Introduction:

The purpose of this assignment is to develop a MIPS Assembly program that calculates a permuted array based on an initial integer array and a provided permutation. The program prompts the user to enter the integer array and the permutation array, performs the permutation according to a specified algorithm, and displays the resulting permuted array.

Implementation:

The program begins by initializing various data structures and variables in memory. It allocates space for integer and permutation arrays, as well as some user prompts and messages. Additionally, it loads the size of the arrays and the size of each element in memory.

User Interaction (Integer Array):

The initial interaction between the program and the user is crucial as it sets the stage for the entire operation. This section provides a more detailed explanation of this interaction.

Displaying the Welcome Message:

The program commences by displaying a welcome message to the user. The welcome message is an informative and polite introduction that sets the tone for the user's experience. It serves to make the user feel comfortable and informed about the program's purpose. The specific message, "Welcome! The program will calculate an array given array and the provided permutation," gives users a clear understanding of what to expect.

Guidance for Entering the Integer Array:

Once the user has been welcomed, the program proceeds to instruct the user on how to enter the integer array. It does this by displaying a set of instructions that help the user provide the necessary input.

```
Welcome! The program will calculate an array given array and the provided permutation.

Please enter the 5 elements for the integer array

Please enter an integer value:
```

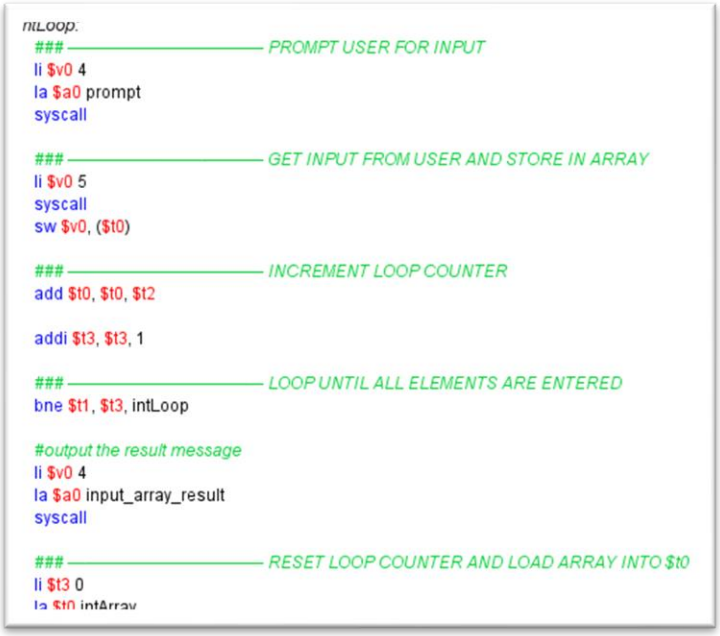
Storing Input in the Integer Array:

After each valid input, the program stores the integer value in the integer array. This process repeats until the user has entered all five required elements. The following steps are taken for each input:

1. The user provides a value, which is captured by the program.
2. The program ensures that the entered value is indeed an integer and handles any issues related to invalid input.
3. If the input is validated as an integer, it is stored in the integer array, taking the position of the array specified by the loop counter. The loop counter ensures that each input is placed in the correct position within the array.

Output for User Review:

Upon successful entry of all five elements, the program outputs a message to confirm the completion of the data entry process. The message "Your array is" is displayed, indicating that the integer array is ready for the subsequent permutation operation.



```
intLoop:
  ### ———————————— PROMPT USER FOR INPUT
  li $v0 4
  la $a0 prompt
  syscall

  ### ———————————— GET INPUT FROM USER AND STORE IN ARRAY
  li $v0 5
  syscall
  sw $v0, ($t0)

  ### ———————————— INCREMENT LOOP COUNTER
  add $t0, $t0, $t2

  addi $t3, $t3, 1

  ### ———————————— LOOP UNTIL ALL ELEMENTS ARE ENTERED
  bne $t1, $t3, intLoop

  #output the result message
  li $v0 4
  la $a0 input_array_result
  syscall

  ### ———————————— RESET LOOP COUNTER AND LOAD ARRAY INTO $t0
  li $t3 0
  la $t0 intArray
```

User Interaction (Permutation Array):

After the program prompts the user to enter the integer array successfully, it moves on to collect the permutation array. The permutation array is essential because it determines how the elements of the original integer array will be rearranged to create the permuted array.

User Instructions:

The program guides the user with a specific set of instructions to ensure the correct entry of the permutation array. Users are prompted with the message: "Please enter the 5 unique elements (0-4) for permutation."

Permutation Algorithm:

The core of the program is the permutation algorithm. It iterates through the integer array and the permutation array to rearrange the elements in the integer array based on the permutation. The algorithm uses nested loops to determine the correct positions for each element, ensuring that the resulting array corresponds to the provided permutation.

Displaying the Result:

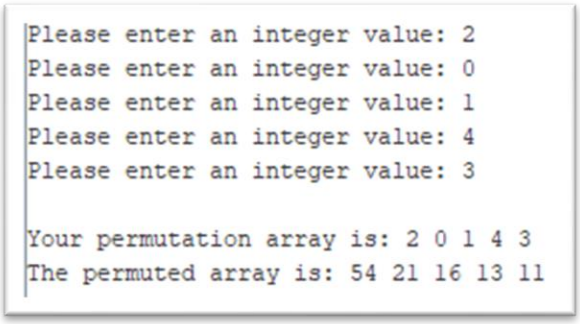
Upon completing the permutation, the program displays the permuted array.

Using System Calls:

To display the permuted array, the program utilizes system calls specific to displaying integer values. In the MIPS Assembly language, the system call with code 1 (li $v0, 1) is used to display integer values, and it takes the integer value to be displayed in $a0.

Loop for Presentation:

To maintain an organized and consistent format, the program employs a loop to iterate through the elements of the permuted array. This loop ensures that each element is correctly loaded into $a0 for display and that the necessary spaces are added between elements for readability. The loop continues until all elements of the permuted array have been displayed, guaranteeing that the user can review the entire permuted array.



Summary:

This assignment was quite engaging and fun, although it came with its share of syntax challenges. Struggling with the syntax and encountering issues such as misplaced commas, was a part of the learning curve. However, these challenges provided opportunities for growth and a deeper understanding of the Assembly language. The primary objective of this assignment was to develop a MIPS Assembly program capable of calculating a permuted array based on user-provided inputs. The process involved several key aspects:

1. Interacting with Memory: This assignment reinforced the importance of effectively managing memory, as we had to allocate and manipulate memory for integer and permutation arrays, as well as data structures like messages and prompts.
2. User Input Handling: The interaction with the user was a pivotal component of the program. It involved prompting the user for input and validating that input, which proved to be a demanding task due to syntax intricacies.
3. Algorithm Implementation: The core of the program revolved around implementing a permutation algorithm that rearranges elements in the integer array based on the provided permutation. Developing and understanding this algorithm was central to the assignment.
4. Control Flow: Leveraging conditional branching and loops was crucial to controlling the program's flow. This allowed us to efficiently navigate through the user input process, permutation calculations, and displaying the final result.

Despite the syntactical challenges and the need for occasional code refactoring, the assignment improved my MIPS skills, and also fostered a deeper understanding of memory management, which I have struggled with visualizing in the past.