Taller 7

```
section .data
    num1 db 6
    num2 db 11
    result db 0
    msg db 'Resultado: ', 0
```

```
Output:

Resultado: A
```

Se modifico Num1 sumando uno para que se volviera 65 que es el ASCII de A

```
section .data
    num1 db 20
    num2 db 24
    result db 0
    msg db 'Resultado: ', 0
```

```
Output:

Resultado: \
```

Modificó Num1 y Num2 para conseguir 95 que en ASCII es \

```
section .data
    num1 db 1
    num2 db 2
    result db 0
    msg db 'Resultado: ', 0
```

```
movzx eax, byte [result]
add eax, 33 ; Convertir el
mov [buffer], al ; Almacen
```

```
Output:

Resultado: $
```

Modificó Num1 y Num2 y add eax de 48 a 33 como valor de inicio para conseguir 36 que en ASCII es **$**.

```
section .data
    num1 db 3
    num2 db 2
    result db 0
    msg db 'Resultado: ', 0
```

```
Output:

Resultado: &
```

```
movzx eax, byte [result]
add eax, 33 ; Convertir el
mov [buffer], al ; Almacen
```

Modificó Num1 y Num2 y add eax de 48 a 33 como valor de inicio para conseguir 38 que en ASCII es **&**.

```
section .data
    num1 db 0
    num2 db 1
    result db 0
    msg db 'Resultado: ', 0
```

Modificó Num1 y Num2 para conseguir **1**

```
section .data
    num1 db 5
    num2 db 11
    result db 0
    msg db 'Resultado: ', 0

section .bss
    buffer resb 4

section .text
    global _start

_start:
    mov al, [num1]
    add al, [num2]
    mov [result], al

    movzx eax, byte [result]
    add eax, '0'
    mov [buffer], al


    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
    int 0x80

    mov eax, 1
    xor ebx, ebx
    int 0x80

```

El cambio de código fue el add eax de 48 a 0 para conseguir el @

Taller 10

```
section .data
    sum db 0            ; Variable para almacenar la suma
    count db 1          ; Variable para el contador (inicializado en 1)

section .text
    global _start

_start:
    ; Inicializamos sum y count
    mov al, 0           ; sum = 0
    mov [sum], al       ; Almacenamos sum en memoria
    mov al, 1           ; count = 1
    mov [count], al     ; Almacenamos count en memoria

loop_start:
    ; Cargar count en el registro AL
    mov al, [count]

    ; Comparar count con 10
    cmp al, 10
    jg loop_end         ; Si count > 10, salir del bucle

    ; Sumar count a sum
    add [sum], al       ; sum = sum + count

    ; Incrementar count
    inc byte [count]    ; count = count + 1

    ; Volver al inicio del bucle
    jmp loop_start

loop_end:
    ; El bucle termina aquí, en este punto tenemos el valor de sum con la suma final
    ; (sum = 55 si todo funciona correctamente)

    ; Salir del programa
    mov eax, 1          ; Código de salida (exit)
    xor ebx, ebx        ; Estado de salida (0)
    int 0x80            ; Llamada al sistema para salir
```

```
section .data
    lista db 5, 3, 7, 2, -1  ; Lista de números (terminada con un número negativo)
    sum db 0                 ; Variable para almacenar la suma

section .text
    global _start

_start:
    ; Inicialización
    mov al, 0               ; sum = 0
    mov [sum], al           ; Almacenar sum en memoria
    lea esi, [lista]        ; Cargar la dirección del inicio de la lista en el puntero ESI

do_while_loop:
    ; Leer el primer número desde la lista
    mov al, [esi]           ; Cargar el número actual de la lista en AL
    add [sum], al           ; sum = sum + al (sumamos el número actual)

    ; Comprobar si el número es negativo
    js  loop_end            ; Si el número es negativo (al estar en AL), salir del bucle

    ; Mover el puntero al siguiente número de la lista
    inc esi                 ; Avanzar el puntero (puntero = puntero + 1)

    ; Repetir el ciclo
    jmp do_while_loop       ; Volver al inicio del ciclo

loop_end:
    ; Aquí termina el bucle, el valor de sum tiene la suma final

    ; Finalizar el programa
    mov eax, 1              ; Código de salida (exit)
    xor ebx, ebx            ; Estado de salida (0)
    int 0x80                ; Llamada al sistema para salir
```

```
section .data
    product dd 1        ; Inicializa el valor de product a 1

section .bss
    i resd 1            ; Variable i

section .text
    global _start

_start:
    ; Inicializa i a 1
    mov dword [i], 1

    ; Bucle for (i <= 5)
for_loop:
    ; Cargar el valor de i en el registro eax
    mov eax, [i]

    ; Comparar i con 6 (si i > 5, salir del bucle)
    cmp eax, 6
    jg end_loop         ; Si i > 5, salta a end_loop

    ; Multiplicar product por i
    mov eax, [product] ; Cargar product en eax
    imul eax, [i]       ; Multiplicar product * i
    mov [product], eax ; Almacenar el resultado en product

    ; Incrementar i
    mov eax, [i]
    inc eax
    mov [i], eax        ; Guardar el nuevo valor de i

    ; Repetir el bucle
    jmp for_loop

end_loop:
    ; Aquí el resultado final de product está almacenado en la variable 'product'
    ; Se podría agregar código para imprimir el resultado si fuera necesario

    ; Finalizar el programa (salida limpia)
    mov eax, 1          ; Código de salida para Linux
    xor ebx, ebx        ; Código de salida 0
    int 0x80            ; Interrupción para salir
```

```
section .data
    num dd 7              ; Número de entrada (puedes cambiarlo)
    result_even db "El numero es par", 0  ; Mensaje para números pares
    result_odd db "El numero es impar", 0 ; Mensaje para números impares

section .bss
    result resb 50        ; Buffer para almacenar el resultado (mensaje)

section .text
    global _start

_start:
    ; Cargar el valor de num en el registro eax
    mov eax, [num]

    ; Verificar si el número es par o impar utilizando AND
    and eax, 1            ; AND con 1, mantiene solo el bit menos significativo

    ; Si el resultado es 0 (par), saltamos a result_even
    cmp eax, 0
    je  even_case         ; Si es igual a 0, es par

    ; Si no es 0, el número es impar, almacenamos el mensaje en result_odd
    mov eax, [result_odd]
    mov [result], eax     ; Guardamos el mensaje en result
    jmp end_program       ; Saltamos al final del programa

even_case:
    ; Si es par, almacenamos el mensaje en result_even
    mov eax, [result_even]
    mov [result], eax     ; Guardamos el mensaje en result

end_program:
    ; El valor de 'result' contiene el mensaje adecuado (par o impar)

    ; Finalizar el programa (salida limpia)
    mov eax, 1            ; Código de salida para Linux
    xor ebx, ebx          ; Código de salida 0
    int 0x80              ; Interrupción para salir
```

```
section .data
    msg db "Numero: %d", 10, 0  ; Mensaje de formato para imprimir, con salto de linea

section .bss
    num resb 4                  ; Buffer para almacenar el número a imprimir

section .text
    global _start

_start:
    ; Inicialización de count en 10
    mov ecx, 10                 ; count = 10

for_loop:
    ; Comprobar si count >= 1
    cmp ecx, 1
    jl loop_end                 ; Si count < 1, salir del bucle

    ; Almacenar el valor de count en el buffer num
    mov eax, [ecx]              ; Cargar el valor de count en eax (por simplicidad)

    ; Llamada al sistema para imprimir el número
    ; Convertir el número en cadena para imprimirlo
    mov [num], eax              ; Guardar el número a imprimir
    ; Llamar a una función para imprimir el valor de num

    ; Decrementar count
    dec ecx                     ; count = count - 1

    ; Continuar con el bucle
    jmp for_loop

loop_end:
    ; Terminar el programa
    mov eax, 1                  ; Código de salida (exit)
    xor ebx, ebx                ; Estado de salida (0)
    int 0x80                    ; Llamada al sistema para salir
```

```
section .data
    num1 db '3'         ; Primer número en formato de carácter (puedes cambiarlo)
    num2 db '4'         ; Segundo número en formato de carácter (puedes cambiarlo)
    msg_zero db "Esto es un cero", 0  ; Mensaje que se muestra si la suma es 0
    msg_result db "Resultado: ", 0     ; Mensaje para mostrar el resultado

section .bss
    result resb 4       ; Buffer para almacenar el resultado de la suma

section .text
    global _start

_start:
    ; Cargar el primer número (num1) en al (como un valor ASCII)
    mov al, [num1]
    sub al, '0'         ; Convertir de ASCII a número

    ; Cargar el segundo número (num2) en bl (como un valor ASCII)
    mov bl, [num2]
    sub bl, '0'         ; Convertir de ASCII a número

    ; Sumar los dos números
    add al, bl

    ; Verificar si el resultado es 0
    cmp al, 0
    je  print_zero      ; Si es igual a 0, salta a imprimir "Esto es un cero"

    ; Si no es 0, imprimir el mensaje de resultado y el valor
    mov edx, msg_result
    call PrintString    ; Llamar a la función PrintString para imprimir "Resultado: "

    ; Convertir el número a carácter ASCII
    add al, '0'         ; Convertir el número de vuelta a carácter ASCII
    mov [result], al    ; Guardar el resultado en el buffer result

    ; Llamar a PrintString para imprimir el resultado
    mov edx, result
    call PrintString

    ; Salir del programa
    jmp exit_program

print_zero:
    ; Imprimir "Esto es un cero"
    mov edx, msg_zero
    call PrintString    ; Llamar a la función PrintString para imprimir el mensaje

    ; Convertir el número a carácter ASCII
    add al, '0'         ; Convertir el número de vuelta a carácter ASCII
    mov [result], al    ; Guardar el resultado en el buffer result

    ; Llamar a PrintString para imprimir el resultado
    mov edx, result
    call PrintString

    ; Salir del programa
    jmp exit_program

; Función para imprimir una cadena (usando llamada al sistema Linux)
PrintString:
    ; edx = dirección de la cadena
    mov eax, 4          ; sys_write
    mov ebx, 1          ; salida estándar (stdout)
    int 0x80            ; llamada al sistema
    ret

exit_program:
    ; Salir del programa
    mov eax, 1          ; sys_exit
    xor ebx, ebx        ; Código de salida 0
    int 0x80            ; Llamada al sistema
```

Taller 11

```asm
section .data
    num1 db 5
    num2 db 11
    result db 0          ; Reserva un byte para almacenar el resultado de la suma
    message db "Resultado: ", 0   ; Cadena de texto que se imprimirá antes del resultado

section .bss
    buffer resb 4        ; Reserva espacio de 4 bytes en buffer (para almacenar el número como texto)

section .text
    global _start

%macro PRINT_STRING 1
                         ; Macro para imprimir una cadena de texto
    mov eax, 4           ; Número de la llamada al sistema (sys_write)
    mov ebx, 1           ; File descriptor 1 (stdout)
    mov ecx, %1          ; Dirección de la cadena a imprimir (pasada como parámetro a la macro)
    mov edx, 13          ; Longitud de la cadena (en este caso, "Resultado: " tiene 13 caracteres)
    int 0x80             ; Interrupción para realizar la llamada al sistema
%endmacro

%macro PRINT_NUMBER 1
                         ; Macro para imprimir un número (suponiendo que es de un solo dígito)
    mov eax, %1          ; Cargar el número en eax
    add eax, '0'         ; Convertir el número a su valor ASCII sumando el valor de '0' (48)
    mov [buffer], eax    ; Almacenar el valor ASCII en buffer
    mov eax, 4           ; Número de la llamada al sistema (sys_write)
    mov ebx, 1           ; File descriptor 1 (stdout)
    mov ecx, buffer      ; Dirección del buffer (donde está el número en formato ASCII)
    mov edx, 1           ; Solo un byte a imprimir (un solo dígito)
    int 0x80             ; Interrupción para realizar la llamada al sistema
%endmacro

_start:
    ; Cálculo de la suma
    mov al, [num1]       ; Cargar el valor de num1 en el registro AL (registro de 8 bits)
    add al, [num2]       ; Sumar el valor de num2 a AL (AL = AL + num2)
    mov [result], al     ; Almacenar el resultado de la suma en la variable result

                         ; Imprimir el mensaje "Resultado: "
    PRINT_STRING message ; Llamar a la macro para imprimir la cadena "Resultado: "

                          ; Imprimir el número (el resultado de la suma)
    PRINT_NUMBER [result] ; Llamar a la macro para imprimir el número almacenado en result

                          ; Salir del programa
    mov eax, 1           ; Número de la llamada al sistema (sys_exit)
    mov ebx, 0           ; Código de salida 0 (sin errores)
    int 0x80             ; Interrupción para realizar la llamada al sistema
```

```
section .data
    message db "La suma de los valores es: ", 0
    newline db 10, 0            ; Nueva línea para la salida

section .bss
    buffer resb 4               ; Buffer para convertir números a caracteres

section .text
    global _start

%macro DEFINE_VALUES 3
    ; Define una "estructura" con tres valores
    val1 db %1                  ; Primer valor
    val2 db %2                  ; Segundo valor
    val3 db %3                  ; Tercer valor
%endmacro

%macro PRINT_STRING 1
    mov eax, 4                  ; Syscall número para 'write'
    mov ebx, 1                  ; File descriptor para stdout
    mov ecx, %1                 ; Dirección del mensaje
    mov edx, 25                 ; Longitud del mensaje
    int 0x80
%endmacro

%macro PRINT_NUMBER 1
    ; Convierte un número en eax a caracteres ASCII y lo imprime
    mov eax, %1
    mov ecx, buffer             ; Usamos el buffer para guardar el resultado
    mov ebx, 10                 ; Divisor para obtener dígitos decimales

.next_digit:
    xor edx, edx                ; Limpia edx para la división
    div ebx                     ; Divide eax entre 10, cociente en eax, residuo en edx
    add dl, '0'                 ; Convierte el dígito a ASCII
    dec ecx                     ; Mueve hacia atrás en el buffer
    mov [ecx], dl               ; Almacena el dígito en el buffer
    test eax, eax               ; Verifica si quedan dígitos
    jnz .next_digit             ; Si quedan dígitos, continúa
    ; Imprime el número
    mov eax, 4                  ; Syscall para write
    mov ebx, 1                  ; Salida estándar
    mov ecx, buffer             ; Comienza en el primer dígito
    mov edx, buffer + 4         ; Longitud máxima asumida en 4 dígitos
    sub ecx, edx                ; Calcula la longitud real
    int 0x80
%endmacro

%macro PRINT_SUM 0
    ; Realiza la suma de tres valores y la imprime
    mov al, [val1]              ; Carga el primer valor en AL
    add al, [val2]              ; Suma el segundo valor
    add al, [val3]              ; Suma el tercer valor
    movzx eax, al               ; Expande AL a EAX para asegurar un valor de 32 bits

    ; Imprime el resultado de la suma
    PRINT_NUMBER eax
    PRINT_STRING newline
%endmacro

; Definimos los tres valores con la macro DEFINE_VALUES
DEFINE_VALUES 3, 5, 7

_start:
    ; Imprime el mensaje inicial
    PRINT_STRING message

    ; Imprime la suma de los valores
    PRINT_SUM

    ; Salir del programa
    mov eax, 1                  ; Syscall para 'exit'
    mov ebx, 0                  ; Código de salida
    int 0x80
```

Input for the program (Op

Output:

```
La suma de los valores es
 15
```

Taller 12

```
.MODEL SMALL          ; Modelo de memoria pequeño
.STACK 100H           ; Reservar espacio para la pila
.DATA
    ; Sección de datos
    mensaje1 DB "Ingresa el primer numero (0-9): $"
    mensaje2 DB "Ingresa el segundo numero (0-9): $"
    resultadoMsg DB "El resultado es: $"
    saltoLinea DB 13, 10, "$"   ; Carácter de nueva línea (CR LF)

    num1 DB ?   ; Para almacenar el primer número ingresado
    num2 DB ?   ; Para almacenar el segundo número ingresado
    suma DB ?   ; Para almacenar el resultado de la suma

.CODE
MAIN PROC
    ; Inicio del programa
    MOV AX, @DATA        ; Cargar la dirección del segmento de datos
    MOV DS, AX           ; Inicializar el segmento de datos

    ; Mostrar mensaje1: Solicitar el primer número
    LEA DX, mensaje1     ; Cargar la dirección del mensaje1 en DX
    MOV AH, 09H          ; Función 09h: Mostrar cadena
    INT 21H              ; Interrupción DOS para mostrar el mensaje

    ; Leer el primer número del teclado
    MOV AH, 01H          ; Función 01h: Leer un carácter del teclado
    INT 21H              ; Interrupción DOS para lectura
    SUB AL, '0'          ; Convertir de carácter ASCII a valor numérico
    MOV num1, AL         ; Guardar el valor en num1

    ; Mostrar mensaje2: Solicitar el segundo número
    LEA DX, mensaje2     ; Cargar la dirección del mensaje2 en DX
    MOV AH, 09H          ; Función 09h: Mostrar cadena
    INT 21H              ; Interrupción DOS para mostrar el mensaje

    ; Leer el segundo número del teclado
    MOV AH, 01H          ; Función 01h: Leer un carácter del teclado
    INT 21H              ; Interrupción DOS para lectura
    SUB AL, '0'          ; Convertir de carácter ASCII a valor numérico
    MOV num2, AL         ; Guardar el valor en num2

    ; Realizar la suma
    MOV AL, num1         ; Cargar el primer número en AL
```

```
42      ; Realizar la suma
43      MOV AL, num1          ; Cargar el primer número en AL
44      ADD AL, num2          ; Sumar el segundo número
45      MOV suma, AL          ; Guardar el resultado en suma
46
47      ; Mostrar un salto de línea
48      LEA DX, saltoLinea    ; Cargar la dirección del salto de línea
49      MOV AH, 09H           ; Función 09h: Mostrar cadena
50      INT 21H               ; Interrupción DOS para mostrar
51
52      ; Mostrar mensaje del resultado
53      LEA DX, resultadoMsg ; Cargar la dirección del mensaje del resultado
54      MOV AH, 09H           ; Función 09h: Mostrar cadena
55      INT 21H               ; Interrupción DOS para mostrar
56
57      ; Mostrar el resultado
58      MOV AL, suma          ; Cargar el resultado en AL
59      ADD AL, '0'           ; Convertir a carácter ASCII
60      MOV DL, AL            ; Cargar el carácter en DL
61      MOV AH, 02H           ; Función 02h: Mostrar un carácter
62      INT 21H               ; Interrupción DOS para mostrar
63
64      ; Terminar el programa
65      MOV AH, 4CH           ; Función 4Ch: Terminar el programa
66      INT 21H               ; Interrupción DOS para salida
67  MAIN ENDP
68  END MAIN
69
```