TALLER #8

---

1. Comparador de Números: Escribir un programa que reciba dos números y determine si son iguales, si uno es mayor que el otro, o si son negativos.

```
sección .data
    msg_equal db  'Los numeros son iguales' , 0xA , 0
    msg_greater db  'El primer número es mayor' , 0xA , 0
    msg_smaller db  'El primer numero es menor' , 0xA , 0
    msg_negative db  'El numero es negativo' , 0xA , 0

sección .bss
    num1 resb  1
    num2 resb  1

sección .texto
     inicio global

_comenzar:
    ; Leer los dos números
    ; (Código de lectura aquí...)

    ; Comparar los números
    mov  al , [núm1]        ; Cargar el primer número
    cmp  al , [núm2]        ; Comparar con el segundo número
    soy   igual_flag        ; Si son iguales, salta a igual_
    jl   bandera_más pequeña      ; Si el primer número es m
    jg   mayor_flag     ; Si el primer número es mayor, sal

bandera de igualdad:
    mov  ecx , msg_equal
    jmp imprimir_resultado

bandera_más_pequeña:
    mov  ecx , msg_smaller
    jmp imprimir_resultado

bandera_mayor:
    mov  ecx , msg_greater
```

2. Clasificación de Números: Leer un número y clasificarlo como positivo, negativo o cero.

```
SECCIÓN .data
    msg_pedir    db   "Ingrese un número: " , 0
    len_pedir    equ $ - msg_pedir

    msg_pos      db   "El numero es positivo" , 10 , 0
    msg_neg      db   "El número es negativo" , 10 , 0
    msg_cero     db   "El numero es cero" , 10 , 0

SECCIÓN .bss
    búfer       resb  16       ; Para leer el número

SECCIÓN .texto
     inicio global

_comenzar:

; Imprimir mensaje "Ingrese un número: "

mov  eax , 4            ; sys_write
mov  ebx , 1            ; stdout
mov  ecx , msg_pedir
mov  edx , len_pedir
entero  0x80


Leer desde el teclado

mov  eax , 3            ; sys_read
mov  ebx , 0            ; stdin
mov  ecx , buffer
mov  edx , 16
entero  0x80
```

3. Par o Impar: Leer un número y determinar si es par o impar usando únicamente la bandera de paridad (PF).

```
SECCIÓN .data
    num db  7                   ; número a evaluar
    msg_par      db  "PAR" , 0
    msg_impar    db  "IMPAR" , 0

SECCIÓN .texto
 inicio global

_comenzar:

    mov  al , [núm]         ; cargar numero en AL
    diciembre  al               ; modifica PF según la paridad

    jp es_impar           ; JP = Saltar si Paridad = 1 → IMPAR
                        ; (porque tras DEC, impar produce par)

es_par:
    mov  ecx , msg_par
    imprimir

es_impar:
    mov  ecx , msg_impar

imprimir:
    ; Aquí iría el código real para imprimir,
    ; omitido para mantener el programa corto.

    mover  eax , 1
    entero  0x80
```

4. Simulación de Overflow: Pedir dos números y sumarlos, verificando si ocurre desbordamiento con la bandera OF (Overflow Flag). Imprimir un mensaje si se detecta overflow.

```
SECCIÓN .data
    número1 dd  2000000000          ; números grandes para provocar desbor
    num2 dd  2000000000

    msg_no_of db  "Desbordamiento de SIN" , 0
    Mensaje de     la base de datos  "Desbordamiento de HAY" , 0

SECCIÓN .texto
 inicio global

_comenzar:

    mover  eax , [num1]
    agregar  eax , [num2]         ; Esta instrucción modifica la bandera O

    jo overflow_detectado ; Si OF = 1 → Hubo desbordamiento

sin desbordamiento:
    mov  ecx , msg_no_of
    imprimir

desbordamiento_detectado:
    mov  ecx , msg_of

imprimir:
    ; Código real de impresión omitido para mantener corto
    ; Aquí solo terminamos el programa

    mover  eax , 1
    entero  0x80
```

5. Simulación de Acarreo: Realizar una suma entre dos números y verificar si hay un acarreo con la bandera CF (Carry Flag). Mostrar si se generó un acarreo o no.

```
SECCIÓN .data
    número1    db  200          ; numeros pequeños para forzar el carr
    num2    db  100

    msg_carry     db  "Se produjo ACARREO" , 0
    msg_nocarry   db  "NO hubo acarreo" , 0

SECCIÓN .texto
 inicio global

_comenzar:

    mov  al , [num1]
    añadir  al , [num2]         ; La instrucción ADD modifica CF autom

    jc hubo_carry           ; Si CF=1 → hubo acarreo
                          (saltar si se lleva)

no_carry:
    mov  ecx , msg_nocarry
    imprimir

hubo_carry:
    mov  ecx , msg_carry

imprimir:
    ; Aquí iría el código real para imprimir usando sys_write.
    ; Omitido para mantener el código lo más corto posible.

    mover  eax , 1
    entero  0x80
```

6. Minimo y Máximo de Tres Números: Leer tres números e identificar el menor y el mayor.

```
SECCIÓN .data
    num1 dd  12
    num2 dd - 8
    num3 dd  50

SECCIÓN .bss
    menor resd  1
    alcalde resd  1

SECCIÓN .texto
     inicio global

_comenzar:
mover  eax , [num1]
mov  ebx , [num2]

cmp  ebx , eax
jle chk3_max
mov  eax , ebx          ; num2 alcalde

hk3_max:
mov  ecx , [num3]
cmp  ecx , eax
jle mayor_listo
mov  eax , ecx          ; num3 alcalde

ista_mayor:
mov [alcalde], eax      ; guardar mayor
```

7. Ordenamiento de Dos Números
Leer dos números e intercambiarlos si no están en orden ascendente usando solo saltos condicionales.

```
SECCIÓN .data
    un dd  30
    b dd  12

SECCIÓN .texto
     inicio global

_comenzar:

    mov  eax , [a]       ; eax = a
    mov  ebx , [b]       ; ebx = b

    cmp  eax , ebx       ; ¿a > b?
    jle ya_ordenados ; si a <= b → ya están en orden

    ; ----------------------------
    Intercambiar valores (swap)
    ; ----------------------------
    mov  edx , eax       ; temp = a
    mov  eax , ebx       ; a = b
    mov  ebx , edx       ; b = temp

    mov [a], eax
    mov [b], ebx

ya_ordenados:

    mover  eax , 1         ; salir
    entero  0x80
```

8. Ciclo de Conteo sin Comparaciones: Implementar un contador de 0 a 9.

```
SECCIÓN .texto
 inicio global

_comenzar:

    xor  eax , eax          ; eax = 0 (contador)
    mov  ecx , 10           ; límite(10)

siguiente:
    ; Aquí se podría imprimir el valor de eax si se quisiera
    ; (omitido para mantener corto)

    inc  eax                ; contador++
    agregar  ecx , -1           ; resta 1 a ecx

    jo fin                  ; si overflow → terminar (sin comparaci
    jmp siguiente           ; seguir contando

aleta:
    mover  eax , 1
    entero  0x80
```