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 10

Ensenada, B.C; a 12 de noviembre del 2024

Investigar sobre los siguientes saltos condicionales, incluyendo su descripción, funcionamiento y un ejemplo de su uso al utilizarlos luego de una instrucción CMP.

Instrucción	Descripción	Funcionamiento	Ejemplo
JE	Salta a una dirección específica si los valores comparados son iguales.	Después de CMP, si los valores son iguales, el flag Zero (ZF) se activa (ZF = 1). JE verifica ZF y salta si es 1.	CMP AX, BX ; Compara AX con BX JE Label ; Salta a "Label" si AX es igual a BX
JZ	Realiza un salto si el flag Zero (ZF) es 1, indicando que el resultado de la comparación fue cero.	Similar a JE, revisa si ZF está en 1. Se puede usar indistintamente con JE después de CMP.	CMP AX, 0 ; Compara AX con 0 JZ Label ; Salta a "Label" si AX es igual a 0 (ZF=1)
JNE	Salta a una dirección específica si los valores comparados no son iguales.	Tras CMP, si los valores no coinciden, ZF se establece en 0. JNE verifica si ZF es 0 y salta si es así.	CMP AX, BX ; Compara AX con BX JNE Label ; Salta a "Label" si AX no es igual a BX (ZF=0)
JNZ	Realiza un salto si el flag ZF está en 0.	Similar a JNE, depende de ZF en 0 para realizar el salto.	CMP AX, 0 ; Compara AX con 0 JNZ Label ; Salta a "Label" si AX no es 0 (ZF=0)
JGE	Salta si el primer valor es mayor o igual al segundo.	Después de CMP, revisa los flags Sign (SF) y Overflow (OF). Salta si SF = OF.	CMP AX, BX ; Compara AX con BX JGE Label ; Salta a "Label" si AX >= BX (SF = OF)
JL	Realiza un salto si el primer valor es menor que el segundo.	Tras CMP, revisa SF y OF. Salta si SF ≠ OF.	CMP AX, BX ; Compara AX con BX JL Label ; Salta a "Label" si AX < BX (SF ≠ OF)
JLE	Salta si el primer valor es menor o igual al segundo.	Comprueba si ZF es 1 o si SF ≠ OF, indicando que el primer valor es menor o igual al segundo.	CMP AX, BX ; Compara AX con BX JLE Label ; Salta a "Label" si AX <= BX
JS	Salta si el flag Sign (SF) está en 1, indicando un resultado negativo.	Tras CMP, revisa SF. Salta si SF está en 1.	CMP AX, BX ; Compara AX con BX JS Label ; Salta a "Label" si el resultado fue negativo (SF=1)

En tu cuaderno o computadora, desarrolla los siguientes ejercicios en ensamblador x86 (no es necesario que compile el código)

EJERCICIO 1: Simular un bucle while

```
section .data
    sum db 0          ; Variable para almacenar la suma
    count db 1        ; Variable contador

section .text
global _start
_start:
    mov al, 0         ; Inicializa sum a 0
    mov bl, 1         ; Inicializa count a 1

while_loop:
    cmp bl, 10        ; Verifica si count <= 10
    jg end_while      ; Si count > 10, termina el bucle

    add al, bl        ; Suma count a sum
    inc bl            ; Incrementa count
    jmp while_loop    ; Repite el ciclo

end_while:
    mov [sum], al     ; Guarda el resultado en sum
```

EJERCICIO 2: Simular un ciclo do-while

```
section .data
    lista db 5, 7, 3, -1, 8  ; Lista de números
    sum db 0                 ; Variable para almacenar la suma

section .text
global _start
_start:
    mov al, 0                ; Inicializa sum a 0
    mov si, lista            ; Apunta al inicio de la lista

do_while_loop:
    mov bl, [si]             ; Carga el número actual de la lista
    add al, bl               ; Añade el número a sum
    cmp bl, 0
    js end_do_while          ; Si el número es negativo, termina el bucle

    inc si                   ; Mueve el puntero al siguiente número
    jmp do_while_loop        ; Repite el ciclo

end_do_while:
    mov [sum], al            ; Guarda el resultado en sum
```

EJERCICIO 3: Simular un bucle for

```
section .data
    product db 1        ; Variable para almacenar el producto
    i db 1              ; Variable contador

section .text
global _start
_start:
    mov al, 1           ; Inicializa product a 1
    mov bl, 1           ; Inicializa i a 1

for_loop:
    cmp bl, 5           ; Verifica si i <= 5
    jg end_for          ; Si i > 5, termina el bucle

    imul al, bl         ; Multiplica product por i
    inc bl              ; Incrementa i
    jmp for_loop        ; Repite el ciclo

end_for:
    mov [product], al   ; Guarda el resultado en product
```

EJERCICIO 4: Simular una estructura if-else

```
section .data
    num db 5                ; Número a verificar
    result_even db 0        ; Resultado si es par
    result_odd db 0         ; Resultado si es impar

section .text
global _start
_start:
    mov al, [num]           ; Carga el valor de num
    test al, 1              ; Verifica el bit menos significativo

    jz is_even              ; Si el bit menos significativo es 0, es par
    jmp is_odd              ; Si no, es impar

is_even:
    mov [result_even], 1    ; Almacena el resultado en result_even
    jmp end_if_else

is_odd:
    mov [result_odd], 1     ; Almacena el resultado en result_odd

end_if_else:
```

EJERCICIO 5: Bucle for con decremento

```
section .data
    count db 10         ; Variable contador

section .text
global _start
_start:
    mov al, 10          ; Inicializa count en 10

for_loop:
    cmp al, 1           ; Verifica si count >= 1
    jl end_for          ; Si count < 1, termina el bucle

    ; Aquí podríamos almacenar o imprimir el valor actual de count
    ; (en un sistema real, podría hacerse una syscall para imprimir)

    dec al              ; Decrementa count
    jmp for_loop        ; Repite el ciclo

end_for:
```

Realiza un código en ensamblador x86 que imprima la suma de dos números positivos de un solo carácter cada uno (0 - 9), pero, si el resultado de la suma de los dos números es igual a 0, debe imprimir Esto es un cero.

```
section .data
    num1 db 3             ; Primer número (puedes cambiar el valor)
    num2 db 5             ; Segundo número (puedes cambiar el valor)
    result db 0           ; Variable para almacenar el resultado de la suma
    msg db "Resultado: ", 0
    resultStr db "00", 10 ; Cadena para el resultado en ASCII y salto de línea
    zeroMsg db "Esto es un cero", 10 ; Mensaje "Esto es un cero" con salto de línea

section .text
global _start
_start:
    ; Realizar la suma de los dos números
    mov al, [num1]        ; Cargar num1 en AL
    add al, [num2]        ; Sumar num2 a AL
    mov [result], al      ; Almacenar el resultado en la variable result

    ; Verificar si el resultado es igual a 0
    cmp al, 0
    je print_zero_msg     ; Si el resultado es cero, saltar a print_zero_msg

    ; Si el resultado no es cero, convertir a ASCII y mostrarlo
    ; Convertir el valor de AL a ASCII
    add al, '0'           ; Convertir el dígito de resultado a carácter ASCII
    mov [resultStr], al   ; Almacenar el carácter ASCII en resultStr

    ; Imprimir el mensaje inicial "Resultado: "
    mov eax, 4            ; Syscall para escribir (sys_write)
    mov ebx, 1            ; Salida estándar (stdout)
    mov ecx, msg          ; Dirección del mensaje
    mov edx, 11           ; Longitud del mensaje
    int 0x80              ; Llamada al sistema

```

```
    ; Imprimir el resultado de la suma
    mov eax, 4          ; Syscall para escribir (sys_write)
    mov ebx, 1          ; Salida estándar (stdout)
    mov ecx, resultStr  ; Dirección de la cadena del resultado
    mov edx, 2          ; Longitud de la cadena (1 dígito y nueva línea)
    int 0x80            ; Llamada al sistema

    jmp exit_program    ; Saltar al final del programa

print_zero_msg:
    ; Imprimir "Esto es un cero"
    mov eax, 4          ; Syscall para escribir (sys_write)
    mov ebx, 1          ; Salida estándar (stdout)
    mov ecx, zeroMsg    ; Dirección del mensaje "Esto es un cero"
    mov edx, 15         ; Longitud del mensaje
    int 0x80            ; Llamada al sistema

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