



**Universidad Autónoma De Baja California**  
**Facultad de Ingeniería, Arquitectura y Diseño**



# Repositorio

Organización de computadoras



Profesor: Jonatan Crespo Ragland  
Alumno: Montano Valencia Mike Armando  
Matricula: 379746  
Grupo: 932

Ensenada, B.C. 3 de Octubre del 2025

## REPOSITORIO

a. Modifica el código para que imprima los siguientes caracteres utilizando solo sumas:

### 1. A

```
1. section .data
2.   num1 db 10      ; Primera variable (entre 1 y 3)
3.   num2 db 7       ; Segunda variable (entre 1 y 3)
4.   result db 0     ; Espacio para almacenar el resultado convertido a
   ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.    mov al, [num1]  ; Cargar num1 en AL
11.    add al, [num2]  ; Sumar num2 a AL
12.    add al, '0'     ; Convertir el resultado a ASCII
13.
14.    mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.    ; Imprimir el número (un solo dígito)
17.    mov eax, 4      ; syscall: sys_write
18.    mov ebx, 1      ; file descriptor: stdout
19.    mov ecx, result ; Dirección del resultado
20.    mov edx, 1      ; Longitud del resultado
21.    int 0x80        ; Llamada al sistema
22.
23.    ; Salir del programa
24.    mov eax, 1      ; syscall: sys_exit
25.    xor ebx, ebx    ; Código de salida 0
26.    int 0x80        ; Llamada al sistema
```

**1. :**

```
1. section .data
2.   num1 db 5      ; Primera variable (entre 1 y 3)
3.   num2 db 5      ; Segunda variable (entre 1 y 3)
4.   result db 0    ; Espacio para almacenar el resultado convertido a
   ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.  mov al, [num1]  ; Cargar num1 en AL
11.  add al, [num2]  ; Sumar num2 a AL
12.  add al, '0'     ; Convertir el resultado a ASCII
13.
14.  mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.  ; Imprimir el número (un solo dígito)
17.  mov eax, 4      ; syscall: sys_write
18.  mov ebx, 1      ; file descriptor: stdout
19.  mov ecx, result ; Dirección del resultado
20.  mov edx, 1      ; Longitud del resultado
21.  int 0x80        ; Llamada al sistema
22.
23.  ; Salir del programa
24.  mov eax, 1      ; syscall: sys_exit
25.  xor ebx, ebx    ; Código de salida 0
26.  int 0x80        ; Llamada al sistema
```

**3. =**

```
1. section .data
2.   num1 db 6      ; Primera variable (entre 1 y 3)
3.   num2 db 7      ; Segunda variable (entre 1 y 3)
4.   result db 0    ; Espacio para almacenar el resultado convertido a
   ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.  mov al, [num1]  ; Cargar num1 en AL
11.  add al, [num2]  ; Sumar num2 a AL
12.  add al, '0'     ; Convertir el resultado a ASCII
13.
14.  mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.  ; Imprimir el número (un solo dígito)
17.  mov eax, 4      ; syscall: sys_write
18.  mov ebx, 1      ; file descriptor: stdout
19.  mov ecx, result ; Dirección del resultado
20.  mov edx, 1      ; Longitud del resultado
21.  int 0x80        ; Llamada al sistema
22.
23.  ; Salir del programa
24.  mov eax, 1      ; syscall: sys_exit
25.  xor ebx, ebx    ; Código de salida 0
26.  int 0x80        ; Llamada al sistema
```

#### 4. ?

```
1. section .data
2.   num1 db 10      ; Primera variable (entre 1 y 3)
3.   num2 db 5       ; Segunda variable (entre 1 y 3)
4.   result db 0     ; Espacio para almacenar el resultado convertido a
   ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.    mov al, [num1]  ; Cargar num1 en AL
11.    add al, [num2]  ; Sumar num2 a AL
12.    add al, '0'     ; Convertir el resultado a ASCII
13.
14.    mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.    ; Imprimir el número (un solo dígito)
17.    mov eax, 4      ; syscall: sys_write
18.    mov ebx, 1      ; file descriptor: stdout
19.    mov ecx, result ; Dirección del resultado
20.    mov edx, 1      ; Longitud del resultado
21.    int 0x80        ; Llamada al sistema
22.
23.    ; Salir del programa
24.    mov eax, 1      ; syscall: sys_exit
25.    xor ebx, ebx    ; Código de salida 0
26.    int 0x80        ; Llamada al sistema
```

## 5. \_

```
1. section .data
2.   num1 db 46      ; Primera variable (entre 1 y 3)
3.   num2 db 1       ; Segunda variable (entre 1 y 3)
4.   result db 0     ; Espacio para almacenar el resultado convertido a
   ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.  mov al, [num1]   ; Cargar num1 en AL
11.  add al, [num2]   ; Sumar num2 a AL
12.  add al, '0'      ; Convertir el resultado a ASCII
13.
14.  mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.  ; Imprimir el número (un solo dígito)
17.  mov eax, 4       ; syscall: sys_write
18.  mov ebx, 1       ; file descriptor: stdout
19.  mov ecx, result   ; Dirección del resultado
20.  mov edx, 1       ; Longitud del resultado
21.  int 0x80         ; Llamada al sistema
22.
23.  ; Salir del programa
24.  mov eax, 1       ; syscall: sys_exit
25.  xor ebx, ebx     ; Código de salida 0
26.  int 0x80         ; Llamada al sistema
```

b. Ahora modificarlo para imprima los siguientes caracteres utilizando al menos una resta dentro del código:

### 1. B

```
1. section .data
2.     num1 db 19      ; Primera variable (entre 1 y 3)
3.     num2 db 1       ; Segunda variable (entre 1 y 3)
4.     result db 0     ; Espacio para almacenar el resultado convertido a
        ASCII
5.
6. section .text
7.     global _start
8.
9. _start:
10.     mov al, [num1]   ; Cargar num1 en AL
11.     sub al, [num2]   ; Sumar num2 a AL
12.     add al, '0'      ; Convertir el resultado a ASCII
13.
14.     mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.     ; Imprimir el número (un solo dígito)
17.     mov eax, 4       ; syscall: sys_write
18.     mov ebx, 1       ; file descriptor: stdout
19.     mov ecx, result  ; Dirección del resultado
20.     mov edx, 1       ; Longitud del resultado
21.     int 0x80         ; Llamada al sistema
22.
23.     ; Salir del programa
24.     mov eax, 1       ; syscall: sys_exit
25.     xor ebx, ebx     ; Código de salida 0
26.     int 0x80         ; Llamada al sistema
```

## 2. x

```
1. section .data
2.   num1 db 74      ; Primera variable (entre 1 y 3)
3.   num2 db 2       ; Segunda variable (entre 1 y 3)
4.   result db 0     ; Espacio para almacenar el resultado convertido a
   ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.  mov al, [num1]   ; Cargar num1 en AL
11.  sub al, [num2]   ; Sumar num2 a AL
12.  add al, '0'      ; Convertir el resultado a ASCII
13.
14.  mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.  ; Imprimir el número (un solo dígito)
17.  mov eax, 4       ; syscall: sys_write
18.  mov ebx, 1       ; file descriptor: stdout
19.  mov ecx, result   ; Dirección del resultado
20.  mov edx, 1       ; Longitud del resultado
21.  int 0x80         ; Llamada al sistema
22.
23.  ; Salir del programa
24.  mov eax, 1       ; syscall: sys_exit
25.  xor ebx, ebx     ; Código de salida 0
26.  int 0x80         ; Llamada al sistema
```



### 3. +

```
1. section .data
2.   num1 db 5      ; Primera variable (entre 1 y 3)
3.   num2 db 10     ; Segunda variable (entre 1 y 3)
4.   result db 0    ; Espacio para almacenar el resultado convertido a
   ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.  mov al, [num1]  ; Cargar num1 en AL
11.  sub al, [num2]  ; Sumar num2 a AL
12.  add al, '0'     ; Convertir el resultado a ASCII
13.
14.  mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.  ; Imprimir el número (un solo dígito)
17.  mov eax, 4      ; syscall: sys_write
18.  mov ebx, 1      ; file descriptor: stdout
19.  mov ecx, result ; Dirección del resultado
20.  mov edx, 1      ; Longitud del resultado
21.  int 0x80        ; Llamada al sistema
22.
23.  ; Salir del programa
24.  mov eax, 1      ; syscall: sys_exit
25.  xor ebx, ebx    ; Código de salida 0
26.  int 0x80        ; Llamada al sistema
```

#### 4. ‘

```
1. section .data
2.   num1 db 10      ; Primera variable (entre 1 y 3)
3.   num2 db 19      ; Segunda variable (entre 1 y 3)
4.   result db 0     ; Espacio para almacenar el resultado convertido a
   ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.    mov al, [num1]  ; Cargar num1 en AL
11.    sub al, [num2]  ; Sumar num2 a AL
12.    add al, '0'     ; Convertir el resultado a ASCII
13.
14.    mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.    ; Imprimir el número (un solo dígito)
17.    mov eax, 4      ; syscall: sys_write
18.    mov ebx, 1      ; file descriptor: stdout
19.    mov ecx, result ; Dirección del resultado
20.    mov edx, 1      ; Longitud del resultado
21.    int 0x80        ; Llamada al sistema
22.
23.    ; Salir del programa
24.    mov eax, 1      ; syscall: sys_exit
25.    xor ebx, ebx    ; Código de salida 0
26.    int 0x80        ; Llamada al sistema
```

```

5. {
1. section .data
2.     num1 db 80      ; Primera variable (entre 1 y 3)
3.     num2 db 5       ; Segunda variable (entre 1 y 3)
4.     result db 0     ; Espacio para almacenar el resultado convertido a
        ASCII
5.
6. section .text
7.     global _start
8.
9. _start:
10.    mov al, [num1]   ; Cargar num1 en AL
11.    sub al, [num2]   ; Sumar num2 a AL
12.    add al, '0'      ; Convertir el resultado a ASCII
13.
14.    mov [result], al ; Guardar el carácter ASCII en 'result'
15.
16.    ; Imprimir el número (un solo dígito)
17.    mov eax, 4       ; syscall: sys_write
18.    mov ebx, 1       ; file descriptor: stdout
19.    mov ecx, result  ; Dirección del resultado
20.    mov edx, 1       ; Longitud del resultado
21.    int 0x80         ; Llamada al sistema
22.
23.    ; Salir del programa
24.    mov eax, 1       ; syscall: sys_exit
25.    xor ebx, ebx     ; Código de salida 0
26.    int 0x80         ; Llamada al sistema

```

## TALLER 6

```
1. section .data
2.   num1  db 2      ; Primera variable (entre 1 y 3)
3.   num2  db 1      ; Segunda variable (entre 1 y 3)
4.   result db 0      ; Espacio para almacenar el resultado convertido
   a ASCII
5.
6. section .text
7.   global _start
8.
9. _start:
10.    mov al, [num1]   ; Cargar num1 en AL
11.    add al, [num2]   ; Sumar num2 a AL      ; AL ahora contiene
   la suma numérica
12.
13.    cmp al, 3        ; Compara la suma con 3
14.    je es_tres       ; Si la suma == 3, salta a la etiqueta 'es_tres'
15.
16.    ; --- Caso por defecto: convertir a ASCII y guardar ---
17.    add al, '0'      ; Convertir el resultado numérico a ASCII ('0' +
   valor)
18.    mov [result], al  ; Guardar el carácter ASCII en 'result'
19.    jmp imprimir      ; Saltar a la rutina de impresión
20.
21. es_tres:
22.    mov byte [result], 'T' ; Ejemplo: si la suma es 3, guardamos 'T'
   en lugar del dígito
23.
24. imprimir:
25.    ; Imprimir el número (un solo dígito o 'T')
26.    mov eax, 4        ; syscall: sys_write
27.    mov ebx, 1        ; file descriptor: stdout
28.    mov ecx, result    ; Dirección del resultado
29.    mov edx, 1        ; Longitud del resultado
30.    int 0x80          ; Llamada al sistema
31.
32.    ; Salir del programa
33.    mov eax, 1        ; syscall: sys_exit
34.    xor ebx, ebx      ; Código de salida 0
35.    int 0x80          ; Llamada al sistema
```

## TALLER 8

1.

section .data

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

section .bss

```
num1 resb 1
num2 resb 1
```

section .text

```
global _start
```

\_start:

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

```
; Comparar los números
mov al, [num1]    ; Cargar el primer número
cmp al, [num2]    ; Comparar con el segundo número
je equal_flag     ; Si son iguales, saltar a equal_flag
jl smaller_flag   ; Si el primer número es menor, saltar a smaller_flag
jg greater_flag   ; Si el primer número es mayor, saltar a greater_flag
```

equal\_flag:

```
mov ecx, msg_equal
jmp print_result
```

smaller\_flag:

```
mov ecx, msg_smaller
jmp print_result
```

greater\_flag:

```
mov ecx, msg_greater
jmp print_result
```

print\_result:

```
; (Código para imprimir el resultado)
; (Código de salida aquí...)
```

2.

section .data

msg\_cero db 'Es cero', 0xA, 0

msg\_pos db 'Es positivo', 0xA, 0

msg\_neg db 'Es negativo', 0xA, 0

section .bss

num resb 1

section .text

global \_start

\_start:

; comparar EAX con 0

cmp eax, 0

je es\_cero

jl es\_negativo

jg es\_positivo

es\_cero:

; ... (comentario: print "Cero")

mov ecx, msg\_cero

jmp print\_result

es\_negativo:

; ... (comentario: print "Negativo")

mov ecx, msg\_neg

jmp print\_result

es\_positivo:

; ... (comentario: print "Positivo")

mov ecx, msg\_pos

jmp print\_result

print\_result:

; (Código para imprimir el resultado)

; (Código de salida aquí...)

3.

```
section .data
```

```
    msg_par db 'Es par', 0xA, 0
```

```
    msg_impar db 'Es impar', 0xA, 0
```

```
section .bss
```

```
    num resb 1
```

```
section .text
```

```
    global _start
```

```
_start:
```

```
    ; aislar el bit 0: AL := AL & 1
```

```
    and     al, 1          ; actualiza banderas; PF = 1 si el número de 1 bits en  
AL&1 es par
```

```
                ; -> si AL&1 = 0 => cero bits a 1 => PF = 1 (par)
```

```
                ; -> si AL&1 = 1 => un bit a 1  => PF = 0 (impar)
```

```
    jp      es_par        ; JP = Jump if Parity (PF = 1) => número par
```

```
    jnp     es_impar      ; JNP = Jump if Not Parity (PF = 0) => número impar
```

```
es_par:
```

```
    ; (comentario) Imprimir "Par"
```

```
    mov ecx, msg_par
```

```
    jmp     print_result
```

```
es_impar:
```

```
    ; (comentario) Imprimir "Impar"
```

```
    mov ecx, msg_impar
```

```
    jmp     print_result
```

```
print_result:
```

```
    ; (Código para imprimir el resultado)
```

```
    ; (Código de salida aquí...)
```

4.

section .data

msg\_overflow db 'Overflow detectado', 0xA, 0

msg\_valido db 'Suma válida', 0xA, 0

section .bss

num1 resb 1

num2 resb 1

section .text

global \_start

\_start:

; call leer\_entero ; resultado en EAX

; mov ebx, eax ; si queremos usar la misma lectura dos veces

; call leer\_entero ; resultado en EAX (otro valor)

; mov ebx, eax

; Realizar la suma (EAX := EAX + EBX). ADD ajusta OF según overflow  
signado

add eax, ebx

; Saltar si hubo overflow signado (OF = 1)

jo hubo\_overflow

; Si no hubo overflow (OF = 0)

jno no\_hubo\_overflow

hubo\_overflow:

; p.ej.: imprimir "Overflow detectado"

; call imprimir\_overflow

mov ecx, msg\_overflow

jmp print\_result

no\_hubo\_overflow:

; p.ej.: imprimir "Suma válida"

; call imprimir\_resultado ; (EAX contiene la suma)

mov ecx, msg\_valido

jmp print\_result

print\_result:

; (Código para imprimir el resultado)

; (Código de salida aquí...)



5.

section .data

msg\_acarreo db 'Hubo acarreo', 0xA, 0

msg\_no\_acarreo db 'No hubo acarreo', 0xA, 0

section .bss

num1 resb 1

num2 resb 1

section .text

global \_start

\_start:

; call leer\_entero -> resultado en EAX

; mov ebx, eax ; (ejemplo)

; call leer\_entero -> resultado en EBX

; Realizar la suma (EAX := EAX + EBX). ADD ajusta CF si hay acarreo  
unsigned

add eax, ebx

jc hubo\_acarreo ; JC = Jump if Carry (CF = 1) -> hubo acarreo  
unsigned

jnc no\_hubo\_acarreo ; JNC = Jump if Not Carry (CF = 0) -> no hubo  
acarreo

hubo\_acarreo:

; p.ej.: imprimir "Acarreo detectado" y opcionalmente mostrar CF=1

mov ecx, msg\_overflow

jmp print\_result

no\_hubo\_acarreo:

; (comentario) Aquí manejar el caso: no se generó acarreo

; p.ej.: imprimir "No hay acarreo" y mostrar resultado (EAX)

mov ecx, msg\_valido

jmp print\_result

print\_result:

; (Código para imprimir el resultado)

; (Código de salida aquí...)

6.

section .data

msg\_acarreo db 'Hubo acarreo', 0xA, 0

msg\_no\_acarreo db 'No hubo acarreo', 0xA, 0

section .bss

; Entradas:

; EAX <- número1

; EBX <- número2

; ECX <- número3

; Salidas (convención aquí):

; EDX <- mínimo

; ESI <- máximo

section .text

global \_start

\_start:

; Inicializar min y max con el primer número (EAX)

mov edx, eax ; EDX := min provisional

mov esi, eax ; ESI := max provisional

cmp ebx, edx ; comparar n2 con min actual

jl poner\_min\_ebx ; si n2 < min -> actualizar min

; si no fue menor, comprobar si es mayor que max

cmp ebx, esi

jg poner\_max\_ebx ; si n2 > max -> actualizar max

jmp continuar\_para\_ecx

poner\_min\_ebx:

mov edx, ebx ; edx := ebx (nuevo min)

; no es necesario re-comparar con max porque si ebx < min\_anterior

; entonces no puede ser > max\_anterior (min<=max siempre)

jmp continuar\_para\_ecx

poner\_max\_ebx:

mov esi, ebx ; esi := ebx (nuevo max)

jmp continuar\_para\_ecx

continuar\_para\_ecx:

cmp ecx, edx ; comparar n3 con min actual

jl poner\_min\_ecx

```
    cmp    ecx, esi
    jg     poner_max_ecx
    jmp    print_result
```

```
poner_min_ecx:
    mov    edx, ecx    ; edx := ecx (nuevo min)
    jmp    print_result
```

```
poner_max_ecx:
    mov    esi, ecx    ; esi := ecx (nuevo max)
    jmp    print_result
```

```
print_result:
    ; Ahora EDX contiene el mínimo y ESI el máximo
    ; (comentario) Imprimir "Minimo = ", EDX
    ; (comentario) Imprimir "Maximo = ", ESI
    ; (Código de salida aquí...)
```

7.

```
section .bss
num1 resb 1
num2 resb 1
```

```
section .text
    global _start
```

```
_start:
```

```
    ; comparar A con B
    cmp    eax, ebx
```

```
    ; si A <= B (menor o igual, signado) -> ya están en orden -> saltar a fin
    jle    print_result
```

```
    ; Si llegamos aquí, A > B -> hay que intercambiar
```

```
    ; Variante con movs temporales:
```

```
    mov    ecx, eax    ; temp := A
```

```
    mov    eax, ebx    ; A := B
```

```
    mov    ebx, ecx    ; B := temp
```

```
    ; alternativa compacta: usar XCHG (si prefieres)
```

```
    ; xchg  eax, ebx
```

```
print_result:
```

```
    ; (comentario) imprimir/usar valores
```

```
    ; (Código de salida aquí...)
```

8.

```
section .bss
    num1 resb 1
```

```
section .text
    global _start
```

```
_start:
```

```
    mov     al, 0      ; AL := 0 ; primer valor a mostrar
```

```
    mov     ecx, 10    ; ECX := 10 (nº de iteraciones: 0..9)
```

```
loop_start:
```

```
    ; (comentario) imprimir AL (valor actual)
```

```
    ; call print_byte_in_al
```

```
    inc     al         ; incrementar para la siguiente iteración
```

```
    loop    loop_start ; ECX := ECX - 1 ; si ECX != 0 -> saltar
                        ; termina cuando ECX == 0
```

## TALLER 9

Del inciso e) ejemplo de carácter '1'

section .data

```
num1 db 1; Reserva bits para la varialbe
num2 db 0 ; Reserva bits para la varialbe
result db 0 ; Reserva bits para la varialbe
msg db 'Resultado: ', 0 ; Reserva bits para la varialbe
```

section .bss

```
buffer resb 4
```

section .text

```
global _start ; Inicia el programa, o bien el apartado '_start'
```

\_start:

```
mov al, [num1] ; Mueve el valor de num1 a 'al'
add al, [num2] ; Le suma el valor de num2 al valor en 'al'
mov [result], al ; Mueve el valor de 'al' a la variable 'result'
```

; Convertir el resultado a ASCII

movzx eax, byte [result] ; Toma el byte en [result] y pone su valor en EAX, pero asegura que los 24 bits altos de EAX queden en 0.

add eax, 48 ; Convertir el valor numérico en su correspondiente ASCII ('0' = 48)

```
mov [buffer], al ; Almacenar el carácter ASCII en el buffer
```

```
mov eax, 4 ; Mueve el valor 4 a eax
mov ebx, 1 ; Mueve el valor 1 a ebx
mov ecx, msg ; Mueve el valor msg a ecx
mov edx, 11 ; Mueve el valor 11 a edx
int 0x80 ; invoca la interrupción de software
```

```
mov eax, 4 ; Mueve el valor 4 a eax
mov ebx, 1 ; Mueve el valor 4 a ebx
mov ecx, buffer ; Mueve el valor buffer a ecx
mov edx, 1 ; Mueve el valor 1 a edx
int 0x80 ; invoca la interrupción de software
```

```
mov eax, 1 ; Mueve el valor 1 a eax
xor ebx, ebx ; realiza una operación XOR a ebx y ebx
int 0x80 ; invoca la interrupción de software
```

Del inciso g)

1. Modo inmediato

; archivo: inmediato.asm

section .data

num1 db 5

num2 db 11

result db 0

msg db 'Resultado: ', 0

section .bss

buffer resb 4

section .text

global \_start

\_start:

; --- realizar la suma (igual que tu código original) ---

mov al, [num1]

add al, [num2]

mov [result], al

mov byte [buffer], 64 ; <-- aquí está el modo inmediato (memoria <- inmediato)

; Imprimir "Resultado: "

mov eax, 4

mov ebx, 1

mov ecx, msg

mov edx, 11

int 0x80

; Imprimir el carácter en buffer (1 byte)

mov eax, 4

mov ebx, 1

mov ecx, buffer

mov edx, 1

int 0x80

; exit

mov eax, 1

xor ebx, ebx

int 0x80

## 2. Modo indirecto

; archivo: indirecto.asm

section .data

num1 db 5

num2 db 11

result db 0

msg db 'Resultado: ', 0

section .bss

buffer resb 4

section .text

global \_start

\_start:

; --- realizar la suma ---

mov al, [num1]

add al, [num2]

mov [result], al

movzx eax, byte [result] ; EAX = resultado (0..255)

add eax, 48 ; convertir a ASCII ('0' = 48)

mov al, al ; AL contiene ahora 64 (si result=16)

; --- direccionamiento indirecto:

lea esi, [buffer] ; ESI := &buffer

mov [esi], al ; memoria apuntada por ESI <- AL (indirecto)

; Imprimir "Resultado: "

mov eax, 4

mov ebx, 1

mov ecx, msg

mov edx, 11

int 0x80

; Imprimir el carácter en buffer (1 byte)

mov eax, 4

mov ebx, 1

mov ecx, buffer

mov edx, 1

int 0x80



```
; exit  
mov    eax, 1  
xor     ebx, ebx  
int     0x80
```

## TALLER 10

Enciso c)

; ror\_to\_g.asm

section .data

char db 0

newline db 10

section .text

global \_start

\_start:

mov al, 0xCE

; Rotar a la derecha 1 -> 0110 0111b = 0x67 ('g')

ror al, 1

mov [char], al

mov eax, 4

mov ebx, 1

mov ecx, char

mov edx, 1

int 0x80

; newline

mov eax, 4

mov ebx, 1

mov ecx, newline

mov edx, 1

int 0x80

; exit

mov eax, 1

xor ebx, ebx

int 0x80

```
; rol_to_g.asm
section .data
    char db 0
    newline db 10

section .text
    global _start

_start:
    mov al, 0xB3

    ; Rotar a la izquierda 1 -> 0110 0111b = 0x67 ('g')
    rol al, 1

    mov [char], al

    mov eax, 4
    mov ebx, 1
    mov ecx, char
    mov edx, 1
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80

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

```

; shl_to_g.asm
section .data
    char db 0
    newline db 10

section .text
    global _start

_start:
    mov al, 0x33

    ; Desplazar a la izquierda 1 -> 0110 0110b = 0x66
    shl al, 1

    ; Ajuste final -> 0x66 + 1 = 0x67 ('g')
    add al, 1

    mov [char], al

    mov eax, 4
    mov ebx, 1
    mov ecx, char
    mov edx, 1
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80

    mov eax, 1
    xor ebx, ebx
    int 0x80

```

```
; shr_to_g.asm
section .data
    char db 0
    newline db 10

section .text
    global _start

_start:
    mov al, 0xCE

    ; Desplazar a la derecha 1 -> 0110 0111b = 0x67 ('g')
    shr al, 1

    mov [char], al

    mov eax, 4
    mov ebx, 1
    mov ecx, char
    mov edx, 1
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80

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

## TALLER 11

```
section .data
    msg db "Resultado: ", 0
    len equ $ - msg

    newline db 10, 0
    lenNL equ $ - newline

; Nuevo mensaje para el error
    error_msg db "Error: Division por cero no permitida!", 10, 0
    len_error equ $ - error_msg

section .bss
    resultado resb 1

section .text
    global _start

_start:
    ; =====
    ; Números hardcoded
    ; =====
    mov al, '8'          ; primer número (ASCII)
    sub al, '0'          ; convertir a entero (8)

    mov bl, '0'          ; Probar con '0' para error
    sub bl, '0'          ; convertir a entero (0)

    ; =====
    ; Comprobación de Division por Cero
    ; =====
    cmp bl, 0            ; Compara el divisor (BL) con 0
    je error_div_cero    ; Salta a 'error_div_cero' si BL es igual a 0 (Jump if
Zero)

    ; =====
    ; División AL / BL (Normal)
    ; =====
    xor ah, ah           ; limpiar AH para div
    div bl               ; resultado en AL

    ; =====
```

```

; Convertir resultado a ASCII
; =====
add al, '0'
mov [resultado], al

; =====
; Imprimir "Resultado: "
; =====
mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, len
int 0x80

; =====
; Imprimir el resultado
; =====
mov eax, 4
mov ebx, 1
mov ecx, resultado
mov edx, 1
int 0x80

; =====
; Imprimir salto de línea
; =====
mov eax, 4
mov ebx, 1
mov ecx, newline
mov edx, lenNL
int 0x80

jmp salir          ; Salta al final para evitar el código de error

; ---
; Manejo de Error de División por Cero
; ---
error_div_cero:
; Imprimir mensaje de error
mov eax, 4
mov ebx, 1
mov ecx, error_msg
mov edx, len_error

```

```
int 0x80

; ---
; Salida del Programa
; ---
salir:
; =====
; Salir
; =====
mov eax, 1
xor ebx, ebx
int 0x80
```