Operaciones lógicas (Lógica booleana) y Bit Shifting

Publicado el El Informatico - 16 de enero de 2022 -

Como en el sistema de numeración binario cada dígito sólo puede tener dos estados únicos, podemos usar otro tipo de aritmética para realizar cálculos lógicos con valores binarios, usando lo que denominamos **operaciones lógicas** (También llamadas **booleanas**, en honor a **George Boole**). En éste tipo de operaciones utilizamos operaciones lógicas en lugar de aritméticas para calcular un resultado que puede ser «verdadero», o «falso» (O 1 o 0). Este tipo de operaciones son muy comunes tanto a la hora de programar, como a la hora de elaborar y analizar circuitos electrónicos digitales.

Introducción: Hoy vas a ser operario de un panel de control de un silo nuclear

Lo siento, pero no se me ocurría una manera menos agresiva de introducir las operaciones lógicas. Fuente: Wikipedia (CC-BY SA)

Para iniciar el lanzamiento de los misiles en los centros de control de los silos nucleares, además de una serie de contraseñas que hay que introducir para validar el lanzamiento, suele haber dos paneles con un cerrojo cada uno, separados por una distancia de tal modo que una persona sola no podría activar los dos cerrojos a la vez. Son necesarias dos personas para poder activar el lanzamiento.

Para hacerlo, ambos operarios (y tú estas ahora en el equipo), deben introducir y girar la llave al mísmo tiempo. De no realizar ésta maniobra de manera correcta, el lanzamiento se suspende. Para éste ejemplo, vamos a llamar a los dos paneles el Panel A, y el Panel B.

En otras palabras, para iniciar el lanzamiento, tanto en el Panel A como en el Panel B la llave de inicio debe estar girada a la vez. Si extrapolamos ésto de forma electrónica, **al girar cada llave el interruptor de cada cerrojo estaría a 1 (Y a 0 cuando no es el caso)**, siendo cada llave las entradas en el circuito. De modo que si giramos ambas llaves a la vez, tanto el cerrojo del Panel A como el del Panel B tendrían su valor a 1. En éste caso, **el circuito que lo controla tendría a su salida un 1 sólo cuando el valor de la llave del Panel A Y el Panel B sean 1**.

Tabla de verdad

Para ilustrar de mejor manera lo que estamos haciendo, podemos usar una **tabla de verdad**. En la tabla de verdad colocamos varias columnas representando las entradas y sus distintos posibles valores, y una columna a la derecha representando

el resultado en función del estado de dichas entradas. Recuerda que en éste caso, cada panel tiene su valor a 1 sólo si se ha girado la llave en el correspondiente cerrojo.

PANEL A	PANEL B	SALIDA
0	0	0
0	1	0
1	0	0
1	1	1

Tabla de verdad

Asumiendo que ya tengamos las contraseñas introducidas y hayamos introducido las coordenadas del lanzamiento, podemos comprobar que si la llave no se ha girado en ninguno de los paneles, el lanzamiento no se inicia. Tampoco lo hará si sólo giramos una de las llaves en un panel. Sin embargo, al girar ambas llaves, la salida del circuito de control se pone a 1. Y esto es una forma de explicar mediante electrónica digital, que has iniciado la tercera guerra mundial (además de iniciarte en operaciones lógicas).

Operaciones lógicas: AND (X, * o &)

La primera operación que voy a introducir es la operación AND (Y). Es la que acabo de explicar en el ejemplo del lanzamiento de los misiles. **En la operación AND, la salida es 1 sólo cuando todas y cada una de las entradas tiene valor 1** (1 & 1 = 1). Por éste motivo, se puede representar con el símbolo de multiplicación (X o *), ya que si multiplicas el valor de las entradas el resultado sólo será 1 si todas las entradas estan a 1, ya que algo multiplicado por 0 será siempre 0. Pero lo más común es que se represente con el símbolo 'and' (&), por su nombre.

La tabla de verdad que la representa es la mísma que he expuesto en el ejemplo, aunque la vuelvo a exponer por redundancia. En éste caso, llamaré a las entradas A y B.

A	B	SALIDA
0	0	0
0	1	0
1	0	0
1	1	1

Tabla de verdad de la operación lógica AND

Aunque en éstos ejemplos estoy usando dos entradas y una salida a modo de simplificarlos, en la práctica se pueden usar tantas entradas y salidas como necesites representar.

OR (+, |, ·)

En la operación O (Or) , **la salida será un 1 siempre y cuando al menos una de las entradas tenga su valor a 1**. La tabla de verdad que la representa es la siguiente:

A	B	SALIDA
0	0	0
0	1	1
1	0	1
1	1	1

Tabla de verdad de la operación lógica OR

La operación que lo representa es la suma (+), ya que en éste caso, cualquier cosa sumada a 1 es 1 (0 + 0 = 1, 0 + 1 = 1, etc). En informática, normalmente lo representamos como una barra vertical (|). Pero su símbolo matemático es '·'.

XOR (⊕)

En la operación OR eXclusiva (eXclussive OR, o XOR), la salida sólo es 1 sólo cuando una y sólo una de las entradas está a 1. La tabla de verdad que la representa es la siguiente:

A	B	SALIDA
0	0	0
0	1	1
1	0	1
1	1	0

Tabla de verdad de la operación lógica XOR

La operación XOR se representa como una *disyunción exclusiva*, que se representa como una cruz dentro de un círculo ( ). Es decir, 0 0 = 0, 0 1 = 1, etc. En informática, normalmente lo representamos como una intercalación (^).

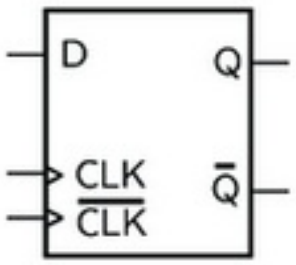
NOT (!, ¬)

La operación de negación (not) invierte cualquier valor a la entrada. Es decir, lo que es un 0, pasa a ser un 1. Y lo que es un 1, pasa a ser un 0.

A	!A
0	1
1	0

Tabla de verdad de la operación lógica NOT

El símbolo matemático para la negación es '¬'. En informática, normalmente usamos el símbolo de exclamación cerrada (!) delante de otro símbolo para indicar que su valor está invertido. Pero en electrónica (y en el álgebra de boole) es comun usar una línea sobre el símbolo para indicar que la entrada o la salida (o un valor) está invertida (Por ejemplo, $\bar{Q}$). En ocasiones es también posible que se represente mediante una comilla símple (').



Símbolo electrónico de un Biestable D (D latch). Tiene dos salidas, un bit (Q), y su valor invertido ($\bar{Q}$). La señal de reloj (CLK) también ofrece una entrada que invierte su valor.

Operaciones invertidas: NAND (.)

La operación NAND es la inversa de la operación AND (!&, o NOT AND). Si en la operación AND la salida es 1 sólo cuando todas las entradas están a 1, en la NAND la salida será 1 cuando no todas las entradas estén a 1.

A	B	SALIDA
0	0	1
0	1	1
1	0	1
1	1	0

Tabla de verdad de la operación lógica NAND

Se representa matemáticamente mediante un punto (.), pero en informática usamos el conjunto de símbolos ! y &. Por ejemplo, !(A & B).

NOR (□)

La operación NOR es la inversa de la operación OR (o NOT OR). Si en la operación OR la salida era 1 cuando al menos una de las entradas está a 1, en éste caso sólo sera 1 cuando ambas entradas estén a 0.

A	B	SALIDA
0	0	1
0	1	0
1	0	0
1	1	0

Tabla de verdad de la operación lógica NOR

Exísten numerosas notaciones para ésta operación. La estandard es la flecha de Peirce (Peirce arrow), que es una flecha hacia abajo ( ). Otras formas de representarla incluyen el uso de los símbolos matemáticos ¬ (NOT) y ⋅ (OR). Por ejemplo, ¬(A ⋅ B). En programación, usamos los símbolos ! y |. Por ejemplo, !(A | B).

XNOR (□)

La operación OR eXclusiva negada (XNOR, o eXclussive NOT OR,) es la inversa de la operación XOR. Si en la XOR la salida era 1 sólo cuando una única entrada estaba a 1,

en éste caso la salida será 1 cuando ambas entradas estén a 0, o a 1.

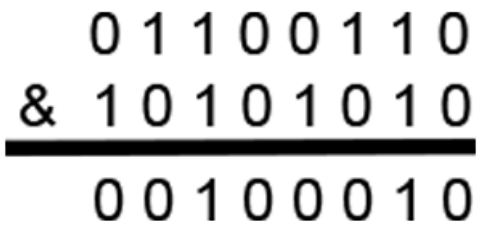
A	B	SALIDA
0	0	1
0	1	0
1	0	0
1	1	1

Tabla de verdad de la operación lógica XNOR

El símbolo usado es un punto dentro de un círculo ( ). En informática usamos los símbolos ! y ^. Por ejemplo, !(A ^ B).

Aritmética lógica

Cuando queremos realizar operaciones lógicas entre grupos de bits, las operaciones se realizan bit a bit, en el orden de los bits (De LSB a MSB, o lo que es lo mísmo, de derecha a izquierda). Por ejemplo, si queremos realizar la operación AND entre los bytes 01100110 y 10101010:



Ejemplo de operación AND entre dos bytes (001100110 y 10101010).

Bit Shifting

La operación de *bit shifting* consiste en desplazar los bits en un conjunto de bits en una dirección específica. Hay cuatro operaciones básicas para hacer bit shifting, dos para desplazar bits (Left Bit Shift y Right Bit Shift), y otras dos para rotar los bits (Left Circular Shift y Right Circular Shift).

En los subsiguientes ejemplos tomaremos un byte (conjunto de 8 bits) para explicar los desplazamientos.

> **Nota**: El término 'bitwise' significa 'en el orden de los bits'. El órden de los bits es siempre de derecha a izquierda, siendo el bit más a la derecha el de menor peso (LSB, Less Significant Bit), y el más a la izquierda el de mayor peso (MSB, Most Significant Bit). Esto es porque el bit más a la izquierda tiene el mayor

> valor. En un byte, el octavo bit (MSB) tiene un valor de 128, mientras que el primero (LSB) tiene un valor de 1.

Left Bit Shift

Left Bit Shift desplaza los bits **hacia la izquierda**, tantas veces como se indique. Esta operación se representa mediante el símbolo '<<' (dos símbolos 'inferior a').

Durante el desplazamiento, los bit de menor peso (LSB) son reemplazados por ceros, mientras que los de mayor peso (MSB) que quedan fuera durante el desplazamiento, desaparecen.

Ejemplos:

```
00000001 << 1  ->  00000010
00000101 << 1  ->  00001010
01001101 << 1  ->  10011010
01001101 << 3  ->  01101000
11111111 << 2  ->  11111100
```

Esta operación es el equivalente a **multiplicar en potencias de 2**. Por ejemplo, 5 (00000101) << 1 = 5 * 2^1 = 5 * 2 = 10 (00001010).

Right Bit Shift

Right Bit Shift es la operación inversa de Left Bit Shift. En éste caso, los bits son desplazados **hacia la derecha** tantas veces como se indique. Esta operación se representa mediante el símbolo '>>' (dos símbolos 'superior a').

Durante el desplazamiento, los bits de mayor peso (MSB) son reemplazados por ceros, mientras que los de menor peso (LSB) que quedan fuera durante el desplazamiento, desaparecen.

Ejemplos:

```
10000000 >> 1  ->  01000000
11001101 >> 1  ->  01100110
01010101 >> 3  ->  00001010
11111111 >> 3  ->  00011111
```

Esta operación es el equivalente a **dividir en potencias de 2**. Por ejemplo, 128 (10000000) >> 1 = 128 / 2^1 = 128 / 2 = 64 (01000000).

Logical Bit Shift y Arithmetical Bit Shift

Por norma general, y ésto es aplicable cuando por ejemplo estamos programando, **los bits desplazados son reemplazados por ceros** tal y como he explicado. En

éste caso hablamos de un desplazamiento lógico (**Logical Bit Shift**). Esta operación es óptima para **números sin signo**.

No obstante, **algunos componentes digitales y algunas instrucciones del procesador podrían hacer que los bits desplazados sean reemplazados por unos en lugar de ceros**. En éste caso hablaríamos de un desplazamiento aritmético (**Arithmetical Bit Shift**). Esta operación es optima para números con signo, con el fin de preservar el bit de signo.

Cuando el desplazamiento es aritmético y no lógico, se usan trés símbolos de superior (>>>) o inferior a (<<<).

Ejemplos:

```
00000001 <<< 1  ->  00000011
00000001 >>> 1  ->  10000000
11001100 >>> 1  ->  11100110
00000000 <<< 4  ->  00001111
11111111 <<< 3  ->  11111111
```

Left Circular Shift

Una operación Left Circular Shift hace que los bits hagan un ciclo de derecha a izquierda. Es decir, los bits que salen fuera durante el desplazamiento (En éste caso serían los MSB) son colocados de vuelta al otro lado del conjunto de bits (En el lado de los LSB). Aunque la operación es distinta a un Left Bit Shift, se representa con el mísmo símbolo ('<<').

Ejemplos:

```
10000001 << 1  ->  00000011
10000011 << 1  ->  00000111
00000011 << 1  ->  00000110
11001100 << 2  ->  00110011
11111110 << 4  ->  11101111
```

Right Circular Shift

Una operación Right Circular Shift realiza la operación inversa al Left Circular Shift. Desplaza los bits de izquierda (MSB) a derecha (LSB), y mueve los bits que quedan fuera por el lado de los LSB de vuelta al lado de los MSB. Se representa con el símbolo ('>>') igual que la operación Right Bit Shift.

Ejemplos:

```
10000001 >> 1  ->  11000000
11000000 >> 1  ->  01100000
11000001 >> 1  ->  11100000
11001100 >> 2  ->  00110011
11111110 >> 3  ->  11011111
```

ANTERIOR

Historia de los medios de almacenamiento digitales

SIGUIENTE

La memoria del ordenador, a fondo

Entradas Recientes

Categorías

RSS

Subscribirse al feed RSS