



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



**Ingeniería en Software y Tecnologías Emergentes**

**Materia: Organización de Computadoras**

**Alumno: Jesus Eduardo Rodríguez Ramírez**

**Profesor: Jonatan Crespo Ragland**

**Grupo 932**

**Trabajo: Taller 11**

**Ensenada, B.C; a 22 de noviembre del 2024**

Funcionamiento y aplicación de macros y saltos condicionales en ensamblador y da una conclusión de al menos dos párrafos que explique la diferencia entre ellos.

En lenguaje ensamblador, las macros y los saltos condicionales cumplen funciones distintas pero fundamentales para el control del flujo y la modularidad del programa.

Las macros son segmentos de código predeterminados que el ensamblador reemplazan directamente en el código fuente durante la fase de ensamblado. Esto permite reutilizar bloques de código sin la sobrecarga de llamadas a procedimientos o subrutinas. Su principal ventaja radica en la eficiencia en tiempo de ejecución, ya que el código generado es directo, sin necesidad de instrucciones adicionales para saltar o retornar. Sin embargo, las macros aumentan el tamaño del código si se usan repetidamente, ya que cada instancia se expande en el programa final.

Por otro lado, los saltos condicionales como JMP, JE, JNE, JG, entre otros, son instrucciones específicas que alteran el flujo del programa en tiempo de ejecución según el resultado de una comparación previa. Estas permiten implementar estructuras como decisiones (if-else), bucles y condiciones dinámicas.

Aunque no generan redundancia en el código, dependen del estado de los registros pero también más susceptibles a errores si no se gestionan cuidadosamente.

21/nov/24.

En conclusión, las macros facilitan la reutilización y la estructura del código a nivel de ensamblado, mientras que los saltos condicionales ofrecen control dinámico del flujo del programa.

La elección entre ambos depende del propósito: las macros son ideales para reducir la complejidad del código fuente y evitar duplicación, mientras que los saltos condicionales son esenciales para implementar lógica adaptativa y estructuras de control dentro del programa.



## Investiga la importancia del '%' en macros en ensamblador x86.

El símbolo % en el ensamblador NASM (Netwide Assembler) desempeña un papel crucial en la definición y uso de macros, permitiendo mayor flexibilidad y funcionalidad en el código. Su uso se destaca en los siguientes aspectos:

### 1. Definir macros con parámetros

- % se emplea para identificar parámetros en una macro. Al definir una macro, los parámetros se especifican utilizando %1, %2, etc., que representan los valores pasados al invocar la macro. Esto facilita la reutilización de código con diferentes valores en un mismo contexto.

### 2. Llamar a una macro

- Una macro se invoca directamente por su nombre, seguido de los parámetros necesarios. El ensamblador expande el código sustituyendo los parámetros en las posiciones definidas con %.

### 3. Macros con parámetros opcionales

- NASM permite definir macros que soporten parámetros opcionales, usando el símbolo % junto con condiciones (%if, %else, %endif). Esto permite generar variaciones del código según los argumentos proporcionados.

## Estructuras de Datos para Nuevos Tipos

En ensamblador se pueden definir estas estructuras con etiquetas para representar los campos necesarios.

Estas estructuras permiten organizar los datos de manera similar a los registros en lenguajes de alto nivel, y facilitan el acceso y manipulación de datos complejos en ensamblador.

### Fecha (dd/mm/yyyy):

```
fecha:  
db 0; Día  
db 0; Mes  
dw 0; Año
```

#### Correo electrónico:

```
section .data
correoUsuario db "usuario", 0
correoDominio db "dominio.com", 0
correoCompleto db "usuario@dominio.com", 0

section .text
    global _start

_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, correoCompleto
    mov edx, 20      ; Tamaño aproximado del correo completo
    int 0x80         ; Imprimir

    mov eax, 1
    xor ebx, ebx
    int 0x80
```

#### Dirección completa:

```
section .data
formato db "Calle: %s, Numero: %d, Colonia: %s", 0

section .text
    ; Simular concatenación y formato similar al printf
```

Curp:

```
_start:
    ; Verificar longitud de la CURP
    mov esi, curp
    xor ecx, ecx
contarCaracteres:
    lodsb
    cmp al, 0
    je mostrarCURP
    inc ecx
    jmp contarCaracteres

mostrarCURP:
    cmp ecx, 18
    jne error

    ; Imprimir CURP
    mov eax, 4
    mov ebx, 1
    mov ecx, curp
    mov edx, 18
    int 0x80

    mov eax, 1
    xor ebx, ebx
    int 0x80

error:
    ; Mostrar mensaje de error
```

## Código 1

```
1 section .data
2     num1 db 5           ; Primer número
3     num2 db 11          ; Segundo número
4     result db 0         ; Resultado de la suma
5     message db "Resultado: ", 0 ; Mensaje antes del resultado, terminado en null
6
7 section .bss
8     buffer resb 4       ; Buffer de 4 bytes para almacenamiento temporal
9
10 section .text
11     global _start      ; Punto de inicio del programa
12
13     ; Macro para imprimir una cadena
14     %macro PRINT_STRING 1
15         mov eax, 4      ; Llamada al sistema: write
16         mov ebx, 1      ; Salida estándar (stdout)
17         mov ecx, %1     ; Dirección de la cadena
18         mov edx, 13     ; Longitud de la cadena a imprimir
19         int 0x80        ; Ejecutar la llamada al sistema
20     %endmacro
21
22     ; Macro para imprimir un número
23     %macro PRINT_NUMBER 1
24         mov eax, %1     ; Carga el número a imprimir en EAX
25         add eax, '0'    ; Convierte el valor numérico a ASCII
26         mov [buffer], al ; Almacena el carácter ASCII en el buffer
27         mov eax, 4      ; Llamada al sistema: write
28         mov ebx, 1      ; Salida estándar (stdout)
29         mov ecx, buffer ; Dirección del buffer
30         mov edx, 1      ; Longitud de la cadena (1 byte)
31         int 0x80        ; Ejecutar la llamada al sistema
32     %endmacro
33
34     _start:
35         ; Realiza la suma de num1 y num2
36         mov al, [num1]   ; Carga el valor de num1 en AL
37         add al, [num2]   ; Suma el valor de num2 al contenido de AL
38         mov [result], al ; Almacena el resultado en la variable result
39
40         ; Imprime el mensaje "Resultado: "
41         PRINT_STRING message
42
43         ; Imprime el resultado de la suma
44         PRINT_NUMBER [result]
45
46         ; Salida del programa
47         mov eax, 1      ; Llamada al sistema: exit
48         xor ebx, ebx    ; Código de salida 0
49         int 0x80        ; Ejecutar la llamada al sistema
50
```

## Código 2

```
1 section .data
2     message db "La suma de los valores es: ", 0 ; Mensaje inicial
3     newline db 10, 0 ; Nueva línea
4     val1 db 3 ; Primer valor
5     val2 db 5 ; Segundo valor
6     val3 db 7 ; Tercer valor
7
8 section .bss
9     buffer resb 4 ; Buffer para la conversión de número
10
11 section .text
12     global _start
13
14     ; Macro para imprimir una cadena
15     %macro PRINT_STRING 1
16         mov eax, 4 ; Syscall para write
17         mov ebx, 1 ; Salida estándar
18         mov ecx, %1 ; Dirección de la cadena
19         mov edx, 25 ; Longitud fija para mensaje
20         int 0x80 ; Ejecuta la syscall
21     %endmacro
22
23     ; Macro para imprimir un número
24     %macro PRINT_NUMBER 1
25         mov eax, %1 ; Número a imprimir
26         mov ecx, buffer + 3 ; Apunta al final del buffer
27         mov ebx, 10 ; Divisor decimal
28
29     .next_digit:
30         xor edx, edx ; Limpia edx
31         div ebx ; Divide eax entre 10
32         add dl, '0' ; Convierte el residuo a ASCII
33         dec ecx ; Retrocede en el buffer
34         mov [ecx], dl ; Almacena el dígito en el buffer
35         test eax, eax ; Verifica si quedan dígitos
36         jnz .next_digit ; Repite si hay más dígitos
37
38     ; Imprime el número
39     mov eax, 4 ; Syscall para write
40     mov ebx, 1 ; Salida estándar
41     mov edx, buffer + 4
42     sub edx, ecx ; Longitud del número
43     mov ecx, ecx ; Dirección del número
44     int 0x80 ; Ejecuta la syscall
45     %endmacro
46
47 _start:
48     ; Imprime el mensaje inicial
49     PRINT_STRING message
50
51     ; Suma los tres valores
52     mov al, [val1] ; Primer valor
53     add al, [val2] ; Suma el segundo valor
54     add al, [val3] ; Suma el tercer valor
55     movzx eax, al ; Expande a 32 bits
56
57     ; Imprime el resultado
58     PRINT_NUMBER eax
59     PRINT_STRING newline
60
61     ; Salida del programa
62     mov eax, 1 ; Syscall para exit
63     xor ebx, ebx ; Código de salida 0
64     int 0x80 ; Ejecuta la syscall
65
```