# TASK 1:

org 0x0100

mov ax, 18

mov bx, 4

mov cx, 16

mov dx, 0

loop1:

test bx, 1

jz skip

add dx, ax

skip:

shl ax, 1

rcl dx, 1

shr bx, 1

loop loop1

mov ah, 4Ch

int 0x21

result db 0

# OUTPUT:

test bx, 1: This instruction tests the least significant bit of the BX. The result of this operation is not stored; it is used to set the CPU flags based on the result. Specifically, the zero flag (ZF) is set if the result is zero, and the zero flag is cleared if the result is non-zero.

jz skip\_multiply: This instruction checks the value of the zero flag that was set by the "test" instruction. If the zero flag is set (meaning the tested bit in BX was zero), the program jumps to the label skip\_multiply, skipping the "add dx, ax" instruction. This conditional jump ensures that the addition is only performed when the tested bit is set in the second operand.

# TASK 2:

org 0x0100

section .data

array db 5, 2, 9, 1, 5, 6, 3, 8, 7, 4

array\_size equ $ - array

section .text

mov ax, 0 ; Initialize outer loop index (ax) to 0

outer\_loop:

mov dx, ax ; Initialize minimum index (dx) to outer loop index

mov al, [array + ax] ; Load array[ax] into AL as the minimum

mov cx, ax ; Initialize inner loop index (cx) to outer loop index

inner\_loop:

inc cx ; Move to the next element in the inner loop

cmp cx, array\_size ; Compare inner loop index with array size

je inner\_done ; If the inner loop is done, jump to inner\_done

mov bx, cx ; Use bx as the index for memory access

mov bl, [array + bx] ; Load array[bx] into BL for comparison

mov bh, al ; Load AL into BH for comparison

cmp bl, bh ; Compare the current element with the minimum

jge continue\_loop ; If not less, continue inner loop

mov al, bl ; Update minimum if a smaller element is found

mov dx, cx ; Update minimum index

continue\_loop:

jmp inner\_loop ; Continue the inner loop

inner\_done:

; Swap array[ax] and array[dx]

mov si, ax ; Use si as the index for memory access

mov di, dx ; Use di as the index for memory access

call swap\_elements ; Call subroutine to swap the elements

inc ax ; Move to the next element in the outer loop

cmp ax, array\_size ; Compare outer loop index with array size

jne outer\_loop ; If the outer loop is not done, repeat outer loop

; Exit the program

mov ah, 0x4C

int 0x21

swap\_elements: ; Subroutine to swap two elements in the array

mov bx, ax ; Use bx as the index for memory access

mov si, [array + bx] ; Load array[bx] into SI

mov bx, dx ; Use bx as the index for memory access

mov di, [array + bx] ; Load array[bx] into DI

mov bx, ax ; Use bx as the index for memory access

mov [array + bx], di ; Store the value of DI in array[bx]

mov bx, dx ; Use bx as the index for memory access

mov [array + bx], si ; Store the value of SI in array[bx]

ret