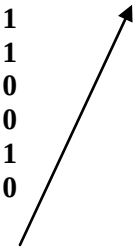
EJERCICIOS RESUELTOS

Realizar la siguiente suma y expresar el resultado en hexadecimal:

$83/_d + 331/_4 + 11111101/_b$

El primer paso consiste en expresar todos lo valores con la misma base. Para eso convertiremos los dos primeros valores a binario.

83	**1**
41	**1**
20	**0**
10	**0**
5	**1**
2	**0**
1	



Realizamos divisiones sucesivas del valor ($83/_d$) por la base del sistema en el que lo queremos expresar (2)
Tomando el último resultado y los restos en orden invertido obtenemos: $1010011/_2$ que es equivalente en binarios de $83/_d$

3		3		1	
1	1	1	1	0	1

Para convertir de base 4 a base 2 expresamos cada digito del valor (base 4) con un binario de 2 dígitos

Teniendo los 3 valores expresados en la misma base, podemos realizar la suma:

			1						Acarreos 2
1	0	1	0	1	1	1	1		Acarreos 1
		1	0	1	0	0	1	1	1º valor
			1	1	1	1	0	1	2º valor
		1	1	1	1	1	0	1	3º valor
1	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**RESULTADO**
i	h	g	f	e	d	c	b	a	

Columna a: La suma de 1 + 1 + 1 da como resultado 3, que en binarios se escribe 11. El 1 de la derecha se coloca en la fila de Resultados y el de la izquierda en Acarreos 1 columna b
Columna b: 1 + 1 + 0 + 0 (el primer 1 es el acarreo de la columna anterior) = 2 (10 en binarios) 0 en Resultado y 1 en Acarreos 1 columna c
Columna c: 1 + 0 + 1 + 1 = 3 (en binarios 11). 1 a Resultados, 1 a Acarreos 1 columna d
Columna d: 1 + 0 + 1 + 1 = 3 (en binarios 11). 1 a Resultados, 1 a Acarreos 1 columna e
Columna e: 1 +1 + 1 + 1 = 4 (en binarios 100) 0 a Resultados, 0 a Acarreos1 columna f, 1 a Acarreos 1 columna g
Columna f: 0 +0 + 1 + 1 = 2 (en binarios 10) 0 a Resultados, 1 a Acarreos 2 columna g
Columna g: 1 +1 + 1 + 1 = 4 (en binarios 100) 0 a Resultados, 0 a Acarreos 1 columna h, 1 a Acarreos1 columna i
Columna h: 0 a Resultados
Columna i: 1 a Resultados

Para finalizar el ejercicio agrupamos los dígitos del binario de a 4 comenzando desde la derecha, colocando (si fuera necesario) ceros a la izquierda hasta completar el último grupo. Expresamos cada uno de estos grupos con el dígito Hexadecimal que corresponde.

0	0	0	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
			1				**0**				**D**

RESULTADO FINAL: $83/_d + 331/_4 + 11111101/_b = 10D/_h$

EJERCICIOS RESUELTOS

Realizar la siguiente diferencia y expresar el resultado en hexadecimal:

$5FC2/_h$ - $776/_d$

Tal como planteamos en el ejercicio anterior, para poder operar con los valores necesitamos expresarlos con la misma base. En función de las habilidades que necesitamos desarrollar para el desarrollo y comprensión de la representación de datos en una computadora, sugiero que antes de realizar la resta se conviertan ambos valores al sistema binario.

5				**F**				**C**				**2**			
0	1	0	1	1	1	1	1	1	1	0	0	0	0	1	0

776	0
388	0
194	0
97	1
48	0
24	0
12	0
6	0
3	1
1	

Expresados los valores en una misma base, realizamos la resta. Para esto aplicaremos un método que consiste en obtener el complemento a la base (Ca2) del sustraendo y sumarlo al minuendo. Una forma fácil de calcular Ca2 es obtener el Ca1 y a este sumarle 1.
El Ca1 es el resultado de invertir los bits (reemplazar ceros por unos y viceversa**).**
Importante: antes de calcular el Ca1, agregamos ceros a la izquierda el sustraendo para igualar la cantidad de dígitos del minuendo.

	1	0	1	1	1	1	1	1	1	0	0	0	0	1	0
	0	**0**	**0**	**0**	**0**	1	1	0	0	0	0	1	0	0	0
	1	**1**	**1**	**1**	**1**	0	0	1	1	1	1	0	1	1	1
															1
	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0

Minuendo
Sustraendo
Cambio 0 por 1 y 1 por 0 para obtener Ca1 del sustraendo
Sumo 1 al Ca1
← Obtengo Ca2

	1	0	1	1	1	1	1	1	1	0	0	0	0	1	0
	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0
~~1~~	1	0	1	1	1	0	0	1	0	1	1	1	0	1	0
	1	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**

Minuendo
Sumo Ca2 de sustraendo
Trunco el valor acarreado
Resultado de la resta

Al resultado de la diferencia 101 1100 1011 1010. Solo resta convertirlo a hexadecimal como pide el enunciado:

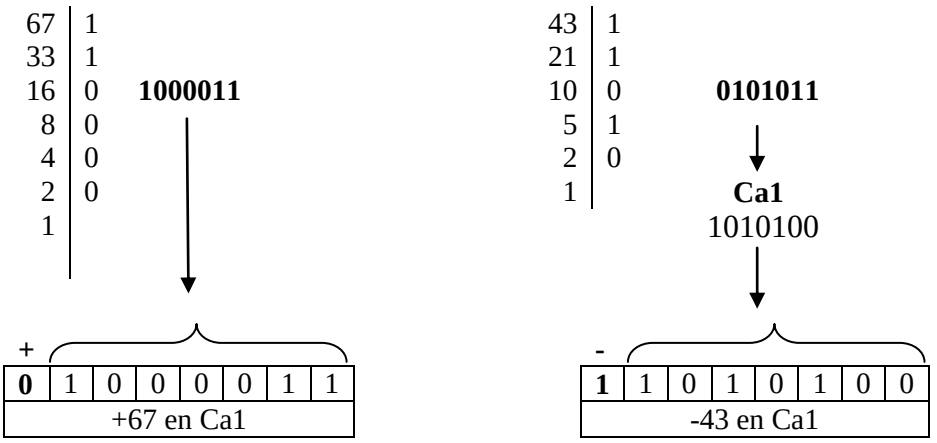
0	1	0	1	1	1	0	0	1	0	1	1	1	0	1	0
5				**C**				**B**				**A**			

RESULTADO FINAL: $5FC2/_h$ - $776/_d$ = $5CBA/_h$

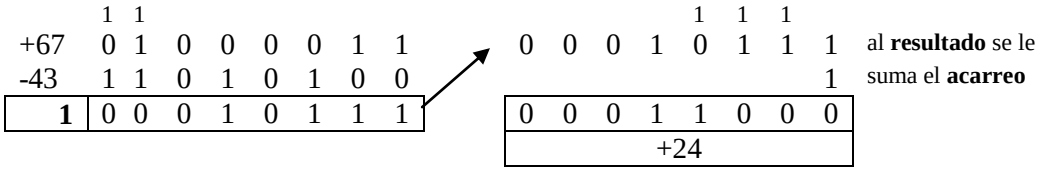
EJERCICIOS RESUELTOS

Sumar los números 67 y -43 y, representándolos en Ca1 en palabras de 8 bits.

Para representar un número en Ca1 se utiliza un bit (en este ejemplo el bit 7) para signo (0 positivo – 1 negativo). El resto de los bits se usa para representar el valor. Si es positivo el binario equivalente, si es negativo el complemento a 1 del valor. El Ca1 de un numero se obtiene reemplazando los ceros por unos y viceversa



Cuando se suman de 2 valores en Ca1, si hay acarreo, este debe sumarse al resultado obtenido.



RESULTADO: La suma de +67 y -43 en Ca1 es 00011000 ($24/_{10}$)

Representar el número -75 en Ca2 en palabra de 8 bits.

Para representar un número en Ca1 se utiliza un bit (en este ejemplo el bit 7) para signo (0 positivo – 1 negativo). El resto de los bits se usa para representar el valor. Si es positivo el binario equivalente, si es negativo el complemento a 2 del valor.

$75/_{10} = 1001011/_2$

Para hallar Ca2 de un número, hallamos su complemento a 1 y al resultado le sumamos 1.

$75/_{10}$ ➔	1	0	0	1	0	1	1
Ca1 ➔	0	1	1	0	1	0	0
+1							1
Ca2 de $75/_{10}$ ➔	0	1	1	0	1	0	1

Agreagmos bit de signo | **1** | **0** | **1** | **1** | **0** | **1** | **0** | **1** |

RESULTADO: El número -75 en Ca2 es 10110101

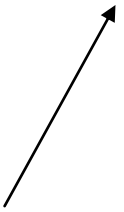
EJERCICIOS RESUELTOS

Representar en exceso a 2^{n-1} en un byte, el número decimal -94.

Para representar valores en exceso, debemos sumarle al numero que queremos representar el exceso. En este caso el exceso indicado es 2^{n-1}, y el dato se almacena en un byte. Sabemos que un byte tiene 8 bits, por lo tanto n=8, o sea que el exceso a utilizar es $2^{8-1} = 2^7 .= 128$.

-94 + 128 = 34

34	**0**
17	**1**
8	**0**
4	**0**
2	**0**
1	



RESULTADO: -94 EN Exceso 2^{n-1} (en un byte)= 00100010

Indicar cuál es la capacidad de representación, la resolución y el rango de un sistema BCS de 7 bits.

Capacidad: (cantidad de valores distintos que se pueden representar) $2^n = 128$
Rango: BCS (Binarios con signo). 1 bit para signo 6 para magnitud. -64 a +64.
Resolución: (mínima distancia entre 2 valores) = 1

Indicar cuál es la capacidad de representación, la resolución y el rango de sistema binario con 5 bits para la parte entera y 2 bits para la parte fraccionaria

Capacidad: $2^n = 2^7 = 128$
Rango: 0 a 31, 75 $(11111{,}11_2)$
Resolución: $0{,}01/_2 = 1 \times 2^{-2} = 0{,}25$

EJERCICIOS RESUELTOS

Representar el número decimal 64852 en IEEE754 de precisión simple.

Este método de representación de coma flotante utiliza la notación científica. Para esto divide los 32 bits con la siguiente estructura 1 bit para signo, 8 para exponente y 23 para mantisa

S	Exponente: 8 bits							Mantisa: 23 bits																						

En este caso, el primer bit es 0 porque el número es positivo

Para obtener exponente y mantisa primero obtenemos el binario equivalente:
$64852/_d = 1111110101010100/_2$

A continuación debemos normalizar el valor: Para esto hay que desplazar el punto decimal (hacia derecha o izquierda, según corresponda) de modo que la parte entera solo tenga 1 dígito:
1111110101010100,
1,111110101010100

MANTISA

En este caso desplazamos 15 lugares hacia la izquierda, por lo tanto nuestro exponente es +15 (Cuando desplazamos hacia la derecha el exponente es negativo).
El exponente se representa en exceso 2^{n-1}-1 en 8 bits : $2^7 - 1 = 127$

$15+127 = 142/_{10} =$ **10001110**

EXPONENTE

Ahora podemos completar nuestra representación: Completamos con ceros

0	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	1	1	1	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
+	Exponente								Mantisa																						
4				7				7				D				5				4				0				0			

Nota: no es necesario almacenar la parte entera del valor normalizado porque siempre es 1

RESULTADO: $64852/_d$ en IEEE754 (p. simple) = $477D5400/_h$

EJERCICIOS RESUELTOS

Que número decimal representa 44C08000 si está expresado en IEEE754 de simple precisión?

4				4				C				0				8				0				0				0			
0	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+	Exponente								Mantisa																							

$10001001/_2 = 137/_{10}$

137 – 127 = 10 (lugares a desplazar, hacia derecha porque el exceso es positivo)
(exceso)

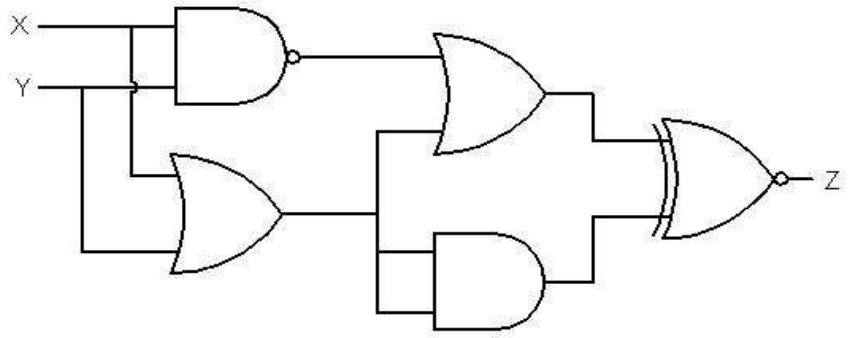
IMPORTANTE: El sentido de desplazamiento se invierte con respecto a la conversión de decimal a IEEE754

Mantisa
P. entera
1,1000000100…
11000000100, 0… ➔ $1x2^{10} + 1 \times 2^9 + 1x2^2 = 1024 + 512 + 4 = 1540$
10 lugares

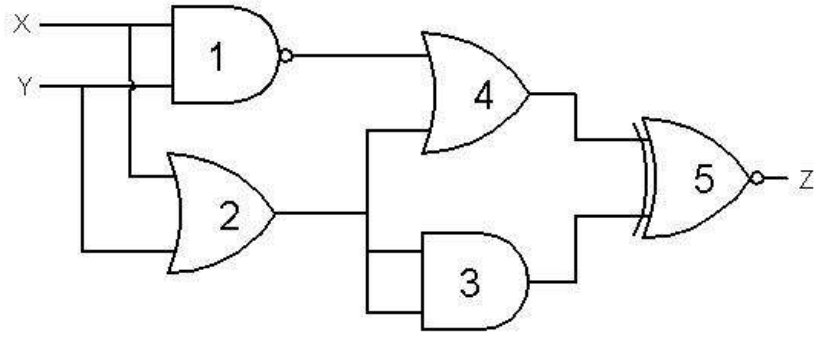
RESULTADO: $44C08000/_h$ en IEEE754 (simple precisión) = $1540/_d$

EJERCICIOS RESUELTOS

Hallar la función lógica para el siguiente circuito e indicar cual es el valor de salida si X=93 , Y= 145 (ambos decimales). Expresar el resultado en Hexadecimal



Para explicar como hallar la función, pondremos números a cada operación para que resulte mas simple.



$$\overline{(\underbrace{(\underbrace{X \cdot Y}_{1})}_{4} + \underbrace{(\underbrace{X + Y}_{2})}_{4}) \oplus \underbrace{(\underbrace{X + Y}_{2} \cdot \underbrace{X + Y}_{2})}_{3}}^{5} = Z$$

La tabla de verdad es un cuadro en el que representamos todas las entradas posibles (diferentes combinaciones de los valores de las variables de entrada, en este ejemplo X, Y) y el valor que toma la variable de salida (Z) para cada una de las entradas.

X	Y	Z
0	0	
0	1	
1	0	
1	1	

El análisis se puede hacer sobre la función o en el circuito. Explicaremos a continuación la resolución por circuito.
Para explicar como se completa la tabla de verdad utilizaremos los mismos números 1, 2, 3, 4, y 5 usados para construir la función, y tomaremos como ejemplo el primer par de valores de entrada: x=0, y=0

EJERCICIOS RESUELTOS

Resolvemos 1: X=0 Y =0. 0 y 0 en la entrada de AND generan en la salida un 0 pero como hay negación en la salida de la operación se convierte en **1.**
Resolvemos 2: 0 y 0 con OR generan un **0**.
Resolvemos 3: ambas entradas son iguales, y son el resultado de la operación 2. En este caso 0 y 0 a la entrada de un AND producen en la salida **0**.
Resolvemos 4: Es una operación OR que tiene como entradas los resultados de 1 (1) y 2 (0). 1 y 0 en un OR da como resultado **1**.
Resolvemos 5: Es una operación XOR cuyas entradas son el resultado de 4 (1) y el resultado de de 3 (0). 1 y 0 en un XOR producen 1, pero como esta negado da **0**.

Para X=0 e Y=0 , Z es igual a 0.

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

Analizando cada combinación de entradas completamos la tabla.
Tenemos la tabla, o sea que sabemos cual es la respuesta del circuito ante cada combinación de entradas posibles. Ahora debemos convertir los valores de entrada a binario y hallar el resultado aplicando para cada par de valores de entrada esta tabla de verdad.

93	1
46	0
23	1
11	1
5	1
2	0
1	

101 1101

145	1
72	0
36	0
18	0
9	1
4	0
2	0
1	

1001 0001

93	**0**	1	0	1	1	1	0	1
145	1	0	0	1	0	0	0	1
Aplicando Tabla	1	1	0	1	1	1	0	1
	D				**D**			

RESPUESTA: $93_{/10}$ y $145_{/10}$ como entradas del circuito producen como salida $DD_{/16}$

EJERCICIOS RESUELTOS

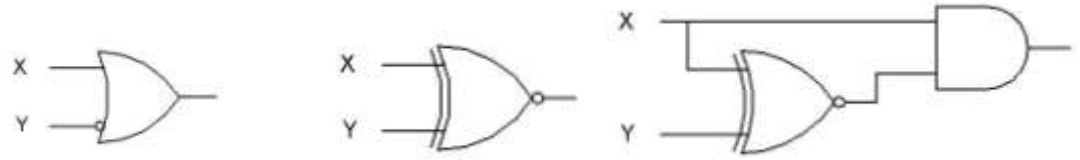
Construir el circuito asociado y la tabla de verdad de la función:

$$\mathbf{Z = (X + \overline{Y}) \,.\, (X \,.\, \overline{(X \oplus Y)})}$$

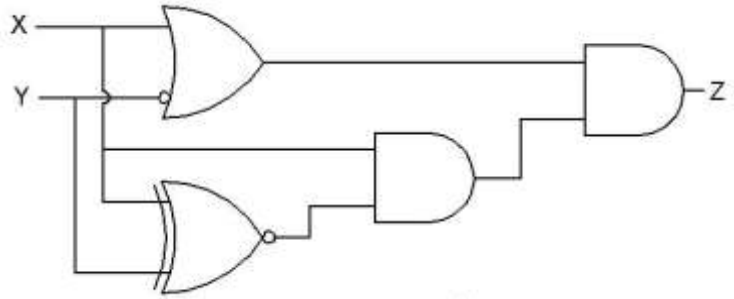
No existe un solo método para obtener el circuito. Plantearemos una estrategia que consiste en "dividir" la función en tramos e ir resolviéndolos individualmente, para finalmente unir los segmentos obtenidos para la construcción del circuito completo.

$$\underbrace{\underbrace{(X + \overline{Y})}_{1} \,.\, \underbrace{(X \,.\, \overbrace{(X \oplus Y)}^{2})}_{3}}_{4}$$

Resolvemos 1 Resolvemos 2 Resolvemos 3



Unimos todo resolviendo 4



Prestar atención a los elementos negados. En el segmento 1, la negación afecta solo al valor de Y, por lo tanto el círculo que simboliza la negación en el circuito se coloca en la entrada de Y a la operación correspondiente.
En el segmento 2 la negación afecta al resultado de la función XOR, por lo tanto el círculo se coloca a la salida del símbolo de la operación.

La tabla de verdad es un cuadro en el que representamos todas las entradas posibles (diferentes combinaciones de los valores de las variables de entrada, en este ejemplo X, Y) y el valor que toma la variable de salida (Z) para cada una de las entradas.

X	Y	Z
0	0	
0	1	
1	0	
1	1	

EJERCICIOS RESUELTOS

El análisis se puede hacer sobre la función o en el circuito. Explicaremos a continuación la resolución por circuito.
Para explicar como se completa la tabla de verdad utilizaremos los mismos segmentos 1, 2, 3, y 4 usados para construir el circuito, y tomaremos como ejemplo el primer par de valores de entrada: x=0, y=0

Resolvemos 1: X=0 Y =0, pero como esta negada la entrada a la operación OR es 1.
0 y 1 en la entrada de OR generan en la salida un 1
Resolvemos 2: 0 y 0 con XOR generan un 0, pero como la operación esta negada el resultado final es 1
Resolvemos 3: una entrada es X=0 la otra es el resultado de 2, o sea 1. Si en una compuerta AND tenemos 0 y 1 en las entradas, la salida es 0.
Resolvemos 4: Es una operación AND que tiene como entradas los resultados de 1 (1) y 3 (0). 1 y 0 en un AND da como resultado 0

Para X=0 e Y=0 , Z es igual a 0.

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

Analizando cada combinación de entradas completamos la tabla.

EJERCICIOS RESUELTOS

Dado un byte X = x7 x6 x5 x4 x3 x2 x1 x0 (indeterminado) que resultado se obtiene a aplicar las siguientes operaciones

X OR 10001000
X AND 11001100
X XOR 11000011

Para resolver analizaremos que efecto produce la aplicación de cada una de estas operaciones con 0 y con 1

	X	OR	
0	0	0	Cualquiera sea el valor de X, cuándo hacemos un OR con **0** conserva su valor
	1	1	
1	0	1	Cualquiera sea el valor de X, cuándo hacemos un OR con **1** en resultado es 1
	1	1	

OR	x7	x6	x5	x4	x3	x2	x1	x0
	1	0	0	0	1	0	0	0
	1	x6	x5	x4	1	x2	x1	x0

RESULTADO: X OR 10001000 = 1 x6 x5 x4 1 x2 x1 x0

	X	AND	
0	0	0	Cualquiera sea el valor de X, cuándo hacemos un AND con **0** en resultado es 0
	1	0	
1	0	0	Cualquiera sea el valor de X, cuándo hacemos un AND con **1** conserva su valor
	1	1	

AND	x7	x6	x5	x4	x3	x2	x1	x0
	1	1	0	0	1	1	0	0
	x7	x6	0	0	x3	x2	0	0

RESULTADO: X AND 11001100 = x7 x6 0 0 x3 x2 0 0

	X	XOR	
0	0	0	Cualquiera sea el valor de X, cuándo hacemos un XOR con **0** conserva su valor
	1	1	
1	0	1	Cualquiera sea el valor de X, cuándo hacemos un XOR con **1** en resultado el valor original negado
	1	0	

XOR	x7	x6	x5	x4	x3	x2	x1	x0
	1	1	0	0	0	0	1	1
	-x7	-x6	x5	x4	x3	x2	-x1	-x0

RESULTADO: X XOR 11000011 = –x7 –x6 x5 x4 x3 x2 –x1 –x0

EJERCICIOS RESUELTOS

Dado un byte X = x7 x6 x5 x4 x3 x2 x1 x0 (indeterminado) ¿Qué operación y que máscara debería utilizar para lograr cada uno de los siguientes resultados?

a. Poner en 1 los bits 2 y 5.
b. Poner en 0 los bits 3, 6 y 7.
c. Invertir (cambiar 0 x 1y viceversa) los bits 0, 1 y 4.
d. Cambiar los bits de modo que el bit 0 quede en 1, el 5 en 0 y el 7 invierta su valor
En todos los casos, los bits no nombrados deben mantener su valor original. Considere cada inciso independientemente (no son operaciones secuenciales, siempre se parte del mismo byte original)

Antes de resolver explicaremos el concepto de máscara. Una máscara es una combinación de 0 y 1 ordenados de tal forma que permita conservar el valor de algunos bits y modificar los otros cuando se realiza entre un valor dado y la máscara una operación lógica (OR, AND, XOR). El cambio que se observara en los bits que no mantienen su valor, dependerá de la operación utilizada.

a. Poner en 1 los bits 2 y 5.
Para lograr esto debemos utilizar la operación **OR** (con 0 conserva, con 1 fuerza a 1).
La máscara a utilizar deberá, por lo tanto contener 1 en los bits a poner en 1 (2 y 5) y 0 en el resto (0, 1, 3, 4, 6 y 7)

Bit 7	Bit 6	**Bit 5**	Bit 4	Bit 3	**Bit 2**	Bit 1	Bit 0	
0	0	**1**	0	0	**1**	0	0	Máscara

OR	x7	x6	**x5**	x4	x3	**x2**	x1	x0	Byte original
	0	0	**1**	0	0	**1**	0	0	Máscara
	x7	x6	**1**	x4	x3	**1**	x1	x0	**RESULTADO**

b. Poner en 0 los bits 3, 6 y 7.
Para lograr esto debemos utilizar la operación **AND** (con 1 conserva, con 0 fuerza a 0).
La máscara a utilizar deberá, por lo tanto contener 0 en los bits a poner en 0 (3, 6 y 7) y 1 en el resto (0, 1, 2, 4, y 5)

Bit 7	**Bit 6**	Bit 5	Bit 4	**Bit 3**	Bit 2	Bit 1	Bit 0	
0	**0**	1	1	**0**	1	1	1	Máscara

OR	**x7**	**x6**	x5	x4	**x3**	x2	x1	x0	Byte original
	0	**0**	1	1	**0**	1	1	1	Máscara
	0	**0**	x5	x4	**0**	x2	x1	x0	**RESULTADO**

c. Invertir (cambiar 0 x 1 y viceversa) los bits 0, 1 y 4.
Para lograr esto debemos utilizar la operación **XOR** (con 0 conserva, con 1 niega valor).
La máscara a utilizar deberá, por lo tanto contener 1 en los bits a cambiar (0, 1 y 4) y 0 en el resto (2, 3, 5, 6 y 7)

Bit 7	Bit 6	Bit 5	**Bit 4**	Bit 3	Bit 2	**Bit 1**	**Bit 0**	
0	0	0	**1**	0	0	**1**	**1**	Máscara

OR	x7	x6	x5	**x4**	x3	x2	**x1**	**x0**	Byte original

EJERCICIOS RESUELTOS

0	0	0	**1**	0	0	**1**	**1**	Máscara
x7	x6	X5	**-x4**	x3	x2	**-x1**	**-x0**	**RESULTADO**

d. Cambiar los bits de modo que el bit 0 quede en 1, el 5 en 0 y el 7 invierta su valor

En este caso tenemos que hacer 3 operaciones sucesivas, cada una con su correspondiente máscara, operando en cada paso sobre el resultado anterior.

Para que bit 0 quede en 1 haremos un OR con 0000 0001
Para que bit 5 quede en 0 haremos un AND con 1101 1111
Para invertir bit 7 haremos un XOR con 1000 0000.

OR	x7	x6	x5	x4	x3	x2	x1	**x0**
	0	0	0	0	0	0	0	**1**
	x7	x6	x5	x4	x3	x2	x1	**1**
AND	x7	x6	**x5**	x4	x3	x2	x1	1
	1	1	**0**	1	1	1	1	1
	x7	x6	**0**	x4	x3	x2	x1	1
XOR	**x7**	x6	0	x4	x3	x2	x1	1
	1	0	0	0	0	0	0	0
	-x7	x6	0	x4	x3	x2	x1	1

RESULTADO:
El contenido del byte luego de las 3 operaciones es: -x7 x6 0 x4 x3 x2 x1 0