Electrónica Digital

Ingeniería Informática – FICH, UNL
Leonardo Giovanini

Circuitos combinacionales

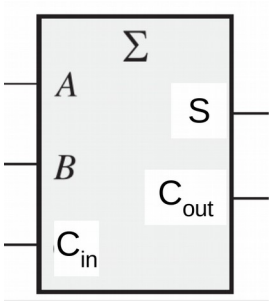
Operaciones Aritméticas y Lógicas

Organización de la clase

En esta se estudiarán los siguientes temas:

- Operaciones aritméticas: Suma;
  Resta;
  Multiplicación;
  Comparación;
- Operaciones lógicas: Operaciones bit a bit (Suma, Producto, Negación);
  Paridad;
  Rotaciones;
  Desplazamientos.

La ***suma*** es la operación aritmética que se realiza con ***mayor frecuencia***. Los sumadores son muy importantes para los sistemas digitales en los que se procesan datos numéricos.



Un sumador combina tres operandos aritméticos (*A*, *B* y C_{in}) que utilizan las reglas de la suma aritmética y genera dos salidas: un bit de suma (S) y un bit de acarreo (C).

El comportamiento del sumador básico está decripto por la siguiente tabla de verdad

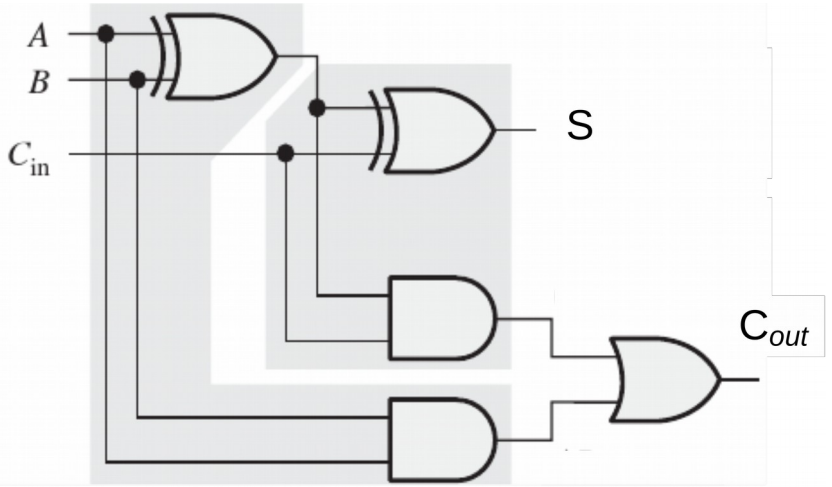
A	*B*	C_{in}	C_{out}	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Las ecuaciones lógicas del sistema están dadas por

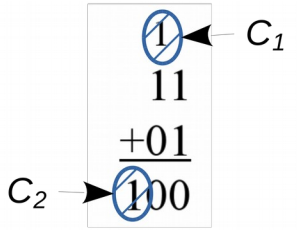
$$S=\overline{A}B\,C_{in}+A\overline{B}\,C_{in}+\overline{A}B\,\overline{C}_{in}+A\overline{B}\,\overline{C}_{in}$$

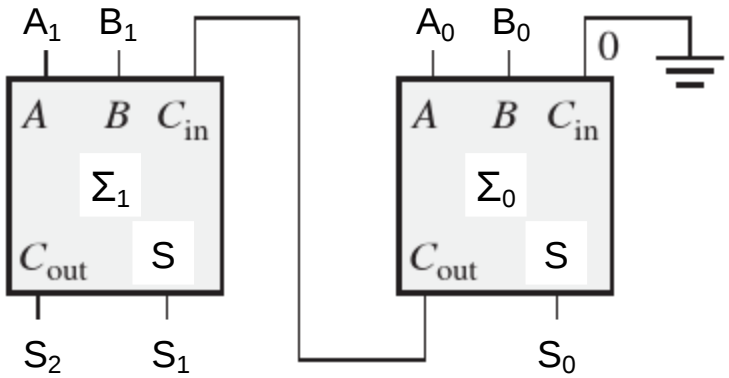
$$C_{out}=AB+(\overline{A}B+A\overline{B})C_{in},$$

cuya implementación está dada por



Para sumar números de dos bits se necesitan dos sumadores. La salida de acarreo de cada sumador se conecta a la entrada de acarreo del sumador siguiente. La entrada de acarreo C_{in} del bit menos significativo se pone a 0. La implementación circuital de la suma aritmética es

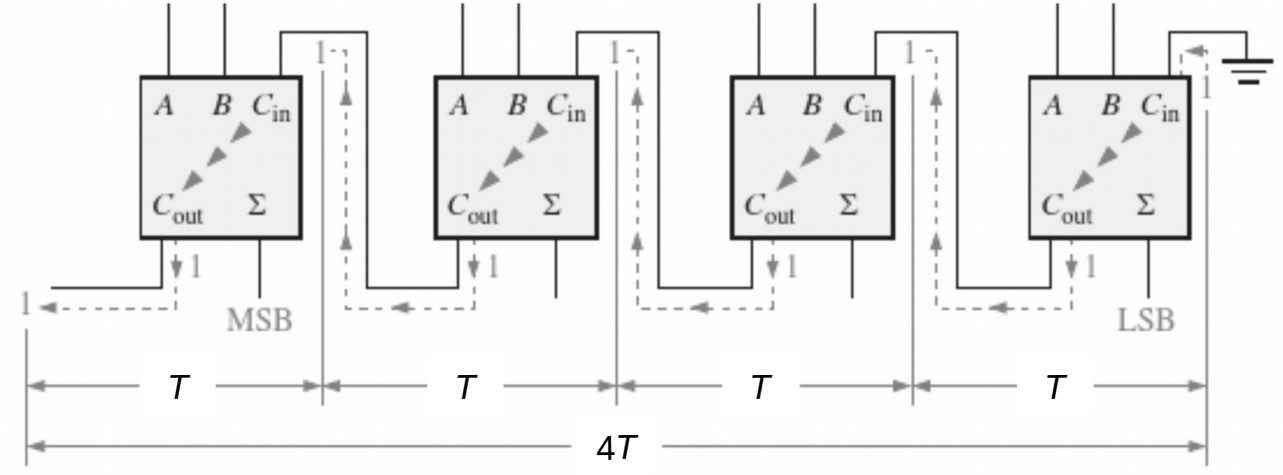




Para sumar dos números binarios con n dígitos se necesitan n sumadores completos, uno por cada bit que tengan los números que se quieren sumar.

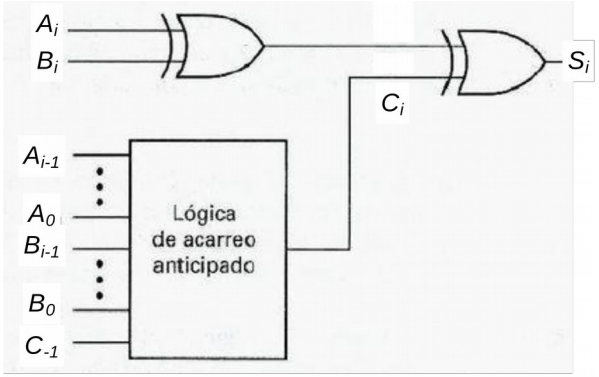
Los sumadores pueden clasificarse en dos categorías dependiendo de la forma en que se manejan los acarreos internos de una etapa a otra. Estas categorías son:

- **Acarreo serie** - Un sumador de acarreo serie es aquel en el que la salida de acarreo de cada sumador completo se conecta a la entrada de acarreo de la siguiente etapa de orden inmediatamente superior. La suma y el acarreo de salida de cualquier etapa no se pueden generar hasta que tiene lugar el acarreo de entrada, lo que da lugar a un retardo temporal en el proceso de adición.



El retardo de propagación del acarreo para cada sumador completo es el tiempo transcurrido desde la aplicación del acarreo de entrada hasta que se produce el acarreo de salida, suponiendo que las entradas se aplican al mismo tiempo.

Acarreo acarreo - El sumador con ***acarreo anticipado*** predice el acarreo de salida de cada etapa, en función de los bits de entrada de cada etapa, utilizando un circuito combinacional externo al sumador.



El acarreo de salida del sumador i se puede escribir

$$C_{\text{out}} = C_g + C_p C_{\text{in}}$$

donde

$$C_g = AB$$
$$C_p = A + B$$

Para el primer sumador tenemos $C_{\text{out1}} = C_{g1} + C_{p1} C_{\text{in1}}$

De manera similar, para los siguientes tenemos

$$C_{\text{out2}} = C_{g2} + C_{p2} C_{in2} = C_{g2} + C_{p2} C_{\text{out1}} = C_{g2} + C_{p2} C_{g1} + C_{p2} C_{p1} C_{\text{in1}}$$

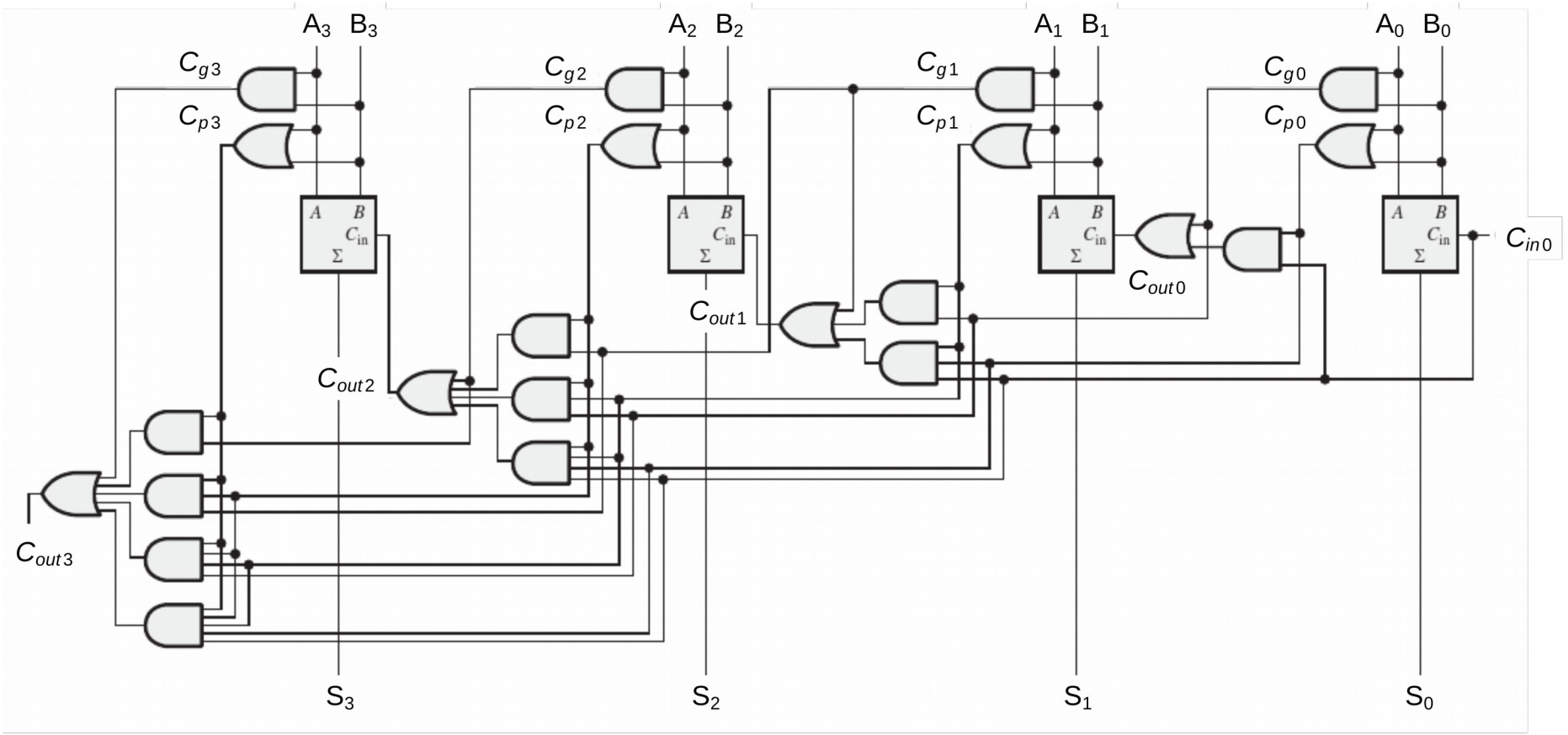
$$C_{\text{out3}} = C_{g3} + C_{p3} C_{in3} = C_{g3} + C_{p3} C_{\text{out2}} = C_{g3} + C_{p3} C_{g2} + C_{p3} C_{p2} C_{g1} + C_{p3} C_{p2} C_{p1} C_{\text{in1}}$$

$$C_{\text{out4}} = C_{g4} + C_{p4} C_{in4} = C_{g4} + C_{p4} C_{\text{out3}} = C_{g4} + C_{p4} C_{g3} + C_{p4} C_{p3} C_{g2} + C_{p4} C_{p3} C_{p2} C_{g1} + C_{p4} C_{p3} C_{p2} C_{p1} C_{\text{in1}}$$

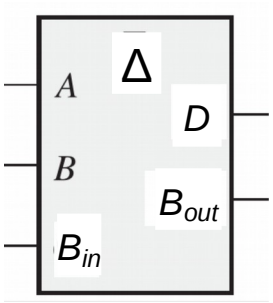
El acarreo de salida de cada etapa sólo depende del acarreo de entrada inicial (C_{in1}), las funciones C_g y C_p de la etapa y las anteriores. Como C_g y C_p pueden expresarse en función de las entradas, todos los acarreos de salida están inmediatamente disponibles y no es necesario esperar a que se propaguen los acarreo a través de todas la etapas antes de obtener el resultado final.

Operaciones Aritméticas – *Suma*

Sumador de acarreo anticipado



La otra operación aritmética que se realiza con mucha frecuencia es la ***resta***. Los restadores son muy importantes para los sistemas digitales en los que se procesan datos numéricos.



Un restador combina tres operandos aritméticos (*A*, *B* y B_{in}) que utilizan las reglas de la resta y genera dos salidas: un bit de resta (*D*) y un bit de prestamo (B_{out}).

El comportamiento del restador básico está decripto por la siguiente tabla de verdad

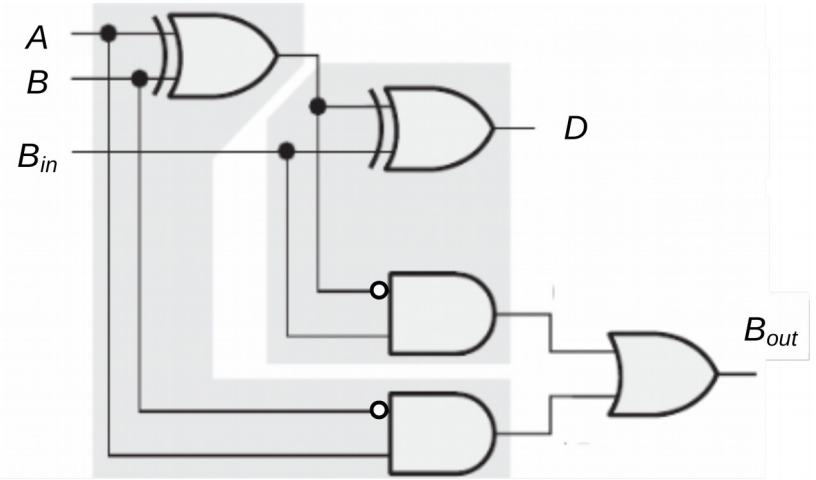
A	*B*	B_{in}	B_{out}	*D*
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Las ecuaciones lógicas del sistema están dadas por

$$D = \overline{A}B\,B_{in} + A\overline{B}\,B_{in} + \overline{A}B\,\overline{B}_{in} + A\overline{B}\,\overline{B}_{in}$$

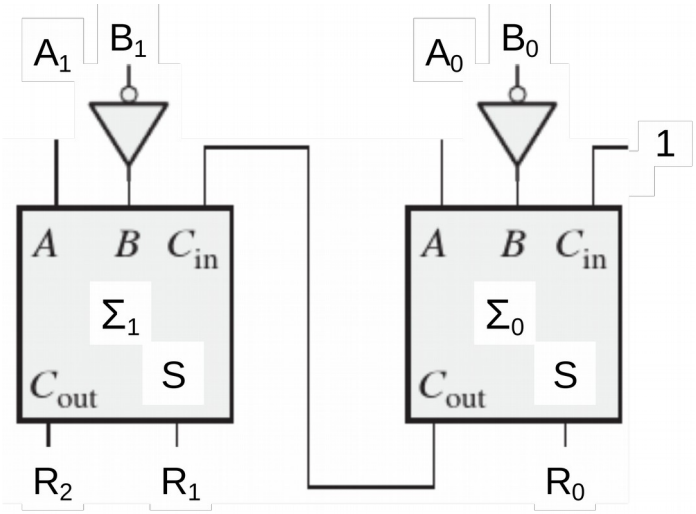
$$B_{out} = AB + (\overline{A}B + A\overline{B})B_{in},$$

cuya implementación está dada por



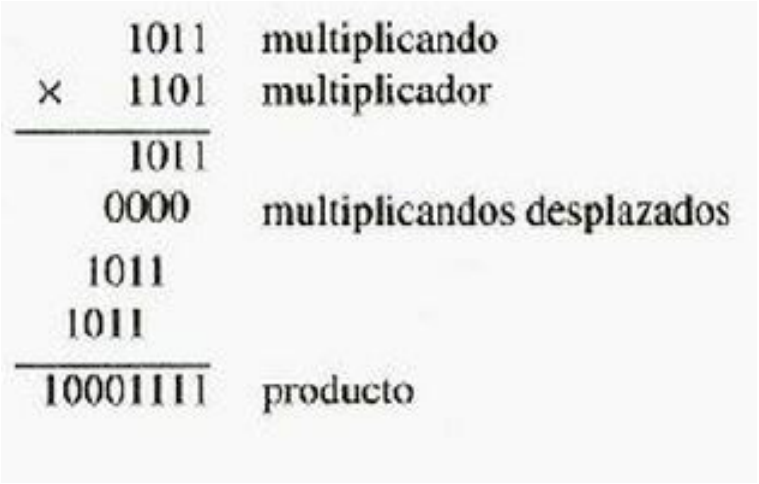
La ***resta*** también se puede implementar a través de la suma del minuendo y el negativo del sustraendo. Es decir que en lugar de desarrollar el circuito combinacional para la resta, primero calcularemos el negativo del sustraendo (operando B del sumador) y luego se lo sumamos al minuendo (operando A del sumador). De esta manera se implifica el diseño de los circuitos restadores.

El negativo de un número binario se calcula como su complemento a 2, es decir el complemento a 1 (negando individualmente los bits) y luego sumandole uno al número resultante.



Es decir, cualquier circuito sumador de n bits puede transformarse en un circuito restador al calcular el complemento a dos del sustraendo y tratando a los acarreos como prestamos (Borrow).

La ***multiplicación*** entre números binarios se puede realizar utilizando el mismo método que utilizamos cuando realizamos una multiplicación con números decimales, mediante la ***suma de multiplicandos trasladados***. La formación de multipicandos trasladados es trivial en la multiplicación binaria, ya que hay solo dos valores posibles : 0 ó 1.



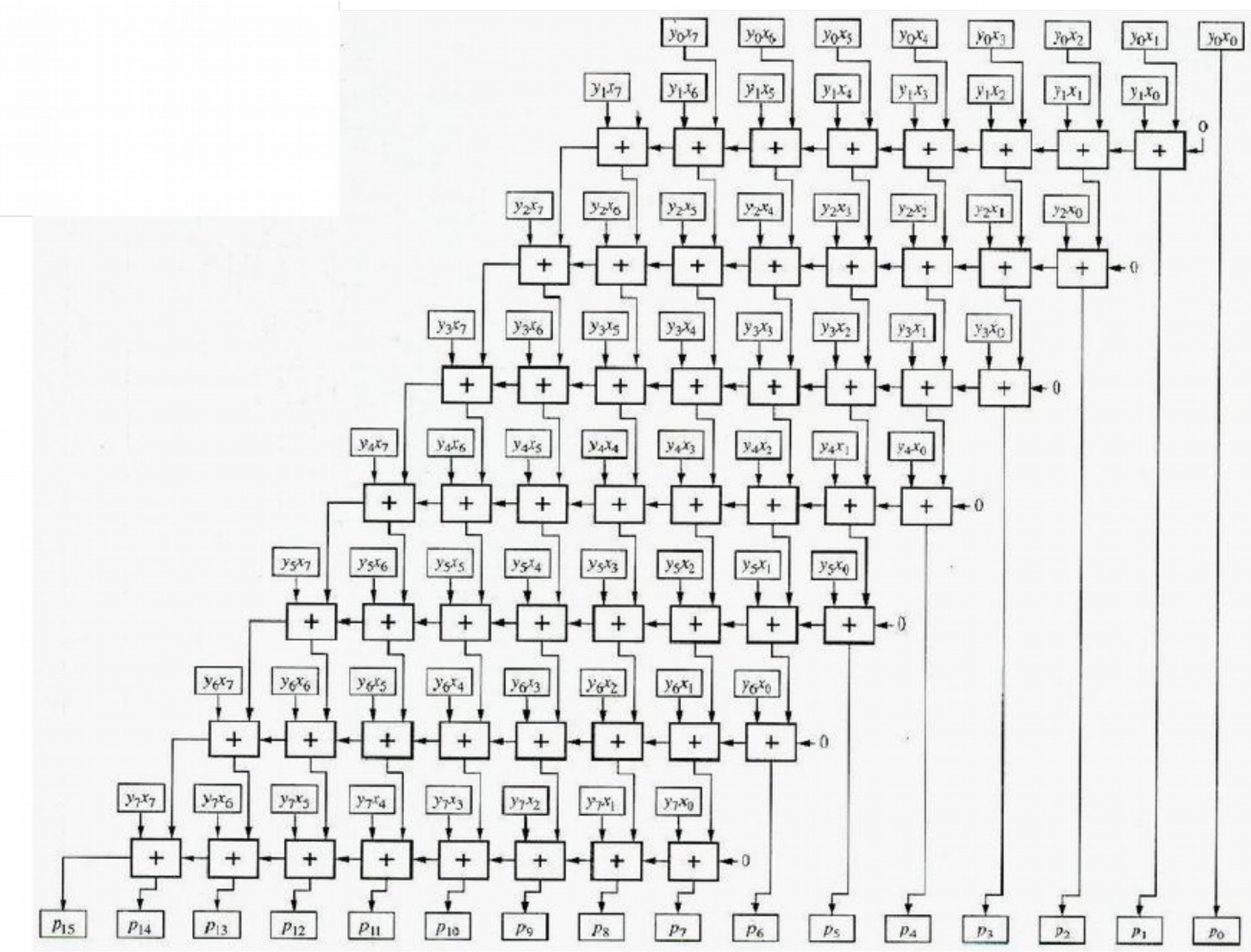
La ***multiplicación de números con signo*** puede realizarse utilizando una multiplicación sin signo de magnitudes y luego determinar el signo a partir de la regla de los signos.

Cuando ***multiplicamos*** un número de n bits por un número de m bits se obtiene, como máximo, un producto $n+m$. El algoritmo de desplazamiento y suma necesita m productos y sumas parciales para obtener el producto. Al mismo tiempo, cada paso produce un bit de producto parcial adicional.

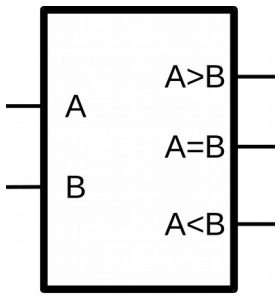
El algoritmo de desplazamiento y sume puede realizarse mediante un circuito digital que incluye un registro de desplazamiento, un sumador y una lógica de control. Sin embargo, en esta clase lo haremos con un circuito combinacional.

Operaciones Aritméticas – *Multiplicación*

Multiplicador binario de 8 bits



La ***comparación*** es una operación que se realiza con mucha frecuencia en sistemas digitales y computadoras que se procesan datos.



Un comparador combina dos operandos (*A* y *B*) y genera tres salidas que indican la relación entre los operandos: A<B, A=B y A>B.

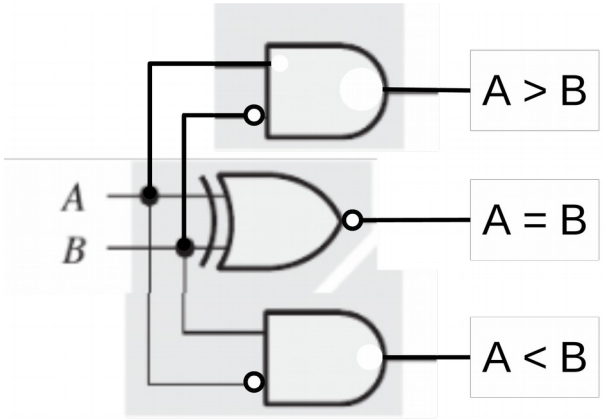
El comportamiento del sumador básico está decripto por la siguiente tabla de verdad

A	*B*	A<B	*A=B*	*A>B*
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Las ecuaciones lógicas del sistema están dadas por

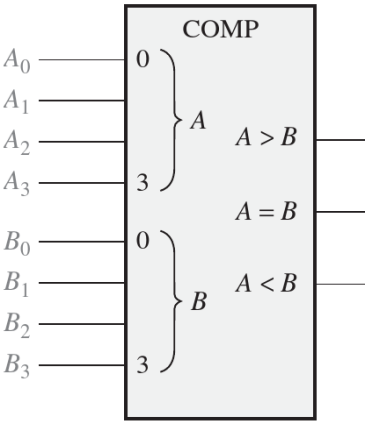
$$Lt = \overline{A}B,$$
$$I = AB + AB$$
$$Gt = A\overline{B},$$

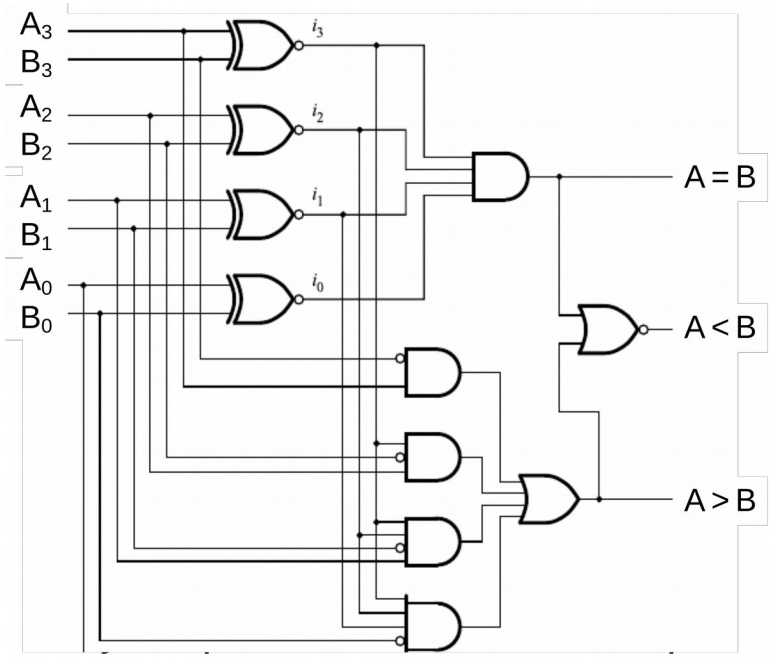
cuya implementación está dada por



La ***comparación*** es una operación que se realiza con mucha frecuencia en sistemas digitales, computadoras e interfaces de dispositivos.

A un circuito combinacionales que comparan ***palabras binarias y solo indica si son iguales***, se lo denomina ***comparador***.





Si lo circuitos interpretan sus entradas como ***números e indica todas las relaciones*** (A<B, A=B y A>B), se los denomina ***comparador de magnitud***.

Al igual que la suma y resta, los comparadores se pueden construir a partir de comparadores de bits individuales en cascada.

Otra opción es diseñar circuitos un específico a partir de la comparación de igualdad.

Operaciones bit a bit

Una ***operación bit a bit*** (*bitwise*) opera sobre datos binarios a nivel de sus bits individuales.

Es una acción simple y directa que se utiliza para manipular lo datos para cálculos y comparaciones. Las operaciones bit a bit se utilizan para setear, resetear, dejar pasar, eliminar o invertir bits individuales o en conjunto, usando la máscara adecuada.

Estas operaciones se clasifican en

- ***Operaciones lógicas*** - ejecutan las operaciones lógicas AND, OR, XOR y NOT (entre otras) sobre los bits individuales de los operandos;

- ***Operaciones de desplazamiento -*** desplazan los bits de los operandos hacia la derecha o hacia la izquierda una o más posiciones agregando ceros en las posiciones iniciales; y

- ***Operaciones de rotación*** - rotan los bits del operando hacia la derecha o hacia la izquierda una o más posiciones, sin perder información. Se puede utilizar un bit adicional en la rotación o no.

Operaciones bit a bit – *Operaciones Lógicas*

Las operaciones bit a bit lógicas toman dos enteros y realiza la ***operación lógica correspondiente en cada par de bits***.
Las operaciones lógicas más comunes son:

- **NOT** - realiza la negación lógica en cada bit, invirtiendo los bits individuales.
  El NOT forma el complemento a uno del dato binario. Por ejemplo

```
NOT 10101011
  = 01010100
```

- **AND** - realiza el producto lógico (AND) en cada par de bits. Esta operación se usa para:
  - ***Filtrar bits***, permitiendo que unos bits mantegan su estado y otros no; y
  - ***Determinar el estado*** de bits uzando mascaras que seleccionen los bits.

```
  0101
AND 0011
  = 0001
```

- **OR** - realiza la suma lógica (OR) en cada par de bits. Esta operación se usa para:
  - **Encender** bits individuales o conjuntos usando una máscara con los bits que se quieren encender en 1 y el resto de los bits en cero.

```
  0101
OR 0011
 = 0111
```

- **XOR** - realiza la OR Exclusiva (XOR) en cada par de bits. Esta operación se usa para:
  - **Encender** bits individuales o conjuntos usando una máscara con los bits que se quieren encender en 1 y el resto de los bits en cero.

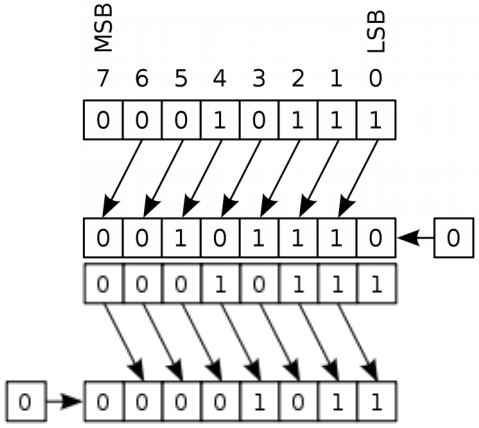
```
  0101
XOR 0011
  = 0110
```

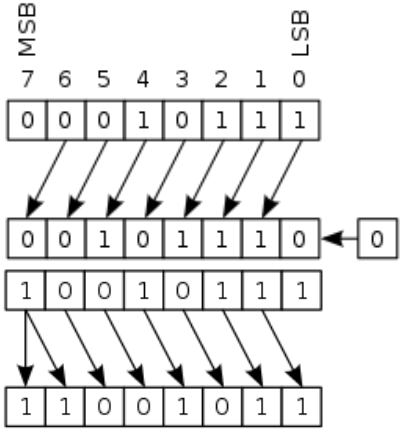
Estas operaciones se implementan a través de ***una compuerta por bit de dato*** procesado.

Operaciones bit a bit – *Desplazamientos*

Los ***desplazamientos*** son operaciones donde los bits son desplazados, hacia la izquierda o hacia la derecha, ingresando y sacando bits de la palabra. Las diferencias entre estos operadores está en cómo se ***determinan los valores de los bits que entran*** a la palabra.

Desplazamiento lógico - los bits de un registro son desplazados una o más posiciones. Los bit que salen del registro por un extremo se pierden y en el otro extremo del registro se rellena con un bit cero por cada bit desplazado. Hay dos desplazamientos lógicos: hacia la izquierda y hacia la derecha.

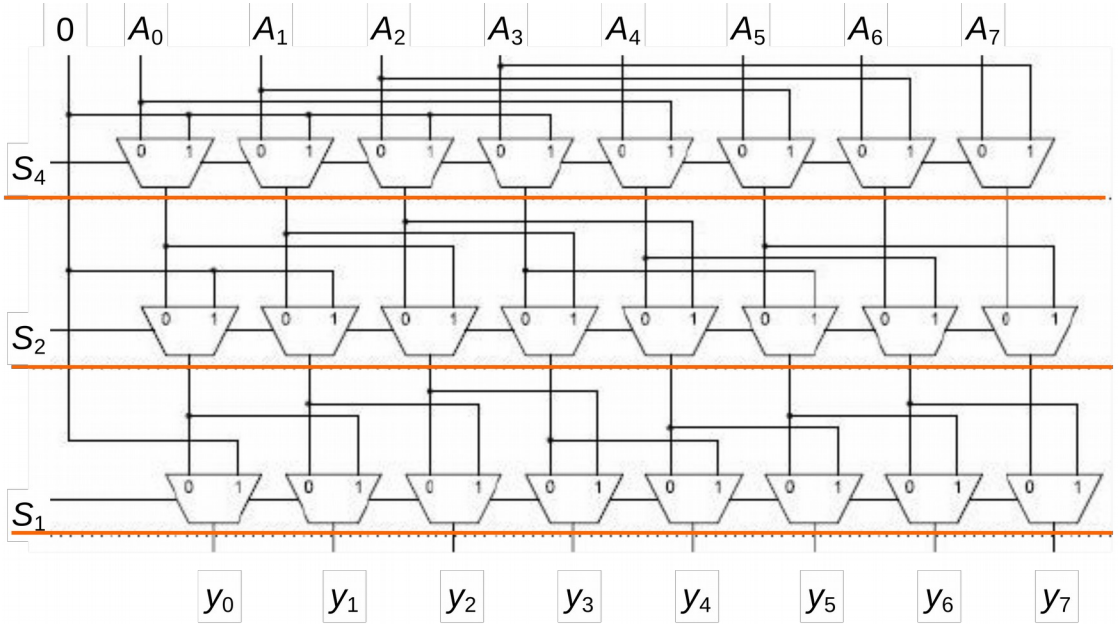


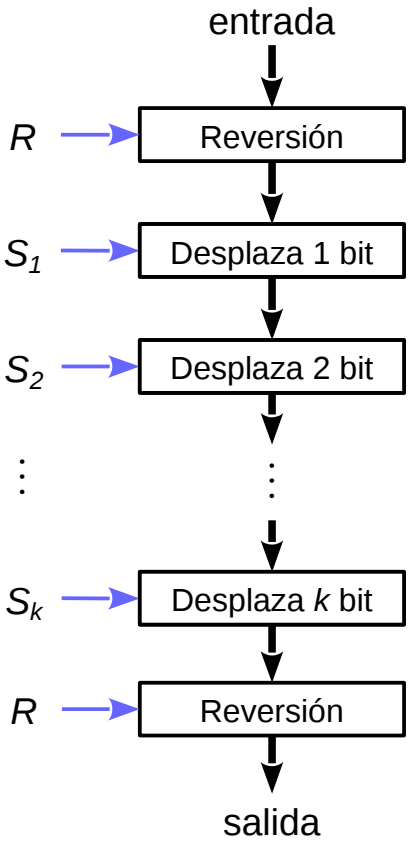


Desplazamiento aritmético - son similares a los desplazamientos lógicos pero están pensados para trabajar con números enteros con signo en representación de ***complemento a dos***. Los desplazamientos aritméticos permiten la multiplicación y la división por dos por una potencia de dos: desplazar *n* bits hacia la izquierda o a la derecha equivale a multiplicar o dividir por 2^n, respectivamente.

Operaciones bit a bit – *Desplazamientos lógicos*

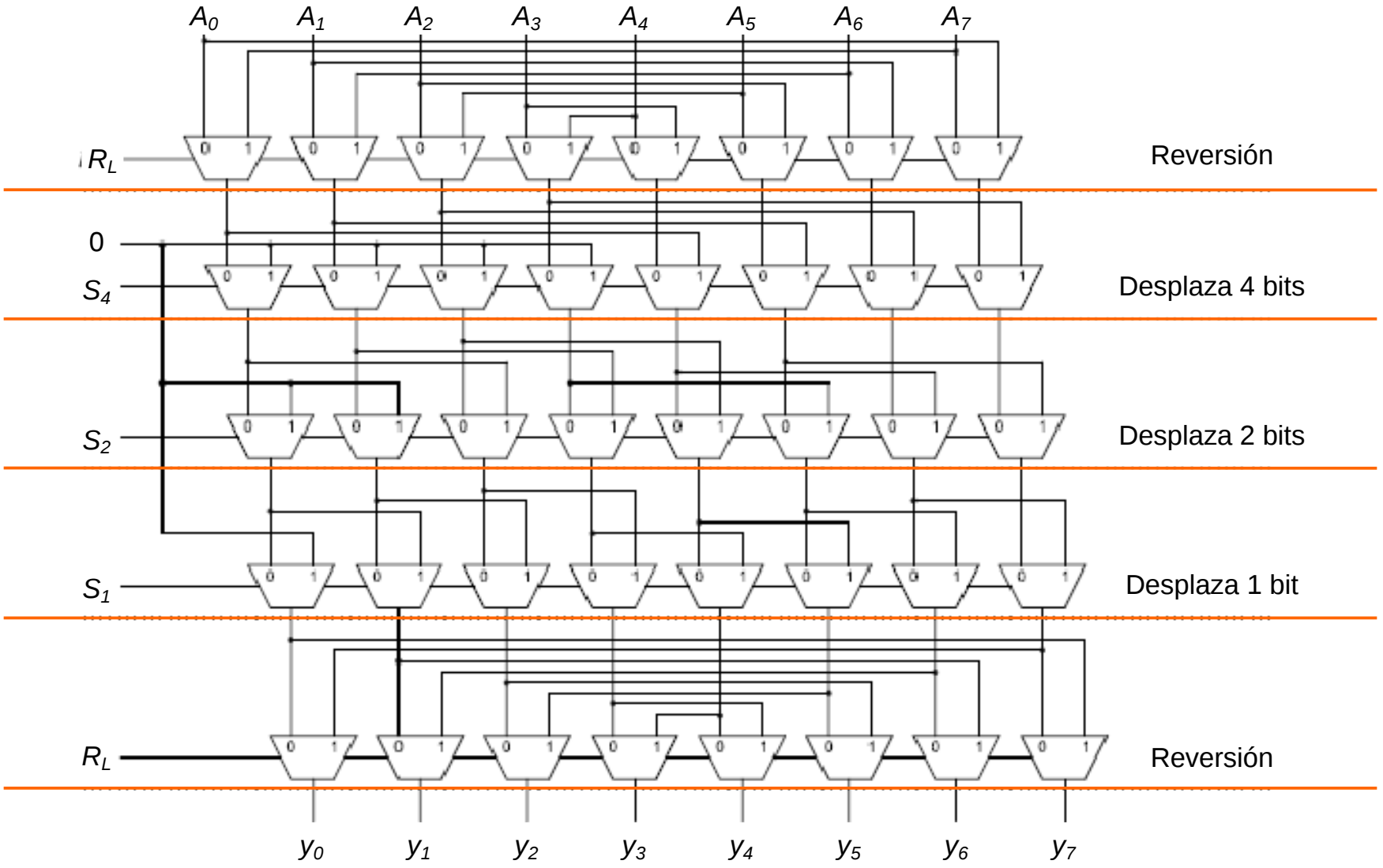
Estas operaciones, desplazamiento a derecha o izquierda, se implementan con circuitos combinacionales que utilizan una estructura de k niveles con multiplexores de dos entradas.

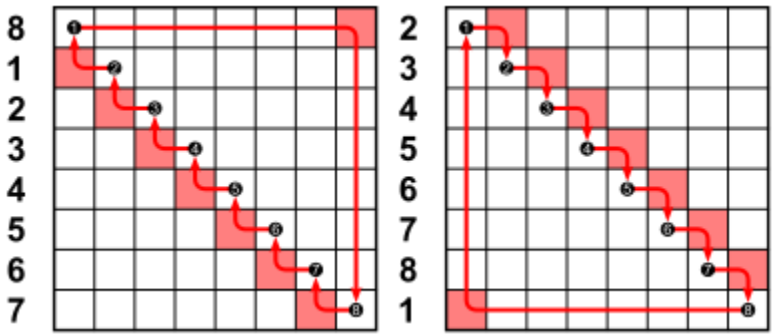




Para implementar los desplazamientos en ambas direcciones se agrega una estructura de reversión de datos a la entrada y a la salida del circuito.

Operaciones bit a bit – *Desplazamientos lógicos*
Unidad de desplazamiento

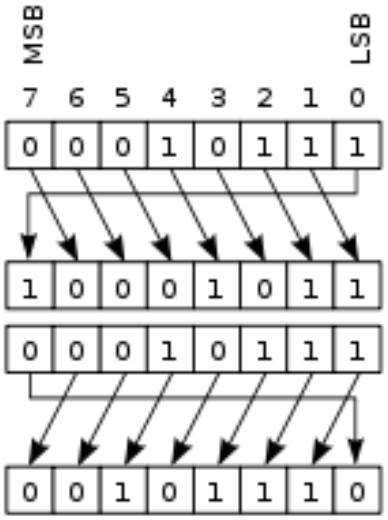




Una rotación es una operación de reordenamiento de las entradas de una n-upla, a partir de mover sus entradas iniciales y finales mientras se desplazan las otras entradas a la siguiente posición (en cualquiera de las direcciones). Una rotación.

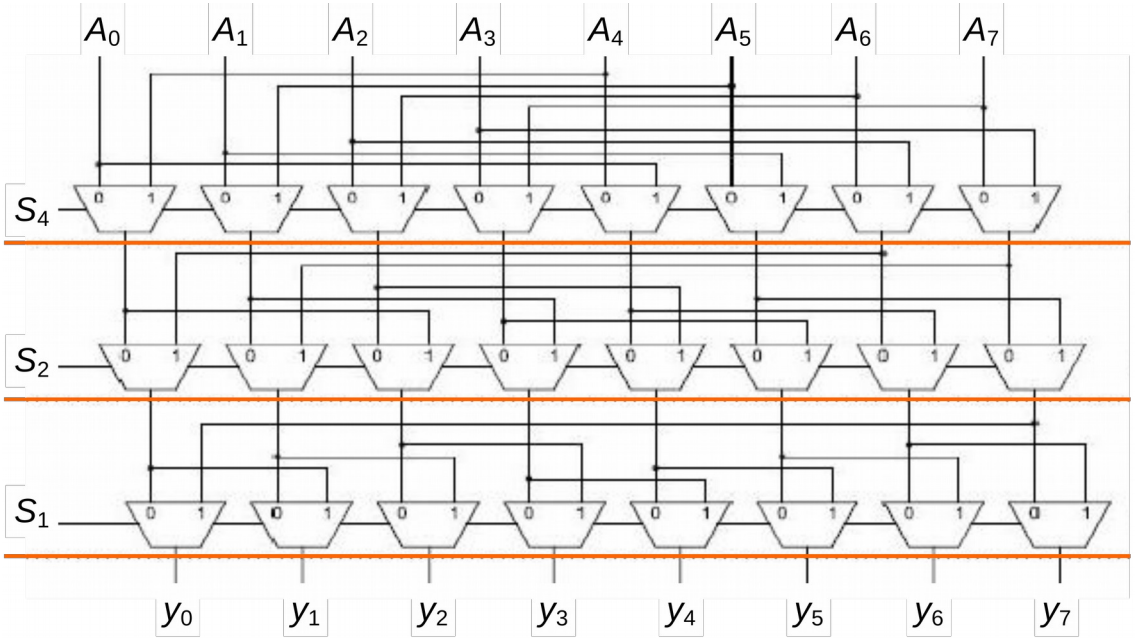
Una rotación es una clase especial de ***permutaciones cíclicas***, que son un caso particular de ***permutaciones***.

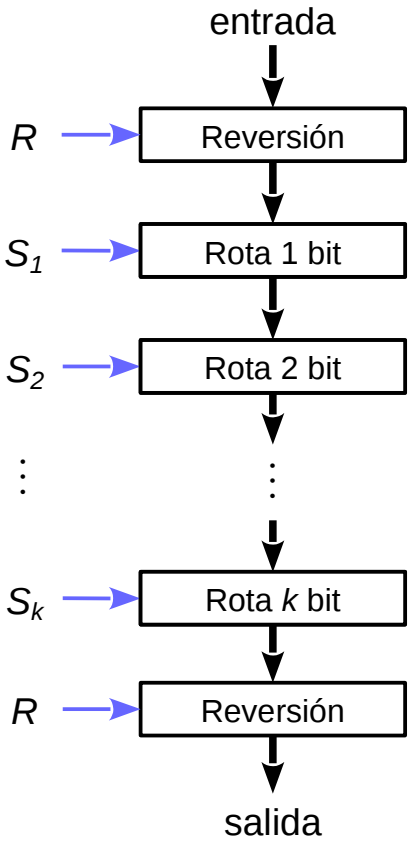
En la operación de rotación, los bits de una palabra son ***rotados*** de una manera circular como si los extremos izquierdo y derecho del registro estuvieran conectados. En la rotación hacia la izquierda, el bit que sale por el extremo izquierdo entrará por el extremo derecho, y viceversa con la rotación hacia la derecha.
Esta operación es útil si es necesario conservar todos los bits existentes.



Operaciones bit a bit – *Rotaciones*

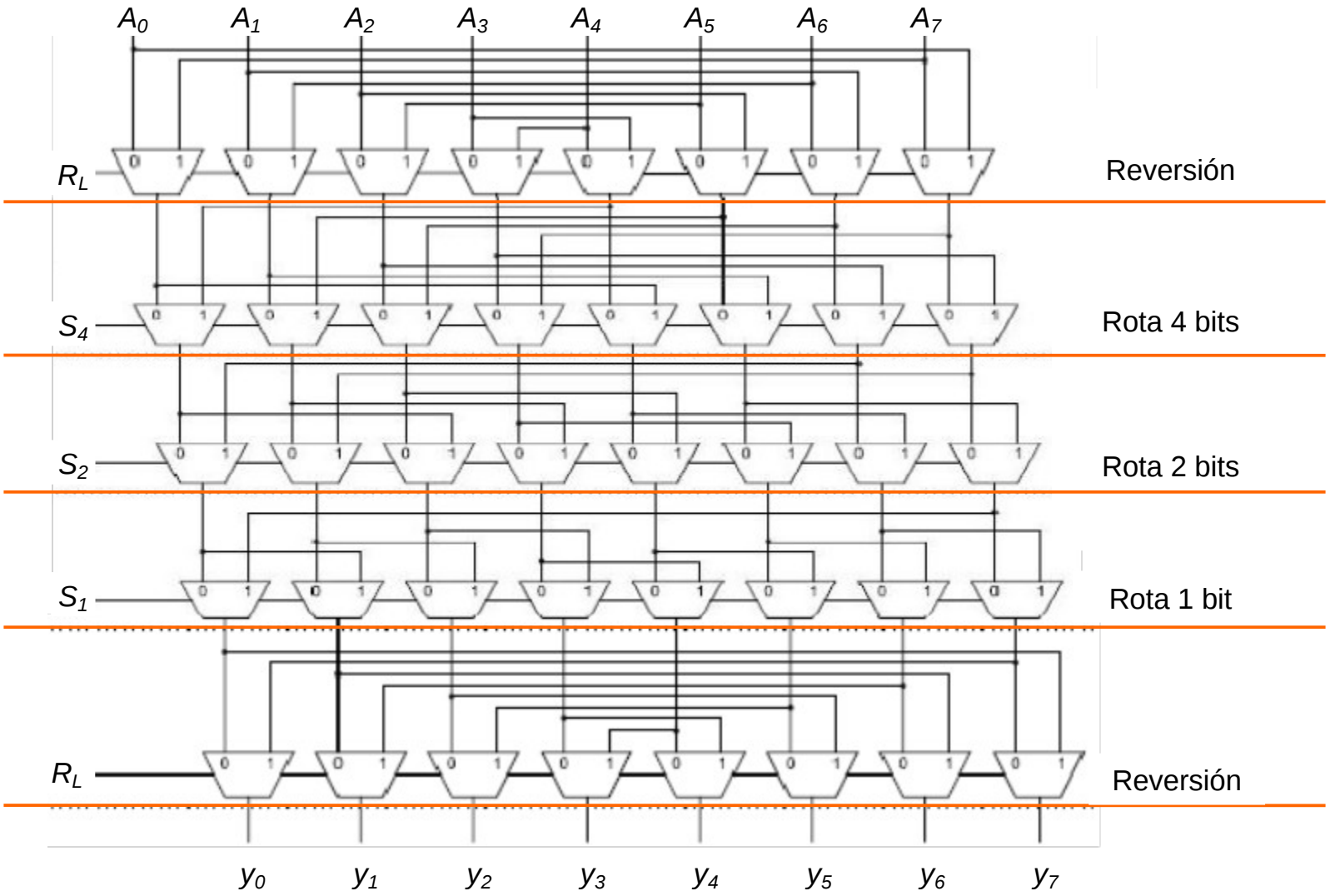
Estas operaciones, rotación a derecha o izquierda, se implementan con circuitos combinacionales que utilizan una estructura de k niveles con multiplexores de dos entradas.





Para implementar las rotaciones en ambas direcciones se agrega una estructura de reversión de datos a la entrada y a la salida del circuito.

Operaciones bit a bit – *Rotaciones*
Unidad de rotación



Operaciones bit a bit – *Barrel shifter*

Un barrel shifter es un circuito digital que puede desplazar o rotar una palabra utilizando sólo lógica combinacional. Son utilizados en las unidades aritmético lógicas para desplazar o rotar n-bits en un solo ciclo de reloj.

Un uso muy común de los barrel shifter es para la implementación en hardware de operaciones aritméticas en punto flotante, donde se lo utiliza para ajustar (alinear) los exponentes de dos números para sumarlos o restarlos.