UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
Facultad de Ingeniería, Arquitectura y Diseño

Ingeniero en Software y Tecnologías Emergentes

Manual de Talleres de la materia Organización de Computadoras

AUTOR(ES)
Jonatan Crespo Ragland

Taller No. 9 Modos de direccionamiento

Objetivo: Desarrollar el uso de los modos de direccionamiento

Fundamentos teóricos del taller

- Lenguaje Ensamblador x86

Instrucciones para el desarrollo del taller

1. Desarrolla los siguientes puntos

Recursos

1. Apuntes de clase.
2. Fuentes bibliográficas.
3. Lápiz.
4. Equipo de cómputo.

Tiempo e instrucciones de entrega

Duración: 2 horas.

Desarrollar lo siguiente en su cuaderno o computadora: 1. De acuerdo al código de prueba https://onecompiler.com/assembly/443pg5uuh responde y desarrolla lo siguiente:

a. Agrega comentarios en el código explicando su funcionamiento.

a. ¿Para qué sirve la instrucción `movzx`?
`movzx` significa Move with Zero-Extend.

Sirve para:

- Copiar un valor pequeño (byte o word) a un registro más grande (word o dword),
- Rellenando los bits superiores con ceros.

En tu código:

```
movzx eax, byte [result]
```

Está copiando 1 byte (result) → eax (32 bits), y pone en ceros los 24 bits de arriba.
Así evitas que EAX tenga basura o números incorrectos.

c. ¿Usa algún modo de direccionamiento? ¿Cuál?

Sí, el programa usa modo de direccionamiento por memoria (directo).

Ejemplos:

```
mov al, [num1]
mov al, [num2]
movzx eax, byte [result]
```

Aquí se accede directamente a la dirección de memoria donde están las variables.

Eso es direccionamiento directo porque usa el nombre de la etiqueta.

d. Explica qué imprime el programa y por qué

El programa:

1. Toma `num1 = 5`
2. Toma `num2 = 11`
3. Suma → `5 + 11 = 16`
4. Convierte 16 a ASCII sumándole 48, lo cual es incorrecto porque solo funciona para 0–9.
5. Imprime:

```
Resultado: (carácter ASCII 16 + 48 = ASCII 64
→ '@')
```

Entonces imprime:

```
Resultado: @
```

Porque 16 + 48 = 64, y ASCII 64 es @.

e. Modifica el programa para imprimir: `A, \, $, & y 1`

✔ Versión modificada (simple y documentada)

Los caracteres deseados en ASCII son:

Carácter	ASCII
A	65
\	92
$	36
&	38
1	49

Entonces los metemos directamente al buffer.

```
section .data
 msg db 'Caracteres: ', 0

section .bss
 buffer resb 5

section .text
 global _start
_start:
```

```
 ; Guardar los caracteres deseados
mov byte [buffer], 'A'
 mov byte [buffer+1],'\' ; diagonal inversa
 mov byte [buffer+2],'$'
 mov byte [buffer+3],'&'
 mov byte [buffer+4],'1'

 ; Imprimir mensaje
 mov eax,4
 mov ebx,1
 mov ecx,msg
 mov edx,12
 int 0x80

 ; Imprimir caracteres
 mov eax,4
 mov ebx,1
 mov ecx,buffer
 mov edx,5
 int 0x80

 ; Salida
 mov eax,1
 xor ebx,ebx
 int 0x80
```

f. ¿Fue la única forma de modificar el código? ¿Qué otra línea pudo cambiarse?

No, no fue la única forma.

Otra forma habría sido:

- Cambiar el valor de `result`
- O evitar totalmente la conversión con `add eax, 48`
- O escribir directamente en memoria con `mov al, 'A'`, etc.
- O modificar `num1` y `num2` para que su suma dé el valor ASCII deseado.

Por ejemplo:
Si quieres imprimir `'A'` (ASCII 65), puedes poner:

```
num1 db 60
num2 db 5
```

Porque 60 + 5 = 65, y al sumarle 48 obtendrías el ASCII equivocado, así que podrías quitar la línea `add eax, 48`.

g. Imprimir el carácter '@' usando direccionamiento inmediato e indirecto

✔ Versión 1 — Direccionamiento inmediato

(Se carga el valor directamente en un registro)

```
section .text
global _start

_start:
 mov al, '@' ; inmediato
 mov [buffer], al

 mov eax,4
 mov ebx,1
 mov ecx,buffer
 mov edx,1
 int 0x80

 mov eax,1
 xor ebx,ebx
 int 0x80
```

✔ Versión 2 — Direccionamiento indirecto

(Se carga la dirección en un registro, luego se usa ese

registro)

```
section .data
 symbol db '@'

section .bss
 buffer resb 1

section .text
global _start

_start:
 mov esi, symbol ; ESI apunta a la variable
 mov al, [esi] ; indirecto
 mov [buffer], al

 mov eax,4
 mov ebx,1
 mov ecx,buffer
 mov edx,1
 int 0x80
 mov eax,1
 xor ebx,ebx
 int 0x80
```

a. ¿Cómo afecta el modo de direccionamiento a la eficiencia?

Los modos de direccionamiento determinan cuántos ciclos toma obtener un dato.

- Inmediato → más rápido (dato directo en la instrucción).
- Registro → también muy rápido.
- Directo / indirecto → más lento porque se accede a memoria.
- Indexado → más lento todavía si requiere cálculos de direcciones.

Mientras más accesos a memoria, menos eficiente es el programa.

b. ¿Qué papel juegan los modos de dirección en sistemas y controladores?

Son esenciales porque permiten:

- Acceso rápido a hardware y memoria.
- Manipular registros especiales del sistema.

- Controlar dispositivos (I/O, buffers, sensores).
- Optimizar rendimiento en rutinas críticas.
- Manejar tablas, vectores e interrupciones eficientemente.

Sin los modos adecuados, un sistema embebido sería mucho más lento.

Código prueba

```
section .data
 num1 db 5
 num2 db 11
 result db 0
 msg db 'Resultado: ', 0

section .bss
 buffer resb 4

section .text
 global _start
_start:
```

```
 mov al, [num1]
 add al, [num2]
 mov [result], al
; Convertir el resultado a ASCII
 movzx eax, byte [result]
 add eax, 48 ; Convertir el valor numérico en su
correspondiente ASCII ('0' = 48)
 mov [buffer], al ; Almacenar el carácter ASCII en el buffer

 mov eax, 4
 mov ebx, 1
 mov ecx, msg
 mov edx, 11
 int 0x80


 mov eax, 4
 mov ebx, 1
 mov ecx, buffer
 mov edx, 1
 int 0x80

 mov eax, 1
 xor ebx, ebx
 int 0x80
```

Referencias

1. Ornare quam viverra orci sagittis eu volutpat. Aenean et tortor at risus. Feugiat in

ante metus dictum at tempor commodo.

2. Senectus et netus et malesuada fames ac. Dictum sit amet justo donec enim diam vulputate ut pharetra. Tristique senectus et netus et malesuada fames

Anexos

Incluir un anexo con modelos de rúbricas, formatos de evaluación y otros recursos que faciliten la implementación de los talleres