Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Códigos de Ensamblador

Alumno: Ana Karen Ruiz Avila

Matrícula: 369435

Grupo: 932

Asignatura: Organización de Computadoras

Docente: Jonatan Crespo Ragland

Ensenada, B. C. a 28 de noviembre de 2024

TALLER 5 CODIGO.

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

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

section .text
global _start

_start:
; Mostrar mensaje en consola
mov eax, 44
mov ebx, 1
mov ecx, msg ; dirección del mensaje
mov edx, 28 ; longitud del mensaje
int 0x80
```

STDIN

organizacion

Output:

timeout: the monitored command dumped core

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

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

section .text
global _start

_start:
; Mostrar mensaje en consola
mov eax, 4
mov ebx, 8
mov ecx, msg ; dirección del mensaje
mov edx, 28 ; longitud del mensaje
int 0x80
```

STDIN

organizacion

Output:

Program did not output anything!

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

section .bss
input resb 1 ; Espacio para almacenar el carácter 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 ; dirección del mensaje
mov edx, 28 ; longitud del mensaje
int 0x80

; Leer un carácter desde el teclado

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

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

; Calcular la suma de los caracteres
mov al, [input]
add al, [input]
```

STDIN

organizacion

Output:

imprimir imput del teclado: organizacion

```
mov ebx, 1
mov ecx, msg ; dirección del mensaje
mov edx, 28 ; longitud del mensaje
int 0x80

; Leer un carácter desde el teclado

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

; Mostrar el carácter ingresado
mov eax, 4 ; syscall número 4 es write (sys_write)
mov ebx, 1 ; descriptor de archivo 1 es stdout
mov ecx, input ; dirección del carácter
mov edx, 100 ; longitud del carácter
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 ; dirección de la suma
mov edx, 1 ; longitud de la suma
int 0x80

; Terminar el programa
mov eax, 1
xor ebx, ebx ; código de salida 0
int 0x50
```

STDIN

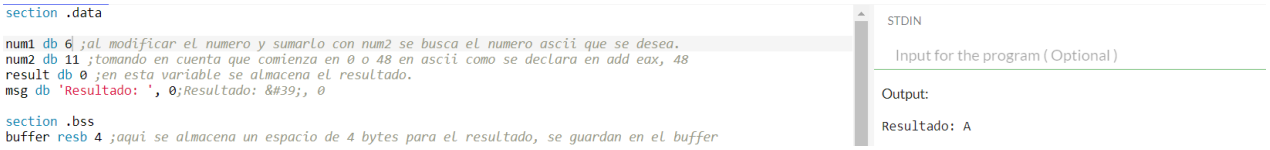
organizacion

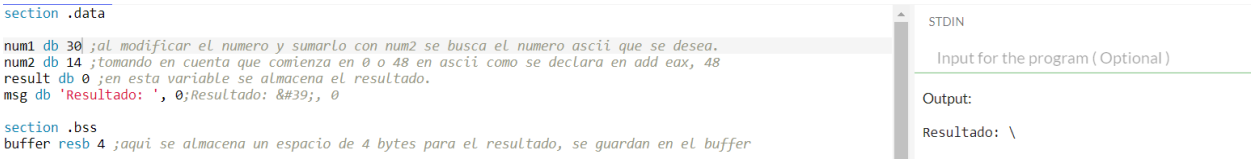
Output:

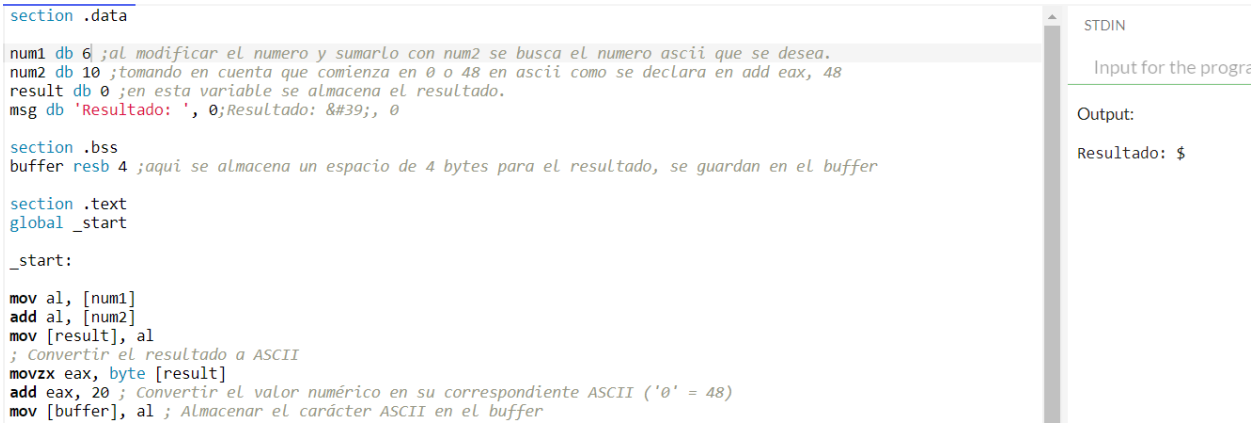
imprimir imput del teclado: organizacion
timeout: the monitored command dumped core

TALLER 7 CODIGO.

```
section .data

num1 db 5 ;al modificar el numero y sumarlo con num2 se busca el numero ascii que se desea.
num2 db 11 ;tomando en cuenta que comienza en 0 o 48 en ascii como se declara en add eax, 48
result db 0 ;en esta variable se almacena el resultado.
msg db 'Resultado: ', 0;Resultado: &#39;, 0

section .bss
buffer resb 4 ;aqui se almacena un espacio de 4 bytes para el resultado, se guardan en el buffer

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 numérico en su correspondiente ASCII ('0' = 48)
mov [buffer], al ; Almacenar el carácter 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 19 ;al modificar el numero y sumarlo con num2 se busca el numero ascii que se desea.
num2 db 10 ;tomando en cuenta que comienza en 0 o 48 en ascii como se declara en add eax, 48
result db 0 ;en esta variable se almacena el resultado.
msg db 'Resultado: ', 0;Resultado: &#39;, 0

section .bss
buffer resb 4 ;aqui se almacena un espacio de 4 bytes para el resultado, se guardan en el buffer
```

STDIN

Input for the program (Optional)

Output:

```
Resultado: 1
```

```
section .data

num1 db 8 ;al modificar el numero y sumarlo con num2 se busca el numero ascii que se desea.
num2 db 10 ;tomando en cuenta que comienza en 0 o 48 en ascii como se declara en add eax, 48
result db 0 ;en esta variable se almacena el resultado.
msg db 'Resultado: ', 0;Resultado: &#39;, 0

section .bss
buffer resb 4 ;aqui se almacena un espacio de 4 bytes para el resultado, se guardan en el buffer
```

STDIN

Input for the program (Optional)

Output:

```
Resultado: &
```

```
1  section .data
2  msg db 'Resultado: ', 0 ; Mensaje de salida
3  section .bss
4  buffer resb 1 ;aqui se almacena un espacio de 4 bytes para el resultado, se guardan en el buffer
5  section .text
6  global _start
7
8  _start:
9
10 mov al, '@'
11 mov [buffer], al ; Almacenar el carácter ASCII en el buffer
12
13 mov eax, 4
14 mov ebx, 1
15 mov ecx, msg
16 mov edx, 11
17 int 0x80
18
19 mov eax, 4
20 mov ebx, 1
21 mov ecx, buffer
22 mov edx, 1
23 int 0x80
24
25 mov eax, 1
26 xor ebx, ebx
27 int 0x80
```

STDIN

Input for the program (Optional)

Output:

```
Resultado: @
```

```
1  section .data
2  char_at db '@'
3  msg db 'Resultado: ', 0 ; Mensaje de salida
4  section .bss
5  buffer resb 1 ;aqui se almacena un espacio de 4 bytes para el resultado, se guardan en el buffer
6  section .text
7  global _start
8
9  _start:
10
11 mov esi, char_at
12 mov al, [esi]
13 mov [buffer], al ; Almacenar el carácter ASCII en el buffer
14
15 mov eax, 4
16 mov ebx, 1
17 mov ecx, msg
18 mov edx, 11
19 int 0x80
20
21 mov eax, 4
22 mov ebx, 1
23 mov ecx, buffer
24 mov edx, 1
25 int 0x80
26
27 mov eax, 1
28 xor ebx, ebx
29 int 0x80
```

STDIN

Input for the program (Optional)

Output:

```
Resultado: @
```

TALLER 8 CODIGO.

```
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 ; Coloca el numero en el registro EAX
  mov ebx, 5 ; Coloca el numero en el registro EBX
  add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX

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

  ; Instrucciones de manipulación de bits
  shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda (15 << 1 = 30)

  ; Guardar el resultado en la sección .bss
  mov [res], eax ; Almacena el valor de EAX (30) 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 ; 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) + 48 del 0 (30 + 48)
  mov [res], eax ; Almacenar el carácter convertido


section .data
    value db 0b10101100   ; Definimos un byte con el valor 10101100 en binario

section .text
global _start
_start:
    ; Cargamos el valor en el registro AL
    mov al, [value]      ; AL = 10101100

    ; Enmascarar los primeros 4 bits usando AND
    and al, 0b00001111   ; Apagamos los primeros 4 bits en AL
                         ; AL ahora es 00001100

    ; Si queremos verificar que el bit este encendido, se crea una mascara que tiene
    ; solo ese bit encendido
    mov ah, al           ; Guardamos el valor en AH para no perderlo
    and al, 0b00000010   ; Comparamos con la máscara

    ; Si el resultado es distinto de cero, significa que el bit estaba encendido
    cmp al, 0            ; Comparamos el resultado con 0
    jne bit_is_set       ; Si es diferente de 0, saltamos a bit_is_set

bit_is_set:
    ; Aquí iría el código si el bit SÍ está encendido
    ; ...

    ; Finalizamos el programa
    mov eax, 60          ; Syscall para salir
    xor edi, edi         ; Código de salida 0
    syscall
```

```
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, 1 ; Coloca el numero en el registro EAX
  mov ebx, 1 ; Coloca el numero en el registro EBX
  add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (1 + 1 = 2)

  ; Instrucción lógica (AND)
  and eax, 0xF

  ; Instrucciones de manipulación de bits
  shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda (2 << 1 = 4)

  ; 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 ; Direccion del mensaje a imprimir
  mov edx, 11
  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) + 48 del 0 (4 + 48)
  mov [res], eax
```

```
  res resb 4 ; Espacio para el resultado

section .text
  global _start

_start:

  ; Instrucciones aritméticas
  mov eax, 1 ; Coloca el numero en el registro EAX
  mov ebx, 0 ; Coloca el numero en el registro EBX
  add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (1 + 0 = 1)

  ; Instrucción lógica (AND)
  and eax, 0xF

  ; Instrucciones de manipulación de bits
  shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda (1 << 1 = 2)

  ; 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 ; Direccion del mensaje a imprimir
  mov edx, 11
  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) + 48 del 0 (2 + 48)
  mov [res], eax
  mov eax, 4
  mov ebx, 1
  mov ecx, res
  mov edx, 1
  int 0x80 ; Interrupción para imprimir el número
```

```
  res resb 4 ; Espacio para el resultado

section .text
  global _start

_start:

  ; Instrucciones aritméticas
  mov eax, 1 ; Coloca el numero en el registro EAX
  mov ebx, 8 ; Coloca el numero en el registro EBX
  add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (1 + 8 = 9)

  ; Instrucción lógica (AND)
  and eax, 0xF

  ; Instrucciones de manipulación de bits
  shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda (9 << 1 = 18)

  ; 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 ; Direccion del mensaje a imprimir
  mov edx, 11
  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) + 48 del 0 (18 + 48)
  mov [res], eax
  mov eax, 4
  mov ebx, 1
  mov ecx, res
  mov edx, 1
  int 0x80 ; Interrupción para imprimir el número
```

```
  res resb 4 ; Espacio para el resultado

section .text
  global _start

_start:

  ; Instrucciones aritméticas
  mov eax, 1 ; Coloca el numero en el registro EAX
  mov ebx, 9 ; Coloca el numero en el registro EBX
  add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (1 + 9 = 10)

  ; Instrucción lógica (AND)
  and eax, 0xF

  ; Instrucciones de manipulación de bits
  shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda (10 << 1 = 20)

  ; 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 ; Direccion del mensaje a imprimir
  mov edx, 11
  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) + 48 del 0 (20 + 48)
  mov [res], eax
  mov eax, 4
  mov ebx, 1
  mov ecx, res
  mov edx, 1
  int 0x80 ; Interrupción para imprimir el número
```

```
  res resb 4 ; Espacio para el resultado

section .text
  global _start

_start:

  ; Instrucciones aritméticas
  mov eax, 7 ; Coloca el numero en el registro EAX
  mov ebx, 7 ; Coloca el numero en el registro EBX
  add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (7 + 7 = 14)

  ; Instrucción lógica (AND)
  and eax, 0xF

  ; Instrucciones de manipulación de bits
  shl eax, 2 ; Desplaza los bits de EAX una posición a la izquierda (14 << 2 = 28)

  ; 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 ; Direccion del mensaje a imprimir
  mov edx, 11
  int 0x80 ; Interrupción para imprimir el mensaje

  ; Imprimir el número (resultado almacenado en 'res')
  mov eax, [res] ; Cargar el resultado en EAX
  add eax, '4' ; Convertir el número en carácter (ASCII) + 44 del 0 (20 + 48)
  mov [res], eax
  mov eax, 4
  mov ebx, 1
  mov ecx, res
  mov edx, 1
  int 0x80 ; Interrupción para imprimir el número
```

TALLER 10 CODIGO.

```
Mov AX, o ;Se inicia suma a 0
Mov CX, 1 ;Se inicia count a 1

while_ciclo:
CMP CX, 10 ;Compara count con 10
JG end while ;Si count es mayor a 10, salta al final
ADD AX, CX ;Suma count a suma
INC CX ;Se incrementa el count
JMP while_ciclo ;Se repite el while_ciclo
end while:

Mov AX, 0 ;Se inicia suma a 0
Mov SI, 0 ;Es un puntero agregado a la lista

do_while_ciclo:
Mov BX, [SI] ;Lee el numero de la lista
ADD AX, BX ;Suma el numero al valor Suma
ADD SI, 2 ;Mueve el puntero al siguiente numero
CMP BX, 0 ;Se verifica si el numero es negativo
JS end_do_while ;Si el numero es negativo, salta
JMP do_while_ciclo ;Se repite el ciclo
end_do_while
```

```
Mov AX, 0 ;Se inicia product a 0
Mov CX, 1 ;Se inicia i a 1

for_ciclo:
CMP CX, 5 ;Se compara i con 5
JG end_for ;Si i es mayor a 5, salta
MUL CX ;Se multiplica product por i
INC CX; Se incrementa i
JMP for_ciclo ;Se repite el ciclo
end_for:
```

```
Mov AX, num ;Se carga el valor de num
TEST AX, 1 ;Se hace una prueba si el bit menos significativo es 0 (par)
JZ par ;Salta si par si tiene el valor de 0 (numero es par)
MOV resul_val, 1 ;Se guarda el resultado en resul_val
JMP end_if_else
par:
MOV result_valor, 1 :Se guarda el resultado en result_valor
end_if_else
```

```
Mov CX, 10 ;Se inicia count con 10

for_ciclo_dec:
CMP CX, 1 ;Se compara count con 1
JL end_for_dec ;Si cout es menor a 1, salta
;Se guarda el valor de count
DEC CX ; Decrementa el count
JMP for_ciclo_dec ;Se repite el ciclo
end_for_dec:
```

```
Mov AL, num1 ;Se carga el primer numero
ADD AL, num2 ;Se suma el segundo numero
CMP AL, 0 ;Compara la suma con 0

JE valor_cero ;Si el resultado es cero, salta a valor_cero
;Se imprime el resultado de la suma
JMP end_sum
valor_cero:
;Se imprime "Esto es cero"
end_sum:
```

TALLER 11 CODIGO.

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

    ;Estructura de cirrei electronico
    correo db "ejemplo@organizacion.com", 0 ;Terminada en NULL

    ;Estructura de direccion completa
    calle db "Calle falsa", 0
    numero db "123", 0
    colonia db "colonia maneayork", 0

    ;CURP (18 caracteres y NULL)
    curp db "ABCD010203HBCRLZ05", 0
```

```
1  section .text
2    global _start
3
4  _start:
5    ;Modificacion de la fecha
6    mov eax, [fecha]        ;dia
7    add eax, 1              ;aumenta el dia
8    mov [fecha], eax        ;guarda el nuevo dia
9
10   mov eax, [fecha + 4]    ;mes
11   mov [fecha + 4], 3      ;establece el mes a marzo
12
13   mov eax, [fecha + 8]    ;año
14   mov [fecha + 8], 2024   ;cambia el año a 2024
15
16   ;modificacion de la direccion
17   mov esi, calle
18   mov edi, calle_nueva
19   call cambiar_cadena
20
21   ;termina el programa
22   mov eax, 60             ;syscall: exit
23   xor edi, edi            ;status 0
24   syscall
25
26 cambiar_cadena:
27   ;copia una nueva cadena en la direccion que se especifique
28   ;entradas: ESI -> origen, EDI -> destino
29   cld                     ;asegura la direccion de incremento
30   rep movsb               ;copia byte a byte
31   ret
```

TALLER 111

```
section .data
    num1 db 5      ;Define el primer byte con 5
    num2 db 11     ;Define el segundo byte con 11
    result db 0    ;Se almacena el resultado
    message db "Resultado: ", 0   ;Mensaje de texto con terminacion NULL

section .bss
    buffer resb 4 ;Reserva 4 bytes en la memoria para el buffer

section .text
    global _start ;Inicio del programa

%macro PRINT_STRING 1   ;Macro para imprimir las cadenas
    mov eax, 4          ;syscall: write
    mov ebx, 1          ;descripta el archivo (1 = stdout)
    mov ecx, %1         ;es la direccion de la cadena que se va a imprimir
    mov edx, 13         ;el largo de la cadena (en este caso 13 caracteres)
    int 0x80            ;se llama al sistema
%endmacro

%macro PRINT_NUMBER 1   ;Macro para imprimir un numero (convertido a cadena)
    mov eax, %1         ;Se carga el numero en eax
    add eax, '0'        ;Convierte el numeroa su caracter equivalente en ASCII
    mov [buffer], eax   ;Almacena el caracter en el buffer
    mov eax, 4          ;syscall: write
    mov ebx, 1          ;Descripta el archivo (1 = stdout)
    mov ecx, buffer     ;es la direccion del buffer a imprimir
    mov edx, 1          ;es el largo del dato a imprimir (1 caracter)
    int 0x80            ;Se llama al sistema
%endmacro

_start:                 ;se realiza la suma
    mov al, [num1]      ;se carga el valor del num1 en el registro de AL
    add al, [num2]      ;suma el valor de num2 al contenido de AL
    mov [result], al    ;se guarda el resultado de la variable result

    PRINT_STRING message    ;llama a la macro PRINT_STRING para imprimirlo

    PRINT_NUMBER [result]   ;llama a la macro PRINT_NUMBER para imprimir el valor de result

    ; Salir del programa
    mov eax, 1        ;syscall: exit
    mov ebx, 0        ;codigo de salida (0 = exito)
    int 0x80          ;llama al sistema para terminar el programa
```

TALLER 112

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

section .bss
    buffer resb 4              ; Buffer para convertir números a caracteres

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
    mov eax, 4                 ; Syscall número para 'write'
    mov ebx, 1                 ; File descriptor para stdout
    mov ecx, %1                ; Dirección del mensaje
    mov edx, 25                ; Longitud del mensaje
    int 0x80
%endmacro

%macro PRINT_NUMBER 1
    ; Convierte un número en eax a caracteres ASCII y lo imprime
    mov eax, %1
    mov ecx, buffer            ; Usamos el buffer para guardar el resultado
    mov ebx, 10                ; Divisor para obtener dígitos decimales

.next_digit:
    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 dígito a ASCII
    dec ecx                    ; Mueve hacia atrás en el buffer
    mov [ecx], dl              ; Almacena el dígito en el buffer
```

```
    test eax, eax              ; Verifica si quedan dígitos
    jnz .next_digit            ; Si quedan dígitos, continúa

    ; Imprime el número
    mov eax, 4                 ; Syscall para write
    mov ebx, 1                 ; Salida estándar
    mov ecx, buffer            ; Comienza en el primer dígito
    mov edx, buffer + 4        ; Longitud máxima asumida en 4 dígitos
    sub ecx, edx               ; Calcula la longitud real
    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

_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
```