

Universidad Autónoma de Baja California



Facultad de Ingeniería, Arquitectura y Diseño

Ingeniería en Software

Organización de Computadoras

Taller 10 GIT HUB

Brayan Arturo Rocha Meneses

Ensenada Baja California Noviembre de 2024



Assembly ▾



```
1 section .data
2 sum db 0 ; Variable para almacenar la suma
3 count db 1 ; Contador inicializado en 1
4 section .text
5 global _start
6 _start:
7 mov [sum], 0 ; Inicializar sum a 0
8 mov [count], 1 ; Inicializar count a 1
9 while_loop:
10 CMP [count], 10 ; Comparar count con 10
11 JG end_while ; Si count > 10, salir del bucle
12 add [sum], [count] ; Sumar count a sum
13 inc [count] ; Incrementar count
14 JMP while_loop ; Repetir el bucle
15 end_while:
```



Assembly ▾



```
1 section .data
2 lista db 5, 3, 8, -1 ; Lista de números
3 sum db 0 ; Variable para almacenar la suma
4 ptr db lista ; Puntero al inicio de la lista
5 section .text
6 global _start
7 _start:
8 mov [sum], 0 ; Inicializar sum a 0
9 do_while_loop:
10 mov al, [ptr] ; Leer el número de la lista
11 add [sum], al ; Sumar el número a sum
12 inc ptr ; Mover al siguiente número
13 CMP al, 0 ; Verificar si el número es negativo
14 JS end_do_while ; Si es negativo, terminar el bucle
15 JMP do_while_loop ; Repetir el bucle
16 end_do_while:
17 |
```



Assembly ▾



```
1 product db 1 ; Inicializar product a 1
2 i db 1 ; Inicializar i a 1
3 section .text
4 global _start
5 _start:
6 mov [product], 1 ; Inicializar product a 1
7 mov [i], 1 ; Inicializar i a 1
8 for_loop:
9 CMP [i], 5 ; Comparar i con 5
10 JG end_for ; Si i > 5, salir del bucle
11 mul [i] ; Multiplicar product por i
12 inc [i] ; Incrementar i
13 JMP for_loop ; Repetir el bucle
14 end_for
15
```



Assembly ▾



```
1 section .data
2 num db 3 ; Número a verificar
3 result_even db 0 ; Almacena el resultado si es par
4 result_odd db 0 ; Almacena el resultado si es impar
5 section .text
6 global _start
7 _start:
8 mov al, [num]
9 and al, 1 ; Verificar el bit menos significativo
10 JZ even ; Si el bit es 0, es par
11 JMP odd ; Si no, es impar
12 even:
13 mov [result_even], 1 ; Almacenar el valor en result_even
14 JMP end_if_else
15 odd:
16 mov [result_odd], 1 ; Almacenar el valor en result_odd
17 end_if_else
```



Assembly ▾




```
1 section .data
2 count db 10 ; Inicializar count en 10
3 section .text
4 global _start
5 _start:
6 mov [count], 10 ; Inicializar count a 10
7 for_loop_dec:
8 CMP [count], 1 ; Comparar count con 1
9 JL end_for_dec ; Si count < 1, salir del bucle
10 ; Aquí se imprimiría o almacenaría el valor de count
11 dec [count] ; Decrementar count
12 JMP for_loop_dec ; Repetir el bucle
13 end_for_dec:
```



Assembly ▾



```
1 section .data
2 num1 db 4 ; Primer número
3 num2 db 5 ; Segundo número
4 result db 0 ; Resultado de la suma
5 msg db "Resultado: ", 0 ; Mensaje inicial
6 resultStr db "00", 10 ; Cadena para el resultado en ASCII y salto de línea
7 zeroMsg db "Esto es un cero", 10 ; Mensaje de cero
8 section .text
9 global _start
10 _start:
11 mov al, [num1] ; Cargar num1 en AL
12 add al, [num2] ; Sumar num2 a AL
13 mov [result], al ; Guardar el resultado
14 CMP al, 0 ; Verificar si el resultado es 0
15 JE print_zero ; Saltar si el resultado es 0
16 print_result:
17 ; Imprimir el mensaje y el resultado ASCII
18 mov eax, 4 ; Syscall para escribir
19 mov ebx, 1 ; Salida estándar
20 mov ecx, msg ; Dirección del mensaje
21 mov edx, 11 ; Longitud del mensaje
22 int 0x80 ; Llamada al sistema
23 mov eax, 4
24 mov ebx, 1
```

 Assembly ▾



```
17 ; Imprimir el mensaje y el resultado ASCII
18 mov eax, 4 ; Syscall para escribir
19 mov ebx, 1 ; Salida estándar
20 mov ecx, msg ; Dirección del mensaje
21 mov edx, 11 ; Longitud del mensaje
22 int 0x80 ; Llamada al sistema
23 mov eax, 4
24 mov ebx, 1
25 mov ecx, resultStr
26 mov edx, 1
27 int 0x80
28 JMP exit_program
29 print_zero:
30 ; Imprimir el mensaje "Esto es un cero"
31 mov eax, 4
32 mov ebx, 1
33 mov ecx, zeroMsg
34 mov edx, 16
35 int 0x80
36 exit_program:
37 ; Terminar el programa
38 mov eax, 1 ; Syscall para salir
39 xor ebx, ebx
40 int 0x80
```