UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

Facultad de Ingeniería, Arquitectura y Diseño.

Ingeniería en Software y Tecnologías Emergentes

Organización de Computadoras

Taller 6

Alejandro Palomares Ceseña

1. Desarrollo de Opcodes (Funcionamiento y Usos)

Aquí tienes la explicación de los mnemónicos solicitados en la segunda imagen:

a. MOV (Move)

- **Funcionamiento:** Copia datos de un origen a un destino. No borra el origen.
- **Uso:** Es la instrucción más común; sirve para cargar valores en registros, mover datos entre registros o guardar en memoria.
- **Ejemplo:** MOV EAX, 5 (Mueve el valor 5 al registro EAX).

b. ADD (Add)

- **Funcionamiento:** Suma el operando de origen al de destino y guarda el resultado en el destino.
- **Uso:** Cálculos aritméticos básicos, incrementar contadores o punteros.
- **Ejemplo:** ADD EAX, EBX (Suma el valor de EBX a EAX; resultado queda en EAX).

c. SUB (Subtract)

- **Funcionamiento:** Resta el operando de origen del destino.
- **Uso:** Cálculos aritméticos, comparaciones manuales (verificando si el resultado es cero).
- **Ejemplo:** SUB ECX, 1 (Resta 1 al valor de ECX).

d. INC (Increment)

- **Funcionamiento:** Suma 1 al operando. Es más rápido/corto que ADD destino, 1.
- **Uso:** Bucles (loops) para contar iteraciones, avanzar en arrays.
- **Ejemplo:** INC EDX (Incrementa el valor de EDX en 1).

e. DEC (Decrement)

- **Funcionamiento:** Resta 1 al operando.
- **Uso:** Contadores regresivos en bucles (ej. for i=10 to 0).
- **Ejemplo:** DEC EAX (Reduce el valor de EAX en 1).

f. CMP (Compare)

- **Funcionamiento:** Realiza una resta (Destino - Origen) **pero descarta el resultado**. Solo modifica las banderas (EFLAGS).
- **Uso:** Esencial antes de cualquier salto condicional (IF/ELSE).
- **Ejemplo:** CMP EAX, 10 (Compara si EAX es 10; activa ZF si es cierto).

g. JMP (Jump)

- **Funcionamiento:** Salto incondicional. Cambia el flujo del programa a una dirección de memoria específica sin preguntar nada.
- **Uso:** Bucles infinitos, saltar partes de código (como un GOTO).
- **Ejemplo:** JMP etiqueta_inicio.

h. JE / JNE (Jump if Equal / Jump if Not Equal)

- **Funcionamiento:**
    - **JE:** Salta si la bandera ZF = 1 (el resultado de la comparación anterior fue igualdad).
    - **JNE:** Salta si la bandera ZF = 0 (eran diferentes).
- **Uso:** Estructuras de control (if (a == b) o if (a != b)).
- **Ejemplo:** JE fin_programa (Si son iguales, ve al final).

i. JZ / JNZ (Jump if Zero / Jump if Not Zero)

- **Funcionamiento:** Son sinónimos técnicos de JE/JNE.
    - **JZ:** Salta si el resultado de la última operación fue Cero.
    - **JNZ:** Salta si no fue Cero.
- **Uso:** Verificar si un contador llegó a cero o si una operación aritmética se anuló.
- **Ejemplo:** DEC ECX seguido de JNZ bucle (Disminuye ECX y si no es cero, repite el bucle).

j. PUSH / POP

- **Funcionamiento:**
    - **PUSH:** Mete un valor en el tope de la Pila (Stack) y decrementa el puntero de pila (ESP).
    - **POP:** Saca el valor del tope de la Pila hacia un destino e incrementa el ESP.
- **Uso:** Guardar registros temporalmente, pasar argumentos a funciones, guardar direcciones de retorno.
- **Ejemplo:**
    - PUSH EAX (Guarda EAX en la pila).
    - POP EBX (Recupera el valor de la pila y lo guarda en EBX).

k. CALL / RET

- **Funcionamiento:**
    - **CALL:** Salta a una subrutina y **guarda la dirección de la siguiente instrucción** en la pila (para saber dónde volver).

    - **RET:** Saca esa dirección de la pila y salta de vuelta a ella.
- **Uso:** Llamada a funciones y retorno de las mismas.
- **Ejemplo:** CALL mi_funcion ... (dentro de la función) ... RET.

l. SHR / SHL (Shift Right / Shift Left)

- **Funcionamiento:** Desplaza los bits de un registro a la derecha o izquierda. Los bits que "salen" se pierden (o van al Carry Flag), y entran ceros por el otro lado.
- **Uso:** Operaciones a nivel de bit. Matemáticamente:
    - **SHL:** Multiplica por 2.
    - **SHR:** Divide entre 2 (división entera).
- **Ejemplo:** SHL EAX, 1 (Multiplica el valor de EAX por 2).

2. Investigación de los bits del registro EFLAGS / RFLAGS

Modelo de Procesador de Referencia: Intel® 64 and IA-32 Architectures (Compatible con procesadores modernos como Intel Core i9, AMD Ryzen, etc.).

Fuente: Intel® 64 and IA-32 Architectures Software Developer's Manual.

El registro **EFLAGS** (32 bits) o **RFLAGS** (64 bits) contiene indicadores de estado, de control y del sistema. Aquí tienes el desglose de los bits del 0 al 21:

Bit	Abreviatura	Nombre (Español / Inglés)	Significado breve
0	**CF**	Acarreo / Carry Flag	Se activa si una operación aritmética genera un acarreo (carry) o préstamo (borrow) fuera del bit más significativo.
1	**—**	Reservado	Siempre es 1. No tiene uso específico en programación.
2	**PF**	Paridad / Parity Flag	Se activa si el byte menos significativo del resultado contiene un número par de 1s.
3	**—**	Reservado	Siempre es 0.

Bit	Abreviatura	Nombre (Español / Inglés)	Significado breve
4	**AF**	Acarreo Auxiliar / Auxiliary Carry	Usado para aritmética BCD. Indica acarreo del bit 3 al 4.
5	**—**	Reservado	Siempre es 0.
6	**ZF**	Cero / Zero Flag	**Muy importante.** Se activa (1) si el resultado de una operación es cero.
7	**SF**	Signo / Sign Flag	Toma el valor del bit más significativo del resultado (indica si es negativo en complemento a dos).
8	**TF**	Trampa / Trap Flag	Si está activo, permite la ejecución paso a paso (debugging).
9	**IF**	Interrupción / Interrupt Enable	Controla si el procesador responde a interrupciones enmascarables (1 = responde).
10	**DF**	Dirección / Direction Flag	Controla la dirección de las instrucciones de cadenas (string). 0 = incrementa, 1 = decrementa.
11	**OF**	Desbordamiento / Overflow Flag	Se activa si el resultado es demasiado grande o pequeño para caber en el destino (enteros con signo).
12-13	**IOPL**	Nivel de Privilegio E/S	Indica el nivel de privilegio necesario para ejecutar instrucciones de entrada/salida.
14	**NT**	Tarea Anidada / Nested Task	Controla el encadenamiento de tareas interrumpidas y llamadas.

Bit	Abreviatura	Nombre (Español / Inglés)	Significado breve
15	**—**	Reservado	Siempre es 0.
16	**RF**	Reanudar / Resume Flag	Controla la respuesta del procesador a excepciones de depuración.
17	**VM**	Virtual-8086 Mode	Habilita el modo virtual 8086.
18	**AC**	Verificación de Alineación	Habilita la comprobación de alineación de memoria si el bit AM en CR0 está activo.
19	**VIF**	Virtual Interrupt Flag	Imagen virtual de la bandera IF.
20	**VIP**	Virtual Interrupt Pending	Indica que una interrupción está pendiente.
21	**ID**	Identification Flag	Si un programa puede cambiar este bit, el procesador soporta la instrucción CPUID.

3. Fragmento de código en Ensamblador (Salto Condicional)

Aquí tienes el código solicitado con las secciones .data, .text y _start. El ejemplo compara dos números y salta si son iguales.

```
section .data
   ; Definimos dos variables inicializadas con valores
   num1 db 10        ; Definimos byte con valor 10
   num2 db 10        ; Definimos byte con valor 10
   msg_igual db "Son iguales", 0xA ; Mensaje por si son iguales

section .text
   global _start      ; Punto de entrada para el linker

_start:
   ; --- Inicio de la lógica ---
   mov al, [num1]     ; Mueve el valor de num1 al registro AL
   mov bl, [num2]     ; Mueve el valor de num2 al registro BL

   cmp al, bl         ; CMP (Comparar): Resta internamente AL - BL
                      ; Esto actualiza las banderas (EFLAGS) pero no guarda el resultado.
                      ; Si son iguales, la Zero Flag (ZF) se pone en 1.

   je son_iguales     ; JE (Jump if Equal): Salta a la etiqueta 'son_iguales'
                      ; solo si la bandera ZF está encendida (es decir, num1 == num2).

   ; --- Si NO son iguales, el programa continúa aquí ---
   ; (Aquí iría código para el caso "no son iguales")
   jmp salir          ; Salto incondicional al final para evitar ejecutar el bloque de abajo
```

```
son_iguales:
    ; --- Bloque que se ejecuta si el salto fue exitoso ---
    ; Aquí podríamos imprimir un mensaje, sumar algo, etc.
    ; Por simplicidad, solo marcamos este punto.

salir:
    ; --- Finalización del programa (syscall exit) ---
    mov eax, 1          ; Número de syscall para sys_exit
    xor ebx, ebx        ; Código de retorno 0
    int 0x80            ; Interrupción para llamar al kernel
```