Parte IV – Representación de números enteros y lógica binaria

1. Representación de números enteros

Los ordenadores utilizan varios métodos para la representación interna de los números (positivos y negativos), que son:

- ✔ Signo magnitud (también llamado módulo y signo).
- ✔ Complemento a 1 (C-1).
- ✔ Complemento a 2 (C-2).
- ✔ Exceso a 2^{N-1}.

En estas representaciones de números se utiliza el sistema binario y se considera que tenemos un número limitado de bits para casa dato numérico (la cantidad de bits suele coincidir con la longitud de la palabra del ordenador que suele ser de 16, 32, 64,… bits).

Palabra: tamaño de información manejada en paralelo por los componentes del sistema, como la memoria, los registros o los buses son comunes las palabras de 8, 32, 64, 128, 256 bits – 1, 4, 8, 16, 32, bytes.

Cuanto mayor sea el tamaño de palabra, mayor es la precisión y potencia de cálculo del ordenador. Este número de bits disponibles se representa por n.

Signo Magnitud

En este sistema de representación el bit que está situado más a la izquierda representa el signo, y su valor será 0 para el signo positivo y 1 para el signo negativo. El resto de bits (n-1) representa el módulo del número.

Este enfoque es directamente comparable a la forma habitual de mostrar el signo (colocando “+” o “-” al lado de la magnitud del número). Algunas de las primeras computadoras binarias (como por ejemplo la IBM 7090) utilizaron esta representación, por su relación obvia con la práctica habitual.

El formato Signo magnitud es además el habitual para la representación del significando en números en punto flotante.

Si tenemos una palabra de 8 bits (n = 8) y queremos representar los números 10 y –10 sería:

Número 10	**0** ↑ Signo +	**0001010** ↑ Magnitud

Número -10	**1** ↑ Signo -	**0001010** ↑ Magnitud

Desventajas de la representación en Signo Magnitud

- ✔ Es más complejo operar aritméticamente. Para realizar una suma, por ejemplo, primero hay que determinar si los dos números tienen el mismo signo, y en caso de que sea así, realizar la suma de la parte significativa, pero en caso contrario, restar el mayor del menor y asignar el signo del mayor.

- ✔ Posee doble representación del cero.

Ventajas de la representación en Signo Magnitud

- ✔ Posee un rango simétrico: los números van del $+127_{(10} = 01111111_{(2}$, hasta el $-127_{(10} = 11111111_{(2}$. Y en forma general, para n-bits, el rango (en decimal) para Signo Magnitud es $(-(2^{n-1}-1); 2^{n-1}-1)$, o bien $\pm (2^{n-1}-1)$.

Complemento a uno (C-1)

Este sistema de representación utiliza el bit de más a la izquierda para el signo, correspondiendo el 0 para el signo + y el 1 para el signo -. Para los números positivos, los n-1 bits de la derecha representan el módulo (igual que en el caso anterior).

El negativo de un número positivo se obtiene complementando todos sus dígitos (cambiando 0 por 1 y viceversa), incluido el bit de signo.

Número 10	**0** ↑ Signo +	**0001010** ↑ Módulo

Número -10	**1** ↑ Signo -	**1110101** ↑ Módulo

Desventajas de la representación en Complemento a uno

- ✔ Posee doble representación del cero. Al representar en Complemento a uno, aparece nuevamente el cero signado (con signo): $00000000_{(2}$ (+010) y $11111111_{(2}$ (-010).

Ventajas de la representación en Complemento a uno

- ✔ Posee un rango simétrico: los números van del $+127_{(10} = 01111111_{(2}$, pasando por el $+0_{(10} = 00000000_{(2}$ y el $-0_{(10} = 11111111_{(2}$, hasta el $-127_{(10} = 10000000_{(2}$. Y en forma general, para n-bits, el rango (en decimal) para Complemento a uno es $(-(2^{n-1}-1); 2^{n-1}-1)$, o bien $\pm (2^{n-1}-1)$.

- ✔ Permite operar aritméticamente. Al operar se debe sumar el acarreo obtenido al final de la suma/resta realizadas (conocido como ***end-around carry***), en caso de haberlo obtenido, para conseguir el resultado correcto.

  Por ejemplo: $00010101_{(2} + 10011110_{(2} = 10110011_{(2}$ $(+21_{(10} + -97_{(10} = -76_{(10})$ puesto que el ***end-around carry*** es cero; pero, $00000010_{(2} + 11111110_{(2} = 100000000_{(2}$ $(+2_{(10} + -1_{(10} = -0_{(10} \neq +1_{(10})$, que corregimos mediante $00000010_{(2} + 11111110_{(2} = 00000000_{(2} + 1_{(2} = 00000001_{(2}$, que es el resultado correcto.

Los protocolos de **Internet IPv4, ICMP, UDP y TCP usan todos el mismo algoritmo de suma de verificación de 16 bits en complemento a uno**.

Complemento a dos (C-2)

Este sistema de representación utiliza el bit más a la izquierda para el signo, correspondiendo el 0 para el signo + y el 1 para el -. En el caso de los números positivos, los n-1 bits de la derecha representan el módulo (igual en los dos casos anteriores).

El negativo de un número se obtiene en dos pasos:

- ✔ Se complementa el número positivo en todos sus bits (cambiando los ceros por 1 y viceversa), incluido el bit de signo, es decir se realiza el complemento a 1.
- ✔ Al resultado obtenido anteriormente se le suma 1 (en binario) despreciando el último acarreo si existe.

La representación en C-2 de los números 10 y –10 sería:

Número 10	**0** ↑ Signo +	**0001010** ↑ Módulo

Número -10		
Primer paso	**1**	**1110101**
Segundo paso	**1** **+**	**1110101** **1**
	1 ↑ Signo -	**1110110** ↑ Módulo

De esta forma, en la representación por Complemento a dos de un número signado de n-bits asignamos:

- ✔ Un bit para representar el signo. Ese bit a menudo es el **bit más significativo** y, por convención: un 0 denota un número positivo, y un 1 denota un número negativo;
- ✔ Los (n-1) bits restantes para representar el **significando que es la magnitud del número en valor absoluto** para el caso de **números positivos**, o bien, en el **complemento a dos del valor absoluto del número, en caso de ser negativo**.

Desventajas de la representación en Complemento a dos

- ✔ Posee un rango asimétrico: los números van del $+127_{(10}$ ($01111111_{(2}$) al $-128_{(10}$ ($10000000_{(2}$). Y aquí aparece la primera diferencia, el $11111111_{(2}$, ya no es $-0_{(10}$ como en la representación anterior, sino que es $-1_{(10}$, y al llegar al $10000000_{(2}$ nos encontramos con que el complemento a dos de $10000000_{(2}$ es $10000000_{(2}$. Por convención, se asigna a este número particular el valor $-128_{(10}$ (para 8 bits). Luego, en forma general, para n-bits, el rango (en decimal) para Complemento a dos es $(-2^{n-1}; 2^{n-1}-1)$.

Ventajas de la representación en complemento a dos

- ✔ No posee doble representación del cero.
- ✔ Permite operar aritméticamente.

Exceso a K – Exceso a 2^{n-1} ($K = 2^{n-1}$)

Un último enfoque al problema de representar un número signado es el exceso a K, donde a cada número se le suma el mismo valor, y está en exceso por dicho valor. Este formato es habitual para la representación del exponente en números en punto flotante.

K no tiene un valor estandarizado, pero suele tomarse como **2^{n-1} (que coincide con el complemento a dos con el bit más significativo negado)**, o como **2^{n-1}-1** (como en el caso de la **norma IEEE-754**).

Este método de representación no utiliza ningún bit para el signo, con lo cual todos los bits representan un módulo o valor. Este valor se corresponde con el número representado más el exceso, que para n bits viene representado por 2^{n-1}.

Por ejemplo, para 8 bits (n=8) el exceso es de $2^{8-1} = 2^7 = 128$, con lo cual:

- ✔ El número 10 vendrá representado por 10 + 128 = **138** (en binario).
- ✔ Para el caso del número –10 tendremos –10 + 128 = **118** (en binario).

Número 10	**10001010**
Número -10	**01110110**

Desventajas de la representación en Exceso 2^{n-1}-1

- ✔ Requiere de operaciones aritméticas intermedias para su obtención, y de cambiar el número de bits se deben actualizar dichas operaciones intermedias para reflejar el nuevo exceso.

- ✔ Posee rango asimétrico: este va desde $+128_{(10} = 11111111_{(2}$ hasta $-127_{(10} = 00000000_{(2}$. Y en forma general, para n-bits, el rango (en decimal) para **Exceso 2^n** es $(-(2^{n-1}-1); 2^{n-1})$.

Ventajas de la representación en Exceso 2^{n-1}-1

- ✔ El menor número posible de representar consiste en todos los bits en cero y el mayor en unos.

- ✔ Permite operar aritméticamente, pero hay que tener en cuenta que cada operación lleva asociado su exceso y esto hay que **restárselo** al resultado final, para corregir la representación. Por ejemplo, $00011110_{(2} + 10110101_{(2} = 11010011_{(2}$ ($-97_{(10} + 54_{(10} = 211_{(10} \neq -43_{(10}$). El resultado, en exceso $127_{(10}$, es $211_{(10} - 127_{(10} = 84_{(10}$. Pero, hay que tener en cuenta que al sumar dos números con exceso $127_{(10}$ debemos restar dos veces el exceso. Luego $84_{(10} - 127_{(10} = -43_{(10}$, que es el resultado correcto.

- ✔ No hay empaquetación del número. Por esto nos referimos a que no hay que recordar qué partes del número son signo y significando, sino que los n-bits, son el número.

La tabla siguiente compara la representación de los enteros entre 8 y -8 (incluidos) usando 4 bits.

Decimal	Entero sin signo	Signo y Magnitud	Complemento a uno	Complemento a dos	En exceso a 7
+8	1000	*n/d*	*n/d*	*n/d*	1111
+7	0111	0111	0111	0111	1110
+6	0110	0110	0110	0110	1101
+5	0101	0101	0101	0101	1100
+4	0100	0100	0100	0100	1011
+3	0011	0011	0011	0011	1010
+2	0010	0010	0010	0010	1001
+1	0001	0001	0001	0001	1000
+0	0000	0000	0000	0000	0111
-0	*n/d*	1000	1111	*n/d*	*n/d*
-1	*n/d*	1001	1110	1111	0110
-2	*n/d*	1010	1101	1110	0101
-3	*n/d*	1011	1100	1101	0100
-4	*n/d*	1100	1011	1100	0011
-5	*n/d*	1101	1010	1011	0010
-6	*n/d*	1110	1001	1010	0001
-7	*n/d*	1111	1000	1001	0000
-8	*n/d*	*n/d*	*n/d*	1000	*n/d*

2. La importancia de la representación en complementos

Se utiliza para que dentro de las máquinas no se tenga que tener más circuitería de la necesaria. La ventaja clara que se tiene es que **las restas se realizarán como sumas, y por tanto nuestra ALU no tendrá que incorporar un restador** (con un circuito sumador nos bastará para realizar tanto sumas como restas).

Complemento a la uno (C-1, si la base es 2 – Complemento a la base menos uno)

Dado un numero N, el Complemento a uno es el número que resulta de restar cada una de las cifras de N a la base menos 1 del sistema de numeración que se esté utilizando. Podremos entonces restar un número de otro simplemente sumando al minuendo el complemento a la base menos uno del sustraendo. La cifra que se arrastra del resultado se descarta, y se suma al resultado obtenido.

Supongamos que queremos restar 30-25. En base 2 calcularemos el Complemento a 1 al número 25 (11001):

$$\begin{array}{rr} & 11111 \\ - & 11001 \\ \hline \text{Complemento a 1 (11001)} = & 00110 \end{array}$$

Como lo que queremos es realizar una resta, una vez hallado el C-1 del sustraendo, se le suma éste último al minuendo, y **el último acarreo (en caso de que exista) se le suma al resultado final**.

```
    11110  →30
+   00110  →C-1 (-25)

    00100
+       1
  -------
    00101  →5
```

Si no se hubiese producido el acarreo final, entonces **no hay que sumar nada al resultado**, pero lo que ocurre es que éste **será negativo**, y por tanto debemos **complementarlo a uno** para interpretarlo.

```
    00001  →C-1 (-30)
+   11001  →25

    11010  No hay acarreo → C-1
           Número negativo
  -------
    00101  →-5
```

Complemento a dos (C-2, si la base es dos – Complemento a la base)

Dado un numero N, el complemento a la base sería el número que resulta de restar cada una de las cifras del numero N a la base menos uno y posteriormente sumar 1 a la diferencia obtenida. Esto es, es lo mismo que obtener el complemento a la base menos uno y después sumar uno al resultado.

El procedimiento para restar es muy parecido al anterior: **calcular el C2 (suponiendo b=2) del sustraendo, y añadirlo al minuendo**. Para realizar la resta 30-25 procedemos a sumar el C2 (25):

```
    11110  →30
+   00111  →C-2 (-25)
  -------
  ↓1 00101  →5
```

Descartamos acarreo

El resultado final es, pues 00101 (5). **Si no se hubiese producido acarreo, entonces, igual que antes, habría que interpretar el resultado como negativo**, **calculando su valor absoluto complementando a 2**, en este caso, el resultado.

```
    00010  →C-2 (-30)
+   11001  →25
  -------
    11011  No hay acarreo → C-2
           Número negativo
  -------
    00100
+       1
  -------
    00101  →-5
```

3. Representación interna de la información

El bit es la unidad mínima de información; con él podemos representar dos valores cualesquiera, como verdadero o falso, abierto o cerrado, blanco o negro, norte o sur, rojo o azul, etc. Basta con asignar uno de esos valores a cada estado. En informática, se utiliza el sistema binario, y sólo se manejan **las cifras cero y uno** (**0 y 1**), ya que los ordenadores trabajan internamente con **dos niveles de voltaje**: "**apagado**" (0) y "**encendido**" (1), por lo que su sistema de numeración natural es el sistema binario.

Cuando se almacena la información no se trabaja a nivel de bit, sino que se trabaja a **nivel de carácter** (letra, número o signo de puntuación), que ocupa lo que se denomina un **byte**, que a su vez está compuesto de **8 bits**. El ordenador trabaja con **agrupaciones de bits fáciles de manipular y suelen ser múltiplos de 2**, la base del sistema binario. Los tamaños más comunes son:

- ✔ **Octeto**, **carácter** o **byte**: es la agrupación de 8 bits, el tamaño típico de información; con él se puede codificar el alfabeto completo (ASCII estándar).

- ✔ **Palabra**: tamaño de información manejada en paralelo por los componentes del sistema, como la memoria, los registros o los buses. Son comunes las palabras de **8, 32, 64, 128 y 256 bits: 1 byte, 4, 8, 16, 32 bytes**. A **mayor tamaño de palabra, mayor es la precisión y la potencia de cálculo** del ordenador. Así, cuando decimos que un archivo de texto ocupa 5000 bytes, queremos decir que contiene el equivalente a 5000 letras o caracteres (entre dos y tres páginas de texto sin formato).

Lo normal es utilizar los **múltiplos del byte**: el **Kilobyte** (**KB**), el **Megabyte** (**MB**), el **Gigabyte** (**GB**), etc. En informática se utilizan las **potencias de 2** (2^3, 2^{10}, 2^{20}...) para representar las medidas de la información. Sin embargo, se ha extendido el uso de las **potencias de 10** (uso decimal), debido a que se ha impuesto el uso del Sistema Internacional de Medidas (SI), o sistema métrico. Así pues, el primer término de medida que se utilizó fue el **Kilobyte** (**KB**), y se eligió este **porque 2^{10} es aproximadamente 1000**, que se asocia con el **kilo** (1000); en realidad debería ser **1024 bytes**, ya que 2^{10} son 1024.

Nombre (símbolo)	Sistema Internacional de Unidades (SI) Estándar (uso decimal)	Prefijo binario (uso binario)	Nombre (símbolo)
Kilobyte (KB)	$1000^1 = 10^3$ bytes	$1024^1 = 2^{10}$ bytes	Kibibyte (Kib)
Megabyte (MB)	$1000^2 = 10^6$ bytes	$1024^2 = 2^{20}$ bytes	Mebibyte (Mib)
Gigabyte (GB)	$1000^3 = 10^9$ bytes	$1024^3 = 2^{30}$ bytes	Gibibyte (Gib)
Terabyte (TB)	$1000^4 = 10^{12}$ bytes	$1024^4 = 2^{40}$ bytes	Tebibyte (Tib)
Petabyte (PB)	$1000^5 = 10^{15}$ bytes	$1024^5 = 2^{50}$ bytes	Pebibyte (Pib)
Exabyte (EB)	$1000^6 = 10^{18}$ bytes	$1024^6 = 2^{60}$ bytes	Exbibyte (Eib)
Zettabyte (ZB)	$1000^7 = 10^{21}$ bytes	$1024^7 = 2^{70}$ bytes	Zebibyte (Zib)
Yottabyte (YB)	$1000^8 = 10^{24}$ bytes	$1024^8 = 2^{80}$ bytes	Yobibyte (Yib)

El **Megabyte** (**MB**) equivale a 10^6 (1000000 bytes) o 2^{20} (1048576 bytes), según el contexto. Es el conjunto de **1024 kilobytes**, 2^{20} bytes → $2^{10} \cdot 2^{10} = 1024 \cdot 1024 = 1048576$; o podemos decir un millón de bytes (10^6).

Un **Gigabyte** (**GB**) equivale a 2^{30} bytes o 10^9 bytes, según el uso. Es la **unidad que más se usa actualmente** para especificar la capacidad de la memoria RAM, de las memorias de tarjetas gráficas, de los CD-ROM o el tamaño de los programas y de los archivos grandes. La capacidad de almacenamiento se mide habitualmente en gigabytes, es decir, en miles de megabytes. Un GB es el conjunto de 1024 megabytes, 2^{30} bytes, o lo que es lo mismo, $2^{10} \cdot 2^{10} \cdot 2^{10} = 1024 \cdot 1024 \cdot 1024 = 1073741824$; mil millones de bytes (10^9).

4. Representación de datos alfabéticos y alfanuméricos

Los datos llegan y salen del ordenador a través de los **periféricos de entrada y de salida**, respectivamente, y cada fabricante de componentes de E/S (entrada/salida) podría asignar una combinación diferente al mismo símbolo de origen (por ejemplo, las letras del alfabeto); sin embargo, esto no sería nada positivo en un mercado abierto como el informático. Por eso se tiende a la **estandarización de códigos**, que ha llevado a la universalización de unos pocos códigos de E/S, como el **BCD**, **EBCDIC**, **ASCII** y **Unicode**. La **mayoría** de estos códigos representan **cada carácter por medio de un byte** (8 bits).

ASCII

El Código Estadounidense Estándar para el Intercambio de Información, o **ASCII** (*American Standard Code for Information Interchange*), es la recomendación X3.4-1977 del Instituto Estadounidense de Normas Nacionales (ANSI). Utiliza grupos de **7 bits por carácter**, permitiendo 2^7 = **128 caracteres diferentes**, lo que es suficiente para el alfabeto en letras mayúsculas y minúsculas y los símbolos de una máquina de escribir corriente, además de algunas combinaciones reservadas para su uso interno. El código **ASCII extendido** usa **8 bits por carácter**, lo que **añade otros 128 caracteres posibles**. Este juego de códigos más amplio **permite que se agreguen los símbolos de lenguajes extranjeros y varios símbolos gráficos**.

Unicode

El **Unicode Standard** es una norma de codificación universal de caracteres. Su objetivo es representar cada elemento usado en la escritura de cualquier idioma del planeta. Los idiomas actuales más importantes del mundo pueden escribirse con Unicode, incluyendo su puntuación, símbolos especiales, símbolos matemáticos y técnicos, formas geométricas, caracteres gráficos y modelos de Braille.

Unicode proporciona un **número único para cada carácter, sin importar la plataforma, sin importar el programa, sin importar el idioma**. Es **compatible con numerosos sistemas operativos**, con todos los exploradores actuales y con muchos otros productos. La aparición de la norma Unicode y la disponibilidad de herramientas que la respaldan se encuentran entre las más recientes e importantes tendencias en tecnología de software.

La incorporación de Unicode en sitios web y en aplicaciones de cliente-servidor o de múltiples niveles permite disminuir los costes del uso de juegos de caracteres heredados, ya que Unicode permite que un producto de software o sitio web específico se oriente a múltiples plataformas, idiomas y países, sin necesidad de rediseñarlo.

Además, permite que los datos se trasladen a través de gran cantidad de sistemas distintos sin sufrir daños. Básicamente, las computadoras solo trabajan con números. Almacenan letras y otros caracteres mediante la asignación de un número a cada uno. Antes de que se inventara Unicode, existían cientos de sistemas de codificación distintos para asignar estos números. Ninguna codificación específica podía contener caracteres suficientes; por ejemplo, la Unión Europea, por sí sola, necesita varios sistemas de codificación distintos para cubrir todos sus idiomas. Incluso para un solo idioma como el inglés no había un único sistema de codificación que se adecuara a todas las letras, signos de puntuación y símbolos técnicos de uso común.

BCD y EBCDIC

BCD, que significa **decimal codificado en binario** (*Binary Coded Decimal*), en realidad no es un código de E/S, sino una **forma de codificar los símbolos numéricos del 0 al 9** que se emplean en varios códigos de E/S, entre los que figuran EBCDIC y ASCII.

BCD divide cada octeto en dos mitades o cuartetos, cada uno de los cuales almacena en binario una cifra. Con este código es **muy fácil convertir del binario al sistema decimal**.

EBCDIC, o código BCD extendido de caracteres decimales codificados en binario para el intercambio de información (EBCDIC, *Extended BDC Interchange Code*), es un **sistema de codificación que tiene como objetivo la representación de caracteres alfanuméricos**. Es el utilizado por la empresa IBM para sus ordenadores de la serie IBM PC (miniordenadores y mainframes).

En este sistema de codificación, **cada carácter tiene 8 bits**. Al tener ocho, podremos representar hasta 2^8 = 256 caracteres. Será posible almacenar letras mayúsculas, minúsculas, caracteres especiales, caracteres de control para dispositivos de E/S y para comunicaciones.

Decimal	Binario	BCD
0	0000	0000 0000
1	0001	0000 0001
2	0010	0000 0010
3	0011	0000 0011
4	0100	0000 0100
5	0101	0000 0101
6	0110	0000 0110
7	0111	0000 0111
8	1000	0000 1000
9	1001	0000 1001
10	1010	0001 0000
11	1011	0001 0010
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

5. Álgebra de Boole

Es la herramienta fundamental de la electrónica digital, constituyendo su base matemática. El álgebra de Boole es un conjunto que consta de dos elementos 0 y 1 que no siempre representan números. Pueden ser:

- ✔ 0 ⇒ **Falso** ⇒ **Apagado** ⇒ No tensión ⇒ Interruptor abierto ⇒ etc.
- ✔ 1 ⇒ **Verdadero** ⇒ **Encendido** ⇒ Tensión ⇒ Interruptor cerrado ⇒ etc.

Operadores, postulados, propiedades, teoremas y leyes

Operadores		
Suma ***a+b***	Producto ***a·b***	Complementación ***a'*** o $\overline{a}$

Postulados		
Existe un complementario	$a + \overline{a} = 1$	$a \cdot \overline{a} = 0$
Idempotencia	$a + a = a$	$a \cdot a = a$
Existe un elemento neutro	$a + 0 = a$	$a \cdot 1 = a$
Dominio del 0 y del 1	$a + 1 = 1$	$a \cdot 0 = 0$
Doble complementación	$\overline{a} = a$	

Propiedades		
Conmutativa	$a + b = b + a$	$a \cdot b + b \cdot a$
Distributiva	$a + b \cdot c = (a + b) \cdot (a + c)$	$a \cdot (b + c) = a \cdot b + a \cdot c$
Asociativa	$a \cdot (b \cdot c) = (a \cdot b) \cdot c = a \cdot b \cdot c$	$a + (b + c) = (a + b) + c = a + b + c$

Teoremas		
Absorción	$a + (a \cdot b) = a$	$a \cdot (a + b) = a$
Unicidad de complementario	$a = 1 \rightarrow \overline{a} = 0$	
	$\overline{a} = 1 \rightarrow a = 0$	
Dualidad	$a \cdot \overline{b} + \overline{a} \cdot b = \overline{(\overline{a} + b) \cdot (a + \overline{b})}$	

Leyes de De Morgan	
$\overline{a \cdot b \cdot c \cdot d} = \overline{a} + \overline{b} + \overline{c} + \overline{d}$	$\overline{a + b + c + d} = \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d}$

Funciones lógicas algebraicas

Es una expresión algebraica en la que se relacionan entre sí las variables binarias por medio de operaciones básicas: producto lógico, suma lógica e inversión. De forma general podemos expresar una función lógica de la forma:

f = f (a, b, c, ...) donde el valor de ***f*** depende del valor de las variables ***a, b, c***...

Un ejemplo de función lógica podría ser: ***f = a · b + a · b + c + c · a · a + b***.

Las variables pueden tomar los valores 0 o 1. Si a una variable le asignamos el valor *a = 1* la variable complementada es *a = 0*, pero si asignamos *a = 0* entonces es *a = 1*. De una función lógica, se dice, que la función se hace cierta para las combinaciones de las variables que expresa.

Lógica binaria

La lógica binaria es la que trabaja con **variables binarias** y **operaciones lógicas del Álgebra de Boole**. Lo que comúnmente en lógica es falso o verdadero, en la lógica binaria lo vemos representado mediante dígitos, utilizando exclusivamente los **valores 0 y 1**, números que de por sí no tienen un valor numérico de tipo real, sino más bien de tipo discreto, es decir, 0 y 1 representan distintos estados del objeto de estudio, por ejemplo, a la hora de poder desarrollar un circuito digital.

Los circuitos digitales funcionan generalmente bajo tensiones de 5 voltios en corriente continua (por ejemplo la tecnología TTL) si bien existen excepciones como la serie CMOS, que trabaja en diferentes rangos que pueden ir desde los 4 a los 18 voltios.

Generalmente, el estado lógico 0 representa una ausencia de tensión, o un nivel bajo; y el estado lógico 1 representa una existencia de tensión, o un nivel alto. Mediante la combinación de estos valores, es posible generar una serie de datos convertible a cualquier código.

Operaciones lógicas compuestas

Siguiendo el **Álgebra de Boole**, se pueden combinar operaciones empleando varias variables y obteniendo resultados más **complejos**. A continuación un ejemplo de tabla de verdad de una operación lógica compuesta.

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

1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

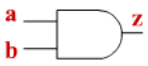
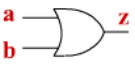
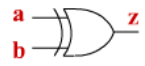
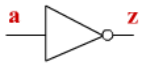
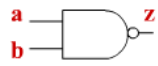
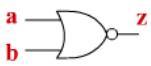
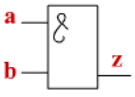
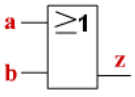
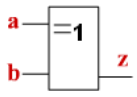
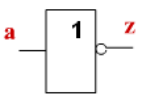
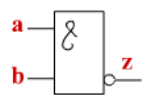
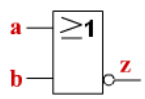
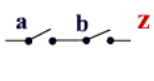
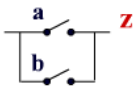
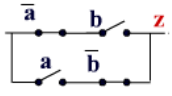
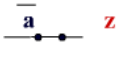
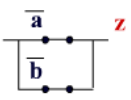
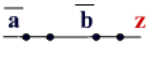
6. Puertas lógicas

Las puertas lógicas son circuitos electrónicos (bloques en la construcción de un circuito digital) formados internamente por transistores que otorgan señales de voltaje a los elementos del álgebra de Boole. Producen señales en binario 1 ó 0 cuando se satisfacen los requisitos de entrada lógica. Cada puerta tiene un símbolo gráfico diferente y su operación puede describirse por medio de una función algebraica.

Existen diferentes tipos de puertas y algunas de estas son más complejas, con la posibilidad de ser simuladas por compuertas más sencillas. La **relación entrada - salida** de las variables binarias para cada puerta pueden representarse en forma tabular en una **tabla de verdad**.

Las diversas puertas lógicas se encuentran comúnmente en **sistemas de computadoras digitales**, pero se pueden aplicar en otras áreas de la ciencia como mecánica, hidráulica o neumática.

FUNCIONES LÓGICAS BÁSICAS

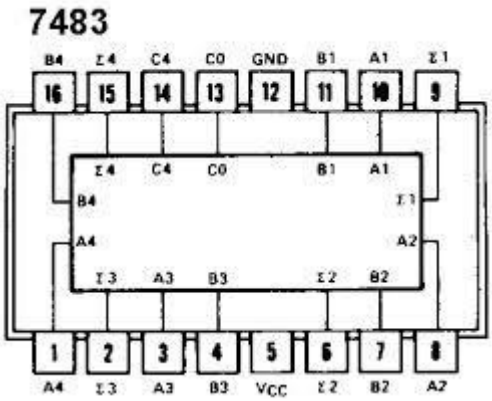
NOMRE	AND - Y	OR - O	XOR O-exclusiva	NOT Inversor	NAND	NOR
SÍMBOLO						
SÍMBOLO						
TABLA DE VERDAD	a b z 0 0 0 0 1 0 1 0 0 1 1 1	a b z 0 0 0 0 1 1 1 0 1 1 1 1	a b z 0 0 0 0 1 1 1 0 1 1 1 0	a z 0 1 1 0	a b z 0 0 1 0 1 1 1 0 1 1 1 0	a b z 0 0 1 0 1 0 1 0 0 1 1 0
EQUIVALENTE EN CONTACTOS						
AXIOMA	$z = a \cdot b$	$z = a + b$	$z = \bar{a} \cdot b + a \cdot \bar{b}$	$z = \bar{a}$	$z = \overline{a \cdot b}$	$z = \overline{a + b}$

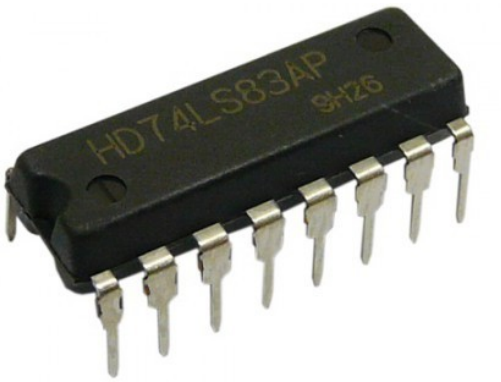
Sumadores

Un sumador es un circuito que realiza la suma de dos palabras binarias. Es **distinta de la operación OR**, con la que no nos debemos confundir. La operación suma de números binarios tiene la **misma mecánica que la de números decimales**.

En la suma de números binarios con dos o más bits, puede ocurrir el mismo caso que podemos encontrar en la suma de números decimales con **varias cifras**: cuando al sumar los dos primeros dígitos se obtiene una **cantidad mayor de 9**, se da como resultado el dígito de menor peso y “me llevo" el anterior a la siguiente columna, para sumarlo allí.

En la suma binaria de los dígitos **1 + 1, el resultado es 0 y me llevo 1**, que debo sumar en la columna siguiente y pudiéndose escribir 10, solamente cuando sea la última columna a sumar. A este bit más significativo de la operación de sumar, se le conoce en inglés como carry (acarreo), equivalente al “me llevo una” de la suma decimal.





Semisumador

Es un dispositivo capaz de **sumar dos bits y dar como resultado la suma de ambos y el acarreo**. La señal de acarreo representa un **desbordamiento en el siguiente dígito en una suma de varios dígitos**. El diseño más simple de semisumador, representado a la derecha, incorpora una **puerta XOR** para S y una **puerta AND para C.** Dos semisumadores pueden ser combinados para hacer un sumador completo, añadiendo una puerta OR para combinar sus salidas de acarreo.

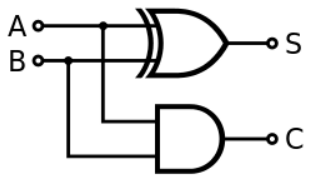
La tabla de verdad correspondiente a esta operación sería:

Entradas		**Salidas**	
A	**B**	**C**	**S**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Con lo que sus funciones canónicas serán:

$$C = A \cdot B$$
$$S = A \cdot B + A \cdot B = A \oplus B$$

Que una vez implementado con puertas lógicas, un semisumador tendría el circuito:

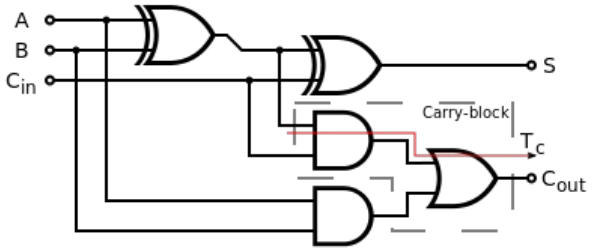


Sumador completo

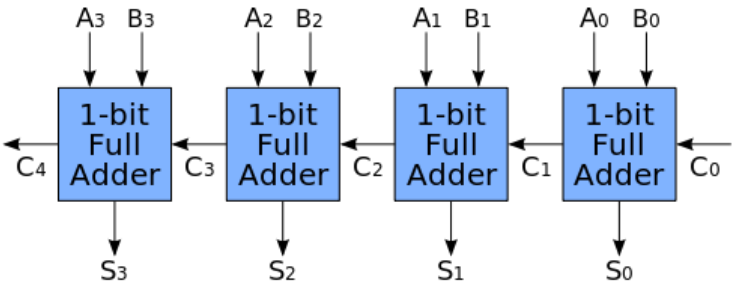
Un sumador completo suma números binarios junto con las cantidades de acarreo. Un sumador completo de un bit añade tres bits, a menudo escritos como A, B y C_{in} siendo A y B son los sumandos y C_{in} es el acarreo que proviene de la anterior etapa menos significativa.

El sumador completo suele ser un componente de una cascada de sumadores, que suman 8, 16, 32, etc. bits. El circuito produce una salida de dos bits, al igual que el semisumador, denominadas acarreo de salida (C_{out}) y suma S.

Entradas			**Salidas**	
A	**B**	**C^{-1}**	**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	0



Un sumador con **propagación de acarreo**, también denominado sumador de **acarreo serie**, es aquel constituido por varios sumadores completos en los cuales el acarreo de salida se conecta a la entrada de acarreo del sumador siguiente. Sin embargo, puede observarse que el primero de los sumadores puede ser **reemplazado por un semisumador suponiendo que el acarreo de entrada C_{in} es cero**.



Semirestador

Un semirestador es un circuito combinacional que sustrae dos bits y produce su diferencia. Requiere dos salidas: una genera la diferencia y la otra genera la señal binaria que informa a la siguiente etapa que se ha tomado un 1. La tabla de verdad para las relaciones de entrada-salida de un medio restador ahora puede derivarse como sigue:

Entradas		Salidas	
A	**B**	**C**	**S**
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0