Question No. 1

➢ Code Screen Shots

```
1   [org 0x0100]
2
3   jmp start
4
5   operand1 : dw 10
6   operand2 : dw 2
7
8   sum_result : dw 0
9   subtraction_result : dw 0
10  multiplication_result : dw 0
11  division_result : dw 0
12
13
14  add1:
15
16      push bp
17      mov bp , sp
18
19      mov bx , [bp + 4]
20      mov ax , [bp + 6]
21
22      add ax , bx
23      mov [sum_result] , ax
24
25      pop bp
26
27      ret
28
29  subtract:
30      push bp
31      mov bp , sp
32
33      mov bx , [bp + 4]
34      mov ax , [bp + 6]
35
36      sub ax , bx
37      mov [subtraction_result] , ax
38
39      pop bp
40
41      ret
```

```
42
43
44  multiply:
45      push bp
46      mov bp , sp
47
48      mov bx , [bp + 4]
49      mov ax , [bp + 6]
50
51      mul bx
52      mov [multiplication_result] , ax
53
54      pop bp
55
56      ret
57
58
59  divide:
60      push bp
61      mov bp , sp
62
63      mov bx , [bp + 4]
64      mov ax , [bp + 6]
65
66      div bx
67      mov [division_result] , ax
68
69      pop bp
70
71      ret
72
73
```

```
74  start:
75      push bx
76      push ax
77
78      mov ax , [operand1]
79      push ax
80
81      mov ax , [operand2]
82      push ax
83
84      call add1
85      call subtract
86      call multiply
87      call divide
88
89      pop ax
90      pop ax
91      pop ax
92      pop bx
93
94  mov ax , 0x4c00
95  int 0x21
96
```

➢ Dos Box Screen Shots

Before Calling The Subroutines

Just Before The Interrupt Command

➢ Explanation

From main im passing the operands into the stack so they can be accessed by every subroutine after which im calling the the subroutines one by one and in each subroutine im accessing these operands that were pushed in main and through bp that has been pushed in each subroutine to access those operands and after the operation is done on operands I pop back bp which was pushed , store the result in the desired data location for storage and ret and similarly every subroutine performs on the same logic but the only difference is that the operation in each subroutine is different in main in last I pop back the values from the stack which were pushed initially to put back the initial values in the registers.

Question No. 2

- **Code Screen Shots**

```
1   [org 0x0100]
2
3   jmp start
4
5   tofindfactorial : dw 5
6   result : dw 0
7
8   factorial:
9
10      push bp
11      mov bp , sp
12
13      mov ax , [bp + 4]
14
15      cmp ax , 1
16      jne moveon
17
18  totrack:
19
20      imul ax , [bp + 4]
21      mov [bp + 4] , ax
22
23      ;mov sp , bp
24      pop bp
25      ret
26
27      moveon:
28
29      sub ax , 1
30      push ax
31
32
33      call factorial
34      pop ax
35
36  jmp totrack
37
38
```

```
39
40  start:
41
42      mov ax , [tofindfactorial]
43      push ax
44
45      call factorial
46
47      mov [result] , ax
48
49      pop ax
50
51
52
53
54  mov ax , 0x4c00
55  int 0x21
```

➢ Dos Box Screen Shots

Before Calling the Factorial Subroutine For The Value 5

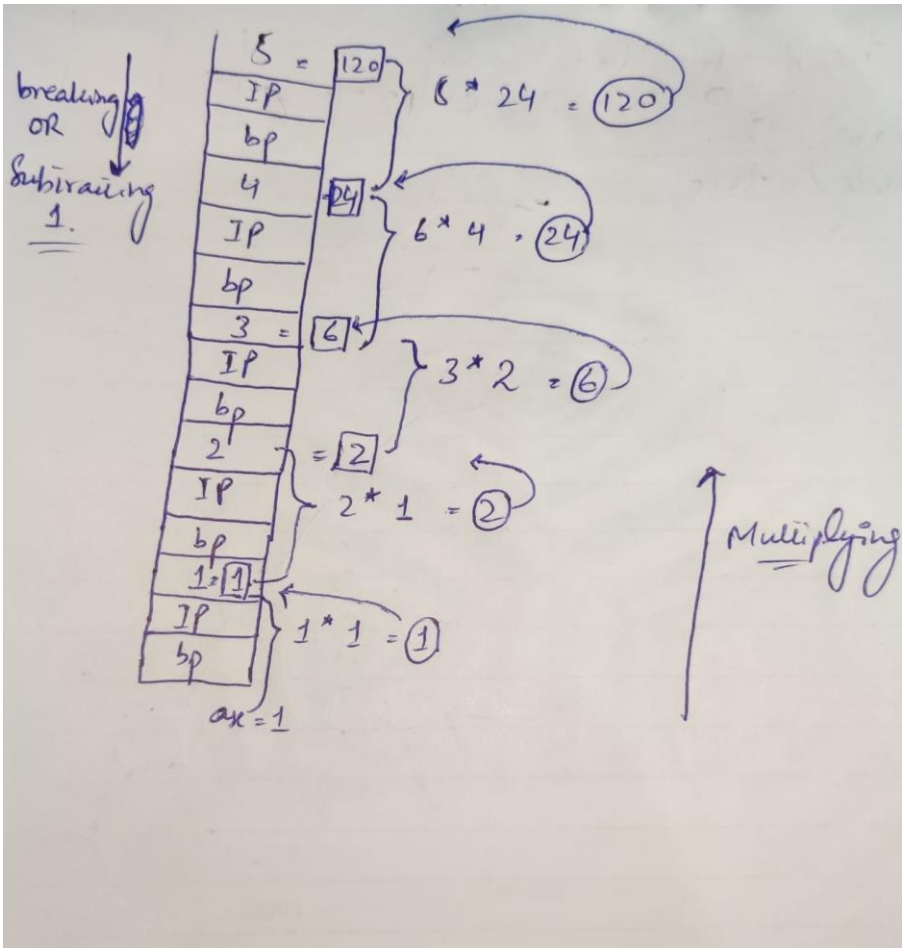
Just Before The Interrupt Command As The Factorial Of 5 Is Stored In The Data Label named as Result We Can see It Gives The Right answer Which Is 78 which's decimal is 120

➢ Explanation

Im using recursion in this question as an open choice was given to us to do anything but with recursion so im finding the factorial of any given number. Here im testing it with the value of 5 so what is happening is that im passing the value 5 to the subroutine as a parameter and is it also in the stack as it was pushed initially so what does the subroutine does is that it accessed the value 5 from the stack store it into ax and decrement it by 1 becomes 4 and again pushed it into the stack and calls the subroutine it continues happening till it reaches the base condition which is if ax is 1 whenever it is equal to 1 it will multiply the current value of ax with the value of ax which was in the previous called subroutine through [bp + 4] and storing it into [bp + 4] and then returning back to the previously called subroutine and popping ax to get the value of ax in that subroutine and continue the process till it returns back into the main from where it was initially called also im using an unconditional jump totrack to put back the program in correct place after it returns from a recursive call

Like this im using proper usage of stack and recursion in assembly to get the factorial of a given number and store it into the data label named result.

A visual representation is also given below



Just Like this it is using recursion to find the factorial of a given number.