UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTADA DE INGENIERÍA ARQUITECTURA Y DISEÑO

INGENIERÍA EN SOFTWARE Y TECNOLOGÍAS EMERGENTES

Organización de computadoras

Taller 10

ADRIAN BALDERAS ROSAS

Jonatan Crespo Ragland

932

Ejercicio 1: Simular un bucle `while` para sumar del 1 al 10

```
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

    ; Fin del programa (en un sistema real, aquí podría hacerse una syscall para salir)
```

Ejercicio 2: Simular un bucle `do-while` para sumar hasta encontrar un número negativo

```
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

    ; Fin del programa
```

Ejercicio 3: Simular un bucle `for` para multiplicar del 1 al 5

```
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

    ; Fin del programa
```

Ejercicio 4: Simular una estructura `if-else` para verificar si un número es par o impar

```
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:
    ; Fin del programa
```

Ejercicio 5: Bucle `for` con decremento para contar del 10 al 1

```
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:
    ; Fin del programa
```

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. Puedes usar el fragmento de código de prueba siguiente como referencia.

```
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
```