UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA ARQUITECTURA Y DISEÑO

ORGANIZACIÓN DE COMPUTADORAS

Jonatan Crespo Ragland

MARITZA GUADALUPE BANDA RANGEL

376426

CÓDIGOS ENSAMBLADOR

28/11/2024

TALLER 5

```
section .data
msg db 'imprimir input del teclado: ';Input: &#39;, 0 ; Mensaje que se mostrará antes de la
entrada

section .bss
input resb 1 ; Espacio para almacenar el carácter ingresado
sum resb 1 ; Espacio para almacenar la suma

section .text
global _start

_start:
; Mostrar mensaje en consola
mov eax, 44
mov ebx, 1
mov ecx, msg ; dirección del mensaje
mov edx, 28 ; longitud del mensaje
int 0x80

; Leer un carácter desde el teclado

mov eax, 33
mov ebx, 0
mov ecx, input ; dirección para almacenar la entrada
mov edx, 100 ; leer 1 byte (1 carácter)
int 0x80
```

STDIN

organizacion

Output:

```
input resb 1 ; Espacio para almacenar el carácter ingresado
sum resb 1 ; Espacio para almacenar la suma

section .text
global _start

_start:
; Mostrar mensaje en consola
mov eax, 4
mov ebx, 111
mov ecx, msg ; dirección del mensaje
mov edx, 28 ; longitud del mensaje
int 0x80

; Leer un carácter desde el teclado

mov eax, 3
mov ebx, 34
mov ecx, input ; dirección para almacenar la entrada
mov edx, 100 ; leer 1 byte (1 carácter)
int 0x80

; Mostrar el carácter ingresado
mov eax, 4 ; syscall número 4 es write (sys_write)
mov ebx, 17 ; descriptor de archivo 1 es stdout
mov ecx, input ; dirección del carácter
mov edx, 100 ; longitud del carácter
int 0x80 ; llamada al sistema

; Calcular la suma de los caracteres
mov al, [input]
add al, [input]
mov [sum], al ; almacenar la suma en la variable sum

; Mostrar la suma
mov eax, 4

mov ebx, 21
mov ecx, sum ; dirección de la suma
mov edx, 1 ; longitud de la suma
```

STDIN

organizacion

Output:

```
timeout: the monitored command dumped core
```

```
; Mostrar mensaje en consola
mov eax, 4
mov ebx, 1
mov ecx, msg ; dirección del mensaje
mov edx, 28 ; longitud del mensaje
int 0x80

; Leer un carácter desde el teclado

mov eax, 3
mov ebx, 0
mov ecx, input ; dirección para almacenar la entrada
mov edx, 100 ; leer 1 byte (1 carácter)
int 0x80

; Mostrar el carácter ingresado
mov eax, 4 ; syscall número 4 es write (sys_write)
mov ebx, 1 ; descriptor de archivo 1 es stdout
mov ecx, input ; dirección del carácter
mov edx, 100 ; longitud del carácter
int 0x80 ; llamada al sistema

; Calcular la suma de los caracteres
mov al, [input]
add al, [input]
mov [sum], al ; almacenar la suma en la variable sum

; Mostrar la suma
mov eax, 4

mov ebx, 1
mov ecx, sum ; dirección de la suma
mov edx, 1 ; longitud de la suma
int 0x80

; Terminar el programa
mov eax, 1
xor ebx, ebx ; código de salida 0
int 0x60
```

STDIN

organizacion

Output:

```
imprimir input del teclado: organizacion
timeout: the monitored command dumped core
```

```
1
2  section .data
3  msg db 'imprimir imput del teclado: ';Input: &#39;, 0 ; Mensaje que se mostrará antes de la
4  entrada
5
6  section .bss
7  input resb 1 ; Espacio para almacenar el carácter ingresado
8  sum resb 1 ; Espacio para almacenar la suma
9
10 section .text
11 global _start
12
13 _start:
14 ; Mostrar mensaje en consola
15 mov eax, 4
16 mov ebx, 1
17 mov ecx, msg ; dirección del mensaje
18 mov edx, 28 ; longitud del mensaje
19 int 0x80
20
21 ; Leer un carácter desde el teclado
22
23 mov eax, 3
24 mov ebx, 0
25 mov ecx, input ; dirección para almacenar la entrada
26 mov edx, 100 ; leer 1 byte (1 carácter)
27 int 0x80
28
29 ; Mostrar el carácter ingresado
30 mov eax, 4 ; syscall número 4 es write (sys_write)
31 mov ebx, 1 ; descriptor de archivo 1 es stdout
32 mov ecx, input ; dirección del carácter
33 mov edx, 100 ; longitud del carácter
34 int 0x80 ; llamada al sistema
35
36 ; Calcular la suma de los caracteres
37 mov al, [input]
38 add al, [input]
39 mov [sum], al ; almacenar la suma en la variable sum
```

STDIN

organizacion

Output:

```
imprimir imput del teclado: organizacion
```

TALLER 7

```
section .data

num1 db 5 ;numero 1 para realizar la sumatoria
num2 db 11 ;numero 2 para realizar la sumatroria
result db 0
msg db 'Resultado: ', 0;Resultado: &#39;, 0
section .bss
buffer resb 4 ; contiene el resultado de la suma con 4 bytres de espacio
section .text
global _start

_start:

mov al, [num1]
add al, [num2]
mov [result], al
; Convertir el resultado a ASCII
movzx eax, byte [result]
add eax, 48 ; Convertir el valor numérico en su correspondiente ASCII ('0' = 48);0&#39; = 48)
mov [buffer], al ; Almacenar el carácter ASCII en el buffer

mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, 11
int 0x80 ; salida del mesaje

mov eax, 4
mov ebx, 1
mov ecx, buffer
mov edx, 1 ; salida de la operacion
```

```
section .data

num1 db 5
num2 db 11
result db 0
msg db 'Resultado: ', 0;Resultado: &#39;, 0
section .bss
buffer resb 4
section .text
global _start

_start:

mov al, [num1]
add al, [num2]
mov [result], al
; Convertir el resultado a ASCII
movzx eax, byte [result]
add eax, 48 ; Convertir el valor numérico en su correspondiente ASCII ('0' = 48);0&#39; =
mov [buffer], al ; Almacenar el carácter ASCII en el buffer

mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, 11
int 0x80

mov eax, 4
mov ebx, 1
mov ecx, buffer
mov edx, 1
```

STDIN

Input for the program (Optional)

Output:

```
Resultado: @
```

```
section .data

num1 db 6 ;numero 1 para realizar la sumatoria
num2 db 11 ;numero 2 para realizar la sumatroria
result db 0
msg db 'Resultado: ', 0;Resultado: &#39;, 0
section .bss
buffer resb 4 ; contiene el resultado de la suma con 4 bytres de espacio
section .text
global _start
```

Input for the pr

Output:

```
Resultado: A
```

```
section .data

num1 db 30 ;numero 1 para realizar la sumatoria
num2 db 14 ;numero 2 para realizar la sumatroria
result db 0
msg db 'Resultado: ', 0;Resultado: &#39;, 0
section .bss
buffer resb 4 ; contiene el resultado de la suma con 4 bytres de espa
section .text
```

STDIN

Output:

Resultado: \

```
section .data

num1 db 3 ;numero 1 para realizar la sumatoria
num2 db 3 ;numero 2 para realizar la sumatroria
result db 0
msg db 'Resultado: ', 0;Resultado: &#39;, 0
section .bss
buffer resb 4 ; contiene el resultado de la suma con 4 bytres de espa
section .text
global _start

_start:

mov al, [num1]
add al, [num2]
mov [result], al
; Convertir el resultado a ASCII
movzx eax, byte [result]
add eax, 30 ; Convertir el valor numérico en su correspondiente ASCII
mov [buffer], al ; Almacenar el carácter ASCII en el buffer
```

STDIN

Output:

Resultado: $

```
section .data

num1 db 5 ;numero 1 para realizar la sumatoria
num2 db 3 ;numero 2 para realizar la sumatroria
result db 0
msg db 'Resultado: ', 0;Resultado: &#39;, 0
section .bss
buffer resb 4 ; contiene el resultado de la suma con 4 bytres de espa
section .text
global _start
```

STDIN

Output:

Resultado: &

```
section .data

num1 db 9 ;numero 1 para realizar la sumatoria
num2 db 10 ;numero 2 para realizar la sumatroria
result db 0
msg db 'Resultado: ', 0;Resultado: &#39;, 0
section .bss
buffer resb 4 ; contiene el resultado de la suma con 4 bytres de espa
section .text
global _start
```

STDIN

Output:

Resultado: 1

```
;INMEDIATO

section .data
msg db 'Resultado: ', 0
section .bss
buffer resb 1 ; contiene solo el byte de espacio para @
section .text
global _start

_start:

mov al, '@'
mov [buffer], al ; almacena el caracter directamente en al y lo envia al buffer

mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, 11
int 0x80 ; imprime el mensaje de salida "Resultado: "

mov eax, 4
mov ebx, 1
mov ecx, buffer
mov edx, 1
int 0x80  ; imprime el caracter que contiene el buffer de 1 byte llamando al sistema

mov eax, 1
xor ebx, ebx
int 0x80 ; finaliza llamando al sistema
```

STDIN

Input for the progr

Output:

```
Resultado: @
```

```
;INDIRECTO

section .data
char_at db '@' ; se guarda "@" directamente en la memoria
msg db 'Resultado: ', 0
section .bss
buffer resb 1 ; contiene solo el byte de espacio para @
section .text
global _start

_start:

mov esi, char_at ; carga la direccion de memoria a esi
mov al, [esi] ; carga el valor de la direccion a al
mov [buffer], al ; almacena el caracter directamente en al y lo envia al buffer

mov eax, 4
mov ebx, 1
mov ecx, msg
mov edx, 11
int 0x80 ; imprime el mensaje de salida "Resultado: "

mov eax, 4
mov ebx, 1
mov ecx, buffer
mov edx, 1
int 0x80  ; imprime el caracter que contiene el buffer de 1 byte llamando al sistema

mov eax, 1
xor ebx, ebx
int 0x80 ; finaliza llamando al sistema
```

STDIN

Input for the pro

Output:

```
Resultado: @
```

TALLER 8

```
section .data
msg db 'Resultado: ', 0 ; mensaje que se mostrara
newline db 0xA
section .bss
res resb 4 ; Espacio para el resultado

section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 10 ; numero que se suma
mov ebx, 5 ; segundo numero a sumar
add eax, ebx ; hace la operacion de la suma anterior 10 + 5

; Instrucción lógica (AND)
and eax, 0xF ;Realiza AND bit a bit, será 15 AND 15, el resultado será 15

; Instrucciones de manipulación de bits
shl eax, 1 ; desplaza a la izquierda la posicion / suma su valos 15 + 15 = 30

; Guardar el resultado en la sección .bss
mov [res], eax ; guarda el resultado (30) en res

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Direccion del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el número (resultado almacenado en res)
mov eax, [res] ; Cargar el resultado en EAX
add eax, '0'; Convertir el número en carácter (ASCII)
;30 + el valor de 0 en ASCII q es 48 = 78 correspondiente a "N"
mov [res], eax ; Almacenar el carácter convertido
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
```

```
; Instrucciones aritmeticas
mov eax, 2 ; numero que se suma
mov ebx, 3 ; segundo numero a sumar
add eax, ebx ; hace la operacion de la suma anterior 2 + 3

; Instrucción lógica (AND)
and eax, 0xF ;Realiza AND bit a bit, será 5 AND 5, el resultado será 5

; Instrucciones de manipulación de bits
shl eax, 1 ; desplaza a la izquierda la posicion / suma su valos 5 + 5 = 10

; Guardar el resultado en la sección .bss
mov [res], eax ; guarda el resultado (10) en res

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Direccion del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el número (resultado almacenado en res)
mov eax, [res] ; Cargar el resultado en EAX
add eax, '0'; Convertir el número en carácter (ASCII)
;10 + el valor de 0 en ASCII q es 48 = 58 correspondiente a ":"
mov [res], eax ; Almacenar el carácter convertido
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir el número
```

STDIN

Input for the pr

Output:

```
Resultado: :
```

```asm
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)
section .bss

res resb 4 ; Espacio para el resultado
section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 100 ; Coloca 100 en el registro EAX ...
mov ebx, 8 ; Coloca 8 en el registro EBX ...
add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (100 + 8 = 108) ...

; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX (108) en la memoria reservada (res) ...

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Direccion del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir 108 (resultado almacenado en 'res') ...
mov eax, [res] ; Cargar el resultado en EAX
mov [res], eax ; Almacenar el carácter convertido
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
int 0x80 ; Interrupción para imprimir el número

; Imprimir nueva línea
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, newline ; Dirección de la nueva línea
```

STDIN

Input for the pro

Output:

```
Resultado: l
```

```asm
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)
section .bss

res resb 4 ; Espacio para el resultado
section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 5 ; Coloca 5 en el registro EAX ...
mov ebx, 5 ; Coloca 5 en el registro EBX ...
add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (5 + 5 = 10) ...

; Instrucción lógica (AND)
and eax, 0xF ; Realiza AND bit a bit con 0xF (10 en decimal), EAX será 10 AND 10, el resultado será 20 ...

; Instrucciones de manipulación de bits
shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda

; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX en la memoria reservada (res)

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Direccion del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el número (resultado almacenado en 'res')
mov eax, [res] ; Cargar el resultado en EAX
add eax, '0' ; Convertir el número en carácter (ASCII) + 48 del 0 (20 + 48) ...
mov [res], eax ; Almacenar el carácter convertido
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
```

STDIN

Input for the pr

Output:

```
Resultado: D
```

```
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)
section .bss

res resb 4 ; Espacio para el resultado
section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 4 ; Coloca 10 en el registro EAX
mov ebx, 5 ; Coloca 5 en el registro EBX
add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (4 + 5 = 9)

; Instrucción lógica (AND)
and eax, 0xF ; Realiza AND bit a bit con 0xF (9 en decimal), EAX será 9 AND 9, el resultado será 18

; Instrucciones de manipulación de bits
shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda

; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX en la memoria reservada (res)

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Direccion del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el número (resultado almacenado en 'res')
mov eax, [res] ; Cargar el resultado en EAX
add eax, '0' ; Convertir el número en carácter (ASCII) + 48 del 0 (18 + 48)
mov [res], eax ; Almacenar el carácter convertido
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
```

STDIN

Input for the progr

Output:

```
Resultado: B
```

```
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)
section .bss
res resb 4 ; Espacio para el resultado
section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 1 ; Coloca 1 en el registro EAX
mov ebx, 1 ; Coloca 1 en el registro EBX
add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (1 + 1 = 2)

; Instrucción lógica (AND)
and eax, 0xF ; Realiza AND bit a bit con 0xF (2 en decimal), EAX será 2 AND 2, el resultado será 4

; Instrucciones de manipulación de bits
shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda

; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX en la memoria reservada (res)

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Direccion del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el número (resultado almacenado en 'res')
mov eax, [res] ; Cargar el resultado en EAX
add eax, '0' ; Convertir el número en carácter (ASCII) + 48 del 0 (4 + 48)
mov [res], eax ; Almacenar el carácter convertido
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
```

STDIN

Input for the prog

Output:

```
Resultado: 4
```

```asm
section .data
msg db 'Resultado: ', 0 ; Mensaje para imprimir
newline db 0xA ; Nueva línea (salto de línea)
section .bss
res resb 4 ; Espacio para el resultado
section .text
global _start

_start:

; Instrucciones aritméticas
mov eax, 1 ; Coloca 1 en el registro EAX
mov ebx, 0 ; Coloca 0 en el registro EBX
add eax, ebx ; Suma EAX + EBX, el resultado se almacena en EAX (1 + 0 = 1)

; Instrucción lógica (AND)
and eax, 0xF ; Realiza AND bit a bit con 0xF (1 en decimal), EAX será 1 AND 1, el resultado será 2

; Instrucciones de manipulación de bits
shl eax, 1 ; Desplaza los bits de EAX una posición a la izquierda

; Guardar el resultado en la sección .bss
mov [res], eax ; Almacena el valor de EAX en la memoria reservada (res)

; Llamar a la rutina para imprimir el resultado
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar (pantalla)
mov ecx, msg ; Direccion del mensaje a imprimir
mov edx, 11 ; Longitud del mensaje
int 0x80 ; Interrupción para imprimir el mensaje

; Imprimir el número (resultado almacenado en 'res')
mov eax, [res] ; Cargar el resultado en EAX
add eax, '0' ; Convertir el número en carácter (ASCII) + 48 del 0 (2 + 48)
mov [res], eax ; Almacenar el carácter convertido
mov eax, 4 ; Syscall para escribir
mov ebx, 1 ; Usar la salida estándar
mov ecx, res ; Dirección del resultado
mov edx, 1 ; Longitud de 1 carácter
```

STDIN

Input for the progra

Output:

```
Resultado: 2
```

TALLER 10

```
MOV AX, 5          ; Carga el valor 5 en el registro AX
MOV BX, 5          ; Carga el valor 5 en el registro BX
CMP AX, BX         ; Compara AX y BX (AX - BX)
JE son_iguales     ; Salta a "son_iguales" si AX == BX

MOV CX, 0          ; Si no son iguales, esta línea se ejecutará
JMP fin            ; Salto a fin

son_iguales:
MOV CX, 1          ; Si AX y BX son iguales, se ejecutará esta instrucción
```

```
MOV AX, 10         ; Carga el valor 10 en el registro AX
MOV BX, 10         ; Carga el valor 10 en el registro BX
CMP AX, BX         ; Compara AX y BX (AX - BX)
JZ son_iguales     ; Salta a "son_iguales" si el resultado es cero (AX == BX)

MOV CX, 0          ; Si AX y BX no son iguales, esta instrucción se ejecutará
JMP fin            ; Salto a "fin" para terminar

son_iguales:
MOV CX, 1          ; Si AX y BX son iguales, esta instrucción se ejecutará
```

```
MOV AX, 5          ; Carga el valor 5 en el registro AX
MOV BX, 10         ; Carga el valor 10 en el registro BX
CMP AX, BX         ; Compara AX y BX (AX - BX)
JNE no_iguales     ; Salta a "no_iguales" si AX no es igual a BX

MOV CX, 1          ; Si AX y BX son iguales, esta instrucción se ejecutará
JMP fin            ; Salta a "fin" para terminar el programa

no_iguales:
MOV CX, 0          ; Si AX y BX no son iguales, esta instrucción se ejecutará
```

```
MOV AX, 3          ; Carga el valor 3 en el registro AX
MOV BX, 5          ; Carga el valor 5 en el registro BX
CMP AX, BX         ; Compara AX y BX (AX - BX)
JNZ no_iguales     ; Salta a "no_iguales" si el resultado no es cero (AX != BX)

MOV CX, 1          ; Si AX y BX son iguales, esta instrucción se ejecutará
JMP fin            ; Salto a "fin" para terminar el programa

no_iguales:
MOV CX, 0          ; Si AX y BX no son iguales, esta instrucción se ejecutará
```

```
MOV AX, 5          ; Carga el valor 5 en el registro AX
MOV BX, 3          ; Carga el valor 3 en el registro BX
CMP AX, BX         ; Compara AX y BX (AX - BX)
JGE mayor_o_igual  ; Salta a "mayor_o_igual" si AX >= BX

MOV CX, 0          ; Si AX < BX, esta instrucción se ejecutará
JMP fin            ; Salta a "fin" para terminar el programa

mayor_o_igual:
MOV CX, 1          ; Si AX >= BX, esta instrucción se ejecutará
```

```
MOV AX, 2          ; Carga el valor 2 en el registro AX
MOV BX, 5          ; Carga el valor 5 en el registro BX
CMP AX, BX         ; Compara AX y BX (AX - BX)
JL es_menor        ; Salta a "es_menor" si AX < BX

MOV CX, 0          ; Si AX >= BX, esta instrucción se ejecutará
JMP fin            ; Salta a "fin" para terminar el programa

es_menor:
MOV CX, 1          ; Si AX < BX, esta instrucción se ejecutará
```

```
MOV AX, 3          ; Carga el valor 3 en el registro AX
MOV BX, 5          ; Carga el valor 5 en el registro BX
CMP AX, BX         ; Compara AX y BX (AX - BX)
JLE menor_o_igual  ; Salta a "menor_o_igual" si AX <= BX

MOV CX, 0          ; Si AX > BX, esta instrucción se ejecutará
JMP fin            ; Salta a "fin" para terminar el programa

menor_o_igual:
MOV CX, 1          ; Si AX <= BX, esta instrucción se ejecutará
```

```
MOV AX, -5         ; Carga el valor -5 en el registro AX
MOV BX, 3          ; Carga el valor 3 en el registro BX
CMP AX, BX         ; Compara AX y BX (AX - BX)
JS es_negativo     ; Salta a "es_negativo" si el resultado es negativo (AX < BX)

MOV CX, 0          ; Si AX >= BX, esta instrucción se ejecutará
JMP fin            ; Salto a "fin" para terminar el programa

es_negativo:
MOV CX, 1          ; Si el resultado es negativo, esta instrucción se ejecutará
```

```
MOV sum, 0 ; suma = 0
MOV cont, 1 ; cot = 1

while_loop:
CMP cont, 10  ; comparacion del cont con 10
JG end while  ; termina al tener cont > 10

ADD sum, cont ; cont + sum
INC cont  ; incremento del contador (cont)
JMP while_loop  ; hace el bucle
end while:
```

```
MOV sum, 0 ; suma = 0
MOV punt, 0 ; punt apuntando al inicio de la lista

do_while_loop:
MOV num, [punt] ; lee el numero de la lista
ADD sum, num ; suma num a sum
ADD punt, 2 ; punt pasa al siguiente num
CMP num, 0 ; verifica si num es negativo
JS end_do_while ; hace un salto si es negativo
JMP do_while_loop  ; hace el bucle
```

```
MOV produc, 1 ; product = 1
MOV i, 1 ; i = 1

for_loop:
CMP i, 5 ; i == 5
JG end_for ; salta si i > 5
MUL i ; product * i
INC i ; i++
JMP for_loop ; hace el bucle
end_for
```

```
MOV par, num ; carga el valor num
TEST par, 1 ; comprueba el bit menos significativo si
;corresponde a 0 siendo el par
JZ es_par ; si es 0, salta a es_par
MOV result_odd, 1; guarda el valor en result_odd
JMP end_if_else
es par:
MOV result_even, 1 ; guarda el valor en result_even
```

```
MOV cont, 10; cont = 10

for_loop_dec:
CMP cont, 1 ; cont == 1
JL end_for_dec ; salta si cont < 1
;(se imprime o almacena cont)
DEC cont ; decrementa cont
JMP for_loop_dec ; hace el bucle
end for_dec:
```

```
MOV sum, num1 ; carga el primer numero
ADD sum, num2; suma el segundo numero
CMP sum, 0 ; sum == 0

JE es_cero ; salta aqui, si el resultado es = 0
; (Se imprime el Resultado)
JMP end_suma
es_cero:
;("Esto es un cero")
end_suma:
```