Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Ángel Gabriel Zúñiga López

Prof. Jonatan Crespo Ragland

Práctica Repositorios

Organización de Computadoras

3er. Semestre

Ensenada B.C a 28 de Noviembre del 2024

Taller 5

```
section .data
msg db 'imprimir imput del teclado: ';Input: &#39;, 0 ; Mensaje que se mostrarsn antes de la entrada

section .bss
input resb 1 ; Espacio para almacenar el carsncter ingresado
sum resb 1 ; Espacio para almacenar la suma

section .text
global _start

_start:
; Mostrar mensaje en consola
mov eax, 4
mov ebx, 1
mov ecx, msg ; direccisnn del mensaje
mov edx, 28 ; longitud del mensaje
int 0x80

; Leer un carsncter desde el teclado

mov eax, 3
mov ebx, 0
mov ecx, input ; direccisnn para almacenar la entrada
mov edx, 100 ; leer 1 byte (1 carsncter)
int 0x80

; Mostrar el carsncter ingresado
mov eax, 4 ; syscall nsnmero 4 es write (sys_write)
mov ebx, 1 ; descriptor de archivo 1 es stdout
mov ecx, input ; direccisnn del carsncter
mov edx, 100 ; longitud del carsncter
int 0x80 ; llamada al sistema

; Calcular la suma de los caracteres
mov al, [input]
add al, [input]
mov [sum], al ; almacenar la suma en la variable sum

; Mostrar la suma
mov eax, 4

mov ebx, 1
mov ecx, sum ; direccisnn de la suma
mov edx, 1 ; longitud de la suma
int 0x80

; Terminar el programa
mov eax, 1
xor ebx, ebx ; csndigo de salida 0
int 0x80
```

STDIN

organizacion

Output:

```
imprimir imput del teclado: organizacion
```

Taller 7

```
section .data

num1 db 5 ;se modifica el numero y al sumarlo con num2 se puede buscar el numero ascii que desea
num2 db 11;esto tomando en cuenta que se comienza en el 0 o el "48" en ascii como se declara en add eax, 48
result db 0 ;define una variable para almacenar el resultado
msg db 'Resultado:', 0

section .bss
buffer resb 4;reserva un espacio de 4 bytes en el buffer para el resultado

section .text
global _start

_start:

mov al, [num1]
add al, [num2]
mov [result], al
;convertir el resultado a ASCII
movzx eax, byte [result]
add eax, 48 ;convertir el valor numerico en su correspondiente ASCII ('0'= 48)
mov [buffer], al ; almacenar el caracter ASCII en el buffer

mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, 11
int 0x80

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

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

```
section .data
    num1 db 'A'           ; Guardar el carácter 'A'
    num2 db '\'           ; Guardar el carácter '\'
    num3 db '$'           ; Guardar el carácter '$'
    num4 db '&'           ; Guardar el carácter '&'
    num5 db '1'           ; Guardar el carácter '1'
    msg db 'Resultado: ', 0 ; Mensaje que se imprimirá
section .bss
    buffer resb 5         ; Reservar 5 bytes en el buffer para los caracteres
section .text
global _start
_start:
    ; Cargar y almacenar cada carácter en el buffer

    mov al, [num1]        ; Cargar 'A' en el registro AL
    mov [buffer], al      ; Almacenar 'A' en el primer byte del buffer

    mov al, [num2]        ; Cargar '\' en el registro AL
    mov [buffer+1], al    ; Almacenar '\' en el segundo byte del buffer

    mov al, [num3]        ; Cargar '$' en el registro AL
    mov [buffer+2], al    ; Almacenar '$' en el tercer byte del buffer

    mov al, [num4]        ; Cargar '&' en el registro AL
    mov [buffer+3], al    ; Almacenar '&' en el cuarto byte del buffer

    mov al, [num5]        ; Cargar '1' en el registro AL
    mov [buffer+4], al    ; Almacenar '1' en el quinto byte del buffer

    ; Imprimir el mensaje 'Resultado: '
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, 11
    int 0x80

    ; Imprimir el buffer con los caracteres A\ $&1
    mov eax, 4
    mov ebx, 1
    mov ecx, buffer
    mov edx, 5
    int 0x80

    ; Terminar el programa
    mov eax, 1
    xor ebx, ebx
    int 0x80
```

```
section .data
    msg db 'Resultado: ', 0   ; Mensaje que se imprimirá
section .bss
    buffer resb 1             ; Reservar un byte en el buffer para el carácter
section .text
global _start
_start:
    ; Usar direccionamiento inmediato para cargar el carácter '@'
    mov al, '@'               ; Cargar directamente el carácter '@' en AL
    mov [buffer], al          ; Almacenar el carácter '@' en el buffer

    ; Imprimir el mensaje 'Resultado: '
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, 11               ; Longitud del mensaje
    int 0x80                  ; Interrupción para realizar la llamada al sistema

    ; Imprimir el carácter '@'
    mov eax, 4
    mov ebx, 1
    mov ecx, buffer           ; Dirección del buffer que contiene '@'
    mov edx, 1                ; Imprimir un solo carácter (1 byte)
    int 0x80                  ; Interrupción para realizar la llamada al sistema

    ; Terminar el programa
    mov eax, 1
    xor ebx, ebx
    int 0x80                  ; Interrupción para realizar la llamada al sistema
```

```
section .data
    char_at db '@'            ; Almacenar el carácter '@' en la memoria
    msg db 'Resultado: ', 0   ; Mensaje que se imprimirá
section .bss
    buffer resb 1             ; Reservar un byte en el buffer para el carácter
section .text
global _start
_start:
    ; Usar direccionamiento indirecto para cargar el carácter '@'
    mov esi, char_at          ; Cargar la dirección de la variable 'char_at' en ESI
    mov al, [esi]             ; Cargar el valor de la dirección almacenada en ESI en AL
    mov [buffer], al          ; Almacenar el carácter '@' en el buffer

    ; Imprimir el mensaje 'Resultado: '
    mov eax, 4
    mov ebx, 1
    mov ecx, msg              ; Dirección del mensaje
    mov edx, 11               ; Longitud del mensaje
    int 0x80

    ; Imprimir el carácter '@'
    mov eax, 4
    mov ebx, 1
    mov ecx, buffer           ; Dirección del buffer que contiene '@'
    mov edx, 1                ; Imprimir un solo carácter (1 byte)
    int 0x80

    ; Terminar el programa
    mov eax, 1
    xor ebx, ebx
    int 0x80                  ; Interrupción para realizar la llamada al sistema
```

Taller 8

```
section .data
    msg db 'Resultado: ', 0
    newline db 0xA

section .bss
    res resb 4                  ; Espacio para el resultado

section .text
    global _start

_start:

    ; Instrucciones aritméticas
    mov eax, 10     ;se coloca los numeros a sumar
    mov ebx, 5      ;se coloca los numeros a sumar
    add eax, ebx      ;se suma eax+ebx, se almacena en eax

    ; Instrucción lógica (AND)
    and eax, 0xF      ; Realiza AND bit a bit con 0xF (15 en deci

    ; Instrucciones de manipulación de bits
    shl eax, 1       ;aqui la sumatoria se por 2

    ; Guardar el resultado en la sección .bss
    mov [res], eax   ;almacena el valor de la sumatoria en res

    ; Llamar a la rutina para imprimir el resultado
    mov eax, 4          ; Syscall para escribir
    mov ebx, 1          ; Usar la salida estándar (pantalla)
    mov ecx, msg        ; Direccion del mensaje a imprimir
    mov edx, 11         ; Longitud del mensaje
    int 0x80            ; Interrupción para imprimir el mensaje

    ; Imprimir el número (resultado almacenado en 'res')
    mov eax, [res]      ; Cargar el resultado en EAX
    add eax, '0'        ; Convertir el número en carácter (ASCII)
    mov [res], eax      ; Almacenar el carácter convertido
    mov eax, 4          ; Syscall para escribir
    mov ebx, 1          ; Usar la salida estándar
```

```
section .data
msg db 'Resultado: ', 0
newline db 0xA
section .bss
res resb 4 ; Espacio para el resultado

section .text
global _start
_start:
; Cargar un valor en EAX
mov eax, 0xF0F0       ; EAX = 1111 0000 1111 0000 (binario)

; Aplicar máscara usando AND
mov ebx, 0x0F0F       ; EBX = 0000 1111 0000 1111 (binario)
and eax, ebx          ; EAX = EAX AND EBX
                      ; Resultado en EAX = 0000 0000 0000 0000 (binario)

; Guardar el resultado en la sección .bss
mov [res], eax        ; Guarda el valor en 'res' para impresión

; Imprimir el mensaje "Resultado: "
mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, 11
int 0x80

; Imprimir el valor en 'res' como carácter
mov eax, [res]
add eax, '0'          ; Convertir a carácter ASCII (para fines de demostraci
mov [res], eax
mov eax, 4
mov ebx, 1
mov ecx, res
mov edx, 1
int 0x80

; Imprimir nueva línea
mov eax, 4
mov ebx, 1
mov ecx, newline
mov edx, 1
int 0x80

; Terminar el programa
mov eax, 1
xor ebx, ebx
int 0x80
```

```
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)

section .bss
res resb 4 ; Espacio para el resultado

section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 100 ; Coloca 100 en el registro EAX
mov ebx, 8 ; Coloca 8 en el registro EBX (100 + 8 = 108)
add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (100 + 8 = 108)

; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX (108) en la memoria reservada (res)

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Dirección del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el carácter correspondiente a 108 ('l')
mov eax, [res] ; Cargar el resultado en EAX
; No necesitamos hacer ninguna conversión aquí, simplemente lo movemos a [res]
mov [res], al ; Almacenar el carácter 'l' en res

mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir el carácter 'l'

; Imprimir nueva línea
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, newline ; Dirección de la nueva línea
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir nueva línea

; Terminar el programa
mov eax, 1 ; Syscall para salir
xor ebx, ebx ; Código de salida 0
int 0x80 ; Interrupción para terminar el programa
```

```
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)

section .bss
res resb 4 ; Espacio para el resultado

section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 30 ; Coloca 30 en el registro EAX
mov ebx, 38 ; Coloca 38 en el registro EBX (30 + 38 = 68)
add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (30 + 38 = 68)
; Instrucción lógica (AND)
and eax, 0xFF ; Realiza AND bit a bit con 0xFF (255 en decimal),
; EAX seguirá siendo 68 (0x44 en hexadecimal)
; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX (68) en la memoria reservada (res)
; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Dirección del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje
; Imprimir el carácter correspondiente a 68 ('D')
mov eax, [res] ; Cargar el resultado en EAX
movzx eax, byte [res] ; Asegurarse de que EAX contiene el valor correcto
mov [res], al ; Almacenar el valor ASCII correspondiente en res
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir la letra 'D'
; Imprimir nueva línea
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, newline ; Dirección de la nueva línea
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir nueva línea
; Terminar el programa
mov eax, 1 ; Syscall para salir
xor ebx, ebx ; Código de salida 0
int 0x80 ; Interrupción para terminar el programa

```

```
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)

section .bss
res resb 4 ; Espacio para el resultado

section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 30 ; Coloca 30 en el registro EAX
mov ebx, 36 ; Coloca 36 en el registro EBX (30 + 36 = 66)
add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (30 + 36 = 66)
; Instrucción lógica (AND)
and eax, 0xFF ; Realiza AND bit a bit con 0xFF (255 en decimal),
; EAX seguirá siendo 66 (0x42 en hexadecimal)
; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX (66) en la memoria reservada (res)
; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Dirección del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje
; Imprimir el carácter correspondiente a 66 ('B')
mov eax, [res] ; Cargar el resultado en EAX
movzx eax, byte [res] ; Asegurarse de que EAX contiene el valor correcto
mov [res], al ; Almacenar el valor ASCII correspondiente en res
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir la letra 'B'
; Imprimir nueva línea
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, newline ; Dirección de la nueva línea
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir nueva línea
; Terminar el programa
mov eax, 1 ; Syscall para salir
xor ebx, ebx ; Código de salida 0
int 0x80 ; Interrupción para terminar el programa

```

```assembly
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)

section .bss
res resb 4 ; Espacio para el resultado

section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 2 ; Coloca 2 en el registro EAX
; Aquí no se necesita una suma adicional, simplemente usamos 2 directamente.

; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX (2) en la memoria reservada (res)

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Dirección del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el carácter correspondiente a 50 ('2')
mov eax, [res] ; Cargar el resultado en EAX
add eax, '0' ; Convertir el número en carácter ASCII (2 + 48 = 50)
mov [res], al ; Almacenar el carácter correspondiente en res

mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir el carácter '2'

; Imprimir nueva línea
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, newline ; Dirección de la nueva línea
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir nueva línea

; Terminar el programa
mov eax, 1 ; Syscall para salir
xor ebx, ebx ; Código de salida 0
int 0x80 ; Interrupción para terminar el programa
```

```
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)

section .bss
res resb 4 ; Espacio para el resultado

section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 4 ; Coloca 4 en el registro EAX
; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX (4) en la memoria reservada (res)

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Dirección del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el carácter correspondiente a 4 ('4')
mov eax, [res] ; Cargar el resultado en EAX
add eax, '0' ; Convertir el número en carácter ASCII (4 + 48 = 52)
mov [res], al ; Almacenar el carácter correspondiente en res

mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir el carácter '4'

; Imprimir nueva línea
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, newline ; Dirección de la nueva línea
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir nueva línea

; Terminar el programa
mov eax, 1 ; Syscall para salir
xor ebx, ebx ; Código de salida 0
int 0x80 ; Interrupción para terminar el programa
```

Taller 10

```
CMP AX, BX    ; Compara el valor de AX con el valor de BX
JE  Etiqueta  ; Salta a "Etiqueta" si AX == BX (si ZF = 1)
```

```
CMP AX, BX    ; Compara el valor de AX con el valor de BX
JZ  Etiqueta  ; Salta a "Etiqueta" si AX == BX (si ZF = 1)
```

```
CMP AX, BX    ; Compara el valor de AX con el valor de BX
JNE Etiqueta  ; Salta a "Etiqueta" si AX != BX (si ZF = 0)
```

```
CMP AX, BX    ; Compara el valor de AX con el valor de BX
JNZ Etiqueta  ; Salta a "Etiqueta" si AX != BX (si ZF = 0)
```

```
CMP AX, BX    ; Compara el valor de AX con el valor de BX
JGE Etiqueta  ; Salta a "Etiqueta" si AX >= BX (para números con signo)
```

```
CMP AX, BX    ; Compara AX con BX
JL  Etiqueta  ; Salta a "Etiqueta" si AX < BX (en números con signo)
```

```
CMP AX, BX    ; Compara el valor de AX con el valor de BX
JLE Etiqueta  ; Salta a "Etiqueta" si AX <= BX (para números con signo)
```

```
CMP AX, BX    ; Compara el valor de AX con el valor de BX
JS  Etiqueta  ; Salta a "Etiqueta" si el resultado de la comparación es negativo (si SF = 1)
```

```
MOV AX, 0 ;Inicializa sum a 0
MOV CX, 1 ;Inicializa count a 1

while_loop:
CMP CX,10 ; Compara count con 10
JG end while ; Salta al final si count > 10
ADD AX, CX ; Suma count a sum
INC Cx ;Incrementa count
JMP while_loop ;·Repite·el·ciclo
end while:
```

```
MOV AX, 0 ;Inicializa sum a 0
MOV SI, 0 ;puntero a la lista

do_while_loop:
 MOV BX,[SI] ;LEE EL NUMERO DE LA lista
 ADD AX, BX ;SUMA EL NUMERO A SUM
 ADD SI, 2 ;MUEVE EL PUNTERO AL SIGUIENTE NUMERO
 CMP BX, 0 ;VERIFICA SI EL NUMERO ES NEGATIVO
 JS end_do_while ; salta si el numero es NEGATIVO
 JMP do_while_loop ;repite el ciclo
end_do_while
```

```
MOV AX,1 ;INICIALIZA PRODUCT A 1
MOV CX,1 ;Inicializa i a 1

for_loop:
CMP CX,5 ; Compara i con 5
JG end_for ; Salta si i>5
MUL CX ; Multiplica product por i
INC CX;INCREMENTA i
JMP for_loop ; Repite el ciclo
end_for:
```

```
MOV AX, num ;Carga el valor de num
TEST AX,1 ;Prueba si el bit menos significativo es 0 (par)
JZ es_par  ;Salta a es_par si es 0 (num es par)
MOV result_odd, 1; Almacena el resultado en result_odd
JMP end_if_else
es par:
MOV result_even, 1; Almacena el resultado en result_even
end if else:
```

```
MOV CX, 10 ; Inicializa count en 10

for_loop_dec:
 CMP CX,1 ; Compara count con 1
 JL end_for_dec ;Salta si count <1
 ;Aquí se imprime o almacena el valor de count
 DEC CX ;Decrementa count
 JMP for_loop_dec ; Repite el ciclo
 end for_dec:
```

Taller 11

```
section .data
    ; Estructura de fecha (dd/mm/yyyy)
    fecha dd 0       ; Día
          dd 0       ; Mes
          dd 0       ; Año

    ; Estructura de correo electrónico
    correo db "usuario@dominio.com", 0 ; Dirección de correo electrónico terminada en NULL

    ; Estructura de dirección completa
    direccion_calle db "Calle Ejemplo", 0
    direccion_numero db "123", 0
    direccion_colonia db "Colonia Centro", 0

    ; Cadena CURP (18 caracteres + NULL)
    curp db "ABCD010101HDFLRC01", 0
```

```
section .text
    global _start

_start:
    ; Acceso y manipulación de la fecha
    mov eax, [fecha]        ; Cargar el día
    add eax, 1              ; Incrementar el día
    mov [fecha], eax        ; Guardar el día actualizado

    mov eax, [fecha + 4]    ; Cargar el mes
    mov [fecha + 4], 2      ; Establecer el mes a febrero

    mov eax, [fecha + 8]    ; Cargar el año
    mov [fecha + 8], 2024   ; Cambiar el año a 2024

    ; Modificar la dirección
    mov esi, direccion_calle
    mov edi, calle_nueva
    call cambiar_cadena

    ; Terminar el programa
    mov eax, 60             ; syscall: exit
    xor edi, edi            ; status 0
    syscall

cambiar_cadena:
    ; Esta función copia una nueva cadena en una dirección específica
    ; Entradas: ESI -> origen, EDI -> destino
    cld                     ; Asegurar dirección de incremento
    rep movsb               ; Copiar byte a byte
    ret
```

```
section .data
    num1 db 5               ; Define un byte con el valor 5
    num2 db 11              ; Define otro byte con el valor 11
    result db 0             ; Espacio para almacenar el resultado (1 byte)
    message db "Resultado: ", 0 ; Mensaje de texto terminado en NULL para impresión

section .bss
    buffer resb 4           ; Reserva 4 bytes en memoria para el buffer (para números)

section .text
    global _start           ; Punto de entrada principal del programa

%macro PRINT_STRING 1       ; Macro para imprimir cadenas
    mov eax, 4              ; syscall: write
    mov ebx, 1              ; descriptor de archivo (1 = stdout)
    mov ecx, %1             ; dirección de la cadena a imprimir
    mov edx, 13             ; longitud de la cadena (13 caracteres en este caso)
    int 0x80                ; llamada al sistema
%endmacro

%macro PRINT_NUMBER 1       ; Macro para imprimir un número (convertido a carácter)
    mov eax, %1             ; Carga el número en eax
    add eax, '0'            ; Convierte el número a su carácter ASCII
    mov [buffer], eax       ; Almacena el carácter en el buffer
    mov eax, 4              ; syscall: write
    mov ebx, 1              ; descriptor de archivo (1 = stdout)
    mov ecx, buffer         ; dirección del buffer a imprimir
    mov edx, 1              ; longitud del dato a imprimir (1 carácter)
    int 0x80                ; llamada al sistema
%endmacro

_start:
    ; Realizar la suma
    mov al, [num1]          ; Carga el valor de num1 en el registro AL
    add al, [num2]          ; Suma el valor de num2 al contenido de AL
    mov [result], al        ; Almacena el resultado en la variable result

    ; Imprimir el mensaje
    PRINT_STRING message    ; llama al macro PRINT_STRING para imprimir "Resultado: "

    ; Imprimir el número resultado
    PRINT_NUMBER [result]   ; llama al macro PRINT_NUMBER para imprimir el valor de result

    ; Salir del programa
    mov eax, 1              ; syscall: exit
    mov ebx, 0              ; código de salida (0 = éxito)
    int 0x80                ; llamada al sistema para terminar el programa

```

```
section .data
    message db "La suma de los valores es: ", 0 ; Mensaje inicial
    newline db 10, 0           ; Nueva línea para la salida

section .bss
    buffer resb 10             ; Buffer para convertir números a caracteres (hasta 10 bytes)

section .text
    global _start

%macro DEFINE_VALUES 3
    ; Define una "estructura" con tres valores
    val1 db %1                 ; Primer valor
    val2 db %2                 ; Segundo valor
    val3 db %3                 ; Tercer valor
%endmacro

%macro PRINT_STRING 1
    ; Macro para imprimir cadenas de texto
    mov eax, 4                 ; Syscall número para 'write'
    mov ebx, 1                 ; File descriptor para stdout
    mov ecx, %1                ; Dirección del mensaje
    mov edx, 25                ; Longitud fija del mensaje (ajustar si es necesario)
    int 0x80
%endmacro

%macro PRINT_NUMBER 1
    ; Convierte un número en eax a caracteres ASCII y lo imprime
    mov eax, %1                ; Carga el número en EAX
    xor ecx, ecx               ; Limpia ECX (contador de longitud)
    mov edi, buffer + 9        ; Apunta al final del buffer (último byte)
    mov ebx, 10                ; Divisor para obtener dígitos decimales

.convert_to_ascii:
    xor edx, edx               ; Limpia EDX para la división
    div ebx                    ; Divide EAX entre 10, cociente en EAX, residuo en EDX
    add dl, '0'                ; Convierte el residuo (dígito) a ASCII
    dec edi                    ; Mueve el puntero hacia atrás en el buffer
    mov [edi], dl              ; Almacena el carácter en el buffer
    inc ecx                    ; Incrementa la longitud del número
    test eax, eax              ; Verifica si quedan dígitos
    jnz .convert_to_ascii      ; Si quedan dígitos, continúa

    ; Imprime el número desde el buffer
    mov eax, 4                 ; Syscall para 'write'
    mov ebx, 1                 ; Salida estándar
    mov ecx, edi               ; Apunta al inicio del número en el buffer
    mov edx, ecx               ; Calcula la longitud real
    add edx, ecx               ; Ajusta el tamaño de la impresión
    int 0x80
%endmacro

%macro PRINT_SUM 0
    ; Realiza la suma de tres valores y la imprime
```

```
    mov al, [val1]              ; Carga el primer valor en AL
    add al, [val2]              ; Suma el segundo valor
    add al, [val3]              ; Suma el tercer valor
    movzx eax, al               ; Expande AL a EAX para asegurar un valor de 32 bits

    ; Imprime el resultado de la suma
    PRINT_NUMBER eax
    PRINT_STRING newline
%endmacro

; Definimos los tres valores con la macro DEFINE_VALUES
DEFINE_VALUES 3, 5, 7           ; Los valores son 3, 5 y 7

_start:
    ; Imprime el mensaje inicial
    PRINT_STRING message

    ; Imprime la suma de los valores
    PRINT_SUM

    ; Salir del programa
    mov eax, 1                  ; Syscall para 'exit'
    mov ebx, 0                  ; Código de salida
    int 0x80
```

Trabajo en Clase Inline Assembly

```
section .data
    msg db "Hello, World!", 0xA  ; Mensaje a imprimir seguido de un salto de línea
    len equ $ - msg              ; Longitud del mensaje

section .text
    global _start                ; Punto de entrada

_start:
    ; Llamar a write (syscall número 4 en Linux x86)
    mov eax, 4                   ; Código de sistema para write
    mov ebx, 1                   ; Descriptor de archivo (1 = salida estándar)
    mov ecx, msg                 ; Dirección del mensaje
    mov edx, len                 ; Longitud del mensaje
    int 0x80                     ; Llamada al kernel

    ; Llamar a exit (syscall número 1 en Linux x86)
    mov eax, 1                   ; Código de sistema para exit
    xor ebx, ebx                 ; Código de salida (0)
    int 0x80                     ; Llamada al kernel
```

Output:

```
Hello, World!
```

Taller 12

```
section .data
    prompt db "Ingrese un numero: ", 0   ; Mensaje para pedir el número
    result_msg db "El numero incrementado es: ", 0  ; Mensaje de resultado
    newline db 10, 0   ; Nueva línea

section .bss
    input resb 2  ; Reservar espacio para la entrada (número)

section .text
    global _start

_start:
    ; Mostrar mensaje para pedir el número
    mov eax, 4         ; syscall write
    mov ebx, 1         ; file descriptor (stdout)
    mov ecx, prompt    ; mensaje
    mov edx, 18        ; longitud del mensaje
    int 0x80           ; llamada al sistema

    ; Leer la entrada del usuario
    mov eax, 3         ; syscall read
    mov ebx, 0         ; file descriptor (stdin)
    mov ecx, input     ; buffer de entrada
    mov edx, 2         ; longitud máxima de la entrada (para un número de dos dígitos)
    int 0x80           ; llamada al sistema

    ; Convertir entrada de ASCII a número
    movzx eax, byte [input]  ; Cargar el primer byte (carácter) de la entrada
    sub eax, '0'             ; Convertir el carácter ASCII a número (restando '0')

    ; Incrementar el número
    inc eax                  ; Incrementar el número en 1

    ; Mostrar mensaje de resultado
    mov edx, 26              ; longitud del mensaje
    mov ecx, result_msg      ; mensaje de resultado
    mov ebx, 1               ; file descriptor (stdout)
    mov eax, 4               ; syscall write
    int 0x80                 ; llamada al sistema

    ; Convertir el número incrementado de vuelta a ASCII
    add eax, '0'             ; Convertir el número de vuelta a ASCII

    ; Mostrar el número incrementado
    mov [input], al          ; Guardar el resultado en el buffer de entrada (input)
    mov edx, 1               ; longitud del número (un solo carácter)
    mov ecx, input           ; puntero al número incrementado
    mov eax, 4               ; syscall write
    int 0x80                 ; llamada al sistema

    ; Imprimir nueva línea
    mov eax, 4               ; syscall write
    mov ebx, 1               ; file descriptor (stdout)
    mov ecx, newline         ; nueva línea
    mov edx, 1               ; longitud de la nueva línea
    int 0x80                 ; llamada al sistema

    ; Salir del programa
    mov eax, 1               ; syscall exit
    xor ebx, ebx             ; código de salida 0
    int 0x80                 ; llamada al sistema
```