Name:	Aniruddh Sawant
Roll No:	52
Class/Sem:	SE/IV
Experiment No.:	2
Title:	Program to perform multiplication without using MUL instruction
Date of Performance:	30/01/2024
Date of Submission:	6/02/2024
Marks:	
Sign of Faculty:	

Aim: Program for multiplication without using the multiplication instruction

Theory:

In the multiplication program, we multiply the two numbers without using the direct instructions MUL. Here we can successive addition methods to get the product of two numbers. For that, in one register we will take multiplicand so that we can add multiplicand itself till the multiplier stored in another register becomes zero.

ORG 100H:

It is a compiler directive. It tells the compiler how to handle source code. It tells the compiler that the executable file will be loaded at the offset of 100H (256 bytes.) **INT 21H:**

The instruction INT 21H transfers control to the operating system, to a subprogram that handles I/O operations.

MUL: MUL Source.

This instruction multiplies an unsigned byte from some source times an unsigned byte in the AL register or an unsigned word from some source times an unsigned word in the AX register.

Source: Register, Memory Location.

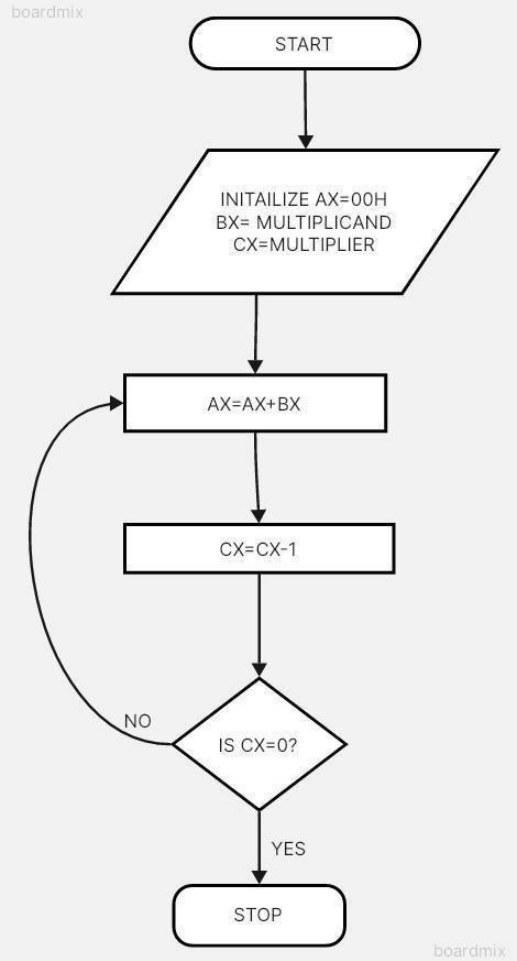
When a byte is multiplied by the content of AL, the result (product) is put in AX. A 16-bit destination is required because the result of multiplying an 8-bit number by an 8-bit number can be as large as 16-bits. The MSB of the result is put in AH and the LSB of the result is put in AL.

When a word is multiplied by the contents of AX, the product can be as large as 32 bits. The MSB of the result is put in the DX register and the LSB of the result is put in the AX register. MUL BH; multiply AL with BH; result in AX.

Algorithm:

1. Start.
2. Set AX=00H, BX= Multiplicand, CX=Multiplier 3 Add the content of AX and BX.
4. Decrement content of CX.
5. Repeat steps 3 and 4 till CX=0.
6. Stop.

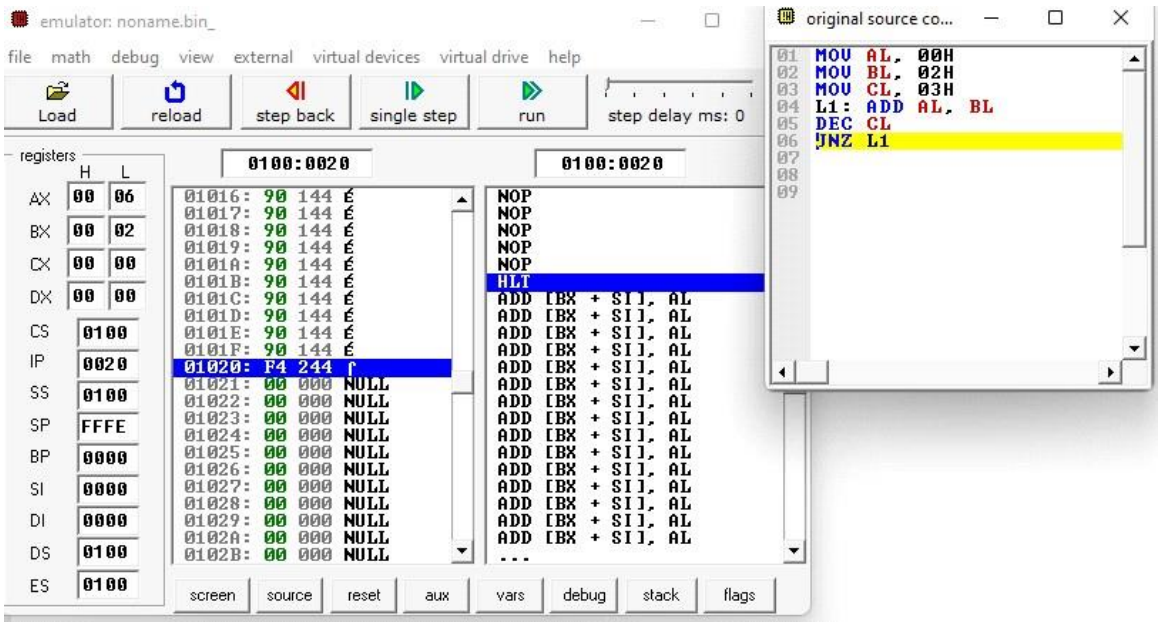
Flowchart:



Code:

```
MOV AL, 00H

MOV BL, 02H

MOV CL, 03H

L1: ADD AL, BL

DEC CL

JNZ L1
```

Output:



Conclusion:

By employing successive addition instead of the direct multiplication instruction, the program effectively computes the product of two numbers. This approach utilizes basic arithmetic operations to achieve multiplication, showcasing the versatility of assembly language in performing complex tasks through simple instructions and algorithms.

1.Explain data transfer instructions.

Data transfer instructions in the 8086 microprocessor facilitate the movement of data between registers, memory, and I/O devices. Key instructions include:

MOV: Copies data from a source operand to a destination operand.
XCHG: Swaps the contents of two operands.
PUSH: Stores data onto the stack.
POP: Retrieves data from the stack.
LEA: Loads the effective address of a memory operand into a register.
LDS and LES: Load pointer instructions used for loading segment and offset values from memory into segment and pointer registers.
LAHF and SAHF: Load and store flags instructions used for manipulating the flags register.
These instructions are essential for manipulating data within the processor and interfacing with external memory and devices in assembly language programming.

2.Explain Arithmetic instructions.

Arithmetic instructions in the 8086 microprocessor perform mathematical operations on data stored in

registers and memory. Key arithmetic instructions include:

ADD: Adds the source operand to the destination operand, storing the result in the destination.
SUB: Subtracts the source operand from the destination operand, storing the result in the destination.
INC: Increments the value of the destination operand by one.
DEC: Decrements the value of the destination operand by one.
MUL: Multiplies the source operand by the accumulator (AL or AX), storing the result in AX or DX:AX.
IMUL: Signed multiplication operation, similar to MUL but for signed numbers.
DIV: Divides the contents of DX:AX by the source operand, storing the quotient in AX and the remainder in DX.
IDIV: Signed division operation, similar to DIV but for signed numbers.
These instructions enable the 8086 processor to perform various arithmetic operations efficiently, facilitating mathematical computations in assembly language programming.

Vidyavardhini's College of Engineering & Technology
Department of Artificial Intelligence and Data Science (AI&DS)

Name:	Sarvesh Surve
Roll No:	73
Class/Sem:	SE/IV
Experiment No.:	2B
Title:	Program for calculating factorial using assembly language
Date of Performance:	24/01/2024
Date of Submission:	31/01/2024
Marks:	
Sign of Faculty:	

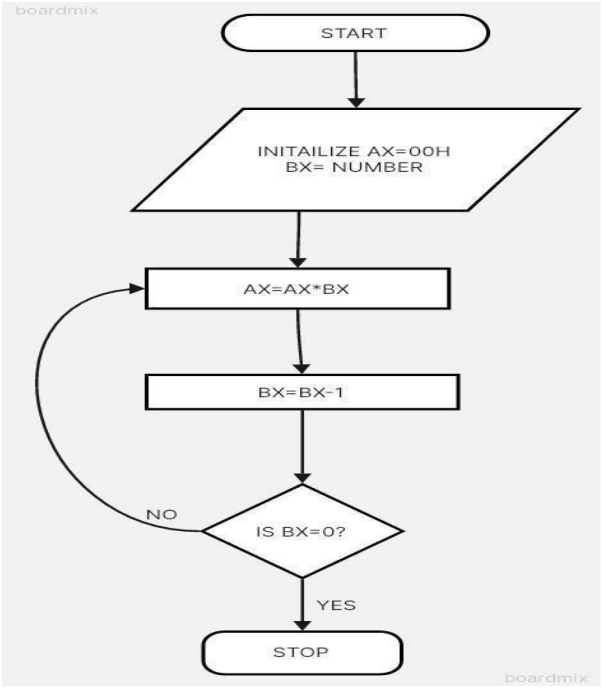
Aim: Program to calculate the Factorial of a number **Theory:**

To calculate the factorial of any number, we use MUL instruction. Here, initially, we initialize the first register by value 1. The second register is initialized by the value of the second register. After multiplication, decrement the value of the second register and repeat the multiplying step till the second register value becomes zero. The result is stored in the first register.

Algorithm:

1. Start.
2. Set AX=01H, and BX with the value whose factorial we want to find.
3. Multiply AX and BX.
4. Decrement BX=BX-1.
5. Repeat steps 3 and 4 till BX=0.
6. Stop.

Flowchart:



Code:

Using Factorial

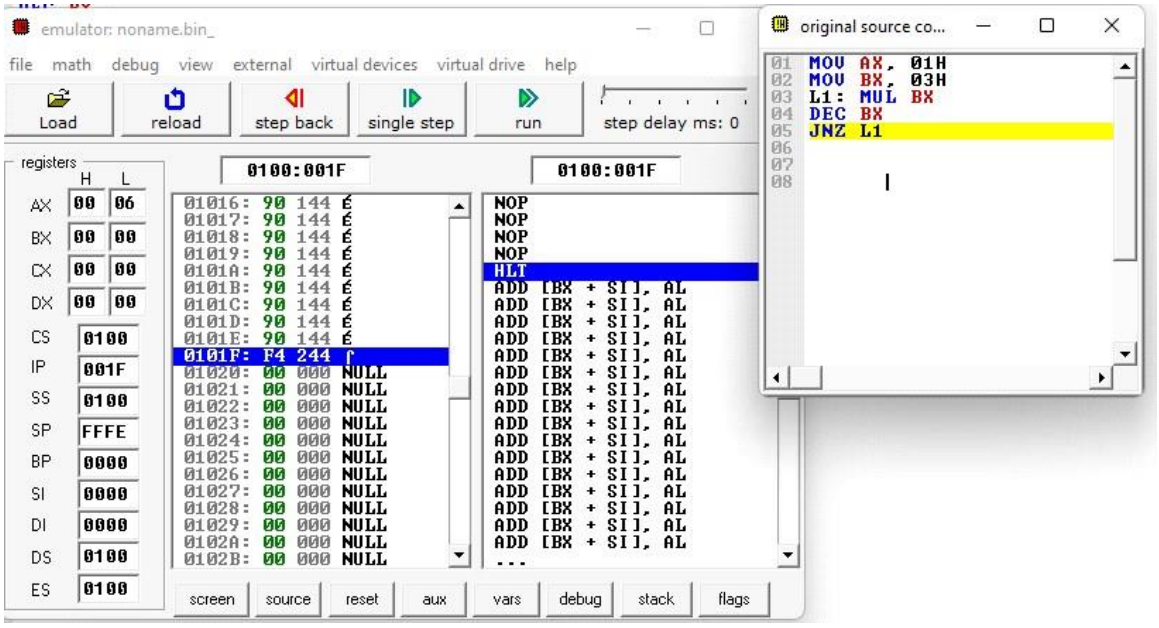
MOV AX, 01H

MOV BX, 03H

L1: MUL BX

DEC BX

JNZ L1 Output:



Conclusion:

Utilizing the MUL instruction, the program efficiently computes the factorial of a given number by iteratively multiplying the current result with decreasing values until reaching zero. This approach demonstrates the power of assembly language in implementing complex mathematical operations using simple instructions.

1.Explain shift instructions.

In the Intel 8086 assembly language, shift instructions are used to shift the bits of a binary number to the left or right. These instructions are handy for manipulating binary data and are frequently used in tasks such as multiplication, division, and bit manipulation.

Here are the commonly used shift instructions in 8086 assembly language:

SHL (Shift Left) / SAL (Arithmetic Shift Left):

SHL and SAL are synonymous in 8086 assembly.
Syntax: SHL/SAL destination, count
Description: This instruction shifts all bits in the destination operand (register or memory) to the left by the number of bits specified in the count operand. Zeros are shifted into the low-order bits, and the high-order bits are discarded. The carry flag is set to the value of the last bit shifted out.

Example: SHL AX, 1 shifts the contents of the AX register one bit to the left. SHR (Shift Right) / SAR (Arithmetic Shift Right):

SHR performs a logical shift right, while SAR performs an arithmetic shift right.
Syntax: SHR/SAR destination, count
Description: This instruction shifts all bits in the destination operand to the right by the number of bits specified in the count operand. For SHR, zeros are shifted into the high-order bits, and the loworder bits are discarded. For SAR, the sign bit is shifted into the high-order bits, preserving the sign of the original value. The carry flag is set to the value of the last bit shifted out.
Example: SHR BX, 1 shifts the contents of the BX register one bit to the right.
These shift instructions are fundamental for various bitwise operations and are extensively used in low-level programming, especially in systems programming and device driver development. They enable efficient manipulation of binary data at the bit level, providing a high degree of control over data representation and processing

2. Explain rotate instructions

In 8086 assembly language, rotate instructions are used to rotate the bits of a binary number to the left or right, similar to shift instructions. However, rotate instructions differ in that the bits that are shifted out on one end are rotated back in on the other end. This circular rotation of bits is useful in certain types of data manipulation and cryptography algorithms.

Here are the rotate instructions commonly used in 8086 assembly language:

RCL (Rotate through Carry Left):

Syntax: RCL destination, count
Description: This instruction rotates all bits in the destination operand (register or memory) to the left by the number of bit positions specified in the count operand. The carry flag is rotated into the least significant bit, and the most significant bit is rotated into the carry flag. This allows for circular rotation of the bits through the carry flag.
Example: RCL AX, 1 rotates the contents of the AX register one bit to the left through the carry flag.
RCR (Rotate through Carry Right):

Syntax: RCR destination, count
Description: This instruction rotates all bits in the destination operand to the right by the number of bit positions specified in the count operand. The carry flag is rotated into the most significant bit, and the least significant bit is rotated into the carry flag. Similar to RCL, this allows for circular rotation of the bits through the carry flag.
Example: RCR BX, 1 rotates the contents of the BX register one bit to the right through the carry flag.

ROL (Rotate Left):

Syntax: ROL destination, count
Description: This instruction rotates all bits in the destination operand to the left by the number of bit positions specified in the count operand. The most significant bit is rotated into the least significant bit, and the carry flag is set to the value of the bit shifted out of the most significant position.
Example: ROL CX, 1 rotates the contents of the CX register one bit to the left. ROR
(Rotate Right):