



**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 7

Ensenada, B.C; a 29 de octubre del 2024

Desarrollar lo siguiente en su cuaderno o computadora:

1. De acuerdo al código de prueba 1, responde y desarrolla lo siguiente:

a. Agrega comentarios en el código explicando su funcionamiento.

```
1 section .data
2     num1 db 5          ; Declarar un byte llamado num1 con el valor 5
3     num2 db 11         ; Declarar un byte llamado num2 con el valor 11
4     result db 0        ; Declarar un byte llamado result inicializado a 0
5     msg db 'Resultado: ', 0 ; Mensaje para mostrar en la salida
6
7 section .bss
8     buffer resb 4      ; Reservar 4 bytes sin inicializar para el buffer
9
10 section .text
11     global _start
12
13 _start:
14     mov al, [num1]      ; Mover el valor de num1 al registro AL
15     add al, [num2]      ; Sumar el valor de num2 al registro AL
16     mov [result], al   ; Almacenar el resultado de la suma en result
17
18     ; Convertir el resultado numérico a su valor ASCII
19     movzx eax, byte [result] ; Mover el byte en result a EAX, extendiéndolo a 32 bits
20     add eax, 48         ; Convertir el valor numérico a su equivalente ASCII ('0' = 48)
21     mov [buffer], al    ; Almacenar el carácter ASCII en buffer
22
23     ; Imprimir el mensaje "Resultado: "
24     mov eax, 4
25     mov ebx, 1
26     mov ecx, msg
27     mov edx, 11
28     int 0x80           ; Llamada al sistema para escribir en salida estándar
29
30     ; Imprimir el valor en buffer (resultado en ASCII)
31     mov eax, 4
32     mov ebx, 1
33     mov ecx, buffer
34     mov edx, 1
35     int 0x80           ; Llamada al sistema para escribir en salida estándar
36
37     ; Salir del programa
38     mov eax, 1
39     xor ebx, ebx
40     int 0x80
```

b. ¿Para qué sirve la instrucción movzx? La instrucción movzx copia el valor del byte de result al registro EAX y extiende este valor a 32 bits, rellendo con ceros los bits más significativos. Esto permite trabajar con un valor expandido de result, útil para realizar operaciones posteriores como la conversión a ASCII en este caso.

c. Está usando algún modo de direccionamiento. Si o no?
Que modo de direccionamiento está utilizando. Sí, el programa está usando el modo de direccionamiento directo. Se accede a los valores de num1, num2 y result directamente a través de sus direcciones, lo cual permite referenciar directamente las ubicaciones de memoria en el código.

d. Explica qué imprime el programa y por qué. El programa imprime "Resultado: 16". Esto se debe a que suma num1 (5) y num2 (11), obteniendo 16 como resultado. Luego, convierte el valor de 16 a su representación ASCII y lo imprime en pantalla junto con el mensaje "Resultado: ".

e. Modifica el programa para que imprima lo siguiente: A, \, \$, & y 1. Documenta tu procedimiento.

```
1 section .data
2     ; Valores para obtener los caracteres A, \, $, & y 1
3     numA db 65      ; Valor ASCII para 'A'
4     numBack db 92   ; Valor ASCII para '\'
5     numDollar db 36 ; Valor ASCII para '$'
6     numAmp db 38    ; Valor ASCII para '&'
7     numOne db 49    ; Valor ASCII para '1'
8     msg db 'Resultado: ', 0
9
10 section .bss
11     buffer resb 1   ; Reservar 1 byte para almacenar cada carácter temporalmente
12
13 section .text
14     global _start
15
16 _start:
17     ; Imprimir el mensaje "Resultado: "
18     mov eax, 4
19     mov ebx, 1
20     mov ecx, msg
21     mov edx, 11
22     int 0x80        ; Llamada al sistema para escribir en salida estándar
23
24     ; Imprimir cada símbolo secuencialmente
25
26     ; Cargar y mostrar 'A'
27     mov al, [numA]  ; Mover el valor de 'A' al registro AL
28     mov [buffer], al ; Almacenar el valor en buffer
29     call print_buffer
30
31     ; Cargar y mostrar '\'
32     mov al, [numBack] ; Mover el valor de '\' al registro AL
33     mov [buffer], al ; Almacenar el valor en buffer
34     call print_buffer
35
36     ; Cargar y mostrar '$'
37     mov al, [numDollar] ; Mover el valor de '$' al registro AL
38     mov [buffer], al ; Almacenar el valor en buffer
39     call print_buffer
40
41     ; Cargar y mostrar '&'
42     mov al, [numAmp] ; Mover el valor de '&' al registro AL
43     mov [buffer], al ; Almacenar el valor en buffer
44     call print_buffer
45
46     ; Cargar y mostrar '1'
47     mov al, [numOne] ; Mover el valor de '1' al registro AL
48     mov [buffer], al ; Almacenar el valor en buffer
49     call print_buffer
50
51     ; Salir del programa
52     mov eax, 1
53     xor ebx, ebx
54     int 0x80
55
56     ; Subrutina para imprimir el contenido de buffer
57 print_buffer:
58     mov eax, 4      ; Syscall número 4 (sys_write)
59     mov ebx, 1      ; File descriptor 1 (stdout)
60     mov ecx, buffer ; Dirección del buffer a imprimir
61     mov edx, 1      ; Número de bytes a escribir
62     int 0x80        ; Llamada al sistema
63     ret
```

f. Después de terminar el punto anterior, contesta lo siguiente: ¿Fue la única forma de modificar el código para llegar a esos resultados? ¿Qué otra línea pudiste modificar para llegar a los mismos resultados? Otra forma sería ajustar directamente el valor de result en cada ejecución. Cambiando el valor de result en lugar de num1 y num2, lograríamos el mismo efecto sin cambiar el flujo del código.

g. Utilizando de nuevo el código de prueba original, modifica el código para que ahora utilice el modo de direccionamiento inmediato e indirecto (en programas separados) para que imprima el carácter '@'. Documenta tus resultados.

```
1 section .data
2     msg db 'Resultado: ', 0
3
4 section .bss
5     buffer resb 1 ; Reservar 1 byte para almacenar el carácter
6
7 section .text
8     global _start
9
10 _start:
11     ; Imprimir el mensaje "Resultado: "
12     mov eax, 4
13     mov ebx, 1
14     mov ecx, msg
15     mov edx, 11
16     int 0x80          ; Llamada al sistema para escribir en salida estándar
17
18     ; Asignar el valor ASCII de '@' directamente usando direccionamiento inmediato
19     mov byte [buffer], 64 ; Mover el valor 64 (ASCII '@') directamente a buffer
20
21     ; Imprimir el contenido de buffer (el carácter '@')
22     mov eax, 4
23     mov ebx, 1
24     mov ecx, buffer
25     mov edx, 1
26     int 0x80          ; Llamada al sistema para escribir en salida estándar
27
28     ; Salir del programa
29     mov eax, 1
30     xor ebx, ebx
31     int 0x80
```

2. Responde lo siguiente:

a. ¿Cómo afecta el modo de direccionamiento a la eficiencia de un programa en ensamblador? El modo de direccionamiento afecta la velocidad y el uso de memoria. El modo inmediato es rápido y eficiente en memoria, mientras que modos como el indirecto o directo pueden ser más lentos y consumir más recursos al acceder a direcciones de memoria externas.

b. ¿Qué papel juegan los modos de direccionamiento en la programación de sistemas y controladores? Los modos de direccionamiento permiten acceder a registros de hardware y controlar recursos de manera eficiente. Facilitan operaciones en tiempo real, optimizando la respuesta del sistema, especialmente en controladores y sistemas embebidos donde el acceso directo y rápido al hardware es esencial.