Ejercicio 1: Simular un bucle while

```
section .data
    sum db 0       ; Variable para almacenar la suma
    count db 1     ; Contador inicializado en 1

section .text
    global _start

_start:
    mov al, [count]  ; Cargar count en AL
    mov bl, 10       ; Comparar con 10

while_loop:
    CMP al, bl       ; Verificar si count <= 10
    JG end_while     ; Si count > 10, salir del bucle

    add [sum], al    ; Sumar count a sum
    inc al           ; Incrementar count

    JMP while_loop   ; Volver al inicio del bucle

end_while:
    ; Código para terminar o mostrar sum
    ; ...
    JMP exit_program

exit_program:
    mov eax, 1       ; Syscall para salir

xor ebx, ebx     ; Código de salida 0
int 0x80
```

ciclo do-while

```
section .data
    lista db 3, 5, 7, -1, 9 ; Lista de números
    sum db 0                ; Acumular la suma
    ptr db 0                ; Puntero para la lista

section .text
    global _start

_start:
    mov si, lista           ; Inicializar el puntero a la lista

do_while_loop:
    mov al, [si]            ; Leer el número en la lista
    add [sum], al           ; Sumar el número a sum

    CMP al, 0               ; Verificar si el número es negativo
    JS end_do_while         ; Si es negativo, salir del bucle

    inc si                  ; Mover al siguiente número
    JMP do_while_loop       ; Repetir el bucle

end_do_while:
    ; Código para mostrar o terminar
    JMP exit_program

exit_program:
    mov eax, 1              ; Syscall para salir
    xor ebx, ebx            ; Código de salida 0
    int 0x80
```

Ejercicio 3: Simular un bucle for

```
section .data
    product db 1    ; Inicializar producto en 1
    i db 1          ; Inicializar contador en 1

section .text
    global _start

_start:
    mov al, [i]     ; Cargar i en AL
    mov bl, 5       ; Límite de i

for_loop:
    CMP al, bl      ; Verificar si i <= 5
    JG end_for      ; Si i > 5, salir del bucle

    imul [product], al   ; Multiplicar product por i
    inc al               ; Incrementar i

    JMP for_loop         ; Repetir el bucle

end_for:
    ; Código para terminar o mostrar product
    JMP exit_program

exit_program:
    mov eax, 1           ; Syscall para salir
    xor ebx, ebx         ; Código de salida 0
    int 0x80
```

Ejercicio 4: Simular una estructura if-else

```
section .data
    num db 5          ; Número a verificar
    result_even db 0   ; Resultado si es par
    result_odd db 0    ; Resultado si es impar

section .text
    global _start

_start:
    mov al, [num]      ; Cargar num en AL
    and al, 1          ; AND con 1 para verificar bit menos significativo
    JZ is_even         ; Si el resultado es cero, es par

is_odd:
    mov [result_odd], 1 ; Almacenar 1 en result_odd
    JMP end_if         ; Saltar al final

is_even:
    mov [result_even], 1 ; Almacenar 1 en result_even

end_if:
    JMP exit_program

exit_program:
    mov eax, 1         ; Syscall para salir
    xor ebx, ebx       ; Código de salida 0
    int 0x80
```

Ejercicio 5: Bucle for con decremento

```
section .data
    count db 10       ; Contador inicializado en 10

section .text
    global _start

_start:
    mov al, [count]   ; Cargar count en AL

for_loop:
    CMP al, 1         ; Verificar si count >= 1
    JL end_for        ; Si count < 1, salir del bucle

    ; Aquí iría el código para imprimir count
    dec al            ; Decrementar count

    JMP for_loop      ; Repetir el bucle

end_for:
    JMP exit_program

exit_program:
    mov eax, 1        ; Syscall para salir
    xor ebx, ebx      ; Código de salida 0
    int 0x80
```

suma de dos numeros

```
section .data
    num1 db 3          ; Primer número
    num2 db 2          ; Segundo número
    result db 0        ; Resultado de la suma
    msg db "Resultado: ", 0
    resultStr db "00", 10
    zeroMsg db "Esto es un cero", 10

section .text
    global _start
```

```
_start:
    mov al, [num1]
    add al, [num2]      ; Sumar num1 y num2
    mov [result], al    ; Guardar el resultado

    CMP al, 0           ; Verificar si el resultado es cero
    JZ print_zero       ; Si es cero, imprimir mensaje

    ; Convertir resultado a ASCII para mostrar
    add al, '0'
    mov [resultStr], al

    ; Imprimir mensaje inicial y resultado
    mov eax, 4          ; Syscall para escribir
    mov ebx, 1          ; Salida estándar
    mov ecx, msg        ; Dirección del mensaje
    mov edx, 11         ; Longitud del mensaje
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, resultStr
    mov edx, 2
    int 0x80
    JMP exit_program
```

```
    mov eax, 4
    mov ebx, 1
    mov ecx, resultStr
    mov edx, 2
    int 0x80
    JMP exit_program

print_zero:
    mov eax, 4
    mov ebx, 1
    mov ecx, zeroMsg
    mov edx, 15
    int 0x80

exit_program:
    mov eax, 1
    xor ebx, ebx
    int 0x80
```