TÉCNICA DIGITAL (86.01)

"ÁLGEBRA DE BOOLE"

1) Introducción

El álgebra booleana, como cualquier otro sistema matemático deductivo, puede definirse con un conjunto de elementos, un conjunto de operadores y un número de axiomas no probados o postulados. En 1854 George **Boole** presentó un tratamiento sistemático de la lógica, y desarrolló para este propósito un sistema algebraico que ahora se conoce como álgebra de Boole o álgebra booleana. En 1938 Claude E. **Shannon** introdujo un álgebra booleana de dos valores denominada álgebra de interruptores, en la cual demostró que las propiedades de los circuitos eléctricos biestables con interruptores pueden representarse con este álgebra. Para la definición formal del álgebra booleana, se emplean los postulados formulados por Edward V. **Huntington** en 1904. Estos postulados o axiomas no son únicos para definir el álgebra booleana, también se han usado otros conjuntos de postulados distintos a los que plantearemos aquí. El álgebra booleana es una estructura algebraica definida en un conjunto de elementos M junto con dos operadores binarios + y • siempre y cuando se cumpla con los postulados de Huntington. Al operador binario + se lo denomina unión, disyunción o suma lógica (**OR**), y al operador binario • se lo denomina intersección, conjunción o producto lógico (**AND**). Además, tenemos una tercera operación que es llamada complemento, negación o inversión (**NOT**). Las operaciones OR, AND y NOT son las fundamentales de este álgebra.
El álgebra booleana rige la lógica binaria, la teoría de conjuntos, la lógica proposicional, la lógica de contactos, interruptores o pulsadores, etc.

2) Principio de Dualidad

El principio de **dualidad** establece que a toda relación o ley lógica le corresponderá su ley dual, formada mediante el intercambio de los operadores de suma con los de producto, y de los 1 con los 0, y viceversa.

3) Postulados de Huntington

Un conjunto M es cerrado respecto a un operador binario si, para cada par de elementos de M, el operador binario especifica una regla para obtener un número único de M.

M es un conjunto cerrado respecto al operador +
M es un conjunto cerrado respecto al operador •
Por lo tanto: para A, B ϵ M

$A + B \epsilon M$
$A \cdot B \epsilon M$

Además, existen al menos 2 elementos A, B ϵ M tales que A ≠ B.

$M = \{0,1\}$

Nótese que, de acuerdo con el principio de Dualidad, veremos que cada postulado se aplica a ambas operaciones lógicas + y •.

P1) Conmutatividad

M es un conjunto conmutativo respecto al operador +
M es un conjunto conmutativo respecto al operador •
Por lo tanto: para A, B ϵ M

P1a) A + B = B + A
P1b) A • B = B • A

P2) Distributividad

El operador • es distributivo sobre +
El operador + es distributivo sobre •
Por lo tanto: para A, B, C ϵ M

P2a) A • (B + C) = (A • B) + (A • C)
P2b) A + (B • C) = (A + B) • (A + C)

P3) Identidad o Invariancia

Existe un elemento identidad en el conjunto M para el operador +
Existe un elemento identidad en el conjunto M para el operador •
Por lo tanto, en el álgebra booleana los elementos identidad son: 0 para la operación + y 1 para la operación •

P3a) A + 0 = A
P3b) A • 1 = A

P4) Complemento

Para cada elemento A ϵ M, existe un elemento A' ϵ M (llamado complemento de A) tal que:

P4a) A + A' = 1
P4b) A • A' = 0

El álgebra booleana se parece en algunos aspectos al álgebra ordinaria. Sin embargo, se debe tener cuidado de no sustituir las reglas del álgebra booleana por las reglas del álgebra tradicional cuando no son aplicables (por ejemplo, distributividad de la suma sobre el producto).

4) Teoremas o Propiedades (se demuestran mediante los postulados de Huntington)

T1) Idempotencia

T1a) A + A = A

Demostración:

A + A = (A + A) • 1	por P3b)	
= (A + A) • (A + A')	P4a)	
= A + (A • A')	P2b)	
= A + 0	P4b)	
= A	P3a)	l.q.q.d.

T1b) $A \cdot A = A$

Demostración:

$A \cdot A = (A \cdot A) + 0$ por P3a)
$= (A \cdot A) + (A \cdot A')$ P4b)
$= A \cdot (A + A')$ P2a)
$= A \cdot 1$ P4a)
$= A$ P3b) l.q.q.d.

T2) Existencia de elementos nulos

T2a) $A + 1 = 1$
T2b) $A \cdot 0 = 0$

T3) Involución

$(A')' = A$

T4) Absorción

T4a) $A + A \cdot B = A$
T4b) $A \cdot (A + B) = A$

T5) Asociatividad

T5a) $A + (B + C) = (A + B) + C$
T5b) $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

T6) Leyes de De Morgan

T6a) $(A + B)' = A' \cdot B'$
T6b) $(A \cdot B)' = A' + B'$

Corolarios:

$A + B = (A' \cdot B')'$
$A \cdot B = (A' + B')'$

Una función booleana es una expresión formada por variables binarias, los operadores OR, AND, NOT y el signo de igual. También puede estar presente el paréntesis y el símbolo de negación. Un ejemplo de función booleana Z = f(A,B,C) sería: $Z = A + B \cdot C$.
Para conocer el valor de Z para diferentes valores de las variables A, B y C se genera la **tabla de verdad** de la función.

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

5) Operaciones lógicas en sistemas digitales

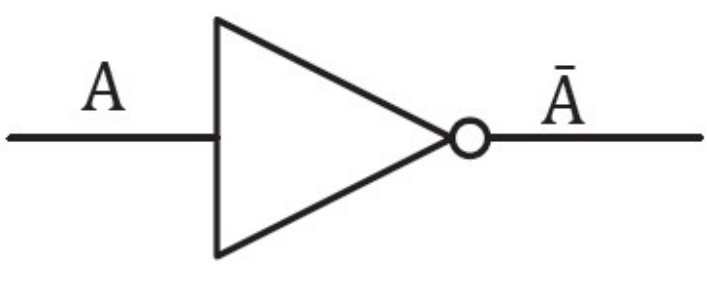
En el mundo digital existen dos maneras de operar las cifras binarias, cada cifra puede ser operada en modo aritmético o en modo lógico. El modo aritmético ya se ha presentado cuando vimos operaciones aritméticas de suma, producto y resta, más adelante veremos el hardware responsable de llevarlo a cabo, la unidad aritmética sumador. Ahora veremos el modo lógico. En un sistema digital, tanto las operaciones aritméticas como las lógicas son ejecutadas por la Unidad Aritmético Lógica (ALU).
El modo lógico se basa en las operaciones lógicas. Estas se obtienen mediante la manipulación de los patrones binarios en base a las reglas que constituyen el álgebra de Boole. El álgebra de Boole opera sobre variables que pueden ser verdaderas o falsas. Las reglas de este álgebra son actualmente utilizadas en las operaciones lógicas binarias en los sistemas digitales, ya que cada cifra binaria puede ser tratada como verdadero (nivel binario '1') y falso (nivel binario '0').
Si bien las operaciones lógicas fundamentales del Álgebra de Bolle son 3 (NOT, OR y AND), las operaciones típicas más empleadas en los sistemas digitales son las siguientes:

- NOT
- OR
- NOR
- AND
- NAND
- XOR
- XNOR

Cabe señalar que las operaciones lógicas son fundamentales en la programación de sistemas digitales, ya que permiten la implementación de una serie de operaciones vitales para el funcionamiento del programa. Por ejemplo, invertir el estado de un pin de entrada y salida de un microprocesador, configurar un registro interno, realizar operaciones de conversión de binario a BCD, o implementar el proceso de división, entre otras. Estos son ejemplos en los cuales se requiere hacer uso intensivo de las operaciones lógicas.

NOT: La operación lógica más simple de todas es la NOT (complemento, negación o inversión). Esta operación simplemente invierte un estado verdadero a falso y viceversa. En el caso binario, negar un bit significa cambiar su estado de '1' a '0' o de '0' a '1'. Por ejemplo, se tiene una variable de nombre A que representa un bit. Por lo tanto, A puede tomar 2 valores, '0' o '1'. Si se niega el valor de A, entonces $\bar{A}$ denota la negación (se lee A negado, o a veces también se denota A' como A negado). El símbolo para la compuerta lógica que simboliza la negación (inversor o negador) se muestra en la figura siguiente, junto con su tabla de verdad. Nótese el círculo en la salida que denota la inversión.

A	$\bar{A}$
0	1
1	0



También es posible realizar la operación lógica NOT sobre un conjunto de bits, que podrían ser un *nibble* (4 bits) o un *byte* (8 bits). Si la variable A equivale al siguiente valor:

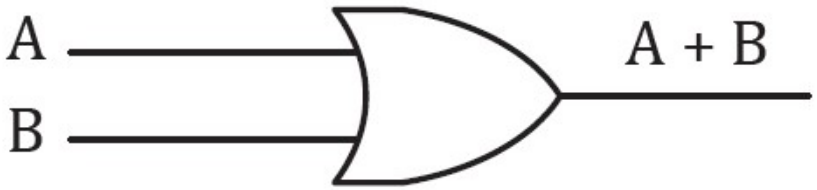
$$A = 10110011_2$$

Entonces:

$$\bar{A} = 01001100_2$$

OR: La operación lógica OR, también conocida como OR inclusiva (IOR), es otra de las operaciones básicas del álgebra de Boole. Su lógica es muy similar a la suma aritmética, excepto cuando ambos operandos son '1'. En una operación OR deben existir al menos dos operandos, y basta que uno de ellos sea '1' para que el resultado de la operación sea '1'. En el gráfico siguiente se observa la tabla de verdad y el símbolo de la compuerta lógica OR.

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

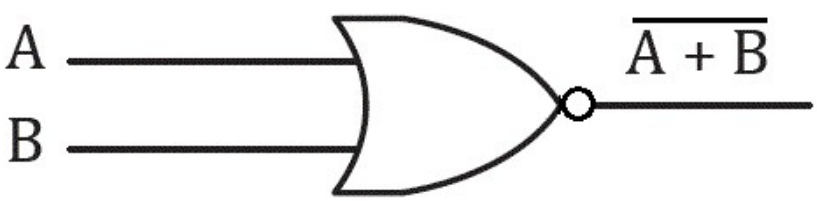


La operación OR se realiza también en conjuntos binarios. Por ejemplo, en el caso de que la variable A = 10000011_2 y la variable B = 00011111_2, el resultado de la operación OR sería el siguiente:

$$A\ OR\ B = A + B = 10000011_2 + 00011111_2 = 10011111_2$$

NOR: También es posible combinar las operaciones. Por ejemplo, se puede negar la operación OR entre A y B, lo cual se conoce como la operación NOR. En el gráfico siguiente se observa la tabla de verdad y el símbolo de la compuerta lógica NOR.

A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0



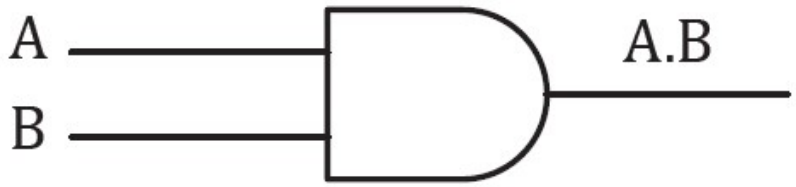
Esto aplicado a los valores de A y B dados anteriormente daría el siguiente resultado:

$$\overline{A\ OR\ B} = \overline{A+B} = \overline{10000011_2 + 00011111_2} = \overline{10011111_2} = 01100000_2$$

En muchos sistemas digitales, la operación NOR no forma parte del conjunto de instrucciones. Por lo cual, para su implementación, se deben ejecutar las operaciones OR y NOT de manera combinada.

AND: La operación lógica AND toma al menos dos variables binarias y realiza un proceso muy similar al de la multiplicación aritmética. En esta operación basta que uno de los operandos sea '0' para que el resultado de la operación sea '0'. Es decir que para que el resultado de la operación sea '1', todos los operandos deben valer '1'. En el gráfico siguiente se muestra la tabla de verdad y el símbolo de la compuerta lógica AND.

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1



En el caso de que la variable A = 11001100_2 y la variable B = 01001011_2, el resultado de la operación AND será el siguiente:

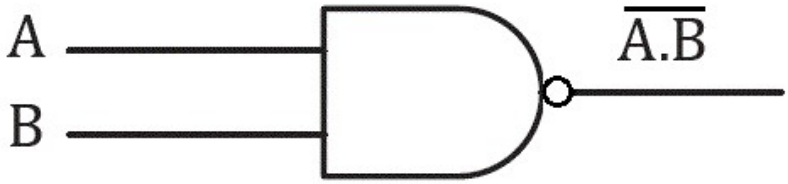
$$A\ AND\ B = A.B = 11001100_2.01001011_2 = 01001000_2$$

En el resultado se puede observar que solo aquellos bits que poseen un '1' en la misma posición de cada número mantienen su valor '1'. El resto son '0'.
También es posible agregar una tercera variable, de nombre C, a la operación. Si se tiene que C = 00001111_2, entonces la operación AND de las tres variables generará el siguiente resultado:

$$A\ AND\ B\ AND\ C = A.B.C = 11001100_2.01001011_2.00001111_2 = 00001000_2$$

NAND: También se puede combinar la operación lógica AND con la NOT para obtener un operador NAND. En el gráfico siguiente se observa la tabla de verdad y el símbolo de la compuerta lógica NAND.

A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0



Este tipo de operación tampoco es muy común en el conjunto de instrucciones de los sistemas digitales actuales. Por ejemplo, la operación NAND de las variables A y B se muestra a continuación:

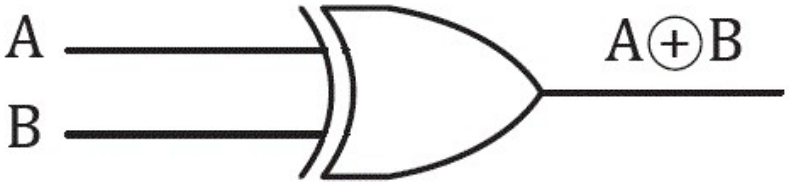
$$\overline{A\ AND\ B} = \overline{A.B} = \overline{11001100.01001011} = \overline{01001000_2} = 10110111_2$$

XOR: Hasta el momento se han analizado las tres operaciones lógicas fundamentales. Si se combinan estos tres operadores, se logrará obtener una nueva operación lógica que se encuentra disponible como compuerta lógica y también en la arquitectura de la gran mayoría de sistemas digitales: la operación XOR (OR exclusiva). Este operador genera un resultado verdadero solo si una de las entradas es verdadera y la otra es falsa. Pero si ambas entradas de la operación XOR son iguales, el resultado será falso.
Si A y B son dos variables binarias, entonces la operación XOR se denota de la siguiente manera:

$$A\ XOR\ B = A \oplus B = A.\overline{B} + \overline{A}.B$$

En el gráfico siguiente se muestra la tabla de verdad de la operación lógica XOR y el símbolo de su compuerta.

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0



La operación XOR es muy utilizada para verificar si 2 números son iguales. Por ejemplo, la variable A = 10001110_2 y la variable B = 10001110_2. Ambas variables tienen asignados el mismo código binario. Entonces, la operación XOR dará el siguiente resultado:

$$A \oplus B = 10001110_2 \oplus 10001110_2 = 00000000_2$$

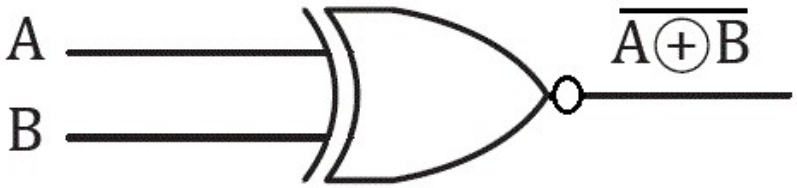
Como se puede observar, al ser todos los bits iguales en cada posición, el resultado de la operación XOR da 00000000_2. Este resultado será el mismo cada vez que los operandos sean iguales. Si, en cambio, se alterase un bit de la variable B y su nuevo valor fuera B = 10011110_2, entonces el resultado de la operación XOR sería el siguiente:

$$A \oplus B = 10001110_2 \oplus 10011110_2 = 00010000_2$$

El valor obtenido es diferente de 00000000_2, lo cual será siempre así si es que ambos operandos no son iguales.

XNOR: De la misma manera que con las operaciones OR y AND, también podemos combinar la operación lógica XOR con la NOT formando la operación XNOR. En el gráfico siguiente se muestra la tabla de verdad de la operación lógica XNOR y el símbolo de su compuerta.

A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1



6) Pseudo operaciones lógicas en sistemas digitales

Muchos sistemas digitales realizan una combinación de operaciones lógicas básicas para modificar el contenido de un byte (8 bits). Un byte está compuesto por 2 *nibbles* (4 bits).
Presentaremos dos pseudo operaciones que son muy utilizadas en la mayoría de los sistemas digitales:

- SWAP
- SHIFT

La operación **SWAP** no posee un símbolo lógico. Si bien es cierto, no es una operación lógica propiamente dicha sino el resultado de la combinación de operaciones básicas, muchos sistemas digitales utilizan una instrucción de nombre *swap* para alterar el contenido de un byte de información.
En la operación *swap*, los *nibbles* de un byte son intercambiados de posición. Con una variable A = 11000101_2, se observa que el *nibble* más significativo es 1100_2, mientras que el *nibble* menos significativo es 0101_2. Si se realiza la operación *swap* sobre A, el nuevo

resultado será A = 01011100_2. Como se puede constatar, simplemente se han intercambiado de posición los *nibbles* del byte.

Otra de las pseudo operaciones clásicas realizadas por todo sistema digital es la operación **SHIFT** o de desplazamiento binario. En una cifra binaria, cada bit posee una posición. Con el operador *shift* es posible realizar un cambio de posición de cada bit, ya sea para la derecha o para la izquierda.

Nuevamente, la operación *shift* no es una compuerta, sino un conjunto de operaciones que involucran el movimiento de los bits en un registro. Esta operación está incluida en el conjunto de instrucciones de todo sistema digital y resulta muy útil para una gama de aplicaciones en las cuales se requiere multiplicar, dividir o convertir de binario a BCD, entre otras.

En la operación *shift* se utilizan los símbolos >> o <<, para denotar un corrimiento binario hacia la derecha o hacia la izquierda, respectivamente. Por ejemplo, si se tiene que la variable A = 10011000_2, entonces, al realizar la operación A>>3, los bits de la variable A se desplazarán tres posiciones a la derecha, quedando de la siguiente manera: A = 00010011_2. Al realizar esta operación han aparecido tres bits de valor cero (’000’) al lado izquierdo del bit que era el MSB de la cifra. Estas tres posiciones en realidad son reemplazadas por bits fuera de las ocho posiciones del byte. Por defecto, cuando se realiza el corrimiento, estos nuevos valores son siempre ceros.

Ahora, en el caso de que la variable A = 00010011_2 sufre un corrimiento a la izquierda con la siguiente operación: A<<4, el resultado quedaría de la siguiente forma: A = 00110000_2. En este caso se puede observar que uno de los bits ‘1’ de la variable A ha desaparecido (el bit en la posición 4). Esto en realidad no ha ocurrido, sino que este bit ha tomado la posición 8 o se ha convertido en el noveno bit, el cual no se muestra, ya que el valor a tratar es un byte y solo tiene 8 posiciones (la novena posición, verdaderamente, solo estará en la mente del programador).