UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTADA DE INGENIERÍA ARQUITECTURA Y DISEÑO

INGENIERÍA EN SOFTWARE Y TECNOLOGÍAS EMERGENTES

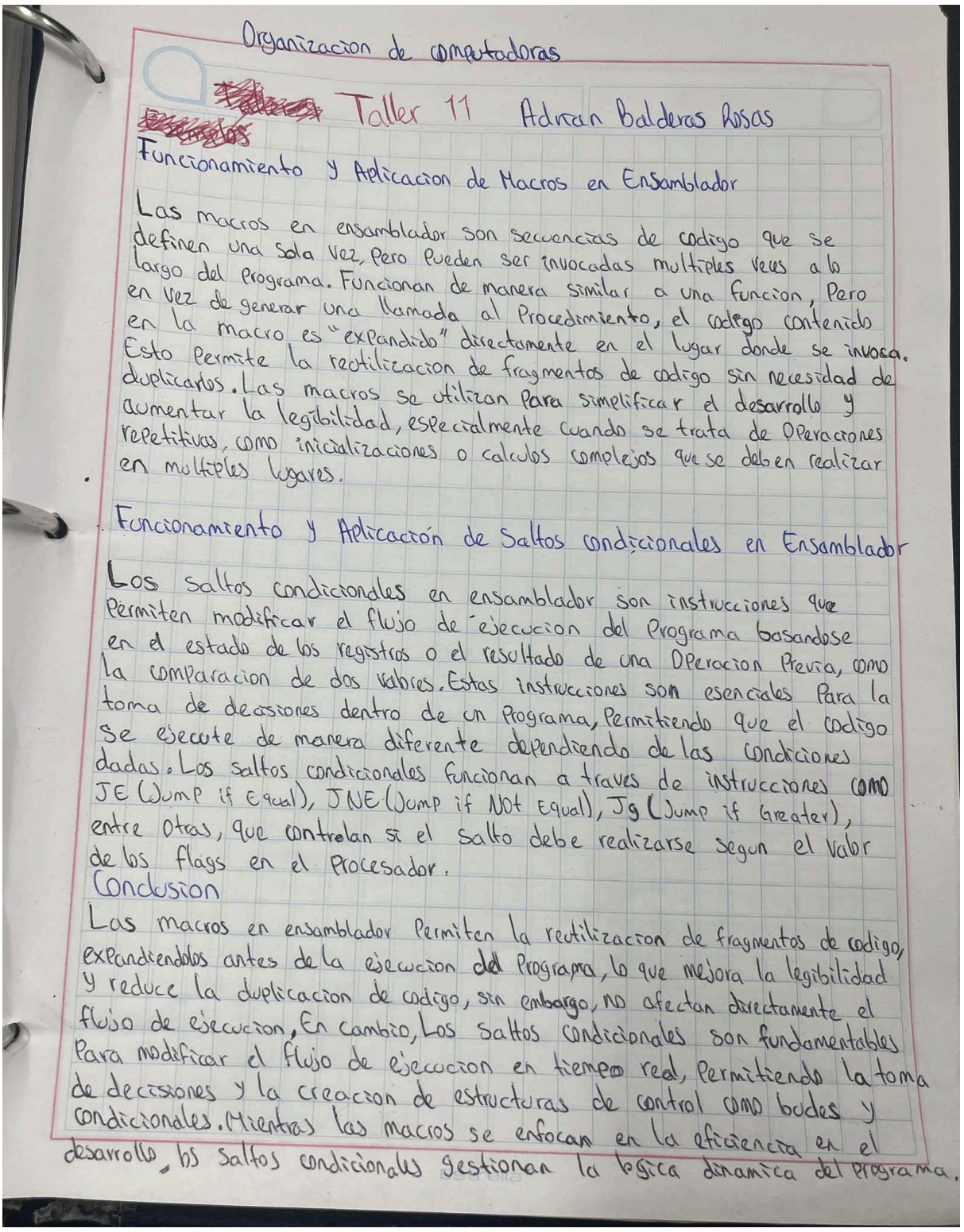
ORGANIZACIÓN DE COMPUTADORAS

Taller 11

ADRIAN BALDERAS ROSAS

Jonatan Crespo Ragland

932


Organizacion de computadoras

Taller 11 Adrian Balderas Rosas

Funcionamiento y Aplicacion de Macros en Ensamblador

Las macros en ensamblador son secuencias de codigo que se definen una sola vez, pero pueden ser invocadas multiples veces a lo largo del programa. Funcionan de manera similar a una funcion, pero en vez de generar una llamada al Procedimiento, el codigo contenido en la macro es "expandido" directamente en el lugar donde se invoca. Esto permite la reutilizacion de fragmentos de codigo sin necesidad de duplicarlos. Las macros se utilizan para simplificar el desarrollo y aumentar la legibilidad, especialmente cuando se trata de Operaciones repetitivas, como inicializaciones o calculos complejos que se deben realizar en multiples lugares.

Funcionamiento y Aplicación de Saltos condicionales en Ensamblador

Los saltos condicionales en ensamblador son instrucciones que permiten modificar el flujo de ejecucion del Programa basandose en el estado de los registros o el resultado de una Operacion Previa, como la comparacion de dos valores. Estas instrucciones son esenciales para la toma de decisiones dentro de un Programa, Permitiendo que el codigo se ejecute de manera diferente dependiendo de las condiciones dadas. Los saltos condicionales funcionan a traves de instrucciones como JE (Jump if Equal), JNE (Jump if NOt Equal), Jg (Jump if Greater), entre otras, que controlan si el salto debe realizarse segun el valor de los flags en el Procesador.

Conclusion

Las macros en ensamblador Permiten la reutilizacion de fragmentos de codigo, expandiendolos antes de la ejecucion del Programa, lo que mejora la legibilidad y reduce la duplicacion de codigo, sin embargo, no afectan directamente el fluso de ejecucion, En cambio, Los saltos condicionales son fundamentales Para modificar el fluso de ejecucion en tiempo real, Permitiendo la toma de decisiones y la creacion de estructuras de control como bucles y condicionales. Mientras las macros se enfocan en la eficiencia en el desarrollo, los saltos condicionales gestionan la logica dinamica del programa.

Importancia del '%' en Macros en Ensamblador x86

En el ensamblador x86, el símbolo % tiene una función esencial cuando se trabaja con macros. Este símbolo se utiliza para definir parámetros dentro de una macro, permitiendo que el programador especifique valores específicos en cada invocación de la macro, lo cual aumenta la versatilidad y personalización del código.

Al utilizar % en macros, se puede definir una plantilla general de código y luego ajustar su comportamiento sin necesidad de reescribir el código repetidamente.

Esto resulta útil para macros con parámetros opcionales, ya que con % se pueden establecer valores predeterminados o personalizados, dependiendo de si se pasa un valor o no al invocar la macro.

Definición de una macro con parámetros:

```
%macro add_two_numbers 2

mov eax, %1

add eax, %2

%endmacro
```

Llamada a la macro:

```
add_two_numbers 5, 10 ; Suma 5 y 10, y el resultado estará en EAX
```

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
```

Acceso y manipulación: La fecha puede leerse y modificarse cargando los valores de día, mes y año en registros para operaciones de cálculo.

Correo Electrónico:

```
email:

 db 'example@domain.com', 0 ; Cadena terminada en null
```

Objetivo: Facilitar el almacenamiento y validación de un correo electrónico en una aplicación que lo requiera.

Dirección Completa:

direccion:

calle db 'Calle', 0

numero db '123', 0

colonia db 'Centro', 0

Acceso: Se puede manipular cada campo de la dirección accediendo a las etiquetas calle, numero, y colonia.

CURP:

curp db 'ABCD010101HDFRLN09', 0 ; CURP simulada

Objetivo: Permitir la verificación de identidad o generación de registros.

Código 1

section .data

num1 db 5 **; Define el primer número, 5, almacenado en una variable de 1 byte** num2 db 11 **; Define el segundo número, 11, almacenado en una variable de 1 byte** result db 0 **;**

Variable para almacenar el resultado de la suma, inicialmente en 0

message db "Resultado: ", 0 **; Mensaje a mostrar antes del resultado, seguido de un terminador null**

section .bss

buffer resb 4 **; Reserva un buffer de 4 bytes en la sección .bss para almacenar datos** temporales

section .text

global _start **; Define la etiqueta _start como el punto de inicio del programa**

; Macro para imprimir una cadena

%macro PRINT_STRING 1

mov eax, 4 **; Llamada al sistema para escribir (syscall número 4 en Linux)** mov ebx, 1 **; Descriptor de archivo 1 (stdout) para la salida estándar** mov ecx, %1 **; La dirección de la cadena que se pasará como argumento a la macro** mov edx, 13 **; Longitud de la cadena que se va a imprimir**

int 0x80 **; Llama a la interrupción 0x80 para ejecutar la llamada al sistema**

%endmacro

; Macro para imprimir un número

%macro PRINT_NUMBER 1

mov eax, %1 **; Carga el número que se desea imprimir en el registro EAX** add eax, '0' **; Convierte el valor numérico a su equivalente en ASCII**

mov [buffer], eax **; Almacena el valor ASCII en el buffer**

mov eax, 4 **; Llamada al sistema para escribir**

mov ebx, 1 **; Descriptor de archivo 1 (stdout) para la salida estándar** mov ecx, buffer **; Dirección del buffer que contiene el número en formato ASCII** mov edx, 1 **; Longitud del dato a imprimir (1 byte)**

int 0x80 **; Llama a la interrupción 0x80 para ejecutar la llamada al sistema**

%endmacro

_start:

; Realiza la suma de los valores en num1 y num2

mov al, [num1] **; Carga el valor de num1 en el registro AL**

add al, [num2] **; Suma el valor de num2 al valor en AL**

mov [result], al **; Almacena el resultado de la suma en la variable result**

; Imprime el mensaje de texto "Resultado: "

PRINT_STRING message ; Llama a la macro PRINT_STRING para imprimir el mensaje

; Imprime el resultado de la suma

PRINT_NUMBER [result] ; Llama a la macro PRINT_NUMBER para imprimir el valor almacenado en result

; Salir del programa

mov eax, 1 **; Llamada al sistema para salir del programa (syscall número 1 en Linux)** mov

ebx, 0 **; Código de salida 0 (sin errores)**

int 0x80 **; Llama a la interrupción 0x80 para ejecutar la salida del programa**

Macros: Se definen dos macros para imprimir mensajes y números, que ayudan a mantener el código limpio y evitan repetición.

Operación Principal: La suma de num1 y num2 se almacena en result y luego se imprime usando las macros.

Salida: El programa usa llamadas al sistema de Linux (int 0x80) para manejar la impresión y la salida del programa.

Código 2

section .data

message db "La suma de los valores es: **", 0 ; Mensaje inicial para mostrar**

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
; Macro para imprimir una cadena de caracteres
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 ; Ejecuta la syscall
%endmacro

%macro PRINT_NUMBER 1
; Convierte un número en eax a caracteres ASCII y lo imprime
mov eax, %1 ; Carga el número a imprimir en eax  mov ecx,
buffer + 3 ; Apunta al final del buffer
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

; Calcula la longitud del número en el buffer
mov edx, buffer + 4 ; Posición final del buffer
```

```
sub edx, ecx ; Calcula la longitud real del número

; Imprime el número

mov eax, 4 ; Syscall para write
mov ebx, 1 ; Salida estándar

mov ecx, ecx ; Dirección inicial en el buffer

int 0x80 ; Ejecuta la syscall

%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 **; Ejecuta la syscall para salir del programa**