


1:

HelloWorld.asm

43d3t36hn 

```
1  section.data
2  msg_igual db
3  msg_mayor
4  msg_menor
5  msg_negativo
6
7  section.bss
8  num1 resb 1
9  num2 resb 2
10
11 _start:
12 ;leer los numeros
13 ;codigo de input
14 ;comparacion
15 mov al, [num1]
16 cmp al, [num2]
17 je flag_igual
18 jl flag_menor
19 jg flag_mayor
20
21 flag_mayor
22 mov ecx, msg_mayor
23 jmp imprimir
```

2:

>HelloWorld.asm

43d3t36hn

```
1 ▾ section .data
2   | msg_positivo db "El número es positivo", 0
3   | msg_negativo db "El número es negativo", 0
4   | msg_cero db "El número es cero", 0
5
6 ▾ section .bss
7   | num resb 1 ; Espacio para el número
8
9 ▾ section .text
10  | global _start
11
12 ▾ _start:
13  | ; Leer número
14  | ; Código de entrada aquí...
15
16  | ; Comparación del número
17  | cmp al, 0
18  | je es_cero
19  | jl es_negativo
20  | jg es_positivo
21
22 ▾ es_positivo:
23  | ; Código para imprimir "El número es positivo"
24  | jmp fin
25
26 ▾ es_negativo:
27  | ; Código para imprimir "El número es negativo"
28  | jmp fin
29
30 ▾ es_cero:
31  | ; Código para imprimir "El número es cero"
32  | jmp fin
33
34 ▾ fin:
35  | ; Salir del programa
36
```

3:

```
1 section .data
2     msg_par db "El número es par", 0
3     msg_impar db "El número es impar", 0
4
5 section .bss
6     num resb 1 ; Espacio para el número
7
8 section .text
9     global _start
10
11 _start:
12     ; Leer número
13     ; Código de entrada aquí...
14
15     ; Verificar paridad
16     test al, 1 ; Verificar el bit menos significativo
17     jpe es_par ; Si PF = 1, el número es par
18     jpo es_impar ; Si PF = 0, el número es impar
19
20 es_par:
21     ; Código para imprimir "El número es par"
22     jmp fin
23
24 es_impar:
25     ; Código para imprimir "El número es impar"
26     jmp fin
27
28 fin:
29     ; Salir del programa
30
```

4:

```
1 section .data
2     msg_overflow db "overflow detectado", 0
3     msg_no_overflow db "No hay overflow", 0
4
5 section .bss
6     num1 resb 1 ; Espacio para el primer número
7     num2 resb 1 ; Espacio para el segundo número
8
9 section .text
10    global _start
11
12 _start:
13     ; Leer los números
14     ; Código de entrada aquí...
15
16     ; Realizar la suma
17     add al, bl ; AL = AL + BL
18
19     ; Verificar overflow
20     jo hubo_overflow ; Si OF = 1, hubo overflow
21     jmp sin_overflow ; Si OF = 0, no hubo overflow
22
23 hubo_overflow:
24     ; Código para imprimir "Overflow detectado"
25     jmp fin
26
27 sin_overflow:
28     ; Código para imprimir "No hay overflow"
29     jmp fin
30
31 fin:
32     ; Salir del programa
33
```

5:

```
1 section .data
2     msg_acarreo db "Hubo acarreo", 0
3     msg_no_acarreo db "No hubo acarreo", 0
4
5 section .bss
6     num1 resb 1 ; Espacio para el primer número
7     num2 resb 1 ; Espacio para el segundo número
8
9 section .text
10    global _start
11
12 _start:
13     ; Leer Los números
14     ; Código de entrada aquí...
15
16     ; Realizar la suma
17     add al, bl ; AL = AL + BL
18
19     ; Verificar acarreo
20     jc hubo_acarreo ; Si CF = 1, hubo acarreo
21     jmp no_hubo_acarreo ; Si CF = 0, no hubo acarreo
22
23 hubo_acarreo:
24     ; Código para imprimir "Hubo acarreo"
25     jmp fin
26
27 no_hubo_acarreo:
28     ; Código para imprimir "No hubo acarreo"
29     jmp fin
30
31 fin:
32     ; Salir del programa
33
```

6:

```

1 section .data
2     msg_min db "El mínimo es: ", 0
3     msg_max db "El máximo es: ", 0
4
5 section .bss
6     num1 resb 1 ; Espacio para el primer número
7     num2 resb 1 ; Espacio para el segundo número
8     num3 resb 1 ; Espacio para el tercer número
9
10 section .text
11     global _start
12
13 _start:
14     ; Leer los tres números
15     ; Código de entrada aquí...
16
17     ; Inicializar AX con el primer número (num1)
18     mov al, [num1]
19
20     ; Comparar el primer número (AX) con el segundo (num2)
21     mov bl, [num2]
22     cmp al, bl
23     jg es_mayor_1 ; Si AX > BX, AX es el mayor hasta ahora
24     mov al, bl    ; Si BX > AX, AX toma el valor de BX
25
26 es_mayor_1:
27     ; Comparar el mayor hasta ahora con el tercer número (num3)
28     mov cl, [num3]
29     cmp al, cl
30     jg es_mayor_2 ; Si AX > CX, AX sigue siendo el mayor
31     mov al, cl    ; Si CX > AX, AX toma el valor de CX
32
33 es_mayor_2:
34     ; Ahora AX contiene el máximo, se guarda en un registro (por ejemplo, DX)
35     mov dl, al    ; DL = máximo
36
37     ; Encontrar el mínimo
38     mov al, [num1] ; Reiniciar AX con el primer número
39     cmp al, bl
40     jl es_menor_1 ; Si AX < BX, AX es el menor hasta ahora
41     mov al, bl    ; Si BX < AX, AX toma el valor de BX
42
43 es_menor_1:
44     cmp al, cl
45     jl es_menor_2 ; Si AX < CX, AX sigue siendo el menor
46     mov al, cl    ; Si CX < AX, AX toma el valor de CX
47
48 es_menor_2:
49     ; Ahora AX contiene el mínimo, se guarda en un registro (por ejemplo, SI)
50     mov si, ax    ; SI = mínimo
51
52     ; Imprimir el máximo (DL) y mínimo (SI)
53     ; Código para imprimir "El máximo es: " seguido de DL
54     ; Código para imprimir "El mínimo es: " seguido de SI
55
56     jmp fin
57
58 fin:
59     ; Salir del programa

```

7:

```
1 section .data
2     msg_ordenado db "Los números están en orden ascendente", 0
3     msg_intercambiado db "Los números fueron intercambiados", 0
4
5 section .bss
6     num1 resb 1 ; Espacio para el primer número
7     num2 resb 1 ; Espacio para el segundo número
8
9 section .text
10    global _start
11
12 _start:
13     ; Leer Los dos números
14     ; Código de entrada aquí...
15
16     ; Comparar Los números
17     mov al, [num1] ; Cargar el primer número en AL
18     mov bl, [num2] ; Cargar el segundo número en BL
19
20     cmp al, bl ; Comparar Los dos números
21     jg intercambiar ; Si el primer número es mayor, intercambiar
22
23     ; Si están en orden ascendente, mostrar mensaje de orden correcto
24     mov edx, 35 ; Longitud del mensaje
25     mov ecx, msg_ordenado
26     jmp imprimir
27
28 intercambiar:
29     ; Intercambiar Los valores
30     xchg al, bl ; Intercambiar AL y BL
31
32     ; Mostrar mensaje que Los números fueron intercambiados
33     mov edx, 41 ; Longitud del mensaje
34     mov ecx, msg_intercambiado
35
36 imprimir:
37     ; Código para imprimir el mensaje
38     ; Código para imprimir el mensaje de texto
39
40     jmp fin
41
42 fin:
43     ; Salir del programa
44
```

8:

```
1 section .data
2     msg_contador db "Contador: ", 0
3     msg_nuevo_valor db "Valor: ", 0
4     newline db 0xA, 0 ; Salto de línea
5
6 section .bss
7     contador resb 1 ; Espacio para almacenar el contador
8
9 section .text
10    global _start
11
12 _start:
13     ; Inicializar el contador en 0
14     mov byte [contador], 0
15
16 ciclo_contador:
17     ; Imprimir "Contador: "
18     mov edx, 10 ; Longitud del mensaje "Contador: "
19     mov ecx, msg_contador
20     call imprimir
21
22     ; Imprimir el valor del contador
23     mov al, [contador] ; Cargar el valor del contador
24     add al, '0' ; Convertir a ASCII
25     mov edx, 1 ; Longitud del valor (1 byte)
26     mov ecx, contador ; Cargar la dirección de contador (valor)
27     call imprimir
28
29     ; Salto de línea
30     mov edx, 1
31     mov ecx, newline
32     call imprimir
33
34     ; Incrementar el contador
35     inc byte [contador] ; Incrementar el contador
36
37     ; Repetir hasta llegar a 10
38     mov al, [contador]
39     sub al, '0' ; Convertir de ASCII a número
40     cmp al, 9 ; Verificar si el contador ha llegado a 9
41     jl ciclo_contador ; Si no ha llegado a 9, continuar el ciclo
42
43 fin:
44     ; Salir del programa
45     mov eax, 1 ; sys_exit
46     xor ebx, ebx ; código de salida 0
47     int 0x80 ; llamada al sistema
48
49 ; Función para imprimir mensajes
50 imprimir:
51     mov eax, 4 ; sys_write
52     mov ebx, 1 ; file descriptor: stdout
53     int 0x80 ; llamada al sistema
54     ret
55
```