**Practical - 22**

**Aim:** Write an assembly program to get the answer of below given series.

1! + 2! + 3! + … + n!

**Description of instructions used:**

**MSG:** Here MSG stands for message here in place of MSG we will also use STRING.

**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.

**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)

**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

**ASSUME:** The ASSUME directive tells the assembler to assume, that a certain register contains the base of some structure (in your case: segments). In your case, CS and DS point to the code segment and the data segmentrespectively, both the one and only of their respective kind.

**DISPLAY:** For display SRTING or in here MSG.

**INT:** INT is an assembly language instruction for x86 processors that generates a software interrupt. It takes the interrupt number formatted as a byte value.

**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.

**JNE:** The JNE (or JNZ) instruction is a conditional jump that follows a test. It jumps to the specified location if the Zero Flag (ZF) is cleared (0). JNZ is commonly used to explicitly test for something not being equal to zero whereas JNE is commonly found after a CMP instruction.

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

**CALL:** The address of the next instruction that exists in the caller program (after the program **CALL** instruction) is stored in the stack. The instruction queue is emptied for accommodating the instructions of the procedure.

**HEX2DEC:** It’s Convert number hexadecimal to decimal.

**Code:**

DATA SEGMENT

NUM DB ?

FACT DB 1H

RES DB 10 DUP ('$')

MSG1 DB "ENTER NUMBER : $"

MSG2 DB 10,13,"RESULT : $"

DATA ENDS

CODE SEGMENT

ASSUME DS:DATA,CS:CODE

START:

MOV AX,DATA

MOV DS,AX

LEA DX,MSG1

MOV AH,9

INT 21H

MOV AH,1

INT 21H

SUB AL,30H

MOV NUM,AL

MOV AH,0

MOV AL,FACT

MOV CH,0

MOV CL,NUM

LABEL1:

MUL CL

LOOP LABEL1

LEA SI,RES

CALL HEX2DEC

LEA DX,MSG2

MOV AH,9

INT 21H

LEA DX,RES

MOV AH,9

INT 21H

MOV AH,4CH

INT 21H

CODE ENDS

HEX2DEC PROC NEAR

MOV CX,0

MOV BX,10

LOOP1:

MOV DX,0

DIV BX

ADD DL,30H

PUSH DX

INC CX

CMP AX,9

JG LOOP1

ADD AL,30H

MOV [SI],AL

LOOP2:

POP AX

INC SI

MOV [SI],AL

LOOP LOOP2

RET

HEX2DEC ENDP

END START

**Output:**

