

EJERCICIOS con ORDENADOR

El 8085 es un microprocesador de Intel, aunque es muy antiguo lo utilizaremos ya que es más manejable que los actuales. Tiene 7 registros accesibles al usuario (A, al acumulador, B, C, D, E, H y L). Cada registro contiene un byte, ya que las direcciones de memoria requieren dos bytes, los registros PC (contador de programa) y SP (puntero de pila) son dobles (de 16 bits). Del mismo modo, para usar un puntero a memoria son necesarios dos registros (las parejas posibles son BC, DE y HL). El registro F es un registro especial que recoge el valor de los flags SZ-AC-P-CY (S: flag de signo, Z: flag de cero, AC: flag de llevada intermedia, P: flag de paridad, CY: flag de llevada).

El conjunto de instrucciones que contempla se recogen en la tabla. La expresión describe la instrucción. Los paréntesis representan contenido de registro, los corchetes representan contenido de memoria. La flecha indica dónde se guarda el resultado. La tercera columna indica sobre qué flags tiene efecto esa instrucción. En la primera columna están los mnemónicos, donde pone r, r1 o r2 hay que poner un registro (A, B, C...), donde pone rp hay que poner una pareja de registros (H, para HL; B, para BC o D, para DE). Donde pone M hay que dejar M (hace referencia a la posición de memoria indicada por el par de registros HL). Las instrucciones de rotación trabajan sobre el dato del acumulador. Las instrucciones de bifurcación pueden ser incondicionadas (jmp, call, ret) o condicionadas. En este segundo caso las letras ccc sustituyen la condición de salto (por ejemplo Jccc → JZ (salta si el resultado de la operación anterior ha sido cero), JNZ (salta si el resultado de la operación anterior no ha sido cero), etc.).

Para poder no solo programar sino además ver qué pasa dentro (cómo se modifican los registros, cómo se carga el programa en memoria etc.) utilizaremos un simulador. Se trata de un simulador que ocupa muy poco y es de distribución gratuita por lo que cualquiera puede instalárselo en casa: [Sim8085](#). Se escribe el programa, se ensambla (se carga el programa a partir de la posición de memoria 8000h, por lo que ojo con modificar los datos de estas posiciones de memoria en el programa) y se puede ejecutar de golpe (*run*), despacito (*slow run*) o paso a paso (*single step*). El asistente de código puede ser de utilidad cuando tenemos dudas con una instrucción.

Nota: si se quiere probar un código más corto que el anterior, se recomienda borrar la memoria (clear) antes de cargar el siguiente, para evitar que queden instrucciones en memoria llevando a resultados inesperados.

A continuación se proponen unos ejercicios para coger soltura con las diferentes instrucciones. El objetivo final es entender el funcionamiento interno del microprocesador y ser capaces de programar en ensamblador los programas propuestos al final.

Juego de instrucciones del 8085

NEMONICO	EXP. GRAF.	FLAGS
INSTRUCCIONES DE TRANSFERENCIA		
MOV r1,r2	(r1)←(r2)	NINGUNO
MOV r,M	(r)←[(HL)]	NINGUNO
MOV M,r	[(HL)]←(r)	NINGUNO
MVI r,byte	(r)←byte	NINGUNO
MVI M,byte	[(HL)]←byte	NINGUNO
LXI rp,doble	(rpl)←1º byte (rph)←2º byte	NINGUNO
LDA addr	(A)←[addr]	NINGUNO
STA addr	[addr]←(A)	NINGUNO
LHLD addr	(L)←[addr] (H)←[addr+1]	NINGUNO
SHLD addr	[addr]←(L) [addr+1]←(H)	NINGUNO
LDAX rp	(A)←[(rp)]	NINGUNO
STAX rp	[(rp)]←(A)	NINGUNO
XCHG	(H)↔(D) (L)↔(E)	NINGUNO
INSTRUCCIONES ARITMÉTICAS		
ADD r	(A)←(A)+(r)	TODOS
ADD M	(A)←(A)+[(HL)]	TODOS
ADI byte	(A)←(A)+byte	TODOS
ADC r	(A)←(A)+(r)+CY	TODOS
ADC M	(A)←(A)+[(HL)]+CY	TODOS
ACI byte	(A)←(A)+byte+CY	TODOS
SUB r	(A)←(A)-(r)	TODOS
SUB M	(A)←(A)-[(HL)]	TODOS
SUI byte	(A)←(A)-byte	TODOS
SBB r	(A)←(A)-(r)-CY	TODOS
SBB M	(A)←(A)-[(HL)]-CY	TODOS
SBI byte	(A)←(A)-byte-CY	TODOS
INR r	(r)←(r)+1	Z, S, P, AC
INR M	[(HL)]←[(HL)]+1	Z, S, P, AC
DCR r	(r)←(r)-1	Z, S, P, AC
DCR M	[(HL)]←[(HL)]-1	Z, S, P, AC
INX rp	(rp)←(rp)+1	NINGUNO
DCX rp	(rp)←(rp)-1	NINGUNO
DAD rp	(HL)←(HL)+(rp)	CY
DAA	Ajuste BCD de (A)	NINGUNO
INSTRUCCIONES LÓGICAS.		
ANA r	(A)←(A) and (r) (CY)←0, (AC)←1	TODOS
ANA M	(A)←(A) and [(HL)] (CY)←0, (AC)←1	TODOS
ANI byte	(A)←(A) and byte (CY)←0, (AC)←1	TODOS
XRA r	(A)←(A) xor (r)	TODOS
XRA M	(A)←(A) xor [(HL)]	TODOS
XRI byte	(A)←(A) xor byte	TODOS
ORA r	(A)←(A) or (r)	TODOS
ORA M	(A)←(A) or [(HL)]	TODOS
ORI byte	(A)←(A) or byte	TODOS
CMP r	(A)-(r)	TODOS
CMP M	(A)-[(HL)]	TODOS
CPI byte	(A)-byte	TODOS

NEMONICO	EXP. GRAF.	FLAGS
INSTRUCCIONES DE ROTACIÓN Y FLAGS		
RLC	Rotacion izqda	CY
RRC	Rotación dcha	CY
RAL	Rot. izqda. con CY	CY
RAR	Rot. dcha. con CY	CY
CMA	Comp. A1 de (A)	NINGUNO
CMC	Invierte (CY)	CY
STC	(CY)←1	CY
INSTRUCCIONES DE BIFURCACIÓN		
ccc=NZ salto si no cero (Z=0), ccc=Z salto si cero (Z=1), ccc=NC salto si no acarreo (CY=0), ccc=C salto si acarreo (CY=1), ccc=PO salto si paridad impar (P=0), ccc=PE salto si paridad par (P=1), ccc=P salto si positivo (S=0), ccc=M salto si negativo (S=1)		
JMP addr	(PC)←addr	NINGUNO
Jccc addr	Si ccc=1, (PC)←addr; Si ccc=0, (PC)←(PC)+3	NINGUNO
CALL addr	Guarda PC en la pila (PC)←addr	NINGUNO
Cccc addr	Si ccc=1, guarda PC en la pila, (PC)←addr; Si ccc=0, (PC)←(PC)+3	NINGUNO
RET	Recupera PC de la pila	NINGUNO
Rccc	Si ccc=1, recupera PC de la pila; Si ccc=0 (PC)←(PC)+1	NINGUNO
RSTn	(PC)←n x 8	NINGUNO
PCHL	(PC)←(HL)	NINGUNO
INSTRUCCIONES DE MANEJO DE LA PILA.		
PUSH rp	[(SP)-1]←(rpl) [(SP)-2]←(rph) (SP)←(SP)-2	NINGUNO
PUSH PSW	[(SP)-1]←(A) [(SP)-2]←(RE) (SP)←(SP)-2	NINGUNO
POP rp	(rph)←[(SP)] (rpl)←[(SP)+1] (SP)←(SP)+2	NINGUNO
POP PSW	(RE)←[(SP)] (A)←[(SP)+1] (SP)←(SP)+2	NINGUNO
XLH	(L)↔[(SP)] (H)↔[(SP)+1]	NINGUNO
SPLH	(HL)↔(SP)	NINGUNO
INSTRUCCIONES DE ENTRADA Y SALIDA.		
IN puerta	(A)←[puerta]	NINGUNO
OUT puerta	[puerta]←(A)	NINGUNO
INSTRUCC. DE CONTROL DE INTERRUPCIONES		
EI	Habilita interrupciones	NINGUNO
DI	Inhabilita interrupciones	NINGUNO
HLT	Para el microprocesador	NINGUNO
NOP	No hace nada	NINGUNO
RIM	Lee linea serie y estado interrupciones.	NINGUNO
SIM	Escribe en linea serie y Programa interrupciones.	NINGUNO

RE=Status Register

PSW=Processor Status Word

REPASO HEXADECIMAL

En el 8085 se utiliza la **base hexadecimal**.

- ¿Cuáles son los valores que puede tomar un número hexadecimal?
- ¿Cuántas cifras binarias hacen falta para representar una cifra hexadecimal?
- ¿Cuántas cifras hexadecimales tienen los registros? ¿Por tanto, cuál es el valor máximo que pueden contener en base 16, en base 2 y en base 10?
- ¿Cuántas cifras hacen falta para especificar una posición de memoria? Entonces, ¿cuántos registros hacen falta para utilizarlos como apuntador de memoria?

INSTRUCCIONES DE TRANSFERENCIA

Address	Data	Address	Data	Address	Data	Addr
8000	00	9000	00	9100	00	FE00
8001	00	9001	01	9101	00	FE01
8002	00	9002	02	9102	00	FE02
8003	00	9003	03	9103	00	FE03
8004	00	9004	04	9104	00	FE04
8005	00	9005	05	9105	00	FE05
8006	00	9006	06	9106	00	FE06
8007	00	9007	07	9107	00	FE07
8008	00	9008	08	9108	00	FE08
8009	00	9009	09	9109	00	FE09
800A	00	900A	0A	910A	00	FE0A
800B	00	900B	0B	910B	00	FE0B
800C	00	900C	0C	910C	00	FE0C
800D	00	900D	0D	910D	00	FE0D
800E	00	900E	0E	910E	00	FE0E
800F	00	900F	0F	910F	00	FE0F
8010	00	9010	10	9110	00	FE10
8011	00	9011	11	9111	00	FE11
8012	00	9012	12	9112	00	FE12

- En la situación del 8086 indicada por la figura de arriba, si se ejecuta la siguiente instrucción, ¿cuáles serían los nuevos valores de los registros del microprocesador? Rellenad la tabla siguiente para cada instrucción partiendo de la situación actual (no para las tres instrucciones seguidas).

		A	B	C	D	E	H	L
a)	mov b,d							
b)	mov a,m							
c)	mov m,e							

- d) Una de estas tres instrucciones no modifica ningún registro, sino una posición de memoria. ¿Cuál es? ¿Cuál es la posición de memoria modificada? ¿Cuál es el cambio que experimenta el contenido de esa posición de memoria?
- e) ¿Qué es el registro **F** (qué información contiene)? ¿Las operaciones indicadas arriba lo modifican? ¿Por qué?
- f) ¿Qué es el registro **PC**? ¿Las operaciones indicadas arriba lo modifican? ¿Por qué y cómo?
2. Queremos llevar el dato de la posición de memoria 9003 a la posición 9008. Realiza esta operación con dos instrucciones.
3. Partiendo de la situación indicada en el ejercicio 1, explica qué ocurrirá al ejecutar las siguientes instrucciones (qué cambia y cómo). Suponer que se ejecutan todas las instrucciones seguidas.
- a) **sta** 9001
- b) **lda** 9001
- c) **shld** 9001
- d) **lhld** 9001
4. En el ejercicio 1, ¿qué pareja de registros se ha utilizado como apuntador de memoria? ¿Cómo se puede inicializar? Es decir, si queremos utilizar la posición 900A como apuntador de memoria, ¿qué operación debemos realizar?
Nota: Si tenemos un array de datos en memoria, podemos inicializar el apuntador con la dirección del primer elemento del array, y luego podemos recorrer todo el array incrementando en uno el contenido del apuntador.
5. ¿Cuál es la diferencia entre las siguientes instrucciones? (Explica qué hace cada una)
- a) **lda addr** $(A) \leftarrow [addr]$
- b) **ldax rp** $(A) \leftarrow [(rp)]$
- c) **sta addr** $[addr] \leftarrow (A)$
- d) **stax rp** $[(rp)] \leftarrow (A)$
6. ¿Cuál es el resultado de ejecutar la instrucción **xchg**? ¿Para qué función resulta útil?

INSTRUCCIONES ARITMÉTICAS

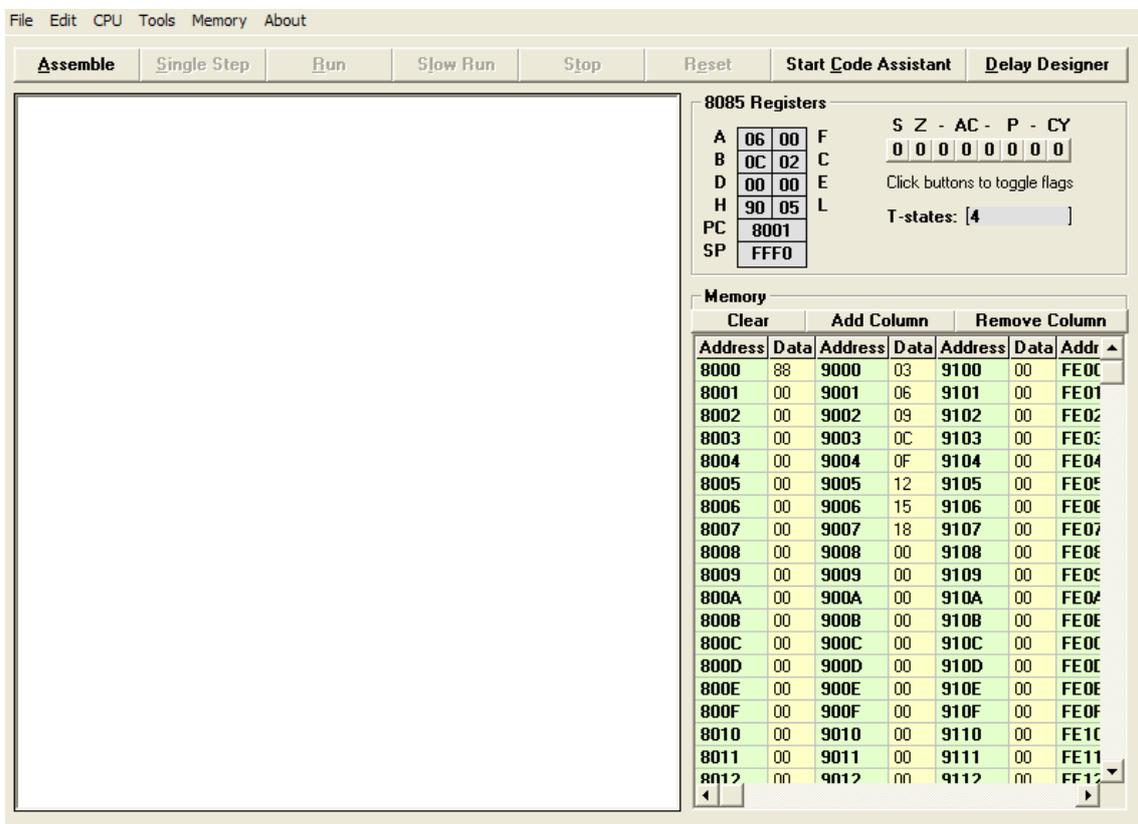
En la mayoría de las instrucciones aritméticas interviene el acumulador. En la instrucción se indica un operando mediante un dato, un registro o una posición de memoria de manera explícita, y para definir el otro operando, siempre se utiliza el acumulador. El resultado de la operación se escribe en el acumulador, de modo que se pierde el dato escrito en el acumulador antes de ejecutar la instrucción. Por tanto:

- a) Si queremos volver a utilizar el dato contenido en el acumulador, debemos guardarlo antes de ejecutar la instrucción (en un registro o en una posición de memoria, mediante instrucciones de transferencia).
- b) El resultado sólo se guarda en el acumulador, por lo que debemos copiarlo en otro sitio antes de perderlo por descuido en la siguiente instrucción.

A diferencia de las instrucciones de transferencia, las aritméticas activan casi todos los flag (indicadores de estado).

1. Explica lo que realizan las siguientes instrucciones y, suponiendo que la situación al comienzo es la que indica la imagen (la primera viene hecha a modo de ejemplo), ¿cuál sería el resultado que se escribiría en el acumulador?

NOTA: (no se ejecuta una instrucción tras otra)



- a) ADD b
- b) ADD c
- c) ADD a
- d) ADD m

- e) ADI 08
- f) ADC b
- g) ADC c
- h) ADC m
- i) ACI 08
- j) SUB b
- k) SUB c
- l) SUB a
- m) SUB m
- n) SUI 08
- o) SBB b
- p) SBB c
- q) SBB m
- r) SBI 08

2. A partir de la posición 9000h hay un array de 12 elementos. Supongamos que se establece un bucle que, con ayuda de un contador, va incrementando los elementos de array uno por uno. Escribe las instrucciones necesarias para realizar lo siguiente:

- a) Inicializar la pareja de registros que usaremos como apuntador.
- b) Incrementar dentro del bucle el i-ésimo elemento (el que toque) del array.
- c) Actualizar el apuntador (para que apunte al siguiente elemento del array).

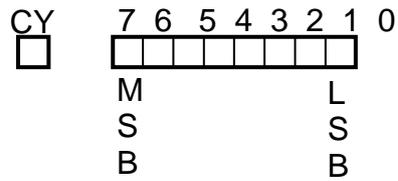
INSTRUCCIONES LÓGICAS

- 1) ¿Cuáles son las instrucciones lógicas que afectan a los flags? ¿A qué flags afectan?
- 2) a) Escribe la tabla de la verdad de las operaciones lógicas fundamentales AND, OR, XOR y NOT.

b) ¿Cómo se puede realizar una operación NOT mediante una puerta XOR?
- 3) En el 8085 el grupo de instrucciones lógicas lo forman AND, XOR, OR y CMP. CMP realiza la comparación entre un registro (CMP r), el contenido de una posición de memoria (CMP M) o un valor concreto (CPI byte) y el valor de acumulador. ¿Perdemos el valor contenido en el acumulador antes de la ejecución de la instrucción? ¿Quién contiene el resultado? ¿Cómo podemos utilizar el resultado de esta operación?
- 4) MÁSCARAS. Tenemos un número binario desconocido de 8 dígitos (XXXXXXXX₂). Utilizando las operaciones lógicas:
 - a) ¿Cómo ponemos todos los bits a '1' (11111111₂)?
 - b) ¿Cómo ponemos todos los bits a '0' (00000000₂)?
 - c) ¿Cómo ponemos los cuatro bits más significativos a '1', sin cambiar el valor de los otros cuatro bits?
 - d) ¿Cómo ponemos los cuatro bits más significativos a '0', sin cambiar el valor de los otros cuatro bits?
 - e) ¿Cómo ponemos todos los bits a '0' salvo el MSB (éste se mantiene como estaba)? ¿Cuáles son los resultados posibles de esta operación?
- 5) Supongamos que el número binario desconocido del ejercicio anterior es 100111012. Escribe en hexadecimal cuáles deben ser las máscaras y los resultados para los apartados de a) a e) del ejercicio anterior.
- 6) Con el dato guardado en la posición de memoria 9002, realiza las siguientes operaciones:
 - a) Anular todos sus bits y guardar el resultado en la posición 9003.
 - b) Sin cambiar el LSB, pon todos sus otros bits a '1' y guarda el resultado en la posición 9004.
 - c) Sin cambiar el MSB, pon todos sus otros bits a '0' y guarda el resultado en la posición 9005.
- 7) Se quiere saber si el dato de la posición de memoria 9100 es mayor o menor que el de la posición de memoria 9101. ¿Qué podemos hacer (empleando operaciones lógicas) para saberlo? Repetir (con el dato de la posición 9100) para el dato que contiene el registro B y para el número 12₁₀.

INSTRUCCIONES DE ROTACIÓN

1) Representa el efecto que tienen las instrucciones RLC, RRC, RAL y RAR en los bits del registro.



2) El dato que rota, ¿en qué registro está?

3) El contenido del registro es 40h. ¿Cuál es la diferencia entre realizar dos instrucciones RLC seguidas y ejecutar dos instrucciones RAL seguidas? Repetir si el contenido del registro es 01h y las instrucciones que se ejecutan seguidas (una vez) son RRC y RAR.

4) Si el contenido del registro es 10001011_2 , ¿cuál es su valor decimal?
¿Cuál será el resultado de ejecutar RLC y RRC (en base binaria y en base decimal)?
¿Y si el contenido es 00010000_2 (valor decimal y resultados de las rotaciones)?
Si en lugar de rotar realizamos desplazamiento y en las posiciones del bit entrante escribimos '0', ¿qué conseguimos si desplazamos un número binario una posición a la izquierda? ¿Y si los desplazamos a la derecha?

5) ¿Cuál es el complemento a uno de un número (definición y método práctico)? ¿Y el complemento a dos?

6) Diferencias entre las instrucciones CMA y CMC (explica qué hace cada una).

INSTRUCCIONES DE BIFURCACIÓN

1- Rellena la siguiente tabla con el significado de los valores de los bits de estado (flags).

Bit de estado	Valor	Significado: el resultado de la operación anterior es...
S	0	
S	1	
Z	0	
Z	1	
P	0	
P	1	
CY	0	
CY	1	

2- ¿Cómo ha sido el resultado de la operación anterior si el valor del registro F es el siguiente?

S Z - AC - P - CY
F

--	--	--	--	--	--	--	--

₂

a) F=04h

b) F=11h

c) F=C4h

3- Si el resultado de la anterior operación es negativo, salta a la etiqueta NEGATIVO

4- Si el resultado es negativo, salta a NEG; si no salta a NO_NEG

EJERCICIOS COMPLETOS

1- ¿Qué hace el siguiente programa?

```
mvi a,00
bucle:
lda 9000
cpi 00
jz fin
jm negativo
jmp bucle
negativo:
;aquí habrá unas operaciones que omitimos
fin:
```

2- ¿Qué hace el siguiente programa?

```
mvi a,00
bucle:
lda 9000
mov b,a
lda 9001
sub b
jz fin
jmp diferentes
diferentes:
;aquí habrá unas operaciones que omitimos
fin:
```

3- En la posición 9000h está el primer elemento de un array. Este array es de 10 elementos y todos son positivos. Escribe un programa que recorra todos los elementos del array decrementando en 1 su valor. Cuando llegue al final del array empezará de nuevo, decrementando en otra unidad todos los elementos del array. El programa se repetirá hasta que algún elemento llegue a cero.

4- Explica qué hace el siguiente programa, cada instrucción y el conjunto.

```
lhld 9100
mvi b, 00
mvi c, 10
mvi d, 08
bucle:
mov a,d
cmp m
jm siguiente
inr b
siguiente:
inx hl
dcr c
jz fin
jmp bucle
fin:
```

SOLUCIONES

REPASO HEXADECIMAL

En el 8086 se utiliza la **base hexadecimal**.

- a) [Sol]: puede tomar 16 valores diferentes: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- b) [Sol]: 4 cifras binarias.
- c) [Sol]: Los registros tienen 8 bits, por tanto 2 cifras hexadecimales; El valor máximo que pueden contener será: $(FF)_{16}$, $(1111\ 1111)_2$ y $(255)_{10}$
- d) [Sol]: Se multiplexan en el tiempo el bus de datos (8 bits) y el bus de direcciones (8 bits) para poder direccionar 64K posiciones de memoria. Son necesarios **2 registros** para usarlos como apuntador de memoria.

INSTRUCCIONES DE TRANSFERENCIA

1.

		A	B	C	D	E	H	L
a)	mov b,d	05	45	0A	45	54	90	05
b)	mov a,m	05	90	0A	45	54	90	05
c)	mov m,e	05	90	0A	45	54	90	05

- d) [Sol]: La instrucción c) sólo modifica la posición de memoria apuntada por el HL, en este caso la 9005. En esa posición de memoria se escribirá el contenido del registro E, es decir, 54.
- e) ¿[Sol]: El registro F contiene la información de los flags de estado (S, Z, AC, P, CY). Las instrucciones indicadas en este ejercicio son todas de transferencia, por lo que no realizan ninguna operación que modifique los flags de estado.
- f) [Sol]: El registro PC contiene la dirección de la siguiente instrucción a ejecutar. Por tanto, ninguna de las instrucciones de este ejercicio lo modifica directamente. Sin embargo, la ejecución de cualquier instrucción provoca el cambio de este registro automáticamente, de modo que apunte a la dirección de la siguiente instrucción después de haberse ejecutado la actual.

2.

[Sol]: **lda 9003** (A)←[9003]
 sta 9008 [9008]←(A)

3.

a) **sta 9001**

[Sol]: **[9001]←(A)**; El contenido del registro A, 05h, se escribe en la dirección de memoria 9001h (sólo cambia la memoria).

b) **lda 9001**

[Sol]: **(A)←[9001]**; El contenido de la dirección de memoria 9001h, 01h, se escribe en el registro A (sólo cambia el registro A).

c) **shld 9001**

[Sol]: **[9001]←(L) y [9002]←(H)**; El contenido del registro L , 05h, se escribe en la dirección de memoria 9001h y el contenido del registro H , 90h, se escribe en la dirección de memoria 9002h (sólo cambia la memoria).

d) **lhld 9001**

[Sol]: **(L)←[9001] eta (H)←[9002]**, El contenido de la dirección de memoria 9001h, 01h se escribe en el registro L y el contenido de la dirección de memoria 9002h, 02h, se escribe en el registro H (sólo cambian los registros H y L).

4.

[Sol]: Se ha utilizado la pareja de registros HL. Para inicializarla:

lxi h, 900A (HL)←900Ah

5.

a) **lda addr (A) ← [addr]**

[Sol]: Escribe en el registro A, el contenido de la posición de memoria de dirección addr.

b) **ldax rp (A) ← [(rp)]**

[Sol]: Escribe en el registro A, el contenido de la posición de memoria cuya dirección está en la pareja de registros rp;

c) **sta addr [addr] ← (A)**

[Sol]: Escribe en memoria en la posición de dirección addr, el contenido del registro A.

d) **stax rp [(rp)] ← (A)**

[Sol]: Escribe en memoria, en la posición de memoria cuya dirección está en el registro doble rp, el contenido del registro A.

Por tanto, en los casos a y c se indica directamente la dirección de memoria, mientras que en los casos b y d se emplea un registro doble como puntero a una dirección de memoria.

6.

[Sol]: La instrucción xchg intercambia el contenido de las parejas de registros HL y DE. Esta operación resulta útil cuando utilizamos dos apuntadores de memoria diferentes mediante las dos parejas de registros. Como la pareja HL es la única que se puede utilizar en las instrucciones mov, podemos intercambiar los apuntadores para los accesos a memoria y luego reintegrarlos a los valores anteriores, sin perder ninguno de los dos.

(H)←→(D) y (L)←→(E)

INSTRUCCIONES ARITMÉTICAS

1.

a) ADD b

[Sol]: **(A) ← (A)+(B)**; 12h ←06h+0Ch; En el acumulador había 06h, en el registro B había 0Ch (12₁₀), por tanto, el resultado que se queda en el acumulador es: (6₁₀+12₁₀=18₁₀) =12h.

b) ADD c

[Sol]: **(A) ← (A)+(C)**; 08h ← 06h+02h; En el acumulador había 06h, en el registro C había 02h (2₁₀), por tanto, el resultado que se queda en el acumulador es: (6₁₀+2₁₀=8₁₀) =08h.

c) ADD a

[Sol]: **(A) ← (A)+(A)**; 0Ch ←06h+06h; En el acumulador había 06h, por tanto, el resultado que se queda en el acumulador es:(6₁₀+6₁₀=12₁₀) =0Ch.

d) ADD m

[Sol]: $(A) \leftarrow (A)+[(HL)]$; $18h \leftarrow 06h+12h$; En el acumulador había $06h$, en la posición de memoria apuntada por el registro HL ($9005h$), había $12h$ (18_{10}), por tanto, el resultado que se queda en el acumulador es: $(6_{10}+18_{10}=24_{10}) =18h$.

e) ADI 08

[Sol]: $(A) \leftarrow (A)+08h$; $0Eh \leftarrow 06h+08h$; En el acumulador había $06h$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}+8_{10}=14_{10}) =0Eh$.

f) ADC b

[Sol]: $(A) \leftarrow (A)+(B)+(CY)$; $12h \leftarrow 06h+0Ch+0b$; En el acumulador había $06h$, en el registro B había $0Ch$ (12_{10}) y el bit de carry había $0b$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}+12_{10}+0_{10}=18_{10}) =12h$.

g) ADC c

[Sol]: $(A) \leftarrow (A)+(C)+(CY)$; $08h \leftarrow 06h+02h+0b$; En el acumulador había $06h$, en el registro C había $02h$ y el bit de carry había $0b$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}+2_{10}+0_{10}=8_{10}) =08h$.

h) ADC m

[Sol]: $(A) \leftarrow (A)+[(HL)]+(CY)$; $18h \leftarrow 06h+12h+0b$; En el acumulador había $06h$, en la posición de memoria apuntada por el registro HL ($9005h$), había $12h$ (18_{10}) y el bit de carry había $0b$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}+18_{10}+0_{10}=24_{10}) =18h$.

i) ACI 08

[Sol]: $(A) \leftarrow (A)+08h+(CY)$; $0Eh \leftarrow 06h+08h+0b$; En el acumulador había $06h$ y el bit de carry había $0b$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}+8_{10}+0_{10}=14_{10}) =0Eh$.

j) SUB b

[Sol]: $(A) \leftarrow (A)-(B)$; $FAh \leftarrow 06h-0Ch$; En el acumulador había $06h$, en el registro B había $0Ch$ (12_{10}), por tanto, el resultado que se queda en el acumulador es: $(6_{10}-12_{10}=-6_{10}) =FAh$.

k) SUB c

[Sol]: $(A) \leftarrow (A)-(C)$; $04h \leftarrow 06h-02h$; En el acumulador había $06h$, en el registro C había $02h$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}-2_{10}=4_{10}) =04h$.

l) SUB a

[Sol]: $(A) \leftarrow (A)-(A)$; $00h \leftarrow 06h-06h$; En el acumulador había $06h$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}-6_{10}=0_{10}) =00h$.

m) SUB m

[Sol]: $(A) \leftarrow (A)-[(HL)]$; $F4h \leftarrow 06h-12h$; En el acumulador había $06h$, en la posición de memoria apuntada por el registro HL ($9005h$), había $12h$ (18_{10}), por tanto, el resultado que se queda en el acumulador es: $(6_{10}-18_{10}=-12_{10}) =F4h$.

n) SUI 08

[Sol]: $(A) \leftarrow (A)-08h$; $FEh \leftarrow 06h-08h$; En el acumulador había $06h$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}-8_{10}=-2_{10}) =FEh$.

o) SBB b

[Sol]: $(A) \leftarrow (A)-(B)-(CY)$; $FAh \leftarrow 06h-0Ch-0b$; En el acumulador había $06h$, en el registro B había $0Ch$ (12_{10}) y el bit de carry había $0b$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}-12_{10}-0_{10}=-6_{10}) =FAh$.

p) SBB c

[Sol]: $(A) \leftarrow (A)-(C)-(CY)$; $04h \leftarrow 06h-02h-0b$; En el acumulador había $06h$, en el registro C había $02h$ y el bit de carry había $0b$, por tanto, el resultado que se queda en el acumulador es: $(6_{10}-2_{10}-0_{10}=4_{10}) =04h$.

q) SBB m

[Sol]: $(A) \leftarrow (A) - [(HL)] - (CY)$; $F4h \leftarrow 06h - 12h - 0b$; En el acumulador había 06h, en la posición de memoria apuntada por el registro HL (9005h), había 12h (18_{10}) y en el bit de carry había 0b, por tanto, el resultado que se queda en el acumulador es: $(6_{10} - 18_{10} - 0_{10} = -12_{10}) = F4h$.

r) SBI 08

[Sol]: $(A) \leftarrow (A) - 08h - (CY)$; $FEh \leftarrow 06h - 08h - 0b$; En el acumulador había 06h y en el bit de carry había 0b, por tanto, el resultado que se queda en el acumulador es: $(6_{10} - 8_{10} - 0_{10} = -2_{10}) = FEh$.

2.

a) [Sol]: Escogemos, por ejemplo, la pareja de registros HL, por lo que cargaremos en ella el valor de la primera posición del array:

lxi h, 9000 (HL) \leftarrow 9000h

b) [Sol]: Si el apuntador está actualizado, el HL contiene la dirección del pésimo elemento, por tanto, se puede incrementar así:

inr m [(HL)] \leftarrow [(HL)]+1

c) [Sol]: Si el actual valor del apuntador es el correspondiente a la posición de memoria del elemento actual del array, como todos los elementos del array están en posiciones consecutivas de memoria, basta con aumentar en uno el valor del apuntador HL para que apunte al siguiente elemento del array:

inx h (HL) \leftarrow (HL)+1

INSTRUCCIONES LÓGICAS

1) [Sol]: Todas las instrucciones lógicas afectan a **todos los flags**.

2) a)[Sol]:

AND			OR			XOR			NOT	
X	Y	F	X	Y	F	X	Y	F	X	Y
0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	1	0
1	0	0	1	0	1	1	0	1		
1	1	1	1	1	1	1	1	0		

2) b) [Sol]: En la tabla de la verdad de XOR se puede ver que, si la x es 1, la F es siempre la negada de la y, por tanto basta con asegurar que **uno de los operandos de la puerta XOR sea siempre 1**.

3) [Sol]: El resultado de la instrucción CMP no se guarda, por lo que **el valor contenido en el acumulador no cambia** después de la ejecución de la instrucción. El único efecto de la ejecución de la instrucción es la modificación de todos **los flags de estado**, por tanto en ellos aparece el valor del resultado. Los valores de los flags de estado definen el resultado de las **instrucciones de salto condicional**.

4)

a) [Sol]: Como $A+1=1$, nos basta con hacer **ORI FFh** ($FF_{16}=(11111111_2)$).

b) [Sol]: Como $A-0=0$, hacemos **ANI 00h** ($00_{16}=(00000000_2)$).

c) [Sol]: Como $A+1=1$ y $A+0=A$, hacemos **ORI F0h** ($F0_{16}=(11110000_2)$).

d) [Sol]: Como $A-0=0$ y $A-1=A$, hacemos **ANI 0Fh** ($0F_{16}=(00001111_2)$).

e) [Sol]: Como $A-0=0$ y $A-1=A$, hacemos **ANI 80h** ($80_{16}=(10000000_2)$).

Si $MSB=0$, el resultado es **00h**, pero si $MSB=1$, el resultado es **80h**.

5) [Sol]:

a) Máscara FFh, resultado FFh.

b) Máscara 00h, resultado 00h.

c) Máscara F0h, resultado FDh.

d) Máscara 0Fh, resultado 0Dh.

e) Máscara 80h, resultado 80h

6) [Sol]:

a)
LDA 9002h
ANI 00h
STA 9003h

b)
LDA 9002h
ORI FEh
STA 9004h

c)
LDA 9002h
ANI 80h
STA 9005h

7) [Sol]:

a) lda 9100 (A) ← [9100]
 lxi h, 9101 (HL) ← 9101
 cmp m (A)-[(HL)]

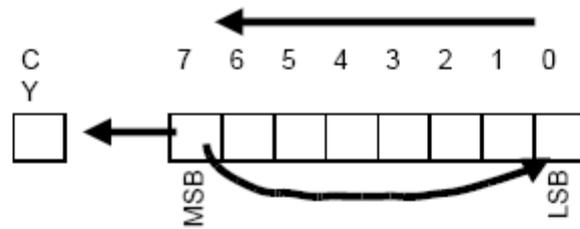
b) lda 9100 (A) ← [9100]
 cmp b (A)-(B)

c) lda 9100 (A) ← [9100]
 cpi 0C (A) - 0Ch (0Ch=12₁₀)

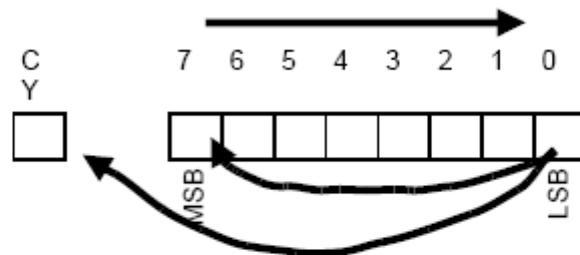
INSTRUCCIONES DE ROTACIÓN

1) [Sol]:

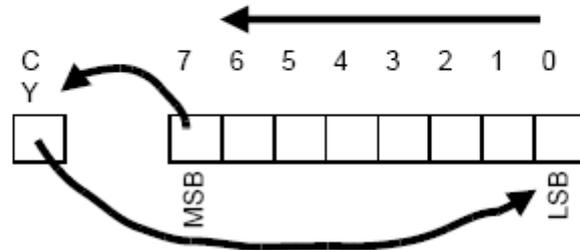
RLC



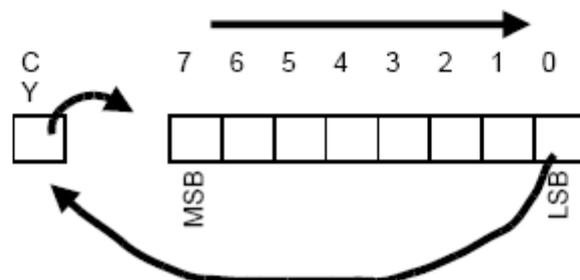
RRC



RAL



RAR



2) [Sol]: En el acumulador, en el registro A.

3) El contenido del registro es 40h. ¿Cuál es la diferencia entre realizar dos instrucciones RLC seguidas y ejecutar dos instrucciones RAL seguidas? Repetir si el contenido del registro es 01h y las instrucciones que se ejecutan seguidas (una vez) son RRC y RAR.

[Sol]:

$$A=01000000_2=40_{16}=64_{10}$$

$$\mathbf{RLC\ A=10000000_2=80_{16}=128_{10}\ CY=0}$$

$$\mathbf{RLC\ A=00000001_2=01_{16}=01_{10}\ CY=1}$$

$$A=01000000_2=40_{16}=64_{10} \text{ Suponiendo que } CY=0 \text{ antes de ejecutarlas:}$$

$$\mathbf{RAL\ A=10000000_2=80_{16}=128_{10}\ CY=0}$$

$$\mathbf{RAL\ A=00000000_2=00_{16}=0_{10}\ CY=1}$$

$$A=00000001_2=01_{16}=1_{10}$$

$$\mathbf{RRC\ A=10000000_2=80_{16}=128_{10}\ CY=1}$$

$$\text{RAR } A=11000000_2=C0_{16}=192_{10} \text{ CY}=0$$

- 4) Si el contenido del registro es 10001011_2 , ¿cuál es su valor decimal?
¿Cuál será el resultado de ejecutar RLC y RRC (en base binaria y en base decimal)?
¿Y si el contenido es 00010000_2 (valor decimal y resultados de las rotaciones)?
Si en lugar de rotar realizamos desplazamiento y en las posiciones del bit entrante escribimos '0', ¿qué conseguimos si desplazamos un número binario una posición a la izquierda? ¿Y si los desplazamos a la derecha?

[Sol]:

$$\begin{aligned} A &= 10001011_2 = 8B_{16} = 139_{10} \\ \text{RLC } A &= 00010111_2 = 17_{16} = 23_{10} \text{ CY}=1 \\ \text{RRC } A &= 11000101_2 = C5_{16} = 197_{10} \text{ CY}=1 \end{aligned}$$

$$\begin{aligned} A &= 00010000_2 = 10_{16} = 16_{10} \\ \text{RLC } A &= 00100000_2 = 20_{16} = 32_{10} \text{ CY}=0 \\ \text{RRC } A &= 00001000_2 = 08_{16} = 8_{10} \text{ CY}=0 \end{aligned}$$

Al desplazar una posición a la izquierda, todos los bits pasan a ocupar una posición de mayor peso, el resultado es que el número se multiplica por dos, pero como el MSB desaparece, hay que restar ese valor: $\text{shl } (N) = 2 \cdot N - b_{n-1} \cdot 2^n$

Al desplazar una posición a la derecha, todos los bits pasan a ocupar una posición de menor peso, el resultado es que el número se divide entre dos, pero como el lsb desaparece, hay que restar ese número: $\text{shr } (N) = N/2 - b_0 \cdot 2^{-1}$

- 5) ¿Cuál es el complemento a uno de un número (definición y método práctico)? ¿Y el complemento a dos?

[Sol]:

Complemento a uno de A:
 $2^n - 1 - |A|$; n: número de bits de A.
Método práctico: **Invertir todos los bits de A.**

Complemento a dos de A:
 $2^n - |A|$; n: número de bits de A.
Método práctico: **Sumar uno al complemento a uno de A.**

- 6) Diferencias entre las instrucciones CMA y CMC (explica qué hace cada una).

[Sol]:

La instrucción **CMA** realiza el **NOT** de todos los bits contenidos en el **registro A**.
La instrucción **CMC** sólo realiza el **NOT** del contenido del **flag de carry CY**.

INSTRUCCIONES DE BIFURCACIÓN

1-Rellena la siguiente tabla con el significado de los valores de los bits de estado (flags).

Bit de estado	Valor	Significado: el resultado de la operación anterior es...
S	0	Positivo
S	1	Negativo
Z	0	Distinto de 0
Z	1	Cero
P	0	Número impar de unos
P	1	Número par de unos
CY	0	Sin acarreo de la última cifra
CY	1	Con acarreo de la última cifra

2-

a) F=04h

[Sol]: F=04h=00000100₂ ; S=0; Z=0; AC=0; P=1; CY=0;

El resultado tiene un número par de 1s, es positivo y distinto de cero y no hay acarreo.

b) F=11h

[Sol]: F=11h=00010001₂ ; S=0; Z=0; AC=1; P=0; CY=1;

El resultado tiene un número impar de 1s, es positivo y distinto de cero y hay acarreo de la última cifra y de la cuarta.

c) F=C4h

[Sol]: F=C4h=11000100₂ ; S=1; Z=1; AC=0; P=1; CY=0;

No es posible que S y Z estén a uno al mismo tiempo.

3- [Sol]: jm NEGATIVO
(...)
NEGATIVO:

4- [Sol]: jm NEG
jmp NO_NEG
(...)
NEG:
(...)
NO_NEG:
(...)

EJERCICIOS COMPLETOS

1-

```
mvi a,00          Se inicializa el acumulador a cero,
bucle:
lda 9000         se lee el contenido de 9000h [a←(9000)]
cpi 00          y se compara con cero
jz fin          si son iguales, fin del programa
jm negativo     si es menor que cero, salta a negativo
jmp bucle       si es positivo, vuelve a ejecutar el bucle
negativo:
;aquí habrá unas operaciones que omitimos
fin:
```

El programa comprueba el dato en la posición 9000h mientras sea positivo; si es negativo ejecutará una parte de código (omitido), y si es cero, termina el programa.

2-

```
mvi a,00          El acumulador se inicializa a cero,
bucle:
lda 9000         se lee el dato de 9000 al acumulador,
mov b,a         lo guarda en b,
lda 9001         se lee el dato de 9001 al acumulador,
sub b          se comparan (a←a-b) y
jz fin          si son iguales fin del programa,
jmp diferentes en caso contrario salta a diferentes
diferentes:
;aquí habrá unas operaciones que omitimos
fin:
```

El programa compara los datos en las posiciones 9000h y 9001h y ejecuta un código (aquí omitido) solo si son diferentes.

3- [Sol]:

```
inicio:
lxi hl,9000
mvi b,0a
bucle:
dcr m
jz fin
inx hl
dcr b
jnz bucle
jmp inicio
fin:
```

4- [Sol]:

```
lhld 9100 ; contenido de dirección 9100 al registro L; y la de 9101 al registro H
mvi b, 00 ; inicializo registro b a 0
mvi c, 10 ; inicializo registro c a 10
```

```

mvi d, 08 ; inicializo registro d a 08
bucle:
mov a,d ; (a)← (d)=08h
cmp m ; (a)-[(HL)] comparación: ; el contenido del acumulador (8) y el contenido del
primer elemento del array que empieza en la posición indicada en 9100 y 9101
jm siguiente; si el resultado es negativo (8 > X); salta a "siguiente" sin incrementar b
inr b ;si no (8 < X) incrementa b
siguiente:
inx hl ; incrementa el apuntador
dcr c ; decrementa el registro c
jz fin ; si el contenido de c es cero salta a "fin"
jmp bucle ; de lo contrario repite el bucle
fin:

```

La dirección de memoria formada por los dos bytes contenidos en las posiciones 9100h y 9101h es la del primer elemento de un array. Este array tiene 16 elementos y por eso introducimos 10h en C, para usarlo como contador al recorrer el array. El registro B también será un contador, en este caso muestra el número de elementos del array que son menores que 08.

Ejercicios individuales (sin solución)

Instrucciones de transferencia

1. Coge el dato de la posición de memoria 9101h y guárdalo en el acumulador (registro A).
2. Guarda el dato contenido en el acumulador en la posición de memoria 9000h.
3. Carga el dato de la posición de memoria 9000h en el registro L y el dato de la posición de memoria 9001h en el registro H.
4. Inicializar los registros HL como punteros a la posición de memoria 900Ah (Notar la diferencia con el anterior y que las instrucciones en las que interviene una pareja de registros tienen una **x**).
5. Llevar el dato de la posición de memoria 9001h a la posición de memoria 9101h (Notar que las transferencias son entre registros o entre registro y memoria; este movimiento requiere dos instrucciones).
6. Carga en el acumulador el contenido de la posición de memoria cuya dirección está en la pareja de registros B,C (empleando los registros BC como punteros).
7. Guarda en la posición de memoria, cuya dirección está en la pareja de registros B,C, el contenido del registro A (acumulador).
8. Intercambia el contenido de las parejas de registros HL y DE.
9. Guarda el contenido del registro B en el registro E.
10. Guarda el contenido de la posición de memoria apuntada por los registros HL, en el registro A.

11. Guarda en la posición de memoria apuntada por los registros HL, el contenido del registro C (HL es una pareja típicamente usada como puntero a memoria, de ahí las instrucciones con M, donde M hace referencia a la posición de memoria indicada por HL).

12. Supongamos que se ejecutan las siguientes instrucciones: mov E,B; mov A,m y mov m,C ¿Modifican estas instrucciones al registro F?, ¿Y al registro PC?,¿Por qué?

13. Guarda el dato 05h en el acumulador (Notar que las instrucciones con un dato inmediato tienen una i).

14. Guardar el dato 05h en la posición de memoria 9001h sin emplear punteros

15. Guardar el dato 05h en la posición de memoria 9001h empleando punteros

Instrucciones aritméticas

1. Suma el contenido del Registro B al Acumulador y a ese valor Réstale el valor 7_{10} (07h).

2. Suma el contenido de la posición de memoria apuntada por los registros HL al acumulador y a ese valor Réstale el contenido del Registro B.

3. Suma el dato 12_{10} (0Ch) al contenido del acumulador, y a ese valor Réstale el contenido del registro C.

4. Resta el contenido de la posición de memoria apuntada por los registros HL al acumulador y a ese valor Súmale el dato 0Ah.

5. Decrementa en 1 el contenido del Registro B e incrementa en 1 el contenido del Registro C.

6. Suma 1 al contenido de la posición de memoria apuntada por los registros HL

7. Incrementa en una unidad el contenido del registro doble HL (para que apunten a la siguiente dirección de memoria). (Notar la diferencia con el anterior).

8. Suma el contenido del Registro B más la llevada de la operación anterior al acumulador y luego incrementa en 1 el contenido del registro B.

9. Suma 9_{10} (09h) más la llevada de la operación anterior al Acumulador y luego réstale 1 el contenido del registro C

10. Resta el contenido del Registro B más la llevada de la operación anterior al acumulador y luego incrementa en 1 el contenido del registro C

11. Resta 2_{10} (02h) más la llevada de la operación anterior al Acumulador y luego réstale 1 el contenido del registro C

12. Suma el contenido de la posición de memoria apuntada por los registros HL más la llevada de la operación anterior al acumulador.

13. Resta el contenido de la posición de memoria apuntada por los registros HL más la llevada de la operación anterior al acumulador.

14. Decrementa en una unidad el contenido del registro doble HL (para que apunten a la anterior dirección de memoria). Decrementa ahora el contenido del registro L
¿Cuál es la diferencia?

Instrucciones lógicas

1. Ejecuta la instrucción lógica AND entre el registro B y el acumulador, dejando el resultado en el registro B. (Notar que al quedar el resultado en el acumulador, son necesarias dos instrucciones).
2. Ejecuta la instrucción lógica AND entre el contenido de la posición de memoria 9001h y el acumulador.
3. Queremos saber si el MSB (Most Significant Bit, el bit más significativo, el de mayor peso) del dato guardado en el acumulador es '1' o '0'. Para ello aplicamos una máscara al acumulador para poner a '0' todos los bits menos ese. ¿Cuál es la instrucción lógica que tenemos que ejecutar (indicando el valor)?
4. Ejecuta la instrucción lógica OR entre el registro C y el acumulador, dejando el resultado en el registro C.
5. Ejecuta la instrucción lógica OR entre el contenido de la posición de memoria 900Ah y el acumulador.
6. Queremos poner los bits de peso 0, 1 y 5 del acumulador a '1' sin modificar el resto, para ello aplicamos una máscara al acumulador ¿Cuál es la instrucción lógica que tenemos que ejecutar (indicando el valor)?
7. Ejecuta la instrucción lógica XOR entre el registro D y el acumulador, dejando el resultado en el registro D.
8. Ejecuta la instrucción lógica XOR entre el contenido de la posición de memoria 9000h y el acumulador.
9. Queremos obtener el valor negado del dato contenido en el acumulador, para ello le aplicamos una máscara. ¿Cuál es la instrucción lógica que tenemos que ejecutar (indicando el valor)?
10. Queremos comparar el contenido del registro B y del Acumulador, para saber si $A < B$. ¿Cuál es la instrucción lógica que tenemos que ejecutar?, ¿Dónde se guarda el

resultado?, ¿Cómo sabemos si A<B? (Pista: atención al registro F, registro de los flags).

Instrucciones de rotación

1. Carga el valor 88h en el acumulador y a continuación rota los bits una posición a la derecha (las instrucciones de rotación se ejecutan siempre sobre el dato en el acumulador) ¿cuál es el resultado? ¿Por qué? ¿Y si estuviéramos trabajando en base 10?

2. Carga el número hexadecimal 11h en el acumulador y róvalo una posición a la izquierda.

3. ¿Cuál es el resultado de ejecutar los programas a) y b) ? Razone la respuesta

a)	mvi A, 81	b)	mvi A, 81
	ral		ral
	ral		rlc

4. ¿Cuál es el resultado de ejecutar los programas A a) y b) ? Razone la respuesta

a)	mvi A, 81	b)	mvi A, 81
	rlc		rlc
	rlc		ral

5. Realiza el complemento a 1 del dato guardado en el registro B, dejando el resultado en el registro B.

6. Realiza el complemento a 2 del dato guardado en el registro C, dejando el resultado en el registro C (no se puede realizar con una sola instrucción).

Instrucciones de salto

1. En el siguiente ejemplo se decrementa el contenido del acumulador, si el resultado de la operación es cero salta a la etiqueta "cero" donde mueve 01 al registro B; si no, salta a la etiqueta "no-cero" y mueve 02 al registro B.

```
dcr a ;decrementa el contenido de A
jz cero ; si el resultado es 0, salta a "cero"
jnz no_cero ; si no, a "no_cero"
;en este caso sería lo mismo jmp no_cero
cero:
mvi B,01
no_cero:
mvi B,02
```

Pon el número 02 en el acumulador y ejecuta el código paso a paso. No se cumple la primera condición, por lo que se cumple la segunda y terminamos con el valor 02 en el registro B. Ahora pon el número 01 en el acumulador y ejecútalo paso a paso ¿qué sucede?

La ejecución es secuencial por lo que si se cumple la primera opción, se ejecuta el código reservado para el caso cero y también el reservado para el caso no cero. Si se desea que se ejecute una opción o la otra podemos poner un salto incondicional al terminar el código reservado a la primera opción.

```
dcr a ;decrementa el contenido de A
jz  cero ; si el resultado es 0, salta a "cero"
jnz no_cero ; si no, a "no_cero"
;en este caso sería lo mismo jmp no_cero
cero:
mvi B,01
no_cero:
jmp fin
mvi B,02
fin:
```

2. Si el resultado de la operación anterior no es cero salta a la etiqueta "no-cero" si no, continúa la ejecución normal del programa

3. Si el resultado de la operación anterior tiene llevada, salta a etiqueta "llevada", si no, salta a la etiqueta "no llevada"

4. Si hay error en un dato enviado mediante un código de paridad par, salta a la subrutina "error" (si se ha elegido paridad par, el número de '1's total ha de ser...).

5. Si el resultado de la operación anterior ha sido positivo, salta a etiqueta "positivo"

6. Si el resultado de la operación anterior ha sido negativo, salta a etiqueta "negativo"

7. Si el dato de la posición de memoria 9000h tiene un número impar de '1's, introduce el número 01h en el registro B, en caso contrario el número 02h

8. Ejecuta los siguiente programas paso a paso ¿cuál es la diferencia? En un caso se realiza un salto, la nueva dirección se carga en el PC y la ejecución sigue desde ahí; sin embargo en el caso de la llamada a subrutina, el valor del PC en el momento de la llamada (el valor de retorno) se guarda en la pila (notar cómo se decrementa el SP), a continuación la nueva dirección se carga en el PC, y tras la ejecución del código de la subrutina se recupera la ejecución desde el punto en el que se llamó (la instrucción de retorno siempre aparece al final del código de la subrutina haciendo que el último elemento de la pila, el TOS, se cargue al PC, actualizando nuevamente el valor del SP).

<pre> jmp salto mov a,b mov b,a jmp fin salto: mvi c,01 fin: </pre>	<pre> call salto mov a,b mov b,a jmp fin salto: mvi c,01 ret fin: </pre>
--	--

9. Llama a la subrutina "subrutina", en ella sumar 05h al valor del acumulador y si es positivo regresa a la rutina principal (Notar que tanto los saltos, como las llamadas a subrutina, como los retornos de subrutina pueden ser condicionados o no).

Llamadas y E/S

El siguiente código recoge dos números introducidos uno por el puerto 1 y el otro por el puerto 2; los coloca en los registros C y D, y llama a la subrutina "sumarBCD". ¿Qué hace dicha subrutina? Escribe el programa y ejecútalo paso a paso para comprobar el funcionamiento. Rellena la tabla de abajo identificando cada instrucción (ten en cuenta que no todas ocupan lo mismo). ¿Cuánto ocupan las etiquetas? ¿Cómo se sustituyen en código máquina?

mnemónico	Dirección	Instrucción hex (1, 2 o 3 bytes)
in 01		
mov c,a		
in 02		
mov d,a		

<code>mvi b,00</code>		
<code>call sumarBCD</code>		
<code>mov a,b</code>		
<code>out 03</code>		
<code>jmp fin</code>		
<code>sumarBCD:</code>		
<code>mov a,c</code>		
<code>add d</code>		
<code>mov b,a</code>		
<code>ret</code>		
<code>fin:</code>		
<code>nop</code>		

Completos

Escribe y comprueba los códigos para los siguientes programas. Ahora vas a tener que combinar todas las instrucciones vistas.

- 1- Lee el contenido de la posición 9000, y si es más pequeño que 07h, guarda su valor en la posición 9010.
- 2- Lee el contenido de la posición 9000h, y si es mayor o igual que 07h, guarda 0Fh en la posición 9010h.
- 3- Escribe y comprueba el código para calcular con el 8085 un factorial.
- 4- En la posición 9000h está el primer elemento de un array. Sabiendo que su longitud es de 10 elementos, escribe un programa que guarda en la posición 9100h la suma de todos sus elementos.
- 5- En la posición 9000h está el primer elemento de un array. Sabiendo que su longitud es de 10 elementos, escribe un programa que ordene los elementos del array de menor a mayor.
- 6- En la posición 9000h está el primer elemento de un array. Sabemos que todos los elementos son positivos. De los 10 elementos que lo componen, busca el valor

más pequeño y guarda su posición relativa (la que tiene dentro del array) en la posición de memoria siguiente a la del final del array.

- 7- Escribe un programa que lea indefinidamente el dato de la posición 9001h y si es igual a tres meta dos retardos sin operación para después poner el valor 01h en el puerto 3.
- 8- Un sensor de nivel pone el bit de peso 2 del byte en la posición 9001h a '1' cuando el agua llega a un determinado nivel. Cuando esto ocurre nuestro programa debe llamar a la subrutina de atención ("atención").

Programación dirigida por eventos

Programación secuencial (Batch)

El programador define el flujo del programa así como los momentos en los que se puede intervenir externamente.

La ejecución es lineal y consiste en leer parámetros de entrada, realizar unas operaciones y obtener unos resultados.

Programación dirigida por eventos

La estructura y el flujo de los programas quedan definidos por acciones externas. El programador define los eventos que gestionará el programa así como la secuencia que se ejecutará como respuesta a cada uno de ellos. Finalmente es el usuario el que, a través de las acciones que realiza, dirige el flujo del programa.

Al comienzo de la ejecución se inicializan los parámetros que el programa utilizará, para seguidamente quedar bloqueado en espera de un evento. El programa principal es por lo tanto un bucle que comprueba indefinidamente si se ha realizado alguno de los eventos que gestiona para, en caso afirmativo, pasar a ejecutar el código correspondiente a dicho evento. Este tipo de programación es la base de la interfaz de usuario.

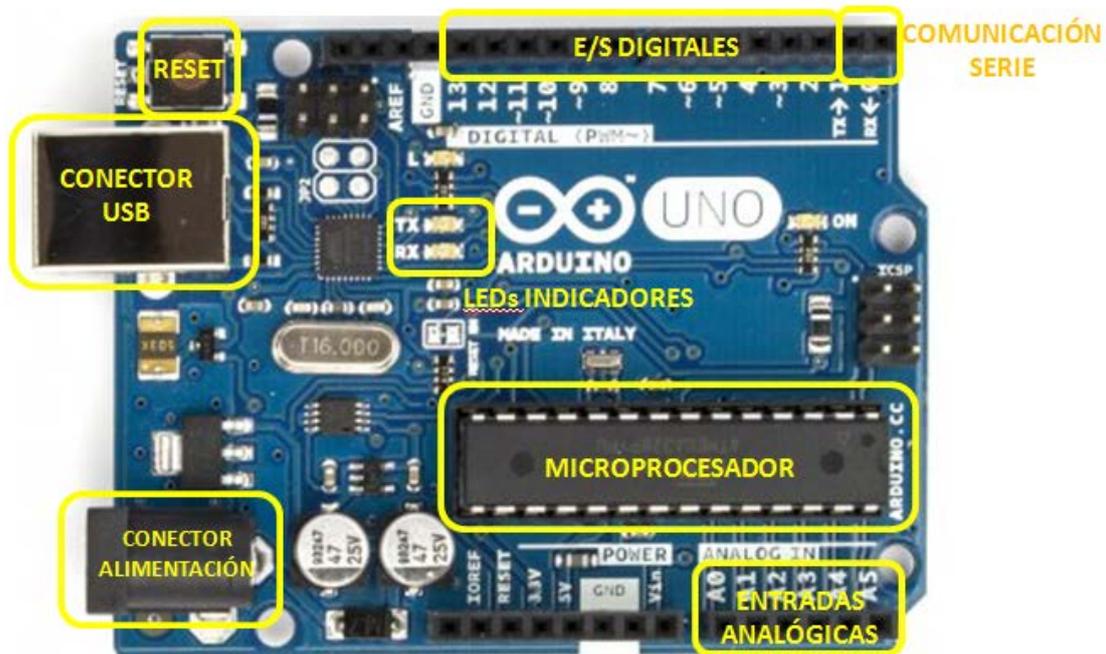
Nosotros utilizaremos la plataforma *Arduino* [1] para realizar pequeños programas dirigidos por eventos. *Arduino* está formado por software y una plataforma hardware libre, un lenguaje de programación y un IDE (entorno de desarrollo integrado). El lenguaje de programación está basado en C, por lo que es necesario (y será suficiente) conocer C, Basic o JAVA para entender los ejemplos. De todas formas el objetivo de estas sesiones es entender el concepto y ver el potencial de la programación dirigida por eventos, no aprender un lenguaje de programación. De manera que tras ver unos conceptos básicos probaremos varios ejemplos que nos darán las claves necesarias para realizar nuestro propio programa: la gestión de las luces de un semáforo (aplicación del ejemplo visto en teoría, T2).

NOTA: para poder realizar estos ejercicios es necesario una tarjeta arduino, un pulsador, un micrófono, unos LEDs, una protoboard y unos cables (además del ordenador).

[1] <http://www.arduino.cc/>

CONCEPTOS BÁSICOS

1. Tarjeta Arduino



2. Lenguaje de programación de Arduino

El cuerpo del programa tiene tres partes:

1. Estructura. *Definición de variables y constantes que se utilizarán*
2. Valores.

```
void setup () {  
  Entre estas llaves se inicializan lo definido en el punto anterior. }  
}
```
3. Funciones.

```
void loop () {  
  El programa queda bloqueado en este punto en un bucle a la espera de eventos. }  
}
```

El lenguaje de Arduino se basa en C/C++, aquí no veremos más que algunos conceptos básicos, más información en arduino.cc/en/Reference/HomePage.

Sintaxis

; al final de la línea
// comentarios de una línea
/ comentarios más largos */*

Estructuras de control

if
if...else
for
while
do...while
switch case
...

Operadores aritméticos

= *asignación* + - * / % *resto*

Operadores compuestos

++ *incremento*
-- *decremento*
...

Operadores Booleanos

&& AND || OR ! NOT

Operadores a nivel de bit

& *and* | *or* ^ *xor* ~ *not* << *shift left* >> *shift right*

Operadores de comparación

= = != < > <= >=

Variables y constantes

HIGH/LOW *para una variable binaria indica que está activada/desactivada*
INPUT/OUTPUT *inicializa un pin como de entrada/salida*
true/false *ojo! En el caso de un int false=0, cualquier otro valor= true*

Tipos de datos

int *número entero*
char *un carácter*
boolean *variable booleana (puede tomar los valores true/false)*
float *número en coma flotante*
...

E/S digitales

pinMode() *define un pin como de entrada o salida*
digitalWrite() *para escribir un pin (ponerlo a nivel alto o bajo)*
digitalRead() *para leer un valor de un pin*

E/S analógicas

analogRead()
analogWrite()

E/S para un altavoz

tone (salida, frecuencia, duración)

noTone *de no definirse la duración en la función tone, debe detenerse con noTone*

Control de tiempo

delay() *detiene el programa por el tiempo especificado (ms)*

delayMicroseconds()

Hay muchas más funciones además de las indicadas; tenéis toda la información disponible en la página web de *Arduino*. Para nosotros de momento son suficientes, ahora las veremos aplicadas en ejemplos.

- a) Temporización y salida digital - *Parpadeo del LED rojo*
- b) Entrada digital (evento=pulsador) y sentencia *if* - *Encendido/apagado de un LED en función de los eventos en los pulsadores*
- c) Sonido y sentencia *for* - *Sonidos de diferentes frecuencias*

a) PARPADEO DEL LED ROJO

```
//Ponemos un LED rojo conectado al pin 13
int ledR = 13; // le llamamos ledR

// rutina de inicialización
void setup() {
  pinMode(ledR, OUTPUT); //definimos este pin como pin de salida
}

// bucle
void loop() {
  digitalWrite(ledR, HIGH); //encendemos el Led rojo
  delay(1000); //esperamos un segundo
  digitalWrite(ledR, LOW); // apagamos el LED
  delay(1000); // esperamos un segundo
}
```

b.1) ENCENDER UN LED AL PULSAR EL PULSADOR

*/*Además del LED rojo (pin 13) conectamos un pulsador al pin 12, con el que encenderemos el LED*/*

```
int pulsador = 12;
int ledR = 13;
// El estado del pulsador no es constante, es una variable
int estadoPulsador = 0; // lo inicializamos a cero

void setup() {
  pinMode(ledR, OUTPUT); // el Led es una salida
  pinMode(pulsador, INPUT); // el pulsador una entrada
}
```

```

void loop(){
  // leemos el estado del pulsador (a la variable)
  estadoPulsador = digitalRead(pulsador);
  // si está pulsado...
  if (estadoPulsador == HIGH) {
    digitalWrite(ledR, HIGH); // ... Led encendido
  }
  // si no...
  else {
    digitalWrite(ledR, LOW); // ... apagado
  }
}

```

b.2) ENCENDER Y APAGAR UN LED CON UN PULSADOR

/ Podríamos añadir otro pulsador para apagar el LED, pero si queremos utilizar el pulsador a modo de interruptor no tenemos que detectar el estado en sí, sino el cambio de estado */*

```

// pulsador conectado al pin 12 y LED conectado al pin 13
int ledR = 13;
int pulsador = 12;

```

```

// variables para detectar y contar el cambio de estado
int contadorPulsador = 0;
int estadoPulsador = 0;
int estadoPrevioPulsador = 0;

```

```

void setup() {
  pinMode(ledR, OUTPUT); // El Led es una salida
  pinMode(pulsador, INPUT); //el pulsador una entrada
}

```

```

void loop() {
  estadoPulsador = digitalRead(pulsador); // leemos el estado del pulsador
  if (estadoPulsador != estadoPrevioPulsador) { //si es diferente al anterior
    if (estadoPulsador == HIGH) {
      // detectamos una transición ascendente
      contadorPulsador++;
    }
  }
  //el código se recorrerá varias veces mientras pulsamos, el estado actual será el
  //estado anterior en la siguiente vuelta*/
  estadoPrevioPulsador = estadoPulsador;
  // la función módulo devuelve el resto
  // si es una cantidad par, apaga el Led

```

```

if (contadorPulsador % 2 == 0) {
    digitalWrite(ledR, LOW);
}
else {
    digitalWrite(ledR, HIGH); //si no, encenderlo
}
}

```

c) SONIDOS DE DIFERENTES FRECUENCIAS

/* Ahora usaremos un pulsador conectado al pin 12 y un micrófono conectado al pin 8. Al pulsar el pulsador escucharemos sonidos cada vez más agudos, seguidos pero espaciados */

```

int pulsador = 12;
int estadoPulsador=0;
int sonido = 8;
int N=1;

```

```

void setup() {
    pinMode(pulsador, INPUT);
    pinMode(sonido, OUTPUT);
}

```

```

void loop(){
    estadoPulsador = digitalRead(pulsador);
    // sakatuta badago...
    if (estadoPulsador == HIGH) {
        for (N; N<=5; N++) {
            tone (sonido, N*2000, 2000);    // sonido de N*2000Hz durante 2 segundos
            delay(2000);                    //retardo de 2 segundos, si no no se oye el siguiente
        }
        N=1;
    }
}

```

/*Atención con la variable utilizada en el for: o se reinicializa al salir del for, o se define dentro del mismo; si no no volverá a entrar */

PASO DE PEATONES

Ahora que ya sabemos cómo detectar si se ha activado un pulsador, encender y apagar unos LEDs y mantenerlos un tiempo concreto en un estado; implementemos el ejemplo del semáforo mediante programación dirigida por eventos. Utilizaremos un pulsador (para simular el botón de solicitud) y un LED rojo, uno amarillo y uno verde (para simular un semáforo de peatones de tres colores).

a) Un semáforo está en rojo hasta que un peatón pulsa el botón de solicitud, en ese momento pasa a naranja, y tras 5 s a verde, transcurridos 10 s (los peatones disponen de 10 s para cruzar) vuelve a rojo.

b) Un semáforo está en rojo hasta que un peatón pulsa el botón de solicitud, en ese momento pasa a naranja, y tras 5 s a verde; pero ahora aunque los peatones disponen de 10 s para cruzar, ha de permanecer 5 s en verde fijo y otros cinco parpadeando, mientras que se escucha un "beep". Transcurrido este tiempo volverá a rojo hasta que otro peatón solicite cruzar.

PROYECTO

Si has llegado hasta aquí es que te has hecho con una placa con microprocesador, así que ¿por qué no aprovechar? Internet está lleno de ejemplos en los que inspirarse, pero si no se te ocurre nada, aquí va una propuesta: un control de temperatura. Necesitarás (además del Arduino y el ordenador):

- Un LM35 → sensor de temperatura que devuelve 10mV/C°
- Un potenciómetro rotacional de un ciclo (10 kΩ)
- Un resistor (de 220 Ω - 1 kΩ)
- Un diodo LED
- Una pequeña placa de conexiones y cables
- Un pulsador

Diseña un control de temperatura haciendo que el usuario pueda seleccionar variando la posición del potenciómetro una temperatura entre 15 y 30 C°. Para ello deberás conectar el potenciómetro a una entrada **analógica** y **mapear** (función *map*) el valor leído para limitarlo al rango especificado. Por otro lado, deberás conectar el sensor de temperatura a otra entrada **analógica** y convertir el valor que da el microprocesador (un valor entre 0 y 1023) a un valor de tensión primero (entre 0 y 5V) y a una temperatura después.

El programa debe adquirir y cotejar estos datos continuamente: en caso de que la temperatura ambiente sea menor que la seleccionada, deberá encender el LED (que en este caso simula la calefacción); en caso contrario deberá apagarlo.

Además, cada vez que el usuario pulse el pulsador, deberá visualizarse utilizando el monitor serial (el icono en forma de lupa en el IDE de Arduino) y las instrucciones `Serial.begin`, `Serial.Print`, `Serial.PrintIn`,... (ver la página oficial de Arduino para información y ejemplo) la temperatura ambiente, la seleccionada y si la calefacción está encendida o apagada.