Organización De Computadoras
Gibran Andres Leon Gomez 379202
Taller 12

1. La importancia del '%' en macros
En NASM, el símbolo % es la señal para el preprocesador. Antes de que el código se convierte a lenguaje máquina, el preprocesador lee todo lo que comienza con % y lo expande o interpreta.

Definir macros con parámetros: Se usa %macro seguido del nombre y el número de parámetros. Dentro de la macro, los argumentos se referencian como %1, %2, etc. Llamar a una macro: Simplemente se escribe el nombre de la macro seguido de los valores. No se usa call.

Macros con parámetros opcionales: NASM permite gestionar parámetros variables usando %0 (que contiene la cantidad de parámetros recibidos) y directivas condicionales %if. También se pueden definir valores por defecto usando %rotate o lógica condicional.

HelloWorld.asm 445ud6zzj AI NEW ASSEMBLY RUN

```
section .data
    ; --- DEFINICIÓN DE TAMAÑOS (Simulación de Structs) ---
    ; Fecha: dd (1 byte) + mm (1 byte) + yyyy (2 bytes) = 4 bytes total
    %assign DATE_SIZE 4

    ; Curp: 18 bytes
    %assign CURP_SIZE 18

    ; Dirección: Calle (20 bytes) + Numero (2 bytes) + Colonia (20 bytes) = 42 bytes
    %assign ADDRESS_SIZE 42

    ; Estructura Completa (Usuario):
    ; Email (30 bytes) + Fecha(4) + Dirección(42) + Curp(18) = 94 bytes por persona
    %assign USER_STRUCT_SIZE 94

    ; --- CREACIÓN DE LA "MATRIZ" (Arreglo de Usuarios) ---
    ; Aquí reservamos espacio para 3 usuarios.
    ; Imagina esto como usuarios[3]
    users_array:
        times 3 * USER_STRUCT_SIZE db 0

section .text
    global _start

_start:
    ; --- EJEMPLO DE ACCESO Y MANIPULACIÓN ---

    ; OBJETIVO: Acceder al AÑO de nacimiento del SEGUNDO usuario (índice 1)

    ; 1. Cargar la dirección base del arreglo
    mov ebx, users_array

    ; 2. Movernos al segundo usuario (índice 1)
    ; Fórmula: Base + (Índice * Tamaño_Struct)
    add ebx, USER_STRUCT_SIZE  ; Ahora ebx apunta al inicio del Usuario 1

    ; 3. Movernos al campo Fecha dentro del usuario
    ; El email son los primeros 30 bytes, así que la fecha empieza en el offset 30
    add ebx, 30

    ; 4. Acceder específicamente al AÑO dentro de la fecha
    ; dd(1 byte) + mm(1 byte) = offset 2 bytes dentro de la fecha
    mov ax, [ebx + 2]  ; Cargamos el año en AX
```

```
    ; --- DEFINICIÓN DE TAMAÑOS (Simulación de Structs) ---
    ; Fecha: dd (1 byte) + mm (1 byte) + yyyy (2 bytes) = 4 bytes total
    %assign DATE_SIZE 4

    ; Curp: 18 bytes
    %assign CURP_SIZE 18

    ; Dirección: Calle (20 bytes) + Numero (2 bytes) + Colonia (20 bytes) = 42 bytes
    %assign ADDRESS_SIZE 42

    ; Estructura Completa (Usuario):
    ; Email (30 bytes) + Fecha(4) + Dirección(42) + Curp(18) = 94 bytes por persona
    %assign USER_STRUCT_SIZE 94

    ; --- CREACIÓN DE LA "MATRIZ" (Arreglo de Usuarios) ---
    ; Aquí reservamos espacio para 3 usuarios.
    ; Imagina esto como usuarios[3]
    users_array:
        times 3 * USER_STRUCT_SIZE db 0

section .text
    global _start

_start:
    ; --- EJEMPLO DE ACCESO Y MANIPULACIÓN ---

    ; OBJETIVO: Acceder al AÑO de nacimiento del SEGUNDO usuario (índice 1)

    ; 1. Cargar la dirección base del arreglo
    mov ebx, users_array

    ; 2. Movernos al segundo usuario (índice 1)
    ; Fórmula: Base + (Índice * Tamaño_Struct)
    add ebx, USER_STRUCT_SIZE  ; Ahora ebx apunta al inicio del Usuario 1

    ; 3. Movernos al campo Fecha dentro del usuario
    ; El email son los primeros 30 bytes, así que la fecha empieza en el offset 30
    add ebx, 30

    ; 4. Acceder específicamente al AÑO dentro de la fecha
    ; dd(1 byte) + mm(1 byte) = offset 2 bytes dentro de la fecha
    mov ax, [ebx + 2]  ; Cargamos el año en AX

    ; --- OBJETIVO DE USO ---
    ; Al usar estas estructuras contiguas, podemos iterar sobre un arreglo de
    ; "Personas" simplemente sumando 94 (USER_STRUCT_SIZE) al puntero en cada
    ; vuelta de un bucle, permitiendo procesar nóminas, listas de asistencia, etc.
```

HelloWorld.asm 445ud6zzj AI NEW ASSEMBLY RUN

```
section .data
    ; Estructura tipo "Triplete" o Vector (x, y, z)
    ; Cada elemento es un byte para simplificar la suma
    triplet db 10, 20, 30

    message db "La suma de los valores del triplete (10+20+30) es: ", 0
    newline db 10       ; Salto de línea

section .bss
    buffer resb 4       ; Espacio para convertir números a texto
    res_sum resb 1      ; Espacio para guardar el resultado de la suma

; ------------------------------------------------------------
; MACRO 1: SUM_TRIPLET
; Realiza la operación solicitada: Sumar cada una de las X de la estructura
; Recibe: La dirección de memoria de la estructura
; Devuelve: El resultado en el registro AL
; ------------------------------------------------------------
%macro SUM_TRIPLET 1
    mov ebx, %1         ; Mover la dirección de la estructura a EBX
    mov al, 0           ; Limpiar acumulador

    add al, [ebx]       ; Sumar el primer elemento (índice 0)
    add al, [ebx+1]     ; Sumar el segundo elemento (índice 1)
    add al, [ebx+2]     ; Sumar el tercer elemento (índice 2)
%endmacro

; ------------------------------------------------------------
; MACRO 2: PRINT_STRING (Auxiliar para tu plantilla)
; Imprime una cadena terminada en null o longitud fija
; ------------------------------------------------------------
%macro PRINT_STRING 1
    mov eax, 4          ; sys_write
    mov ebx, 1          ; stdout
    mov ecx, %1         ; mensaje
    mov edx, 45         ; longitud estimada para el ejemplo
    int 0x80
%endmacro

section .text
    global _start

_start:
    ; 1. Imprime el mensaje inicial usando tu macro
    PRINT_STRING message

    ; 2. Usar la MACRO PRINCIPAL para operar la estructura
    SUM_TRIPLET triplet

    ; El resultado está en AL (60).
```

```
    add al, [ebx+2]     ; Sumar el tercer elemento (índice 2)
%endmacro

; ---------------------------------------------------------
; MACRO 2: PRINT_STRING (Auxiliar para tu plantilla)
; Imprime una cadena terminada en null o longitud fija
; ---------------------------------------------------------
%macro PRINT_STRING 1
    mov eax, 4          ; sys_write
    mov ebx, 1          ; stdout
    mov ecx, %1         ; mensaje
    mov edx, 45         ; longitud estimada para el ejemplo
    int 0x80
%endmacro

section .text
    global _start

_start:
    ; 1. Imprime el mensaje inicial usando tu macro
    PRINT_STRING message

    ; 2. Usar la MACRO PRINCIPAL para operar la estructura
    SUM_TRIPLET triplet

    ; El resultado está en AL (60).
    ; Para imprimirlo visualmente rápido como caracter (para demo):
    ; El 60 en ASCII es '<', así que verás ese símbolo.
    ; Si quisieras ver el número "60", requeriría una rutina de conversión compleja.
    ; Para cumplir el requisito, guardamos el resultado en memoria e imprimimos.
    mov [res_sum], al

    ; Imprimimos el resultado (raw byte)
    mov eax, 4
    mov ebx, 1
    mov ecx, res_sum
    mov edx, 1
    int 0x80

    ; Imprimir nueva línea
    mov eax, 4
    mov ecx, newline
    mov edx, 1
    int 0x80

    ; 3. Salir del programa
    mov eax, 1                  ; Syscall para 'exit'
```