Name:	Rahul Singh
Roll No:	56
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.

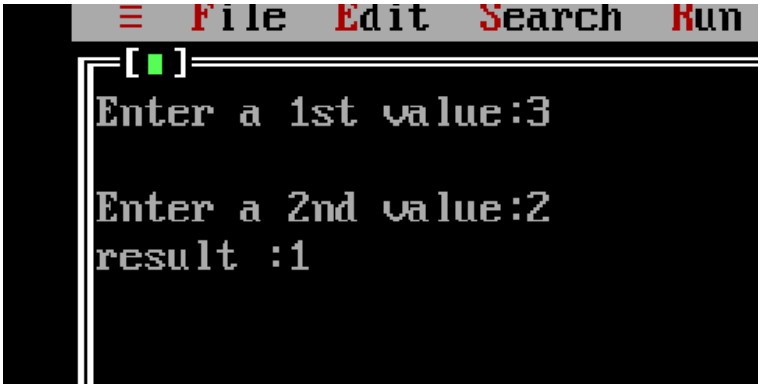
In mixed language program, prefix the keyword asm for a function and write Assembly instruction in the curly braces in a C program

Assembly Code:

```
#include  <stdio.h>

#include <conio.h>

int main(){

int a,b,c;

clrscr();

printf("Enter a 1st value:");

scanf("%d",&a);
```

```
printf("\nEnter a 2nd value:");

scanf("%d",&b);

asm{ mov

ax,a mov

bx,b SUB

ax,bx

mov c,ax

}

printf("result :%d",c);

return 0;

}
```

Output:



Conclusion:

1. Explain any 2 branch instructions.

Ans. Branch instructions are crucial in assembly language programming as they enable conditional and unconditional branching, altering the program flow based on certain conditions or criteria. Here are explanations of two commonly used branch instructions:

a) JMP (Jump):

   a. JMP is an unconditional branch instruction that directs the CPU to jump to a specified memory location without any condition evaluation.

   b. It allows altering the program flow unconditionally, transferring control to the target instruction specified by the operand.

c. Syntax: JMP destination

d. Example: JMP label directs the CPU to jump to the instruction labeled "label" without any condition evaluation.

b) JE (Jump if Equal):

a. JE is a conditional branch instruction that performs a jump to a specified memory location if the Zero Flag (ZF) is set, indicating that the previous comparison resulted in equality.

b. It's commonly used in conjunction with comparison operations to implement conditional branches based on equality.

c. Syntax: JE destination

d. Example: CMP AX, BX followed by JE label would jump to the instruction labeled "label" if the contents of AX and BX registers are equal.

These instructions are fundamental for implementing decision-making and loop constructs in assembly language programs, enabling efficient control flow management.

2. Explain the syntax of loop.

Ans. The LOOP instruction is a handy construct in assembly language for implementing loops. Here's the syntax of the LOOP instruction:

a) Syntax:
LOOP destination

b) Explanation:

- LOOP: This is the mnemonic for the loop instruction. It's followed by the destination, which is the label or memory address where the CPU should jump if the loop counter (CX register) is not zero.
- Destination: This specifies the target location in the code where the CPU should jump if the loop counter (CX register) is not zero. It can be specified as a label or a memory address.

c) Usage:

- The LOOP instruction decrements the loop counter (CX register) by one and checks if it's zero.
- If the loop counter is not zero after decrementing, the CPU jumps to the destination specified by the label or memory address.
- This process repeats until the loop counter becomes zero.

Example:

```assembly
MOV CX, 10          ; Initialize loop counter to 10
L1:
; Loop body instructions
DEC CX              ; Decrement loop counter
LOOP L1             ; Jump back to L1 if CX is not zero
```

In this example, the LOOP instruction jumps back to label L1 if the loop counter CX is not zero, effectively creating a loop that iterates 10 times.