

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA ARQUITECTURA Y DISEÑO

INGENIERÍA EN SOFTWARE Y TECNOLOGÍAS EMERGENTES



Organización de computadoras

Taller 12

ADRIAN BALDERAS ROSAS

Jonatan Crespo Ragland

Entrada de datos:

- Solicita dos números al usuario.

- Convierte las cadenas ingresadas (representadas en ASCII) a enteros con la función atoi.

Cálculo:

- Suma los números utilizando las instrucciones básicas de ensamblador.

Salida de datos:

- Convierte el resultado de entero a cadena con la función itoa.

- Muestra el resultado en la consola.

Justificación: Este programa ejemplifica cómo manejar entrada y salida de datos en ensamblador, lo cual es útil para desarrollar sistemas donde se requiere bajo nivel de abstracción y control directo sobre el hardware.

The screenshot shows an online NASM compiler IDE. The main editor displays assembly code for a program that requests two numbers, converts them to integers, sums them, and displays the result. The code is as follows:

```
1- section .data
2 msg1 db "Ingrese el primer numero: ", 0 ; Mensaje para solicitar el primer número
3 msg2 db "Ingrese el segundo numero: ", 0 ; Mensaje para solicitar el segundo número
4 result_msg db "El resultado es: ", 0 ; Mensaje para mostrar el resultado
5 buffer db 10 ; Buffer para leer números desde el teclado
6
7- section .bss
8 num1 resb 4 ; Reservar espacio para el primer número (4 bytes)
9 num2 resb 4 ; Reservar espacio para el segundo número (4 bytes)
10 result resb 4 ; Reservar espacio para el resultado (4 bytes)
11
12- section .text
13 global _start
14
15- _start:
16 ; Solicitar el primer número al usuario
17 mov eax, 4 ; syscall write
18 mov ebx, 1 ; stdout
19 mov ecx, msg1 ; dirección del mensaje
20 mov edx, 24 ; longitud del mensaje
21 int 0x80 ; llamada al sistema
22
23 ; Leer el primer número desde el teclado
24 mov eax, 3 ; syscall read
25 mov ebx, 0 ; stdin
26 mov ecx, num1 ; almacenar el número ingresado
27 mov edx, 10 ; tamaño del buffer
28 int 0x80 ; llamada al sistema
29
30 ; Solicitar el segundo número al usuario
31 mov eax, 4 ; syscall write
32 mov ebx, 1 ; stdout
33 mov ecx, msg2 ; dirección del mensaje
34 mov edx, 24 ; longitud del mensaje
35 int 0x80 ; llamada al sistema
36
37 ; Leer el segundo número desde el teclado
38 mov eax, 3 ; syscall read
39 mov ebx, 0 ; stdin
40 mov ecx, num2 ; almacenar el número ingresado
41 mov edx, 10 ; tamaño del buffer
42 int 0x80 ; llamada al sistema
43
44 ; Convertir los números de cadena a entero
45 mov esi, num1 ; Dirección del primer número
46 call atoi ; Convierte a entero y lo guarda en eax
47 mov [num1], eax ; Almacenar el primer número convertido
48
49 mov esi, num2 ; Dirección del segundo número
50 call atoi ; Convierte a entero y lo guarda en eax
51 mov [num2], eax ; Almacenar el segundo número convertido
52
53 ; Realizar la suma
54 mov eax, [num1] ; Cargar el primer número en eax
55 add eax, [num2] ; Sumar el segundo número
56 mov [result], eax ; Guardar el resultado en memoria
57
58 ; Mostrar el resultado
59 mov eax, 4 ; syscall write
60 mov ebx, 1 ; stdout
61 mov ecx, result_msg ; Mensaje del resultado
62 mov edx, 18 ; longitud del mensaje
63 int 0x80 ; llamada al sistema
```

The right sidebar shows the 'Input/Output' section with the following output:

```
Ingrese el primer numero1
Ingrese el segundo numero2
El resultado es: 3
.....
```

At the bottom of the sidebar, it states: 'Compiled and executed in 11.301 sec(s)'.

section .data

msg1 db "Ingrese el primer numero: ", 0 ; Mensaje para solicitar el primer número

msg2 db "Ingrese el segundo numero: ", 0 ; Mensaje para solicitar el segundo número

result_msg db "El resultado es: ", 0 ; Mensaje para mostrar el resultado

buffer db 10 ; Buffer para leer números desde el teclado

section .bss

```
num1 resb 4    ; Reservar espacio para el primer número (4 bytes)
num2 resb 4    ; Reservar espacio para el segundo número (4 bytes)
result resb 4  ; Reservar espacio para el resultado (4 bytes)
```

```
section .text
global _start
```

```
_start:
```

```
; Solicitar el primer número al usuario
mov eax, 4          ; syscall write
mov ebx, 1          ; stdout
mov ecx, msg1       ; dirección del mensaje
mov edx, 24         ; longitud del mensaje
int 0x80            ; llamada al sistema
```

```
; Leer el primer número desde el teclado
mov eax, 3          ; syscall read
mov ebx, 0          ; stdin
mov ecx, num1       ; almacenar el número ingresado
mov edx, 10         ; tamaño del buffer
int 0x80            ; llamada al sistema
```

```
; Solicitar el segundo número al usuario
mov eax, 4          ; syscall write
mov ebx, 1          ; stdout
mov ecx, msg2       ; dirección del mensaje
mov edx, 24         ; longitud del mensaje
int 0x80            ; llamada al sistema
```

```
; Leer el segundo número desde el teclado
mov eax, 3          ; syscall read
mov ebx, 0          ; stdin
mov ecx, num2       ; almacenar el número ingresado
mov edx, 10         ; tamaño del buffer
int 0x80            ; llamada al sistema
```

```
; Convertir los números de cadena a entero
```

```
mov esi, num1          ; Dirección del primer número
call atoi              ; Convierte a entero y lo guarda en eax
mov [num1], eax        ; Almacenar el primer número convertido
```

```
mov esi, num2          ; Dirección del segundo número
call atoi              ; Convierte a entero y lo guarda en eax
mov [num2], eax        ; Almacenar el segundo número convertido
```

```
; Realizar la suma
mov eax, [num1]        ; Cargar el primer número en eax
add eax, [num2]        ; Sumar el segundo número
mov [result], eax      ; Guardar el resultado en memoria
```

```
; Mostrar el resultado
mov eax, 4             ; syscall write
mov ebx, 1             ; stdout
mov ecx, result_msg    ; Mensaje del resultado
mov edx, 18            ; Longitud del mensaje
int 0x80               ; llamada al sistema
```

```
; Convertir el resultado de entero a cadena
mov eax, [result]      ; Cargar el resultado
call itoa              ; Convierte a cadena, resultado en esi
```

```
; Imprimir el resultado convertido
mov eax, 4             ; syscall write
mov ebx, 1             ; stdout
mov ecx, esi           ; Dirección del resultado convertido
mov edx, 10            ; Tamaño máximo
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
```

; Función atoi (cadena a entero)

```

atoi:
    xor eax, eax          ; Limpiar eax (acumulador)
    xor ebx, ebx          ; Limpiar ebx (multiplicador)
next_digit:
    mov bl, byte [esi]    ; Leer un carácter
    cmp bl, 0x0A          ; Comparar con salto de línea (Enter)
    je end_atoi           ; Si es Enter, terminar
    sub bl, '0'           ; Convertir de ASCII a número
    imul eax, eax, 10      ; Multiplicar el acumulador por 10
    add eax, ebx           ; Agregar el dígito convertido
    inc esi               ; Avanzar al siguiente carácter
    jmp next_digit        ; Repetir para el siguiente carácter
end_atoi:
    ret

; Función itoa (entero a cadena)
itoa:
    mov edi, buffer       ; Apuntar al buffer
    xor ecx, ecx          ; Limpiar ecx (contador de dígitos)
itoa_loop:
    xor edx, edx          ; Limpiar edx
    div dword [ten]       ; Dividir eax por 10
    add dl, '0'           ; Convertir el residuo en un carácter
    dec edi               ; Retroceder el puntero del buffer
    mov [edi], dl         ; Almacenar el carácter en el buffer
    inc ecx               ; Incrementar el contador de dígitos
    test eax, eax         ; Verificar si eax es 0
    jnz itoa_loop         ; Si no es 0, continuar
    mov esi, edi          ; Configurar esi para que apunte al inicio de la
cadena
    ret

section .data
    ten dd 10             ; Constante 10 utilizada para itoa

```