert node at the beginning of the list
169     lw $t1, list_head
170     sw $t1, 4($t0)
171     sw $t0, list_head
172     jr $ra
```

1. '*insert_float*' Function:

- This function is responsible for inserting a float value into the linked list.
- It begins by allocating memory for a new node in the linked list. This is done by setting '***$v0***' to 9, indicating a '***syscall***' for memory allocation, and loading the node size into **'$a0'**. The '***syscall***' allocates memory for a new node and returns the address of the allocated memory in '***$v0***', which is stored in '***$t0***'.
- Next, it stores the float value ('***$f12***') into the newly allocated node. This is achieved by using the swc1 (store word from Coprocessor 1) instruction to store the float value in the memory location pointed to by '***$t0***'.
- Finally, it inserts the new node at the beginning of the linked list. It does this by updating the pointer of the new node to point to the current head of the list ***($list_head)***. Then, it updates the '***$list_head***' to point to the newly inserted node, effectively making it the new head of the list.

2. *'jr $ra'*:

- This line of code is a jump instruction that returns control to the calling function.
- '***$ra***' (return address) contains the address of the instruction following the '***jal***' (jump and link) instruction that called the current function. So, by jumping to '***$ra***', the program returns to the instruction immediately after the '***jal***' instruction that called the '***insert_float***' function.
- Essentially, after inserting the float value into the linked list, this line ensures that the program returns to the point in the code where the insertion was requested.

11. Linked List Float Deletion Function

```
174  # Function to delete a node containing a specific float value
175  # $f12 = float value to delete
176
177  delete_float:
178      li $v0,6
179      syscall
180      lw $t0, list_head
181      li $t2, 0
182
183  delete_loop:
184      beq $t0, $zero, delete_end
185      lwc1 $f1, 0($t0)
186      c.eq.s $f1, $f0
187      bc1t delete_node
188      move $t2, $t0
189      lw $t0, 4($t0)
190      j delete_loop
```

```
191
192 delete_node:
193     beq $t2, $0, DeleteHead
194     lw $t1, 4($t0)
195     sw $t1, 4($t2)
196     j start
197
198 DeleteHead:
199     lw $t2, 4($t0)
200     sw $t2, list_head
201     lw $s5, list_head
202     j start
203
204 delete_end:
205     # Print message indicating the node was not found
206     li $v0, 4
207     la $a0, nodeNotFound
208     syscall
209     j start
```

1. '*delete_float*' Function:

- This function is responsible for deleting a node from the linked list that contains a specific float value.
- It begins by prompting the user to input the float value they want to delete. This is done using '***syscall***' 6 (read float) and storing the input value in register '***$f0***'.
- Then, it loads the address of the head of the linked list into register '***$t0***'.
- It initializes '***$t2***' to 0, which will be used to keep track of the previous node's address.
- The function then enters a loop labeled '***delete_loop***':
  - It checks if the current node is null. If it is null, the loop ends, and the program jumps to the '***delete_end***' label.
  - Inside the loop, it loads the float value stored in the current node into register '***$f1***'.
  - It checks if the float value in the current node matches the float value input by the user (stored in '***$f0***'). If they match, it jumps to the '***delete_node***' label to delete the current node.
  - If the values don't match, it moves to the next node in the linked list.

- The '***delete_node***' label handles the deletion of the current node:
  - If the node to be deleted is not the head of the linked list, it updates the pointer of the previous node to skip over the node to be deleted.
  - If the node to be deleted is the head of the linked list, it updates the '***$list_head***' to point to the next node.
- After deleting the node, the program jumps back to the '***start***' label to continue the main loop.

2. *'j start'*:

- This line of code is an unconditional jump instruction. It directs the program to jump back to the '***start***' label, which marks the beginning of the main loop.
- After deleting the node containing the specific float value (if found) or indicating that the node was not found, the program returns to the main loop to await further user input and continue the calculator functionalities.

12. Node Data Printing Function

```
211  # Function to print the data of the first node in the list
212  printItem:
213      # Check if the list is empty
214      beq $s5, $zero, empty
215      # Print the data of the first node
216      li $v0, 2
217      lwc1 $f12, 0($s5)
218      syscall
219      lw $s5,4($s5)
220      j end
221
222  empty:
223      # Print message indicating that the list is empty
224      li $v0, 4
225      la $a0, msg_empty
226      syscall
227
228  end:
229      j start
```

1. *'PrintItem'* Function:

- This function is responsible for printing the data of the first node in the linked list.
- It begins by checking if the linked list is empty. It does so by comparing the value in register '***$s5***' (which presumably holds the address of the head of the linked list) with zero. If it's zero, it means the list is empty, and the program jumps to the '***empty***' label.
- If the list is not empty, it proceeds to print the data of the first node:
  - It sets '***$v0***' to 2, indicating a system call for printing a float value.
  - It loads the float value stored in the first node of the linked list into floating-point register '***$f12***'.
  - It prints the float value to the console using '***syscall***'.
  - It then updates the pointer to the head of the linked list to point to the next node.
  - Finally, it jumps to the '***end***' label.

2. *'empty'*:

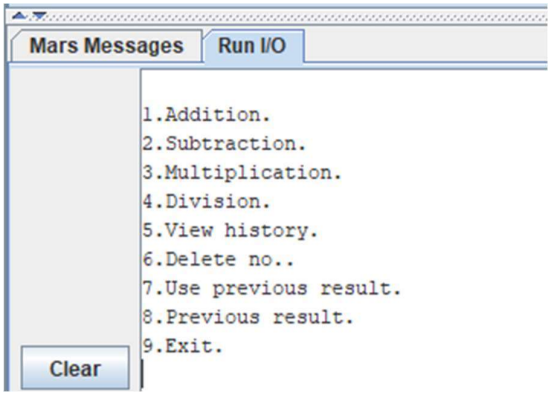
- If the linked list is empty, this part of the code is executed.
- It sets '***$v0***' to 4, indicating a system call for printing a string.
- It loads the address of the string message indicating that the list is empty into register '***$a0***'.
- It invokes the '***syscall***' to print the message to the console.

3. *'end'*:

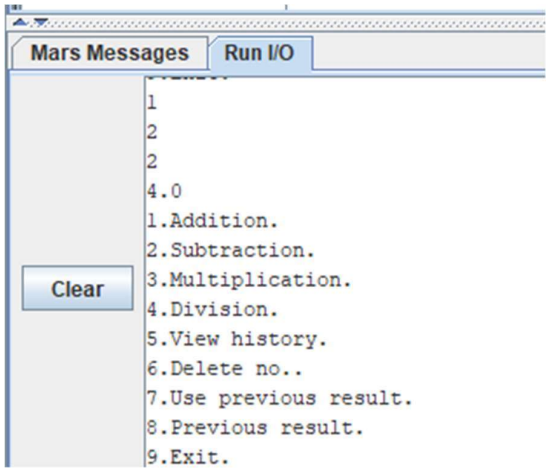
- This label marks the end of the '***printFirstItem***' function.
- After printing the data of the first node or indicating that the list is empty, the program jumps to the '***start***' label, which marks the beginning of the main loop.
- Essentially, after printing the data or message, the program returns to the main loop to await further user input and continue the calculator functionalities.

II. Running Code

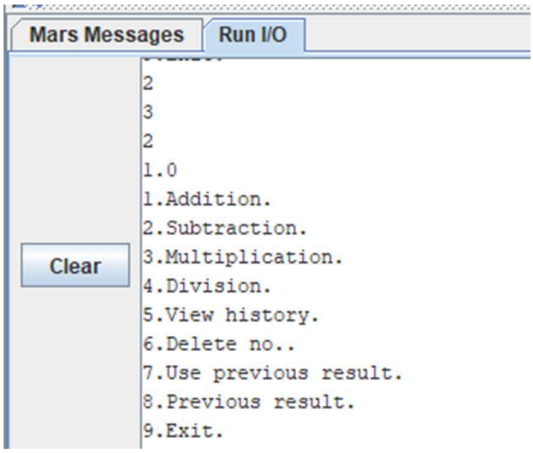
First when we run the program:



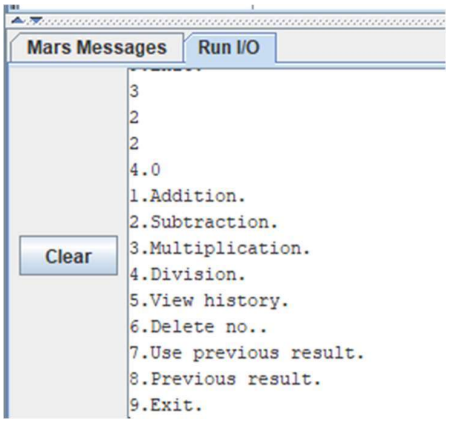
When the user selects option 1 for 'Addition' and inputs, for instance, the numbers 2 and 2..



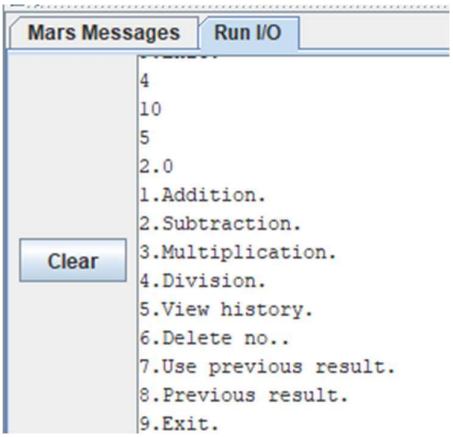
When the user selects option 2 for 'Subtraction' and provides, for example, the numbers 3 and 2...



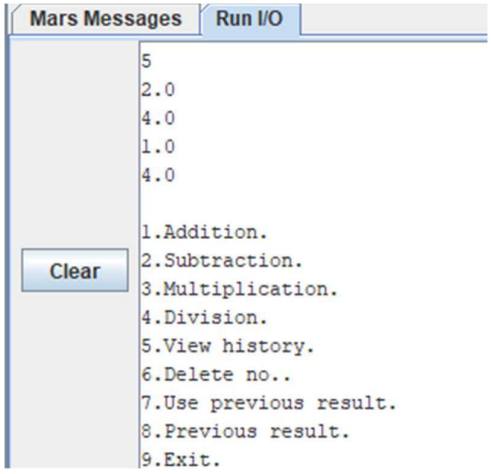
When the user selects option 3 for 'Multiplication' and inputting, for instance, the numbers 2 and 2...



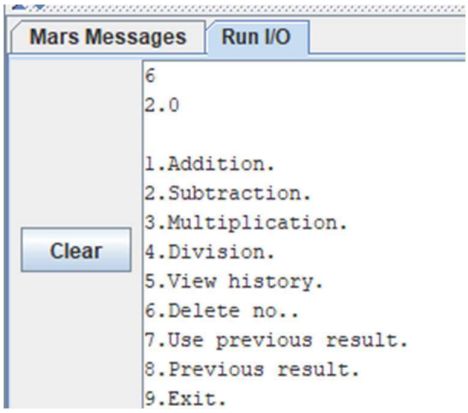
When the user selects option 4 for 'Division' and inputs, for example, the numbers 10 and 5...



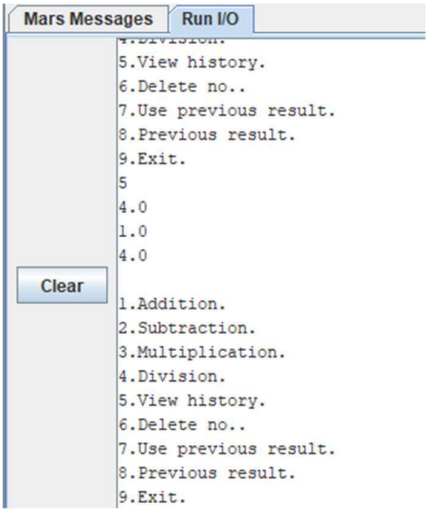
When the user selects option 5 for 'View history':



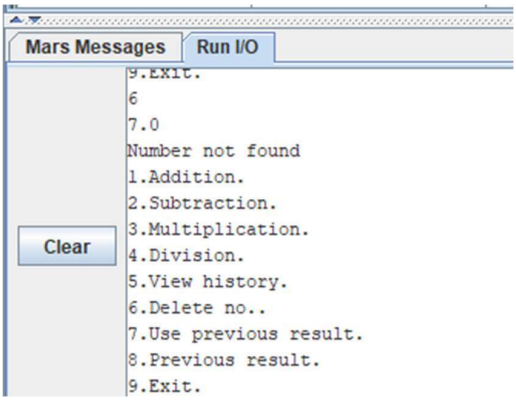
If the user selects option 6 for 'Delete no.' and inputs a number that exists in the history, such as 2.0, the program will proceed to delete the specified number..



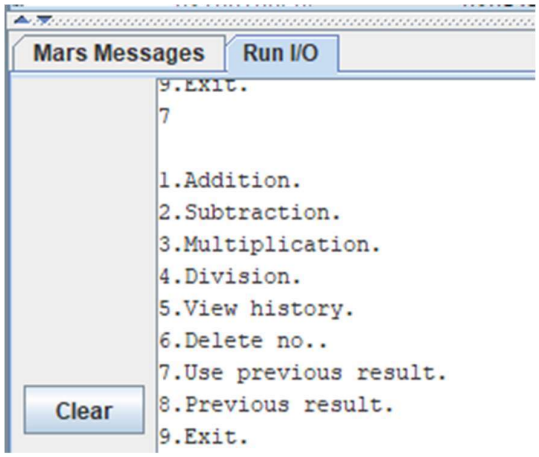
Then when the user selects option 5 for 'View history':

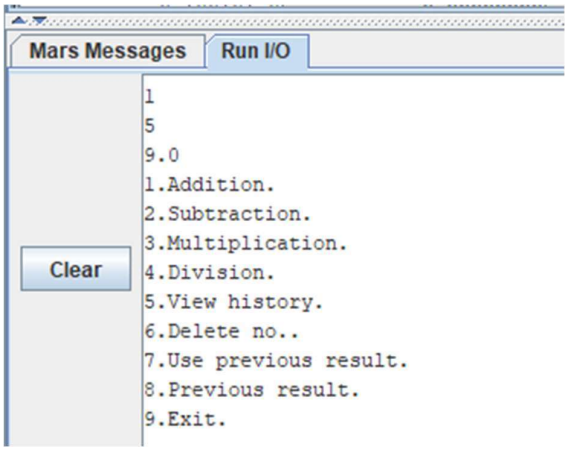


If the user selects option 6 for 'Delete no. ' and inputs a number that not exists in the history, such as 7.0..

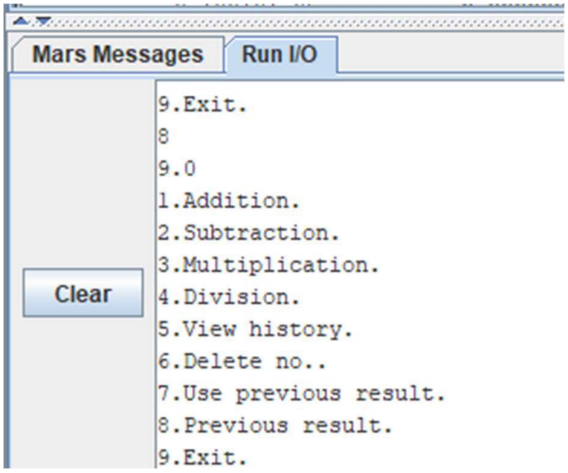


Upon selecting option 7 for 'Use previous result', the program will prompt the user to enter a single number. It will then perform the selected operation using this number and the previous result stored in memory. For instance, if the user enters the number 5...

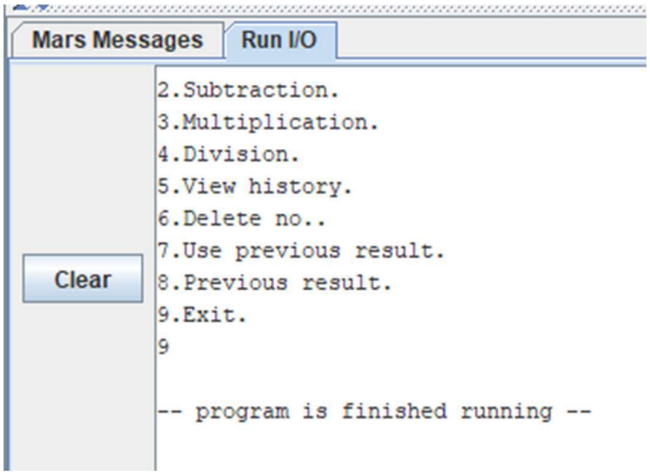




When the user selects option 8 for 'Previous results'...



When the user selects option 9 for 'Exit'...



Task Assignment

	Name	Task	%
1	**Abdelrahman Salah El-dein Abdelaziz**	• Searching for how to implement Linked List using MIPS Assembly. • Converting the C code to MIPS Assembly code. • Testing calculator operations. • Writing and coordinating the report. • Creating the PowerPoint slides.	20
2	**Farida Waheed Abd El Bary**		20
3	**Mohamed Ahmed Mohamed Hassan**		20
4	**Raneem Ahmed Refaat Ahmed**		20
5	**Razan Ahmed Fawzy**		20