

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.

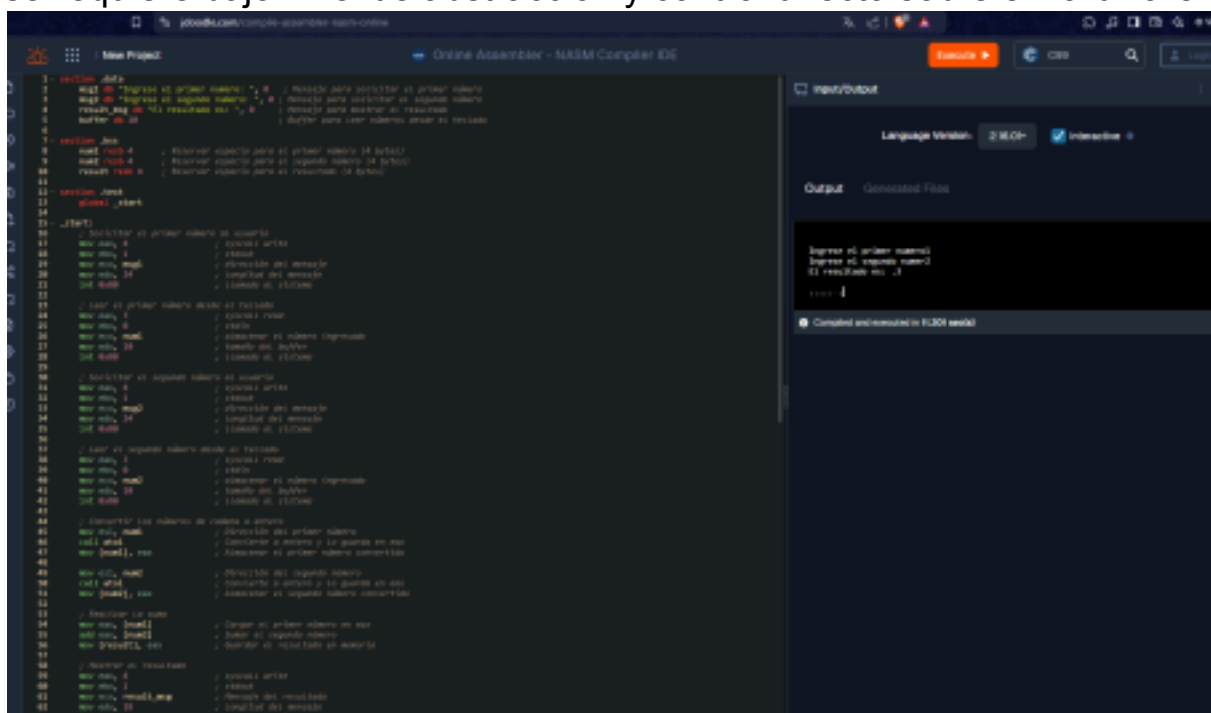
- Convierte las cadenas ingresadas (representadas en ASCII) a enteros con la función atoi.

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.



```
1 section .data
2     msg1 db "Ingrese el primer numero: ", 0 ; Mensaje para solicitar el primer numero
3     msg2 db "Ingrese el segundo numero: ", 0 ; Mensaje para solicitar el segundo numero
4     result db "El resultado es: ", 0 ; Mensaje para mostrar el resultado
5     buffer db 256
6
7 section .bss
8     num1 resb 4 ; Reservar espacio para el primer numero (4 bytes)
9     num2 resb 4 ; Reservar espacio para el segundo numero (4 bytes)
10    result resb 4 ; Reservar espacio para el resultado (4 bytes)
11
12 section .text
13     _start:
14
15     ; Solicitar el primer numero al usuario
16     mov eax, 4 ; System call: write
17     mov ebx, 1 ; File descriptor: stdout
18     mov ecx, msg1 ; Address of message
19     mov edx, 24 ; Length of message
20     int 0x80 ; Linux syscall
21
22     ; Leer el primer numero desde el teclado
23     mov eax, 0 ; System call: read
24     mov ebx, 0 ; File descriptor: stdin
25     mov ecx, num1 ; Address of buffer
26     mov edx, 24 ; Length of buffer
27     int 0x80 ; Linux syscall
28
29     ; Solicitar el segundo numero al usuario
30     mov eax, 4 ; System call: write
31     mov ebx, 1 ; File descriptor: stdout
32     mov ecx, msg2 ; Address of message
33     mov edx, 24 ; Length of message
34     int 0x80 ; Linux syscall
35
36     ; Leer el segundo numero desde el teclado
37     mov eax, 0 ; System call: read
38     mov ebx, 0 ; File descriptor: stdin
39     mov ecx, num2 ; Address of buffer
40     mov edx, 24 ; Length of buffer
41     int 0x80 ; Linux syscall
42
43     ; Convertir los numeros de cadena a enteros
44     mov ecx, num1 ; Address of first number
45     call atoi ; Convertir a entero y lo guarda en eax
46     mov [num1], eax ; Almacenar el primer numero convertido
47
48     mov ecx, num2 ; Address of second number
49     call atoi ; Convertir a entero y lo guarda en eax
50     mov [num2], eax ; Almacenar el segundo numero convertido
51
52     ; Realizar la suma
53     mov ecx, [num1] ; Cargar el primer numero en ecx
54     mov ebx, [num2] ; Cargar el segundo numero en ebx
55     add ebx, ecx ; Sumar el segundo numero al primero
56     mov [result], ebx ; Guardar el resultado en result
57
58     ; Mostrar el resultado
59     mov eax, 4 ; System call: write
60     mov ebx, 1 ; File descriptor: stdout
61     mov ecx, result ; Address of result
62     mov edx, 24 ; Length of message
63     int 0x80 ; Linux syscall
```

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