



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

Name:	Yash Ravindra Kerkar
Roll No:	67
Class/Sem:	SE/IV
Experiment No.:	2B
Title:	Program for calculating factorial using assembly language
Date of Performance:	24/01/24
Date of Submission:	31/01/24
Marks:	
Sign of Faculty:	



Aim: Program to calculate the Factorial of a number.

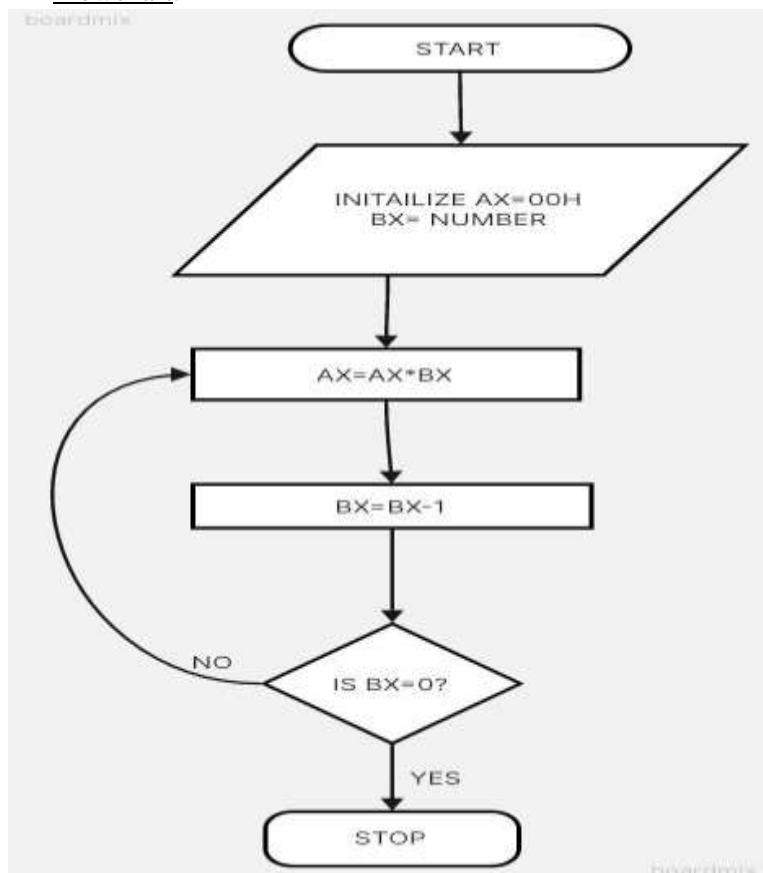
Theory:

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:





Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science (AI&DS)

Program:

```
MOV AX, 0001H
MOV BX, 0003H
L1: MUL BX
    DEC BX
    JNZ L1
```

Output:

emulator: fact.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	06
BX	00	00
CX	00	00
DX	00	00
CS	0100	
IP	001F	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

0100:001F

Address	Hex	Dec	Comment
01019:	90	144	E
0101A:	90	144	E
0101B:	90	144	E
0101C:	90	144	E
0101D:	90	144	E
0101E:	90	144	E
0101F:	F4	244	F
01020:	00	000	NULL
01021:	00	000	NULL
01022:	00	000	NULL
01023:	00	000	NULL
01024:	00	000	NULL
01025:	00	000	NULL
01026:	00	000	NULL
01027:	00	000	NULL
01028:	00	000	NULL
01029:	00	000	NULL
0102A:	00	000	NULL
0102B:	00	000	NULL
0102C:	00	000	NULL
0102D:	00	000	NULL
0102E:	00	000	NULL

0100:001F

Address	Instruction
01019:	NOP
0101A:	NOP
0101B:	NOP
0101C:	NOP
0101D:	NOP
0101E:	NOP
0101F:	HLT
01020:	ADD [BX + SI], AL
01021:	ADD [BX + SI], AL
01022:	ADD [BX + SI], AL
01023:	ADD [BX + SI], AL
01024:	ADD [BX + SI], AL
01025:	ADD [BX + SI], AL
01026:	ADD [BX + SI], AL
01027:	ADD [BX + SI], AL
01028:	ADD [BX + SI], AL
01029:	ADD [BX + SI], AL
0102A:	ADD [BX + SI], AL
0102B:	ADD [BX + SI], AL
0102C:	ADD [BX + SI], AL
0102D:	ADD [BX + SI], AL
0102E:	ADD [BX + SI], AL
...	...

screen source reset aux vars debug stack flags



Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science (AI&DS)

Conclusion: We successfully completed the assembly language program designed to calculate the factorial of a number. By implementing a combination of MOV, MUL, DEC, and JNZ instructions, we effectively computed the factorial of a given value. This hands-on experience deepened our understanding of arithmetic operations and control flow mechanisms in assembly language programming.

1. Explain shift instructions.

Ans. Arithmetic Shift Left (SHL/ SAL): Syntax: SHL destination, count OR SAL destination, count. Function: This instruction shifts the bits of the destination operand (register or memory location) to the left by the number of bit positions specified by the count operand. The empty bit positions on the right side are filled with zeros. This operation effectively multiplies the value by 2 for each shift left by one bit.

Arithmetic Shift Right (SHR): Syntax: SHR destination, count. Function: This instruction shifts the bits of the destination operand to the right by the number of bit positions specified by the count operand. The empty bit positions on the left side are filled with zeros. In signed numbers, the sign bit is retained, which means the leftmost bit (sign bit) is replicated to preserve the sign.
Logical Shift Right (SHR).

Logical Shift Right (SHR): Syntax: SHR destination, count. Function: Similar to arithmetic shift right, but in this case, the leftmost bit is always filled with zero, regardless of the sign bit of the original value. It's commonly used for unsigned integers.

Rotate Instructions (ROL, ROR, RCL, RCR): Rotate instructions are variations of shift instructions that shift the bits of the operand left or right, but the shifted out bits are circulated back to the opposite end. These instructions are useful for implementing circular buffers, encryption algorithms, and other tasks that require rotating bit patterns.

2. Explain rotate instructions.

Ans. Rotate Left (ROL): Syntax: ROL destination, count. Function: This instruction circularly shifts the bits of the destination operand (register or memory location) to the left by the number of bit positions specified by the count operand. The bits shifted out from the left side are rotated back and inserted into the right side. This operation effectively rotates the bits to the left.

Rotate Right (ROR): Syntax: ROR destination, count. Function: Similar to ROL, but rotates the bits to the right. The bits shifted out from the right side are rotated back and inserted into the left side.

Rotate Through Carry Left (RCL): Syntax: RCL destination, count. Function: This instruction performs a left rotation similar to ROL but includes the value of the Carry Flag (CF) in the rotation. The Carry Flag is shifted into the least significant bit (LSB) during the rotation, and the original LSB is shifted into the Carry Flag.

Rotate Through Carry Right (RCR): Syntax: RCR destination, count. Function: Similar to RCL, but rotates the bits to the right. The Carry Flag is shifted into the most significant bit.

