Instrucciones 8086

Estas son las instrucciones más utilizadas del 8086 en clase y sus equivalencias en C, si las tiene:

MOV: copia el 2do operando al 1er operando.
Ejemplo: **MOV AX, 53h** → guarda el valor 0053 hexadecimal en el registro AX.
Equivalencia en C → *variable1 = variable2*

INC: incrementa en 1 el valor indicado.
Ejemplo:

```
MOV AL, 53h
INC AL → Ahora AL valdrá 54h
```

Equivalencia en C → *variable++*

DEC: decrementa en 1 el valor indicado.
Ejemplo:

```
MOV AL, 53h
DEC AL → Ahora AL valdrá 52h
```

Equivalencia en C → *variable--*

ADD: suma un valor a otro que ya habíamos asignado.
Ejemplo:

```
MOV AL, 53h → guarda el valor 53 en hexadecimal en el registro AL.
ADD AL, 3 → ahora, AL tendrá un valor de 56
```

Equivalencia en C → *variable1 += variable2*

AND: Es un operador lógico. Desglosa ambos valores, idependientemente de la base que utilizemos, en bits y luego compara: Si y sólo si ambos valores son 1, entonces el resultado es 1. De lo contrario será 0.
Ejemplo:

```
MOV AX, 0AB53h    1010 1011 0101 0011
AND AX, 0AB04h    1010 1011 0000 0100
                   A    B    0    0   → AX tendrá un valor de AB00
```

Equivalencia en C → *es el operador & binario*

XOR: Otro operador lógico. Resulta 1 si y sólo si ambos valores son diferentes. Es útil para dejar en 0 un registro utilizando como 2do valor el mismo número que el valor almacenado.
Ejemplo:

```
MOV AX, 0AB53h    1010 1011 0101 0011
XOR AX, 0AB04h    1010 1011 0000 0100
                   0    0    5    7   → AX tendrá un valor de 0057
```

Equivalencia en C → es el operador ^

DAA: Se usa para aritmética BCD. Cuando se realiza una operación BCD, automáticamente suma 6 si es necesario. Cambia el flag carry y el AF. Sólo sirve para el registro AL, ya que en el procesador 8080 sólo se podía hacer la suma en el registro A.

CALL: Llama a un procedimiento.
Procedimiento: *Es un fragmento de código que puede ejecutarse cuando se lo llama. Es el quivalente a la función en C. (También el profe llama función al procedimiento. Otras palabras equivalentes pueden ser "subrutina" y "método").*

Ejemplo:

```
miproc:
    MOV AX, 1234h
    RET
```

Equivalencia en C → *void mifuncion(){*
variable = 0x1234;
return
}

Otra posibilidad → int mifuncion(){
return 0x1234;
}

Ejemplo: **CALL miproc**
Equivalencia en C → mifuncion()

LEA: Computa una dirección y en vez de cargar lo que está en la memoria, guarda la dirección en el registro.
Equivalencia en C → *es como el & que se encuentra al lado de la variable en scanf*

JMP: goto, se encuentra en c al final del bucle while implícitamente.
Equivalencia en C → *Break o continue pueden ser JMP.*

JNE, JNC, JNZ: Son jumps con condiciones.
Ejemplo:

```
   CMP AX, CX
   JNZ coso
   MOV AX, 53
   RET
coso:
```

Equivalencia en C → if (a == b) *return 53;*

Y otro ejemplo:

```
   CMP AX, CX
   JL coso          ; o jb si a y b son unsigned
   ADD AX, 53
   RET
coso:
   MOV AX, 13
   RET
```

Equivalencia en C → *if (a < b) {*
return a + 3;
} else {
return 13;
}

CMP: Compara sin guardar el valor en ningún lado. Se sigue mediante las flags. (Ver el ejemplo en la instrucción de arriba)

INT: Interrupciones.
Por software: Funciona como "call". Salvaguarda el contenido para volver y salta al n° de interrupción correspondiente.
Normalmente saltan a la BIOS o al DOS.
Ejemplo: **INT 20h** ; Termina tu programa en DOS.

PUSH: Ingresa datos a la pila del programa. Los ahorra para más tarde, permitiendo utilizar un registro libremente sabiendo que se puede restaurar después.

POP: Restaura un registro de la pila.

Para más instrucciones, se puede ir a la sección “help” del emu8086.