Bahria University,

Karachi Campus



COURSE: CSC-221

DATA STRUCTURES AND ALGORITHM

TERM: FALL 2020, CLASS: BSE- 3 (B)

Submitted By:

USMAN AHMED KHAN 02-131192-0

MUHAMMAD ASIM 02-131192-050

MUHAMMAD HASNAIN 02-131192-017

(Name) (Reg. No.)

Project Report

Of

FINDING NUMBER IN THE ARRAY

Submitted To:

Engr. MUHAMMAD REHAN BAIG

Signed Remarks: Score:

1. **TABLE OF CONTENTS**
2. [**Abstract:** 3](#_Toc62178207)
3. [**Introduction:** 3](#_Toc62178208)
4. [**Description with Code:** 3](#_Toc62178209)
   1. [**.data:** 3](#_Toc62178210)
   2. [**.text:** 4](#_Toc62178211)

[**Conclusion:** 23](#_Toc62178223)

# **Abstract:**

**Finding Number in the Array** is such an interactive program. In which user can easily find the required number in the array if that number available in the array. We have made such a helping program using MIPS assembly language.

# **Introduction:**

**Finding Number in the Array** is simple program. In which user can easily find the required number in the array if that number available in the array. In this program we implement the array using different techniques of MIPS. In this program first our program read the values of number u want to enter then it creates an array of given number of indexes then it allows you to find the required number then it will return the index of the required number where it is present.

**Operations:**

1. Find the number in the given Array

# **Description with Code:**

## **.data:**

In .data section we have initialized the variables which we have used in our program.

**.data**

amount\_num: .asciiz "How many numbers do you have? "

num: .asciiz "Enter a number: "

nl: .asciiz "\n"

array\_contains: .asciiz "\nThe array contains the following: \n"

complete: .asciiz "\nprogram complete"

.align 2

array: .space 40

search\_num: .asciiz "\nEnter number to search for: "

f: .asciiz " was found at array["

cl\_bracket: .asciiz "]"

nf: .asciiz " was not found"

## **.text:**

.text

.globl main

main:

jal enter\_amount # Prompts the user to enter the amount of integers to be entered.

jal enter\_num # Enter the integers into the array.

jal contains # Prints the contents of the array.

jal sort # Sorts the integers in the array using a bubble sort.

jal contains # Prints the sorted contents of the array.

jal search\_for # Enter the integer you're searching for.

jal search # Searches the array using a iterative binary search.

jal search\_results # Prints the results of the search for the integer.

j done # Exit

enter\_amount:

la $a0, amount\_num # Asks the user to input the number of elements for the array.

li $v0, 4 # Prints the message.

syscall

li $v0, 5 # Reads the user's input (number of elements).

syscall

move $t0, $v0 # Stores the user's input (number of elements) in $t0.

li $t1, 0 # Offset for the array: array[x] = array[0]

li $t2, 1 # Counter for the number of integers printed: i = 1

jr $ra # Return to main.

enter\_num:

addi $t2, $t2, 1 # Increment the counter: i++

la $a0, num # Asks the user to enter an integer.

li $v0, 4 # Prints the message.

syscall

li $v0, 5 # Reads the integer that the user inputted.

syscall

sw $v0, array($t1) # Stores the inputted integers into the array.

addi $t1, $t1, 4 # Increments the address of the array: array[x+1].

ble $t2, $t0, enter\_num # if $t2 (the counter) <= $t0 (total # of integers): enter\_num

jr $ra # Return to main.

contains:

li $t1, 0 # Resets the offset for the array: array[x] = array[0]

li $t2, 1 # Resets the counter for print the integers: i = 0

la $a0, array\_contains # Prints "The array contains the following: "

li $v0, 4 # Prints the message.

syscall

print\_array:

addi $t2, $t2, 1 # Increment the counter: i++

lw $a0, array($t1) # Loads the integer into $a0: a = array[x].

li $v0, 1 # Prints the integer.

syscall

la $a0, nl # nl = newline (The next integer will be printed on a newline).

li $v0, 4 # Prints a newline.

syscall

addi $t1, $t1, 4 # Increments the offset of the array: x+1

ble $t2, $t0, print\_array # if $t2 (integer counter) <= $t0 (total # of integers): print\_array

jr $ra # Return to main.

sort:

li $t2, 0 # i = 0

outer:

addi $t2, $t2, 1 # Increments i: i++

la $a1, array # Load array address into $a1

li $t1, 0 # array[x] = array[0]

sub $t3, $t0, 1 # $t3 = n - 1 (n = total # of integers)

addi $t4, $t2, 1 # j = i + 1

ble $t2, $t3, inner # if i <= (n-1): inner

jr $ra # Return to main.

inner:

lw $t5, 0($a1) # Ex: a = 6

lw $t6, 4($a1) # Ex: b = 5

bgt $t5, $t6, swap # if a > b: swap

j continue # else: continue

swap:

sw $t6, 0($a1) # Ex: a = 5

sw $t5, 4($a1) # Ex: b = 6

continue:

addi $a1, $a1, 4 # Array[x+1]

addi $t4, $t4, 1 # j + 1

bgt $t4, $t0, outer # if (j = i + 1) > n: outer

j inner # else: inner

search\_for: # Enter the integer you're looking for.

la $a0, search\_num # Prompts the user to enter an integer.

li $v0, 4 # Prints the message.

syscall

li $v0, 5 # Reads the integer.

syscall

move $t0, $v0 # Stores the integer to search for in $t0

li $t2, 0 # First element in the array: array[0]

li $t4, 2 # Used to divide the amount of elements in the array by two.

li $t6, 4 # Used to increment the array

jr $ra # Return to main.

search:

bgt $t2, $t3, return # if $t2(first element) > $t3(last element (n-1)): The number wasn't found and return to main

middle\_num:

add $t7, $t2, $t3 # Store the sum of $t2(first element) and $t3(last element(n-1)) in $t7.

div $t7, $t4 # $s0 # Get the middle element: $t7 / $t4 = (first element + last element) / 2.

mflo $t8 # Store the middle element in $t8.

mult $t8, $t6 # Multiply the middle element by 4 to get the index for the array.

mflo $t1 # Store the product in $t1.

lw $t5, array($t1) # Load the value of array[(middle) \* 4] in $t5.

# Ex: middle = 5, x = 5 \* 4 = 20, y = array[x]

bgt $t5, $t0, lower # if middle element > x (User's desired integer): check the lower half of the array

blt $t5, $t0, upper # if middle element < x (User's desired integer): check the upper half of the array

li $s0, 1 # The integer was found (1 = true)

jr $ra # if middle element == x (User's desired integer): return to main

upper: # Upper half of the array.

add $t2, $t8, 1 # $t2 (first element) = $t8 (middle element) + 1

j search # Jump to search and check the rest of the array.

lower: # Lower half of the array.

sub $t3, $t8, 1 # $t3 (last element (n-1)) = $t8 (middle element) - 1

j search # Jump to search and check the rest of the array.

return:

li $s0, 0 # The integer wasn't found (0 = false)

jr $ra # Return to main.

search\_results:

beqz $s0, not\_found # if $s0 == 0: not\_found

found:

move $a0, $t0 # Store the user's desired integer in $a0 to print.

li $v0, 1 # Print the user's integer.

syscall

la $a0, f # Prompt the user that the integer was found.

li $v0, 4 # Print the message.

syscall

div $t1, $t6 # Divide the offset of the array by 4 to get the

mflo $a0 # location of the number in the array and store it in $a0.

li $v0, 1 # Print the location of the number.

syscall

la $a0, cl\_bracket # Print ] for the location of the number.

li $v0, 4 # Print the bracket.

syscall

jr $ra # Return to main.

not\_found:

move $a0, $t0 # Store the user's desired integer in $a0 to print.

li $v0, 1 # Print the user's integer.

syscall

la $a0, nf # Prompt the user that the integer wasn't found.

li $v0, 4 # Print the message.

syscall

jr $ra # Return to main.

done:

la $a0, complete # Tells the user that the program is complete.

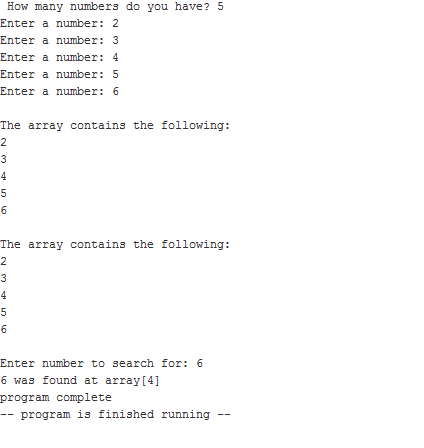
li $v0, 4 # Prints the message.

syscall

li $v0, 10 # Ends the program.

syscall

**OUTPUT:**



# **Concept used:**

Jumping statements

Branching statements

Store and Load operations

# **Conclusion:**

Finding Number in the Array is easy to use and find the numbers in the array .