****

**Green University of Bangladesh**

**Department of Computer Science and Engineering (CSE)**

**Faculty of Sciences and Engineering**

**Semester: (Fall: Year 2023), B.Sc. in CSE (Day)**

**Lab Report No:** 07

**Course Title:** Microprocessor & Microcontroller Lab

**Course Code:** CSE 304 **Section:** 213D2

**Lab Experiment Name:** Implement Macro in Assembly Language

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
| **1.** | Md Masud Rana | 213902101 |

**Lab Date :** 26-10-2023

**Submission Date :** 03-12-2023

## **Course Teacher’s Name :** Sudip Ghoshal

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature:.....................**  **Comments:.............................................. Date:..............................** |

**1. TITLE OF THE LAB REPORT EXPERIMENT**

Implement Macro in Assembly Language Programming

**2. OBJECTIVES/AIM**

* To understand the use Macro in Assembly Language
* To understand the reusability of block of code
* To understand modular program, code readability

**3. PROCEDURE**

**Problem: Sum of even and odd number from natural number series 1+2+3…+n using macro**

**Step 1:** start

**Step 2:** initialize data segment arr of 10 size, i & temp to 0 and oddSum, evenSum to empty

**Step 3:** Include data segment into main procedure into ‘DS’ register

**Step 4:** take array size from user and copy to ‘bl’ and set ‘cl’ as ‘bl’ like ‘cx’ to ‘bl’

**Step 5:** set pointer or index register to the ‘arr’ using ‘si’.

**Step 6:** continue loop. That will continue cx=cl to 0. **‘taken\_input’**

**Step 7:** Take input and store arry [si], al.

Increment ‘si’ pointer by 1 ‘inc si’. Print a space between each input

Continue **‘loop taken\_input’** and repeat **step-7**

**Step 8:** macro ‘oddEven’ call by passing arrary ‘arr’ and address pointer ‘si’ to it

**Step 9:** inside macro ‘**oddEven’** set ‘dl’ as 2 to divide each element of the array to check even or odd number

**Step 10:** take ‘arr\_traverse’ level to jumping conditionally again and again

Initially set ‘ah’ as 0.

Take element from array using [si] into ‘al’.copy ‘al’ to ‘temp’ variable for future

Now divide ‘al’ by ‘dl’ note: ah=remainder and al= quotient

Compare ‘ah’ with 0 or not. If 0 this is even number else odd number

Jump ‘JE’ even\_sum :**jump equal to 0**

Jump ‘JNE’ odd\_sum **jump not equal to 0**

After jumping we need to get back into ‘arr\_traverse’ loop. So take a level ‘back’

After backing, increment variable ‘i’ by 1 and ‘inc si’

Compare ‘i’ with ‘bl’. If equal called ‘print\_result’ level

If less then again called ‘arr\_traverse’ level

**Step 11:** continue **step-10**

**Step 12:** print even and odd summation into console.

**Step 13:** end

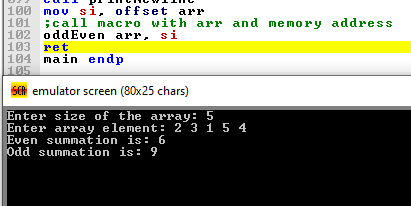
**4. IMPLEMENTATION**

**Problem: Sum of even and odd number from natural number series 1+2+3…+n using macro**

2

1

|  |  |
| --- | --- |
| ;using marco odd even summation in an array  ;print odd and even number summation  oddEven macro arr, si    ;point array usnig source index register  mov dl, 2    ;traverse & calculate odd,even sum  arr\_traverse:  mov ah, 0  mov al, [si]  ;temporary store current element  mov temp, al  ;after divide by 2 ah=remainder al=quotien  div dl  cmp ah, 0  JE even\_sum  JNE odd\_sum  back:  inc i  inc si  cmp i, bl  JL arr\_traverse  JE print\_result    ;even number calculation  even\_sum:  mov al, temp  add al, evenSum  sub al, 48  mov evenSum, al  jmp back    ;odd number calculation  odd\_sum:  mov al, temp  add al, oddSum  sub al, 48  mov oddSum, al  jmp back    ;print the result  print\_result:  print 'Even summation is: '  mov dl, evenSum  add dl, 48  call printNumber    call printNewline  print 'Odd summation is: '  mov dl, oddSum  add dl, 48  call printNumber    endm  ;emu8086 library  include 'emu8086.inc'  .stack 100h  .model small  .data  arr db 10 dup(?)  i db 0  temp db 0  oddSum db ?  evenSum db ?  newline db 10, 13, '$' | .code  main proc  ;import data segment  mov ax, @data  mov ds, ax    mov si, offset arr  print 'Enter size of the array: '  call scan  mov bl, al ;store input size into bl register  sub bl, 48    mov cx, 0  mov cl, bl ;cx=user input size    call printNewline  print 'Enter array element: '  take\_input:  call scan  mov [si], al  inc si  ;pirnt a space between each input  mov dl, 32  call printNumber    loop take\_input    call printNewline  mov si, offset arr  ;call macro with arr and memory address  oddEven arr, si  ret  main endp      ;custom procedure  printNewline proc  mov ah, 9  lea dx, newline  int 21h  RET  printNewline endp  ;printNumber  printNumber proc  mov ah, 2  int 21h  RET  printNumber endp  ;scan number  scan proc  mov ah, 1  int 21h  RET  scan endp  end main |

**Output:**

*Figure-1: Summation of even and odd number from an array of size 5*

**5. TEST RESULT / OUTPUT**

**For the first test:** showed in figure-1

- Size of the array: 5

- Array elements: 2 3 1 5 4

**Output:**

Even summation is: 6

Odd summation is: 9

**For the second test**: showed in figure-2

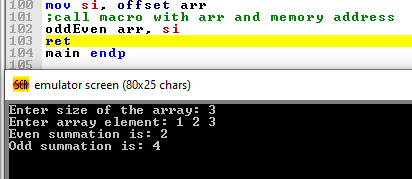
- Size of the array: 3

- Array elements: 1 2 3

**Output:**

Even summation is: 2

Odd summation is: 4

****

*Figure-2: Summation of even and odd number from an array of size 3*

**6. ANALYSIS AND DISCUSSION**

The program utilizes macros effectively for modularizing the code. The **oddEven** macro encapsulates the logic for traversing the array, identifying even and odd numbers, and calculating their summations. This enhances code **readability**, **maintainability**, and **reusability**.

The program tested two input cases with different array size which are showed in figure-1 and figure-2. Where we have seen that the program gives proper answer.

Registers such as si, cx, dl, and al are used strategically for efficient storage and manipulation of data.

The program prompts the user to input the size of the array and its elements.

Conditional jumps (JE and JNE) are used to control the flow of execution based on whether the current array element is even or odd.

The program uses custom procedures (printNewline, printNumber) to format and display output.

**7. SUMMARY**

The assembly language program successfully calculates the summation of even and odd numbers in an array using macros. The program takes user input for the size of the array and its elements. It then traverses the array, identifies even and odd numbers, and calculates their summation using the defined macros.

The macro `**oddEven**` efficiently handles the traversal and summation process, and the program prints the results for even and odd summations accurately. The program demonstrates the use of registers and conditional jumps to achieve the desired functionality.

The program is structured with modular code, utilizing macros for the repetitive tasks, making it more readable and maintainable. Overall, the assembly program fulfills the requirements of the lab problem, providing a clear and effective solution for calculating even and odd summations in an array using macro