

MÓDULO Nº15

PROGRAMACIÓN DEL 8085

UNIDAD: MICROPROCESADORES Y MICROCONTROLADORES

TEMAS:

- Diagramas de Flujo.
- Lenguaje Ensamblador.
- Set de Instrucciones.
- Ejemplos de Programación.

OBJETIVOS:

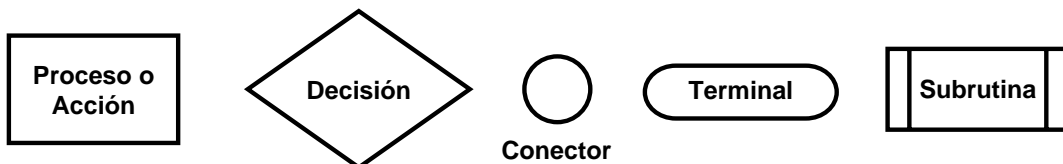
- Entender los diferentes elementos del lenguaje ensamblador.
- Explicar por grupos la función de las diferentes instrucciones del 8085.
- Analizar y desarrollar programas básicos para el 8085.

DESARROLLO DE TEMAS

1. Diagramas de Flujo:

Antes de profundizar en la programación del 8085, es necesario, realizar un breve repaso de los diagramas de flujo, los cuales facilitarán en gran manera el entendimiento y desarrollo de aplicaciones basadas en este procesador.

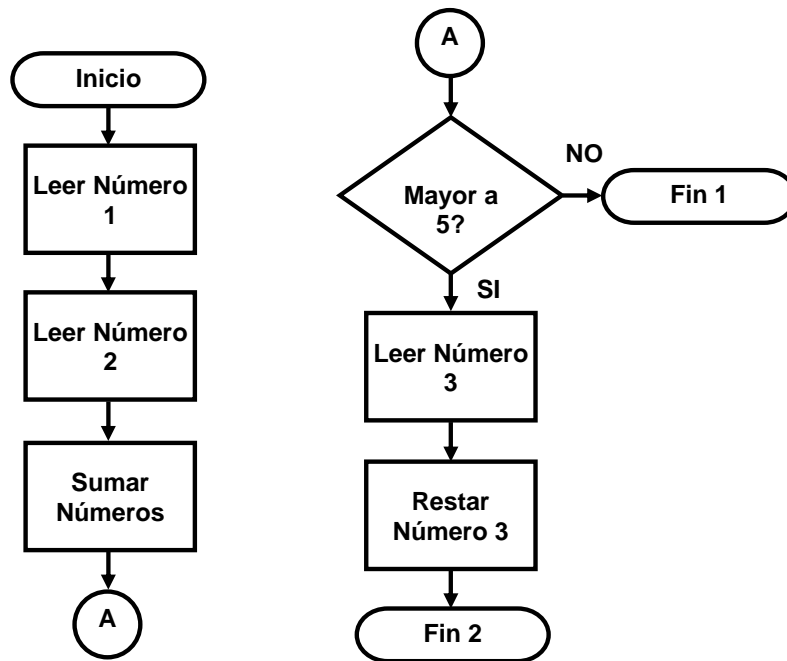
Un diagrama de flujo es una representación gráfica que muestra los pasos y/o procesos para solucionar un problema o realizar una tarea. Los símbolos más comúnmente usados en diagrama de flujo se presentan a continuación:



Donde:

- **Proceso o Acción:** Representa una operación a realizarse para la resolución del problema.
- **Decisión:** Simboliza una toma de decisión y permita la rotura de la secuencia normal de programa si se cumple cierta condición.
- **Terminal:** Generalmente se utiliza al principio y al final del diagrama para definir el nombre que llevará el programa.
- **Subrutina:** En general, representa un conjunto de procesos, que se encuentran definidos por otro diagrama de flujo, su finalidad es la reducir la complejidad de un programa al dividirlo por bloques.
- **Conector:** Se utiliza para enlazar diagramas, su utilización es más estética que funcional.

En la siguiente página se puede apreciar la utilización de estos símbolos, para construir un diagrama que representa los pasos para realizar la suma de dos números y a dicho resultado restarle un tercer número si el resultado es mayor a 5:



Si bien se trata de un problema sencillo, permite realizar varios análisis. Por ejemplo ¿Dónde se guardó los números almacenados?, ¿Dónde fue almacenado el resultado de la suma así como de la resta?, ¿Cómo realizó la comparación del resultado con el número 5?, etc. Las respuestas dependen de muchos factores, tales como: El lenguaje de programación usado, el procesador usado, los periféricos requeridos, etc., y más importante aún, de lo que el programador desee dar a conocer o requiera del diagrama para entender o desarrollar el programa.

De lo anterior, un diagrama de flujo puede ser tan simple o preciso como lo requiera el programador.

2. Lenguaje Ensamblador:

Si bien los diagramas de flujo son útiles en definir los pasos para resolver un problema, no son más que letras sin sentido, si se utiliza el punto de vista del procesador.

Para el procesador, los datos almacenados en la memoria de instrucciones representan el programa o pasos a realizarse, sin embargo dicho programa resulta muy difícil de entender desde el punto de vista del programador. Un ejemplo de programa se muestra a continuación:

Direcciones (En hexadecimal)	Instrucciones, datos (En binario)
0 0 0 0 H	1 0 1 1 1 1 0 1
0 0 0 1 H	0 0 0 0 0 0 0 1
0 0 0 2 H	1 1 1 0 0 1 1 0
0 0 0 3 H	0 0 0 0 1 1 1 1
0 0 0 4 H	0 0 1 1 0 0 1 0
0 0 0 5 H	0 1 0 0 0 0 0 0
0 0 0 6 H	0 0 1 0 0 0 0 0
0 0 0 7 H	0 1 1 1 0 1 1 0

A esta forma de representar un programa se le denomina **Lenguaje de Máquina**, ya que es el único lenguaje que entiende el procesador.

Una ligera variación de este lenguaje, recibe el nombre de **Código Objeto**. En este caso se reemplaza los códigos binarios por su equivalente en hexadecimal, con lo que son más fáciles de manejar para el programador, tal como se muestra en seguida:

Direcciones (En hexadecimal)	Contenido Instrucción, dato (Hex)
0 0 0 0	BD
0 0 0 1	01
0 0 0 2	E6
0 0 0 3	0F
0 0 0 4	32
0 0 0 5	40
0 0 0 6	20
0 0 0 7	76

La dificultad para recordar cientos de códigos y la función que desempeñan, llevo a los programadores a desarrollar un lenguaje que permitiera el uso de **Nemónicos**. Un nemónico es un conjunto de hasta 4 letras, por lo general, son las iniciales o abreviatura de una palabra en inglés que describe la función a realizarse.

Por ejemplo, IN es el nemónico utilizado para referirse a una operación de ingreso de datos, MOV es el nemónico que representa el movimiento de datos, ADD es el nemónico que representa una operación de suma, etc.

Un programa escrito utilizando nemónicos recibe el nombre de Lenguaje Ensamblador, y es considerado como un lenguaje básico de programación. Seguidamente se muestra un ejemplo de dicho programa:

Código objeto
(Lenguaje de máquina)

Dirección	Instrucciones, datos
Instrucciones no ejecutables	
0010H	3E
0011H	08
0012H	06
0013H	07
0014H	80
0015H	32
0016H	20
0017H	00
0018H	76

Código fuente
(Lenguaje ensamblador)

Etiqueta	OP-CODE (Nemónico)	Operando	Comentarios
	ORG	0010H	; Empezar en 0010H
SUM 1:	EQU	08H	; Hacer SUM 1 = 08H
SUM 2:	EQU	07H	; Hacer SUM 2 = 07H
RESUL:	EQU	0020H	; Hacer RESUL = 0020H
	MVI	A, SUM1	; Cargar A con primer dato
	MVI	B, SUM2	; Cargar B con segundo dato
	ADD	B	; Sumar A con B
	STA	RESUL	; Llevar resultado de la suma a la posición RESUL convenida
	HLT		; Parar

↑
Directivas de ensamblador
↓

Como se puede observar, al comparar las dos representaciones de un mismo programa, se comprueba que el uso de nemónicos hace más legible al programa.

Un detalle especial que también puede ser observado, es que el lenguaje ensamblador no solo reemplaza códigos hexadecimales por nemónicos, sino que adiciona elementos extras con el fin de facilitar aún más la escritura de programas.

En general un programa en ensamblador consta de los siguientes elementos:

- **Etiquetas:** Son palabras o abreviaturas en español o inglés seguidas por dos puntos, que se utilizan para representar direcciones, variables y/o constantes utilizadas por el programa. Es importante recalcar que las etiquetas no deben tener el mismo nombre que los nemónicos a fin de evitar confusiones.
- **Nemónicos:** Representan instrucciones del procesador (ej.: MOV, ROT, SUB, etc.) o directivas utilizadas por el ensamblador para definir direcciones, variables y/o constantes utilizadas por el programa (ej.: ORG, EQU, END, etc.)
- **Operadores:** Son parámetros requeridos por la instrucción y/o directiva, y pueden ser registros del procesador, posiciones de memoria, valores numéricos, etiquetas, etc. (ej.: CALL SALIR, MOV A, B, EQU 08H, etc.).
- **Comentarios:** En este campo se escriben notas especiales antepuestas de un punto y coma, que ayudan al programador o al analizador, a entender la función de una instrucción o conjunto de instrucciones específicas. No son obligatorias.

3. Set de Instrucciones:

A todo el conjunto de comandos reconocibles por un procesador se le conoce como Set de Instrucciones del procesador. Dicho conjunto es único para cada procesador o familia de procesadores y depende exclusivamente del fabricante del procesador.

En la siguiente página se muestra el conjunto de instrucciones del 8085:

Para facilitar el estudio de las diferentes instrucciones del 8085, es conveniente agruparlas según la función que realizan, de esta manera se tienen los siguientes grupos de instrucciones:

- **Instrucciones para transferencia de datos:**

Básicamente se pueden tener los siguientes tipos de instrucciones:

- **MOV r1, r2:** Mueve el contenido del registro **r2** a **r1** (ej.: MOV A, B, MOV L, C).
- **MOV r, M; MOV M, r:** Mueve el contenido de un registro a la posición de memoria M definida por la pareja de registros HL, y viceversa (ej.: MOV C, M, MOV M, A).
- **MVI r, byte:** Carga un registro con un valor numérico de un byte de longitud (ej.: MVI D, 03H).
- **LXI rp, dbyte:** Carga una pareja de registros con un valor de 2 bytes de longitud (ej.: LXI H, 0010H, LXI B, 25AFH).
- **LDA addr:** Carga al acumulador con el contenido de la dirección indicada (ej.: LDA FF00H).
- **STA addr:** Guarda el contenido del acumulador en la dirección indicada (ej.: STA FF00H).

INSTRUCCIONES DEL MICROPROCESADOR 8085

GRUPO DE INSTRUCCIONES PARA LA TRANSFERENCIA DE DATOS

MOVIMIENTO	MOVIMIENTO (CONT)	MOVIMIENTO INMEDIATO
MOV	A,A 7F	EA 5F
	A,B 78	E,B 58
	A,C 79	E,C 59
	A,D 7A	E,D 5A
	A,E 7B	E,E 5B
	A,H 7C	E,H 5C
	A,L 7D	E,L 5D
A,M 7E	E,M 5E	
MOV	B,A 47	H,A 67
	B,B 40	H,B 60
	B,C 41	H,C 61
	B,D 42	H,D 62
	B,E 43	H,E 63
	B,H 44	H,H 64
	B,L 45	H,L 65
B,M 46	H,M 66	
MOV	C,A 4F	L,A 6F
	C,B 48	L,B 68
	C,C 49	L,C 69
	C,D 4A	L,D 6A
	C,E 4B	L,E 6B
	C,H 4C	L,H 6C
	C,L 4D	L,L 6D
C,M 4E	L,M 6E	
MOV	D,A 57	M,A 77
	D,B 50	M,B 70
	D,C 51	M,C 71
	D,D 52	M,D 72
	D,E 53	M,E 73
	D,H 54	M,H 74
	D,L 55	M,L 75
D,M 56		
	XCHG EB	

GRUPO DE INSTRUCCIONES ARITMETICAS Y LOGICAS

SUMA *	INCREMENTO **	LOGICAS *
A 87	A 3C	A A7
B 80	B 04	B A0
C 81	C 0C	C A1
D 82	D 14	D A2
E 83	E 1C	E A3
H 84	H 24	H A4
L 85	L 2C	L A5
M 86	M 34	M A6
ADC	A 8F	B 03
	B 88	D 13
	C 89	H 23
	D 8A	SP 33
	E 8B	
	H 8C	
	L 8D	
M 8E		
SUB	A 97	B 0B
	B 90	D 1B
	C 91	H 2B
	D 92	SP 3B
	E 93	
	H 94	
	L 95	
M 96		
SBB	A 9F	
	B 98	
	C 99	
	D 9A	
	E 9B	
	H 9C	
	L 9D	
M 9E		
DAD	B 09	RAL 17
	D 19	RAR 1F
	H 29	
	SP 39	

GRUPO DE INSTRUCCIONES DE CONTROL DE BIFURCACION

INSTRUCCIONES DE CONTROL Y ENTRADA/SALIDA

SALTO	LLAMADA	SOBRE LA PILA
JMP adr C3	CALL adr CD	PUSH B C5
JNZ adr C2	CNZ adr C4	D D5
JZ adr CA	CZ adr CC	H E5
JNC adr D2	CNC adr D4	PSW F5
JC adr E2	CC adr DC	
JPO adr E2	CPO adr E4	POP B C1
JPE adr EA	CPE adr EC	D D1
JP adr F2	CPF adr F4	H E1
JM adr FA	CPM adr F4	PSW * F1
PCHL E9	CM adr FC	
		XTHL E3
		SPHL F9
RET C9	RST 0 C7	ENTRADA/SALIDA
RNZ CO	1 CF	OUT byte D3
RZ C8	2 D7	IN byte DB
RNC DO	3 DF	
RC R8	4 E7	CONTROL
RPO EO	5 EF	DI F3
RPE EB	6 F7	EI FB
RP FO	7 FF	NOP 00
RM F8		HLT 76
		NUEVAS INSTRUCCIONES DEL 8085
		RIM 20
		SIM 30

NOTAS:

- byte:** CONSTANTE ó EXPRESION LOGICA ó ARITMETICA DE UN DATO DE 8 BITS. (ES EL SEGUNDO BYTE DE UNA INSTRUCCION QUE CONSTE DE 2 BYTES).
- db1e:** CONSTANTE ó EXPRESION LOGICA ó ARITMETICA DE UN DATO DE 16 BITS. FORMAN EL 2° Y 3° BYTE DE UNA INSTRUCCION DE 3 BYTES.
- adr:** DIRECCION DE 16 BITS. FORMAN EL 2° Y 3° BYTE DE UNA INSTRUCCION DE 3 BYTES.
- ***: TODOS LOS SEÑALIZADORES QUEDAN AFECTADOS.
- **:** TODOS LOS SEÑALIZADORES QUEDAN AFECTADOS, MENOS EL DE ACARREO. LAS INSTRUCCIONES INX Y DCX NO AFECTAN A LOS SEÑALIZADORES.
- +**: SOLO QUEDA AFECTADO EL ACARREO. TODOS LOS NEMONICOS TIENEN COPYRIGHT DE INTEL CORPORATION, 1976.

- **Instrucciones lógicas y aritméticas:**

La mayoría de instrucciones de este tipo afectan las banderas del registro de estado (Z, S, AC, P, CY). El resultado de toda operación que incluya al acumulador, es almacenado en el mismo acumulador. Básicamente se pueden tener los siguientes tipos de instrucciones:

- **ADD r:** Suma el contenido del registro especificado con el contenido del acumulador (ej.: ADD C, ADD L).
- **ADD M:** Suma el contenido almacenado en la dirección de memoria M especificada por los registros HL, con el contenido del acumulador (ej.: ADD M).
- **ADI byte:** Suma al acumulador el valor numérico especificado (ej.: ADI 05H).
- **ADC r; ADC M; ACI byte:** Realizan lo mismo que ADD r, ADD M y ADI byte, excepto que si se produce un acarreo, este también es sumado.
- **SUB r; SUB M; SUI byte; SBB r; SBB M; SBI byte:** Restan un valor del acumulador (ej.: SUB B, SUB M, SUI 05H).
- **INR r:** Incrementa en una unidad al registros especificado (ej.: INR B).
- **INR M:** Incrementa en una unidad al contenido de la dirección de memoria M (ej.: INR M).
- **DCR r; DCR M:** Realizan las operaciones inversas a INR r y INR M

- **ANA r:** Realiza la operación lógica AND entre el acumulador y el registro r (ej.: ANA C, ANA L).
- **ANA M:** Realiza la operación lógica AND entre el acumulador y el contenido de la dirección de memoria M (ej.: ANA M).
- **ANI byte:** Realiza la operación lógica AND entre el acumulador y el valor numérico especificado (ej.: ANI 05H).
- **ORA r; ORA M; ORI byte:** Realizan la operación lógica OR.
- **XRA r; XRA M; XRI byte:** Realizan la operación lógica XOR.
- **CMP r; CMP M; CPI byte:** Realizan una comparación con el acumulador. Si son iguales, la bandera Z se pone en 1 y si el acumulador es el menor, CY se pone en 1.
- **RLC:** Rota el acumulador una posición a la izquierda (en el sentido del bit 0 al bit 7)
- **RRC:** Rota el acumulador una posición a la derecha (en el sentido del bit 7 al bit 0)
- **CMA:** Complementa el valor del acumulador, es decir, realiza una operación lógica NOT con cada bit del acumulador.

- **Instrucciones de control de bifurcación o cambio de secuencia del programa:**

Estas instrucciones alteran el contenido del contador de Programa y pueden o no depender de una condición específica. Básicamente se pueden tener los siguientes tipos de instrucciones:

- **JMP addr:** Carga el contador de programa con la dirección especificada, es decir, "salta" a la dirección especificada para continuar la ejecución del programa desde esa dirección (ej.: JMP 0010H, JMP SIGUIENTE).
- **JNZ addr:** Salta si la bandera Z es cero.
- **JZ addr:** Salta si la bandera Z es uno.
- **JNC addr:** Salta si la bandera CY es cero.
- **JC addr:** Salta si la bandera CY es uno.
- **JP addr:** Salta si la bandera S es cero.
- **JM addr:** Salta si la bandera S es uno.
- **CALL addr:** Opera de la misma manera que JMP, sin embargo en este caso guarda la dirección actual más uno, en la pila. Debido a esto su uso es exclusivo para llamadas subrutinas (ej.: CALL 1010H, CALL LUCES)

- **CNZ addr; CZ addr; CNC addr; CC addr; CP addr; CM addr:** Operan como CALL pero basadas en una condición específica.
- **RET:** Trabaja en conjunto con CALL, su función es recuperar la última dirección guardada en la pila, es decir, restablece el contador de programa.
- **RNZ; RZ; RNC; RC; RP; RM:** Operan como RET pero basadas en una condición específica.

- **Instrucciones de control, entrada / salida y de manejo de pila:**

Básicamente se pueden tener los siguientes tipos de instrucciones:

- **PUSH rp:** Mueve a la pila el par de registros especificado, se utiliza para respaldar los registros en el caso de que vayan a ser utilizados por una subrutina (ej.: PUSH B, PUSH H).
- **PUSH PSW:** A diferencia de PUSH rp, en este caso respalda al acumulador junto al registro de estado.
- **POP rp; POP PSW:** Realizan la función contraria a PUSH, es decir, recuperan los valores almacenados en la pila.

- **IN port:** Carga en el acumulador el dato proveniente de un periférico específico cuya dirección es PORT (ej.: IN 00H, IN PORTA).
- **OUT port:** Envía el dato almacenado en el acumulador a un periférico específico cuya dirección es PORT (ej.: OUT 01H, IN PORTB).

- **HLT:** Detiene al contador de programa, por lo que el procesador se dice esta “congelado”, dicho estado se mantiene hasta que se le de un reset al procesador.
- **NOP:** No realiza ninguna operación, por lo que se dice que es un “código muerto”. Se utiliza para perder tiempo de ejecución del procesador, de esta manera se retarda el funcionamiento del procesador sin alterar ningún registro. Es común encontrarlo en demoras de tiempo.

4. Ejemplos de Programación:

Para finalizar, se presentan algunos programas básicos que demuestran la aplicación de varias de las instrucciones antes analizadas así como también detalles de la programación en lenguaje ensamblador.

- **Ejemplo 1:**

```

;El siguiente programa realiza la siguiente operación: SUM = NUM1 + NUM2
;si el resultado es igual a un número LIM,
;realiza la siguiente operación RES = SUM - NUM3.

;*****
;*****

;Define constantes utilizadas por el programa

NUM1      .EQU    05H          ;Número 1
NUM2      .EQU    02H          ;Número 2
NUM3      .EQU    03H          ;Número 3
LIM       .EQU    07H          ;Límite de Comparación

;*****
;*****

;El siguiente código representa al programa principal

        .ORG    0000H          ;Inicia en la posición de memoria 0000H
    
```

```

MVI    A, NUM1      ;Carga número 1 en el Registro A
MVI    B, NUM2      ;Carga número 2 en el Registro B
MVI    C, NUM3      ;Carga número 3 en el Registro c
MVI    D, LIM       ;Carga límite en el Registro D

ADD    B             ;Realiza NUM1 + NUM2
CMP    D             ;Compara el resultado de la suma
JZ     RESTAR       ;Si es igual al límite entonces salta a RESTAR
HLT    ;Caso contrario se detiene

RESTAR:
SUB    C             ;Realiza SUM - NUM3
HLT    ;Detiene al procesador

;*****
;*****

.END

```

• **Ejemplo 2:**

```

;El siguiente programa llama a dos rutinas de tiempo de un lazo y de doble lazo,
;mientras rota el dato en el registro A.
;El tiempo que demore la rutina depende del cristal utilizado.
;Para un cristal de 2MHz, la primera rutina se demora 1 mseg y la segunda 3mseg

;*****
;*****

;Define constantes utilizadas por el programa

TIEMPO1 .EQU 03H      ;Número de veces que se repite el lazo 1
TIEMPO2 .EQU 02H      ;Número de veces que se repite el lazo 2

;*****
;*****

;El siguiente código representa al programa principal

.ORG 0000H           ;Inicia en la posición de memoria 0000H

MVI    A, 01H        ;Carga registro de prueba
REPITE:
CALL   DEMORA1       ;Llama rutina 1
RLC    ;Rota el acumulador 1 bit a la izquierda
CALL   DEMORA2       ;Llama rutina 2
RLC    ;Rota el acumulador 1 bit a la izquierda
JMP    REPITE        ;Salta a etiqueta REPITE

;*****
;*****

;El siguiente código representa una demora de tiempo de un lazo

.ORG 0040H           ;Inicia en la posición de memoria 0040H

DEMORA1:
MVI    B, TIEMPO1    ;Carga contador del lazo 1
LAZO1: DCR    B        ;Decrementa por uno el registro B
NOP    ;No hace nada
JNZ    LAZO1         ;Salta si el resultado no es cero
RET    ;Regresa al programa principal

;*****
;*****

;El siguiente código representa una demora de tiempo de dos lazos

.ORG 0060H           ;Inicia en la posición de memoria 0060H

DEMORA2:
MVI    B, TIEMPO1    ;Carga contador del lazo 1
LAZO3: MVI    C, TIEMPO2 ;Carga contador del lazo 2
LAZO2: DCR    C        ;Decrementa por uno el registro C

```

```

NOP                ;No hace nada
JNZ     LAZO2      ;Salta si el resultado no es cero
DCR     B          ;Decrementa por uno le registro B
JNZ     LAZO3      ;Salta si el resultado no es cero
RET                    ;Regresa al programa principal

```

```

;*****
;*****

```

.END

• **Ejemplo 3:**

;El siguiente programa realiza un sencillo juego de luces utilizando al 8155
;para tal motivo configura el puerto A como salida de datos.

```

;*****
;*****

```

;Define constantes utilizadas por el programa

```

REG_CONF     .EQU    00H        ;Dirección del registro de configuración del 8155
PORTA        .EQU    01H        ;Dirección del PORTA del 8155
TIEMPO       .EQU    01H        ;Establece el tiempo de la demora

```

```

;*****
;*****

```

;El siguiente código representa al programa principal

```

                .ORG    0000H        ;Inicia en la posición de memoria 0000H

                MVI     A, 0FH        ;Configura PORTA como salidas
                OUT     REG_CONF
                MVI     A, 01H        ;Envía 00000001b al puerto A
ROTAR:         OUT     PORTA
                CALL    DEMORA        ;Espera cierto tiempo
                RLC                    ;Rota el acumulador 1 bit a la izquierda
                JMP     ROTAR        ;Salta a la etiqueta ROTAR

```

```

;*****
;*****

```

;El siguiente código representa una demora de tiempo la cual nos permite observar más
;detenidamente la secuencia de luces.

```

                .ORG    0040H        ;Inicia en la posición de memoria 0040H

DEMORA:        MVI     B, TIEMPO     ;Carga contador del lazo 1
LAZO2:         MVI     C, 0FFH       ;Carga contador del lazo 2
LAZO1:         DCR     C            ;Decrementa por uno le registro C
                JNZ     LAZO1        ;Salta si el resultado no es cero
                DCR     B            ;Decrementa por uno le registro B
                JNZ     LAZO2        ;Salta si el resultado no es cero
                RET                    ;Regresa al programa principal

```

```

;*****
;*****

```

.END

EJERCICIOS

1. Desarrolle un programa que reemplace al esquema del ejercicio de diseño del módulo 8.
2. Desarrolle un programa que reemplace al esquema del ejercicio de diseño del módulo 9.
3. Desarrolle un programa que reemplace al esquema del ejercicio de diseño del módulo 10.
4. Desarrolle un programa que reemplace al esquema del ejercicio de diseño del módulo 12.

OBSERVACIONES PARA LA SIGUIENTE CLASE

- Realizar en hojas perforadas los ejercicios que falten o que el profesor indique.
- Revisar los temas tratados en este módulo.

