# TASK 1:

org 0x0100

mov si, num

mov cx, 0

mov dx, 0

count\_bits:

mov al, [si]

test al, al

jz done

cmp al, '0'

jne inc\_count\_comp

inc\_count:

inc cx

inc\_count\_comp:

inc dx

inc si

jmp count\_bits

done:

cmp cl, dl

jz equal

not\_equal:

mov al, 0 ; Set AL to 0 to indicate counts are not equal

jmp exit

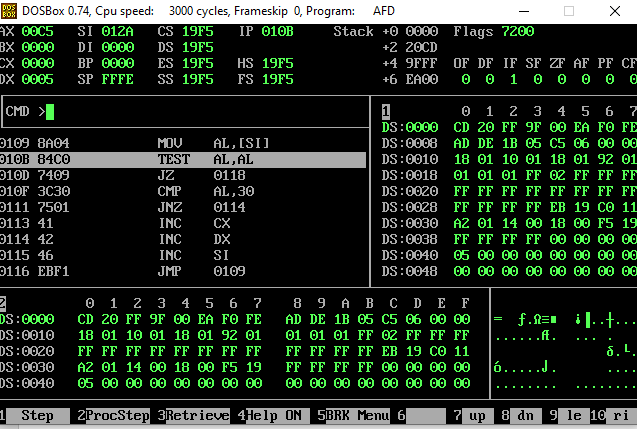
equal:

mov al, 1 ; Set AL to 1 to indicate counts are equal

exit:

int 0x21

num db 55

OUTPUT:  


TASK 2:  
org 0x0100

jmp start

calculate\_grade:

mov al, [marks] ; Load the marks from memory

cmp al, 91

ja assign\_grade\_AA

cmp al, 81

ja assign\_grade\_AB

cmp al, 71

ja assign\_grade\_BB

cmp al, 61

ja assign\_grade\_BC

cmp al, 51

ja assign\_grade\_CD

cmp al, 41

ja assign\_grade\_DD

jmp assign\_grade\_FF

assign\_grade\_AA:

mov si, grade

mov byte [si], 'A'

mov byte [si+1], 'A'

ret

assign\_grade\_AB:

mov si, grade

mov byte [si], 'A'

mov byte [si+1], 'B'

ret

assign\_grade\_BB:

mov si, grade

mov byte [si], 'B'

mov byte [si+1], 'B'

ret

assign\_grade\_BC:

mov si, grade

mov byte [si], 'B'

mov byte [si+1], 'C'

ret

assign\_grade\_CD:

mov si, grade

mov byte [si], 'C'

mov byte [si+1], 'D'

ret

assign\_grade\_DD:

mov si, grade

mov byte [si], 'D'

mov byte [si+1], 'D'

ret

assign\_grade\_FF:

mov si, grade

mov byte [si], 'F'

mov byte [si+1], 'F'

ret

start:

call calculate\_grade

mov ax, 0x4c00

int 0x21

section .data

marks db 75

section .bss

grade resb 2

# OUTPUT:

# TASK 3:

org 0x0100

start:

push word 5

call calculate\_factorial ; The result is now stored in the 'result' variable

add sp, 2 ; Clear the stack

mov ax, 0x4c00

int 0x21

calculate\_factorial:

push bx ; Save register values

push cx

push dx

mov word [result], 1 ; Initialize result to 1

; Loop to calculate factorial

pop ax ; Get the input number from the stack

mov bx, ax ; Copy input number to BX

calculate\_loop:

; Multiply result by BX

mov cx, ax ; Copy input number to CX

mov ax, [result]

call multiply ; Call multiply subroutine

mov [result], ax

dec bx ; Decrement BX

jnz calculate\_loop ; Check if BX is greater than 1

pop dx ; Restore register values

pop cx

pop bx

ret

multiply:

push dx ; Input: CX (factor1), BX (factor2), Output: AX (result)

mov ax, 0

mov dx, 0

multiply\_loop:

test cx, cx ; Check if factor1 is zero

jz multiply\_done

test bx, bx ; Check if factor2 is zero

jz multiply\_done

shl bx, 1 ; Left shift factor2 (multiply by 2)

rcl dx, 1 ; Rotate carry flag into DX

shl cx, 1 ; Left shift factor1 (divide by 2)

rcl ax, 1 ; Rotate carry flag into AX

jmp multiply\_loop

multiply\_done:

pop dx

ret

section .bss

number resw 1 ; Variable to store the input number

result resw 1 ; Variable to store the factorial result

# OUTPUT:

