**Practical - 17**

**Aim:** Write an assembly program to create two data series as mentioned below. Calculate the sum of all elements in each of them and save them in respective variables. Also display both answers on console.

1. Data series of byte type data
2. Data series of word type data

**Description of instructions used:**

**MARCO:** A **Macro** is a set of instructions grouped under a single unit. It is another method for implementing modular programming in the **8086** microprocessors (The first one was using Procedures)

**DB** **(DEFINE BYTE):** The **DB** directive is used to declare a byte type variable, or a set aside one or more storage locations of type byte in memory.

**LEA (Load Effective Address):** LEA and MOV both are same but in that there are quite difference between both of them.

* LEA means Load Effective Address
* MOV means Load Value

In short, LEA loads a pointer to the item you're addressing whereas MOV loads the actual value at that address. The purpose of LEA is to allow one to perform a non-trivial address calculation and store the result.

**LEA AX, [BP+SI+5]; Compute address of value**

**MOV AX, [BP+SI+5]; Load value at that address**

**INT:** Used to interrupt the program during execution and calling service specified.

**CMP:** The CMP instruction compares two operands. It is generally used in conditional execution. This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not. It does not disturb the destination or source operands.

**JL:** The **JL instruction** is a conditional jump that follows a test. It performs a signed comparison jump after a **CMP** if the destination operand is less than the source operand.

**JG:** Jumps to the destination label mentioned in the instruction if the result of previous instruction (generally compare) causes ZF to have value equal to 0 and CF and OF to have same values, else no action is taken.

**JMP:** the JMP instruction performs an unconditional jump. Such an instruction transfers the flow of execution by changing the instruction pointer register.

**INC:** Used to increment the provided byte/word by 1.

**LOOP:** Used to loop a group of instructions until the condition satisfies, i.e., CX = 0

**Code:**

.MODEL SMALL

.DATA

SERIES1 DB 1H, 2H,3H,4H,5H

SERIES2 DW 0123H, 0456H, 0122H, 1000H, 3000H

MSG1 DB "SUM OF SERIES1: $"

MSG2 DB "SUM OF SERIES2: $"

SUM1 DW 01H DUP(0)

SUM2 DW 01H DUP(0)

.CODE

MOV AX, @DATA

MOV DS, AX

MOV CX, 05H

XOR AX, AX

MOV SI, OFFSET SERIES1

MOV DI, OFFSET SERIES2

ADDITION:

ADD AL, [SI]

ADD BX, [DI]

INC SI

ADD DI, 02H

LOOP ADDITION

MOV SUM1, AX

MOV SUM2, BX

MOV DX, OFFSET MSG1

MOV AH, 09H

INT 21H

MOV AX, SUM1

CALL PRINT

MOV AH , 2

MOV DL , 0DH

INT 21H

MOV DL , 0AH

INT 21H

MOV DX, OFFSET MSG2

MOV AH, 09H

INT 21H

MOV AX, SUM2

CALL PRINT

MOV AH,4CH

INT 21H

PRINT PROC

;initilize count

mov cx,0

mov dx,0

label1:

; if ax is zero

cmp ax,0

je print1

;initilize bx to 10

mov bx,10

; extract the last digit

div bx

;push it in the stack

push dx

;increment the count

inc cx

;set dx to 0

xor dx,dx

jmp label1

print1:

;check if count

;is greater than zero

cmp cx,0

je exit

;pop the top of stack

pop dx

;add 48 so that it

;represents the ASCII

;value of digits

add dx,48

;interuppt to print a

;character

mov ah,02h

int 21h

;decrease the count

dec cx

jmp print1

exit:

ret

PRINT ENDP

END

**Output:**



