```
section .data
    sum db 0            ; Variable para almacenar la suma
    count db 1          ; Variable para el contador (inicializado en 1)

section .text
    global _start

_start:
    ; Inicializamos sum y count
    mov al, 0           ; sum = 0
    mov [sum], al       ; Almacenamos sum en memoria
    mov al, 1           ; count = 1
    mov [count], al     ; Almacenamos count en memoria

loop_start:
    ; Cargar count en el registro AL
    mov al, [count]

    ; Comparar count con 10
    cmp al, 10
    jg loop_end         ; Si count > 10, salir del bucle

    ; Sumar count a sum
    add [sum], al       ; sum = sum + count

    ; Incrementar count
    inc byte [count]    ; count = count + 1

    ; Volver al inicio del bucle
    jmp loop_start

loop_end:
    ; El bucle termina aquí, en este punto tenemos el valor de sum con la suma final
    ; (sum = 55 si todo funciona correctamente)

    ; Salir del programa
    mov eax, 1          ; Código de salida (exit)
    xor ebx, ebx        ; Estado de salida (0)
    int 0x80            ; Llamada al sistema para salir
```

```
section .data
    lista db 5, 3, 7, 2, -1  ; Lista de números (terminada con un número negativo)
    sum db 0                 ; Variable para almacenar la suma

section .text
    global _start

_start:
    ; Inicialización
    mov al, 0               ; sum = 0
    mov [sum], al           ; Almacenar sum en memoria
    lea esi, [lista]        ; Cargar la dirección del inicio de la lista en el puntero ESI

do_while_loop:
    ; Leer el primer número desde la lista
    mov al, [esi]           ; Cargar el número actual de la lista en AL
    add [sum], al           ; sum = sum + al (sumamos el número actual)

    ; Comprobar si el número es negativo
    js  loop_end            ; Si el número es negativo (al estar en AL), salir del bucle

    ; Mover el puntero al siguiente número de la lista
    inc esi                 ; Avanzar el puntero (puntero = puntero + 1)

    ; Repetir el ciclo
    jmp do_while_loop       ; Volver al inicio del ciclo

loop_end:
    ; Aquí termina el bucle, el valor de sum tiene la suma final

    ; Finalizar el programa
    mov eax, 1              ; Código de salida (exit)
    xor ebx, ebx            ; Estado de salida (0)
    int 0x80                ; Llamada al sistema para salir
```

```
section .data
    product dd 1        ; Inicializa el valor de product a 1

section .bss
    i resd 1            ; Variable i

section .text
    global _start

_start:
    ; Inicializa i a 1
    mov dword [i], 1

    ; Bucle for (i <= 5)
for_loop:
    ; Cargar el valor de i en el registro eax
    mov eax, [i]

    ; Comparar i con 6 (si i > 5, salir del bucle)
    cmp eax, 6
    jg end_loop         ; Si i > 5, salta a end_loop

    ; Multiplicar product por i
    mov eax, [product] ; Cargar product en eax
    imul eax, [i]       ; Multiplicar product * i
    mov [product], eax ; Almacenar el resultado en product

    ; Incrementar i
    mov eax, [i]
    inc eax
    mov [i], eax        ; Guardar el nuevo valor de i

    ; Repetir el bucle
    jmp for_loop

end_loop:
    ; Aquí el resultado final de product está almacenado en la variable 'product'
    ; Se podría agregar código para imprimir el resultado si fuera necesario

    ; Finalizar el programa (salida limpia)
    mov eax, 1          ; Código de salida para Linux
    xor ebx, ebx        ; Código de salida 0
    int 0x80            ; Interrupción para salir
```

```
section .data
    num dd 7              ; Número de entrada (puedes cambiarlo)
    result_even db "El numero es par", 0  ; Mensaje para números pares
    result_odd db "El numero es impar", 0 ; Mensaje para números impares

section .bss
    result resb 50        ; Buffer para almacenar el resultado (mensaje)

section .text
    global _start

_start:
    ; Cargar el valor de num en el registro eax
    mov eax, [num]

    ; Verificar si el número es par o impar utilizando AND
    and eax, 1            ; AND con 1, mantiene solo el bit menos significativo

    ; Si el resultado es 0 (par), saltamos a result_even
    cmp eax, 0
    je  even_case         ; Si es igual a 0, es par

    ; Si no es 0, el número es impar, almacenamos el mensaje en result_odd
    mov eax, [result_odd]
    mov [result], eax     ; Guardamos el mensaje en result
    jmp end_program       ; Saltamos al final del programa

even_case:
    ; Si es par, almacenamos el mensaje en result_even
    mov eax, [result_even]
    mov [result], eax     ; Guardamos el mensaje en result

end_program:
    ; El valor de 'result' contiene el mensaje adecuado (par o impar)

    ; Finalizar el programa (salida limpia)
    mov eax, 1            ; Código de salida para Linux
    xor ebx, ebx          ; Código de salida 0
    int 0x80              ; Interrupción para salir
```

```
section .data
    msg db "Numero: %d", 10, 0  ; Mensaje de formato para imprimir, con salto de línea

section .bss
    num resb 4                  ; Buffer para almacenar el número a imprimir

section .text
    global _start

_start:
    ; Inicialización de count en 10
    mov ecx, 10                 ; count = 10

for_loop:
    ; Comprobar si count >= 1
    cmp ecx, 1
    jl loop_end                 ; Si count < 1, salir del bucle

    ; Almacenar el valor de count en el buffer num
    mov eax, [ecx]              ; Cargar el valor de count en eax (por simplicidad)

    ; Llamada al sistema para imprimir el número
    ; Convertir el número en cadena para imprimirlo
    mov [num], eax              ; Guardar el número a imprimir
    ; Llamar a una función para imprimir el valor de num

    ; Decrementar count
    dec ecx                     ; count = count - 1

    ; Continuar con el bucle
    jmp for_loop

loop_end:
    ; Terminar el programa
    mov eax, 1                  ; Código de salida (exit)
    xor ebx, ebx                ; Estado de salida (0)
    int 0x80                    ; Llamada al sistema para salir
```

```
section .data
    num1 db '3'         ; Primer número en formato de carácter (puedes cambiarlo)
    num2 db '4'         ; Segundo número en formato de carácter (puedes cambiarlo)
    msg_zero db "Esto es un cero", 0  ; Mensaje que se muestra si la suma es 0
    msg_result db "Resultado: ", 0     ; Mensaje para mostrar el resultado

section .bss
    result resb 4       ; Buffer para almacenar el resultado de la suma

section .text
    global _start

_start:
    ; Cargar el primer número (num1) en al (como un valor ASCII)
    mov al, [num1]
    sub al, '0'         ; Convertir de ASCII a número

    ; Cargar el segundo número (num2) en bl (como un valor ASCII)
    mov bl, [num2]
    sub bl, '0'         ; Convertir de ASCII a número

    ; Sumar los dos números
    add al, bl

    ; Verificar si el resultado es 0
    cmp al, 0
    je  print_zero      ; Si es igual a 0, salta a imprimir "Esto es un cero"

    ; Si no es 0, imprimir el mensaje de resultado y el valor
    mov edx, msg_result
    call PrintString    ; Llamar a la función PrintString para imprimir "Resultado: "

    ; Convertir el número a carácter ASCII
    add al, '0'         ; Convertir el número de vuelta a carácter ASCII
    mov [result], al    ; Guardar el resultado en el buffer result

    ; Llamar a PrintString para imprimir el resultado
    mov edx, result
    call PrintString

    ; Salir del programa
    jmp exit_program

print_zero:
    ; Imprimir "Esto es un cero"
    mov edx, msg_zero
    call PrintString    ; Llamar a la función PrintString para imprimir el mensaje
    ; Convertir el número a carácter ASCII
    add al, '0'         ; Convertir el número de vuelta a carácter ASCII
    mov [result], al    ; Guardar el resultado en el buffer result

    ; Llamar a PrintString para imprimir el resultado
    mov edx, result
    call PrintString

    ; Salir del programa
    jmp exit_program

print_zero:
    ; Imprimir "Esto es un cero"
    mov edx, msg_zero
    call PrintString    ; Llamar a la función PrintString para imprimir el mensaje

    ; Salir del programa
    jmp exit_program

; Función para imprimir una cadena (usando llamada al sistema Linux)
PrintString:
    ; edx = dirección de la cadena
    mov eax, 4          ; sys_write
    mov ebx, 1          ; salida estándar (stdout)
    int 0x80            ; llamada al sistema
    ret

exit_program:
    ; Salir del programa
    mov eax, 1          ; sys_exit
    xor ebx, ebx        ; Código de salida 0
    int 0x80            ; Llamada al sistema
```