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. 6 Generación de un programa ejecutable

Objetivo: Desarrollar los códigos de operación y proceso de ensamblaje

---

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. **Desarrolla el funcionamiento, usos o aplicaciones y un ejemplo de su uso los siguientes opcodes en ensamblador x86:**
    a. **MOV**

Funcion: Copia datos de un lugar a otro (registro ↔ registro, registro ↔ memoria, inmediato → registro/memoria).
Uso: Cargar valores iniciales, mover datos entre registros o memoria.
Ejemplo:

```
MOV AX, 5      ; AX = 5
MOV BX, AX     ; BX = 5 (copia de AX)
```

b. ADD

Funcion: Suma dos operandos (resultado queda en el destino).
Uso: Operaciones aritméticas, acumuladores.
Ejemplo:

```
MOV AX, 3
ADD AX, 2      ; AX = 3 + 2 = 5
```

c. SUB

Funcion: Resta el segundo operando al primero (resultado en destino).
Uso: Calcular diferencias, decrementos.
Ejemplo:

```
MOV AX, 10
SUB AX, 4      ; AX = 10 - 4 = 6
```

d. INC

Funcion: Incrementa en 1 el valor de un registro o memoria.
Uso: Contadores, bucles.
Ejemplo:

```
MOV CX, 7
INC CX        ; CX = 8
```

e. **DEC**

Funcion: Decrementa en 1 el valor de un registro o memoria.
Uso: Contadores regresivos, bucles.
Ejemplo:

```
MOV CX, 5
DEC CX        ; CX = 4
```

f. **CMP**

Funcion: Resta dos operandos sin guardar el resultado, solo afecta banderas (flags).
Uso: Comparaciones lógicas para saltos condicionales.
Ejemplo:

```
MOV AX, 5
CMP AX, 5     ; compara AX con 5 → ZF = 1 (son iguales)
```

g. **JMP**

Funcion: Salto incondicional a una dirección o etiqueta.
Uso: Alterar flujo, bucles infinitos, saltos de código.

Ejemplo:

```
JMP etiqueta
; ...
etiqueta:
MOV AX, 1
```

h. JE/JNE

Funcion:

JE (Jump if Equal): Salta si ZF = 1 (los valores comparados son iguales).

JNE (Jump if Not Equal): Salta si ZF = 0 (los valores son distintos).

Uso: Decisiones condicionales.

Ejemplo:

```
MOV AX, 4
CMP AX, 4
JE sonIguales    ; salta porque AX == 4
JNE noIguales
sonIguales:
MOV BX, 1
noIguales:
```

i. JZ/JNZ

Funcion:

JZ (Jump if Zero): Igual que JE, salta si ZF = 1.

JNZ (Jump if Not Zero): Igual que JNE, salta si ZF = 0.

Uso: Se emplea después de operaciones aritméticas.

Ejemplo:

```
SUB AX, AX      ; AX = 0 → ZF = 1
JZ ceroEncontrado
JNZ noCero
ceroEncontrado:
MOV BX, 100
```

j. **PUSH/POP**

Funcion:

PUSH: Guarda (apila) un valor en la pila.

POP: Recupera (desapila) el último valor guardado.

Uso: Guardar registros temporales, paso de parámetros, manejo de stack.

EJemplo:

```
MOV AX, 10
PUSH AX         ; guarda 10 en la pila
POP BX          ; recupera en BX → BX = 10
```

k. **CALL/RET**

Funcion:

CALL: Llama a un procedimiento (guarda la dirección de retorno en la pila).

RET: Regresa al punto donde se hizo CALL.

Uso: Subrutinas, organización modular.

Ejemplo:

```
CALL miFuncion
; ...
miFuncion:
MOV AX, 1
RET
```

I. **SHR/SHL**

Funcion:

SHR (Shift Right): Desplaza bits a la derecha (divide entre 2).

SHL (Shift Left): Desplaza bits a la izquierda (multiplica por 2).

Uso:Operaciones rápidas de multiplicación/división por potencias de 2, manipulación de bits.

Ejemplo:

```
MOV AL, 8       ; AL = 00001000b
SHR AL, 1       ; AL = 00000100b = 4
SHL AL, 2       ; AL = 00010000b = 16
```

2. **Investiga y desarrolla cada uno de los bits de 0 al 21 de EFLAGS O RFLAGS para un procesador con arquitectura de x86 o x64. Identificando el modelo del procesador, con una referencia. Ejemplo:**

Ejemplo de X Procesador(EFLAGS/RFLAGS)

Bit	Nombre	Significado breve
0	**CF** (Carry)	Acarreo/borrow en aritmética sin signo
1	—	Reservado (siempre 1 en algunos modos antiguos; ignóralo)
2	**PF** (Parity)	Paridad del byte menos significativo (1 = par)
3	—	Reservado
4	**AF** (Aux Carry)	Acarreo entre nibble bajo/alto (BCD)

Bit	Nombre	Significado breve
0	CF (Carry)	Acarreo/borrow en aritmética sin signo.
1	—	Reservado (en modos antiguos siempre leído como 1, ignóralo).
2	PF (Parity)	Paridad del byte menos significativo (1 = par).
3	—	Reservado.
4	AF (Aux Carry)	Acarreo entre nibble bajo/alto (BCD).
5	—	Reservado.
6	ZF (Zero)	Se activa si el resultado de una operación es 0.
7	SF (Sign)	Refleja el signo del resultado (bit más alto = 1 → negativo).
8	TF (Trap)	Activa modo de depuración paso a paso (genera excepción tras cada instrucción).
9	IF (Interrupt Enable)	Permite o bloquea interrupciones externas enmascarables (1 = habilitadas).
10	DF (Direction)	Dirección en operaciones de cadena (0 = adelante/incrementa, 1 = atrás/decrementa).
11	OF (Overflow)	Indica desbordamiento en operaciones con signo.
12–13	IOPL (I/O Privilege Level)	Nivel de privilegio requerido para instrucciones de E/S (0–3).

14	NT (Nested Task)	Marca tarea anidada (encadenamiento en cambio de tareas).
15	—	Reservado.
16	RF (Resume Flag)	Reanuda tras excepción de depuración (evita repetición del breakpoint).
17	VM (Virtual 8086 Mode)	Indica ejecución en modo virtual-8086 (compatibilidad con 8086 real).
18	AC (Alignment Check)	Genera excepción si una dirección de memoria no está alineada (en nivel de privilegio 3).
19	VIF (Virtual Interrupt Flag)	Estado virtual del IF usado en virtualización.
20	VIP (Virtual Interrupt Pending)	Señala la interrupción pendiente en modo virtual.
21	ID (ID Flag)	Permite o bloquea la instrucción CPUID (si = 1, la instrucción CPUID está disponible).

3. **Generar un fragmento de código en ensamblador x86 (No es necesario que compile correctamente), para los bloques section .data, section .text y _start: que impliquen el uso de un salto condicional en ensamblador. El entregable en esta práctica sería el fragmento de código, agregando comentarios de su funcionamiento. El entregable de este taller para su repositorio sería solamente este fragmento de código.**

```
; ==============================
; Ejemplo de ensamblador x86 con salto condicional
; Uso de NASM-style syntax
; ==============================

section .data
      mensaje_igual db "Los valores son iguales", 0xA ; Cadena a mostrar si AX == BX
    mensaje_noigual db "Los valores son distintos", 0xA ; Cadena a mostrar si AX != BX
   len_igual equ $ - mensaje_igual
   len_noigual equ $ - mensaje_noigual

section .text
   global _start

_start:
   ; Inicializamos registros con valores de prueba
   mov eax, 5        ; EAX = 5
   mov ebx, 5        ; EBX = 5

   ; Comparamos los valores
    cmp eax, ebx       ; Resta lógica (EAX - EBX), ajusta banderas
    je son_iguales     ; Salta a etiqueta si ZF=1 (EAX == EBX)
   jne no_iguales    ; Salta si ZF=0 (EAX != EBX)

son_iguales:
```

```
    ; Llamada al sistema write (imprime mensaje_igual)
    mov eax, 4        ; syscall: sys_write
    mov ebx, 1        ; file descriptor: stdout
    mov ecx, mensaje_igual
    mov edx, len_igual
    int 0x80          ; interrupción para syscall
    jmp salir         ; saltamos al final

no_iguales:
    ; Llamada al sistema write (imprime mensaje_noigual)
    mov eax, 4
    mov ebx, 1
    mov ecx, mensaje_noigual
    mov edx, len_noigual
    int 0x80

salir:
    ; Terminar el programa (exit)
    mov eax, 1        ; syscall: sys_exit
    xor ebx, ebx      ; código de salida = 0
    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