



**Vidyavardhini's College of Engineering and Technology**  
**Department of Artificial Intelligence & Data Science**

---

<b>Name:</b>	Charmi Ketan Jani
<b>Roll No:</b>	17
<b>Class/Sem:</b>	SE/IV
<b>Experiment No.:</b>	2B
<b>Title:</b>	Program for calculating factorial using assembly language
<b>Date of Performance:</b>	
<b>Date of Submission:</b>	
<b>Marks:</b>	
<b>Sign of Faculty:</b>	



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

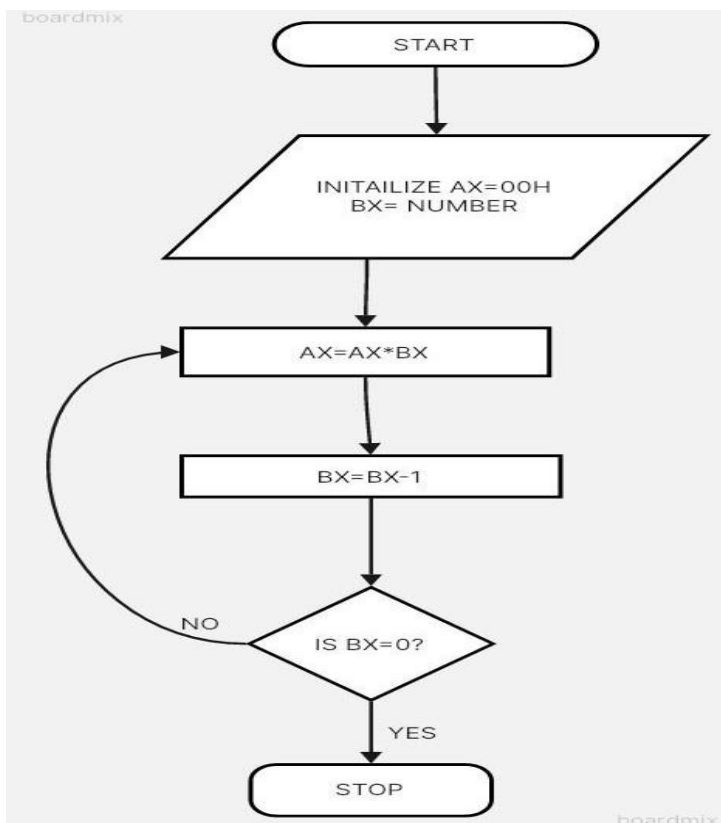
**Theory:**

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:**





Assembly Code:

```
original source co...  
01 MOV AL,01H  
02 MOV BL,04H  
03 L1:MUL BL  
04 DEC BL  
05 JNZ L1  
06  
07
```

Output:

emulator: noname.bin\_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	00	18
BX	00	01
CX	00	00
DX	00	00
CS	0100	
IP	0008	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

0100:0008

Address	Hex	Dec	Symbol
01000	B0	176	
01001	01	001	
01002	B3	179	
01003	04	004	
01004	F6	246	
01005	E3	227	
01006	FE	254	
01007	CB	203	
01008	75	117	
01009	FA	250	
0100A	90	144	
0100B	90	144	
0100C	90	144	
0100D	90	144	
0100E	90	144	
0100F	90	144	
01010	90	144	
01011	90	144	
01012	90	144	
01013	90	144	
01014	90	144	
01015	90	144	

0100:0008

```
MOV AL, 01h  
MOV BL, 04h  
MUL BL  
DEC BL  
JNE 04h  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
NOP  
...
```

screen source reset aux vars debug stack flags



### **Conclusion:**

1. Explain shift instructions.

Ans. Shift instructions are vital operations in computer architecture used to manipulate the binary representation of data by shifting its bits left or right. Here's an explanation of these instructions:

- a) Logical Shift Left (SHL): Shifts the bits of a binary number to the left by a specified number of positions, filling the vacant positions with zeros. This operation effectively multiplies the number by powers of 2.
- b) Logical Shift Right (SHR): Shifts the bits of a binary number to the right by a specified number of positions, filling the vacant positions with zeros. This operation effectively divides the number by powers of 2.
- c) Arithmetic Shift Left (SAL): Similar to logical shift left, but in arithmetic shift left, the most significant bit (MSB) is shifted into the carry flag, and the least significant bit (LSB) is set to zero. It preserves the sign of signed numbers.
- d) Arithmetic Shift Right (SAR): Similar to logical shift right, but in arithmetic shift right, the MSB is shifted into the carry flag, and the vacant positions are filled with copies of the original MSB, preserving the sign of signed numbers.

Shift instructions are essential for operations such as multiplication and division by powers of 2, as well as for manipulating binary data in bitwise operations.

2. Explain rotate instructions.

Ans. Rotate instructions are essential operations in computer architecture that manipulate the binary representation of data by circularly shifting its bits left or right. Here's an explanation of these instructions:

- a) Rotate Left (ROL): Circularly shifts the bits of a binary number to the left by a specified number of positions. The shifted-out bits are rotated back to the rightmost positions. This operation is useful for creating circular patterns in binary data.
- b) Rotate Right (ROR): Circularly shifts the bits of a binary number to the right by a specified number of positions. The shifted-out bits are rotated back to the leftmost positions. This operation is helpful for circular buffer implementations and encryption algorithms.
- c) Rotate Through Carry Left (RCL): Similar to ROL, but includes the carry flag in the rotation. The carry flag is shifted into the least significant bit position, and the most significant bit is shifted into the carry flag.
- d) Rotate Through Carry Right (RCR): Similar to ROR, but includes the carry flag in the rotation. The carry flag is shifted into the most significant bit position, and the least significant bit is shifted into the carry flag.

Rotate instructions are versatile and find applications in various bitwise operations, data manipulation, and cryptographic algorithms.

