1. Funcionamiento, usos y ejemplo de cada opcode x86

**Función:** Copia datos de un origen a un destino.
**Usos:** Transferencia de valores entre registros, memoria y constantes.

```
mov eax, 5     ; EAX = 5

mov ebx, eax   ; EBX = EAX = 5
```

**Función:** Suma el operando fuente al destino.
**Usos:** Sumar enteros, punteros, índices.
**Ejemplo:**

```
add eax, 3     ; EAX = EAX + 3
```

**c. SUB**

**Función:** Resta el operando fuente al destino.
**Usos:** Restas, decrementos, cálculos aritméticos.
**Ejemplo:**

```
sub eax, 2     ; EAX = EAX – 2
```

**d. INC**

**Función:** Incrementa en 1 el operando.
**Uso:** Contadores, ciclos.
**Ejemplo:**

```
inc ecx        ; ECX = ECX + 1
```

**e. DEC**

**Función:** Decrementa en 1 el operando.
**Uso:** Ciclos hacia abajo, contadores inversos.
**Ejemplo:**

```
dec ecx        ; ECX = ECX – 1
```

**f. CMP**

**Función:** Compara dos valores realizando una resta interna (sin guardar el resultado).
**Uso:** Tomar decisiones con saltos condicionales.
**Ejemplo:**

```
cmp eax, ebx   ; Compara EAX con EBX
```

**g. JMP**

**Función:** Salto incondicional a otra etiqueta.
**Uso:** Ciclos, saltos de control, evitar bloques de código.
**Ejemplo:**

jmp inicio

**h. JE / JNE**

**JE (Jump if Equal):** salta si ZF = 1
**JNE (Jump if Not Equal):** salta si ZF = 0
**Uso:** Comparaciones de igualdad.
**Ejemplo:**

cmp eax, ebx

je iguales

jne diferentes

**i. JZ / JNZ**

**JZ:** Salta si el resultado previo fue cero (ZF = 1)
**JNZ:** Salta si NO fue cero
**Uso:** Validar resultados nulos.
**Ejemplo:**

cmp eax, 0

jz es_cero

**j. PUSH / POP**

**PUSH:** Inserta un valor en la pila.
**POP:** Extrae un valor de la pila.
**Uso:** Guardar valores temporales, parámetros de funciones.
**Ejemplo:**

push eax    ; Guarda EAX en la pila

pop ebx     ; Recupera en EBX

**k. CALL / RET**

**CALL:** Salta a una función guardando la dirección de retorno.
**RET:** Regresa al punto después del CALL.

**Uso:** Subrutinas.
**Ejemplo:**

```
call mi_funcion

...

mi_funcion:

    mov eax, 10

    ret
```

**I. SHR / SHL**

**SHR:** Desplaza bits a la derecha (divide por 2).
**SHL:** Desplaza bits a la izquierda (multiplica por 2).
**Uso:** Operaciones bit a bit, multiplicaciones rápidas, máscaras.
**Ejemplo:**

```
shl eax, 1   ; EAX = EAX * 2

shr ebx, 1   ; EBX = EBX / 2
```

## 2. Bits 0 a 21 del registro EFLAGS/RFLAGS

**Procesador referenciado:** Intel Core i7 (Arquitectura x86-64).
**Fuente:** Intel® 64 and IA-32 Architectures Developer's Manual, Vol. 1.

| Bit | Nombre | Descripción |
|---|---|---|
| **0** | **CF – Carry Flag** | Indica acarreo en operaciones aritméticas. |
| **1** | *Reservado* | No se usa. |
| **2** | **PF – Parity Flag** | 1 si el número de bits en 1 (LSB) es par. |
| **3** | *Reservado* | |
| **4** | **AF – Auxiliary Carry Flag** | Acarreo entre nibble 3 y 4 (BCD). |
| **5** | *Reservado* | |
| **6** | **ZF – Zero Flag** | 1 si el resultado fue 0. |
| **7** | **SF – Sign Flag** | Copia del bit más significativo. |
| **8** | **TF – Trap Flag** | Modo paso-a-paso (debug). |
| **9** | **IF – Interrupt Enable Flag** | 1: habilita interrupciones. |
| **10** | **DF – Direction Flag** | 0=incremento, 1=decremento en cadenas. |
| **11** | **OF – Overflow Flag** | Indica desbordamiento aritmético. |
| **12–13** | **IOPL – I/O Privilege Level** | Nivel de privilegio para I/O. |
| **14** | **NT – Nested Task** | Indica tarea anidada. |
| **15** | *Reservado* | |
| **16** | **RF – Resume Flag** | Evita reactivar breakpoints. |
| **17** | **VM – Virtual 8086 Mode** | Activa modo virtual 8086. |
| **18** | **AC – Alignment Check** | Verifica alineación de memoria. |
| **19** | **VIF – Virtual Interrupt Flag** | IF virtualizado. |

| | | |
|---|---|---|
| **20** | **VIP – Virtual Interrupt Pending** | Indica interrupción virtual pendiente. |
| **21** | **ID – Identification Flag** | Permite CPUID. |

3 Fragmento de código x86 con salto condicional

```
section .data
    mensaje_igual db "Los valores son iguales", 0Ah
    mensaje_difer db "Los valores son diferentes", 0Ah

section .text
global _start

_start:
    mov eax, 5          ; Primer valor
    mov ebx, 5          ; Segundo valor

    cmp eax, ebx        ; Compara EAX y EBX
    je son_iguales      ; Si son iguales, salta

    ; Si NO son iguales
    mov ecx, mensaje_difer
    mov edx, 25
    mov ebx, 1
    mov eax, 4
    int 80h
    jmp fin
```

```
son_iguales:
    mov ecx, mensaje_igual
    mov edx, 25
    mov ebx, 1
    mov eax, 4
    int 80h

fin:
    mov eax, 1
    mov ebx, 0
    int 80h             ; Salir del programa
```