

Org. del Computador

Apunte

Contents

1	Primer Parcial	4
1.1	Sistemas de Numeración	5
1.1.1	Sistema Binario	5
1.1.2	Sistema Hexadecimal	5
1.1.3	Métodos de Conversión	5
1.1.4	Complemento a 2 para números negativos	6
1.1.5	Suma de números binarios	7
1.1.6	Resta de números binarios	7
1.1.7	Multiplicación de números binarios	7
1.1.8	Conversiones de bases	8
1.1.9	Sistema IEEE 754	8
1.1.10	Convertir de decimal a IEEE 754	9
1.1.11	Convertir de IEEE 754 a decimal	9
1.1.12	Ejemplo de conversión de decimal a IEEE 754 - Completo	9
1.2	Álgebra de Boole	11
1.2.1	Elementos del álgebra booleana	11
1.2.2	Teoremas y postulados del álgebra booleana	11
1.2.3	Funciones booleanas	11
1.2.4	Compuertas Lógicas digitales	12
1.3	Lógica Combinacional	13
1.3.1	Producto de Sumas y Suma de Productos (Maxitérminos y Mintérminos)	13
1.3.2	Mapas de Karnaugh	13
1.3.3	PLA (Programmable Logic Array)	18
1.3.4	Ejemplo completo de Lógica Combinacional	19
1.3.5	Decodificadores	21
1.3.6	Multiplexores	23
1.4	Direccionamiento y Lógica de Decodificación de Memorias	25
1.4.1	Comunicación con la Memoria	25
1.4.2	Operaciones de Lectura y Escritura	26
1.4.3	Direccionamiento y decodificación	26
1.4.4	Direccionamiento	29
1.4.5	Construcción de memorias	29
1.4.6	Ampliación de palabra	30
1.4.7	Ampliación de capacidad	30
1.5	Circuitos Secuenciales	32
1.5.1	Flip Flops tipo D	32

List of Figures

1.1	Ejemplo de un mapa de Karnaugh	14
1.2	Mapa de Karnaugh para las variables xy	15
1.3	Ejemplo de representación	16
1.4	Mapa de Karnaugh completo del ejemplo.	16
1.5	Mapa de Karnaugh para agrupar.	17
1.6	Mapa de Karnaugh agrupado.	17
1.7	Estructura de un PLA	18
1.8	Estructura de la función lógica en una PLA.	20
1.9	Decodificador de 3 entradas y 8 salidas	21
1.10	Implementación de un sumador completo con decodificador	22
1.11	Multiplexor de 2 a 1	23
1.12	Implementación de una función booleana con un multiplexor	24
1.13	Diagrama de bloques - Unidad de Memoria	25
1.14	Contenido de una memoria de 1024×16	26
1.15	Ejemplo de organización de una memoria de $1k \times 8$	27
1.16	Diagrama de RAM 4×6	28
1.17	Decodificación bidimensional	28
1.18	Diagrama de bloques de una memoria genérica	29
1.19	Diagrama de una memoria	30
1.20	Diagrama de un circuito de memoria	33

Primer Parcial 1

Fecha: 24/04/2024

Prácticos: 1, 2, 3, 4 y 5.

Temas:

- Sistemas de Numeración.
- Álgebra de Boole.
- Lógica Combinacional.
- Direccionamiento y Lógica de Decodificación de Memorias.
- Circuitos secuenciales.

1.1 Sistemas de Numeración

Un número decimal, como por ejemplo 4543, representa una cantidad igual a 4 millares, 5 centenas, 4 decenas y 3 unidades. En general, un número decimal se puede expresar como una suma de potencias de 10, multiplicadas por los dígitos que lo componen. Por ejemplo, el número 4543 se puede expresar como:

$$4543 = 4 \cdot 10^3 + 5 \cdot 10^2 + 4 \cdot 10^1 + 3 \cdot 10^0$$

En general un número decimal se puede expresar como:

$$a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \dots + a_1 \cdot 10^1 + a_0 \cdot 10^0$$

donde a_i es un dígito decimal y n es el número de dígitos menos uno.

Los coeficientes a_i son números enteros no negativos menores que 10. El valor i indica la posición del dígito en el número, comenzando por la derecha con $i = 0$ y por tanto, la potencia de 10 correspondiente es 10^i .

Decimos que un sistema numérico decimal es base 10 porque usa diez dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. En general, un sistema numérico de base b usa b dígitos, que son los números enteros no negativos menores que b . Por ejemplo, el **sistema binario** es base 2 porque usa dos dígitos: 0 y 1.

1.1.1 Sistema Binario

El sistema binario es un sistema de numeración en el que los números se representan utilizando solamente dos dígitos: 0 y 1.

En este sistema, cada coeficiente a_i es un dígito binario, es decir, un 0 o un 1 y se multiplica por una potencia de 2 en lugar de 10. Por ejemplo, el número binario 1011 se puede expresar como:

$$1011 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

O tomando un número con punto decimal, por ejemplo 11010.11, se puede expresar como:

$$11010.11 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

1.1.2 Sistema Hexadecimal

El sistema hexadecimal es un sistema de numeración en el que los números se representan utilizando dieciséis dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Los dígitos A a F representan los números 10 a 15.

En este sistema, cada coeficiente a_i es un dígito hexadecimal y se multiplica por una potencia de 16. Por ejemplo, el número hexadecimal 2A3 se puede expresar como:

$$2A3 = 2 \cdot 16^2 + A \cdot 16^1 + 3 \cdot 16^0$$

1.1.3 Métodos de Conversión

Hexadecimal a Binario

El sistema hexadecimal es una base 16, por lo que cada dígito puede representar 4 bits. Para pasar de hexadecimal a binario, simplemente se reemplaza cada dígito por su representación en 4 bits. La tabla de conversión es la siguiente:

Hexadecimal	Binario	Hexadecimal	Binario	Hexadecimal	Binario
0	0000	5	0101	A	1010
1	0001	6	0110	B	1011
2	0010	7	0111	C	1100
3	0011	8	1000	D	1101
4	0100	9	1001	E	1110
				F	1111

Por ejemplo, para pasar de 0x123456 a binario, simplemente se reemplaza cada dígito por su representación en 4 bits:

$$0x123456 \rightarrow 0001\ 0010\ 0011\ 0100\ 0101\ 0110$$

Binario a Hexadecimal

Para pasar de binario a hexadecimal, simplemente se agrupan los bits de a 4 y se reemplaza cada grupo por su representación en hexadecimal, teniendo en cuenta las siguientes consideraciones:

- Si el número tiene parte decimal, se agrupan los bits de a 4 a partir del punto decimal y luego se hace la parte entera.
- Si el número de bits no es múltiplo de 4, se agrega un 0 a la izquierda para completar el último grupo.
- En el caso de que en la parte fraccionaria se necesite agregar ceros se agregan a la derecha.

Por ejemplo, para pasar de 10 1100 1101 1011.1100 0010 000 a hexadecimal:

$$\begin{array}{ccccccc} \underbrace{0010}_2 & \underbrace{1100}_C & \underbrace{1101}_D & \underbrace{1011}_B & \underbrace{1100}_C & \underbrace{0010}_2 & \underbrace{0000}_0 \\ (10110011011011.11000010000)_2 = 0x2CDB.C20 \end{array}$$

Hexadecimal a Decimal

Para pasar de hexadecimal a decimal, simplemente se reemplaza cada dígito por su representación en decimal y se multiplica por la potencia de 16 correspondiente. Por ejemplo, para pasar de 0x123456 a decimal:

$$0x123456 = 1 \cdot 16^5 + 2 \cdot 16^4 + 3 \cdot 16^3 + 4 \cdot 16^2 + 5 \cdot 16^1 + 6 \cdot 16^0 = 1193046$$

Decimal a Binario

Para pasar de decimal a binario, simplemente se divide el número por 2 y se toma el resto. Luego se divide el cociente por 2 y se toma el resto, y así sucesivamente hasta que el cociente sea 0. Luego se toman los restos en orden inverso. Por ejemplo, para pasar de 59 a binario:

$$59 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 111011$$

1.1.4 Complemento a 2 para números negativos

El complemento a 2 es una forma de representar números negativos en binario. Para obtener el complemento a 2 de un número negativo, primero se agregan ceros a la izquierda para **completar la cantidad de bits** que de el registro, luego se **invierten todos los bits** y **se le suma 1 al resultado**. Por ejemplo, para obtener el complemento a 2 de -121 en 8 bits:

$$\begin{array}{ll} 121/2 = 60 & \text{residuo } 1 \\ 60/2 = 30 & \text{residuo } 0 \\ 30/2 = 15 & \text{residuo } 0 \\ 15/2 = 7 & \text{residuo } 1 \Rightarrow 01111001 \rightarrow 10000110 + 1 = 10000111 \\ 7/2 = 3 & \text{residuo } 1 \\ 3/2 = 1 & \text{residuo } 1 \\ 1/2 = 0 & \text{residuo } 1 \end{array}$$

Convertir decimal negativo a binario con complemento a 2

1. Calcular el valor absoluto del número en binario.
2. Completar con ceros a la izquierda para que el número tenga la cantidad de bits que se desea.
3. Aplicar el complemento a 2.

Convertir binario con complemento a 2 a decimal

1. Si el bit más significativo es 1, el número es negativo. Se aplica el complemento a 2 para obtener el valor absoluto.
2. Se multiplica cada bit por la potencia de 2 correspondiente y se suma.

1.1.5 Suma de números binarios

La suma de dos números binarios se realiza de la misma forma que la suma de dos números decimales, pero con la diferencia de que el acarreo se produce cuando el resultado de la suma de dos bits es 2 o más. En este caso, el bit de la suma se coloca en la posición correspondiente y se lleva un acarreo a la posición siguiente. Por ejemplo, la suma de 1011 y 1101 se realiza de la siguiente forma: El resultado es 10000 en binario, que es

$$\begin{array}{r}
 1 \ 0 \ 1 \ 1 \ (11) \\
 + \ 1 \ 1 \ 0 \ 1 \ (13) \\
 \hline
 1 \ 0 \ 0 \ 0 \ 0 \ (24)
 \end{array}$$

igual a 16 en decimal. Hay que tener en cuenta las siguientes reglas:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10$

1.1.6 Resta de números binarios

La resta de dos números binarios se realiza de la misma forma que la resta de dos números decimales, pero con la diferencia de que el préstamo se produce cuando el resultado de la resta de dos bits es negativo. En este caso, se toma prestado un bit de la posición siguiente. Por ejemplo, la resta de 1101 y 1011 se realiza de la siguiente forma: El resultado es 100 en binario, que es igual a 4 en decimal. Hay que tener en cuenta las

$$\begin{array}{r}
 1 \ 1 \ 0 \ 1 \ (13) \\
 - \ 1 \ 0 \ 1 \ 1 \ (11) \\
 \hline
 0 \ 1 \ 0 \ 0 \ 0 \ (2)
 \end{array}$$

siguientes reglas:

- $0 - 0 = 0$
- $0 - 1 = 1$ (y llevamos 1)
- $1 - 0 = 1$
- $1 - 1 = 0$

1.1.7 Multiplicación de números binarios

La multiplicación de dos números binarios se realiza teniendo en cuenta dos reglas, la primera regla dice. **Todo número multiplicado por cero es igual a cero** y la segunda, que **uno por uno, es igual a uno**. Luego, el producto se puede hacer de la misma forma a la que se hace en el sistema decimal, esto consiste en multiplicar el multiplicando por cada uno de los dígitos del multiplicador y luego se realiza la suma de los productos. Por ejemplo, la multiplicación de 110 y 10 se realiza de la siguiente forma:

$$\begin{array}{r} \\ \\ \hline \\ + \\ \hline \end{array}$$

1.1.8 Conversiones de bases

El proceso de convertir un número de un sistema numérico a otro se realiza plantenado el número en el sistema original y luego se divide sucesivamente por la base del sistema al que se quiere convertir, tomando el residuo de cada división. El resultado se obtiene tomando los residuos en orden inverso. Por ejemplo, para convertir el número 23 en base 10 a base 2, se realiza el siguiente proceso: Por lo tanto el número 23 en base

División	Cociente	Residuo
23	11	1
11	5	1
5	2	1
2	1	0
1	0	1

10 es igual a 10111 en base 2. Para convertir un número de base 2 a base 10, se realiza el proceso inverso, multiplicando cada dígito por la potencia de 2 correspondiente a su posición y sumando los resultados.

La conversión de enteros decimales a cualquier base r se puede realizar mediante el algoritmo de la división sucesiva. El algoritmo consiste en dividir el número decimal por la base r y tomar el residuo. Luego, se divide el cociente obtenido por la base r y se toma el residuo. Este proceso se repite hasta que el cociente sea cero. El número en base r se obtiene tomando los residuos en orden inverso.

1.1.9 Sistema IEEE 754

El sistema IEEE 754 es un estándar para la representación de números en punto flotante. Este sistema establece las características de las tres partes de un número en punto flotante: el **signo**, **exponente** y **fracción (mantisa)**. Se enfoca en el método de precisión simple que utiliza 32 bits.

Características

Bit de signo:

El bit de signo es el primer bit del número en punto flotante. **Si el bit de signo es 0**, el número es **positivo**, **si es 1**, el número es **negativo**.

Exponente:

Indica cuántos “lugares” se debe desplazar hacia la derecha o hacia la izquierda la coma binaria de la parte significativa. El exponente de un número puede ser tanto positivo como negativo. En el caso de la precisión simple, el exponente se representa con 8 bits y se suma 127 al valor del exponente para obtener el valor.

Mantisa o parte fraccionaria:

La mantisa es la parte fraccionaria del número en punto flotante. En el caso de la precisión simple, la mantisa se representa con 23 bits. Debe ser normalizada, eso se logra moviendo la coma binaria a la izquierda hasta que el primer bit sea 1.

$$1,xxxxx \times 2^{yyyy}$$

Al hacer esto, se debe adaptar el exponente para que el valor no se modifique, es decir, el exponente será ahora **la cantidad de veces que se movió la coma binaria a la izquierda**.

1.1.10 Convertir de decimal a IEEE 754

1. Determinar el bit de signo.
2. Convertir la parte entera y la parte fraccionaria a binario.
3. Normalizar.
4. Encontrar el exponente moviendo la coma y sumando 127, expresado en binario.
5. Encontrar la parte fraccionaria.
6. Conformar el número en punto flotante.

1.1.11 Convertir de IEEE 754 a decimal

1. Dividir el número en punto flotante en sus tres partes.
2. Encontrar el bit de signo.
3. Encontrar el exponente y restarle 127.
4. Desnormalizar el número y pasarlo a decimal.

1.1.12 Ejemplo de conversión de decimal a IEEE 754 - Completo

Se busca convertir el número 723.125_{10} a IEEE 754.

1. El bit de signo es 0, ya que el número es positivo.
2. Se convierte la parte entera y la parte fraccionaria a binario.
 - Parte entera:

$$723 \div 2 = 361 \text{ residuo } 1$$

$$361 \div 2 = 180 \text{ residuo } 1$$

$$180 \div 2 = 90 \text{ residuo } 0$$

$$90 \div 2 = 45 \text{ residuo } 0$$

$$45 \div 2 = 22 \text{ residuo } 1$$

$$22 \div 2 = 11 \text{ residuo } 0$$

$$11 \div 2 = 5 \text{ residuo } 1$$

$$5 \div 2 = 2 \text{ residuo } 1$$

$$2 \div 2 = 1 \text{ residuo } 0$$

$$1 \div 2 = 0 \text{ residuo } 1$$

Por lo que la parte entera en binario es 1011010011.

- Parte fraccionaria:

$$0.125 \times 2 = 0.25$$

$$0.25 \times 2 = 0.5$$

$$0.5 \times 2 = 1$$

Por lo que la parte fraccionaria en binario es 001.

El número en binario es 1011010011.001.

3. Para normalizar movemos la coma tantos lugares como sea necesario para que quede un uno seguido por una parte fraccionaria.

- **Sin normalizar:** $1011010011.001 \times 2^0$
- **Normalizado:** $1.011010011001 \times 2^{10}$

4. El exponente es la cantidad de veces que se movió la coma binaria a la izquierda, en este caso, 10 veces. Por lo que el exponente es $127 + 10 = 137_{10} = 10001001_2$.
5. La parte fraccionaria se compone por los 23 bits que siguen a la coma decimal del número normalizado.

01101001100100000000000

6. Para conformar el número en punto flotante, se coloca el bit de signo, el exponente y la parte fraccionaria.

0 10001001 01101001100100000000000

1.2 Álgebra de Boole

El álgebra booleana, al igual que todos los sistemas matemáticos deductivos, se define con un conjunto de elementos, un conjunto de operadores y varios axiomas o postulados no demostrados. Un conjunto de elementos es cualquier colección de objetos con alguna propiedad en común.

1.2.1 Elementos del álgebra booleana

- **0 y 1:** Son los dos elementos básicos del álgebra booleana. Se les llama elementos binarios o bits. El 0 representa el estado de apagado o falso, mientras que el 1 representa el estado de encendido o verdadero.
- **Variables:** Son símbolos que representan cantidades desconocidas o no especificadas. En el álgebra booleana, las variables pueden tomar uno de los dos valores posibles, 0 o 1.

1.2.2 Teoremas y postulados del álgebra booleana

Los teoremas y postulados que se presentan son las relaciones más básicas del álgebra booleana. Los teoremas, al igual que los postulados, se presentan por pares; cada relación es el dual de su pareja. Los postulados son axiomas básicos de la estructura algebraica y no requieren demostración. Los teoremas deben demostrarse a partir de los postulados.

Postulados y teoremas del álgebra booleana

Postulado 2	a)	$x + 0 = x$	b)	$x \cdot 1 = x$
Postulado 5	a)	$x + x' = 1$	b)	$x \cdot x' = 0$
Teorema 1	a)	$x + x = x$	b)	$x \cdot x = x$
Teorema 2	a)	$x + 1 = 1$	b)	$x \cdot 0 = 0$
Teorema 3, involución		$(x')' = x$		
Postulado 3, conmutatividad	a)	$x + y = y + x$	b)	$xy = yx$
Teorema 4, asociatividad	a)	$x + (y + z) = (x + y) + z$	b)	$x(yz) = (xy)z$
Postulado 4, distributividad	a)	$x(y + z) = xy + xz$	b)	$x + yz = (x + y)(x + z)$
Teorema 5, DeMorgan	a)	$(x + y)' = x'y'$	b)	$(xy)' = x' + y'$
Teorema 6, absorción	a)	$x + xy = x$	b)	$x(x + y) = x$

1.2.3 Funciones booleanas

El álgebra booleana es un álgebra que se ocupa de variables binarias y operaciones lógicas. Una función booleana descrita por una expresión algebraica consta de variables binarias, las constantes 0 y 1, y los símbolos lógicos de operación. Para un valor dado de las variables binarias, la función puede ser igual a 1 o bien a 0.

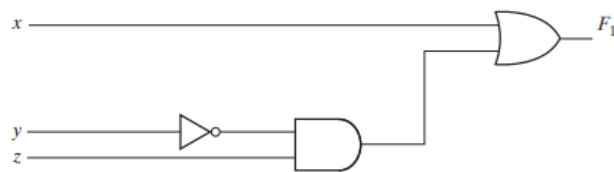
Una función booleana expresa la relación lógica entre variables binarias. Se evalúa determinando el valor binario de la expresión para todos los posibles valores de las variables. Podemos representar una función booleana en una tabla de verdad. Una tabla de verdad es una lista de combinaciones de unos y ceros asignados a las variables binarias y una columna que muestra el valor de la función para cada combinación binaria. El número de filas de la tabla es 2^n , donde n es el número de variables de la función. Las combinaciones binarias para la tabla de verdad se obtienen de los números binarios, contando de 0 hasta $2^n - 1$.

Por ejemplo, supongamos que se tiene la función: $F_1 = x + y'z$:

La tabla de la verdad es la siguiente:

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1


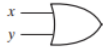


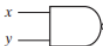



Y la implementación de la función en un circuito lógico sería:



1.2.4 Compuertas Lógicas digitales

Puesto que las funciones booleanas se expresan en términos de operaciones AND, OR y NOT, es más fácil implementar una función booleana con estos tipos de compuertas. La posibilidad de construir compuertas para las otras operaciones lógicas tiene interés práctico.

Cada compuerta tiene una o dos variables binarias de entrada designadas con x y y , y una variable binaria de salida designada con F .

Nombre	Símbolo gráfico	Función algebraica	Tabla de verdad															
AND		$F = xy$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inversor		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Búfer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
OR exclusivo (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
NOR exclusivo o equivalencia		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

1.3 Lógica Combinacional

1.3.1 Producto de Sumas y Suma de Productos (Maxitérminos y Mintérminos)

Las funciones lógicas se pueden expresar de dos formas diferentes, como una suma de productos o como un producto de sumas. La forma de obtener dicha expresión se basa en identificar que valores de salida tomarán el valor de 1 y el valor de 0.

Supongamos que se tiene una función lógica de tres variables x , y y z , con la siguiente tabla de verdad:

x	y	z	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

1. Para expresar como **suma de productos**, se toman los minterminos que hacen que la función devuelva 1, luego se suman los minterminos.
2. Para expresar como **producto de sumas**, se toman los maxiterminos que hacen que la función devuelva 0, luego se multiplican los maxiterminos.

x	y	z	S	Minterminos	Maxiterminos
0	0	0	0	$m_0 = x'y'z'$	$M_0 = x + y + z$
0	0	1	1	$m_1 = x'y'z$	$M_1 = x + y + z'$
0	1	0	1	$m_2 = x'yz'$	$M_2 = x + y' + z$
0	1	1	0	$m_3 = x'yz$	$M_3 = x + y' + z'$
1	0	0	1	$m_4 = xy'z'$	$M_4 = x' + y + z$
1	0	1	0	$m_5 = xy'z$	$M_5 = x' + y + z'$
1	1	0	0	$m_6 = xyz'$	$M_6 = x' + y' + z$
1	1	1	1	$m_7 = xyz$	$M_7 = x' + y' + z'$

Con esto se tienen las formas canónicas de la función lógica, las cuales se pueden simplificar utilizando mapas de Karnaugh.

- **Suma de productos:** $S = m_1 + m_2 + m_4 + m_7 = x'y'z + x'yz' + xy'z' + xyz$,
- **Producto de sumas:** $S = M_0 \cdot M_3 \cdot M_5 \cdot M_6 = (x + y + z)(x + y' + z')(x' + y + z')(x' + y' + z)$.

1.3.2 Mapas de Karnaugh

Los mapas de Karnaugh son una herramienta útil que nos permite aplicar un método de simplificación de funciones lógicas.

El Mapa de Karnaugh tiene la característica de que puede ser visto como una representación bidimensional de una tabla de verdad. En la tabla de verdad, se colocan las variables por columnas y las combinaciones de tales variables determinan un valor de salida, 0 o 1, sin embargo, en el mapa las variables se colocan como si de un plano cartesiano se tratara, respetando cada una de las combinaciones que de ellas se generan, y colocando en la intersección de las combinaciones de las variables, el valor de salida.

		zw			
		00	01	11	10
xy	00	1	1	1	1
	01	1	0	0	0
	11	1	0	0	0
	10	1	1	1	0

Figure 1.1: Ejemplo de un mapa de Karnaugh

Los mapas muestran la relación que existe entre las entradas y las salidas de un circuito lógico, si se aplica adecuadamente el resultado será el más simplificado posible. Pueden ser utilizados para cualquier número de variables de entrada sin embargo se recomienda un máximo de seis variables.

Representación de tabla de verdad en un mapa de Karnaugh

Las tablas de la verdad se pueden representar en un mapa de Karnaugh, para ello se deben seguir los siguientes pasos:

1. Se deben identificar los términos de la tabla de verdad que son 1.
2. Se deben identificar los términos de la tabla de verdad que son 0.
3. Se deben identificar los términos de la tabla de verdad que son 1 y que se pueden agrupar.
4. Se deben identificar los términos de la tabla de verdad que son 0 y que se pueden agrupar.

Observaciones:

- Los términos de la tabla de verdad que son 1 se representan en el mapa de Karnaugh con un 1, y sirven para expresar a la función como una suma de productos.
- Los términos de la tabla de verdad que son 0 se representan en el mapa de Karnaugh con un 0 y sirven para expresar a la función como un producto de sumas.

Ejemplo de representación de tabla de verdad en un mapa de Karnaugh

Supongamos que se tiene la siguiente tabla de la verdad:

x	y	z	w	S
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Para hacer un mapa de Karnaugh, primero se separan las variables xy y zw y se colocan en el mapa de la siguiente forma:

		zw			
		00	01	11	10
xy	00	-	-	-	-
	01	-	-	-	-
	11	-	-	-	-
	10	-	-	-	-

Figure 1.2: Mapa de Karnaugh para las variables xy

Luego identificamos los términos de la tabla de verdad que son 1 y los colocamos en el mapa de Karnaugh. Por ejemplo esta línea de la tabla

x	y	z	w	S
0	0	0	1	1

Se coloca en el mapa de Karnaugh en la posición donde xy toma 00 y zw toma 01.

		zw			
		00	01	11	10
xy	00	0	1	0	0
	01	0	0	0	0
	11	0	0	0	0
	10	0	0	0	0

Figure 1.3: Ejemplo de representación

Pasar por cada línea de la tabla de verdad y colocar los términos en el mapa de Karnaugh. Los términos que no son 1 se colocan como 0 en el mapa de Karnaugh.

		yx			
		00	01	11	10
wz	00	0	1	0	1
	01	1	0	1	0
	11	0	1	0	1
	10	1	0	1	0

Figure 1.4: Mapa de Karnaugh completo del ejemplo.

Agrupación de términos

Para simplificar la función lógica, se deben agrupar los términos que son 1 en el mapa de Karnaugh. Los términos se pueden agrupar siguiendo las siguientes reglas:

1. Los grupos deben contener una cantidad de elementos igual a potencias de 2. Es decir, los grupos solo podrán realizarse con cantidades de celdas iguales a: 1, 2, 4, 8, 16, 32, etc.
2. Se deben generar grupos con combinaciones de variables adyacentes, es decir, sólo debe de haber un cambio entre cada una de las combinaciones, por ejemplo la combinación ABC' es adyacente con $AB'C'$, ya que sólo cambia la variable B .

3. Grupos en las posiciones en los extremos y esquinas del mapa. Las posiciones de los extremos y las esquinas son adyacentes, por lo que se pueden agrupar. Esta adyacencia se debe a que el mapa de Karnaugh puede ser visto como un toroide.
4. Grupos de unos darán lugar a una suma de productos o mini términos.
5. Grupos de ceros darán lugar a un producto de sumas o maxi términos.
6. Los grupos deben ser lo más grande posible. Se buscará realizar grupos con la mayor cantidad de elementos posibles, entre más grande el grupo, se obtiene una función más simplificada.
7. No se pueden generar grupos en diagonal. Solo se permitirán grupos en vertical y horizontal dentro del mapa.
8. Puede existir solapamiento de grupos, siempre y cuando exista al menos un elemento que no haya sido agrupado anteriormente.
9. No deben existir grupos redundantes, es decir, no se puede realizar un grupo dentro de otro grupo, y no se puede realizar un grupo con elementos que ya hayan sido completamente agrupados en otros conjuntos.

Por ejemplo, si se tiene este mapa de Karnaugh:

		<i>yx</i>			
		00	01	11	10
<i>wz</i>	00	1	0	0	1
	01	1	0	0	1
	11	1	0	0	1
	10	0	0	0	0

Figure 1.5: Mapa de Karnaugh para agrupar.

Se debería agrupar de la siguiente forma:

		<i>zw</i>			
		00	01	11	10
<i>xy</i>	00	1	0	0	1
	01	1	0	0	1
	11	1	0	0	1
	10	0	0	0	0

Figure 1.6: Mapa de Karnaugh agrupado.

Simplificación de la función lógica

Para la simplificación de la función lógica, se usan los grupos que se han formado en el mapa de Karnaugh. Cada grupo se convierte en un término de la función lógica, y se deben sumar los términos que se obtienen de los grupos, teniendo en cuenta lo siguiente:

1. Si un grupo es de una sola celda, se incluye igualmente en la función lógica.
2. En el grupo, las variables que cambian de valor "se me van" del término, y las que no cambian su valor permanecen.
3. Si un grupo se solapa con otro, no cambia la forma de trabajarlos, cada grupo es un término distinto.

Por ejemplo, tomando el mapa de Karnaugh anterior, se obtiene la siguiente función lógica:

$$F = x'w' + yw'$$

1.3.3 PLA (Programmable Logic Array)

Programmable Logic Array (PLA) es un dispositivo lógico de arquitectura fija con puertas AND programables seguidas de puertas OR programables. PLA es básicamente un tipo de dispositivo lógico programable que se utiliza para construir un circuito digital reconfigurable. La estructura de un PLA es la siguiente:

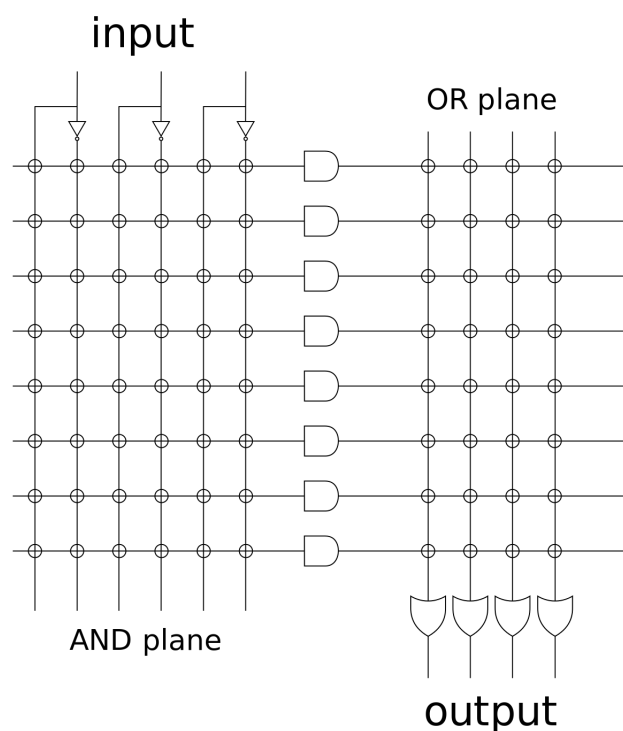


Figure 1.7: Estructura de un PLA

El funcionamiento se podría resumir en tres pasos:

1. **Programación:** el usuario define la función lógica que se desea implementar.
2. **Generación de términos del producto:** las entradas se aplican a la matriz de puertas AND para producir un conjunto de términos de producto.
3. **Generación de suma de términos:** los términos de producto se aplican a la matriz de puertas OR para producir la salida.

Método para implementar una función lógica en un PLA

1. Una vez se tenga la función lógica simplificada, se deben identificar los términos de la función.
2. Cada término de la función se convierte en una fila de la matriz de puertas AND.
3. Luego se suman los términos de la función y se convierten en una fila de la matriz de puertas OR.
4. La salida de la matriz de puertas OR es la función lógica implementada en el PLA.

1.3.4 Ejemplo completo de Lógica Combinacional

Un detector de paridad impar de 4 entradas y una salida funciona de la siguiente manera: si la cantidad de entradas con valor '1' es impar la salida se pone en '1', en el resto de los casos la salida toma valor '0'.

1. Construir la tabla de verdad para dicho sistema.
2. Obtener la ecuación lógica como suma de minitérminos y producto de maxitérminos (funciones canónicas).
3. Implementar el sistema con compuertas NAND de la cantidad de entradas requeridas.
4. Implementar el sistema con una PLA.

Punto 1

Para construir la tabla de verdad, se deben considerar las 4 entradas posibles y la salida que se obtiene para cada combinación de entradas.

A	B	C	D	S	Minitérminos	Maxitérminos
0	0	0	0	0	$m_0 = A'B'C'D'$	$M_0 = A+B+C+D$
0	0	0	1	1	$m_1 = A'B'C'D$	$M_1 = A+B+C+D'$
0	0	1	0	1	$m_2 = A'B'CD'$	$M_2 = A+B+C'+D$
0	0	1	1	0	$m_3 = A'B'CD$	$M_3 = A+B+C'+D'$
0	1	0	0	1	$m_4 = A'BC'D'$	$M_4 = A+B'+C+D$
0	1	0	1	0	$m_5 = A'BC'D$	$M_5 = A+B'+C+D'$
0	1	1	0	0	$m_6 = A'BCD'$	$M_6 = A+B'+C'+D$
0	1	1	1	1	$m_7 = A'BCD$	$M_7 = A+B'+C'+D'$
1	0	0	0	1	$m_8 = AB'C'D'$	$M_8 = A'+B+C+D$
1	0	0	1	0	$m_9 = AB'C'D$	$M_9 = A'+B+C+D'$
1	0	1	0	0	$m_{10} = AB'CD'$	$M_{10} = A'+B+C'+D$
1	0	1	1	1	$m_{11} = AB'CD$	$M_{11} = A'+B+C'+D'$
1	1	0	0	0	$m_{12} = ABC'D'$	$M_{12} = A'+B'+C+D$
1	1	0	1	1	$m_{13} = ABC'D$	$M_{13} = A'+B'+C+D'$
1	1	1	0	1	$m_{14} = ABCD'$	$M_{14} = A'+B'+C'+D$
1	1	1	1	0	$m_{15} = ABCD$	$M_{15} = A'+B'+C'+D'$

Punto 2

Para obtener la ecuación lógica como suma de minitérminos y producto de maxitérminos, se deben identificar los términos que hacen que la función devuelva 1 y los términos que hacen que la función devuelva 0.

- **Suma de minitérminos:** tomo los términos que hacen 1 la función y los sumo.

$$F = m_1 + m_2 + m_4 + m_7 + m_8 + m_{11} + m_{13} + m_{14}$$

Desarrollando la ecuación, se obtiene:

$$F = A'B'C'D + A'B'CD' + A'BC'D' + A'BCD + AB'C'D' + AB'CD + ABC'D + ABCD'$$

- **Producto de maxitérminos:** tomo los términos que hacen 0 la función y los multiplico.

$$F = M_0 \cdot M_3 \cdot M_5 \cdot M_6 \cdot M_9 \cdot M_{10} \cdot M_{12} \cdot M_{15}$$

Desarrollando la ecuación, se obtiene:

$$F = (A + B + C + D)(A + B + C' + D')(A + B' + C + D')(A + B' + C' + D)(A' + B + C + D') \\ (A' + B + C' + D')(A' + B' + C + D)(A' + B' + C' + D')$$

Punto 3

Para implementar el sistema con compuertas NAND, se deben seguir los siguientes pasos:

1. Se toma la forma canónica de la función lógica, en este caso se tomará la forma de suma de minitérminos.
2. Luego se niega dos veces la función lógica, para obtener la forma de producto de maxitérminos.
3. Se aplica una vez más la ley de De Morgan para obtener la forma buscada.

$$F = A'B'C'D + A'B'CD' + A'BC'D' + A'BCD + AB'C'D' + AB'CD + ABC'D + ABCD' \\ = (A'B'C'D + A'B'CD' + A'BC'D' + A'BCD + AB'C'D' + AB'CD + ABC'D + ABCD')'' \\ = (A'B'C'D)' \cdot (A'B'CD')' \cdot (A'BC'D')' \cdot (A'BCD)' \cdot (AB'C'D')' \cdot (AB'CD)' \cdot (ABC'D)' \cdot (ABCD')'$$

Punto 4

Para implementar el sistema con una PLA, se utilizará la forma canónica de la función lógica, en este caso se tomará la forma de suma de minitérminos.

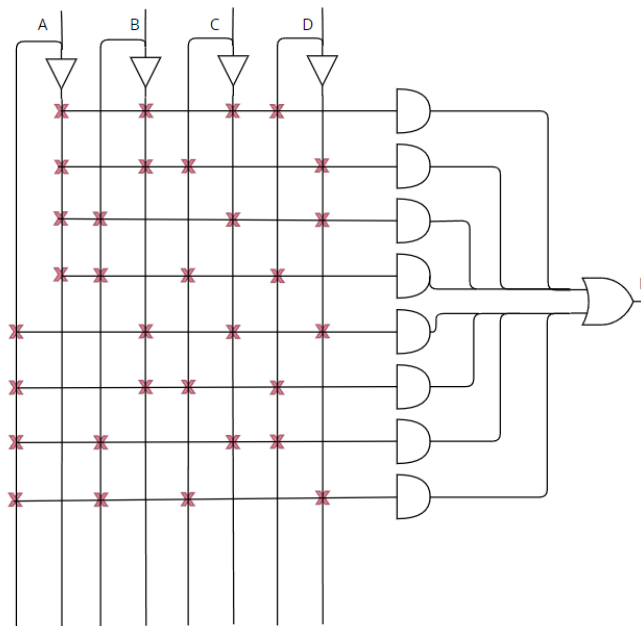


Figure 1.8: Estructura de la función lógica en una PLA.

1.3.5 Decodificadores

Es un circuito combinacional cuya función es detectar la presencia de una determinada combinación de bits en sus entradas y señalar la presencia de este código mediante un cierto nivel de salida.

Un decodificador posee N líneas de entrada para gestionar N bits y en una de las 2^N líneas de salida indica la presencia de una o mas combinaciones de n bits. Es decir, para cualquier código dado en las entradas solo se activa una de las N posibles salidas.

Por ejemplo un decodificador de 3 entradas y 8 salidas, se activa una de las 8 salidas según la combinación de los 3 bits de entrada.

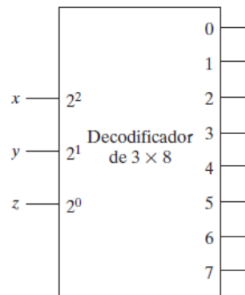


Figure 1.9: Decodificador de 3 entradas y 8 salidas

La tabla de verdad de un decodificador de 3 entradas y 8 salidas es la siguiente:

X	Y	Z	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Implementación de lógica con decodificadores

Un decodificador produce los 2^n minitérminos de n variables de entrada. Puesto que cualquier función booleana es susceptible de expresarse como suma de minitérminos, es posible utilizar un decodificador para generar los minitérminos y una compuerta OR externa para formar la suma lógica. Así, cualquier circuito combinacional con n entradas y m salidas se puede implementar con un decodificador de n a 2^n líneas y m compuertas OR.

El procedimiento para implementar un circuito combinacional con un decodificador y compuertas OR requiere expresar la función booleana del circuito como suma de minitérminos. Entonces se escoge un decodificador que genere todos los minitérminos de las variables de entrada. Las entradas a cada compuerta OR se escogen de entre las salidas del decodificador, de acuerdo con la lista de minitérminos de cada función.

Ejemplo de Implementación

Se tiene que implementar la lógica de un sumador completo, la tabla es la siguiente:

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

De la tabla obtenemos las funciones para el circuito combinacional en forma de suma de minitérminos:

$$S(x, y, z) = \sum(1, 2, 4, 7)$$

$$C_{out}(x, y, z) = \sum(3, 5, 6, 7)$$

Puesto que hay tres entradas y un total de ocho minitérminos, se necesita un decodificador de 3 a 8 líneas. El decodificador genera los ocho minitérminos para x , y , z . La compuerta OR de la salida S forma la suma lógica de los minitérminos 1, 2, 4 y 7. La compuerta OR de la salida C_{out} forma la suma lógica de los minitérminos 3, 5, 6 y 7.

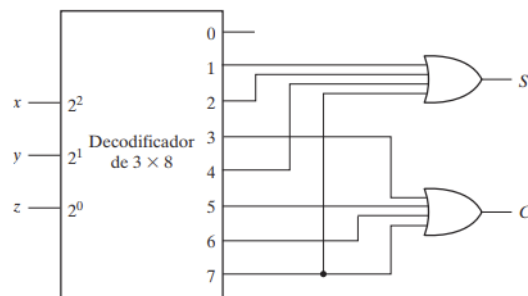


Figure 1.10: Implementación de un sumador completo con decodificador

1.3.6 Multiplexores

Un multiplexor es un circuito combinacional que selecciona información binaria de una de muchas líneas de entrada y la envía a una sola línea de salida. La selección de una línea de entrada dada se controla con un conjunto de líneas de selección. Normalmente, hay 2^n líneas de entrada y n líneas de selección cuyas combinaciones de bits determinan cuál entrada se selecciona.

Multiplexor de 2 a 1

Un multiplexor de 2 líneas a 1 conecta una de dos fuentes de un bit a un destino común. El circuito tiene dos líneas de entrada de datos, una línea de salida y una línea de selección S . Cuando $S = 0$, se habilita la compuerta AND de arriba e I_0 cuenta con una trayectoria hacia la salida. Cuando $S = 1$, la compuerta AND inferior está habilitada e I_1 tiene una trayectoria hacia la salida. El multiplexor actúa como un interruptor electrónico que selecciona una de dos fuentes. El diagrama de bloques de un multiplexor a veces se representa con un símbolo en forma de cuña. Esto sugiere visualmente cómo una fuente de datos, seleccionada de entre varias, se dirige a un solo destino. En los diagramas de bloques es común rotular los multiplexores como MUX.

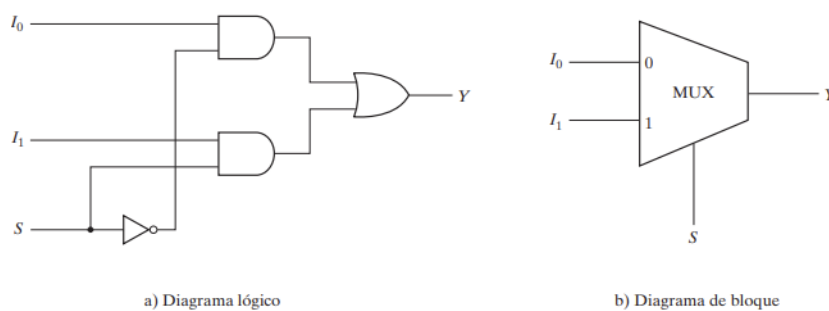


Figure 1.11: Multiplexor de 2 a 1

Implementación de funciones lógicas con multiplexores

Un examen del diagrama lógico de un multiplexor revela que básicamente es un decodificador con una compuerta OR incluida en la unidad. Los minitérminos de una función se generan en un multiplexor mediante el circuito asociado a las entradas de selección. Los minitérminos individuales se pueden seleccionar con las entradas de datos. Esto ofrece un método para implementar una función booleana de n variables con un multiplexor que tiene n entradas de selección y 2^n entradas de datos, una para cada minitérmino.

Para implementar una función booleana de n variables con un multiplexor que tiene $n - 1$ entradas de selección. Las primeras $n - 1$ variables de la función se conectan a las entradas de selección del multiplexor. La variable restante de la función se utiliza para las entradas de datos. Si denotamos esa variable con z , cada entrada de datos del multiplexor será, z , z' , 1 o 0.

Ejemplo de Implementación

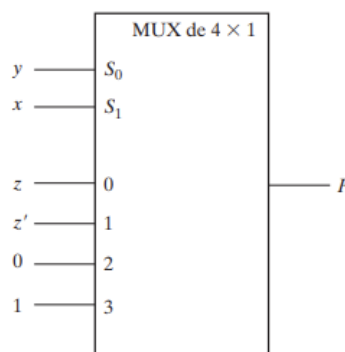
Consideremos la función booleana de tres variables:

$$F(x, y, z) = \sum(1, 2, 6, 7)$$

La función puede implementarse con un multiplexor de 4 líneas a 1. Las dos variables x e y se aplican a las líneas de selección en ese orden; x se conecta la entrada S_1 y y se conecta a S_0 . Los valores de las líneas de entrada de datos se deducen de la tabla de verdad de la función. Cuando $xy = 00$, la salida F es igual a z porque $F = 0$ cuando $z = 0$ y $F = 1$ cuando $z = 1$. Esto requiere aplicar la variable z a la entrada de datos 0. El funcionamiento del multiplexor es tal que, cuando $xy = 00$, la entrada de datos 0 tiene una trayectoria hacia la salida y eso hace que F sea igual a z . De forma similar, podemos determinar las entradas que deben recibir las líneas de datos 1, 2 y 3, a partir del valor de F cuando $xy = 01, 10$ y 11 , respectivamente.

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

a) Tabla de verdad



b) Implementación con multiplexor

Figure 1.12: Implementación de una función booleana con un multiplexor

1.4 Direccionamiento y Lógica de Decodificación de Memorias

Una unidad de memoria es un conjunto de celdas de almacenamiento junto con los circuitos asociados que se requieren para transferir información al y del dispositivo. El tiempo que toma transferir información a o de cualquier posición al azar deseada siempre es el mismo, de ahí el nombre memoria de acceso aleatorio o RAM. Una unidad de memoria almacena información binaria en grupos de bits llamados **palabras**. Una palabra de memoria es una *entidad de bits que siempre se guardan o sacan juntos*, como una unidad. Una palabra de memoria es un *grupo de unos y ceros y podría representar un número, una instrucción, uno o más caracteres alfanuméricos o cualquier otra información codificada en binario*. Un grupo de ocho bits es un byte. Casi todas las memorias de computadora manejan palabras **cuya longitud es un múltiplo de ocho bits**. Así, una palabra de 16 bits contiene dos bytes, y una de 32 bits consta de cuatro bytes. La capacidad de una unidad de memoria por lo regular se da como el número total de bytes que es capaz de guardar.

1.4.1 Comunicación con la Memoria

La comunicación entre la memoria y su entorno se efectúa a través de líneas de entrada y salida de datos, líneas de selección de direcciones y líneas de control que especifican la dirección de la transferencia.

Las n líneas de entrada de datos alimentan la información que se guardará en la memoria, y las n líneas de salida de datos proporcionan la información que viene de la memoria. Las k líneas de dirección especifican la palabra específica escogida, de entre muchas disponibles. Las dos entradas de control especifican la dirección de la transferencia deseada: la entrada de escritura hace que se transfieran datos binarios a la memoria; la de lectura hace que se saquen datos binarios de la memoria.

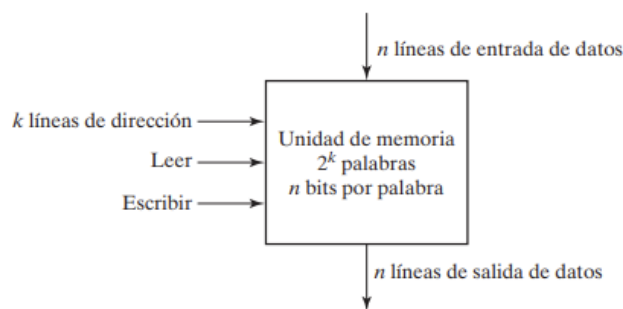


Figure 1.13: Diagrama de bloques - Unidad de Memoria

La unidad de memoria se especifica con el número de palabras que contiene y el número de bits que hay en cada palabra. Las líneas de dirección seleccionan una palabra específica. A cada palabra de la memoria se asigna un número de identificación, llamado dirección, entre 0 y $2^k - 1$, donde k es el número de líneas de dirección. La selección de una palabra específica de la memoria se efectúa aplicando los k bits de dirección a las líneas de dirección. Un decodificador acepta esta dirección y abre las trayectorias necesarias para seleccionar la palabra especificada.

Para nombrar a las memorias, se suele decir que es de una capacidad y de tantos bits. La capacidad de palabras (o bytes) de la memoria se da con letras tales como: K (kilo), M (mega), G (giga), T (tera). K es igual a 2^{10} , M es igual a 2^{20} , G es igual a 2^{30} y T es igual a 2^{40} .

Por ejemplo, la unidad de memoria con capacidad de 1K palabras de 16 bits cada una. Puesto que $1K=1024=2^{10}$ y 16 bits constituyen dos bytes, se afirma que la memoria puede dar cabida a $2048=2K$ bytes.

El diagrama correspondiente a una memoria de 1K palabras de 16 bits cada una se muestra en la figura 1.14.

Dirección de memoria		Contenido de la memoria
Binaria	Decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Figure 1.14: Contenido de una memoria de 1024 x 16

Cada palabra contiene 16 bits que se dividen en dos bytes. Las palabras se reconocen por su dirección decimal, de 0 a 1023. La dirección binaria equivalente consta de 10 bits. La primera dirección se especifica con 10 ceros, y la última, con 10 unos. Ello se debe a que 1023 en binario es 111111111. Seleccionamos una palabra de memoria por su dirección binaria. Cuando se lee o escribe una palabra, la memoria opera sobre los 16 bits como una sola unidad.

La memoria de $1K \times 16$ de la figura 1.14 tiene 10 bits en la dirección y 16 bits en cada palabra.

1.4.2 Operaciones de Lectura y Escritura

Las dos operaciones que efectúa una memoria de acceso aleatorio son escritura y lectura. La señal de escritura especifica una operación de transferencia hacia adentro, y la de lectura, una de transferencia hacia afuera. Al aceptar una de estas señales de control, los circuitos internos de la memoria efectúan la operación deseada. Los pasos que deben seguirse para transferir una nueva palabra a la memoria son:

1. Aplique la dirección binaria de la localidad deseada a las líneas de dirección.
2. Aplique a las líneas de entrada de datos los bits de datos que se guardarán en la memoria.
3. Active la entrada escribir.

La unidad de memoria tomará entonces los bits de las líneas de datos de entrada y los almacenará en la localidad especificada por las líneas de dirección. Los pasos que deben seguirse para sacar de la memoria una palabra almacenada son:

1. Aplique a las líneas de dirección la dirección binaria de la localidad deseada.
2. Active la entrada leer.

La unidad de memoria tomará entonces los bits de la localidad seleccionada por la dirección y los aplicará a las líneas de datos de salida. El contenido de la localidad seleccionada no cambia después de la lectura.

1.4.3 Direccionamiento y decodificación

Construcción interna de una memoria

La construcción interna de una memoria de acceso aleatorio de m palabras y n bits por palabra consta de $m \times n$ **celdas binarias de almacenamiento** y los correspondientes circuitos de decodificación que *seleccionan*

palabras individuales. La celda binaria de almacenamiento es el bloque de construcción básico de una unidad de memoria y es un circuito electrónico con entre cuatro y seis transistores.

La celda binaria almacena un bit en su latch interno. La entrada de selección habilita a la celda para leer o escribir, y la entrada leer/escribir determina la operación de la celda. Un 1 en esa entrada hace que se efectúe la operación de lectura porque establece una trayectoria entre el latch y la terminal de salida. Un 0 en la entrada de leer/escribir hace que se efectúe la operación de escritura porque establece una trayectoria entre la terminal de entrada y el latch

Por ejemplo, una memoria que contiene palabras de 8 bits, las celdas de almacenamiento de la memoria serían de manera ilustrativa como se muestra en la figura 1.15.

		Largo de Palabra (8 bits)							
Binario	Decimal								
000000000	0	0	1	1	1	0	0	0	0
000000001	1	1	1	1	1	1	1	1	1
000000010	2	0	0	0	1	0	0	1	1
.
.
.
.
111111101	1021	1	1	1	1	0	0	0	0
111111110	1022	1	1	0	0	0	0	0	0
111111111	1023	0	0	0	0	1	1	1	1

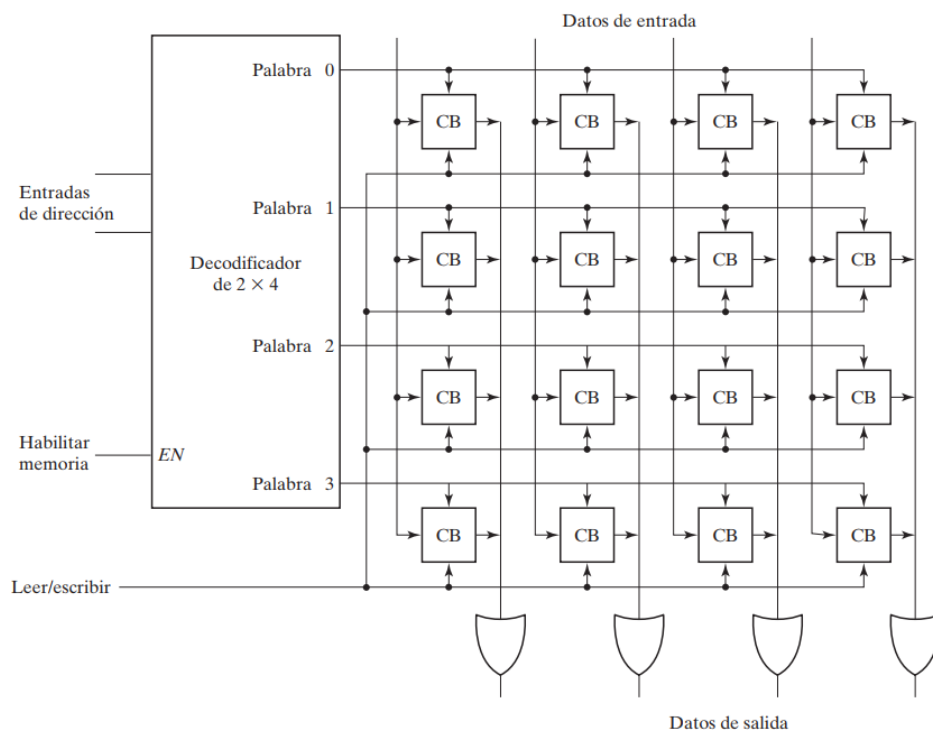
← Palabra
direccionada

← Celda de
Almacenamiento
de bits

Figure 1.15: Ejemplo de organización de una memoria de $1k \times 8$

Selección de una palabra de memoria

Para seleccionar una palabra de memoria, se requiere un decodificador que acepte la dirección de la palabra y abra las trayectorias necesarias para seleccionar la palabra especificada. La dirección de la palabra se aplica a las entradas del decodificador, y la salida correspondiente se activa. La salida activada se conecta a la línea de selección de palabra de memoria. La figura 1.16 se ilustra la construcción lógica de una RAM pequeña. Consta de cuatro palabras de cuatro bits cada una y tiene un total de 16 celdas binarias. Los pequeños bloques rotulados CB representan las celdas binarias con sus tres entradas y una salida. Una memoria de cuatro palabras necesita dos líneas de dirección. Las dos entradas de dirección pasan por un decodificador de 2×4 para seleccionar una de las cuatro palabras. El decodificador se habilita con la entrada de habilitar memoria. Si esa señal es 0, todas las salidas del decodificador son 0 y no se selecciona ninguna de las palabras de memoria. Si la señal es 1, se selecciona una de las cuatro palabras, especificada por el valor de las dos líneas de dirección. Una vez seleccionada una palabra, la entrada leer/escribir determina la operación. Durante la operación de lectura, los cuatro bits de la palabra seleccionada pasan por compuertas OR a las terminales de salida. Durante la operación de escritura, los datos disponibles en las líneas de entrada se transfieren a las cuatro celdas binarias de la palabra seleccionada. Las celdas binarias que no están seleccionadas se inhabilitan, y sus valores binarios anteriores no cambian. Si la entrada de selección de memoria que llega al decodificador es 0, no se selecciona ninguna de las palabras y el contenido de todas las celdas permanece inalterado, sea cual sea el valor de la entrada leer/escribir.

Figure 1.16: Diagrama de RAM 4×6

Decodificación bidimensional

La idea básica de la decodificación bidimensional es acomodar las celdas de memoria en un arreglo lo más cercano posible a un cuadrado. En esta configuración, se usan dos decodificadores con $k/2$ entradas cada uno, en vez de un decodificador con k entradas. Un decodificador selecciona la fila y el otro selecciona la columna de una configuración de matriz bidimensional.

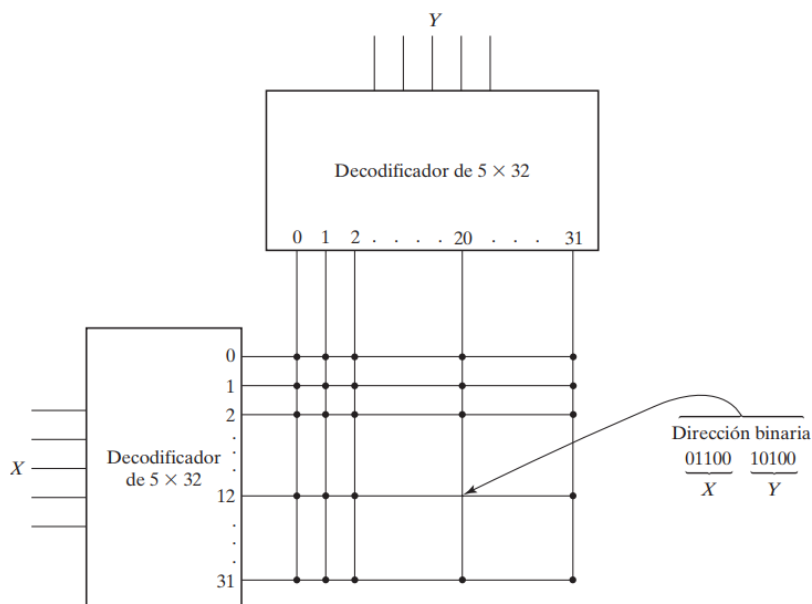


Figure 1.17: Decodificación bidimensional

1.4.4 Direcccionamiento

Una memoria de $1K \times 16$ tiene 10 bits en la dirección y 16 bits en cada palabra. La dirección de la memoria se especifica con 10 bits, y cada dirección selecciona una palabra de 16 bits. La primera dirección se especifica con 10 ceros, y la última, con 10 unos. Ello se debe a que 1023 en binario es 111111111. Se selecciona una palabra de memoria por su dirección binaria. Cuando se lee o escribe una palabra, la memoria opera sobre los 16 bits como una sola unidad.

Para generalizar, podríamos decir que una memoria de $m \times n$ tiene m palabras y n bits por palabra. La dirección de la memoria se especifica con k bits, y cada dirección selecciona una palabra de n bits. La primera dirección se especifica con k ceros, y la última, con k unos. Se selecciona una palabra de memoria por su dirección binaria. Cuando se lee o escribe una palabra, la memoria opera sobre los n bits como una sola unidad.

El bus de dirección es un conjunto de líneas que se usan para especificar la dirección de la palabra que se va a leer o escribir. El bus de datos es un conjunto de líneas que se usan para transferir información entre la memoria y el procesador. El bus de control es un conjunto de líneas que se usan para especificar la dirección de la transferencia de datos. La figura 1.18 muestra un diagrama de bloques de una memoria genérica.

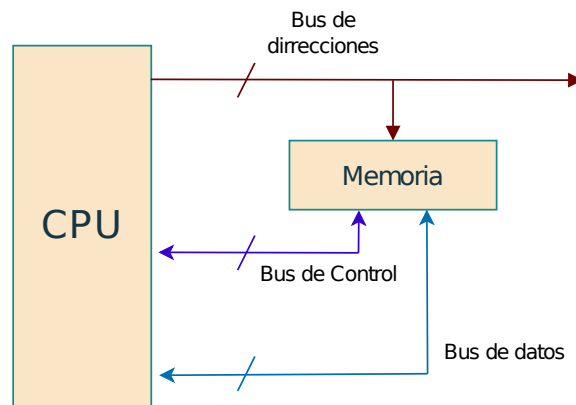


Figure 1.18: Diagrama de bloques de una memoria genérica

1.4.5 Construcción de memorias

Cuando surge la pregunta: ¿Cuántos chips de memoria se necesitan para almacenar $1K$ palabras de 16 bits cada una? La respuesta es: $1K \times 16 = 16K$ bits. Si cada chip de memoria tiene $1K$ bits, entonces se necesitan 16 chips.

Para generalizar, si se tienen m palabras de n bits cada una, se necesitan $m \times n$ bits. Si cada chip de memoria tiene k bits, entonces se necesitan $\frac{m \times n}{k}$ chips.

Tamaño de la memoria y dirección

La cantidad de bits de una memoria se calcula como $2^{\text{dirección}} \times \text{tamaño de palabra}$. Por ejemplo, si se tiene una memoria de 16 palabras de 8 bits cada una, la cantidad de bits de la memoria es $2^4 \times 8 = 128$ bits. Generalmente el gráfico de una memoria se representa de la siguiente manera:

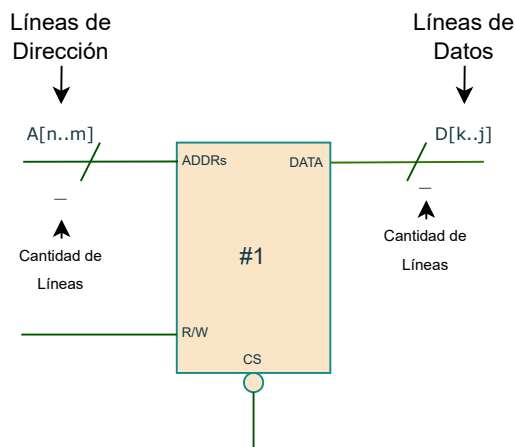
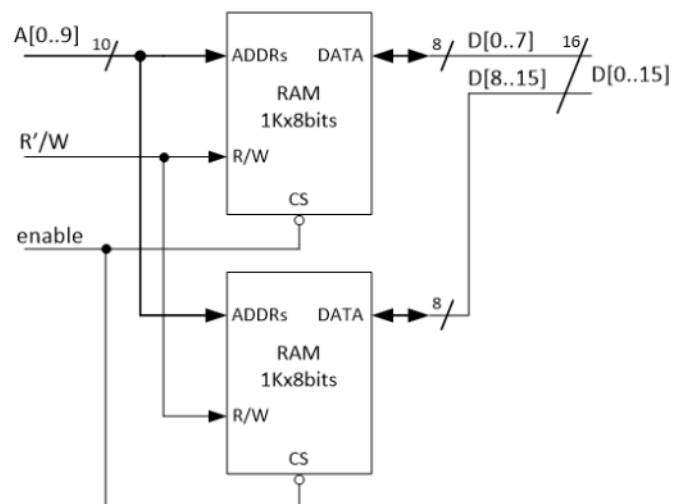


Figure 1.19: Diagrama de una memoria

Donde de la cantidad de líneas de dirección se obtiene la cantidad de palabras y de la cantidad de líneas de datos se obtiene el tamaño de la palabra.

1.4.6 Ampliación de palabra

Para ampliar la palabra de una memoria, se debe agregar un chip de memoria adicional. Por ejemplo, si se tiene una memoria de $1k$ palabras de 8 bits cada una y se desea ampliar la palabra a 16 bits, se debe agregar un chip de memoria adicional.



En este ejemplo, las líneas de datos de D0 a D7 corresponden a la primera palabra de 8 bits y las líneas de datos de D8 a D15 corresponden a la segunda palabra de 8 bits, se juntan para formar una palabra de 16 bits.

1.4.7 Ampliación de capacidad

Para ampliar la capacidad de una memoria, se deben agregar chips de memoria en serie y se deben conectar las líneas de dirección de manera que se seleccione el chip adecuado.

Ejemplo

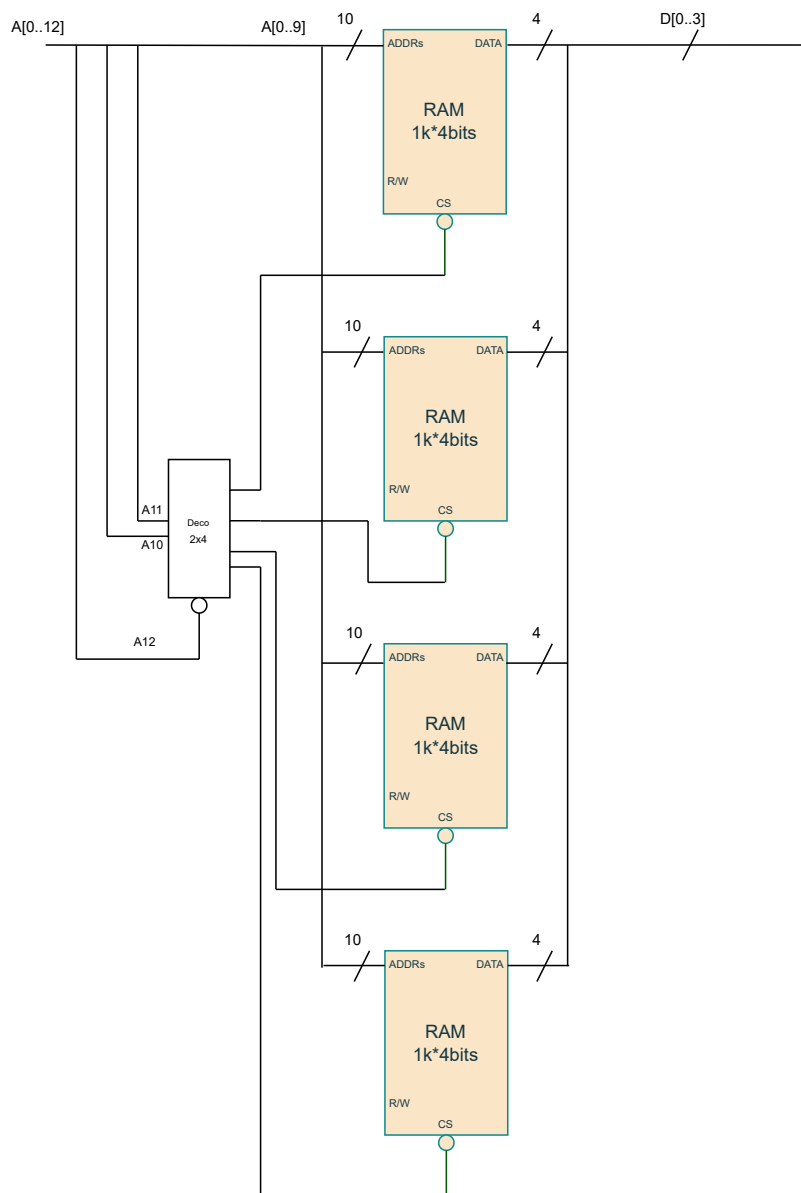
En el ejemplo se debe conformar un banco de 4K direcciones, cada una de 4 bits, y para hacerlo se cuenta con chips de 1K x 4 bits. En primer lugar dividimos la capacidad total de memoria necesaria por la capacidad de cada chip, a los efectos de obtener la cantidad de chips que debemos emplear. En este caso se necesitarán 4 chips.

Se pretende aumentar la cantidad de direcciones disponibles. Cada chip posee 10 líneas de dirección, pero el banco es de 4K, o sea que necesita 12 líneas. Las líneas A0, hasta A9 (10 bits) van a todos los chips simultáneamente. Las líneas restantes (A10 y A11) entran a un decodificador cuya función es la de ir habilitando uno a uno, a los diferentes chips de memorias del banco, con el objeto de evitar un conflicto en el Bus de Datos. De este modo, sólo un chip por vez estará activo, quedando el resto, en alta impedancia, con lo que cada chip pondrá, o recibirá, datos del bus de datos en forma individual.

Lo vamos a representar con una tabla, primero tendremos el chip, cada chip tiene las direcciones que tomará, y luego las líneas de dirección que se le asignan. Podemos notar que como se busca tener 4K direcciones, se necesitan 12 líneas de dirección.

Chip	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
#1	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	1	1	1	1	1	1	1	1	1
#2	0	0	1	0	0	0	0	0	0	0	0	0	0
	0	0	1	1	1	1	1	1	1	1	1	1	1
#3	0	1	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	1	1	1	1	1	1	1	1	1	1
#4	0	1	1	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	1	1

Se puede observar que el banco funcionará siempre que A12 sea 0, ya que en caso contrario se generarían espejos. Luego la parte verde son las líneas de dirección que comparten todos los chips, y la parte azul son las líneas de dirección que se le asignan a cada chip. Es decir las que usaremos para seleccionar el chip que queremos leer o escribir. Para ello podemos notar que usar un decodificador de 2x4 nos permitirá seleccionar el chip que queremos leer o escribir.



1.5 Circuitos Secuenciales

1.5.1 Flip Flops tipo D

Un flip-flop puede mantener un estado binario indefinidamente (en tanto se alimente al circuito), hasta que una señal de entrada le indique que debe cambiar de estado. El flip-flop D es un dispositivo secuencial que se utiliza para almacenar un bit de información. El flip-flop D se puede construir a partir de un flip-flop tipo JK, conectando las entradas J y K a un nivel lógico alto. La tabla de verdad de un flip-flop D es la siguiente:

D	CLK	Q
0	0	0
0	1	0
1	0	1
1	1	1

Esto quiere decir que si la entrada D es 0, la salida Q no cambia, sin importar el valor de CLK. Si la entrada D es 1, la salida Q toma el valor de D en el flanco de subida de CLK.

Registros

Un registro consiste en un grupo de flip-flops. Estos pueden contener, además, compuertas lógicas. Las compuertas lógicas se utilizan para efectuar la transición de información entre los flip-flops.

Cada flip-flop puede almacenar un bit de información. Un registro de n bits consiste en un grupo de n flip-flops capaces de almacenar n bits de información binaria.

1. Si el problema a resolver depende de alguna entrada externa que diferencie entre diferentes funcionamientos, generalmente se utilizan multiplexores para seleccionar la entrada adecuada.
2. Si se habla de un registro de desplazamiento, se puede utilizar un multiplexor para seleccionar la entrada adecuada.
3. Cuando se habla de paridad se debe tener en cuenta que el bit de paridad se suelen colocar compuertas XOR en la entrada de datos.

Contadores

Un contador es un circuito secuencial que genera una secuencia de números binarios. Los contadores se pueden clasificar en dos tipos: contador síncrono y contador asíncrono.

1. **Contador asíncrono:** En un contador asíncrono, la entrada de reloj de cada flip-flop es la salida del flip-flop anterior. La entrada de reloj del primer flip-flop es la señal de reloj del sistema. La salida de un contador asíncrono es la salida de los flip-flops.
2. **Contador síncrono:** En un contador síncrono, la entrada de reloj de todos los flip-flops es la misma señal de reloj del sistema. La entrada de reloj de todos los flip-flops es la misma señal de reloj del sistema. La salida de un contador síncrono es la salida de los flip-flops.

Circuitos secuenciales

Un circuito secuencial es un circuito que contiene elementos de almacenamiento, como flip-flops, y compuertas lógicas. Un circuito secuencial es de la forma:

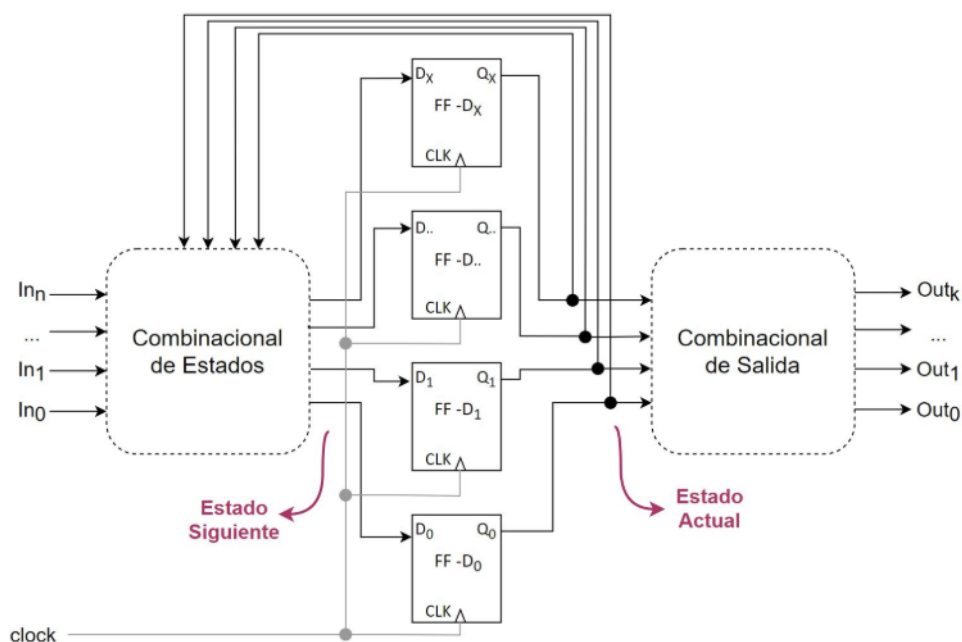


Figure 1.20: Diagrama de un circuito de memoria

- **Combinacional de estados:** Se basa en un circuito combinacional, ya sea mediante el uso de decodificadores, multiplexores, etc. Son encargados de generar las señales de estados que se van a almacenar en los flip-flops. **Toman como entrada los estados actuales y las entradas del sistema.**
- **Memoria de estados:** Se basa en un conjunto de flip-flops que almacenan los estados del sistema. **Se encargan de almacenar los estados generados por el circuito combinacional.**
- **Combinacional de salidas:** Se basa en un circuito combinacional que genera las salidas del sistema. **Toman como entrada los estados almacenados en la memoria de estados y generan las salidas del sistema.**

Las salidas de los flip-flops se conectan a las entradas de los flip-flops, de esta manera se logra que el circuito se comporte de manera secuencial.