Name: **Tushar Nankani** Roll No: **1902112** Batch: **C23**

**Microprocessor: Experiment 1**

**Aim:** Write an assembly language program for 8-bit addition, subtraction, multiplication and division.

Theory:

Assembly Language:-

In [computer programming,](https://en.wikipedia.org/wiki/Computer_programming) assembly language (or assembler language), often abbreviated asm, is any [low-level](https://en.wikipedia.org/wiki/Low-level_programming_language) [programming language](https://en.wikipedia.org/wiki/Low-level_programming_language) in which there is a very strong correspondence between the instructions in the language and the [architecture's](https://en.wikipedia.org/wiki/Computer_architecture) [machine code](https://en.wikipedia.org/wiki/Machine_code) [instructions.](https://en.wikipedia.org/wiki/Instruction_set_architecture) Because assembly depends on the machine code instructions, every assembly language is designed for exactly one specific computer architecture. Assembly language may also be called symbolic machine code. Assembly code is converted into executable machine code by a [utility](https://en.wikipedia.org/wiki/Utility_software)

[program](https://en.wikipedia.org/wiki/Utility_software) referred to as an [assembler.](https://en.wikipedia.org/wiki/Assembly_language#Assembler)

ASSEMBLER DIRECTIVE:-

An assembler is a program used to convert an assembly language program into the equivalent machine code modules. The assembler decides the address of each label and substitutes thevalues for each of the constants and variables. It then forms the machine codefor mnemonics and data in assembly language program. Assembler directives help the assembler to correctly understand assembly language programs to prepare the codes. Commonly used assembler directives are DB, DD, DW, DUP, ASSUME, BYTE, SEGMENT, MACRO, PROC, OFFSET, NEAR, FAR, EQU, STRUC, PTR, END, ENDM, ENDP etc. Some

directives generate and store information in the memory, while others do not. 1)DB :-Define byte directive stores bytes of data in memory.

2)BYTE PTR :-This directive indicates the size of data referenced by pointer. 3)SEGMENT :-This directive is to indicate the start of the segment.

1. DUP (Duplicate) :-The DUP directive reserves memory locations given by the number preceding it, but stores no specific values in any of these locations.
2. ASSUME : -The ASSUME statement is only used with full segment definitions. This statement tells the assembler what names have been chosen for the code, data, extra and stack segments.
3. EQU : -The equate directive equates a numeric ASCII or label to another label. 7)ORG : -The ORG (origin) statement changes the starting offset address in a segment.

8)PROC and ENDP : -The PROC and ENDP directives indicate start and end of a procedure (Sub routine). Both the PROC and ENDP directives require a label to indicate the name of the procedure. The PROC directive, must also be followed with the NEAR or FAR. A NEAR procedure is one that resides in the same code segmentas the program. A FAR procedure may reside at any location in the memory system.

**Function 01h- Character input with echo**

|  |  |
| --- | --- |
| **Action:** | Reads a character from the standard input device and echoes it to the standard output device.  If no character is ready it waits until one is available. I/O can be re-directed, but prevents detection of OEF. |
| **On entry:** | AH = 01h |
| **Returns:** | AL = 8 bit data input |

**Function 02h - Character output**

|  |  |
| --- | --- |
| **Action:** | Outputs a character to the standard output device. I/O can be re-directed, but prevents detection of 'disc full'. |
| **On entry:** | AH = 02h  DL = 8 bit data (usually ASCII character) |
| **Returns:** | Nothing |

**Function 09h- Output character string**

|  |  |
| --- | --- |
| **Action:** | Writes a string to the display. |
| **On entry:** | AH = 09h  DS:DX = segment:offset of string |
| **Returns:** | Nothing |

**Function 2lh - Random read**

|  |  |
| --- | --- |
| **Action:** | Reads a selected record from an opened file. |
| **On entry:** | AH = 21h  DS:DX = Segment:offset of previously opened FCB |
| **Returns:** | AL = 0 if successful  AL = 1 if end of file reached AL = 2 if segment wrap occurs  AL = 3 if partial record read at end of file |

**ROL**: - Rotate operand1 left. The number of rotates is set by operand2.

Algorithm: shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.

Example: MOV AL, 1Ch ;

AL = 00011100b ROL AL, 1 ; AL = 00111000b, CF=0. RET

**ROR**: - Rotate operand1 right. The number of rotates is set by operand2.

Algorithm: shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.

Example: MOV AL, 1Ch ;

AL = 00011100b ROR AL, 1 ; AL = 00001110b, CF=0. RET

**SUB**: - Subtract.

Algorithm: operand1 = operand1 - operand2 Example: MOV AL, 5

SUB AL, 1 ; AL = 4 RET

**ADD**: - Add.

Algorithm: operand1 = operand1 + operand2 Example: MOV AL, 5 ;

AL = 5 ADD AL, -3 ; AL = 2 RET

**MOV**: - Copy operand2 to operand1. The MOV instruction cannot: -

z set the value of the CS and IP registers. Z

copy value of one segment register to another segment register (should copy to general register first). copy immediate value to segment register (should copy to general register first).

Algorithm: operand1 = operand2

Example: ORG 100h MOV AX, 0B800h; set AX = B800h (VGA memory). MOV DS, AX; copy value of AX to DS.

MOV CL, 'A'; CL = 41h (ASCII code).

MOV CH, 01011111b; CL = color attribute.

MOV BX, 15Eh; BX = position on screen.

MOV [BX], CX; w.[0B800h:015Eh] = CX.

RET; returns to operating system.

**JC**: - Short Jump if Carry flag is set to 1. Algorithm: if CF = 1 then jump Example: include 'emu8086.inc' ORG 100h

MOV AL, 255

ADD AL, 1

JC label1

PRINT 'no carry.' JMP exit label1:

PRINT 'has carry.' exit:

RET

# Algorithm for 8 bit addition:

1. Start
2. Initialize data segmentthrough AX register in the DS register.
3. Display the message as “Enter the first number”
4. Read first digit in AL register through keyboard(e.g. AL=31h)
5. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=01h
6. Move contents of AL register to a BL. (BLAL so BL=01h)7)Rotate the contents of BL register by 4 positions at left side. (BL=10h) 8)Read a second digit in AL register through keyboard AL=35h
7. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=05h
8. Add the contents of BL and AL store the result in BL (BLBL+AL so BL=15h)
9. Display the message as “Enter the second number”
10. Read first digit in AL register through keyboardAL=32h
11. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=02h
12. Move contents of AL register to a CL. (CLALso CL=02h)
13. Rotate the contents of CL register by 4 positions at left side. (CL=20h) 16)Read a second digit in AL register through keyboard (AL=33h)
14. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimalnumber. AL=03h
15. Add the contents ofCL and AL store the result in CL (CLCL+ALso CL=23h) (Now both numbers are accepted as 15h and 23h)
16. Add the contents of BL and CL and result gets stored in BL (E.g ADD BL,CLso BL=38h) 20)Preserve the result of addition in some temporary variable say temp from BL.
17. Mask the first nibble by AND operation with number F0h (AND BL,F0h so BL=30h)
18. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BL=33h) 23)Move the contents ofBL to DL and display it on the screen
19. Move result from temporary variable to BL again (So BL=38h)
20. Mask the second nibble by AND operation with number 0Fh (AND BL,0Fh so BL=08h)
21. Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BL=38h) 27)Move the contents of BL to DL and display it on the screen

28)Stop

Algorithm for Input procedure: (To accept input from 0 to F)

1)Compare the contents ofAL with 41h.

2)Jump to step no 4 if carry flag is set(digit is in the range of 0 to 9 so add only 30h) 3)Sub 07hto AL register(If digit is in the range from A to F then add 30h and 7h both) 4)Sub 30hto AL register

5)Return

Algorithm for Output procedure:

Compare the contents of BL with 0Ah

Jump to step no 4 if carry flag is set(digit is in the range of 0 to9 so add only 30h) 3)Add 07h to BL register(If digit is in the range from A to F then add 30h and 7h both) 4)Add 30hto BL register

5)Return

Algorithm for 8 bit subtraction:

Start

Initialize data segment through AX register in the DS register.

Display the message as “Enter the first number”

Read first digit in AL register through keyboard(e.g. AL=31h)

Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=01h

Move contents of AL register to a BL. (BLAL so BL=01h)

Rotate the contents of BL register by 4 positions at left side. (BL=10h) 8.Read a second digit in AL register through keyboard AL=35h

Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=05h

Add the contents of BL and AL store the result in BL (BLBL+AL so BL=15h)

Display the message as “Enter the second number”

Read first digit in AL register through keyboardAL=32h

Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=02h

Move contents of AL register to a CL. (CLAL so CL=02h)

Rotate the contents of CL register by 4 positions at left side. (CL=20h) 16.Read a second digit in AL registerthrough keyboard (AL=33h)

Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=03h

Add the contents of CL and AL store the result in CL (CLCL+AL so CL=23h) (Now both numbers are accepted as 15h and 23h)

Subtractthe contents of CL fromBL and result gets stored in BL (E.g SUBBL,CLso BL=F2h) 20.Preserve the result in some temporary variable say temp from BL.

Mask the first nibble by AND operation with number F0h (AND BL,F0h so BL=30h)

Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BL=33h) 23.Move the contents of BL to DL and display it on the screen

Move result from temporary variable to BL again (So BL=38h)

Mask the second nibble by AND operation with number 0Fh (AND BL,0Fh so BL=08h)

Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BL=38h) 27.Move the contents of BL to DL and display it on the screen

28.Stop

Algorithm for Input procedure:(To accept input from 0 to f) 1.Compare the contents of AL with 41h

2.Jump to step no 4 if carry flag is set 3.Sub 07h to AL register

4.Sub 30h to AL register 5.Return

Algorithm for Output procedure: 1.Compare the contents of BL with 0Ah

2.Jump to step no 4 if carry flag is set

3.Add 07h to BL register

4.Add 30h to BL register   
5)Return

# 8 BIT Addition:-

CODE:-

Data segment

msg db 0dh,0ah,"Enter first number: $" msg1 db 0dh,0ah,"Enter second number: $" result db 0dh,0ah,"The Result is: $"

Data ends Code segment

assume CS:Code,DS:Data start:

mov ax,Data mov DS,ax

mov dx,offset msg mov ah,09h

int 21h mov ah,01h int 21h

sub al,30h mov bl,al

rol bl,4 mov ah,01h int 21h

sub al,30h add bl,al

mov dx,offset msg1 mov ah,09h

int 21h mov ah,01h int 21h

sub al,30h mov cl,al rol cl,4

mov ah,01h int 21h

sub al,30h add cl,al add bl,cl

mov dx,offset result mov ah,09h

int 21h mov cl,bl and bl,0f0h ror bl,4

call AsciiConv

mov dl,bl mov ah,02h int 21h mov bl,cl and bl,0fh

call AsciiConv

mov dl,bl mov ah,02h int 21h mov ah,4ch int 21h

AsciiConv

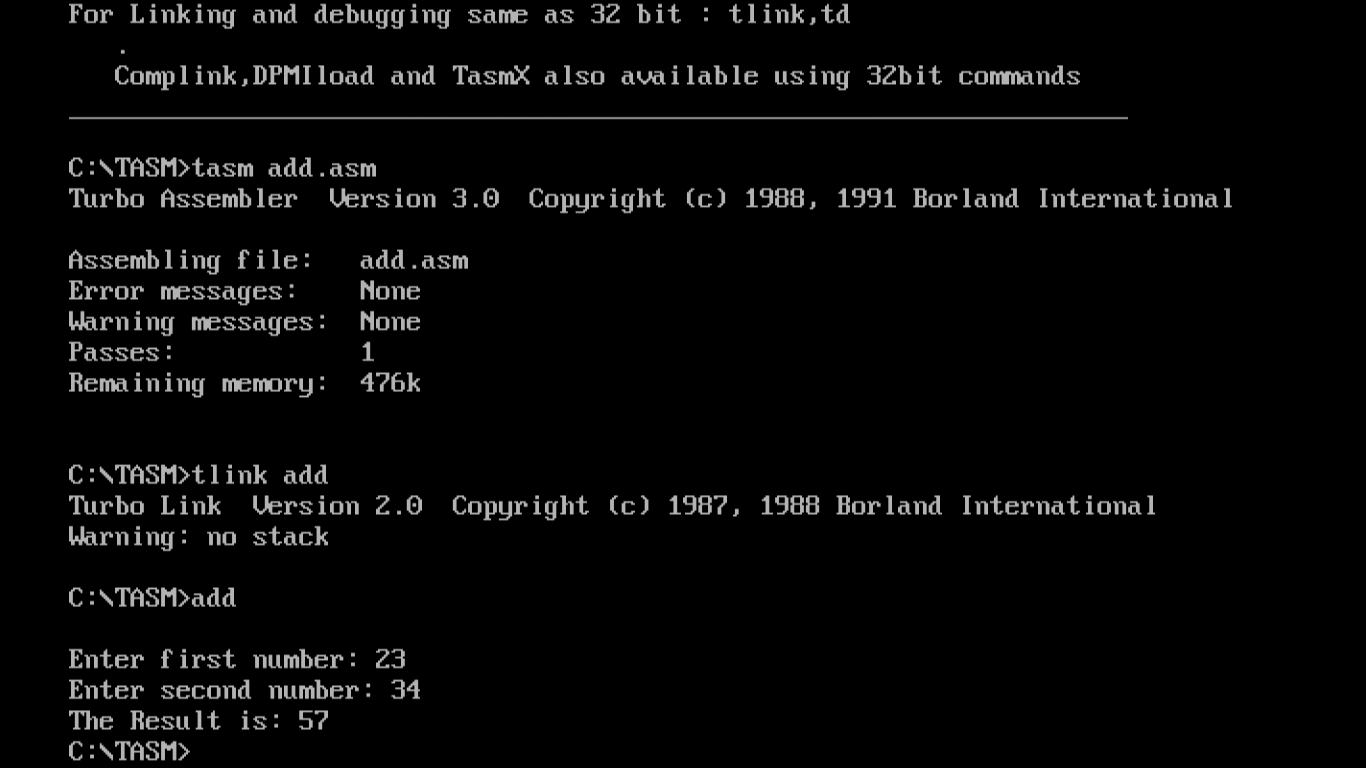
proc cmp bl,0ah jc skip

add bl,07h

skip: add bl,30h ret

endp Code ends end start

**OUTPUT:-**



# 8 BIT SUBTRACTION: -

CODE:-  
  
Data segment

msg db 0dh,0ah,"Enter first number: $"

msg1 db 0dh,0ah,"Enter second number: $"

result db 0dh,0ah,"The Result is: $"

Data ends

Code segment

assume CS:Code,DS:Data

start:

mov ax,Data

mov DS,ax

mov dx,offset msg

mov ah,09h

int 21h

mov ah,01h

int 21h

sub al,30h

mov bl,al

rol bl,4

mov ah,01h

int 21h

sub al,30h

add bl,al

mov dx,offset msg1

mov ah,09h

int 21h

mov ah,01h

int 21h

sub al,30h

mov cl,al

rol cl,4

mov ah,01h

int 21h

sub al,30h

add cl,al

sub bl,cl

mov dx,offset result

mov ah,09h

int 21h

mov cl,bl

and bl,0f0h

ror bl,4

call AsciiConv

mov dl,bl

mov ah,02h

int 21h

mov bl,cl

and bl,0fh

call AsciiConv

mov dl,bl

mov ah,02h

int 21h

mov ah,4ch

int 21h

AsciiConv proc

cmp bl,0ah

jc skip

add bl,07h

skip: add bl,30h

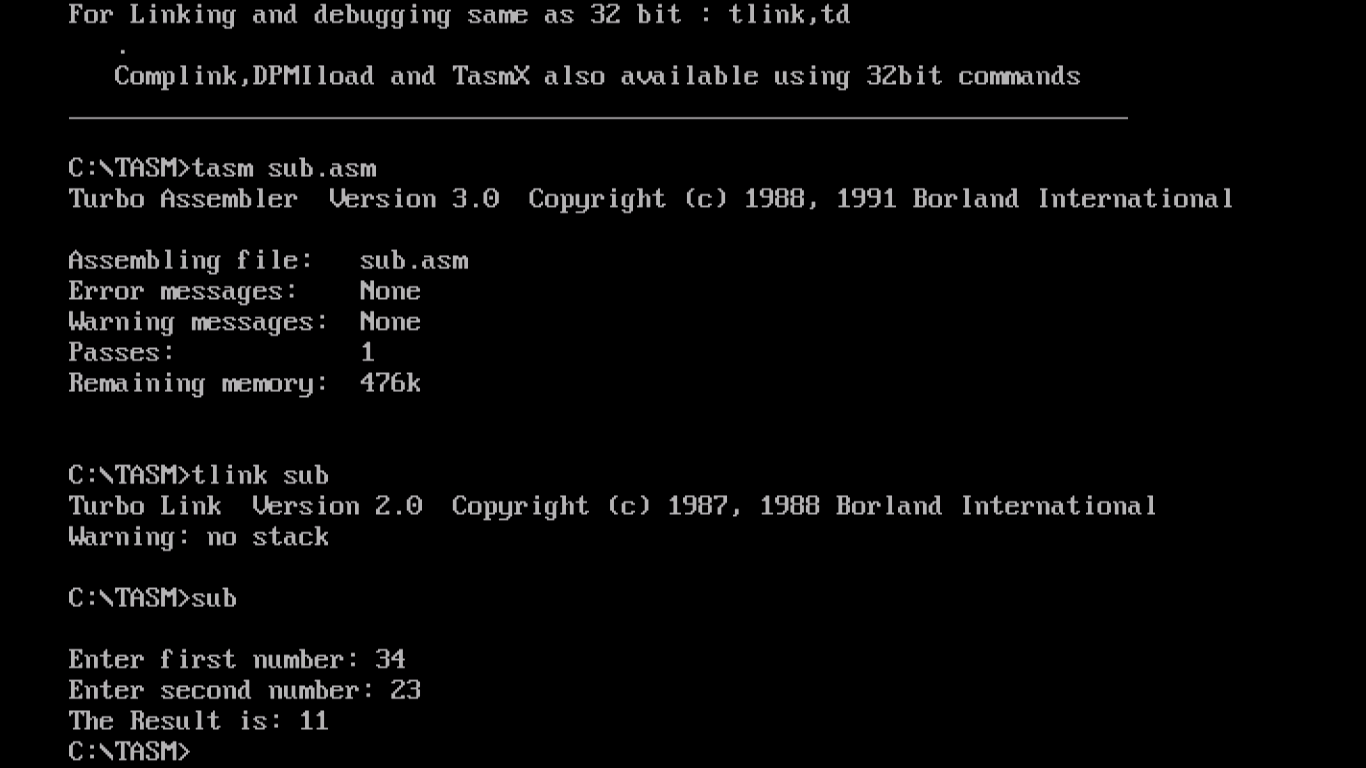
ret

endp

Code ends

end start

**OUTPUT: -**



# 8 BIT MULTIPILICATION: -

CODE:-

data segment

msg1 db 0dh,0ah,"PROGRAM ON 8 BIT Multiplication$";

msg2 db 0dh,0ah,"Enter First Number: $";

msg3 db 0dh,0ah,"Enter Second number: $";

msg4 db 0dh,0ah,"Product :$";

data ends

code segment

assume cs:code,ds:data

start:

mov ax,data

mov ds,ax

mov dx,offset msg1

mov ah,09h

int 21h

mov dx,offset msg2

mov ah,09h

int 21h

call input

rol al,04

mov bl,al

call input

add bl,al

mov dx,offset msg3

mov ah,09h

int 21h

call input

rol al,04

mov cl,al

call input

add cl,al

mov ah,00H

mov al,bl

mul cl

MOV BX,AX

MOV CX,BX

mov dx,offset msg4

mov ah,09h

int 21h

mov aX,bX

and aX,0F000h

ror ax,12

mov dl,al

call output

mov ax,bx

and ax,0F00h

ror ax,08

mov dl,al

call output

mov ax,bx

and ax,00F0h

ror ax,04

mov dl,al

call output

mov ax,bx

and ax,000Fh

mov dl,al

call output

mov ah,4ch

int 21h

input proc

mov ah,01h

int 21h

cmp al,41h

jc lb

sub al,07h

lb:sub al,30h

mov ah,00h

ret

endp

output proc

cmp dl,0Ah

jc lb1

add dl,07h

lb1:add dl,30h

mov ah,02h

int 21h

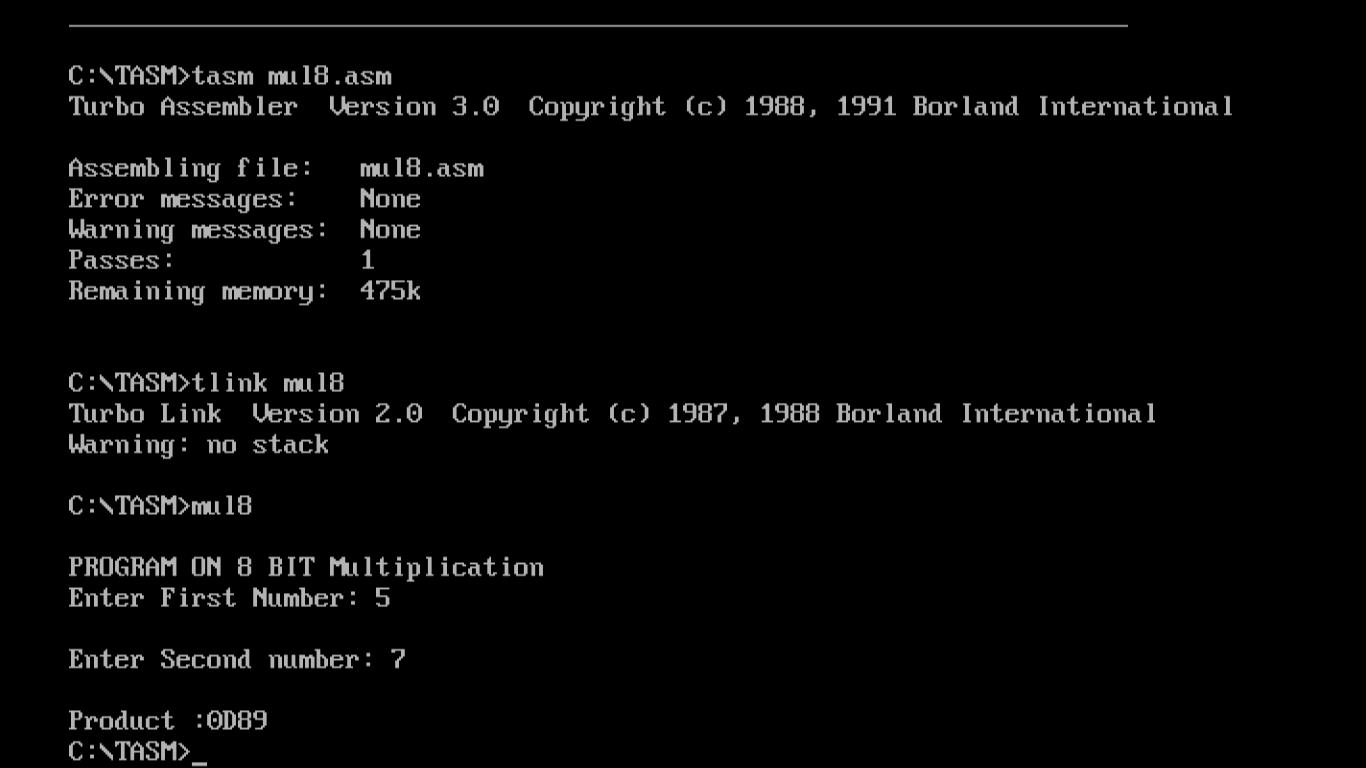
ret

endp

code ends

end start

**OUTPUT: -**



**8 BIT DIVISION: -**

**CODE:-**

data segment

msg1 db 0dh,0ah,"PROGRAM ON 8 BIT Division$";

msg2 db 0dh,0ah,"Enter First Number: $";

msg3 db 0dh,0ah,"Enter Second number: $";

msg4 db 0dh,0ah,"Quotient :$";

msg5 db 0dh,0ah,"Remainder :$";

data ends

code segment

assume cs:code,ds:data

start:

mov ax,data

mov ds,ax

mov dx,offset msg1

mov ah,09h

int 21h

mov dx,offset msg2

mov ah,09h

int 21h

call input

rol al,04

mov bl,al

call input

add bl,al

mov dx,offset msg3

mov ah,09h

int 21h

call input

rol al,04

mov cl,al

call input

add cl,al

mov ah,00H

mov al,bl

DIV cl

MOV BX,AX

MOV CX,BX

mov dx,offset msg4

mov ah,09h

int 21h

mov aX,bX

and al,0F0h

ror al,04

mov dl,al

call output

mov ax,bx

and al,0Fh

mov dl,al

call output

mov dx,offset msg5

mov ah,09h

int 21h

mov ax,bx

and ah,0F0h

ror ah,04

mov dl,ah

call output

mov ax,bx

and ah,0Fh

mov dl,ah

call output

mov ah,4ch

int 21h

input proc

mov ah,01h

int 21h

cmp al,41h

jc lb

sub al,07h

lb:sub al,30h

mov ah,00h

ret

endp

output proc

cmp dl,0Ah

jc lb1

add dl,07h

lb1:add dl,30h

mov ah,02h

int 21h

ret

endp

code ends

end start

**OUTPUT: -**

