Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Estudiante:

Moreno Calderón Troy Leonardo

Grupo: 932

Organización de Computadoras

Jonatan Crespo Ragland

Repositorio de Codigos

TALLER 5

6. De acuerdo al código ensamblador anexo:

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

i. A

```
section .data
    num1 db 9        ; Primera variable (entre 1 y 3)
    num2 db 8        ; Segunda variable (entre 1 y 3)
    result db 0      ; Espacio para almacenar el resultado convertido a ASCII

section .text
    global _start

_start:
    mov al, [num1]    ; Cargar num1 en AL
    add al, [num2]    ; Sumar num2 a AL
    add al, '0'       ; Convertir el resultado a ASCII

    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

    ; Salir del programa
    mov eax, 1        ; syscall: sys_exit
    xor ebx, ebx      ; Código de salida 0
```

```
    int 0x80         ; Llamada al sistema
```

ii. :

```
section .data
    num1 db 5        ; Primera variable (entre 1 y 3)
    num2 db 5        ; Segunda variable (entre 1 y 3)
    result db 0      ; Espacio para almacenar el resultado convertido a ASCII

section .text
    global _start

_start:
    mov al, [num1]    ; Cargar num1 en AL
    add al, [num2]    ; Sumar num2 a AL
    add al, '0'       ; Convertir el resultado a ASCII

    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

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

iii. =

```
section .data
    num1 db 7        ; Primera variable (entre 1 y 3)
    num2 db 6        ; Segunda variable (entre 1 y 3)
    result db 0      ; Espacio para almacenar el resultado convertido a ASCII

section .text
    global _start

_start:
    mov al, [num1]    ; Cargar num1 en AL
    add al, [num2]    ; Sumar num2 a AL
    add al, '0'       ; Convertir el resultado a ASCII

    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

    ; Salir del programa
```

```
    mov eax, 1        ; syscall: sys_exit
    xor ebx, ebx      ; Código de salida 0
    int 0x80          ; Llamada al sistema
```

iv. ? 7+8

```
section .data
    num1 db 7         ; Primera variable (entre 1 y 3)
    num2 db 8         ; Segunda variable (entre 1 y 3)
    result db 0       ; Espacio para almacenar el resultado convertido a ASCII

section .text
    global _start

_start:
    mov al, [num1]    ; Cargar num1 en AL
    add al, [num2]    ; Sumar num2 a AL
    add al, '0'       ; Convertir el resultado a ASCII

    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

    ; Salir del programa
    mov eax, 1        ; syscall: sys_exit
```

```
    xor ebx, ebx      ; Código de salida 0
    int 0x80          ; Llamada al sistema
```

v. _ 24 + 23

```
section .data
    num1 db 24        ; Primera variable (entre 1 y 3)
    num2 db 23        ; Segunda variable (entre 1 y 3)
    result db 0       ; Espacio para almacenar el resultado convertido a ASCII

section .text
    global _start

_start:
    mov al, [num1]    ; Cargar num1 en AL
    add al, [num2]    ; Sumar num2 a AL
    add al, '0'       ; Convertir el resultado a ASCII

    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

    ; Salir del programa
    mov eax, 1        ; syscall: sys_exit
```

```
    xor ebx, ebx      ; Código de salida 0
    int 0x80          ; Llamada al sistema
```

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

i. B 67-1

```
section .data
    num1 db 67      ; Primera variable (entre 1 y 3)
    num2 db 1       ; Segunda variable (entre 1 y 3)
    result db 0     ; Espacio para almacenar el resultado convertido a ASCII
section .text
    global _start
_start:
    mov al, [num1]    ; Cargar num1 en AL
    sub al, [num2]    ; resta num2 a AL
    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

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

ii. x

```
section .data
    num1 db 121       ; Primera variable (entre 1 y 3)
    num2 db 1         ; Segunda variable (entre 1 y 3)
    result db 0       ; Espacio para almacenar el resultado convertido a ASCII
section .text
    global _start
_start:
    mov al, [num1]    ; Cargar num1 en AL
    sub al, [num2]    ; resta num2 a AL
    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

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

iii. +

```
section .data
    num1 db 44       ; Primera variable (entre 1 y 3)
    num2 db 1        ; Segunda variable (entre 1 y 3)
```

```
    result db 0       ; Espacio para almacenar el resultado convertido a ASCII
section .text
    global _start
_start:
    mov al, [num1]    ; Cargar num1 en AL
    sub al, [num2]    ; resta num2 a AL
    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

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

iv. ‘

```
section .data
    num1 db 40       ; Primera variable (entre 1 y 3)
    num2 db 1        ; Segunda variable (entre 1 y 3)
    result db 0       ; Espacio para almacenar el resultado convertido a ASCII
section .text
    global _start
_start:
    mov al, [num1]    ; Cargar num1 en AL
```

```
    sub al, [num2]    ; resta num2 a AL
    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, result   ; Dirección del resultado
    mov edx, 1        ; Longitud del resultado
    int 0x80          ; Llamada al sistema

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

v. {

```
section .data
    num1 db 124       ; Primera variable (entre 1 y 3)
    num2 db 1         ; Segunda variable (entre 1 y 3)
    result db 0       ; Espacio para almacenar el resultado convertido a ASCII
section .text
    global _start
_start:
    mov al, [num1]    ; Cargar num1 en AL
    sub al, [num2]    ; resta num2 a AL
    mov [result], al  ; Guardar el carácter ASCII en 'result'

    ; Imprimir el número (un solo dígito)
```

```
mov eax, 4        ; syscall: sys_write
mov ebx, 1        ; file descriptor: stdout
mov ecx, result   ; Dirección del resultado
mov edx, 1        ; Longitud del resultado
int 0x80          ; Llamada al sistema

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

TALLER 6
SALTO CONDICIONAL

```
section .data
    num1 db 3        ; Primera variable (entre 1 y 3)
    num2 db 2        ; Segunda variable (entre 1 y 3)
    result db 0      ; Espacio para almacenar el resultado convertido a ASCII

section .text
    global _start

_start:
    ; Comparar num1 y num2 antes de sumar
    mov al, [num1]      ; Cargar num1 en AL
    cmp al, [num2]      ; Comparar AL con num2
    je iguales          ; Si son iguales, saltar a la etiqueta "iguales"

    ; Si no son iguales, se ejecuta este bloque
    add al, [num2]      ; Sumar num2 a AL
    add al, '0'         ; Convertir el resultado a ASCII
    mov [result], al    ; Guardar el carácter ASCII en 'result'
    jmp imprimir        ; Saltar a imprimir resultado

iguales:
    ; Si num1 y num2 son iguales, poner el carácter '0' como resultado
    mov al, '0'
```

```
    mov [result], al

imprimir:
    ; Mostrar el resultado en pantalla
    mov eax, 4          ; syscall: sys_write
    mov ebx, 1          ; file descriptor: stdout
    mov ecx, result     ; Dirección del resultado
    mov edx, 1          ; Longitud del resultado
    int 0x80            ; Llamada al sistema

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

TALLER 8

Instrucciones de control de flujo

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.

```
; Comparador de Números

mov eax, [num1]
mov ebx, [num2]

cmp eax, ebx
je iguales    ;Salta si los operandos son iguales
jg mayor      ;Salta si el primer operando es mayor que el segundo
jl menor      ;Salta si el primer operando es menor que el segundo

iguales:
   ; imprimir "Son iguales"
   jmp verificar

mayor:
   ; imprimir "El primero es mayor"
   jmp verificar

menor:
   ; imprimir "El segundo es mayor"
```

```
    jmp verificar

verificar:
    test eax, eax
    js negativo
    test ebx, ebx
    js negativo
    jmp fin

negativo:
    ; imprimir "Alguno es negativo"
    jmp fin

fin:
    ; terminar
```

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

```
; Clasificación de Números

mov eax, [num]      ; cargar número en eax

cmp eax, 0
je es_cero          ;Salta si los operandos son iguales (ZF=1)
jg es_positivo      ;Salta si el primer operando es mayor que el segundo
jl es_negativo      ;Salta si el primer operando es menor que el segundo

es_cero:
    ; imprimir "El número es cero"
```

```
    jmp fin

es_positivo:
    ; imprimir "El número es positivo"
    jmp fin

es_negativo:
    ; imprimir "El número es negativo"
    jmp fin

fin:
    ; terminar
```

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

```
; Par o Impar

mov eax, [num]      ; cargar número en eax

test eax, eax       ; ajusta banderas según el número
jp es_par           ;Salta si el resultado tiene paridad par (PF=1)

; Si no saltó con JP, entonces es impar
; imprimir "El número es impar"
jmp fin

es_par:
    ; imprimir "El número es par"
    jmp fin
```

```
fin:
   ; terminar
```

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.

```
; Simulación de Overflow

mov eax, [num1]      ; cargar primer número
mov ebx, [num2]      ; cargar segundo número

add eax, ebx         ; sumar num1 + num2
jo hay_overflow      ;Salta si hubo desbordamiento (Overflow Flag = 1)

; Si no hubo overflow no salta e imprime lo de abajo
; imprimir "La suma es correcta"
jmp fin

hay_overflow:
   ; imprimir "Ocurrió overflow en la suma"
   jmp fin

fin:
   ; terminar
```

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.

```
; Simulación de Acarreo
```

```
mov eax, [num1]      ; cargar primer número
mov ebx, [num2]      ; cargar segundo número

add eax, ebx         ; sumar num1 + num2
jc hay_carry         ;Salta si hubo acarreo (Carry Flag = 1)

; Si no hubo acarreo
; imprimir "No se generó acarreo"
jmp fin

hay_carry:
   ; imprimir "Se generó un acarreo en la suma"
   jmp fin

fin:
   ; terminar
```

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

```
; Mínimo y Máximo de Tres Números

mov eax, [num1]      ; cargar primer número
mov ebx, [num2]      ; cargar segundo número
mov ecx, [num3]      ; cargar tercer número

; --- Encontrar el máximo ---
cmp eax, ebx
jge cmp_eax_ecx      ;Salta si eax >= ebx
mov eax, ebx         ; si no, máximo provisional = ebx
```

```
cmp_eax_ecx:
cmp eax, ecx
jge max_listo        ;Salta si eax >= ecx
mov eax, ecx         ; si no, máximo = ecx

max_listo:
   ; imprimir "Máximo encontrado en eax"

; --- Encontrar el mínimo ---
mov edx, [num1]      ; cargar primer número en edx
cmp edx, ebx
jle cmp_edx_ecx      ;Salta si edx <= ebx
mov edx, ebx         ; si no, mínimo provisional = ebx

cmp_edx_ecx:
cmp edx, ecx
jle min_listo        ;Salta si edx <= ecx
mov edx, ecx         ; si no, mínimo = ecx

min_listo:
   ; imprimir "Mínimo encontrado en edx"

fin:
   ; terminar
```

7. Ordenamiento de Dos Números

Leer dos números e intercambiarlos si no están en orden ascendente usando solo saltos condicionales.

```
; Ordenamiento de Dos Números
```

```
mov eax, [num1]      ; cargar primer número
mov ebx, [num2]      ; cargar segundo número

cmp eax, ebx
jle orden_ok         ;Salta si el primer operando es menor o igual al segundo

; Si no están en orden ascendente, intercambiarlos
mov ecx, eax         ; usar registro auxiliar
mov eax, ebx
mov ebx, ecx

orden_ok:
   ; ahora eax <= ebx
   ; imprimir "Números ordenados en forma ascendente"

fin:
   ; terminar
```

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

```
; Ciclo de Conteo sin Comparaciones (0 a 9)

mov eax, 0           ; inicializar contador en 0

ciclo:
   ; imprimir valor de eax

   inc eax           ; incrementar contador
   cmp eax, 10
```

```
    jne ciclo          ;Salta si los operandos no son iguales (ZF=0)

fin:
    ; terminar
```

TALLER 9

MODOS DE DIRECCIONAMIENTO

- Un ejemplo del punto e.

```
section .data
    num1 db 36              ; primer número
    num2 db 0               ; segundo número
    result db 0             ; variable para almacenar el resultado
    msg db 'Resultado: ',0 ; mensaje a imprimir antes del resultado

section .bss
    buffer resb 4           ; espacio reservado para almacenar el resultado convertido a ASCII

section .text
    global _start

_start:

    mov al, [num1]          ; cargar num1 en AL
    add al, [num2]          ; sumar num2 a AL
    mov [result], al        ; guardar el resultado en la variable result

    movzx eax, byte [result] ; cargar el resultado en EAX con extensión cero
    add eax, 0              ; convertir valor numérico a carácter ASCII
    mov [buffer], al        ; almacenar el carácter ASCII en buffer
```

```
    mov eax, 4            ; syscall write
    mov ebx, 1            ; descriptor de archivo (stdout)
    mov ecx, msg          ; dirección del mensaje
    mov edx, 11           ; longitud del mensaje
    int 0x80              ; llamada al sistema

    mov eax, 4            ; syscall write
    mov ebx, 1            ; descriptor de archivo (stdout)
    mov ecx, buffer       ; dirección del buffer con el resultado ASCII
    mov edx, 1            ; longitud (1 carácter)
    int 0x80              ; llamada al sistema

    mov eax, 1            ; syscall exit
    xor ebx, ebx          ; código de salida 0
    int 0x80
```

g. Utilizando de nuevo el código de prueba original, modifica el código para que ahora utilice el modo de direccionamiento inmediato e indirecto (en programas separados) para que imprima el carácter '@'. Documenta tus resultados.

- Modo de direccionamiento inmediato:

```
section .data

section .text
    global _start

_start:
    ; imprimir '@' usando inmediato
    mov eax, 4          ; syscall write
```

```
    mov ebx, 1          ; descriptor stdout
    mov ecx, char       ; dirección del carácter
    mov edx, 1          ; longitud
    int 0x80

    ; terminar
    mov eax, 1
    xor ebx, ebx
    int 0x80

section .data
char db '@'             ; carácter definido directamente
```

- Modo de direccionamiento indirecto:

```
section .data
    char db '@'         ; carácter almacenado en memoria

section .text
    global _start

_start:
    ; cargar carácter desde memoria (indirecto)
    mov al, [char]      ; acceder al contenido de la dirección de 'char'
    mov [buffer], al    ; guardar en buffer

    ; imprimir '@'
    mov eax, 4          ; syscall write
    mov ebx, 1          ; descriptor stdout
    mov ecx, buffer     ; dirección del buffer
```

```
    mov edx, 1          ; longitud
    int 0x80

    ; terminar
    mov eax, 1
    xor ebx, ebx
    int 0x80

section .bss
    buffer resb 1       ; espacio para almacenar el carácter
```

TALLER 10

DESPLAZAMIENTO DE BITS

Inciso c. Imprimir ‘g’

```
section .data
    char db 0
    newline db 10

section .text
    global _start

_start:
    ; ---- c) Imprimir 'g' ----
    mov al, 206        ; valor base

    shr al, 1          ; 206 >> 1 = 103 ('g')
    rol al, 1          ; rota izquierda → 206
    ror al, 1          ; rota derecha → 103 ('g')
    shl al, 0

    ; Guardar en char
    mov [char], al

    ; Escribir carácter en consola
    mov eax, 4         ; syscall write
    mov ebx, 1         ; stdout
    mov ecx, char
    mov edx, 1         ; 1 byte
    int 0x80
```

```
; Salto de línea
mov eax, 4
mov ebx, 1
mov ecx, newline
mov edx, 1
int 0x80

; Salir
mov eax, 1          ; syscall exit
xor ebx, ebx        ; código de salida 0
int 0x80
```

TALLER 11
INTERRUPCIONES

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

    newline db 10, 0
    lenNL equ $ - newline

    ; --- NUEVO: Mensaje para la interrupción simulada ---
    err_msg db "Error: Division por Cero!", 10, 0
    lenErr equ $ - err_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)

    ; CAMBIO: Puse '0' aquí para probar la detección de error.
    mov bl, '0'
    sub bl, '0'      ; convertir a entero
```

```
; ============================
; VALIDACIÓN (Simulación de Interrupción)
; ============================
cmp bl, 0        ; Comparamos el divisor con 0
je error_div_zero ; Si es igual (Jump Equal), saltamos a la rutina de error

; ============================
; División AL / BL (Solo se ejecuta si BL != 0)
; ============================
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 numérico
    ; ============================
    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

    ; Saltamos al final para no ejecutar el código de error por accidente
    jmp salir

; ========================================================
; RUTINA DE SERVICIO SIMULADA (ISR para División por Cero)
; ========================================================
error_div_zero:
    mov eax, 4
    mov ebx, 1
    mov ecx, err_msg
    mov edx, lenErr
    int 0x80
```

```
    ; Después de mostrar el error, continuamos hacia la salida


; ============================
; Salir
; ============================
salir:
    mov eax, 1
    xor ebx, ebx
    int 0x80
```

TALLER 12

Macros y estructuras de datos

```
; ============================================================
; 1. DEFINICIÓN DE MACRO
; ============================================================
; Macro: OPERAR_TRIPLETA
; Descripción: Recibe un puntero a una estructura de 3 números
;              y suma cada uno de sus elementos.
; Parámetro %1: Registro que contiene la dirección base de la estructura.
; Salida: El resultado de la suma queda en el registro AX.
; ============================================================
%macro OPERAR_TRIPLETA 1
    xor ax, ax          ; Limpiamos AX (AX = 0) para empezar la suma

    ; Accedemos a los elementos usando desplazamientos (offsets)
    add ax, [%1]        ; Sumar el primer elemento (Offset 0)
    add ax, [%1 + 2]    ; Sumar el segundo elemento (Offset 2 bytes)
    add ax, [%1 + 4]    ; Sumar el tercer elemento (Offset 4 bytes)
%endmacro

section .data
    ; ============================================================
    ; 2. ESTRUCTURA DE DATOS (X / X / X)
    ; ============================================================
    ; Definimos 3 palabras (words) de 16 bits contiguas.
    ; Valores: 10, 20, 50. (La suma esperada es 80)
    datos_x dw 10, 20, 50

section .text
```

```
    global _start

_start:
    ; ============================================================
    ; 3. PROGRAMA PRINCIPAL
    ; ============================================================

    ; Cargamos la dirección de memoria de nuestra estructura en EBX
    mov ebx, datos_x

    ; --- LLAMADA A LA MACRO ---
    ; Le pasamos el puntero (ebx) para que realice la operación
    OPERAR_TRIPLETA ebx

    ; En este punto, el registro AX contiene el valor 80.
    ; (Aquí iría el código para imprimir AX si fuera necesario)

    ; ============================================================
    ; 4. SALIDA
    ; ============================================================
    mov eax, 1          ; Syscall: sys_exit
    xor ebx, ebx        ; Retorno: 0
    int 0x80            ; Interrupción del kernel
```

TRABAJO EN CLASE
MACROS

```
; --- Macro para imprimir un entero ---
%macro print_int 1
    mov eax, 4        ; syscall write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, %1       ; dirección del dato a imprimir
    mov edx, 4        ; tamaño: 4 bytes (entero de 32 bits)
    int 0x80          ; llamada al sistema
%endmacro

section .data
array dd 1, 2, 3, 4, 5    ; arreglo de 5 enteros

section .text
global _start

_start:
    mov ecx, 0        ; índice del arreglo
    mov eax, 0        ; acumulador para la suma

bucle:
    mov ebx, [array + ecx*4] ; cargar array[ecx] en ebx
    add eax, ebx             ; sumar al acumulador
    inc ecx                  ; siguiente índice
    cmp ecx, 5               ; ¿ya sumamos los 5 elementos?
    jl bucle                 ; si ecx < 5, repetir
```

```
    print_int eax     ; imprimir resultado

    mov eax, 1        ; syscall exit
    xor ebx, ebx      ; código de salida 0
    int 0x80          ; terminar programa
mov ecx, 0        ; índice del arreglo, empieza en 0
mov eax, 0        ; acumulador para la suma

bucle:
    mov ebx, [array + ecx*4]    ; cargar array[ecx] en ebx
    add eax, ebx                ; sumar al acumulador
    inc ecx                     ; avanzar al siguiente índice
    cmp ecx, 5                  ; comparar con 5 (tamaño del arreglo)
    jl bucle                    ; si ecx < 5, repetir el bucle

print_int eax     ; imprimir el resultado

mov eax, 1        ; syscall exit
xor ebx, ebx      ; código de salida 0
int 0x80          ; terminar programa
```

TRABAJO EN CLASE
INTERRUPCIONES

Generar un snippet de código en ensamblador que simula el uso de un bloque de Try/Catch en ensamblador, utilizando saltos condicionales/incondicionales (No tiene que compilar).

```
    .DATA
ERROR_FLAG: .BYTE 0   ; Flag para indicar si ocurrió un error (0 = No, 1 = Sí)
EXIT_CODE:  .DWORD 0  ; Código de salida/retorno de la función

; ************************************
; Bloque TRY
; ************************************

INICIO_TRY:
    MOV R1, 10 ; Cargar un valor en R1
    MOV R2, 0  ; Cargar un valor en R2 (Simula una posible división por cero)

    CMP R2, 0                ; Comparar R2 con 0
    JE  MANEJAR_ERROR    ; Si es igual a cero, saltar al manejador

    DIV R1, R2               ; Operación riesgosa (e.g., división)

    JMP FIN_TRY_CATCH      ; Salto incondicional al final

; ************************************
; Bloque CATCH
; ************************************

MANEJAR_ERROR:
```

```
    ; Simula la lógica de 'catch'
    MOV ERROR_FLAG, 1       ; Establecer el flag de error
    MOV R3, 99              ; Cargar código de error en R3

    PRINT_STRING "¡Error capturado en la operación!"
    PRINT_REGISTER R3       ; Mostrar el código de error


; ************************************
; Finalización
; ************************************

FIN_TRY_CATCH:
    CMP ERROR_FLAG, 1       ; Verificar si hubo un error
    JNE TRY_EXIT_SUCCESS    ; Si no hubo error, saltar al éxito

    ; Si hubo error (Error Flag = 1):
    MOV EXIT_CODE, 1        ; Establecer código de salida de fallo
    PRINT_STRING "El programa finalizó con un error."
    JMP PROGRAM_EXIT

TRY_EXIT_SUCCESS:
    ; Si no hubo error (Error Flag = 0):
    MOV EXIT_CODE, 0        ; Establecer código de salida de éxito
    PRINT_STRING "Operación exitosa."

PROGRAM_EXIT:
    HALT
```

Realizar un programa en C que utilice las funciones setjmp y longjmp para simular el manejo de excepciones.

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf buffer;

void validarNumero(int n);

int main()
{
   int numero;

      if (setjmp(buffer) == 0)
      {
      printf("Ingrese un numero positivo: ");
      scanf("%d", &numero);

      validarNumero(numero);

      printf("Numero valido....\n");
      }
      else
      {
                  printf("Se capturo una excepcion: numero invalido.\n");
      }

   printf("Programa terminado.\n");
   return 0;
```

```
}

void validarNumero(int n)
{
   if (n <= 0)
   {
      printf("Error: el numero debe ser positivo.\n");
      longjmp(buffer, 1);
    }
}
```

5. Realizar un snippet de código que utilice los breakpoints en su IDE (En el lenguaje de su elección).

```
#include <stdio.h>

int puedeComprar(int saldo, int precio);

int main()
{
   int saldo_usuario = 120;
   int precio_producto = 150;

   // <-- Breakpoint 1: revisar valores antes de llamar la función
   int resultado = puedeComprar(saldo_usuario, precio_producto);

   // <-- Breakpoint 2: revisar el resultado devuelto antes de imprimir
   if (resultado == 1)
   {
      printf("Compra realizada con exito.\n");
   }
```

```
    else
    {
        printf("Saldo insuficiente para realizar la compra.\n");
    }

    return 0;
}

int puedeComprar(int saldo, int precio)
{
    // <-- Breakpoint 3: ya dentro de la función, revisar saldo/precio
    if (saldo >= precio)
    {
        // <-- Breakpoint 4: condición verdadera
        return 1; // Compra permitida
    }
    else
    {
        // <-- Breakpoint 5: condición falsa
        return 0; // Compra rechazada
    }
}
```