2016

ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORAS

Series de Trabajos Prácticos

Guía N° 2: Sistemas de Numeración: Sistemas Enteros y Punto Fijo

- **Tabla de Valores (Binario$_2$, Octal$_8$, Decimal$_{10}$, Hexadecimal$_{16}$)**

Binario	Octal	Decimal	Hexadecimal
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

1. Convertir $(1962)_{10}$ en su equivalente hexadecimal, binario y octal.

   - **$(1962)_{10}$ a Hexadecimal$_{16}$: 7AA**

$$\begin{array}{r|l} 1962 & 16 \\ 10 & 122 \end{array} \qquad \begin{array}{r|l} 122 & 16 \\ 10 & 7 \end{array} \qquad \begin{array}{c} 7 \\ 7 \end{array} \quad \begin{array}{c} 10 \\ A \end{array} \quad \begin{array}{c} 10 \\ A \end{array}$$

   $(1962)_{10}$ = **$7AA_{16}$**

   - **$(1962)_{10}$ a Binario$_2$:** como ya tenemos el valor en hexadecimal, hacemos el pasaje directo de hexadecimal a binario de tal forma que evitamos realizar una extensa división por 2. Transformamos el valor de cada digito hexadecimal a su correspondiente en binario, luego unimos los resultados y obtenemos su equivalente en binario.

$$\begin{array}{c} 7 \\ 0111 \end{array} \quad \begin{array}{c} A \\ 1010 \end{array} \quad \begin{array}{c} A \\ 1010 \end{array}$$

   $(1962)_2$ = **011110101010_2**

   - **$(1962)_{10}$ a Octal$_8$:** como ya tenemos el valor en binario, hacemos el pasaje directo de binario a octal de tal forma que evitamos realizar una extensa división por 8. Tomamos de a tres dígitos binarios y obtenemos su correspondiente en sistema octal

$$\begin{array}{c} 011110101010 \\ \hline 011 \mid 110 \mid 101 \mid 010 \\ \mathbf{3 \quad 6 \quad 5 \quad 2} \end{array}$$

   $(1962)_{10}$ = **3652_8**

2. Representar en Ca1 y Ca2 los números 512 y -29 tomando como base una computadora con palabra de 16 bits.

$$\begin{array}{c} 512 \mid 256 \mid 128 \mid 64 \mid 32 \mid 16 \mid 8 \mid 4 \mid 2 \mid 1 \\ \hline Valor\ de\ las\ posiciones\ en\ binario \end{array}$$

   - **Complemento a 1 (C-1): 512_{10} = 0000001000000000_2**

Para pasar un numero decimal negativo a binario negativo en Complemento a 1 (C-1), primero debemos obtener el binario del número decimal positivo y luego complementarlo para obtener su negativo.

- **Complemento a 1 (C-1): 29_{10} = 0000000000011101_2**
- **Complemento a 1 (C-1): -29_{10} = 1111111111100010_2**

- **Complemento a 2 (C-2): 512_{10} = 0000001000000000_2**

Para pasar un numero decimal negativo a binario negativo en Complemento a 2 (C-2), primero debemos obtener el binario del número decimal positivo, complementarlo y luego sumarle 1, despreciando el ultimo acarreo si existiese.

- **Complemento a 2 (C-2): 29_{10} =** 0000000000011101_2
- **Complemento a 2 (C-2): 29_{10} =** 1111111111100010_2 **complemento**
  1 **sumamos 1**
- **Complemento a 2 (C-2): -29_{10} = 1111111111100011_2**

3. Averiguar que números decimales expresa los binarios 11010011 y 00100111 en **Ca1** en una palabra de 8 bits.

1 1 0 1 0 0 1 1 (Ca1 negativo)
0 0 1 0 1 1 0 0 pasamos a positivo (complementamos)
32+0+8+4+0+0 obtenemos el valor correspondiente a cada posición
-44 obtenemos el numero 44 al cual lo convertimos en negativo.

0 0 1 0 0 1 1 1 (Ca1 positivo)
32+0+0+4+2+1 obtenemos el valor correspondiente a cada posición
39

4. ¿Qué números decimales representan los siguientes números representados en Ca2 en una palabra de 7 bits?

1 1 0 1 0 1 1 (Ca2 negativo)
\- 1 restamos uno
1 1 0 1 0 1 0 obtenemos el complemento del número positivo
0 0 1 0 1 0 1 complementamos
16+0+4+0+1 obtenemos el valor correspondiente a cada posición
-21 obtenemos el número 21 al cual lo convertimos en negativo.

0 1 0 1 1 0 1 (Ca2 positivo)
32+0+8+4+0+1 obtenemos el valor correspondiente a cada posición
45 obtenemos el número 45 positivo

5. Interpretar el valor 10110110 que se encuentra en exceso en palabra de 8 bits, $2^{n-1} = 2^{8-1} = 128$.

1 0 1 1 0 1 1 0
128+0+32+16+0+4+2+0 obtenemos el valor correspondiente a cada posición
182 obtenemos el numero 182
\- 128 restamos 128 ($2^{n-1} = 2^{8-1}$)
54

- Para los números positivos, en caso de 2^{n-1} Exceso, la representación es la misma que para Complemento a 1 C-1 con la salvedad que el digito más significativo es 1. **Ej.:** 1 = 10000001 = 129 = 129 - 128 = 1

- Para los números negativos, en caso de 2^{n-1} Exceso, se debe realizar la operación del Numero negativo sumado a 128, por ejemplo, para representar el numero negativo -100, se debe realizar la siguiente operación: -100 + 128 = 28 = 00011100. Ejemplo para obtener el numero decimal que se encuentra en binario en exceso 2^{n-1}: -100 = 00011100 = 28 = 28 - 128 = -100. Nota: el bit más significativo para los números negativos es 0.

6. Escriba el -123 en exceso con 8 bits: **-123 + 128 = 5 = 00000101**

7. Realice los siguientes cálculos:

   - -28 + 122 = 94 representándolos en Ca1 en palabras de 8 bits

     28 = 00011100, -28 = 11100011, 122 = 01111010

     ```
       11     1
       11100011  -28
     +  0     1
       01111010  122
      101011101
      ↳ 1  Carry (Acarreo)
      01011110   94
     ```

   - 6 + 13 = 19 representándolos en Ca2 en palabras de 8 bits

     6 = 00000110, 13 = 00001101

     ```
         11
       00000110  6
     +  0  11
       00001101  13
       00010011  19
     ```

   - -18 + 118 = 100 representándolos en Ca1 en palabras de 8 bits

     18 = 00010010, -18 = 11101101, 118 = 01110110

     ```
       11 111
       11101101   -18
     +  00 010
       01110110   118
          11
      101100011
      ↳ 1  Carry (Acarreo)
      01100100   100
     ```

   - 5 + 3 = 8 en representación en binario sin signo (BSS) en palabra de 3 bits

     5 = 101, 3=011

     ```
       11
       101  5
     +  01
      1011  3
      1000  8  > Una palabra de 3 bits no puede contener al número 8, se necesitaría 4 bits.
      ↳ Carry y OverFlow (Acarreo y Sobrecarga)
     ```

   - 92 + 53 = 145 representándolos en Ca1 en palabras de 8 bits

     92 = 01011100, 53 = 00110101

     ```
       11111
       01011100    92
     +  10 100
       00110101    53
       10010001   145
     ```

8. Indicar cuál es la capacidad de representación, la resolución y el rango de un sistema binario con 7 bits para la parte entera y 4 para la parte fraccionaria.

   7bits 4bits = 11bits

   0000000 , 0000 = 00000000000

- **Capacidad de Representación**: Es la cantidad de tiras distintas que se pueden representar. Para un sistema restringido a 11 bits, sería $\mathbf{2^{11}}$ tiras, es decir, **2048**.

- **Resolución**: Es la mínima diferencia entre un número representable y el siguiente. Para un sistema restringido a 11 bits: **[0000000,0001]**

- **Rango:** El rango de un sistema está dado por el número mínimo y el número máximo representables. Para un sistema restringido a 11 bits: **[0, 2048]** : **[0000000,0000 ; 1111111,1111]**

Guía N° 2: Sistemas de Numeración: Punto Flotante

1. Convertir el número 9,375 * 10^{-2} a un formato de coma flotante según las especificaciones: 1 bit para el signo, 3 bits para el exponente, y 5 bits para la mantisa, normalizada (no oculta), ambos expresados en MS.

   - Obtenemos el número: 9,375 * 10^{-2} = 0.09375
   - Convertimos a binario

0.09375	x 2	0.1875	< 1 =	0
0.1875	x 2	0.375	< 1 =	0
0.375	x 2	0.75	< 1 =	0
0.75	x 2	1.5	>= 1	1
1.5	x 2	3	>= 1	1

   - 0.09375 * 10^0 = 0,0001 * 2^0 (obtenemos la igual en ambas bases, 10 y 2)
   - Normalizamos moviendo la coma tantos lugares sean necesarios hasta llegar al primer 1 (en este caso movemos la coma hacia la derecha dando resultado un numero negativo), movemos 3 lugares obteniendo como resultado: **0,1 * 2^{-3}, nuestro exponente será -3**
   - Como es un número positivo, el **Signo es 0,** el exponente debemos expresarlo como MS (modulo y signo), por tal motivo, el digito más significativo será el signo y el resto es el modulo, obteniendo **exponente *1*11**

Signo	Exponente	Mantisa
0	*1*11	00011

2. Representar el numero hexadecimal positivo BF,0D1A con la notación IEEE 754 de simple precisión (32 bits – Signo 1 bit, Exponente 8 bits, Mantisa 23 bits).

   - Obtenemos la representación de BF,01DA en binario

B	F	,	0	D	1	A	* 2^0
1011	1111	,	0000	1101	0001	1010	

   - Procedemos a normalizar moviendo la coma hasta la posición más significativa, para este caso movemos 7 lugares hacia la izquierda obteniendo el exponente positivo.

1	,	011	1111	0000	1101	0001	1010	* 2^7

   - **Exponente**: Obtenemos el Sesgo (**2^{k-1}-1**) 2^{8-1}-1=127 y sumamos el exponente 127 + 7 = 134. Procedemos a convertir el numero obtenido a binario: **10000110**
   - **Mantisa**: como el sistema es con bit implícito, solo estableceremos los bits desde la coma hacia la derecha: **01111110000110100011010**

Signo	Exponente	Mantisa
0	10000110	01111110000110100011010

3. Dados los siguientes números

   a. 1 0001 10000010
   b. 0 0111 11111000
   c. 1 0110 11111101
   d. 0 0010 11101011

Realice las operaciones aritméticas que se expresan a continuación. Verifique el resultado en decimal.

a) b + d (0 0111 11111000 + 0 0010 11101011)

124 + 2.671875 = **126.671875**

Tomamos el exponente de menor valor (0010) y lo igualamos al de mayor valor incrementando su valor de a un bit y desplazando la mantisa hacia la derecha.

Signo	Exponente	Mantisa
0	0010	11101011
0	0011	01110101
0	0100	00111010
0	0101	00011101
0	0110	00001110
0	0111	00000111

Una vez igualado procedemos a sumar las mantisas

Suma de mantisas
00000111 11111000
11111111

Resultado de la suma

Signo	Exponente	Mantisa
0	0111	11111111

Comprobación de suma y precisión

Signo	Exponente	Mantisa	Precisión
0	2^7	1111100,0 = 124	Pérdida de precisión en 0,171875
0	2^7	0000011,1 = 2,5	
		126,5	

b) c + a (1 0110 11111101 + 1 0001 10000010)

63,25 + 1,015625 = **64,265625**

Tomamos el exponente de menor valor (0001) y lo igualamos al de mayor valor incrementando su valor de a un bit y desplazando la mantisa hacia la derecha.

Signo	Exponente	Mantisa
0	0001	10000010
0	0010	01000001
0	0011	00100000
0	0100	00010000
0	0101	00001000
0	0110	00000100

Una vez igualado procedemos a sumar las mantisas

Suma de mantisas
00000100 11111101
1 00000001

Resultado de la suma

Signo	Exponente	Mantisa
0	0110	00000001

Comprobación de suma y precisión

Signo	Exponente	Mantisa	Precisión
0	2^6	111111,01 = 63,25	Pérdida de precisión en 0,515625
0	2^6	000000,01 = 0,5	
		63,75	

4. Dado el siguiente formato: 1 bit de signo, 4 bits de exponente en exceso 4-1 y 8 bits de mantisa con bit oculto, realice las siguientes sumas:

a) 0 0001 00111000 + 0 0111 10100010 (Resultados teniendo en cuenta el bit oculto)

1,21875 + 104,5 = 105,71875

Tomamos el exponente de menor valor (0001) y lo igualamos al de mayor valor incrementando su valor de a un bit y desplazando la mantisa hacia la derecha. Como se considera el bit oculto, agregamos dicho bit a la mantisa y desplazamos a la derecha

Signo	Exponente	Mantisa
0	0001	10011100
0	0010	01001110
0	0011	00100111
0	0100	00010011
0	0101	00001001
0	0110	00000100
0	0111	00000010

Una vez igualado procedemos a sumar las mantisas

Suma de mantisas
00000010 11010001
11010011

Resultado de la suma

Signo	Exponente	Mantisa
0	0111	11010011

Comprobación de suma y precisión

Signo	Exponente	Mantisa	Precisión
0	2^7	0000001,0 = 1	Pérdida de precisión en 0,21875
0	2^7	1101000,1 = 104,5	
		105,5	

b) 0 1001 00111000 + 0 0111 10100010

Tomamos el exponente de menor valor (0111) y lo igualamos al de mayor valor incrementando su valor de a un bit y desplazando la mantisa hacia la derecha. Como se considera el bit oculto, agregamos dicho bit a la mantisa y desplazamos a la derecha

Signo	Exponente	Mantisa
0	0111	11010001
0	1000	01101000
0	1001	00110100

Una vez igualado procedemos a sumar las mantisas

Suma de mantisas
1 1 00110100 **10011100**
11010000

Resultado de la suma

Signo	Exponente	Mantisa
0	1001	**11010000**

Guía N° 3: Primera Parte: Lógica Combinacional

1. Considere las siguientes tres parejas de sucesiones de bit: Determinar cómo se procesaría cada par de sucesiones por una compuerta OR y por una compuerta AND.

Ejercicio	Sucesiones	OR	AND
i)	110001 101101	111101	100001
ii)	10001111 00111100	10111111	00001100
iii)	101100111000 000111001101	101111111101	000100001000

2. ¿Cómo procesaría una compuerta NOT cada sucesión?

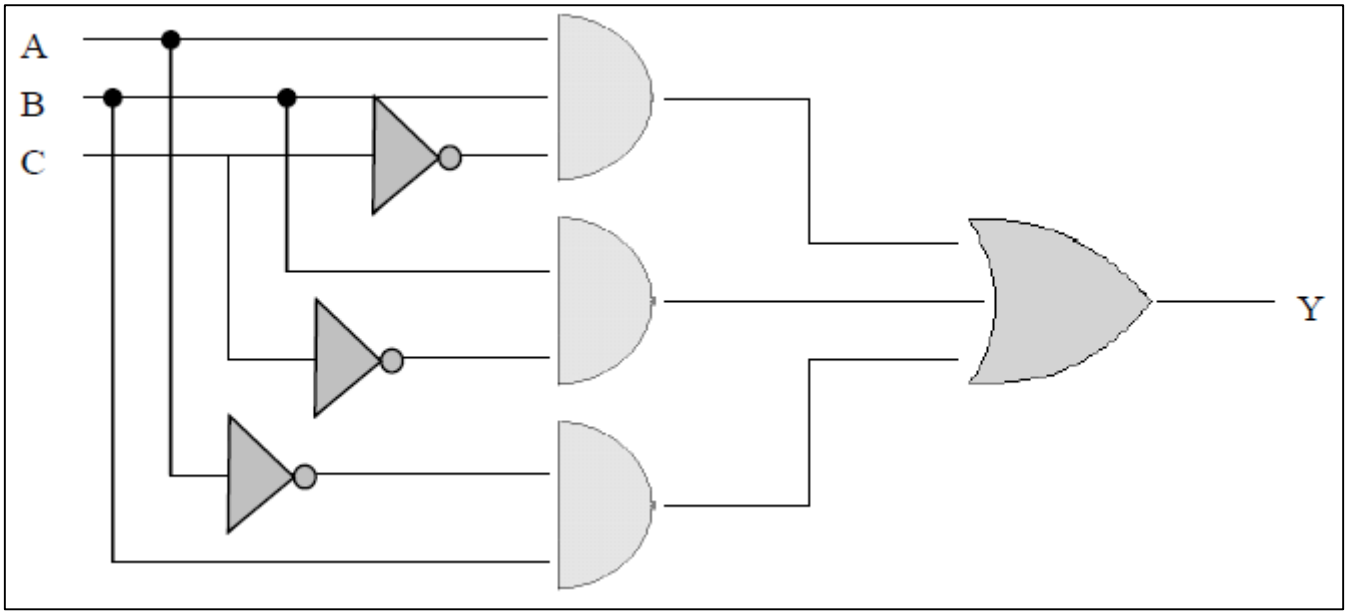
Ejercicio	Sucesiones	NOT
i)	110001	001110
ii)	10001111	01110000
iii)	101100111000	010011000111

3. Dados: A = 1100110110; B = 1110000111; C = 1010010110, obtener:

Calculo Auxiliar: A' = 0011001001

Ejercicio	Operaciones
A+B+C	1100110110 (A) 1110000111 (B) 1010010110 (C) **1110110111**
A*B*C	1100110110 (A) 1110000111 (B) 1010010110 (C) **1000000110**
C*(A'+B)	0011001001 (A') 1110000111 (B) 1111001111 1010010110 (C) **1010000110**
(B+C)'*A	1110000111 (B) 1010010110 (C) 1110010111 0001101000 ' 1100110110 (A) **0000100000**

4. Encuentre una expresión de Boole y la tabla de verdad para el circuito lógico de la siguiente figura:

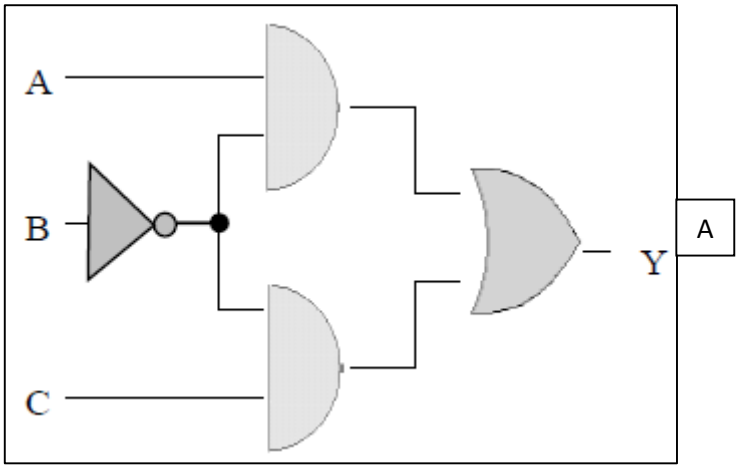


- **Expresión de Boole**: (A*B*C')+(B*C')+(A'*B)
  1 1 0 1 0 0 1

- **Tabla de Verdad**

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

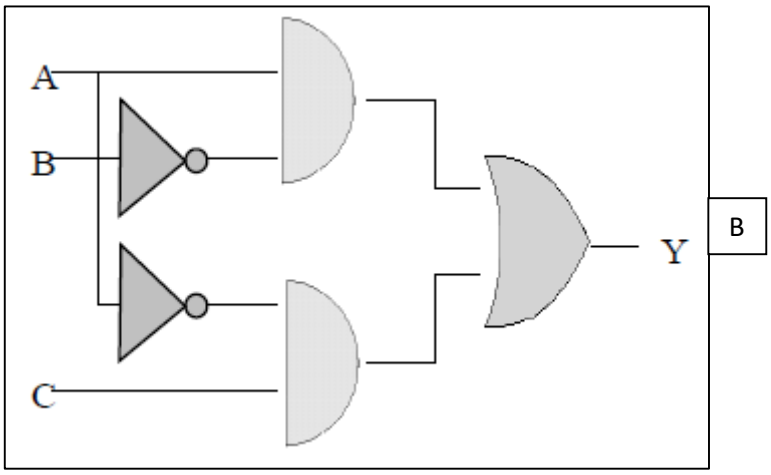
5. Considere el siguiente circuito lógico: De la salida Y como expresión de Boole con entradas A, B y C, encuentre la tabla de verdad del circuito.



- **Expresión de Boole**: (A*B')+(B'*C)
  1 0 0 1

- **Tabla de Verdad**

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



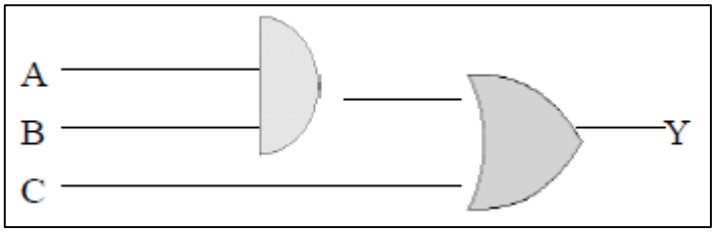
- **Expresión de Boole**: (A*B')+(A'*C)
  1 0 0 1

- **Tabla de Verdad**

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

6. Describir el comportamiento de la función F del siguiente circuito por medio de la lógica, presentarla en forma de suma de producto:

   Expresión Lógica: (A*B) + C (Suma de Productos)



7. Determine por medio de una tabla de verdad la validez del teorema de DeMorgan para tres variables: (ABC)' = A' + B' + C'

A	B	C	A'	B'	C'	A*B*C	(A*B*C)'	A'+B'+C'
0	0	0	1	1	1	0	1	1
0	0	1	1	1	0	0	1	1
0	1	0	1	0	1	0	1	1
0	1	1	1	0	0	0	1	1
1	0	0	0	1	1	0	1	1
1	0	1	0	1	0	0	1	1
1	1	0	0	0	1	0	1	1
1	1	1	0	0	0	1	0	0

8. Liste la tabla de verdad de una función XOR (impar) de tres variables: F= A ⊕ B ⊕ C

A	B	C	A ⊕ B ⊕ C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

9. Simplifique las siguientes expresiones usando álgebra booleana.

   a) A + AB (Aplicar Teorema de Absorción)
   Resultado final: **A**

   b) AB + AB' (Aplicar Propiedad Distributiva)
   A(B+B') (Aplicar Propiedad de los complementos)
   A(1) (Aplicar Propiedad de identidad)
   Resultado final: **A**

   c) A'BC + AC (Factor común C)
   C(A'B+A) (Aplicar Distributiva)
   C(A+A')(A+B) (Aplicar Propiedad de los complementos)
   C(1)(A+B)
   Resultado final: **C(A+B)**

   d) A'B + ABC' + ABC (Factor común en los dos últimos términos AB)
   A'B + AB(C'+C) (Aplicar Propiedad de los Complementos)
   A'B + AB (Factor común B)
   B(A'+A) (Aplicar Propiedad de los Complementos)
   B(1)
   Resultado final: **B**

   e) AB + A(CD + CD') (Factor común C)
   AB + A(C(D + D'))) (Aplicar Propiedad de los Complementos)
   AB + A(C(1)) (Factor común A)
   Resultado final: **A(B + C)**

   f) (BC'+A'D)(AB'+CD') X

10. Mediante una tabla de verdad demuestre que:
    a) (AB)' ≠ A'B'

A	B	A’	B’	A*B	(A*B)’	A’*B’
0	0	1	1	0	1	1
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	1	0	0	1	0	0

b) A’B’ + AB = (A ⊕ B)’

A	B	A’	B’	A*B	A’*B’	A ⊕ B	(A’*B’)+(A*B)	(A ⊕ B)’
0	0	1	1	0	1	0	1	1
0	1	1	0	0	0	1	0	0
1	0	0	1	0	0	1	0	0
1	1	0	0	1	0	0	1	1

c) A’B + AB’ = A ⊕ B (Además demostrar mediante algebra booleana que (A’B+AB’)’ = (A ⊕ B)’)

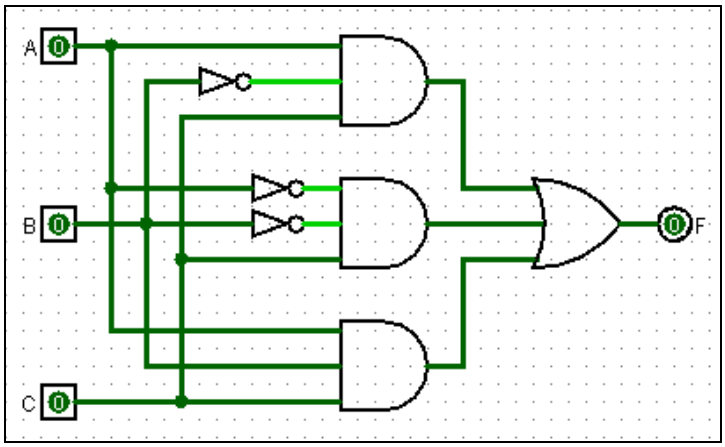
A	B	A’	B’	A’*B	A*B’	(A’*B)+(A*B’)	A ⊕ B
0	0	1	1	0	0	0	0
0	1	1	0	1	0	1	1
1	0	0	1	0	1	1	1
1	1	0	0	0	0	0	0

11. Dada la función booleana: F = AB’C + A’B’C + ABC
10 1 0 0 1 111

a) Liste la tabla de verdad para la función.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

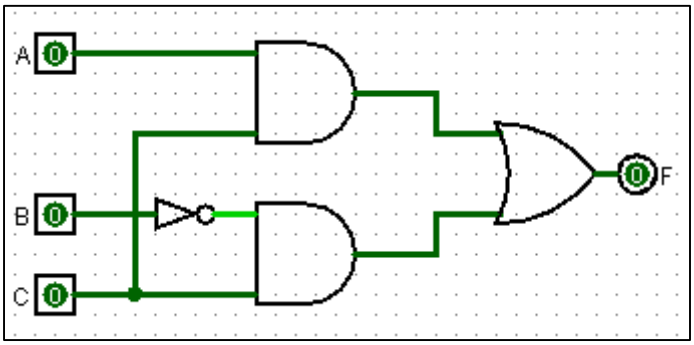
b) Dibuje un diagrama lógico por medio de la expresión booleana original.



c) Simplifique la expresión algebraica mediante el álgebra booleana.

AB’C + A’B’C + ABC (Factor común AC, primer y último término)
AC(B’ + B) + A’B’C (Aplicar Propiedad de los complementos)
AC + A’B’C (Factor común C)
C(A + A’B’) (Aplicar Distributiva)
C((A+A’)(A+B’)) (Aplicar Propiedad de los complementos)
C(A + B’) (Aplicar Distributiva)
Resultado final: **B’C + AC**

d) Dibuje el diagrama lógico de la expresión simplificada y compare el número total de compuertas con el diagrama original.



12. Simplifique las siguientes funciones booleanas mediante mapas de tres variables:

F(A ,B, C) = Σ (0, 1, 5, 7) | F(A, B, C) = Σ (1, 2, 3, 6, 7) | F(A, B, C) = Σ (3, 5, 6, 7) | F(A, B, C) = Σ (0, 2, 3, 4,6)

Posición	A	B	C	∑(0,1,5,7)	∑(1,2,3,6,7)	∑(3,4,6,7)	∑(0,2,3,4,6)
0	0	0	0	1	0	0	1
1	0	0	1	1	1	0	0
2	0	1	0	0	1	0	1
3	0	1	1	0	1	1	1
4	1	0	0	0	0	1	1
5	1	0	1	1	0	0	0
6	1	1	0	0	1	1	1
7	1	1	1	1	1	1	0

∑(0,1,5,7)				
C\AB	**00**	**01**	**11**	**10**
0	1	0	0	0
1	1	0	1	1
A’B’ + AC				

∑(1,2,3,6,7)				
C\AB	**00**	**01**	**11**	**10**
0	0	1	1	0
1	1	1	1	0
A’C + B				

∑(3,4,6,7)				
C\AB	**00**	**01**	**11**	**10**
0			1	1
1		1	1	
BC + AC’				

∑(0,2,3,4,6)				
C\AB	**00**	**01**	**11**	**10**
0	1	1	1	1
1		1		
C’ + A’B				

13. Simplifique las siguientes funciones booleanas mediante mapas de cuatro variables:

F(A, B, C, D) = Σ (4,6,7,15) | F(A, B, C, D) = Σ (3,7,11,13,14,15) | F(A, B, C, D) = Σ (0,1,2,4,5,7,11,15)

Posición	A	B	C	D	Σ (4,6,7,15)	Σ (3,7,11,13,14,15)	Σ (0,1,2,4,5,7,11,15)
0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1
2	0	0	1	0	0	0	1
3	0	0	1	1	0	1	0
4	0	1	0	0	1	0	1
5	0	1	0	1	0	0	1
6	0	1	1	0	1	0	0
7	0	1	1	1	1	1	1
8	1	0	0	0	0	0	0
9	1	0	0	1	0	0	0
10	1	0	1	0	0	0	0
11	1	0	1	1	0	1	1
12	1	1	0	0	0	0	0
13	1	1	0	1	0	1	0
14	1	1	1	0	0	1	0
15	1	1	1	1	1	1	1

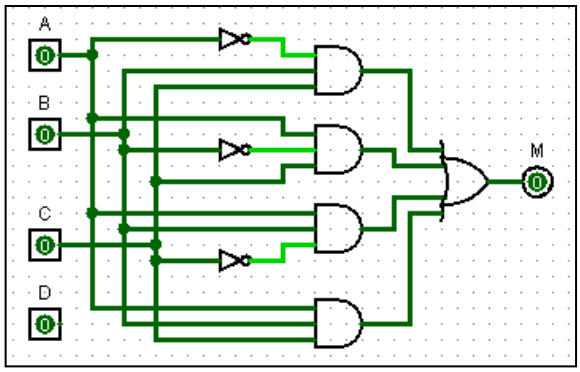
Σ (4,6,7,15)				
CD\AB	00	01	11	10
00		1		
01				
11		1	1	
10		1		
A'BD' + BCD				

Σ (3,7,11,13,14,15)				
CD\AB	00	01	11	10
00				
01			1	
11	1	1	1	1
10			1	
CD+ABD+ABC				

Σ (0,1,2,4,5,7,11,15)				
CD\AB	00	01	11	10
00	1	1		
01	1	1		
11		1	1	1
10	1			
A'B'D'+A'C'+A'BD+ACD				

14. Una función mayoría se genera en un circuito combinatorio cuando la salida vale uno si en las variables de entrada hay más unos que ceros; de otra forma la salida vale cero. Diseñe una función mayoría de tres entradas.

A	B	C	Mayoría
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

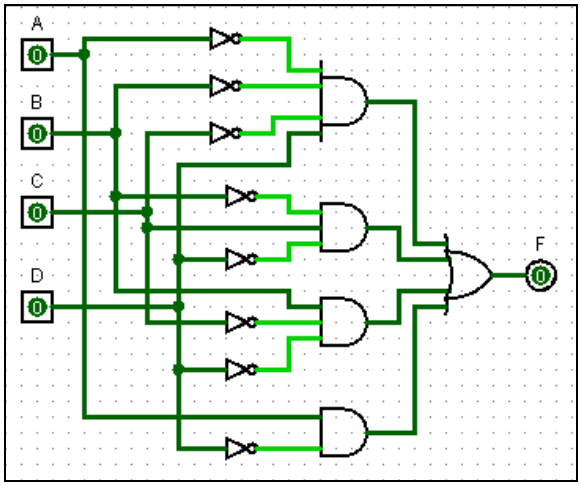


Expresión Lógica: A'BC + AB'C+ABC'+ABC

15. Las cuatro líneas que entran a un circuito lógico combinacional (x_3, x_2, x_1, x_0) llevan un dígito decimal codificado en binario (BCD), es decir, los equivalentes binarios de los números 0-9, siendo x3 el bit más significativo.

Las combinaciones de los valores correspondientes a los valores 10 –15 nunca aparecerán en las líneas de entrada. La salida Z del circuito deberá ser 1 si y solo si la combinación entrante es una potencia de 2. Obtenga un circuito mínimo de dos niveles y construya un diagrama lógico. (Potencias de 2: 1,2,4,8,16…)

$F(x_3, x_2, x_1, x_0)$				
$X_2X_3\backslash X_0X_1$	**00**	**01**	**11**	**10**
00	0	1	X	1
01	1	0	X	0
11	0	0	X	X
10	1	0	X	X
$X_0'X_1'X_2'X_3 + X_1X_2'X_3' + X_1'X_2X_3' + X_0X_3'$				



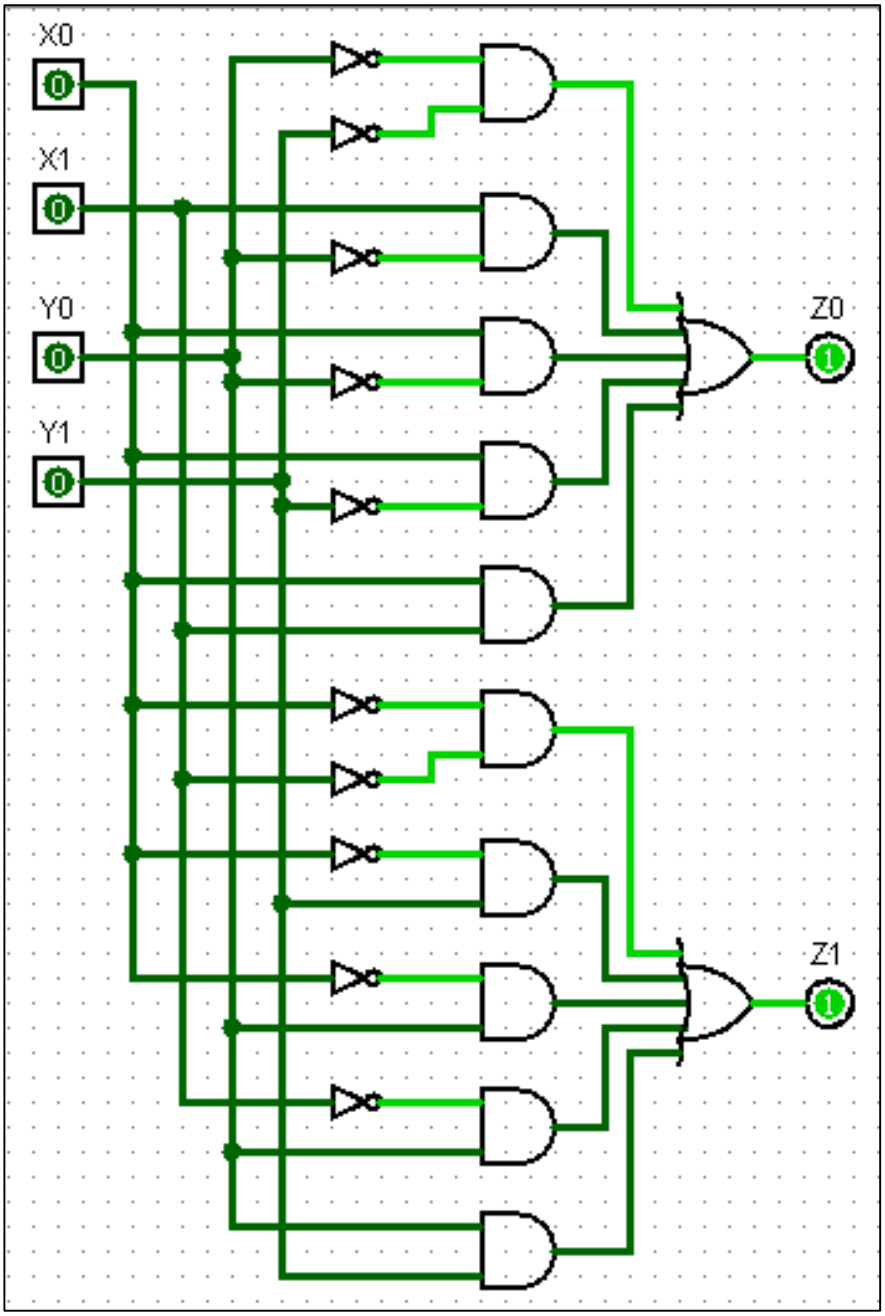
Posición	X_0	X_1	X_2	X_3	F
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	X
11	1	0	1	1	X
12	1	1	0	0	X
13	1	1	0	1	X
14	1	1	1	0	X
15	1	1	1	1	X

16. Un circuito recibe dos números binarios de dos bits: $X = X_1 X_0$ e $Y = Y_1 Y_0$. La salida de dos bits $Z = Z_1 Z_0$ debe ser igual a 11 si X = Y, 10 si X > Y y 01 si X < Y. Diseñe una realización mínima de suma de productos

Posición	X_0	X_1	Y_0	Y_1	Z_0	Z_1
0	0	0	0	0	1	1
1	0	0	0	1	0	1
2	0	0	1	0	0	1
3	0	0	1	1	0	1
4	0	1	0	0	1	0
5	0	1	0	1	1	1
6	0	1	1	0	0	1
7	0	1	1	1	0	1
8	1	0	0	0	1	0
9	1	0	0	1	1	0
10	1	0	1	0	1	1
11	1	0	1	1	0	1
12	1	1	0	0	1	0
13	1	1	0	1	1	0
14	1	1	1	0	1	0
15	1	1	1	1	1	1

Z_0				
$Y_0Y_1\backslash X_0X_1$	**00**	**01**	**11**	**10**
00	1	1	1	1
01		1	1	1
11			1	
10			1	1
$Y_0'Y_1' + X_1Y_0' + X_0Y_0' + X_0Y_1' + X_0X_1$				

Z_1				
$Y_0Y_1\backslash X_0X_1$	**00**	**01**	**11**	**10**
00	1			
01	1	1		
11	1	1	1	1
10	1	1		1
$X_0'X_1' + X_0'Y_1 + X_0'Y_0 + X_1'Y_0 + Y_0Y_1$				



17. Para la siguiente tabla de verdad encuentre la expresión lógica correspondiente:

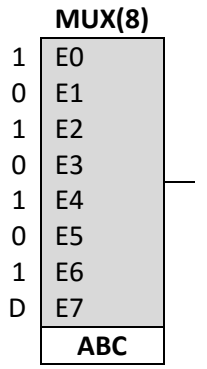
A	B	F	A'	A' + B
0	0	**1**	**1**	**1**
0	1	**1**	**1**	**1**
1	0	**0**	**0**	**0**
1	1	**1**	**0**	**1**
A' + B				

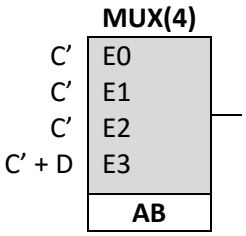
18. Implemente la función F(A, B, C, D) = $\sum$ (0, 1, 4, 5, 8, 9, 12,13, 15)

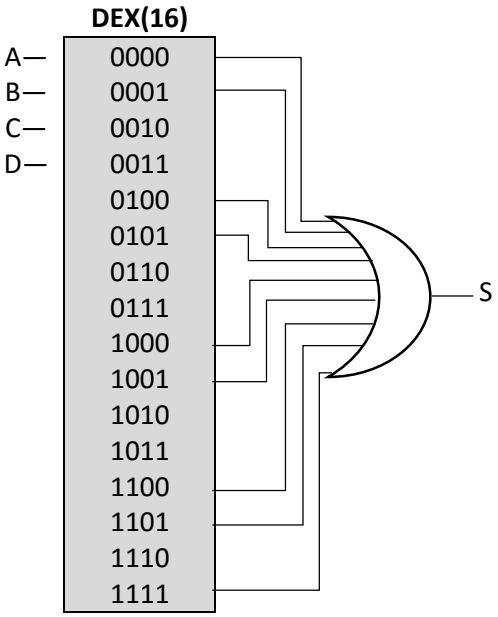
a) con un multiplexor de 8 canales.
b) con un multiplexor de 4 canales.
c) con un decodificador de 4 a 16.
d) con dos decodificadores de 3 a 8

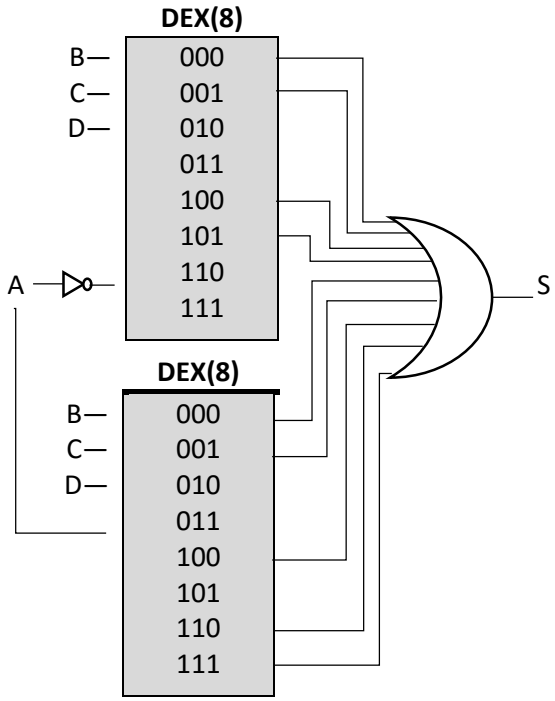
Obtenemos la tabla de verdad

Posición	A	B	C	D	F	MUX(8)	MUX(4)
0	0	0	0	0	1	1	C’
1	0	0	0	1	1		
2	0	0	1	0	0	0	
3	0	0	1	1	0		
4	0	1	0	0	1	1	C’
5	0	1	0	1	1		
6	0	1	1	0	0	0	
7	0	1	1	1	0		
8	1	0	0	0	1	1	C’
9	1	0	0	1	1		
10	1	0	1	0	0	0	
11	1	0	1	1	0		
12	1	1	0	0	1	1	C’ + D
13	1	1	0	1	1		
14	1	1	1	0	0	D	
15	1	1	1	1	1		









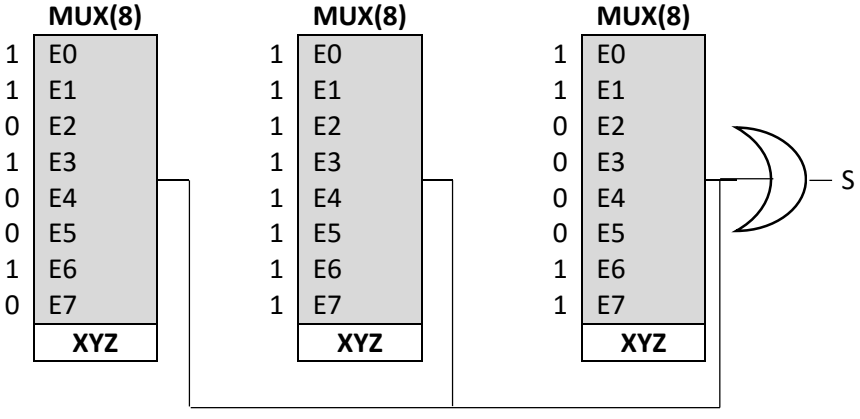
19. Un circuito combinacional implementa las siguientes tres funciones: Diseñe el circuito con un decodificador binario de 3 a 8 y compuertas externas.

F1 (x, y, z) = x’ y’ + x y z’

F2 (x, y, z) = x’ + y

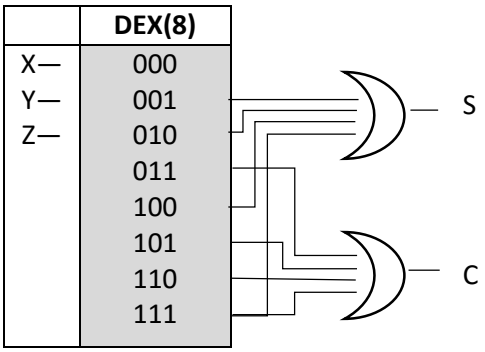
F3 (x, y, z) = x y + x’ y’

X	Y	Z	F1	F2	F3
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	0	1	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	0	1	1



20. Diseñe un sumador completo:
    a) Con un decodificador de 3 a 8

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



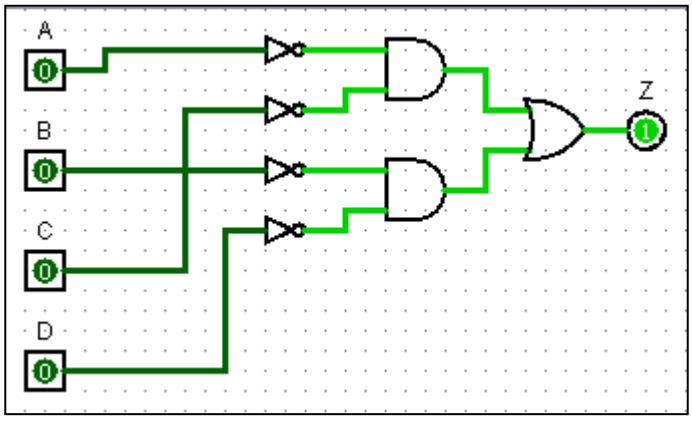
21. Una máquina se controla por cuatro fusibles a, b, c y d. La máquina debe parar si:
    a) Si tres o los cuatro se funden juntos. -
    b) Si a y c, ó b y d se funden Juntos. –

Confeccionar el circuito lógico simplificado que cumpla con las condiciones de protección deseadas. –

Se considera 1 la maquina se apaga, 0 la maquina continúa encendida. Se considera que un fusible esta quemado cuando su señal es 0, caso contrario, es 1.

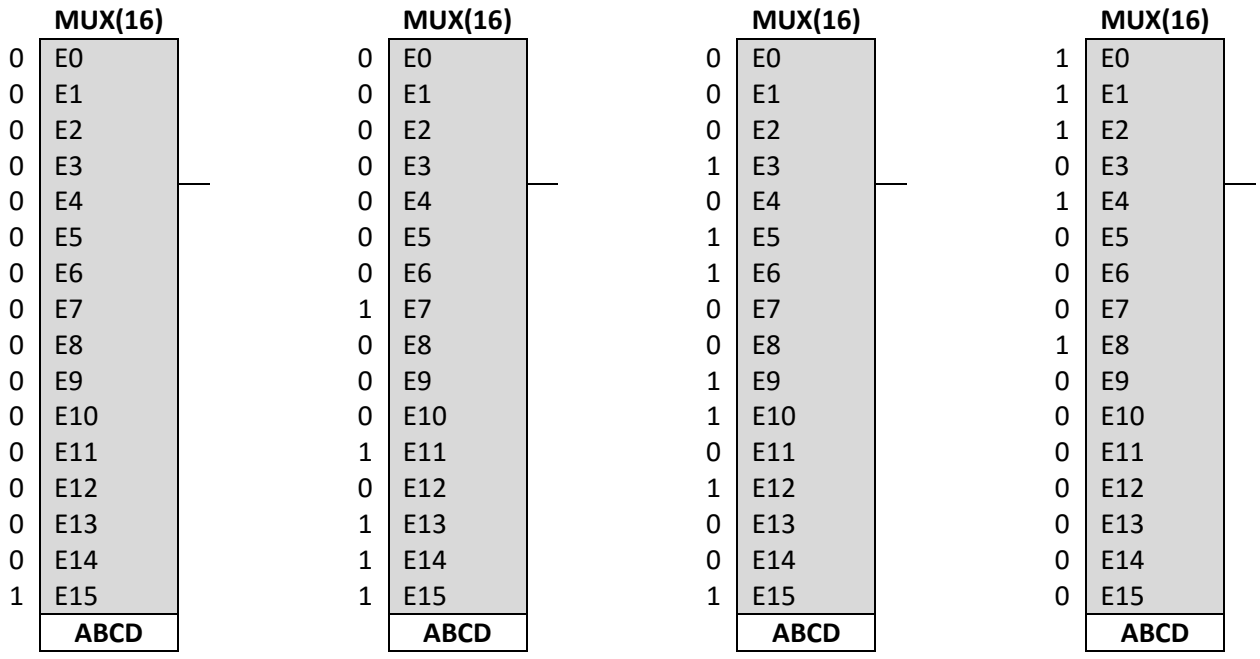
Posición	A	B	C	D	Z
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Z				
CD\AB	00	01	11	10
00	1	1		1
01	1	1		
11				
10	1			1
A'C' + B'D'				



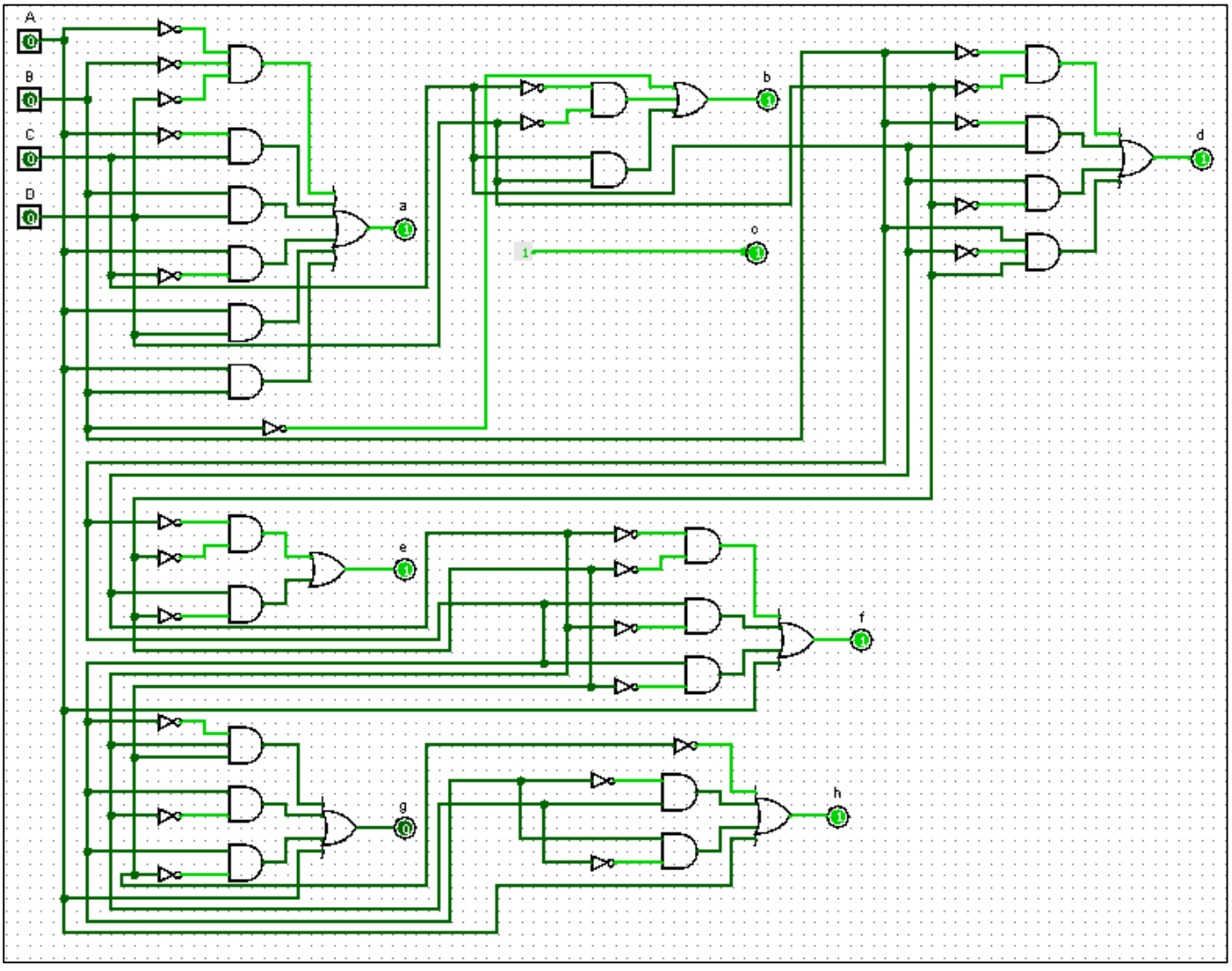
22. En un sistema de control de calidad se extraen muestras de 4 unidades. Cada unidad se examina, indicándose con 1 si fue aprobada, y con 0 si fue rechazada. Las cuatro señales lógicas con los resultados de cada muestra entran a un circuito lógico, que se quiere implementar, cuyas salidas deben indicar:
    a. Si todas las unidades han sido aprobadas. -
    b. Si la mayoría ha sido aprobada. -
    c. Si hay igual número de aprobadas y rechazadas. -
    d. Si hay mayoría de rechazadas. –

Posición	A	B	C	D	a	b	c	d
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	0	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	0	0	1
5	0	1	0	1	0	0	1	0
6	0	1	1	0	0	0	1	0
7	0	1	1	1	0	1	0	0
8	1	0	0	0	0	0	0	1
9	1	0	0	1	0	0	1	0
10	1	0	1	0	0	0	1	0
11	1	0	1	1	0	1	0	0
12	1	1	0	0	0	0	1	0
13	1	1	0	1	0	1	0	0
14	1	1	1	0	0	1	0	0
15	1	1	1	1	1	1	0	0



23. Diseñe un circuito combinacional que responda al funcionamiento de un display de siete segmentos.

N°	A	B	C	D	a	b	c	d	e	f	g	h
0	0	0	0	0	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	0	0	0	0	0
2	0	0	1	0	1	1	1	1	1	0	0	1
3	0	0	1	1	1	1	1	1	0	0	1	1
4	0	1	0	0	0	1	1	0	0	1	1	1
5	0	1	0	1	1	0	1	1	0	1	1	1
6	0	1	1	0	1	0	1	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1	1
X	1	0	1	0	X	X	X	X	X	X	X	X
X	1	0	1	1	X	X	X	X	X	X	X	X
X	1	1	0	0	X	X	X	X	X	X	X	X
X	1	1	0	1	X	X	X	X	X	X	X	X
X	1	1	1	0	X	X	X	X	X	X	X	X
X	1	1	1	1	X	X	X	X	X	X	X	X



Guía N° 3: Primera Parte: Lógica Secuencial

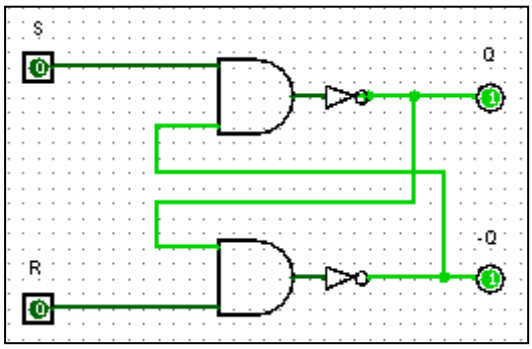
1. Dibuje los circuitos y tabla de verdad, y obtenga las ecuaciones características de los biestables asincrónicos: R-S, J-K

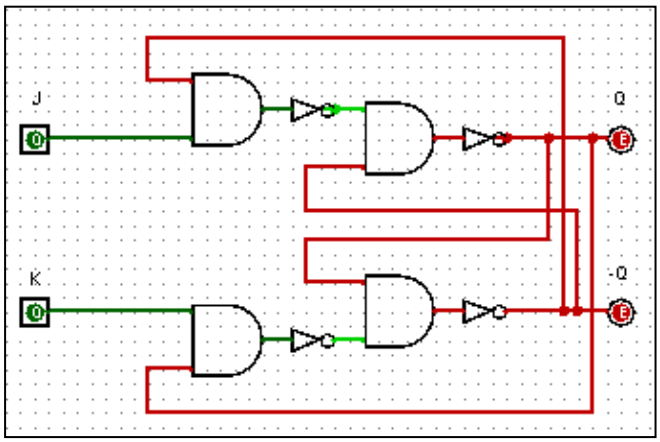
R-S

S	R	Estado presente Q(t)	Estado siguiente Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

J-K

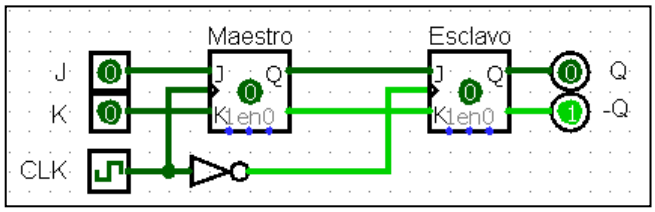
J	K	Estado presente Q(t)	Estado siguiente Q(t+1)
0	0	0	1
1	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1



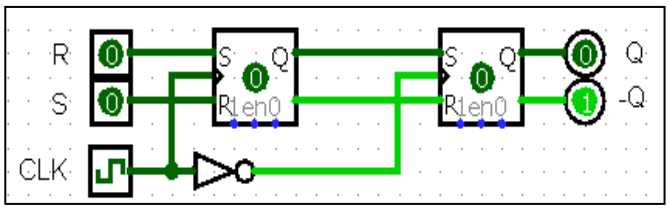


2. Dibuje los circuitos de los biestables R-S y J-K sincrónicos en configuración maestro-esclavo

JK Maestro-Esclavo



RS Maestro-Esclavo



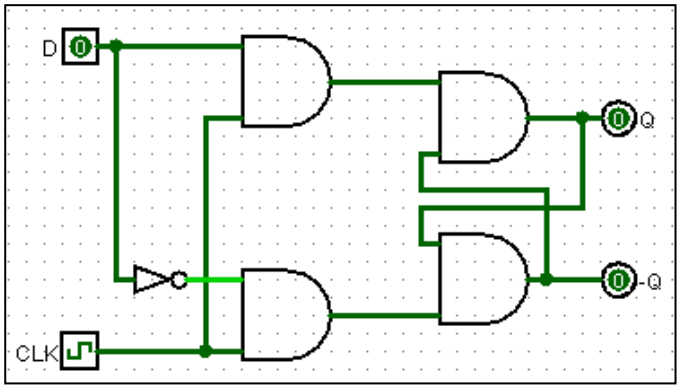
3. Dibuje los circuitos y tabla de verdad, y obtenga las ecuaciones características de los biestables sincrónicos: T, D

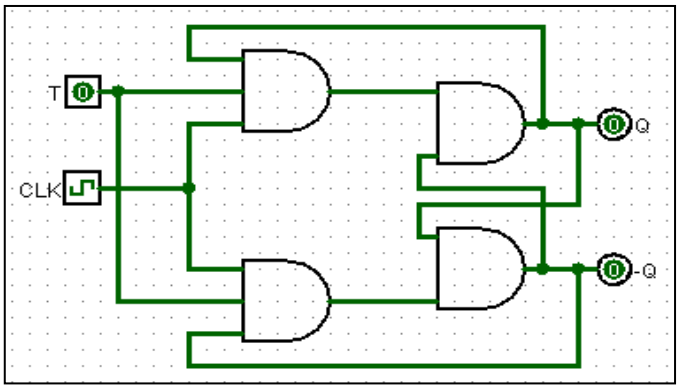
T

Qn	T	Q(n+1)
0	0	0
0	1	1
1	0	1
1	1	0

D

Qn	D	Q(n+1)
0	0	0
0	1	1
1	0	0
1	1	1





4. **Ejercicio Complementario:** Una máquina de estados finitos tiene 4 flip-flops A, B, C, D y una entrada X. Está descripto por las siguientes ecuaciones de estado:
   A(t+1) = (C D' + C' D) X + (C D + C'D') X´
   B(t+1) = A
   C(t+1) = B
   D(t+1) = C
   a) Obtener la secuencia de estados cuando X = 1, empezando por el estado ABCD = 0001
   b) Idem para X = 0, empezando por el estado ABCD = 0000

Ítem a

N°	A	B	C	D	A'	B'	C'	D'	Da	Db	Dc	Dd
1	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	1	0	0	1	1	0	0	1
3	0	0	1	1	0	0	0	1	0	0	0	1
4	0	1	0	0	0	0	1	0	0	0	1	0
5	0	1	0	1	1	0	1	0	1	0	1	0
6	0	1	1	0	1	0	1	1	1	0	1	1
7	0	1	1	1	0	0	1	1	0	0	1	1
8	1	0	0	0	0	1	0	0	0	1	0	0
9	1	0	0	1	1	1	0	0	1	1	0	0
10	1	0	1	0	1	1	0	1	1	1	0	1
11	1	0	1	1	0	1	0	1	0	1	0	1
12	1	1	0	0	0	1	1	0	0	1	1	0
13	1	1	0	1	1	1	1	0	1	1	1	0
14	1	1	1	0	1	1	1	1	1	1	1	1
15	1	1	1	1	0	1	1	1	0	1	1	1

Ítem b

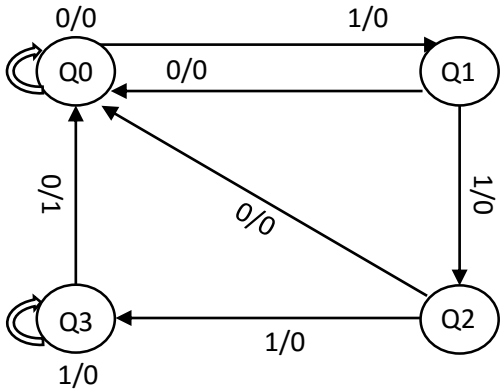
N°	A	B	C	D	A'	B'	C'	D'	Da	Db	Dc	Dd
0	0	0	0	0	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	1	0	0	0	1
3	0	0	1	1	1	0	0	1	1	0	0	1
4	0	1	0	0	1	0	1	0	1	0	1	0
5	0	1	0	1	0	0	1	0	0	0	1	0
6	0	1	1	0	0	0	1	1	0	0	1	1
7	0	1	1	1	1	0	1	1	1	0	1	1
8	1	0	0	0	1	1	0	0	1	1	0	0
9	1	0	0	1	0	1	0	0	0	1	0	0
10	1	0	1	0	0	1	0	1	0	1	0	1
11	1	0	1	1	1	1	0	1	1	1	0	1
12	1	1	0	0	1	1	1	0	1	1	1	0
13	1	1	0	1	0	1	1	0	0	1	1	0
14	1	1	1	0	0	1	1	1	0	1	1	1
15	1	1	1	1	1	1	1	1	1	1	1	1

5. Construir una máquina de estados finitos que detecte una secuencia de tres unos consecutivos en su entrada X. Su salida Z debe valer 1 durante un ciclo de reloj, cuando a la entrada X valga cero y se hubieran suministrado tres unos consecutivos en los ciclos precedentes; si durante cuatro o más ciclos hubiera unos en la entrada X, la salida deberá ser Z = 0. Por ejemplo:

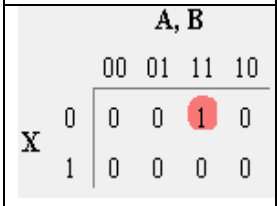
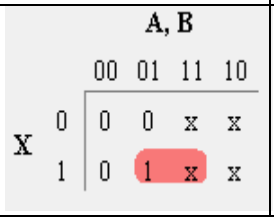
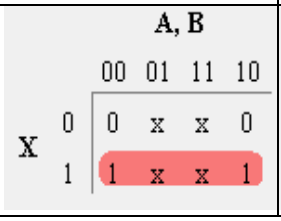
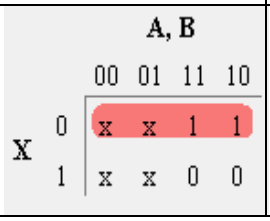
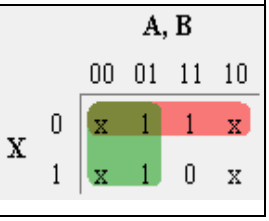
X	0	1	1	0	1	1	1	0	0	1	1	1	1	0
Z	0	0	0	0	0	0	0	1	0	0	0	0	0	1

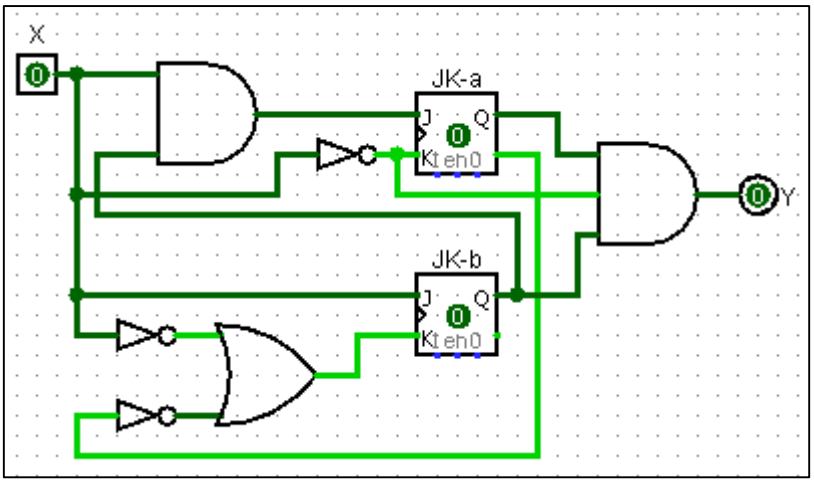
- Dibuje el diagrama de estados

- Obtenga el circuito secuencial utilizando flip flops J-K



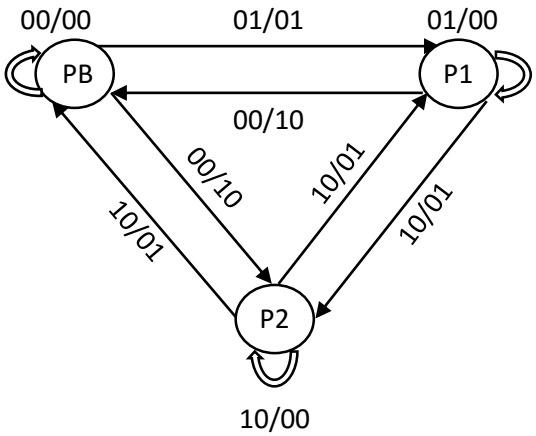
X	A	B	A'	B'	Y	Ja	Ka	Jb	Kb
0	0	0	0	0	0	0	X	0	X
0	0	1	0	0	0	0	X	X	1
0	1	0	0	0	0	X	1	0	X
0	1	1	0	0	1	X	1	X	1
1	0	0	0	1	0	0	X	1	X
1	0	1	1	0	0	1	X	X	1
1	1	0	1	1	0	X	0	1	X
1	1	1	1	1	0	X	0	X	0

Y=X'AB	FF Ja = XB	FF Jb = X	FF Ka = X'	FF Kb = X'+A'
				

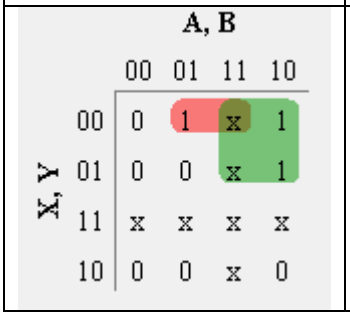
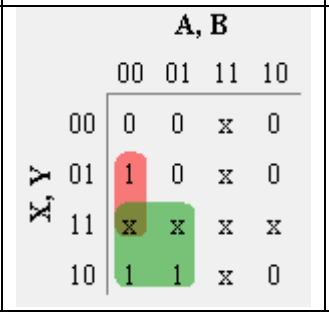
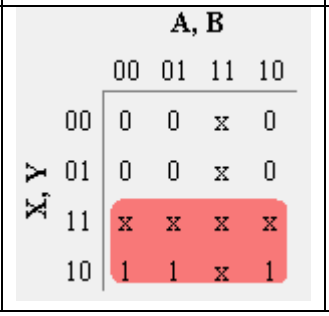
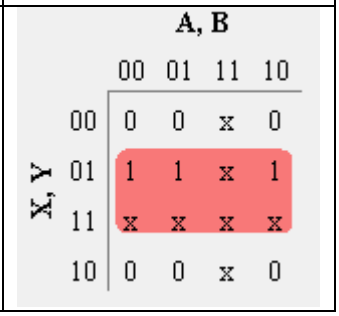


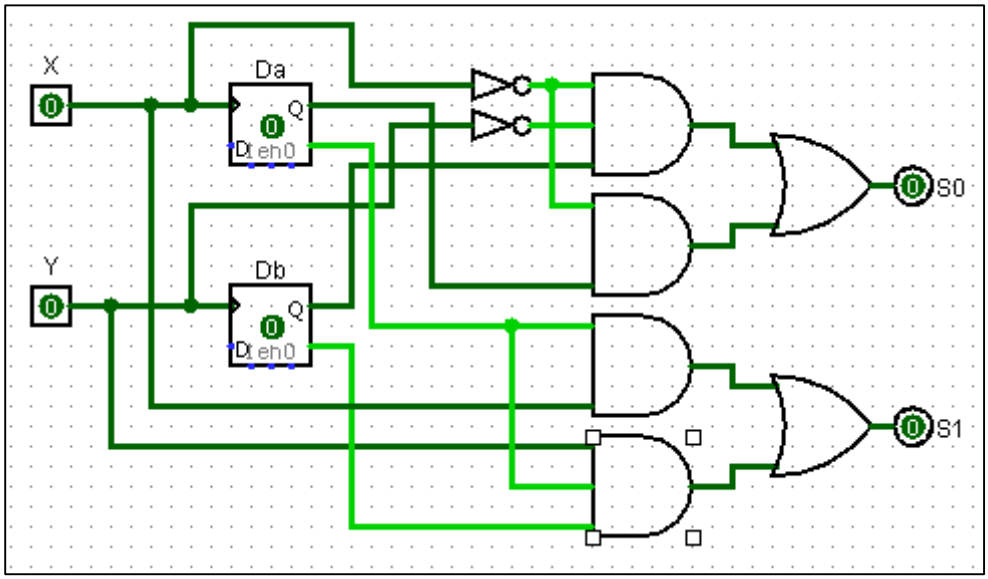
6. Considerando a un ascensor como un autómata, se puede analizar su funcionamiento en respuesta a diversos estímulos externos, como ser la llamada desde otro piso, o el pedido de ascenso y descenso de un pasajero. Diseñar el autómata finito que representa a un ascensor que recorre el camino entre la planta baja (PB), Piso 1 (P1) y Piso 2 (P2), indicando: Diagrama de Transición, Tabla de estados, Simplificaciones, Circuito correspondiente.

Estados		Entradas		Salidas	
PB	00	Ir a PB	00	Permanecer	00
P1	01	Ir a P1	01	Subir	01
P2	10	Ir a P2	10	Bajar	10



X	Y	A	B	A'	B'	S0	S1	Db	Dc
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	1	X	X	X	X	X	X
0	1	0	0	0	1	0	1	0	1
0	1	0	1	0	1	0	0	0	1
0	1	1	0	0	1	1	0	0	1
0	1	1	1	X	X	X	X	X	X
1	0	0	0	1	0	0	1	1	0
1	0	0	1	1	0	0	1	1	0
1	0	1	0	1	0	0	0	1	0
1	0	1	1	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X

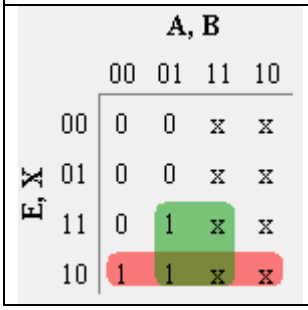
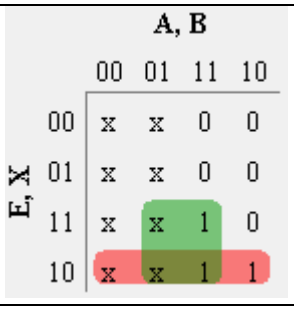
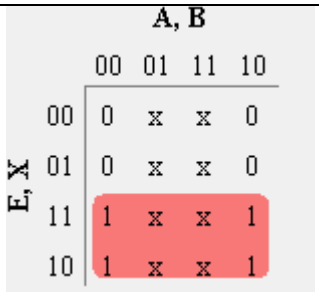
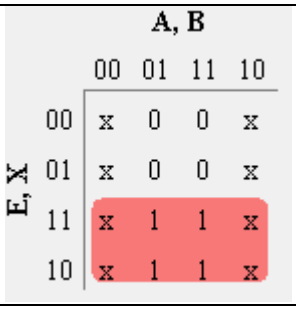
$S_0 = X'Y'B+X'A$	$S_1 = XA'+YA'B'$	F-F $D_A = X$	F-F $D_B = Y$
			

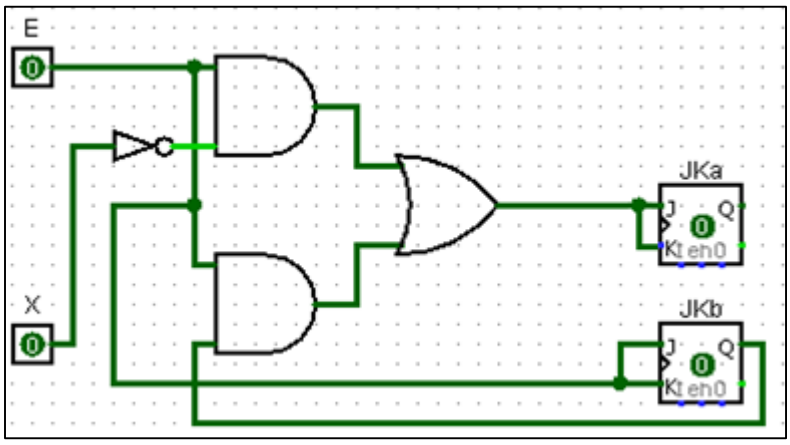


7. Diseñe un circuito secuencial con F-F JK, A y B y dos entradas E y X. Si E = 0, el circuito se mantiene en el mismo estado independientemente del valor de X. Cuando E=1 y X = 1, el circuito pasa por los estados 00 a 01 a 10 a 11

y de nuevo a 00 y repite. Cuando E = 1 y X = 0 el circuito pasa por los estados 00 a 11 a 10 a 01 y de regreso a 00 y repite.

E	X	A	B	A'	B'	Ja	Ka	Jb	Kb
0	0	0	0	0	0	0	X	0	X
0	0	0	1	0	1	0	X	X	0
0	0	1	0	1	0	X	0	0	X
0	0	1	1	1	1	X	0	X	0
0	1	0	0	0	0	0	X	0	X
0	1	0	1	0	1	0	X	X	0
0	1	1	0	1	0	X	0	0	X
0	1	1	1	1	1	X	0	X	0
1	0	0	0	1	1	1	X	1	X
1	0	0	1	1	0	1	X	X	1
1	0	1	0	0	1	X	1	1	X
1	0	1	1	0	0	X	1	X	1
1	1	0	0	0	1	0	X	1	X
1	1	0	1	1	0	1	X	X	1
1	1	1	0	1	1	X	0	1	X
1	1	1	1	0	0	X	1	X	1

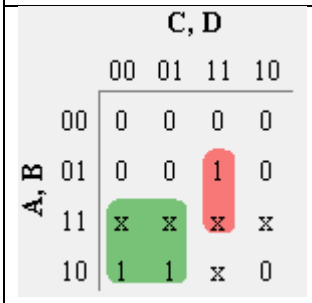
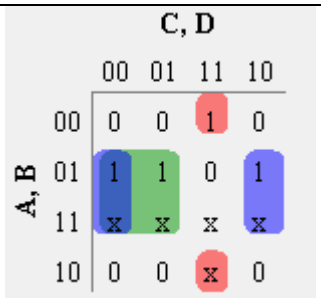
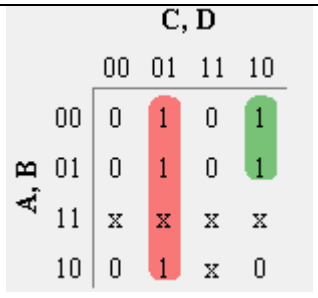
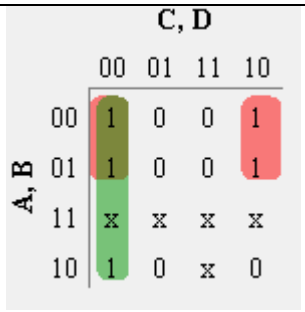
F-F J_a = EX'+EB	F-F K_a = EX'+EB	F-F J_b = E	F-F K_b = E
			

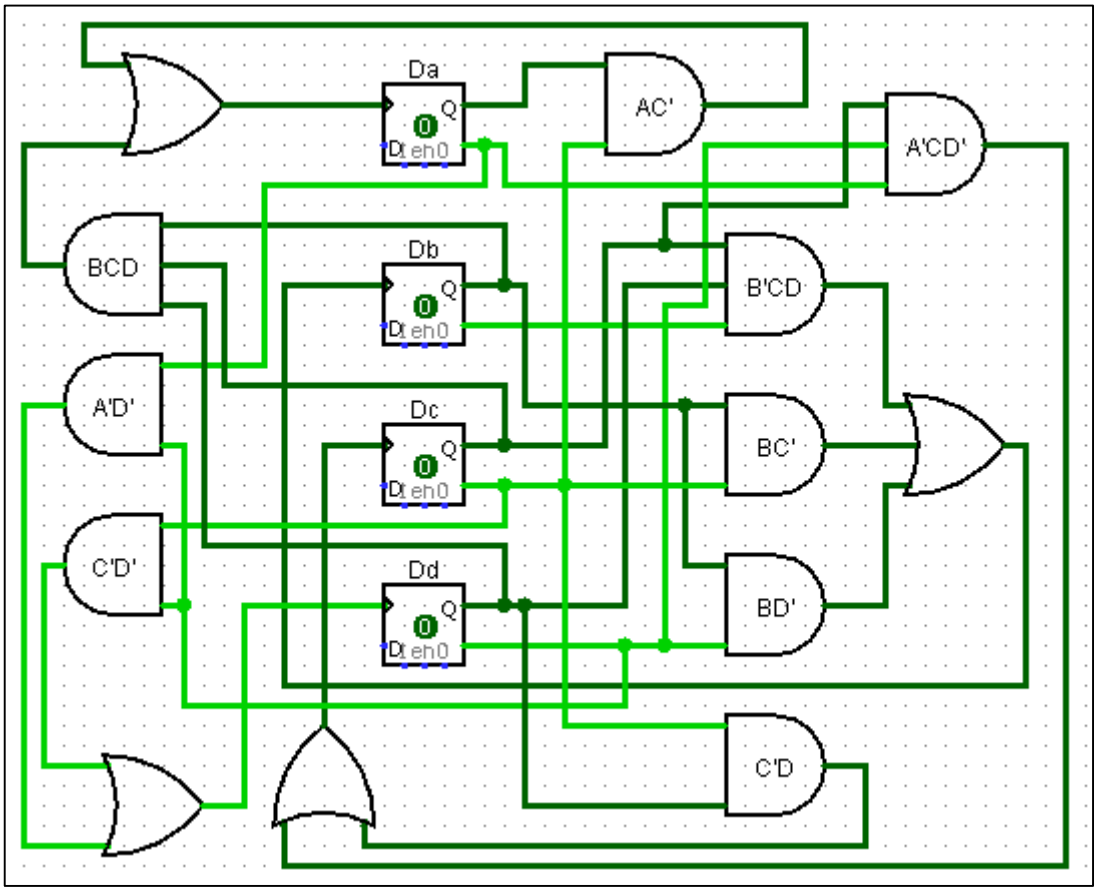


8. Diseñar un contador sincrónico de módulo 11. Utilizar flip-flops D y compuertas NAND como condicionantes.

A	B	C	D	A'	B'	C'	D'	Da	Db	Dc	Dd
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1

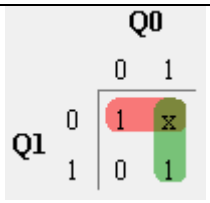
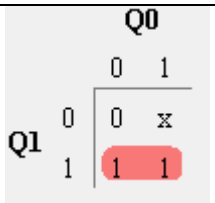
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0	1	0	1	0
1	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

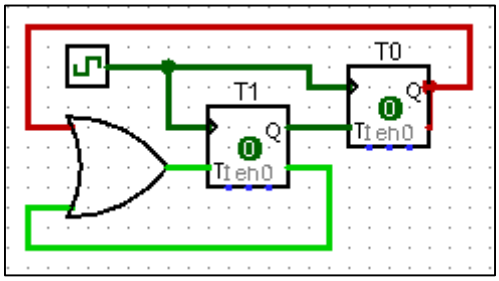
$D_a = BCD + AC'$	$D_b = B'CD + BC' + BD'$	$D_c = C'D + A'CD'$	$D_d = A'D' + C'D'$
			



9. Diseñar un contador sincrónico de módulo 3 usando flip-flops T. Imponer que el contador recorra los estados $Q_1Q_0 = 00 – 11 – 10 – 00$.

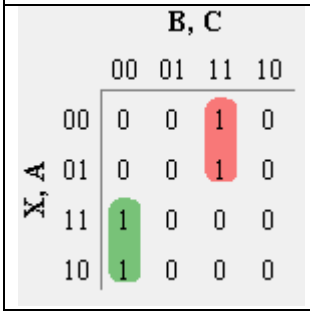
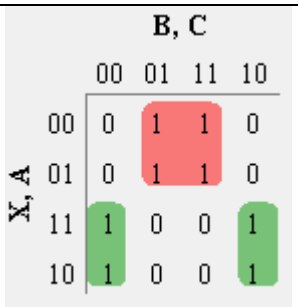
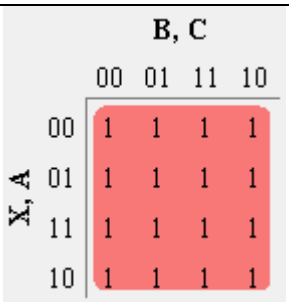
Q_1	Q_0	$Q1_1$	$Q0_0$	T_1	T_0
0	0	1	0	1	0
0	1	X	X	X	X
1	0	1	1	0	1
1	1	0	0	1	1

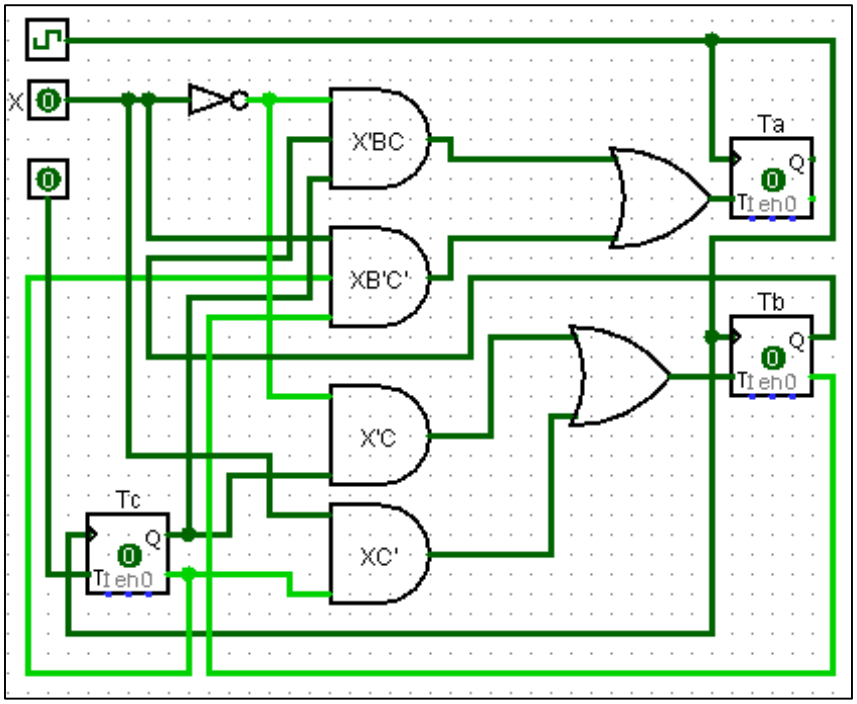
$T_1 = Q1' + Q0$	$T_0 = Q1$
	



10. Diseñar un contador sincrónico reversible de módulo 8.

X	A	B	C	A'	B'	C'	Ta	Tb	Tc
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	1	1	0	0	1
0	1	1	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1
1	1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	1

$T_a = X'BC + XB'C'$	$T_b = X'C + XC'$	$T_c = 1$
		


X
X'BC
XB'C'
X'C
XC'
Ta
Tb
Tc
Q
Tieh0

Guía N° 4: Introducción al Assembly

1. Cargar el dato 20h en el registro AL.

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV AL, 20H; Movemos el valor 20h al registro AL
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

2. Escribir el dato 20 en la dirección de memoria 500h.

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV AX, 20H; Movemos el valor 20h al registro AX
    MOV BX, 500H; Referenciamos la posición 500 de memoria
    MOV [BX], AX; Movemos el valor del registro AX a la posición referenciada por
    BX
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

3. Leer el dato presente en la dirección de memoria 600h dejándolo en el registro AL.

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV BX, 600H; Referenciamos la posición 600h de memoria
    MOV AL, [BX]; Movemos el valor de la posición de memoria referenciada por BX
    al registro AL
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

4. Sumar los números 3 y 7 dejando el resultado en el registro AL.

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV AL, 3H; Movemos el valor 3h al registro AL
    MOV CL, 7H; Movemos el valor 7h al registro CL
    ADD AL, CL; Sumamos los valores de los registros AL y CL y lo almacenamos en
    AL
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

5. Sumar el contenido de la dirección 500h con la dirección 501h dejando el resultado en 502h.

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV BX, 500H; Referenciamos la posición 500h de memoria
    MOV AX, [BX]; Movemos el valor de la posición de memoria referenciada por BX
    al registro AX
    MOV BX, 501H; Referenciamos la posición 501h de memoria
    MOV CX, [BX]; Movemos el valor de la posición de memoria referenciada por BX
    al registro CX
    ADD AX, CX; Sumamos los valores de los registros AX y CX y lo almacenamos en
    AX
    MOV BX, 502H; Referenciamos la posición 502h de memoria
    MOV [BX], AX; Movemos el valor del registro AX a la posición de memoria
    referenciada por BX
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

6. Escribir un programa que realice la suma de dos datos inmediatos (el 10h y el 20h) que se encuentran en los registros AX y DX del procesador. El resultado lo almacenará en AX.

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV AX, 10H; Movemos el valor 10h al registro AX
    MOV DX, 20H; Movemos el valor 20h al registro DX
    ADD AX, DX; sumamos los valores de los registros AX y DX y lo almacenamos en
    AX
    HLT; Establecemos el procesador en estado de pausa
```

```
END; Finalizamos
```

7. Cambiar los datos que se suman por los datos 1234h y 1000h. Repetir los pasos del Ejercicio 6 hasta terminar la simulación. (El resultado es el mismo que el ejercicio 6 porque se sobrescriben los valores)

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV AX, 1234H; Movemos el valor 1234h al registro AX
    MOV DX, 1000H; Movemos el valor 1000h al registro DX
    ADD AX, DX; sumamos los valores de los registros AX y DX y lo almacenamos en
    AX
    MOV AX, 10H; Movemos el valor 10h al registro AX
    MOV DX, 20H; Movemos el valor 20h al registro DX
    ADD AX, DX; sumamos los valores de los registros AX y DX y lo almacenamos en
    AX
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

8. Cambiar la operación aritmética en el ejercicio 6 por una resta. Para ello, cambiar la instrucción ADD AX, DX por SUB AX, DX en el código.

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV AX, 10H; Movemos el valor 10h al registro AX
    MOV DX, 20H; Movemos el valor 20h al registro DX
    SUB AX, DX; restamos al valor del registro AX el valor del registro DX y lo
    almacenamos en AX
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

9. Cambiar el orden de los operandos del ejercicio anterior. Escribir SUB DX, AX y observar que pasa con los flags (banderas) de estado del computador.

```
ORG 2000H; Iniciamos en posición de memoria 2000h
    MOV AX, 10H; Movemos el valor 10h al registro AX
    MOV DX, 20H; Movemos el valor 20h al registro DX
    SUB DX, AX; restamos al valor del registro DX el valor del registro AX y lo
    almacenamos en DX
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

Ningún Flag fue afectado por las operaciones anteriores debido a que se le resta al número 20H, el número 10H obteniendo como resultado un numero positivo el cual no genera cambios de signo, desbordamiento, acarreos, etc.

Flags						
I	Z	S	O	C	A	P
-	-	-	-	-	-	-
Interruptions	Zero	Sign	Overflow	Carry	Auxiliary	Parity

Preguntas

a) Realizar las siguientes acciones, indicando a continuación los comandos que se utilizan:

- Visualizar la dirección 1000h de memoria.
  - **Comando**: D 1000
- Grabar en la dirección 1000h de memoria el byte 8Ah
  - **Comando**: E 1000 8A
- Cargar en el registro BH el byte FF.-
  - **Comando**: R BH FF

- Cargar en el registro CX la palabra 5678h.-
  - **Comando**: R CX 5678

- Cambiar a la pantalla 1, visualizar los diferentes conexionados de periféricos y volver a la pantalla 0.
  - **Comando**: p1 (pantalla 1), p0 (pantalla 0), c0, c1, c2, c3 (conexionados de periféricos)

b) ¿Cuál es el estado de los FLAGS después de la ejecución de las instrucciones en los ejercicios 8 y 9?

Flags Ejercicio N° 8: (Flags afectados determinados con X)

Flags						
I	Z	S	O	C	A	P
-	-	X	-	X	-	X

Flags Ejercicio N° 9: (Flags afectados determinados con X)

Flags						
I	Z	S	O	C	A	P
-	-	-	-	-	-	-

c) Explique que realizan las instrucciones MOV, ADD Y SUB

MOV: Es una instrucción de transferencia de datos, mueve datos de memoria a registro y viceversa, de registro a posición registro, datos inmediatos a memoria y datos inmediatos a registros.
ADD: Es una instrucción aritmética de suma, suma datos de memoria a registro y viceversa, de registro a posición registro, datos inmediatos a memoria y datos inmediatos a registros.
SUB: Es una instrucción aritmética de resta, resta datos de memoria a registro y viceversa, de registro a posición registro, datos inmediatos a memoria y datos inmediatos a registros.

d) Ejecución de un programa:

¿Qué ocurre en el registro IP después de leer un byte del programa de la memoria? Después de leer un byte, el registro IP (Instruction Pointer – Contador de programa) contendrá la dirección de la próxima instrucción a ser ejecutada. El contador de programa es incrementado automáticamente en cada ciclo de instrucción de tal manera que las instrucciones son leídas en secuencia desde la memoria.

¿Qué valores se van cargando en el registro IR? En el registro IR (Instruction's Register – Registro de Instrucciones) se cargan las instrucciones a ser ejecutadas

¿Qué instrucciones va indicando el Decodificador? El Decodificador va indicando que operaciones están por ser realizadas, MOV, ADD, HLT, etc.

¿Con qué valores queda el registro de indicadores (flags)? El registro de indicadores (flags) varía entre los valores 0 y 1, que dependen de la operación realizada, suma resta, etc.

Guía N° 5: Subrutinas y transferencia

1. Multiplicación de números sin signo. Escribir un programa que calcule el producto entre dos números sin signo almacenados en la memoria del microprocesador:

   a) Sin hacer llamadas a subrutinas, resolviendo el problema desde el programa principal.

```
ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV AX, 5H; movemos el valor 5h al registro AX
    MOV BX, 2H; movemos el valor 2h al registro BX
    MOV CX, 0; establecemos en 0 el valor del registro CX
    MUL: ADD CX, AX; establecemos una etiqueta MUL, Sumamos los valores de los
    registros CX y AX y lo almacenamos en el registro CX
    DEC BX; Decrementamos en uno el valor del registro BX
    JNZ MUL; Jump If Not Zero, hacer un salto a la etiqueta MUL si la última
    operación no devolvió como resultado cero.
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

   b) Llamando a una subrutina MUL para efectuar la operación, pasando los parámetros por valor desde el programa principal a través de registros.

```
ORG 3000H; Posición de memoria para la subrutina
     MUL: ADD CX, AX; establecemos una etiqueta MUL, Sumamos los valores de los
     registros CX y AX y lo almacenamos en el registro CX
     DEC BX; Decrementamos en uno el valor del registro BX
     JNZ MUL; Jump If Not Zero, hacer un salto a la etiqueta MUL si la última
     operación no devolvió como resultado cero.
RET; Instrucción de cambio de flujo, retorna a la instrucción desde donde fue
llamado, asociado a subrutinas.

ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV AX, 5H; movemos el valor 5h al registro AX
    MOV BX, 2H; movemos el valor 2h al registro BX
    MOV CX, 0; establecemos en 0 el valor del registro CX
    CALL MUL; llamamos a la subrutina MUL
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

   c) Llamando a una subrutina MUL, pasando los parámetros por referencia desde el programa principal a través de registros.

```
ORG 1000H; Posición de memoria para variables establecidas por el usuario
    NUM1 DW 5H; establecemos variable NUM1 de tipo Word con el valor 5h
    NUM2 DW 3H; establecemos variable NUM2 de tipo Word con el valor 3h
    NUMR DW ?; establecemos variable de tipo Word sin valor inicial

ORG 3000H; Posición de memoria 3000h para subrutinas
    MUL: CMP DX, 0; etiqueta MUL, CMP: comparamos fuente con destino, si son
    iguales devuelve 0 (cero)
    JZ FIN; Jump if Zero, Instrucción de cambio de flujo, si la última operación
    tuvo como resultado el valor 0 (cero), va a saltar a la etiqueta FIN.
    ADD CX, AX; sumamos los valores de los registros CX y AX y lo almacenamos en
    CX
    DEC DX; Decrementamos en uno el valor del registro DX
    CALL MUL; llamamos a subrutina MUL
FIN: RET; etiqueta FIN, RET: Instrucción de cambio de flujo, retorna a la
instrucción desde donde fue llamado.

ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
     MOV AX, OFFSET NUM1; movemos al registro AX la posición en memoria de la
     variable NUM1
     MOV DX, OFFSET NUM2; movemos al registro DX la posición en memoria de la
     variable NUM2
```

```
        MOV CX, OFFSET NUMR; movemos al registro CX la posición en memoria de la
        variable NUMR
        MOV BX, AX; movemos al registro BX el valor del registro AX, en este caso es
        la posición en memoria de la variable NUM1
        MOV AX, [BX]; movemos al registro AX el valor almacenado en la posición de
        memoria establecida en BX (valor del NUM1)
        MOV BX, DX; movemos al registro BX el valor del registro DX, en este caso es
        la posición en memoria de la variable NUM2
        MOV DX, [BX]; movemos al registro DX el valor almacenado en la posición de
        memoria establecida en BX (valor del NUM1)
        MOV BX, CX; movemos al registro BX el valor del registro CX, en este caso es
        la posición en memoria de la variable NUMR
        MOV CX, 0; movemos el valor 0 (cero) al registro CX
        CALL MUL; llamamos a la subrutina MUL (posición de memoria 3000h)
        HLT; Establecemos el procesador en estado de pausa
    END; Finalizamos
```

Explicar detalladamente:

- **Todas las acciones que tienen lugar al ejecutarse la instrucción CALL MUL:** Llama a la subrutina MUL que se encuentra en la posición de memoria 3000h, compara (CMP) si el valor del registro DX es igual a 0 (cero), si lo es hace un salto (JZ) a la etiqueta FIN la cual contiene la instrucción RET que corresponde a un cambio de flujo, retornando a la instrucción de llamada original (CALL MUL) en el programa principal, caso contrario, suma los valores de los registros CX y AX y lo almacenamos en CX, decrementa en uno el valor del registro DX y vuelve a llamar a la subrutina MUL.

- **¿Qué operación se realiza con la instrucción RET?:** RET es una instrucción de cambio de flujo, retorna a la instrucción desde donde fue llamado. Está asociado a subrutinas, retorna a llamadas CALL.

2. Escribir un programa que calcule el producto entre dos números sin signo almacenados en la memoria del microprocesador llamando a una subrutina MUL, pero en este caso pasando los parámetros por valor y por referencia a través de la pila.

```
ORG 1000H; Posición de memoria para variables
    NUM DW 5h; almacenar 5h en variable NUM
    NUM2 DW 2h; almacenar 2h en variable NUM2
    RES DW ?; establecemos un valor nulo para utilizar luego

ORG 3000H; Posición de memoria 3000h para subrutinas
    MUL: POP AX; Lo último que quedo en Pila antes de llamar a la subrutina es la
    posición del CALL MULL, por tal motivo lo quitamos y almacenamos en AX
    POP CX; Extraemos el ultimo valor de la Pila y lo almacenamos en el registro CX,
    en este caso el valor de la variable NUM2
    POP DX; Extraemos el ultimo valor de la Pila y lo almacenamos en el registro DX,
    en este caso el valor de la variable NUM
    MOV BX, 0; Establecemos en 0 el registro BX
    BUCLE: add BX, CX; Sumamos los valores de los registros BX y CX y lo almacenamos
    en BX
    DEC DX; Decrementamos en uno el valor del registro DX
    JNZ BUCLE; sino es cero, volvemos al BUCLE
    POP DX; Extraemos el ultimo valor de la Pila y lo almacenamos en el registro DX,
    en este caso la posición de la variable RES
    PUSH AX; Introducimos en la Pila la posición de la subrutina CALL MULL que esta
    almacenada en AX para poder retornar
RET; Instrucción de cambio de flujo, retorna a la instrucción desde donde fue llamado.

ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV AX, OFFSET RES; Movemos al registro AX la posición de la variable RES
    PUSH AX; Introducimos en la Pila el valor de AX
    MOV AX, NUM; Movemos al registro AX el valor de la variable NUM
    PUSH AX; Introducimos en la Pila el valor de AX
    MOV AX, NUM2; Movemos al registro AX el valor de la variable NUM2
    PUSH AX; Introducimos en la Pila el valor de AX
```

```
    CALL MUL; Llamamos a subrutina
    MOV AX, BX; Movemos al registro AX el valor del registro BX, en este caso el
    resultado de la multiplicación
    MOV BX, DX; Movemos al registro BX el valor del registro DX, en este caso la
    posición de la variable RES
    MOV [BX], AX; Movemos al registro BX que esta referenciado a la variable RES, el
    valor del resultado que está almacenado en AX
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

Responder brevemente:

a) **¿Qué función cumple el registro temporal RI?** RI Registro temporal de direcciones (sólo aparece cuando es necesario). Por ejemplo, cuando se desea sumar a un registro el valor de una variable, en este caso el registro RI contendría temporalmente la posición en memoria de la variable.

b) **¿Qué se guarda en AX al ejecutarse MOV AX, OFFSET RES?** En el registro AX al ejecutarse la instrucción MOV AX, OFFSET RES, se guarda la posición en memoria de la variable RES.

c) **¿Cómo se pasa la variable RES a la pila, por valor o por referencia? ¿Qué ventaja tiene esto?** La variable RES se pasa por referencia a la pila, esto tiene como ventaja un mayor rendimiento debido a que solo se pasa la posición en memoria de la variable, permitiendo también mantener la mayor cantidad posible de registro libres.

d) **¿Cómo trabajan las instrucciones PUSH y POP?** La instrucción PUSH introduce un valor o referencia a posición de memoria en la cola de la pila, LIFO (Last In First Out – Ultimo en entrar, primero en salir), la instrucción POP quita de la cola de la pila el último elemento almacenado, sea un valor o referencia a posición de memoria.

3. Suma de números de 32 bits. Escribir un programa que calcule la suma de dos números de 32 bits almacenados en la memoria del microprocesador:

a) Sin hacer llamados a subrutinas, resolviendo el problema desde el programa principal

```
ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV AX, 5234H; movemos el valor 5234h al registro AX
    MOV CX, 2123H; movemos el valor 2123h al registro CX
    ADD AX, CX; sumamos los valores de los registros AX y CX y lo almacenamos en
    el registro AX.
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

b) Llamando a una subrutina SUM32 para efectuar la operación, pasando los parámetros por valor desde el programa principal a través de registros

```
ORG 3000H; Posición de memoria 3000h para subrutinas
    SUM32: ADD AX, CX; sumamos los valores de los registros AX y CX y lo almacenamos
    en el registro AX.
RET; RET: Instrucción de cambio de flujo, retorna a la instrucción desde donde
fue llamado.

ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV AX, 5234H; movemos el valor 5234h al registro AX
    MOV CX, 2123H; movemos el valor 2123h al registro CX
    CALL SUM32; Llamamos a subrutina
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

c) Llamando a una subrutina SUM32, pasando los parámetros por referencia desde el programa principal a través de registros.

```
ORG 1000H; Posición de memoria para variables establecidas por el usuario
```

```
    NUM1 DW 5234H; establecemos variable NUM1 de tipo Word con el valor 5234h
    NUM2 DW 2123H; establecemos variable NUM2 de tipo Word con el valor 2123h
    NUMR DW ?; establecemos variable de tipo Word sin valor inicial

ORG 3000H; Posición de memoria 3000h para subrutinas
     SUM32: MOV BX, AX; movemos al registro BX el valor del registro AX, en este
     caso es la posición en memoria de la variable NUM1
     MOV AX, [BX]; movemos al registro AX el valor almacenado en la posición de
     memoria establecida en BX (valor del NUM1)
     MOV BX, CX; movemos al registro BX el valor del registro CX, en este caso es
     la posición en memoria de la variable NUM2
     MOV CX, [BX]; movemos al registro DX el valor almacenado en la posición de
     memoria establecida en BX (valor del NUM1)
     ADD AX, CX; sumamos los valores de los registros AX y CX y lo almacenamos en
     el registro AX.
     MOV BX, DX; movemos al registro BX el valor del registro DX, en este caso,
     la posición en memoria de la variable NUMR
     MOV [BX], AX; Movemos al registro BX que esta referenciado a la variable
     NUMR, el valor del resultado que está almacenado en AX
RET; RET: Instrucción de cambio de flujo, retorna a la instrucción desde donde
fue llamado.

ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV AX, OFFSET NUM1; movemos el valor de posición en memoria de la variable
    NUM1 al registro AX
    MOV CX, OFFSET NUM2; movemos el valor de posición en memoria de la variable
    NUM2 al registro CX
    MOV DX, OFFSET NUMR; movemos el valor de posición en memoria de la variable
    NUMR al registro DX
    CALL SUM32; Llamamos a subrutina
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

d) Llamando a una subrutina SUM32, pero en este caso pasando los parámetros por valor y por referencia a través de la pila.

```
ORG 1000H; Posición de memoria para variables
    NUM1 DW 5234H; establecemos variable NUM1 de tipo Word con el valor 5234h
    NUM2 DW 2123H; establecemos variable NUM2 de tipo Word con el valor 2123h
    NUMR DW ?; establecemos variable de tipo Word sin valor inicial

ORG 3000H; Posición de memoria 3000h para subrutinas
    SUM32: POP AX; Lo último que quedo en Pila antes de llamar a la subrutina es la
    posición del CALL SUM32, por tal motivo lo quitamos y almacenamos en AX
    POP CX; Extraemos el ultimo valor de la Pila y lo almacenamos en el registro CX,
    en este caso el valor de la variable NUM2
    POP DX; Extraemos el ultimo valor de la Pila y lo almacenamos en el registro DX,
    en este caso el valor de la variable NUM1
    add CX, DX; Sumamos los valores de los registros CX y DX y lo almacenamos en CX
    MOV BX, CX; movemos al registro BX el valor del registro CX, el resultado de la
    suma
    POP DX; Extraemos el ultimo valor de la Pila y lo almacenamos en el registro DX,
    en este caso la posición de la variable NUMR
    PUSH AX; Introducimos en la Pila la posición de la subrutina CALL SUM32 que esta
    almacenada en AX para poder retornar
RET; Instrucción de cambio de flujo, retorna a la instrucción desde donde fue llamado.

ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV AX, OFFSET NUMR; Movemos al registro AX la posición de la variable NUMR
    PUSH AX; Introducimos en la Pila el valor de AX
    MOV AX, NUM1; Movemos al registro AX el valor de la variable NUM1
    PUSH AX; Introducimos en la Pila el valor de AX
    MOV AX, NUM2; Movemos al registro AX el valor de la variable NUM2
    PUSH AX; Introducimos en la Pila el valor de AX
    CALL SUM32; Llamamos a subrutina
```

```
    MOV AX, BX; Movemos al registro AX el valor del registro BX, en este caso el
    resultado de la suma
    MOV BX, DX; Movemos al registro BX el valor del registro DX, en este caso la
    posición de la variable NUMR
    MOV [BX], AX; Movemos al registro BX que esta referenciado a la variable NUMR, el
    valor del resultado que está almacenado en AX
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

4. Escribir una subrutina SWAP que intercambie dos datos de 16 bits almacenados en memoria. Los parámetros deben ser pasados por referencia desde el programa principal a través de la pila.

```
ORG 1000H; Posición de memoria para variables
    NUM1 DW 34H; establecemos variable NUM1 de tipo Word con el valor 34h
    NUM2 DW 23H; establecemos variable NUM2 de tipo Word con el valor 23h

ORG 3000H; Posición de memoria 3000h para subrutinas
    SWAP: POP AX; Lo último que quedo en Pila antes de llamar a la subrutina es la
    posición del CALL SWAP, por tal motivo lo quitamos y almacenamos en AX
    POP CX; Extraemos el ultimo valor de la Pila y lo almacenamos en el registro CX,
    en este caso el valor de la posición en memoria de la variable NUM2
    POP DX; Extraemos el ultimo valor de la Pila y lo almacenamos en el registro DX,
    en este caso el valor de la posición en memoria de la variable NUM2
    MOV BX, CX; movemos al registro BX la posición en memoria del registro CX (NUM2)
    PUSH [BX]; Introducimos en la Pila el valor de BX (valor de NUM2)
    MOV BX, DX; movemos al registro BX la posición en memoria del registro DX (NUM1)
    PUSH [BX]; Introducimos en la Pila el valor de BX (valor de NUM1)
    MOV BX, CX; movemos al registro BX la posición en memoria del registro CX (NUM2)
    POP [BX]; Quitamos de la pila el ultimo valor almacenado (NUM1) y lo almacenamos
    en la dirección referenciada por BX (NUM2)
    MOV BX, DX; movemos al registro BX la posición en memoria del registro DX (NUM1)
    POP [BX]; Quitamos de la pila el ultimo valor almacenado (NUM2) y lo almacenamos
    en la dirección referenciada por BX (NUM1)
    PUSH AX; Introducimos en la Pila la posición de la subrutina CALL SWAP que esta
    almacenada en AX para poder retornar
RET; Instrucción de cambio de flujo, retorna a la instrucción desde donde fue llamado.

ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV AX, OFFSET NUM1; Movemos al registro AX la posición de la variable NUM2
    PUSH AX; Introducimos en la Pila el valor de AX
    MOV AX, OFFSET NUM2; Movemos al registro AX la posición de la variable NUMR
    PUSH AX; Introducimos en la Pila el valor de AX
    CALL SWAP; Llamamos a subrutina
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

5. Escriba la subrutina RESTO que calcule el resto de la división entre 2 números positivos. Dichos números deben pasarse por valor desde el programa principal a la subrutina a través de registros. División por medio de restas sucesivas.

```
    ORG 3000H; Posición de memoria 3000h para subrutinas
        RESTO: SUB AX, CX; restamos al valor del registro AX, el valor del registro
        CX y lo almacenamos en el registro AX
        JS FIN; Si el resultado anterior es con signo saltamos a subrutina FIN
        movemos al registro BX el valor del registro CX, en este caso es la posición
        en memoria de la variable NUM2
        CALL RESTO; Llamamos a subrutina
        FIN: ADD AX, CX; como el resultado final fue negativo, sumamos el valor del
        registro CX al registro AX obteniendo asi el resto de la división
         MOV DX, AX; movemos el resultado de la operación anterior al registro DX
    RET; etiqueta FIN, RET: Instrucción de cambio de flujo, retorna a la instrucción
    desde donde fue llamado.


    ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
```

```
    MOV AX, 35; movemos el valor 35 al registro AX
    MOV CX, 5; movemos el valor 5h al registro CX
    CALL RESTO; Llamamos a subrutina
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

- Analizar el funcionamiento de la siguiente subrutina y su programa principal:

```
ORG 3000H; Posición de memoria 3000h para subrutinas
    MUL: CMP AX, 0; comparamos el registro AX con el valor 0 (cero)
    JZ FIN; si el resultado anterior dio 0 (cero) saltamos a la subrutina FIN
    ADD CX, AX; Sumamos los valores de los registros CX y AX y lo almacenamos en
    el registro AX
    DEC AX; Decrementamos en uno el valor del registro AX
    CALL MUL; llamamos a la subrutina
FIN: RET; Instrucción de cambio de flujo, retorna a la instrucción desde donde
fue llamado, asociado a subrutinas.

ORG 2000H; Iniciamos en posición de memoria 2000h, programa principal
    MOV CX, 0; movemos el valor 0 al registro CX
    MOV AX, 3; movemos el valor 3 al registro AX
    CALL MUL; llamamos a subrutina
    HLT; Establecemos el procesador en estado de pausa
END; Finalizamos
```

- **¿Qué hace la subrutina?** La subrutina realiza una suma sucesiva de un mismo número decrementandolo en uno en cada ciclo de la suma y almacenando el valor de la suma en una variable diferente. Los pasos de la subrutina son: comparar si el valor del registro AX es igual a cero, si lo es, el programa hará un salto a la subrutina FIN, caso contrario sumará los valores de los registros CX y AX y lo almacenará en el registro CX, paso siguiente decrementará en uno el valor del registro AX.

- **¿Cuál será el valor final de CX?** El valor final de CX es 6

- **¿Cuál será la limitación para determinar el valor más grande que se le puede pasar a la subrutina a través de AX?** Las limitaciones para valores del registro AX es de una palabra de 32bits (FF FF)

Guía N° 6: Direccionamiento e Interrupciones

1. Escritura de datos en la pantalla de comandos: implementar un programa en el lenguaje assembler del simulador MSX88 que muestre en la pantalla de comandos un mensaje previamente almacenado en memoria de datos, aplicando la interrupción por software INT 7.

```
ORG 1000H
    MSJ DB "ARQUITECTURA DE COMPUTADORAS-"
    DB "FACENA - "
    DB "UNNE"
FIN DB ?

ORG 2000H
    MOV BX, OFFSET MSJ
    MOV AL, OFFSET FIN-OFFSET MSJ
    INT 7
    INT 0
END
```

2. Lectura de datos desde el teclado: Escribir un programa que solicite el ingreso de un número por teclado e inmediatamente lo muestre en la pantalla de comandos, haciendo uso de las interrupciones por software INT 6 e INT 7.

```
ORG 1000H
    MSJ DB "INGRESE UN NUMERO:"
FIN DB ?

ORG 1500H
    NUM DB ?

ORG 2000H
    MOV BX,OFFSET MSJ
    MOV AL,OFFSET FIN-OFFSET MSJ
    INT 7
    MOV BX,OFFSET NUM
    INT 6
    MOV AL,1
    INT 7
    MOV CL,NUM
    INT 0
END
```

Responder brevemente:

a) **Con referencia a la interrupción INT 7, ¿qué se almacena en los registros BX y AL**? En el registro BX se almacena la posición inicial en memoria de la variable MSJ, en el registro AL se almacena el total de posiciones ocupadas por la variable MSJ.

b) Con referencia a la interrupción INT 6, ¿qué se almacena en BX? La interrupción INT 6 almacena el valor ingresado por teclado en la posición de memoria establecida por BX.

c) En el programa anterior, ¿qué hace la segunda interrupción INT 7? ¿qué queda almacenado en el registro CL? La interrupción INT 7 muestra en pantalla el valor almacenado en la referencia de memoria del registro BX, en este caso el ultimo valor ingresado por teclado, en el registro CL queda almacenado el valor que fue ingresado por teclado.

d) ¿Cuáles son los códigos de los siguientes caracteres: A, Z, a, z, 0, 1, 9?

A = 1E 41	**Z= 2C 5C**	**a= 1E 61**	**z= 2C 7A**	**0= 52 00**	**1= 4F 00**	**9= 49 00**

3. Escribir un programa que efectúe la suma de dos números ingresados por teclado y muestre el resultado en la pantalla de comandos. Tener en cuenta que el código de los caracteres ingresados no coincide con el número que representan.

```
ORG 1000H; posición en memoria para variables
     MSJ1 DB "INGRESE PRIMER NUMERO:"
     FIN1 DB ?
     MSJ2 DB "INGRESE SEGUNDO NUMERO:"
     FIN2 DB ?
     NUM1 DB ?
     NUM2 DB ?

ORG 2000H; posición en memoria para el programa principal
     MOV BX, OFFSET MSJ1; establecemos en el registro BX la posición en memoria de
     la variable MSJ1
     MOV AL, OFFSET FIN1-OFFSET MSJ1; establecemos en el registro AL la cantidad de
     posiciones en memoria que ocupa la variable MSJ1
     INT 7; mostramos en pantalla el contenido de la variable MSJ1
     MOV BX, OFFSET NUM1; establecemos en BX la posición de memoria de la variable
     NUM1
     INT 6; interrupción para introducción por teclado, se almacena en la posición
     establecida por el registro BX
     MOV BX, OFFSET MSJ2; establecemos en el registro BX la posición en memoria de
     la variable MSJ2
     MOV AL, OFFSET FIN2-OFFSET MSJ2; establecemos en el registro AL la cantidad de
     posiciones en memoria que ocupa la variable MSJ2
     INT 7; mostramos en pantalla el contenido de la variable MSJ2
     MOV BX, OFFSET NUM2; establecemos en BX la posición de memoria de la variable
     NUM2
     INT 6; interrupción para introducción por teclado, se almacena en la posición
     establecida por el registro BX
     ADD NUM1, NUM2; sumamos los valores de las variables NUM1 y NUM2 y lo almacenamos
     en la variable NUM1
     MOV BX, OFFSET NUM1; establecemos en el registro BX la posición en memoria de
     la variable NUM1
     MOV AL,1; establecemos en el registro AL la cantidad de posiciones en memoria
     que ocupa la suma en el proceso anterior
     INT 7; mostramos en pantalla el contenido de la variable NUM1
     INT 0; Finalizamos el programa
END
```

4. Interrupción por hardware: tecla F10: Escribir un programa que, mientras ejecuta un lazo infinito, cuente el número de veces que se presiona la tecla F10 y acumule este valor en el registro DX

```
PIC EQU 20H
EOI EQU 20H
N_F10 EQU 10

ORG 40
    IP_F10 DW RUT_F10

ORG 2000H
    CLI
    MOV AL,0FEH
    OUT PIC+1,AL ; PIC: registro IMR
    MOV AL,N_F10
    OUT PIC+4,AL ; PIC: registro INT0
    MOV DX,0
    STI
    LAZO: JMP LAZO

ORG 3000H
    RUT_F10: PUSH AX
    INC DX
    MOV AL,EOI
    OUT EOI,AL ; PIC: registro EOI
    POP AX
    IRET
END
```

Explicar detalladamente:

a) La función de los registros del PIC: ISR, IRR, IMR, INT0-INT7, EOI. Indicar la dirección de cada uno

- **ISR**: Registro de Interrupción en Servicio. Indica cual es la interrupción que está siendo atendida, mediante la puesta a 1 del bit asociado a esa entrada de interrupción (bit 0 se asocia a la entrada INT0 ... bit 7 se asocia a la entrada INT7).
- **IRR**: Registro de petición de interrupción. Almacena las interrupciones demandadas hasta el momento. Así, al activarse una entrada de interrupción el bit correspondiente se pone a 1, tornándose a 0 cuando ésta pasa a ser atendida (bits 0…7 se asocian a las entradas INT0...INT7, respectivamente).
- **IMR**: Registro de Máscara de Interrupciones. Permite el enmascaramiento selectivo de cada una de las entradas de interrupción mediante la puesta a 1 de su bit asociado (bit 0 se asocia a INT0 ... bit 7 se asocia a INT7). Tras un reset (los bits de este registro quedarán establecidos a cero) todas las entradas de interrupción quedarán desenmascaradas.
- **INT0, ..., INT7**: Cada uno de estos registros contiene el valor del vector de interrupción correspondiente a la entrada del mismo nombre.
- **EOI**: Registro de comandos de sólo escritura, donde la CPU deberá mandar el comando de final de interrupción (EOI), representado por el valor 20H, al final de la rutina de tratamiento de la interrupción. El efecto de está escritura es la puesta a cero del bit del registro ISR correspondiente a la interrupción.

A3	A2	A1	A0	Registro
0	0	0	0	EOI
0	0	0	1	IMR
0	0	1	0	IRR
0	0	1	1	ISR
0	1	0	0	INT0
0	1	0	1	INT1
0	1	1	0	INT2
0	1	1	1	INT3
1	0	0	0	INT4
1	0	0	1	INT5
1	0	1	0	INT6
1	0	1	1	INT7

b) **¿Cuáles de estos registros son programables y cómo trabaja la instrucción OUT?** El registro que es programable es el **IMR** o Registro de Mascara de Interrupciones. **OUT**: se conecta a la entrada de interrupción INT1 del PIC, por lo que provocará una interrupción al activarse cuando los valores de los dos registros internos del dispositivo coincidan.

c) **Qué hacen y para qué se usan las instrucciones CLI y STI: Instrucciones de gestión de las interrupciones**: **CLI:** pone en cero la bandera de interrupciones, deshabilitando así aquellas interrupciones enmascarables. Una interrupción enmascarable es aquella cuyas funciones son desactivadas cuando IF = 0. **STI:** La instrucción activa la bandera IF, esto habilita las interrupciones externas enmascarables (las que funcionan únicamente cuando IF = 1).

5. Interrupción por hardware: TIMER. Implementar a través de un programa un reloj segundero que muestre en pantalla los segundos transcurridos (00-59 seg) desde el inicio de la ejecución.

```
TIMER EQU 10H
PIC EQU 20H
EOI EQU 20H
N_CLK EQU 10

ORG 40
    IP_CLK DW RUT_CLK

ORG 1000H
```

```
    SEG DB 30H
    DB 30H
    FIN DB ?

ORG 3000H
    RUT_CLK: PUSH AX
    INC SEG+1
    CMP SEG+1,3AH
    JNZ RESET
    MOV SEG+1,30H
    INC SEG
    CMP SEG,36H
    JNZ RESET
    MOV SEG,30H
    RESET: INT 7
    MOV AL,0
    OUT TIMER,AL
    MOV AL,EOI
    OUT PIC,AL
    POP AX
    IRET

ORG 2000H
    CLI
    MOV AL,0FDH
    OUT PIC+1,AL ; PIC: registro IMR
    MOV AL,N_CLK
    OUT PIC+5,AL ; PIC: registro INT1
    MOV AL,1
    OUT TIMER+1,AL ; TIMER: registro COMP
    MOV AL,0
    OUT TIMER,AL ; TIMER: registro CONT
    MOV BX,OFFSET SEG
    MOV AL,OFFSET FIN-OFFSET SEG
    STI
    LAZO: JMP LAZO
END
```

Explicar detalladamente:

a) **Cómo funciona el TIMER y cuándo emite una interrupción a la CPU:** Contador de eventos, que realiza una cuenta ascendente los pulsos de la señal aplicada a su entrada INT, restaurándose el valor inicial de cuenta al final de la misma. Posee dos registros internos de ocho bits: COMP y CONT. Cuando el timer expira, genera una interrupción causando que el CPU ejecute la rutina de servicio de la interrupción timer.

b) La función que cumplen sus registros, la dirección de cada uno.
   - **COMP:** Registro de comparación, que determina el módulo de la cuenta del TIMER.
   - **CONT:** Registro contador, que muestra la cuenta de los pulsos de la señal aplicada a la entrada INT del periférico. La coincidencia de su valor con el del registro anterior provoca la activación de la salida OUT.

Dirección de Registros

A0	Registro
0	CONT
1	COMP

Para lograr un acceso, se debe activar también la señal CS del TIMER El tipo de acceso (escritura o lectura) lo determina el estado de las señales IOW e IOR.

Guía N° 7: Entrada y Salida DMA

1. Transferencia de datos memoria-memoria. Escribir un programa que copie una cadena de caracteres almacenada a partir de la dirección 1000H en otra parte de la memoria, utilizando el CDMA en modo de transferencia por bloque. La cadena original se debe mostrar en la pantalla de comandos antes de la transferencia. Una vez finalizada, se debe visualizar en la pantalla la cadena copiada para verificar el resultado de la operación. Ejecutar el programa en la configuración P1 C3.

```
PIC EQU 20H
DMA EQU 50H
N_DMA EQU 20

ORG 80
    IP_DMA DW RUT_DMA

ORG 1000H
    MSJ DB "FACENA"
    FIN DB ?
    NCHAR DB ?

ORG 1500H
    COPIA DB ?

ORG 3000H ; rutina de interrupción del CDMA
    RUT_DMA:MOV AL,0FFH
    OUT PIC+1,AL ; inhabilita las interrupciones del PIC
    MOV BX,OFFSET COPIA
    MOV AL,NCHAR
    INT 7 ; muestra en pantalla la cadena copiada
    MOV AL,20H
    OUT PIC,AL ; EOI
    IRET

ORG 2000H
    CLI
    MOV AL,N_DMA
    OUT PIC+7,AL ; carga en el registro INT3 del PIC el tipo de int.
    MOV AX,OFFSET MSJ
    OUT DMA,AL
    MOV AL,AH
    OUT DMA+1,AL ; carga en el CDMA la dirección de comienzo del; bloque de datos a
    transferir
    MOV AX,OFFSET FIN-OFFSET MSJ
    OUT DMA+2,AL
    MOV AL,AH
    OUT DMA+3,AL ; carga en el CDMA la cantidad de bytes a transferir
    MOV AX,OFFSET COPIA
    OUT DMA+4,AL
    MOV AL,AH
    OUT DMA+5,AL ; carga en el CDMA la dirección de destino del bloque de datos
    MOV AL,0AH
    MOV AL,0F7H
    OUT PIC+1,AL ; habilita INT3
    STI
    MOV BX,OFFSET MSJ
    MOV AL,OFFSET FIN-OFFSET MSJ
    MOV NCHAR,AL
    INT 7 ;muestra en pantalla el mensaje original
    MOV AL,7H
    OUT DMA+7,AL ;arranque de la transferencia
    INT 0
    ENDOUT DMA+6,AL ; configura el CDMA para transf. mem-mem por bloque
```

Cuestionario:

a) **Controlador de DMA (CDMA) – Acceso Directo a Memoria**: Ofrece la posibilidad de realizar transferencias de datos de 8 bits memoria-memoria o memoria-periférico y a la inversa, sin intervención directa de la CPU y en robo de ciclo (la CPU le ha de ceder los buses con el protocolo HRQ/HLDA), siendo el tamaño máximo del bloque a transferir de 64Kbytes. Posee un único canal de DMA, pudiendo realizar transferencias en modo bloque (una vez iniciada la transferencia, transmite datos sin parar hasta que se agota el bloque a transferir), o bajo demanda del periférico al que se encuentre conectado.

b) **Describir el significado de los bits del registro CTRL:** Registro de control. El significado de sus bits varía en función de la operación (lectura o escritura) que sobre él se realice. Así:

ESCRITURA	**LECTURA**
C0: **0:** No tiene sentido **1:** Al escribirlo la CPU detiene momentáneamente la transferencia en curso.	**C0:** **0:** Transferencia en curso. **1:** Transferencia detenida por la CPU temporal-mente.
C1 (TT=Tipo Transferencia): **0:** Transferencia Periférico -Memoria, o a la inversa. **1:** Transferencia Memoria-Memoria.	**C7(TC = Terminal Count):** **0:** Transferencia no finalizada. **1:** Transferencia ya finalizada.
C2 (ST = Sentido Transferencia): Sólo tiene sentido si C1=0. Entonces: **0:** Sentido transferencia Periférico-Memoria. **1:** Sentido transferencia Memoria-Periférico.	
C3 (MT = Modo Transferencia): **0:** Transferencia por demanda. **1:** Transferencia en modo bloque.	
C4...C7: No usados. No importa su contenido.	

c) **¿Qué diferencia hay entre transferencia de datos por bloque y bajo demanda?**

- **Transferencia de bloque:** Se transfieren todas las palabras de las que consta la transferencia. Se transmite datos sin parar hasta que se agota el bloque a transferir.
- **Transferencia bajo demanda:** la transferencia se realiza sólo mientras el dispositivo siga solicitando el canal de DMA. Esta modalidad permite dejar ciclos a la CPU cuando no es realmente necesario que el DMA opere.

d) **¿Cómo se le indica al CDMA desde el programa que debe arrancar la transferencia de datos**?: Una vez cargados los valores adecuados en los registros, para lograr el arranque de la transferencia programada, es necesario forzar los bits A3, A2 y A1 del controlador a 1 a la vez que se activa CS, sin importar el valor de las líneas IOR e IOW, acción que, además pondrá a 0 el bit C0. Si se desea detener momentáneamente la transferencia, será necesario poner C0 a 1, pero su rearranque no se logrará poniendo C0 a 0, sino a través de la condición CS=A0=A1=A2.

e) **¿Qué le indica el CDMA a la CPU a través de la línea HRQ? ¿Qué significa la respuesta que le envía la CPU a través de la línea HLDA?:** El CDMA, a través de la línea HRQ le solicita a la CPU la petición de Bus**.** La CPU a través de la línea HLDA habilita al CDMA la petición de Bus.

f) **Registros RF, CONT y RD:**

- **RF**: Registro de direcciones Fuente. En transferencias memoria-periférico o a la inversa, se carga en él la dirección del bloque de memoria a transferir o recibir. En transferencias memoria-memoria actúa tal como su nombre indica.

- **RD**: Registro de direcciones Destino. Sólo tiene sentido su utilización en transferencias memoria-memoria, actuando tal como su nombre indica.
- **CONT**: Registro Contador. Indica el número de octetos a transferir.

g) **¿Cómo se configura el PIC para atender la interrupción del CDMA?:** El PIC se configura en modo 3 para atender las interrupciones del CDMA

2. Transferencia de datos memoria-periférico. Escribir un programa que transfiera datos desde la memoria hacia la impresora sin intervención de la CPU, utilizando el CDMA en modo de transferencia bajo demanda.

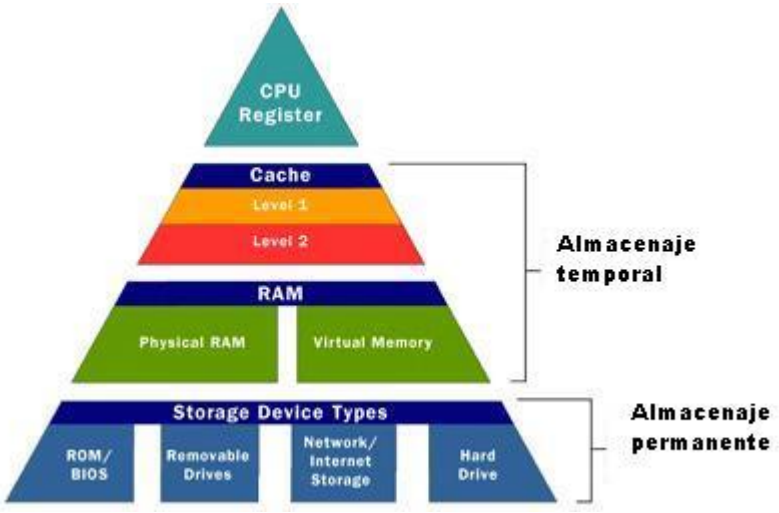
```
PIC EQU 20H
HAND EQU 40H
DMA EQU 50H
N_DMA EQU 20

ORG 80
    IP_DMA DW RUT_DMA

ORG 1000H
    MSJ DB "FACENA"
    FIN DB ?
    FLAG DB 0

ORG 3000H ; rutina de interrupción del CDMA
    RUT_DMA:MOV AL,0
    OUT HAND+1,AL ; inhabilita interrupción del HAND
    MOV FLAG,1
    MOV AL,0FFH
    OUT PIC+1,AL ; inhabilita las interrupciones del PIC
    MOV AL,20H
    OUT PIC,AL ; EOI
    IRET

ORG 2000H
    CLI
    MOV AL,N_DMA
    OUT PIC+7,AL ; carga en el registro INT3 del PIC el tipo de interrupción
    MOV AX,OFFSET MSJ
    OUT DMA,AL
    MOV AL,AH
    OUT DMA+1,AL ; carga en el CDMA la dirección de comienzo del bloque de datos a
    transferir
    MOV AX,OFFSET FIN-OFFSET MSJ
    OUT DMA+2,AL
    MOV AL,AH
    OUT DMA+3,AL ; carga en el CDMA la cantidad de bytes a transferir
    MOV AL,4H
    OUT DMA+6,AL ; configura el CDMA para transferencia mem-perif bajo demanda
    MOV AL,0F7H
    OUT PIC+1,AL ; habilita INT3
    OUT DMA+7,AL ;arranque de la transferencia
    MOV AL,80H
    OUT HAND+1,AL ; habilita interrupción del HAND
    STI
    LAZO: CMP FLAG,1
    JNZ LAZO
    INT 0
END
```

Memoria

Cuestionario

1. Grafique y explique la Jerarquía de Memorias.

Niveles jerárquicos de Memorias



Nivel 0: Registros del microprocesador o CPU
Nivel 1: Memoria caché
Nivel 2: Memoria primaria (RAM)
Nivel 3: Disco duro (con el mecanismo de memoria virtual), Cintas magnéticas (consideradas las más lentas, con mayor capacidad, de acceso secuencial), Redes (actualmente se considera un nivel más de la jerarquía de memorias), Discos Externos, Almacenamiento distribuido.

La jerarquía de memoria es la organización piramidal de la memoria en niveles que tienen las computadoras. Tiene como objetivo conseguir el rendimiento de una memoria de gran velocidad al coste de una memoria de baja velocidad, basándose en el principio de cercanía de referencias [1].

Los puntos esenciales relacionados con la memoria pueden resumirse en:

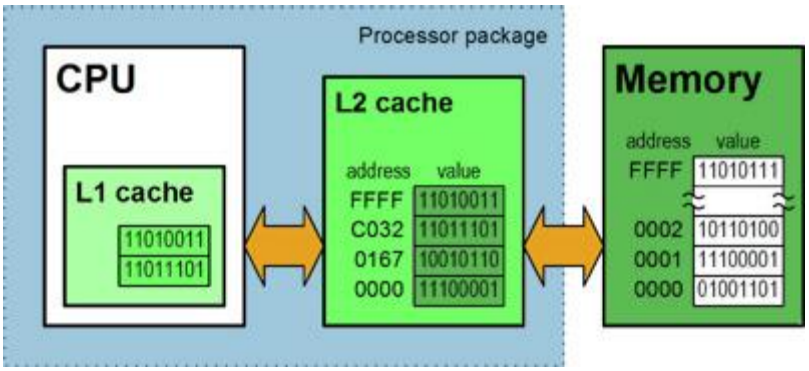
- **Capacidad**: cuanta más memoria haya disponible, más podrá utilizarse.
- **Velocidad**: la velocidad óptima para la memoria es la velocidad a la que el microprocesador puede trabajar, de modo que no haya tiempos de espera entre calculo y calculo, utilizados para traer operandos o guardar resultados.
- **Costo por bit**: el costo de la memoria no debe ser excesivo, para que sea factible construir un equipo accesible.

Los tres factores compiten entre sí, por lo que hay que encontrar un equilibrio, teniendo en cuenta que, a menor tiempo de acceso mayor coste, mayor capacidad menor coste por bit y a mayor capacidad menor velocidad, entonces, se busca contar con capacidad suficiente de memoria, con una velocidad que sirva para satisfacer la demanda de rendimiento y con un coste que no sea excesivo. Gracias a un principio llamado cercanía de referencias [1], es factible utilizar una mezcla de los distintos tipos y lograr un rendimiento cercano al de la memoria más rápida.

[1]: Cercanía de Referencias: hace referencia al agrupamiento de las lecturas de memoria por medio de la unidad central de procesamiento. Las mismas, ya sean para instrucciones o para leer datos, se mantienen por lo general dentro de grupos de direcciones relativamente cercanas entre sí, esto es fundamental en la utilización de diferentes tecnologías de la jerarquía de memoria para lograr un desempeño favorable. La cercanía de referencias también implica que para que un programa ejecute no son necesarias todas sus páginas cargadas en la memoria principal. Si en un período corto de tiempo el procesador referencia direcciones cercanas de memoria, un proceso puede ejecutar varias instrucciones con sólo algunas

partes de su código en memoria principal. Esto permite tener más procesos en memoria listos para correr, como así también evita tener que cargar y descargar sus páginas en el momento de un intercambio, perdiendo ciclos del procesador.

2. Grafique y explique los principios básicos de la memoria cache.



La Memoria Caché es la solución al problema de rendimiento del sistema de memoria. Es muy pequeña y está incluida normalmente en el interior del procesador haciendo de intermediario entre la memoria principal y los distintos componentes de la Unidad Central de Proceso. Su función es sencilla, conseguir que los datos más usados estén lo más cerca del procesador para ser accedidos de la manera más rápida posible.
Se organiza en niveles, de menor a mayor tamaño según lo alejada que esté del micro. Si el procesador necesita un dato de la memoria se comprueba si este se encuentra en el primer nivel. En caso de no encontrarlo, se busca en el segundo nivel y si no en el tercero. Todo se acelera si se colocan los datos más utilizados en los niveles más cercanos al procesador.
Cada uno de estos niveles tiene un bloque de control el cual se encarga de almacenar y poner los datos a disposición del procesador. El tiempo que tarda en buscar la información es proporcional al tamaño de la propia memoria que administra. Como queremos que los datos lleguen lo antes posible al micro los niveles más bajos tendrán menor capacidad. Cada nivel superior, por tanto, es bastante más grande que el anterior.
La memoria cache es muy pequeña. En comparación con la memoria RAM unas mil veces más pequeña. Por suerte, los programas suelen realizar muchas operaciones sobre los mismos datos y por lo tanto se consiguen grandes mejoras al usar esta técnica.

3. ¿Qué es una memoria de tipo “RAM”? ¿Qué significa? ¿Cuáles son sus características principales?

La memoria de acceso aleatorio (Random Access Memory, RAM) se utiliza como memoria principal de las computadoras para el sistema operativo, los programas y la mayor parte del software, en ella se pueden efectuar operaciones de lectura y escritura. En la RAM se cargan todas las instrucciones que ejecuta la unidad central de procesamiento (procesador) y otras unidades del computador. Se denominan «de acceso aleatorio» porque se puede leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición, no siendo necesario seguir un orden para acceder (acceso secuencial) a la información de la manera más rápida posible. Es de tipo volátil lo que significa que una vez que deje de recibir estímulos eléctricos, todos los datos almacenados se pierden.

4. Explique las dos variantes de tecnología RAM (DRAM Y SRAM). ¿Cuáles son sus diferencias?

Ambas tecnologías tienen el mismo propósito, proveer de acceso aleatorio a posiciones de memoria, ambas son de tipo volátil, al perder el suministro de energía se pierden los datos almacenados en memoria. La SRAM tiene como ventaja velocidad y consumo de energía, la DRAM puede almacenar mayores cantidades de datos en un mismo espacio físico en comparación con la SRAM y es más económica para su fabricación, por tal motivo se opta por usar memorias DRAM como memoria principal de un computador.

- La RAM estática o Static RAM (SRAM por sus siglas en inglés) recibe su nombre del hecho de que una vez que los datos se almacenan, se mantendrán siempre y cuando el módulo sea alimentado con electricidad. Una vez escrito, el controlador de memoria puede olvidar los datos hasta que necesite recuperarlos, permitiéndole ser más eficiente.

- La RAM dinámica o Dynamic RAM (DRAM por sus siglas en inglés) almacena los datos mediante un transistor y un condensador emparejado para cada bit de datos. Los condensadores pierden constantemente la electricidad, lo que requiere que el controlador de memoria actualice la DRAM varias veces por segundo para mantener los datos. Ya que la DRAM sólo requiere un transistor por bit de datos, los chips de DRAM son mucho más densos y pueden almacenar más datos que las SRAM en un paquete del mismo tamaño.

5. Explique mecanismo de lectura/escritura de un disco magnético.

   Los datos que se almacenan en los discos magnéticos son almacenados en bits, ceros y unos, los cuales son guardados en diminutas partículas magnetizadas. Las superficies de los platos de un disco magnético están divididas en microscópicas regiones denominadas dominios magnéticos; los cuales están compuestos por pequeños granos o cristales de material magnético de aproximadamente 10 nm (nanómetros) de tamaño. Cada cristal está compuesto por algunos cientos de átomos. Al magnetizarse estas partículas sus átomos apuntan hacia un lado o el otro formando un dipolo magnético el cual genera un campo magnético a su alrededor; según hacia la dirección que éstos apunten tendremos representado un 1 o un 0. Cada cristal o partícula es un dominio magnético. Al pasar el cabezal de escritura por encima del plato con las partículas magnéticas, las magnetiza haciendo que éstas sean alineadas hacia un lado o el otro; por ejemplo, si el dato a guardar es una letra "A" ésta se representa mediante el código binario de ocho dígitos 01000001; por lo tanto, el cabezal hará que dos de los dominios magnéticos apunten hacia un lado representando los 1 y seis dominios magnéticos queden alineados hacia la dirección opuesta representando los 0. De esta manera se almacenan los datos que guardamos en el disco rígido.

6. Explique cómo se organiza los datos en un disco magnético.

   Antes de que el computador pueda usar un disco magnético para almacenar datos, este debe ser mapeado en forma magnética de modo que la computadora pueda ir directo a un punto específico sin buscar a lo largo de todos los datos. El proceso de mapeo se llama formateo o inicialización. En la actualidad, muchos discos magnéticos vienen pre-formateados. Lo primero que hace una unidad de disco cuando formatea un disco es crear un conjunto de círculos concentración magnéticos llamados pistas. El número de pistas en un disco varía con el tipo. Las pistas en un disco no forman una espiral continua como las de una grabación fonográfica; más bien, cada uno es un circulo separado, como los círculos en una diana. Las pistas son numeradas desde el círculo más exterior hasta el más interior, empezando desde cero. Cada pista en un disco se divide también en partes más pequeñas llamadas sectores. Todos los sectores en el disco son numerados en una secuencia larga de modo que la computadora puede tener acceso a cada área pequeña en el disco con un número único. Este esquema simplifica de forma muy efectiva lo que sería un conjunto de coordenadas bidimensionales en una sola dirección numérica. Sin importar el tamaño físico, todos los sectores contienen el mismo número de bytes; es decir, los sectores más cortos y más internos contienen la misma cantidad de datos que los sectores más largos y más externos. En el formateo se puede establecer el tamaño en bytes de los sectores, siendo el más común 512 bytes. Un sector es la unidad más pequeña con la que puede trabajar cualquier unidad de disco- Cada bit y byte dentro de un sector puede tener valores diferentes, pero la unidad puede leer o escribir sólo sectores enteros a la vez. Si la computadora necesita cambiar sólo un byte de 512, debe reescribir el sector entero.

Memoria Externa - Capacidad de Discos magnéticos

1. Un disco magnético con 5 platos (dos caras cada plato), tiene 2048 pistas/plato, 1024 sectores/pistas y 512 bytes por sector. ¿Cuál es la capacidad total?

   - 10(5 platos) X 2048(pistas) X 1024(sectores) X 512(bytes) = 10.737.418.240Bytes -> 10.485.760KB -> 10.240MB -> **10GB**

2. Un disco magnético con 4 platos (dos caras cada plato), 36000 pistas/plato, 580 sectores/pista y 256 bytes por sector. ¿Cuál es la capacidad total?

- 8(4 platos) X 36000(pistas) X 580(sectores) X 256(bytes) = 42.762.240.000Bytes -> 41.760.000KB -> 40.7MB -> **39,8GB**

3. Un disco magnético tiene una capacidad total de 15 Gb, 4 platos (dos caras por plato), 56000 pistas/plato y 512 bytes por sector. ¿Cuantos sectores/pistas tiene el disco?

   15GB -> 15.360MB -> 15.728.640KB -> 16.106.127.360Bytes
   8(4platos) X 56.000(pistas) = 448.000Pistas
   16.106.127.360Bytes % 448.000Pistas % 512Bytes = 70,21 -> **71 Sectores/Pistas**

   Comprobación: 8(4 platos) X 56.000(pistas) X 71(sectores) X 512(bytes) = 16.285.696.000Bytes -> 15.904.000KB -> 15.531,25MB -> **15,16GB**

4. Una unidad de disco tiene 16 sectores por pista de 1024 bytes cada uno. El disco gira a 3600 rpm y tiene un tiempo medio de búsqueda de 25 ms. Calcular el tiempo que se necesita para transferir 25 sectores dispuestos de forma contigua.

   - Los 25 sectores ocupan en total 2 pistas, una pista completa con 16 sectores y otra con 9 sectores

Tabla de Referencias		
TMB = Tiempo de Búsqueda Medio	**TRR** = Tiempo de Retardo Rotacional	**TT** = Tiempo de transferencia
F = Velocidad Rotacional	**B** = Byte	**P** = Bytes por Pista

Tiempo de posicionamiento del cabezal sobre el sector	**Tiempo de transferencia para la primer pista**
TRR= $\frac{1}{2\times f}$ = $\frac{1\times 1000}{2\times\frac{3600}{60}}$ = **8,3 ms**	**TT**= $\frac{b}{P\times f}$ = $\frac{16\times 1024\times 1000ms}{16\times 1024\times\frac{3600}{60}}$ = **16,66ms**
Tiempo total para la primer pista	
$T_{1° pista}$ = TMB + TRR + TT = 25ms + 8,3ms + 16,66ms = **50ms**	

Lectura de la segunda pista			**Tiempo de transferencia para la primer pista**
TMB = 0	**TRR= 8,3ms**	**9 sectores**	**TT**= $\frac{b}{P\times f}$ = $\frac{9\times 1024\times 1000ms}{16\times 1024\times\frac{3600}{60}}$ = 9,375ms
Tiempo total para la segunda pista			
$T_{2° pista}$ = TMB + TRR + TT = 0ms + 8,3ms + 9,375ms = **17,705ms**			

Tiempo total
$T_{25 sectores}$ = $T_{1° pista}$ + $T_{2° pista}$ = **67,705ms**

5. Repetir el problema anterior cuando los 25 sectores están dispuestos de forma aleatoria sobre la superficie del disco.

Tiempo de posicionamiento del cabezal sobre el sector	**Tiempo de transferencia por sector**
TRR= $\frac{1}{2\times f}$ = $\frac{1\times 1000}{2\times\frac{3600}{60}}$ = **8,3 ms**	**TT**= $\frac{b}{P\times f}$ = $\frac{1\times 1024\times 1000ms}{16\times 1024\times\frac{3600}{60}}$ = **1042ms**
Tiempo total para un sector	
$T_{un\ sector}$= TMB + TRR + TT = 25ms + 8,3ms + 1042ms = **34,372ms**	

Tiempo total
$T_{25\ sectores}$ = T_{un} sector X 25 = **859,3ms**

6. Considérese un disco con un tiempo de búsqueda medio especificado de 4 ms, una velocidad de rotación de 15000 rpm, y sectores de 512 bytes con 500 sectores por pistas. Supóngase que queremos leer un fichero que consta de 2500 sectores. Queremos estimar el tiempo total de transferencia. Los ficheros están organizados secuencialmente.

Tiempo de posicionamiento del cabezal sobre el sector	**Tiempo de transferencia para la primer pista**
TRR= $\frac{1}{2\times f}$ = $\frac{1\times 1000}{2\times\frac{15000}{60}}$ = **2 ms**	**TT**= $\frac{b}{P\times f}$ = $\frac{500\times 512\times 1000ms}{500\times 512\times\frac{15000}{60}}$ = **4ms**
Tiempo total para una pista	
$T_{x\ pista}$ = TRR + TT = 2ms + 4ms = **6ms**	

Tiempo total
$T_{5\ pistas}$ = TMB + ($T_{x\ pista}$ X 5) = **34ms**

7. Repetir el problema anterior cuando los 2500 sectores están dispuestos de forma aleatoria sobre la superficie del disco.

Tiempo de posicionamiento del cabezal sobre el sector	**Tiempo de transferencia por sector**
TRR= $\frac{1}{2\times f}$ = $\frac{1\times 1000}{2\times\frac{15000}{60}}$ = **2 ms**	**TT**= $\frac{b}{P\times f}$ = $\frac{1\times 512\times 1000ms}{500\times 512\times\frac{15000}{60}}$ = **0,008ms**
Tiempo total por sector	
$T_{x\ sector}$ = TMB + TRR + TT = 4ms + 2ms + 0,008ms = **6,008ms**	

Tiempo total

T = $T_{x\ sector}$ X 2500 = **15020ms -> 15,02s**