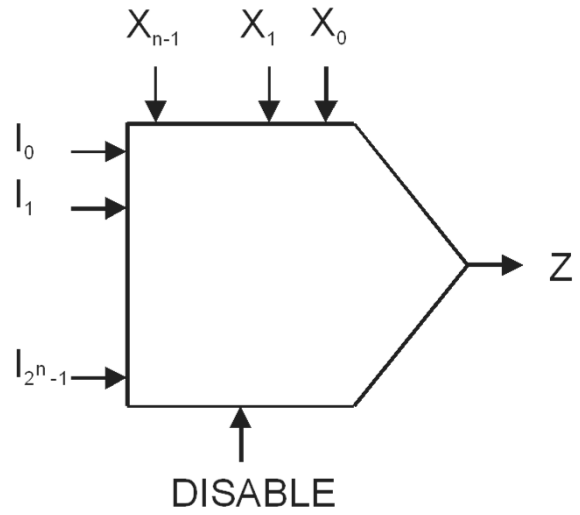
- **Bloques para el encaminamiento y/o transferencia de datos**
  - Multiplexor
  - Demultiplexor
  - Decodificador
  - Codificador
- **Bloques para el procesamiento de datos**
  - Comparador
- **Bloques para la generación de funciones booleanas**
  - ROM
  - PLA
  - PAL
- **Bloques combinacionales aritméticos**
  - Semisumador
  - Sumador binario completo
  - Sumador serie de *n* bits
  - Sumador paralelo.

Bloques para el encaminamiento y/o transferencia de datos

- **Multiplexor**

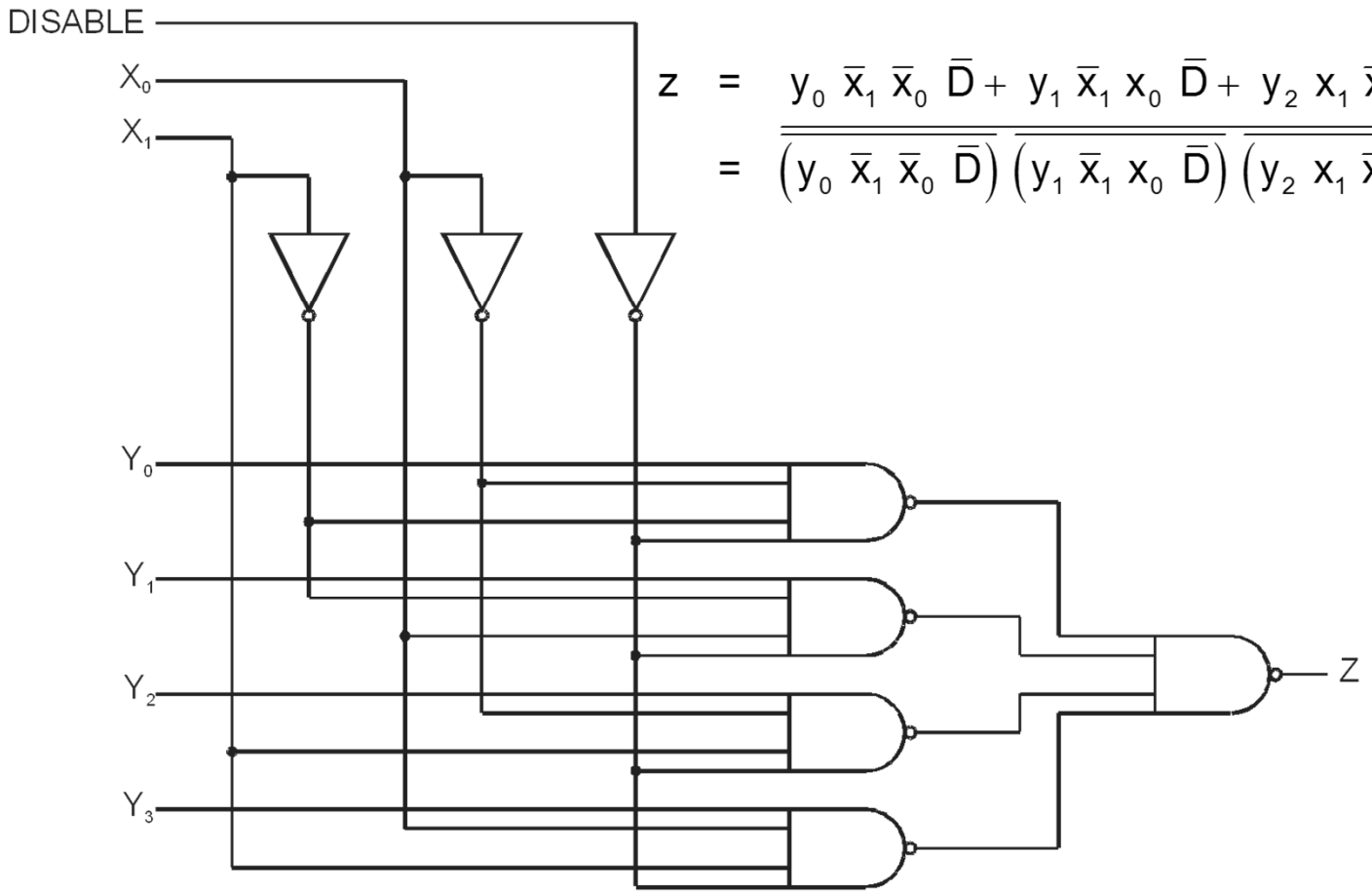
Es un conmutador electrónico que conecta a una única señal de salida, una de las diversas señales de entrada, en función de los valores de las señales de control.



Función de salida

$$z = \left[(\overline{x}_{n-1}\ \overline{x}_{n-2}\ \ldots.\ \overline{x}_1\ \overline{x}_0) * I_0\right] + \left[(\overline{x}_{n-1}\ \overline{x}_{n-2}\ \ldots.\ \overline{x}_1\ x_0) * I_1\right] +$$
$$\left[(\overline{x}_{n-1}\ \overline{x}_{n-2}\ \ldots.\ x_1\ \overline{x}_0) * I_2\right] + \left[(\overline{x}_{n-1}\ \overline{x}_{n-2}\ \ldots.\ x_1\ x_0) * I_3\right] +$$
$$+ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ +$$
$$\left[(x_{n-1}\ x_{n-2}\ \ldots.\ x_1\ \overline{x}_0) * I_{2^n-2}\right] + \left[(x_{n-1}\ x_{n-2}\ \ldots.\ x_1\ x_0) * I_{2^n-1}\right]$$

- **Implementación de un multiplexor-2 mediante puertas NAND**



$$z = y_0\,\bar{x}_1\,\bar{x}_0\,\bar{D} + y_1\,\bar{x}_1\,x_0\,\bar{D} + y_2\,x_1\,\bar{x}_0\,\bar{D} + y_3\,x_1\,x_0\,\bar{D} =$$

$$= \overline{\left(y_0\,\bar{x}_1\,\bar{x}_0\,\bar{D}\right)}\;\overline{\left(y_1\,\bar{x}_1\,x_0\,\bar{D}\right)}\;\overline{\left(y_2\,x_1\,\bar{x}_0\,\bar{D}\right)}\;\overline{\left(y_3\,x_1\,x_0\,\bar{D}\right)}$$

Para implementar una función combinacional utilizando multiplexores de ***N*** señales de selección (MUX-N), debemos aplicar a la función el teorema del desarrollo para ***N*** variables.

$$
\begin{aligned}
f(x_1, x_2, \ldots, x_m) = {} & \left[(\overline{x}_1 * \overline{x}_2 * \ldots * \overline{x}_n) * f(0,0, \ldots, 0, x_{n+1}, x_{n+2}, \ldots, x_m)\right] + \\
& \left[(\overline{x}_1 * \overline{x}_2 * \ldots * x_n) * f(0,0, \ldots, 1, x_{n+1}, x_{n+2}, \ldots, x_m)\right] + \\
& + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + \\
& \left[(x_1 * x_2 * \ldots * x_n) * f(1,1, \ldots, 1, x_{n+1}, x_{n+2}, \ldots, x_m)\right]
\end{aligned}
$$

a) Si los residuos obtenidos son constantes o dependen de una sola variable, el proceso ha terminado.

b) Si los residuos obtenidos dependen de 2 ó más variables, hay que repetir el proceso con cada uno de estos residuos.

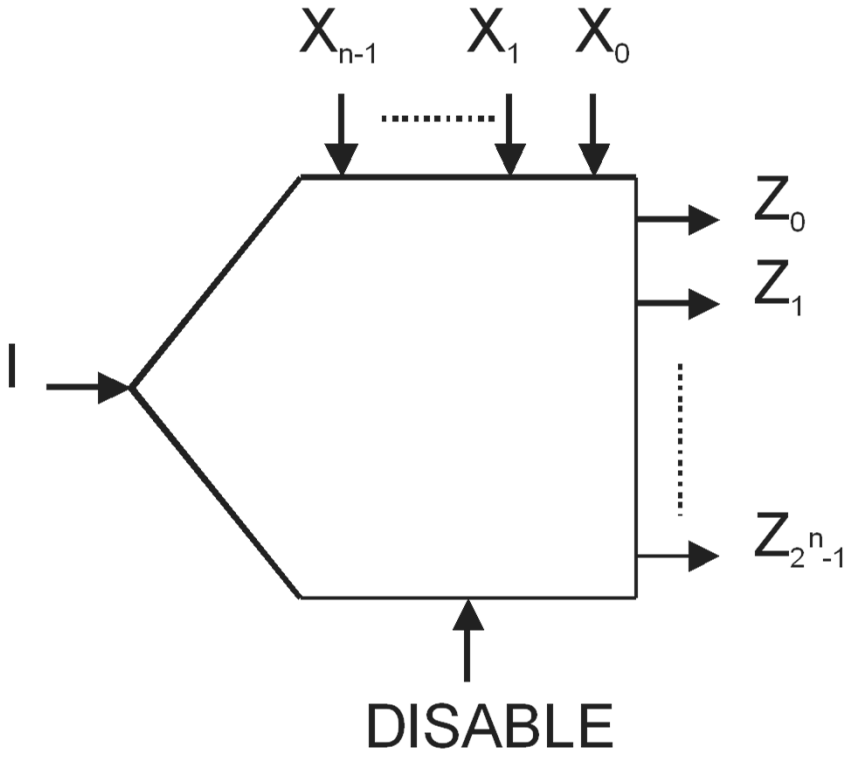
Dado un multiplexor de ***N*** señales de selección (MUX-N), podemos implementar una función de ***N+1*** variables.

N variables como señales de control.

1 variable en las entradas.

- **Demultiplexor**

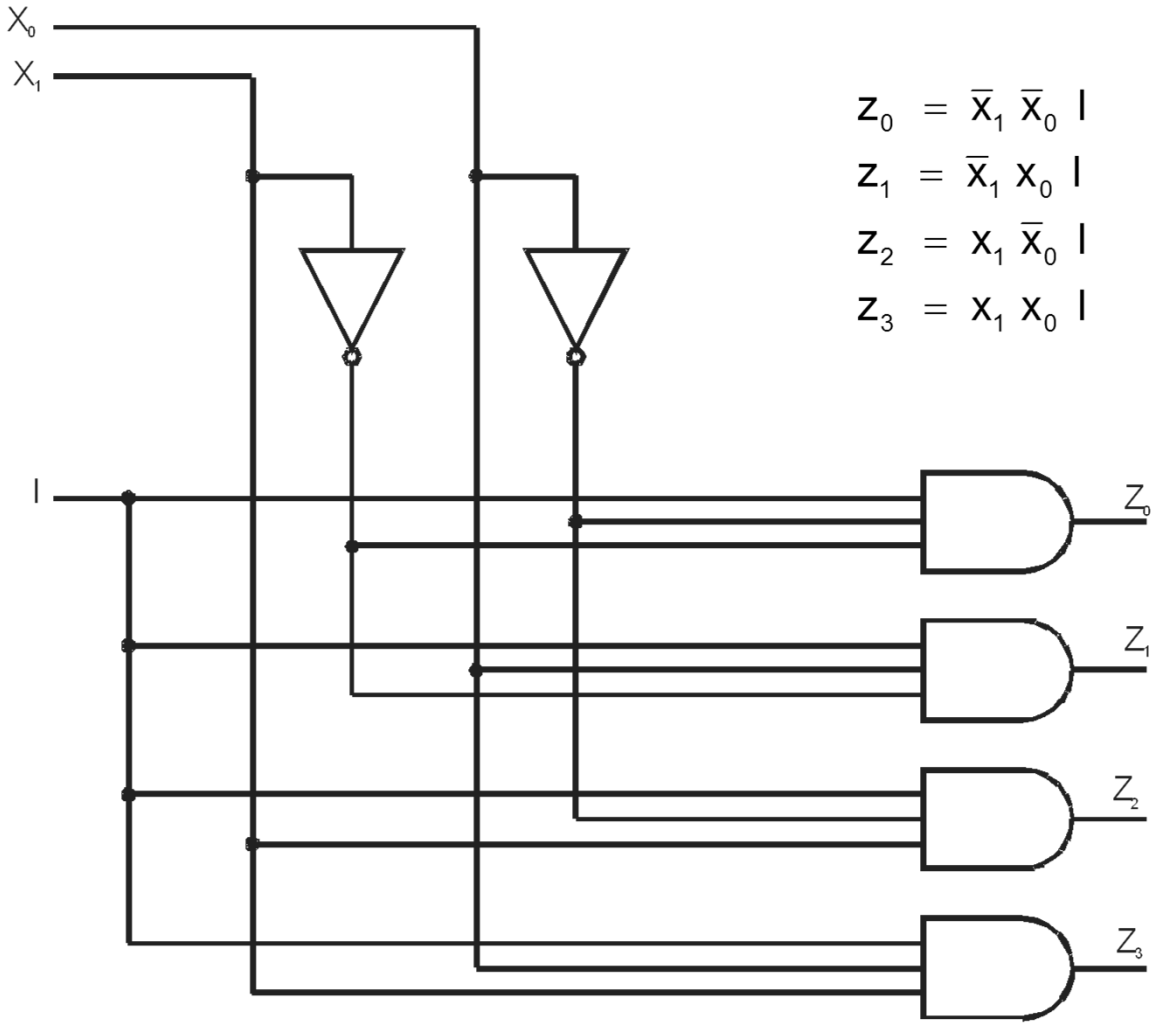
Es un conmutador electrónico que conecta una única señal de entrada a una de las diversas señales de salida, en función de los valores de las señales de control



Funciones de salida

$$z_0 = I\,(\overline{x}_{n-1}\,\overline{x}_{n-2}\,\ldots\ldots\,\overline{x}_1\,\overline{x}_0)$$
$$z_1 = I\,(\overline{x}_{n-1}\,\overline{x}_{n-2}\,\ldots\ldots\,\overline{x}_1\,x_0)$$
$$\ldots\ldots\ldots$$
$$z_{2^n-2} = I\,(x_{n-1}\,x_{n-2}\,\ldots\ldots\,x_1\,\overline{x}_0)$$
$$z_{2^n-1} = I\,(x_{n-1}\,x_{n-2}\,\ldots\ldots\,x_1\,x_0)$$

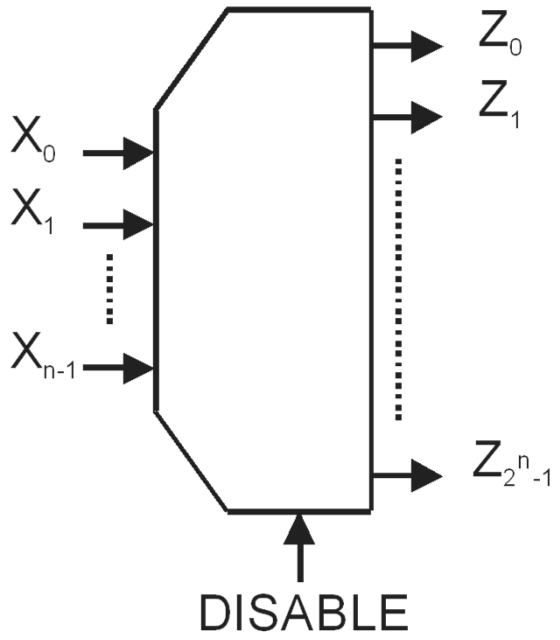
- **Implementación de un demultiplexor-2 mediante puertas AND**



$$z_0 = \overline{x}_1\ \overline{x}_0\ I$$
$$z_1 = \overline{x}_1\ x_0\ I$$
$$z_2 = x_1\ \overline{x}_0\ I$$
$$z_3 = x_1\ x_0\ I$$

- **Decodificador**

Circuito combinacional de *n* entradas y 2^n salidas, que activa una de estas salidas en función de los valores de entrada.

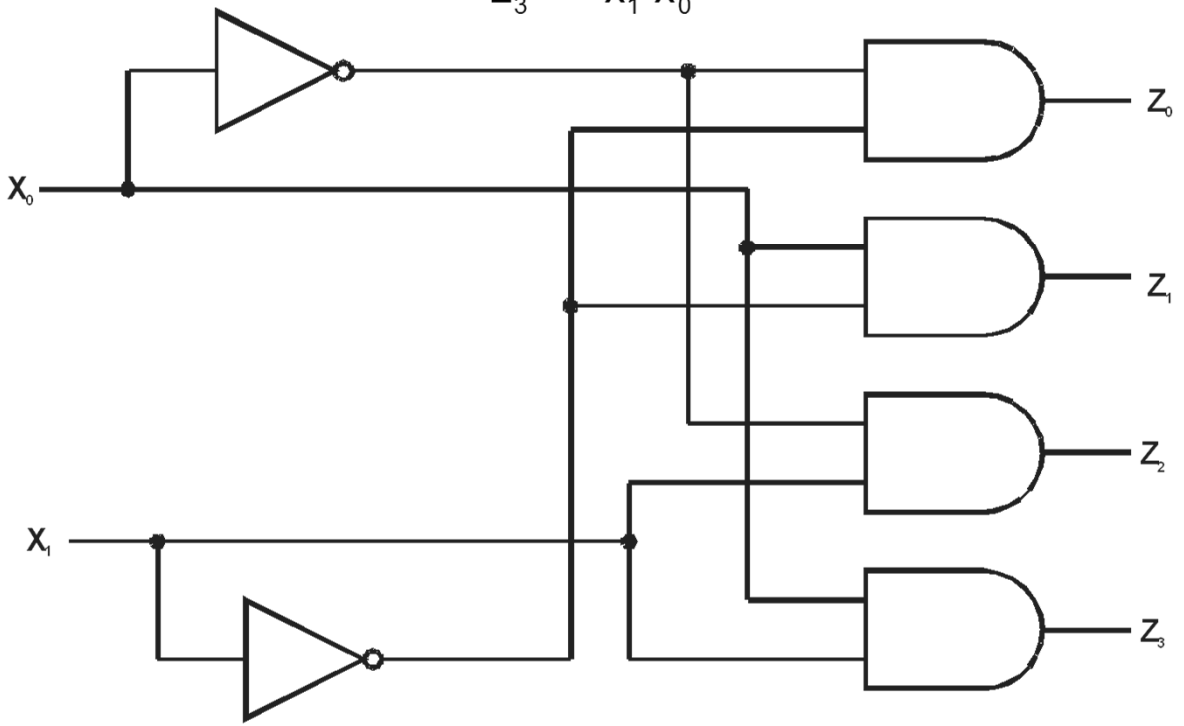


Funciones de salida:

$$Z_0 = \overline{x}_{n-1} \ldots \overline{x}_1 \overline{x}_0$$
$$Z_1 = \overline{x}_{n-1} \ldots \overline{x}_1 x_0$$
$$\ldots \qquad \ldots\ldots\ldots$$
$$Z_{2^n-1} = x_{n-1} \ldots x_1 x_0$$

- **Implementación de un decodificador de 2 entradas con puertas AND**

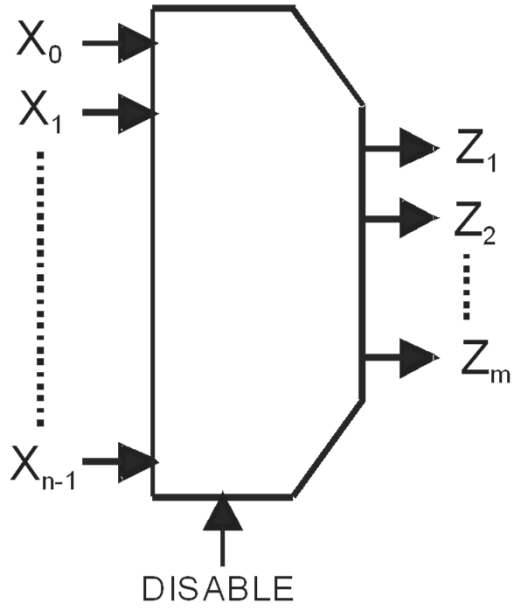
$$z_0 = \overline{x}_1\,\overline{x}_0$$
$$z_1 = \overline{x}_1\,x_0$$
$$z_2 = x_1\,\overline{x}_0$$
$$z_3 = x_1\,x_0$$

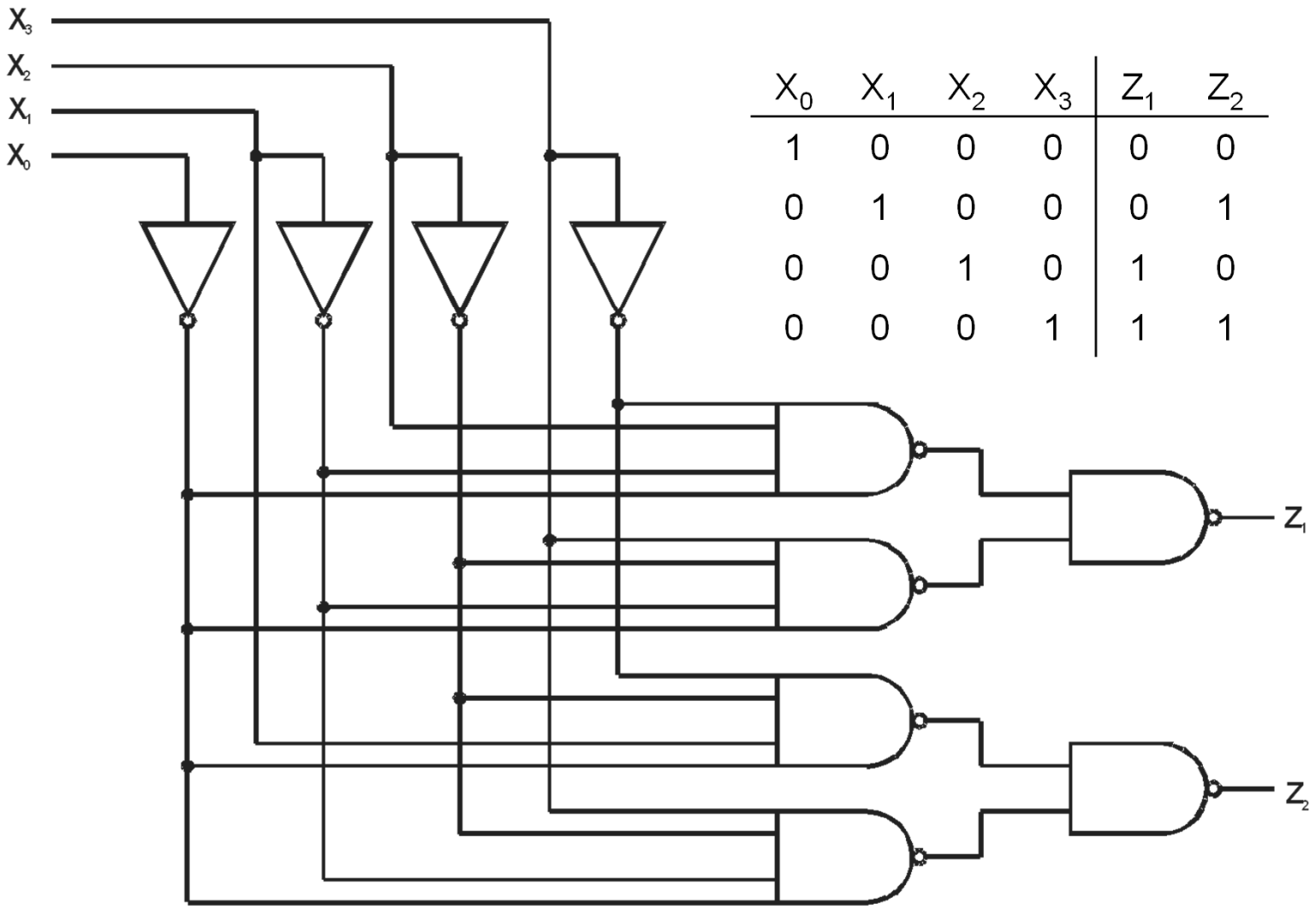


- **Codificador**

Sistema combinacional de *n* entradas y *m* salidas ($n \leq 2^m$), que efectúa la conversión de un código en otro

Ejp: Codificador de BCD a código 7-segmentos





X_0	X_1	X_2	X_3	Z_1	Z_2
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Comparadores

- **Comparador**

Un comparador es un dispositivo combinacional que compara dos números y especifica por medio de tres señales binarias el resultado de la comparación

Datos a comparar:

$A = a_{n-1}a_{n-2} \ldots a_1a_0$

$B = b_{n-1}b_{n-2} \ldots b_1b_0$

Resultado de comparación previa:

$f_n=1, g_n=0, h_n=0 \Rightarrow$ Previo mayor $\Rightarrow F=1, G=0, H=0$

$f_n=0, g_n=1, h_n=0 \Rightarrow$ Previo igual $\Rightarrow$ Comparar A,B

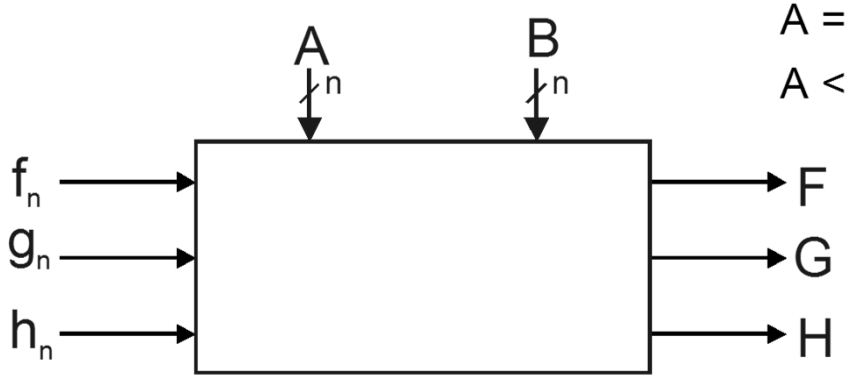
$f_n=0, g_n=0, h_n=1 \Rightarrow$ Previo menor $\Rightarrow F=0, G=0, H=1$

Resultado de la comparación:

$A > B \Rightarrow F=1, G=0, H=0$

$A = B \Rightarrow F=0, G=1, H=0$

$A < B \Rightarrow F=0, G=0, H=1$



Un comparador con un número muy grande de bits es en circuito con un gran número de entradas, que resulta difícil de construir mediante tabla de verdad, por lo que se recurre a un método recursivo.

- **Comparador de dos bits**

$x_i = 1$ si $a_i > b_i \Rightarrow x_i = a_i \overline{b_i}$

$y_i = 1$ si $a_i = b_i \Rightarrow y_i = \overline{a_i}\,\overline{b_i} + a_i\,b_i$

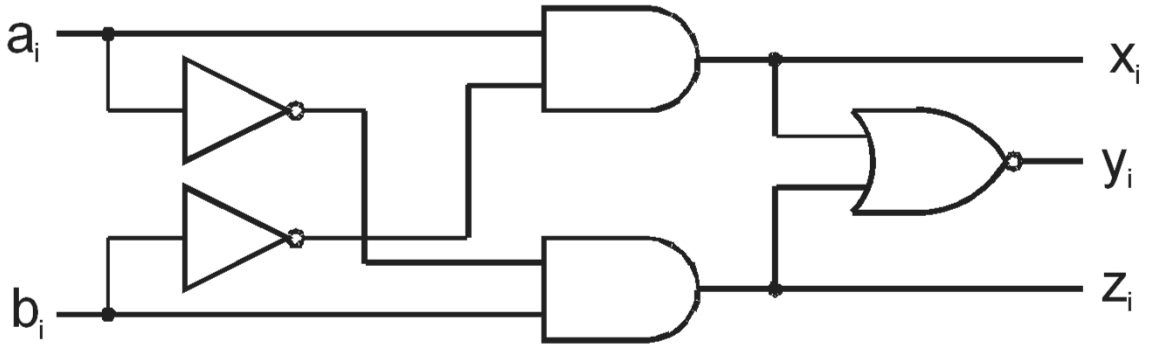
$$= \overline{\overline{\overline{a_i}\,\overline{b_i}}\;\overline{a_i\,b_i}} = \overline{(a_i+b_i)\,(\overline{a_i}+\overline{b_i})} =$$

$$= \overline{(a_i+b_i)\,\overline{a_i} + (a_i+b_i)\,\overline{b_i}} =$$

$$= \overline{a_i\overline{a_i} + \overline{a_i}b_i + a_i\overline{b_i} + b_i\overline{b_i}} =$$

$$= \overline{\overline{a_i}b_i + a_i\overline{b_i}}$$

$z_i = 1$ si $a_i < b_i \Rightarrow z_i = \overline{a_i}\,b_i$



Definimos

$$\begin{array}{cccccccccc} a_{n-1} & a_{n-2} & \dots & a_{i+1} & a_i & a_{i-1} & \dots & a_1 & a_0 \\ b_{n-1} & b_{n-2} & \dots & b_{i+1} & b_i & b_{i-1} & \dots & b_1 & b_0 \end{array}$$

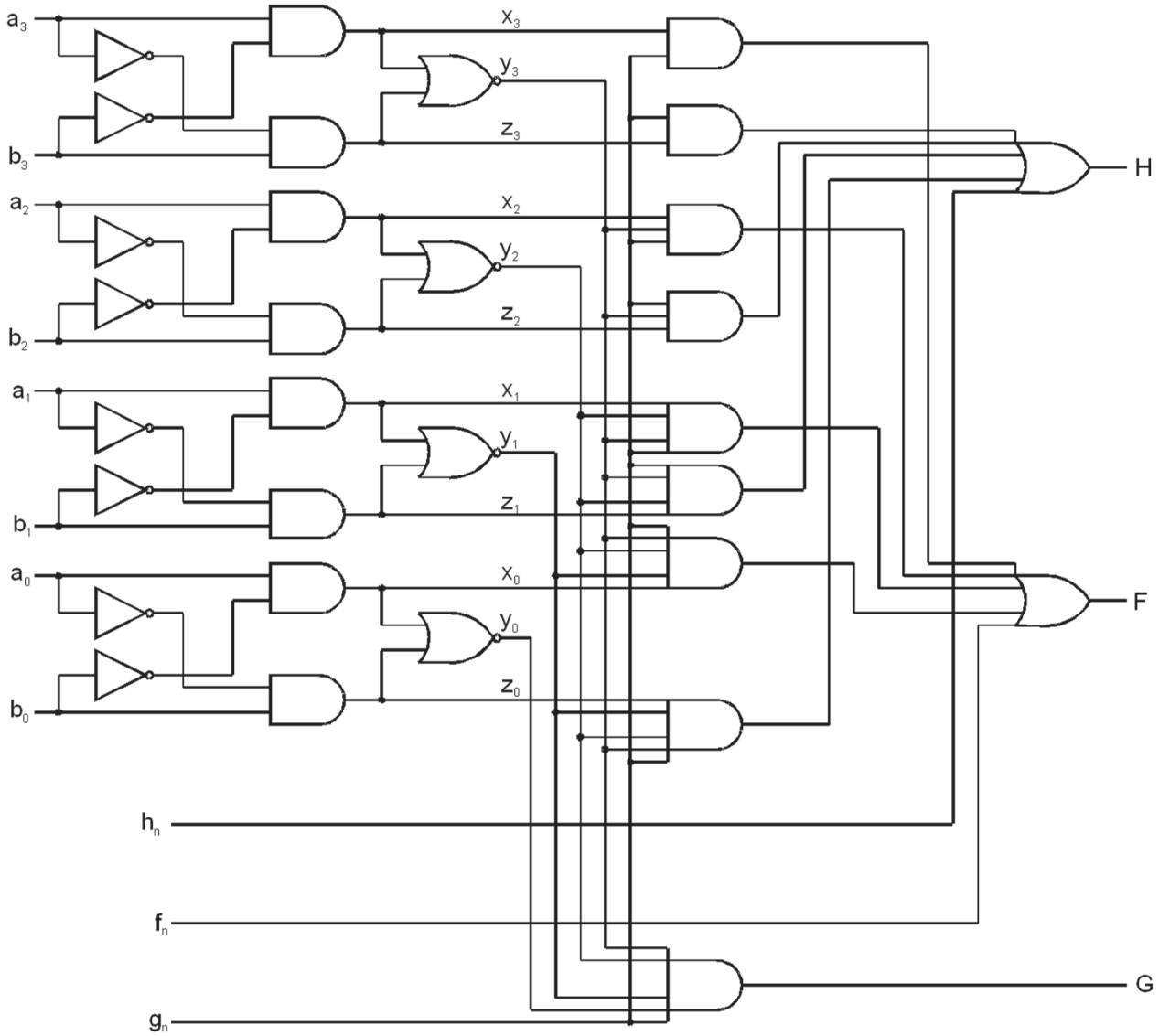
$$\underbrace{\qquad\qquad}_{f_{i+1},g_{i+1},h_{i+1}}$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{f_i,g_i,h_i}$$

$$\begin{aligned} f_i = 1, g_i = 0, h_i = 0 \quad &\text{si} \quad a_{n-1}a_{n-2} \dots a_{i+1}a_i > b_{n-1}b_{n-2} \dots b_{i+1}b_i \\ f_i = 0, g_i = 1, h_i = 0 \quad &\text{si} \quad a_{n-1}a_{n-2} \dots a_{i+1}a_i = b_{n-1}b_{n-2} \dots b_{i+1}b_i \\ f_i = 0, g_i = 0, h_i = 1 \quad &\text{si} \quad a_{n-1}a_{n-2} \dots a_{i+1}a_i < b_{n-1}b_{n-2} \dots b_{i+1}b_i \end{aligned}$$

Ecuación de recurencia:

$$\begin{aligned} f_i &= f_{i+1} + g_{i+1}x_i & & f_n = 0 \\ g_i &= g_{i+1}\, y_i & \text{donde} \quad & g_n = 1 \\ h_i &= h_{i+1} + g_{i+1}z_i & & h_n = 0 \end{aligned}$$


a3
b3
a2
b2
a1
b1
a0
b0
x3
y3
z3
x2
y2
z2
x1
y1
z1
x0
y0
z0
hn
fn
gn
H
F
G

Bloques para la generación de funciones booleanas

- **ROM**

Estructura lógica organizada en *N* palabras de *M* bits que contiene:

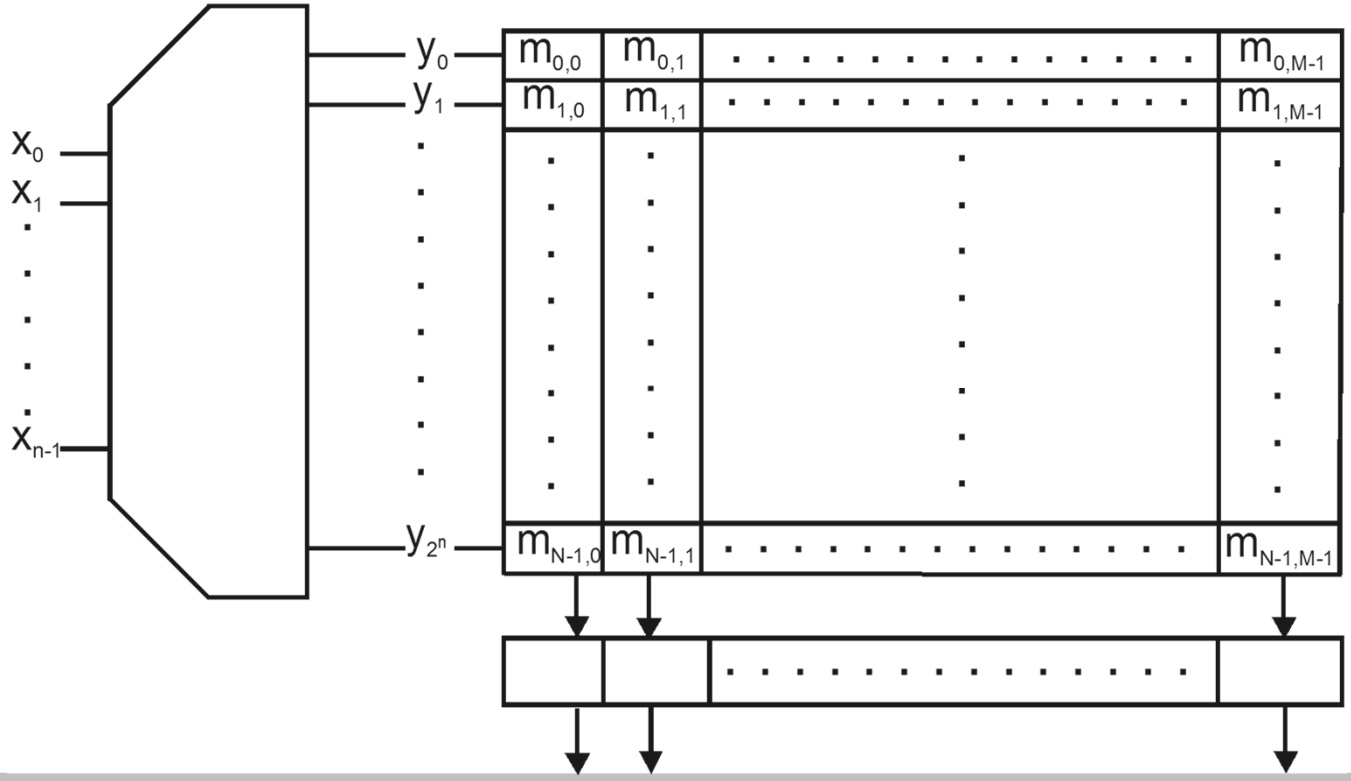
a) *n* entradas binarias $x_0,x_1,...,x_{n-1}$
b) *M* salidas binarias $z_0,z_1,...,z_{M-1}$
c) $N \times M$ elementos de memoria de un bit denotados por m_{ij}

donde:

$N = 2^n$
$i = 0,1,...,N-1$
$j = 0,1,...,M-1$

El sistema está formado por una matriz de celdas organizadas por palabras (filas de celdas a las que se puede acceder separadamente), más un decodificador de direcciones para seleccionar una palabra completa de la memoria.

La palabra seleccionada se coloca en el registro de salida y sus valores serán la salida correspondiente con la entrada fijada.



El comportamiento E/S vendrá dado por:

$$z_j = \sum_{i=0}^{N-1} y_i m_{ij} \qquad \text{donde:} \begin{cases} 0 \le j \le M\text{-}1 \\ y_i \text{ es la salida del decodificador} \\ m_{ij} \text{ es el contenido de la fila } i \text{ columna } j \end{cases}$$

$$\begin{pmatrix} m_{0,0} & m_{0,1} & \dots & m_{0,M-1} \\ m_{1,0} & m_{1,1} & \dots & m_{1,M-1} \\ \dots & \dots & \dots & \dots \\ m_{N-1,0} & m_{N-1,1} & \dots & m_{N-1,M-1} \end{pmatrix}$$

($m_{i,0}$, $m_{i,,1}$, ... , $m_{i,M-1}$) representa la palabra i-ésima de la matriz de memoria.

- **PROM (Programmable Read Only Memories)**

  Es una memoria ROM que puede ser programada por el fabricante, a petición del usuario.

  Una vez programada no se puede volver a programar.

- **EPROM (Erasable PROM)**

  Es una memoria PROM que puede se programada y borrada por el propio usuario.

  Para realizar la operación de grabado se utiliza un dispositivo especial (grabador de EPROM), y para borrarlas se someten a radiación ultravioleta (por lo que una vez grabadas deben ser tapadas por un material opaco que las protege de la luz).

- **EEPROM (Electric Erasable PROM)**

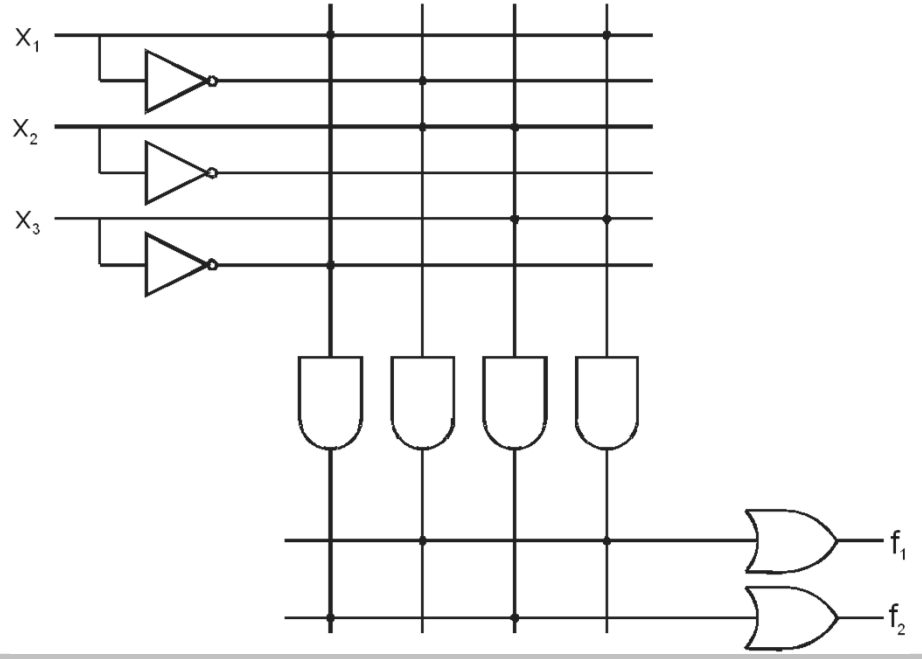
  Es una memoria PROM que puede ser grabada y borrada por el propio usuario.

  La operación de grabado y borrado se realiza simplemente activando unas señales de control en la propia memoria, sin necesidad de ningún dispositivo adicional.

- **PLA (Programmable Logic Arrays)**

  Una ROM de *N* palabras de *M* bits cada palabra permite implementar *M* funciones de *n* variables (N = 2n), pero frecuentemente las funciones a implementar no utilizan todas las combinaciones posibles de entrada, por lo que se desperdicia parte de la ROM.

  Una PLA está compuesta por una matriz de términos productos, y una matriz de términos suma.

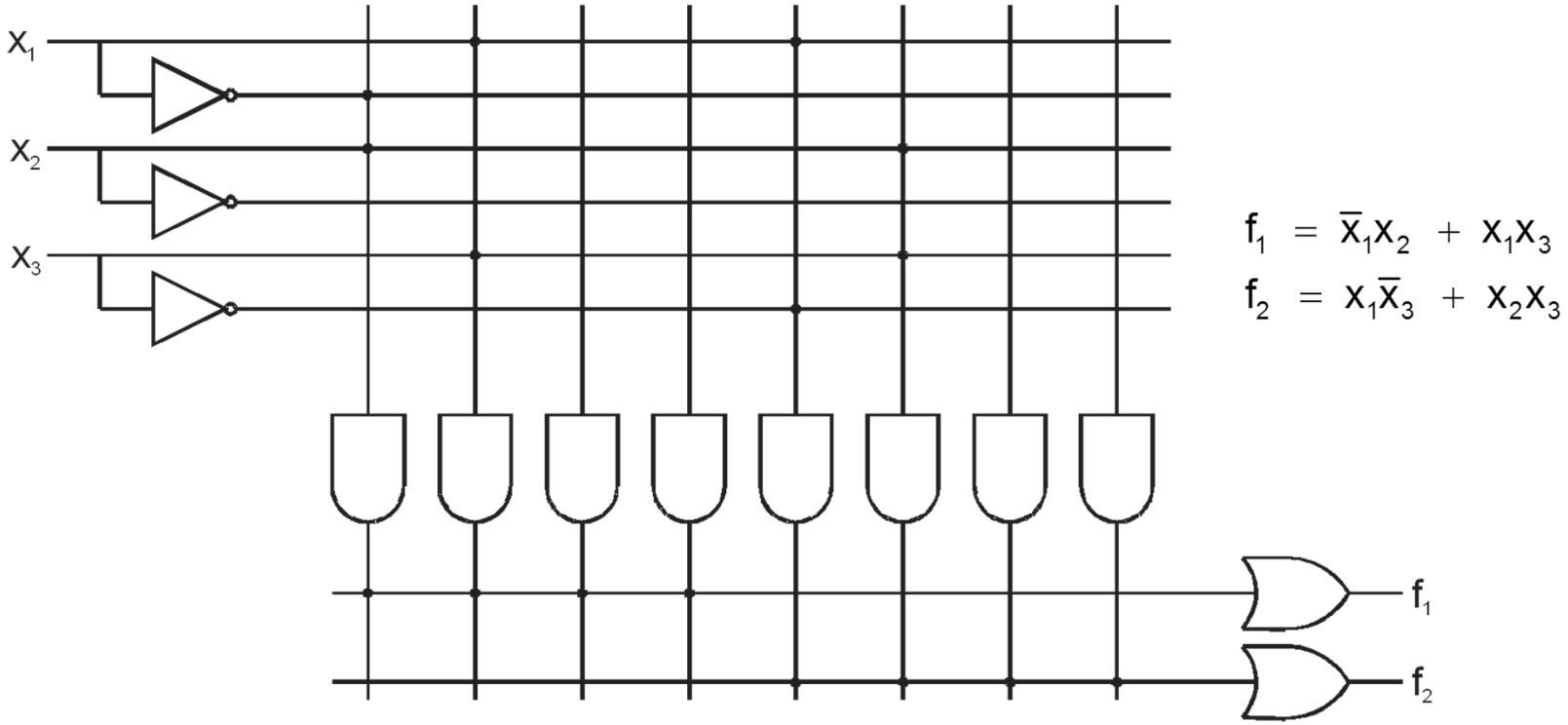


$$f_1 = \overline{x}_1 x_2 + x_1 x_3$$
$$f_2 = x_1 \overline{x}_3 + x_2 x_3$$

- **PAL (Programmable Array Logic)**

  Es un caso particular de PLA en el que no es programable la matriz de términos suma.

  Es el caso más utilizado por su bajo coste y flexibilidad de programación



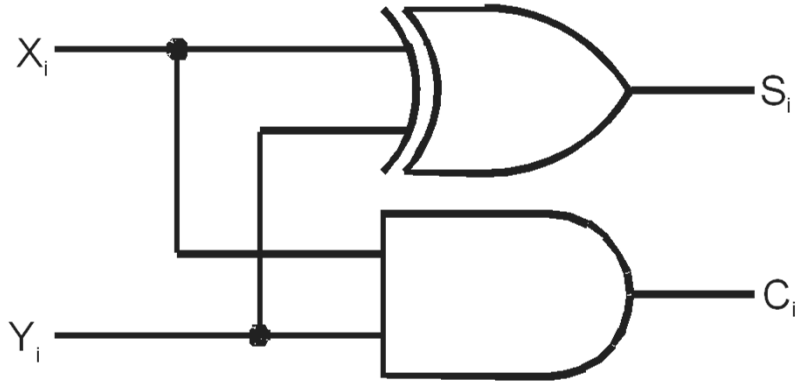
Bloques combinacionales aritméticos

- **Semisumador**

x_i	y_i	c_{i+1}	s_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$c_{i+1} = x_i y_i$$

$$s_i = \overline{x}_i y_i + x_i \overline{y}_i = x_i \oplus y_i$$



- **Sumador binario completo**

x_i	y_i	c_i	c_{i+1}	s_i
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

c_{i+1}

c_i \ $x_i y_i$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

s_i

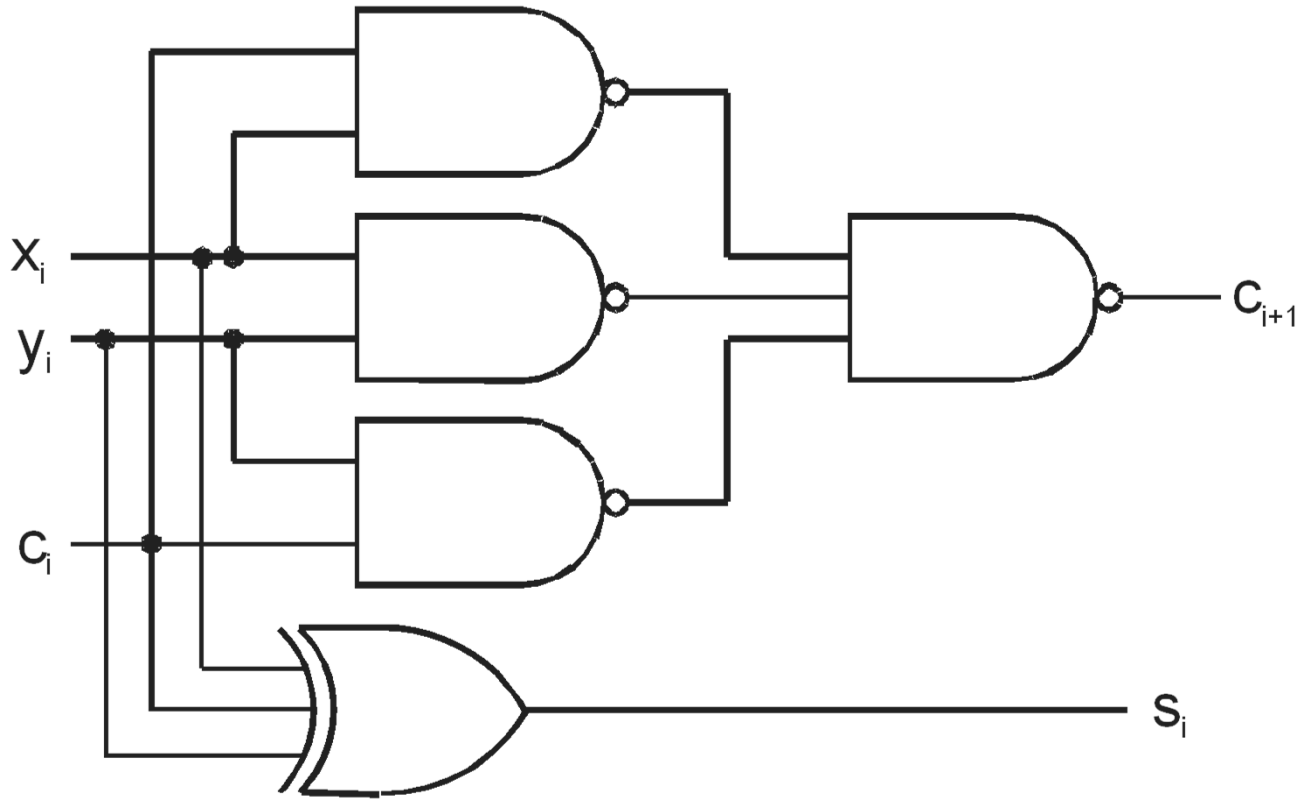
c_i \ $x_i y_i$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$c_{i+1} = x_i y_i + y_i c_i + x_i c_i$$

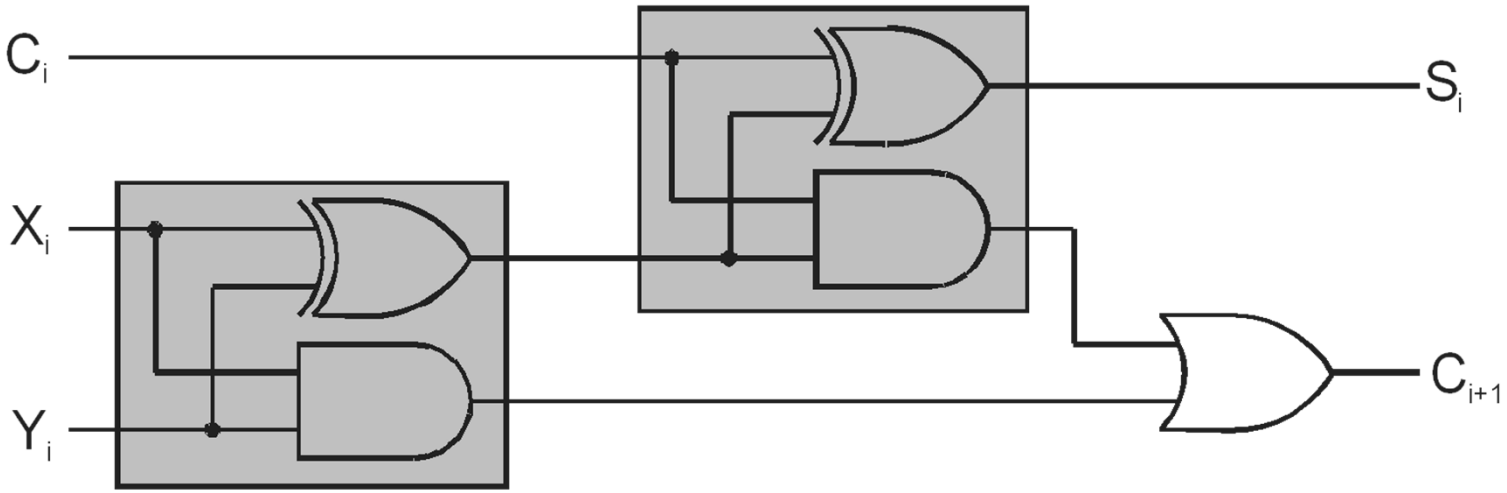
$$s_i = \overline{x}_i \overline{y}_i c_i + \overline{x}_i y_i \overline{c}_i + x_i y_i c_i + x_i \overline{y}_i \overline{c}_i = (\overline{x}_i y_i + x_i \overline{y}_i)\,\overline{c}_i + (\overline{x}_i \overline{y}_i + x_i y_i)\,c_i =$$

$$= \left\| \begin{array}{l} \overline{x}_i \overline{y}_i + x_i\, y_i = \overline{\overline{(\overline{x}_i\, \overline{y}_i)}\;\overline{(x_i\, y_i)}} = \overline{(x_i + y_i)\,(\overline{x}_i + \overline{y}_i)} = \\ = \overline{(x_i + y_i)\,\overline{x}_i + (x_i + y_i)\,\overline{y}_i} = \overline{x_i \overline{x}_i + \overline{x}_i y_i + x_i \overline{y}_i + y_i \overline{y}_i} = \\ = \overline{\overline{x}_i y_i + x_i \overline{y}_i} = \overline{x_i \oplus y_i} \end{array} \right\| =$$

$$= (x_i \oplus y_i)\,\overline{c}_i + \overline{(x_i \oplus y_i)}\,c_i = x_i \oplus y_i \oplus c_i$$

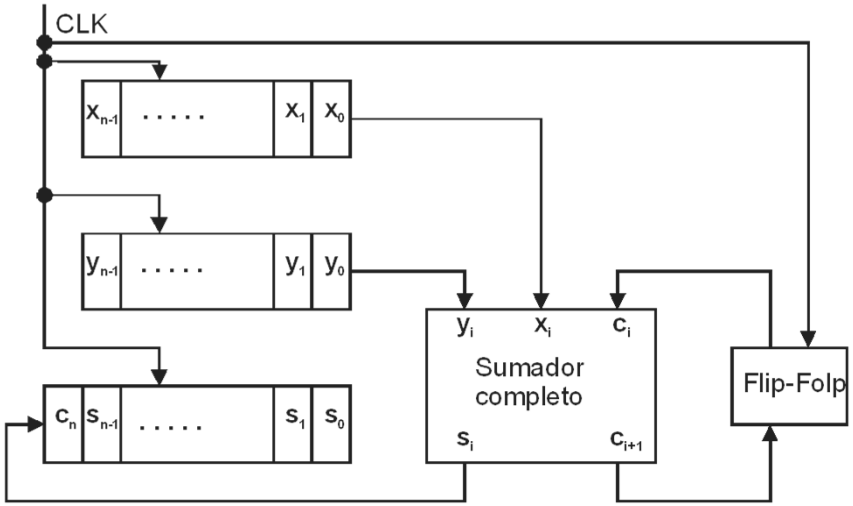

x_i
y_i
c_i
c_{i+1}
s_i

- **Sumador completo construido con 2 semisumadores**



- **Sumador serie de *n* bits**

  Podemos construir un sumador de *n* bits, utilizando un *sumador completo* y añadiendo un circuito secuencial, de forma que mediante un reloj, vamos sumando en cada instante de tiempo un par de bits y el acarreo del par anterior.



Ventajas. Tiene bajo coste hardware. (utiliza un sólo sumador completo independientemente de los bits que tenga la palabra que queremos sumar).

Desventajas. Es lento. (el tiempo que tarda en realizar la suma depende del número de bits que tengan las palabras que queremos sumar).

- **Sumador paralelo**

  El sumador paralelo realiza simultáneamente la suma de los *n* bits, y para ello utiliza *n* sumadores completos

  Hay dos tipos de sumadores paralelos

  a) Con generación del acarreo en serie

  b) Con generación del acarreo en paralelo (acarreo anticipado).

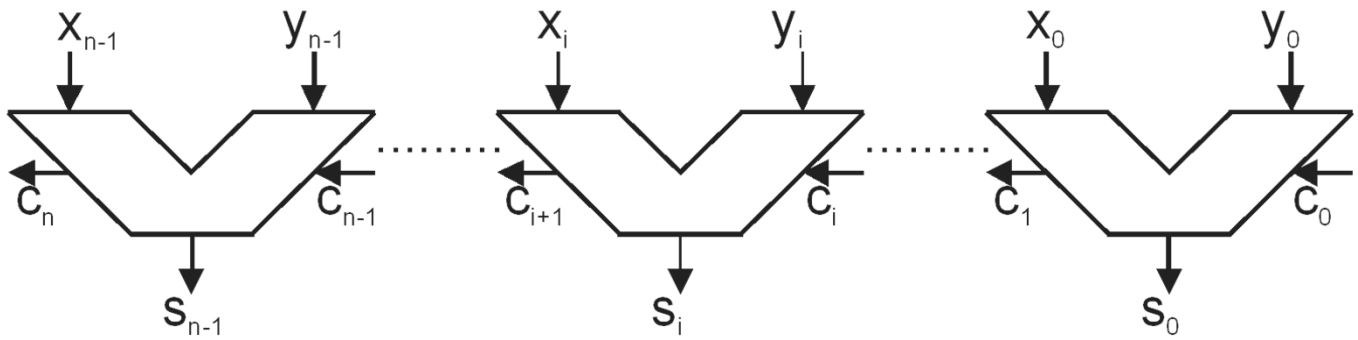
- **Sumador paralelo con generación del acarreo en serie**

Cada sumador completo realiza la suma en función del acarreo de la suma de los bits de peso inferior

El tiempo que se tarda en realizar la suma, viene dador por la suma de los tiempos de propagación de todos los sumadores, y dependerá del número de bits de las palabras que vamos a sumar

Si el tiempo de propagación de un sumador completo es t, para un sumador de *n* bits tenemos

$$\text{tiempo de operación} = n \times t$$



- **Sumador paralelo con generación del acarreo en paralelo (acarreo anticipado).**

  Los acarreos de todas las etapas son generados simultáneamente a partir de x_i , y_i , c_0

  Es necesario añadir un hardware adicional, que crece conforme aumenta el número de bits

  El tiempo necesario para realizar una suma no depende del tiempo de propagación de los sumadores de 1 bit