UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
Facultad de Ingeniería Arquitectura y Diseño

Ingeniería en Software y Tecnologías Emergentes

Organización de Computadoras
Taller #12

Alumno: Andres Mendez Huerta
Profesor: Jonatan Crespo Ragland

Grupo: 932 **Matricula:** 380179

Fecha: 24 de noviembre del 2025

Investiga la importancia del '%' en macros en ensamblador x86. Por ejemplo:

En NASM, el símbolo **%** es fundamental porque le indica al ensamblador que algo pertenece al **preprocesador**, es decir, a la parte que procesa macros, constantes y condiciones *antes* de ensamblar el código real.

El ensamblador normal trabaja con instrucciones como `mov`, `add`, `int`, pero cuando ve **%**, sabe que debe hacer un trabajo diferente, NO generar instrucciones de máquina, sino transformar el código.

■ Definir macros con parámetros

Cuando defines una macro en NASM, el símbolo % es indispensable porque le indica al ensamblador que debe sustituir los parámetros que recibe la macro por los valores reales que usas cuando la llamas.

Sin el símbolo %, NASM no sabría que se trata de un parámetro y pensaría que es texto literal o una instrucción normal.

■ Llamar a una macro

En NASM, el símbolo % es fundamental porque indica que algo pertenece al preprocesador, no al conjunto de instrucciones del procesador. Las macros en NASM son manejadas por el preprocesador, por eso necesitan el uso del símbolo %
El % señala que una instrucción pertenece al preprocesador.
Todo lo que comienza con % es interpretado por el preprocesador de NASM.
Sin este símbolo, NASM vería esa línea como instrucciones de ensamblador normales y daría error.
El % se usa para declarar una macro, sin %macro y %endmacro la macro no existiría.

■ **Macros con parámetros opcionales**

En NASM, el símbolo % es esencial para que las macros con parámetros opcionales funcionen correctamente. Esto se debe a que:

1. El % permite identificar los parámetros dentro de la macro. Los parámetros se escriben como %1, %2, %3, etc.
   Cuando un parámetro es opcional, el preprocesador necesita el % para saber qué parámetro se está usando o si está vacío.

2. El % es necesario para detectar si un parámetro opcional fue usado o no.
   En NASM puedes verificar si un parámetro opcional fue pasado usando directivas del preprocesador como %if, %ifdef o %ifempty. Todas estas directivas requieren el símbolo % para funcionar.

3. El % permite asignar valores por defecto a parámetros opcionales.
   NASM no tiene “valores por defecto” automáticos, así que se simulan usando condiciones del preprocesador como:
   %ifempty %2
   ; valor por defecto
   %endif
   Sin el %, no habría forma de saber si el parámetro se dio o está vacío.

○ **En tu computadora o cuaderno, genera estructuras de datos en ensamblador, para simular nuevos tipos de datos como fecha dd/mm/yyyy, correo electrónico, dirección completa (compuesto de datos como calle, número de casa y colonia) y una cadena de texto tipo curp. Además, añade ejemplos de cómo podías acceder, manipular y objetivos de utilizar esos tipos de datos. Debes tratar de usar arreglos y/o matrices o en los ejemplos. No es necesario que se compilen sin errores.**

Fecha:

```
; Estructura Fecha (4 bytes)
; offset 0: day  db
; offset 1: month db
; offset 2: year  dw   (little-endian)  -> ocupa 2 bytes

section .data
FechaEj1:
    db 15        ; day = 15
    db 11        ; month = 11
    dw 2025      ; year = 2025

; Representación como cadena (para imprimir)
FechaStrEj1: db '15/11/2025',0
```

acceso a campos

```
; asumiendo que RSI apunta a la estructura FechaEj1
mov al, [rsi]        ; al = day
mov bl, [rsi+1]      ; bl = month
mov ax, [rsi+2]      ; ax = year (low word)
; Si quieres comparar el día:
cmp al, 1
je  _es_primero_del_mes
```

Correo electrónico:

```
section .data
   EmailUsuario:
      db "usuario.ejemplo@servidor.com", 0

   EmailMaxLength equ 64 ; Longitud máxima permitida

section .bss
   ; Arreglo de emails: reservar espacio para 10 emails
   ListaEmails resb EmailMaxLength * 10
```

Acceso al campo

```
; El índice comienza en 0. Tercer email está en el índice 2.
mov ecx, 2              ; Índice del email deseado (0-based)
mov ebx, EmailMaxLength  ; 64 bytes por email

; Calcular el desplazamiento: Desplazamiento = Índice * LongitudMax
imul ecx, ebx           ; ECX = 2 * 64 = 128

; Acceder al inicio del tercer email
; ESI (Source Index) apunta al inicio del tercer email
mov esi, ListaEmails
add esi, ecx            ; ESI ahora apunta a ListaEmails + 128
```

Dirección completa

```
section .data
   ; Constantes para offsets y tamaños
   TAM_CALLE equ 32
   TAM_NUMERO equ 4
   TAM_COLONIA equ 32
   TAM_DIRECCION equ TAM_CALLE + TAM_NUMERO + TAM_COLONIA ; 68 bytes

   DireccionPersonal:
      Calle   db "Av. Principal", 0, times (TAM_CALLE - 14) db 0 ; Rellenar hasta 32B
      Numero  db "123", 0, times (TAM_NUMERO - 4) db 0           ; Rellenar hasta 4B
      Colonia db "Centro", 0, times (TAM_COLONIA - 8) db 0       ; Rellenar hasta 32B
```

acceso al campo

```
; El campo 'Numero' comienza en el Offset 32.
mov esi, DireccionPersonal  ; ESI apunta al inicio de la estructura

; ESI + 32 ahora apunta al inicio de 'Numero'
mov esi, DireccionPersonal + TAM_CALLE ; O usar la constante: mov esi,
DireccionPersonal + 32

; Ahora se puede usar ESI para copiar la cadena '123' (e.g., usando MOVSB)
```

CURPs

```
section .data
    TAM_CURP equ 18
    NUM_REGISTROS equ 5

    ; Matriz de 5 registros CURP (5 * 18 = 90 bytes)
    MatrizCURP:
        ; Registro 1
        db "AAAA990101HDFRRC01", 0 ; 18 bytes
        ; Registro 2
        db "BBBB880202MJNXXF02", 0 ; 18 bytes
        ; ... y así sucesivamente para 5 registros

    ; Podemos definir el espacio en .bss si fuera una matriz de entrada
section .bss
    MatrizCURP_Entrada resb TAM_CURP * NUM_REGISTROS
```

acceso al campo

```
section .bss
    CURP_Temp resb TAM_CURP ; 18 bytes para copia temporal

section .text
    ; Queremos acceder al registro en el índice 2 (tercer CURP)
    mov ebx, 2          ; Índice
```

```
mov ecx, TAM_CURP    ; 18 bytes por registro

; Calcular el desplazamiento: Desplazamiento = 2 * 18 = 36
imul ebx, ecx        ; EBX = 36

; Dirección de origen (Source) y destino (Destination)
mov esi, MatrizCURP  ; ESI apunta al inicio de la matriz
add esi, ebx         ; ESI apunta al inicio del CURP deseado (offset 36)

mov edi, CURP_Temp   ; EDI apunta al destino (variable temporal)

; Copiar la cadena usando instrucciones de string (copiar 18 bytes)
mov ecx, TAM_CURP    ; Contador para el loop (18)
rep movsb            ; Repetir MOVSB (mover byte de [ESI] a [EDI]) 18 veces

; El tercer CURP ya está copiado en CURP_Temp
```

● Compila el código del siguiente enlace: https://onecompiler.com/assembly/4458pwk48 documenta en tu taller mediante comentarios en el código el funcionamiento del programa.

https://onecompiler.com/assembly/4458pwk48

● **Completa el código del siguiente enlace:**
https://onecompiler.com/assembly/4458pwk48
con el objetivo de imprimir la suma de 3 números que pertenecen a una estructura de datos. Debes utilizar al menos un macro. No es necesario que se compile sin errores. Para el repositorio de código solo es necesario que subas el código de este punto en particular.

```
%macro PRINT_NUM 1
    ; Simula la llamada a la función que convierte y escribe el número.
    mov eax, %1     ; Mueve el resultado a EAX (o al registro de retorno esperado)
    ; Llamadas reales para impresión irían aquí (sys_write, conversiones, etc.)
%endmacro

; SECCIÓN DE DATOS INICIALIZADOS (.data)**
section .data
    ; Definición de la ESTRUCTURA DE DATOS (Mi_Estructura)
    ; Contiene los 3 números a sumar
    Mi_Estructura:
        num1 dd 5    ; Primer número (Double Word = 4 bytes)
        num2 dd 10   ; Segundo número
        num3 dd 15   ; Tercer número
    msg db "La suma de los elementos es: ", 0


section .text
    global _start

_start:
    ; 1. Inicializar la suma
    mov ebx, 0         ; EBX será nuestro acumulador de suma.

    ; 2. Cargar y sumar el primer número (num1)
    mov eax, [Mi_Estructura + 0] ; Cargar num1 (offset 0)
    add ebx, eax                 ; EBX = 0 + 5 = 5

    ; 3. Cargar y sumar el segundo número (num2)
    ; num2 está en el offset 4 (Mi_Estructura + 4), ya que num1 ocupa 4 bytes
    mov eax, [Mi_Estructura + 4] ; Cargar num2 (offset 4)
    add ebx, eax                 ; EBX = 5 + 10 = 15

    ; 4. Cargar y sumar el tercer número (num3)
    mov eax, [Mi_Estructura + 8] ; Cargar num3 (offset 8)
    add ebx, eax                 ; EBX = 15 + 15 = 30

    PRINT_NUM ebx
```

https://onecompiler.com/assembly/4458pwk48