



**Universidad Autónoma de Baja California
Facultad de Ingeniería Arquitectura y Diseño**

Organización de Computadoras

Docente:

Crespo Regland Jonatan

Alumno:

Garcia Ceseña Edgar Rafael

GRUPO 932

Ingeniería en Software y Tecnologías Emergentes

TALLER 10



1.

```
1 section .data
2     sum db 0           ; Variable para almacenar la suma
3     count db 1         ; Variable contador
4
5 section .text
6 global _start
7 _start:
8     mov al, 0           ; Inicializa sum a 0
9     mov bl, 1           ; Inicializa count a 1
10
11 while_loop:
12     cmp bl, 10          ; Verifica si count <= 10
13     jg end_while        ; Si count > 10, termina el bucle
14
15     add al, bl           ; Suma count a sum
16     inc bl              ; Incrementa count
17     jmp while_loop      ; Repite el ciclo
18
19 end_while:
20     mov [sum], al        ; Guarda el resultado en sum
21
22     ; Fin del programa (en un sistema real, aquí podría hacerse una syscall para salir)
23
```

2.

```
1 section .data
2     lista db 5, 7, 3, -1, 8 ; Lista de números
3     sum db 0                ; Variable para almacenar la suma
4
5 section .text
6 global _start
7 _start:
8     mov al, 0               ; Inicializa sum a 0
9     mov si, lista           ; Apunta al inicio de la lista
10
11 do_while_loop:
12     mov bl, [si]            ; Carga el número actual de la lista
13     add al, bl              ; Añade el número a sum
14     cmp bl, 0               ; Verifica si el número es negativo
15     js end_do_while         ; Si el número es negativo, termina el bucle
16
17     inc si                  ; Mueve el puntero al siguiente número
18     jmp do_while_loop       ; Repite el ciclo
19
20 end_do_while:
21     mov [sum], al           ; Guarda el resultado en sum
22
23     ; Fin del programa
24
```

3.

```
1 section .data
2     product db 1          ; Variable para almacenar el producto
3     i db 1                ; Variable contador
4
5 section .text
6 global _start
7 _start:
8     mov al, 1              ; Inicializa product a 1
9     mov bl, 1              ; Inicializa i a 1
10
11 for_loop:
12     cmp bl, 5              ; Verifica si i <= 5
13     jg end_for             ; Si i > 5, termina el bucle
14
15     imul al, bl            ; Multiplica product por i
16     inc bl                 ; Incrementa i
17     jmp for_loop           ; Repite el ciclo
18
19 end_for:
20     mov [product], al      ; Guarda el resultado en product
21
22     ; Fin del programa
23
```

4.

```
1 section .data
2     num db 5               ; Número a verificar
3     result_even db 0       ; Resultado si es par
4     result_odd db 0        ; Resultado si es impar
5
6 section .text
7 global _start
8 _start:
9     mov al, [num]          ; Carga el valor de num
10    test al, 1              ; Verifica el bit menos significativo
11
12    jz is_even              ; Si el bit menos significativo es 0, es par
13    jmp is_odd              ; Si no, es impar
14
15 is_even:
16     mov [result_even], 1   ; Almacena el resultado en result_even
17     jmp end_if_else
18
19 is_odd:
20     mov [result_odd], 1    ; Almacena el resultado en result_odd
21
22 end_if_else:
23     ; Fin del programa
24
```

5.

```
1 ▾ section .data
2   | count db 10           ; Variable contador
3
4   section .text
5   global _start
6 ▾ _start:
7   | mov al, 10            ; Inicializa count en 10
8
9 ▾ for_loop:
10  | cmp al, 1             ; Verifica si count >= 1
11  | jl end_for           ; Si count < 1, termina el bucle
12
13  | ; Aquí podríamos almacenar o imprimir el valor actual de count
14  | ; (en un sistema real, podría hacerse una syscall para imprimir)
15
16  | dec al                ; Decrementa count
17  | jmp for_loop          ; Repite el ciclo
18
19 ▾ end_for:
20  | ; Fin del programa
21 |
```

6.

```

1 * section .data
2     num1 db 3          ; Primer número (puedes cambiar el valor)
3     num2 db 5          ; Segundo número (puedes cambiar el valor)
4     result db 0        ; Variable para almacenar el resultado de la suma
5     msg db "Resultado: ", 0
6     resultStr db "00", 10 ; Cadena para el resultado en ASCII y salto de línea
7     zeroMsg db "Esto es un cero", 10 ; Mensaje "Esto es un cero" con salto de línea
8
9 section .text
10 global _start
11 _start:
12     ; Realizar la suma de los dos números
13     mov al, [num1]      ; Cargar num1 en AL
14     add al, [num2]      ; Sumar num2 a AL
15     mov [result], al    ; Almacenar el resultado en la variable result
16
17     ; Verificar si el resultado es igual a 0
18     cmp al, 0
19     je print_zero_msg   ; Si el resultado es cero, saltar a print_zero_msg
20
21     ; Si el resultado no es cero, convertir a ASCII y mostrarlo
22     ; Convertir el valor de AL a ASCII
23     add al, '0'         ; Convertir el dígito de resultado a carácter ASCII
24     mov [resultStr], al ; Almacenar el carácter ASCII en resultStr
25
26     ; Imprimir el mensaje inicial "Resultado: "
27     mov eax, 4          ; Syscall para escribir (sys_write)
28     mov ebx, 1          ; Salida estándar (stdout)
29     mov ecx, msg        ; Dirección del mensaje
30     mov edx, 11         ; Longitud del mensaje
31     int 0x80            ; Llamada al sistema
32
33     ; Imprimir el resultado de la suma
34     mov eax, 4          ; Syscall para escribir (sys_write)
35     mov ebx, 1          ; Salida estándar (stdout)
36     mov ecx, resultStr  ; Dirección de la cadena del resultado
37     mov edx, 2          ; Longitud de la cadena (1 dígito y nueva línea)
38     int 0x80            ; Llamada al sistema
39
40     jmp exit_program    ; Saltar al final del programa
41
42 * print_zero_msg:
43     ; Imprimir "Esto es un cero"
44     mov eax, 4          ; Syscall para escribir (sys_write)
45     mov ebx, 1          ; Salida estándar (stdout)
46
47     mov eax, 4          ; Syscall para escribir (sys_write)
48     mov ebx, 1          ; Salida estándar (stdout)
49     mov ecx, zeroMsg    ; Dirección del mensaje "Esto es un cero"
50     mov edx, 15         ; Longitud del mensaje
51     int 0x80            ; Llamada al sistema
52
53 * exit_program:
54     ; Terminar el programa
55     mov eax, 1          ; Syscall para salir (sys_exit)
56     xor ebx, ebx        ; Código de salida 0
57     int 0x80            ; Llamada al sistema

```