Name:	Viraj Oza
Roll No:	69
Class/Sem:	SE/IV
Experiment No.:	8
Title:	Mixed language program to add two numbers
Date of Performance:	
Date of Submission:	
Marks:	
Sign of Faculty:	

Aim: Mixed language program for adding two numbers.

Theory:

C generates an object code that is extremely fast and compact but it is not as fast as the object code generated by a good programmer using assembly language. The time needed to write a program in assembly language is much more than the time taken in higher level languages like C.

However, there are special cases wher a function is coded in assembly language to reduce the execution time.

Eg: The floating point math package must be loaded assembly language as it is used frequently and its execution speed will have great effect on the overall speed of the program that uses it.

There are also situations in which special hardware devices need exact timing and it is must to write a program in assembly language to meet this strict timing requirement. Certain instructions cannot be executed by a C program

Eg: There is no built in bit wise rotate operation in C. To efficiently perform this it is necessary to use assembly language routine.

Inspite of C being very powerful , routines must be written in assembly language to:

1.Increase the speed and efficiency of the routine

2.Perform machine specific function not available in Microsoft C or Turbo C.

3. Use third party routines

Combining C and assembly:

Built-In-Inline assembles is used to include assembly language routines in C program without any need for a specific assembler.

Such assembly language routines are called in-line assembly.

They are compiled right along with C routines rather than being assembled separately and then linked together using linker modules provided by the C compiler.

Turbo C has inline assembles.

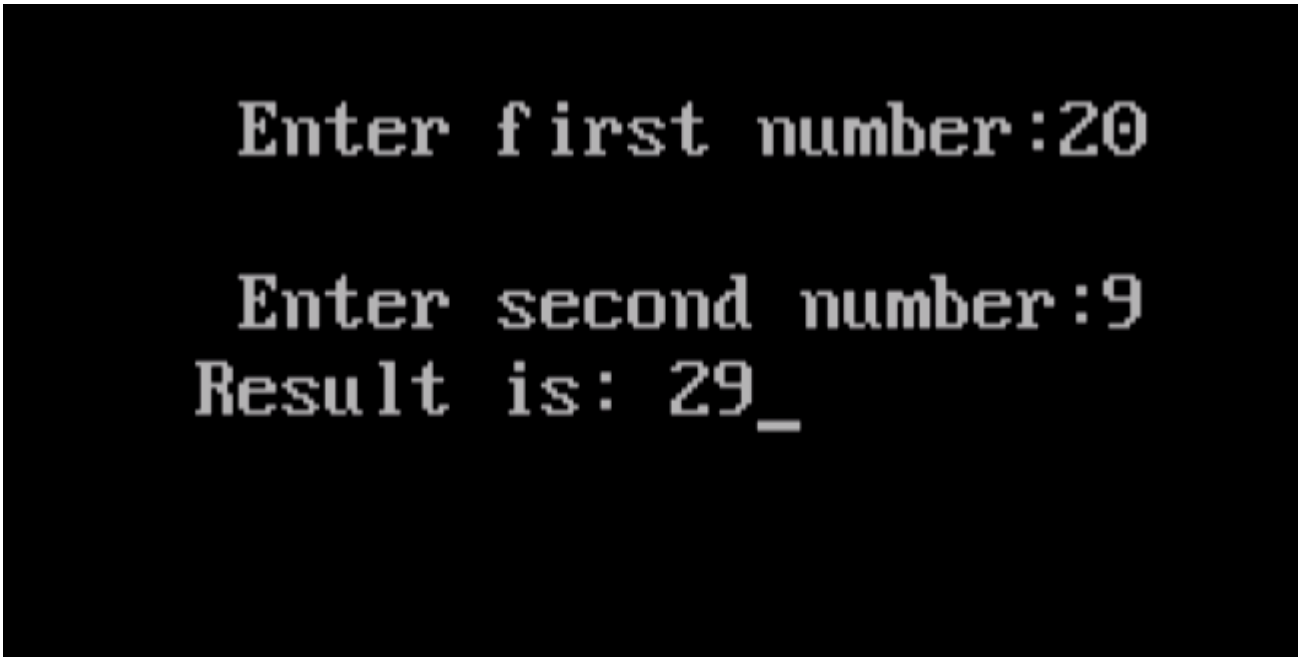
In mixed language program, prefix the keyword asm for a function and write Assembly instruction in the curly braces in a C program.

Program:

```
#include<stdio.h>
#include<conio.h>
void main(){
        int a,b,c;
        clrscr();

        printf("\n Enter first number:");
        scanf("%d",&a);
        printf("\n Enter second number:");
        scanf("%d",&b);
        asm{
            mov ax,a
            mov bx,b
            add ax,bx
            mov c,ax
           }
       printf("Result is: %d",c);
       getch();
       }
```

Output:



Conclusion: In this mixed-language programming exercise, we successfully created a program to add two numbers using a combination of assembly language and C. The program leverages the strengths of each language to achieve efficient and robust functionality. The C language portion of the program serves as the main framework, providing high-level control flow and input/output operations. It prompts the user to input two numbers, reads the inputs, and calls an assembly language function to perform the addition.

1. Explain any 2 branch instructions.

Ans. JMP (Jump):

a. The JMP instruction is used to transfer control unconditionally to another location in the program.
b. Syntax: `JMP destination`
c. `destination` can be specified as a label, memory location, or a register.
d. When the JMP instruction is executed, the instruction pointer (EIP) is set to the address specified by the destination operand, causing the program to continue execution from that address.
e. JMP is often used for implementing loops, implementing switch-case constructs, and for performing unconditional jumps in program flow.

2. JZ/JE (Jump if Zero/Jump if Equal):
   a. The JZ/JE instruction is used to perform a conditional jump based on the state of the Zero Flag (ZF).
   b. Syntax: `JZ/JE destination`
   c. `destination` specifies the address to jump to if the Zero Flag is set.
   d. If the Zero Flag is set (indicating that the result of the previous operation was zero), the program jumps to the specified destination.
   e. JZ/JE is commonly used in conjunction with comparison or arithmetic instructions to implement conditional branching in the program flow.
   f. It is often used in if-else constructs, switch-case statements, and other decision-making logic in assembly language programs.

2. Explain the syntax of loop.

Ans. In assembly language, the LOOP instruction is used to implement loops, allowing for repetitive execution of a block of code. Here's a breakdown of the syntax:

LABEL:

Loop body

Instructions to be executed repeatedly

```
LOOP LABEL ; Decrement ECX and jump to LABEL if ECX ≠ 0
```

`LABEL`: This is a symbolic name assigned to the start of the loop. It serves as the target for the LOOP instruction to jump back to at the end of each iteration.

`LOOP LABEL`: This instruction decrements the ECX register (CX register in 16-bit mode) and jumps to the specified label if the result is not zero. It effectively implements a loop counter, and the loop continues until ECX reaches zero.

`; Loop body`: This is the section of code within the loop that will be executed repeatedly until the loop terminates. It typically contains the instructions that perform the desired operations or computations.

`ECX` (or `CX`): This is the loop counter register. It holds the number of iterations remaining in the loop. The LOOP instruction decrements the value of ECX by 1 in each iteration until it reaches zero.