

# Tema 1

## INTRODUCCIÓN A LA TECNOLOGÍA DE COMPUTADORES

### 1.1. ORGANIZACIÓN ESTRUCTURAL DE UN COMPUTADOR

Desde el punto de vista estructural, el computador se considera dividido en varios niveles organizados jerárquicamente, de modo que para poder construir y hacer funcionar los elementos de un nivel determinado, es necesario utilizar los elementos construidos en el nivel inmediatamente inferior.

La figura 1.1 muestra la organización del computador según estos niveles estructurales, y en ella pueden apreciarse ocho niveles de estudio divididos en dos grandes zonas: una zona inferior donde los niveles son de naturaleza “hardware”, y una zona superior donde los niveles son de naturaleza “software”. Cada nivel se diferencia de los demás porque se rige por unas leyes de comportamiento, unas reglas y unos métodos de diseño que le son propios, además de utilizar un lenguaje específico para describir su estructura y funcionamiento.

En la figura 1.1, los niveles se representan por bloques cuyas entradas y salidas son los elementos constituyentes del computador en sus diversos grados de elaboración y niveles de integración. Las flechas son bidireccionales porque, dependiendo del punto de vista, la entrada a un bloque puede transformarse en salida y viceversa. En este sentido, la flecha unidireccional hacia arriba, que está situada en la parte izquierda de dicha figura, indica la dirección seguida en el proceso de diseño de un computador, ya que un nivel determinado utiliza los elementos inferiores –situados debajo del bloque– para diseñar y construir los elementos del nivel inmediatamente superior –situados encima.

Sin embargo, la flecha unidireccional hacia abajo, que está situada en la parte derecha de la figura 1.1, indica la dirección seguida a la hora de realizar el análisis funcional

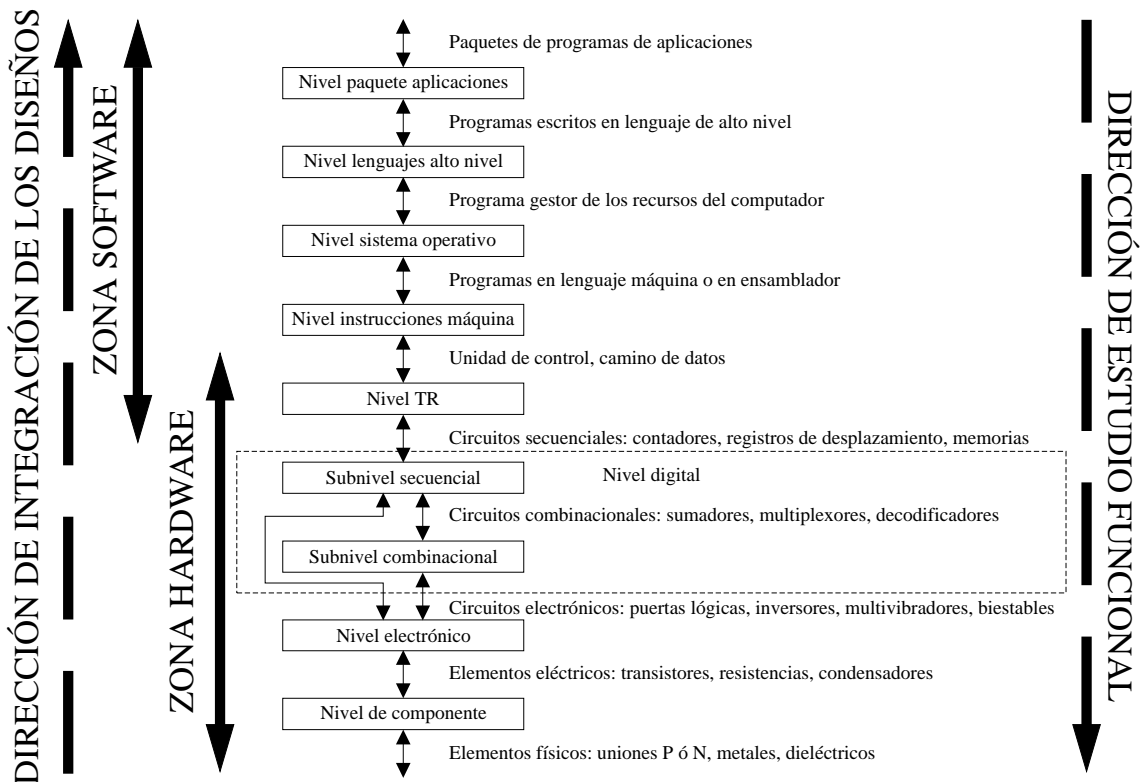


Figura 1.1: Organización estructural de un computador.

de la máquina, en el que se comienza analizando los elementos más complejos para ir subdividiendo sucesivamente el computador en sus elementos más básicos. Esta segunda dirección es la que también deberá seguir un usuario que desee profundizar en el manejo de la máquina: comenzará por el nivel más alto, que es el más complejo estructuralmente, pero cuyo uso es el más fácil de aprender y, a medida que vaya necesitando funciones y prestaciones más específicas, irá bajando a niveles estructuralmente más simples, pero cuya utilización correcta precisa de mayores y más detallados conocimientos del computador.

A continuación se describen los ocho niveles, empezando por los más básicos y en el sentido de ir aumentando paulatinamente los grados de integración y complejidad.

El primer nivel está constituido por los **componentes**. Las leyes aplicadas son las de la electrónica física. Los componentes son semiconductores de tipo  $n$  y  $p$ , metales, etc. Los sistemas construidos son diodos, transistores, resistencias, condensadores, etc. Este nivel es estudiado por la **Física del Estado Sólido**.

El segundo es el nivel de **circuito electrónico**. El comportamiento del circuito se describe en términos de corrientes, tensiones, flujos, etc. Se construyen sistemas como las puertas lógicas, los biestables, los osciladores, etc., a partir de los componentes. Las leyes utilizadas en este nivel son las de la electricidad, que son de naturaleza continua, pero los circuitos o sistemas construidos son generalmente de tipo digital. Este nivel es estudiado

por la **Electrotecnia o la Electrónica Analógica**.

El tercer nivel es el nivel de **circuito digital**. Las leyes que rigen su comportamiento son las del álgebra de Boole. Como puede observarse en la figura 1.1, este nivel puede dividirse en dos subniveles, el de **circuitos combinacionales** y el de **circuitos secuenciales**. En el subnivel combinacional se utilizan los circuitos electrónicos del nivel anterior como por ejemplo las puertas lógicas digitales, para obtener circuitos o sistemas más complejos de carácter combinacional, como son los multiplexores, decodificadores, sumadores, etc. Los circuitos combinacionales son circuitos sin memoria, cuya salida en todo momento es función exclusiva de sus entradas. En el subnivel secuencial se construyen circuitos secuenciales complejos a partir de circuitos de memoria tipo biestable, junto con circuitos combinacionales (por eso dicho subnivel se comunica en la figura 1.1 con el subnivel combinacional y con el nivel electrónico). Los circuitos secuenciales se caracterizan porque su salida es función tanto de la entrada que tienen en ese momento como de la historia anterior (memoria o estado) del circuito, lo cual implica que la variable tiempo desempeña un valor fundamental en su comportamiento. Este nivel es estudiado por la **Electrónica Digital**.

El siguiente nivel de estudio es el de **TR** o de **Transferencia entre Registros**. En este caso se estudia el computador centrándose en el flujo de información que se envía de un registro a otro, pasando, en su caso, por el correspondiente circuito combinacional que lo encamina o lo transforma. Las transferencias se representarán mediante flechas, de la forma siguiente:

$$\begin{aligned} A &\leftarrow B \\ A &\leftarrow B + A \\ A &\leftarrow B \text{ AND } A \end{aligned}$$

donde  $A$ ,  $B$  y  $C$  representan registros o posiciones de memoria, y las operaciones  $+$  y  $\text{AND}$  se realizan mediante el correspondiente circuito combinacional.

Los elementos constructivos de este nivel son los buses, los registros, los bloques combinacionales, las memorias, etc. Los sistemas formados serán bloques constructivos del computador o éste mismo. Este es el primer nivel que se estudia propiamente desde el punto de vista de la **Arquitectura de Computadores**.

El funcionamiento de la unidad de control puede definirse mediante hardware (por cableado electrónico) o por software (microprogramación). Ello hace que en la figura 1.1 aparezca una superposición entre las dos grandes zonas de software y hardware en este nivel, que además actúa como frontera entre lo que es circuitería electrónica y lo que es programación.

El siguiente nivel es el de la **UCP** o de **instrucciones de máquina**, y constituye el primer nivel específicamente de programación. En este nivel se construyen programas en lenguaje máquina, que es el lenguaje que es capaz de entender la UCP. También se incluye en este nivel la programación en ensamblador, al ser simplemente una forma simbólica de utilizar el lenguaje máquina. La programación de la UCP en este nivel recibe el nombre

genérico de **programación en bajo nivel**. Las técnicas utilizadas en este nivel también son estudiadas por la **Arquitectura de Computadores**.

El sexto nivel es el del **sistema operativo**. Los sistemas operativos son un conjunto de programas que ayudan al usuario en la explotación del computador, siendo, por tanto, una capa software con la que se rodea al hardware para facilitar su utilización, y realiza funciones como las de asignar recursos a los distintos procesos que se desean ejecutar, gestionar la memoria auxiliar del computador, estar al cargo de la protección y contabilidad del computador, etc. La **Arquitectura de Computadores** se encarga de estudiar este nivel, aunque hay aspectos que también son estudiados desde el punto de vista de los **Lenguajes de Programación**.

El séptimo nivel es el de los **programas en lenguajes de alto nivel** como puedan ser el FORTRAN, Pascal, COBOL, C, Ada, etc. Los programas escritos en estos lenguajes deberán pasar por un proceso intermedio de traducción, denominado **compilación**, antes de ejecutarse. El objetivo de la compilación es traducir los programas escritos en los lenguajes de alto nivel — que resultan cómodos para que el programador trabaje — al lenguaje de bajo nivel — que es el que entiende la UCP de la máquina. El resultado de dicha traducción recibe el nombre de **código objeto** y, según la aplicación, puede en ciertos casos traspasar el nivel de sistema operativo e interactuar directamente con el nivel de UCP.

Finalmente nos queda el último nivel de **aplicaciones**. Este nivel está formado por paquetes de programas de aplicación que sirven para resolver problemas en campos específicos de la ciencia o gestión. Dado el creciente coste del “software” frente al coste del “hardware”, cada día cobran mayor importancia los paquetes estándar, como prueba el que una gran parte del mercado de la informática se encuentra en este campo.

Según esta visión estructural del computador, en la asignatura *Tecnología de Computadores* estudiaremos básicamente el nivel 3, centrándonos en la descripción funcional de los circuitos, y en el estudio de técnicas de diseño que nos conduzcan a entender los bloques básicos de un computador y cómo se implementan.

## 1.2. SISTEMAS DIGITALES

Se puede definir un *sistema digital* como cualquier sistema de transmisión o procesamiento de información en el cual la información se encuentre representada por medio de cantidades físicas (señales) que se hayan tan restringidas que sólo pueden asumir valores discretos. Los sistemas digitales más sencillos tratan con señales binarias, que sólo pueden tomar dos valores, que representaremos de ahora en adelante con los símbolos “0” y “1”, por lo que se les llama sistemas *binarios*. Los sistemas digitales también se denominan sistemas de conmutación dado que el origen histórico de la teoría de sistemas digitales se encuentra en la necesidad de analizar y diseñar las redes de relés de los primeros sistemas telefónicos.

En contraposición a los sistemas digitales están los sistemas *analógicos* en los cuales las señales tanto de entrada como de salida no poseen ningún tipo de restricción y pueden asumir todo un continuo de valores (es decir, infinitos). Algunos de estos sistemas se estudian en la asignatura de *Tecnología Electrónica*.

La principal ventaja de los sistemas digitales respecto a los analógicos es que son más fáciles de diseñar, de implementar y de depurar, ya que las técnicas utilizadas en cada una de esas fases están bien establecidas. Por lo tanto, es más sencillo y flexible realizar un diseño digital que uno analógico. Las operaciones digitales también son mucho más precisas y la transmisión de señales dentro del circuito y entre circuitos es más fiable porque utilizan un conjunto discreto de valores, fácilmente discernibles entre sí, lo que reduce la probabilidad de cometer errores de interpretación.

El mejor argumento a favor de la mayor flexibilidad de los sistemas digitales se encuentra en los actuales ordenadores o computadoras digitales, basados íntegramente en diseños y circuitos digitales. En esta asignatura estudiaremos cuáles son y cómo se diseñan los componentes y bloques básicos de un ordenador de propósito general. En cursos posteriores se ahondará en el conocimiento de la constitución íntima de los ordenadores hasta llegar a los actuales sistemas avanzados que incorporan sofisticadas técnicas para mejorar la eficiencia y velocidad de los computadores.

Los sistemas digitales se definen a través de *funciones digitales* que son, ni más ni menos, que aplicaciones entre dos conjuntos discretos: el conjunto de todas las entradas posibles  $X$  y el conjunto de todas las salidas posibles  $Y$ . Es decir,

$$F : X \mapsto Y$$

Sin embargo, para nosotros los sistemas que tienen mayor interés, por ser los que se pueden implementar electrónicamente, son los sistemas binarios. Un *sistema binario* es aquel en el que tanto las señales de entrada como de salida así como las señales internas sólo pueden ser “0” ó “1”. Por lo tanto, una *función digital binaria*, o simplemente *función binaria*, de  $n$  variables binarias,  $F(x_{n-1}, \dots, x_0)$ , se define como la aplicación del producto cartesiano  $K^n$  en el conjunto  $K$ , donde  $K = \{0, 1\}$ . Al contar el mencionado producto cartesiano con  $2^n$  combinaciones, existirán un total de  $2^{2^n}$  funciones binarias distintas de  $n$  variables.

Para poder implementar una función digital como una función binaria es preciso utilizar señales con sólo dos valores “0” ó “1”. Para ello es necesario hacer que las señales pasen de tomar valores de un conjunto arbitrario (pero finito) a tomar sólo 2 valores. La única forma de conseguirlo es agrupar un conjunto de señales binarias (*bits*) y que, juntas, codifiquen todos o parte de los elementos del conjunto discreto de entrada y de salida. A ese conjunto de señales o bits le llamaremos una *variable numérica* o simplemente *variable*.

Para entender mejor este concepto supongamos que tenemos un sistema digital cuyas entradas son las 5 vocales del alfabeto. Está claro que ese conjunto es discreto y que con

una sólo variable binaria no se puede codificar (con un bit sólo podemos codificar dos valores). Para poder representar los 5 elementos es necesario utilizar 3 señales binarias (bits) y agruparlas formando una variable binaria, por ejemplo  $A$ . En este caso,  $A$  estará formada por 3 bits, es decir,  $A = A_2A_1A_0$ , con lo cual es capaz de representar hasta 8 elementos diferentes ( $2^3$ ). En el caso general una variable de  $n$  bits  $A = A_{n-1}A_{n-2}\dots A_1A_0$  puede codificar hasta  $2^n$  posibilidades diferentes. La codificación utilizada, es decir, que representa cada combinación de bits, es totalmente arbitraria y no influirá en el resultado final. En el caso de las vocales podemos escoger, por ejemplo, la siguiente codificación:

$A_2$	$A_1$	$A_0$	<i>Vocales</i>
0	0	0	(no usada)
0	0	1	<i>a</i>
0	1	0	<i>e</i>
0	1	1	(no usada)
1	0	0	<i>u</i>
1	0	1	<i>i</i>
1	1	0	<i>o</i>
1	1	1	(no usada)

Una variable no tiene por qué codificar todos los elementos del conjunto de entrada o de salida, sino que puede limitarse a codificar un subconjunto de ellos. En ese caso es necesario que existan una o más variables que codifiquen los elementos restantes de cada conjunto.

Por convenio, para codificar números naturales (números enteros positivos), se suele utilizar la *representación posicional* en donde el bit  $i$ -ésimo tiene asociado un peso  $2^i$ , contando los bits de derecha a izquierda y empezando por el cero. Por lo tanto, el bit más significativo (el que tiene asociado un mayor peso) es el de la izquierda. Con esta representación, la variable numérica  $A = A_{n-1}A_{n-2}\dots A_1A_0$  representará el número natural

$$A = A_{n-1}2^{n-1} + A_{n-2}2^{n-2} + \dots + A_12^1 + A_02^0 = \sum_{i=0}^{n-1} A_i2^i$$

Una función digital se puede representar de muchas formas diferentes. La representación más usual son las Tablas de Verdad que consisten en una tabla en donde se indica la salida para todas y cada una de las combinaciones de los bits de entrada. En el tema siguiente veremos otras representaciones de las funciones binarias: los mapas de Karnaugh y las expresiones booleanas.

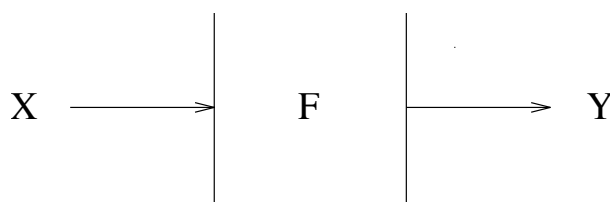


Figura 1.2: Ejemplo de un sistema combinacional.

### 1.3. SISTEMAS SECUENCIALES Y COMBINACIONALES

Un sistema digital *combinacional* se define, en general, como aquel sistema en el que las salidas son solamente función de las entradas actuales, es decir, dependen únicamente de las combinaciones de las entradas, de ahí su nombre. Estos sistemas se pueden representar a través de una función del tipo:

$$F : X \mapsto Y$$

donde  $X$  es el conjunto (discreto) de entradas e  $Y$  el conjunto (también discreto) de salidas. Se suele representar como en la figura 1.2.

Un ejemplo sencillo de sistema combinacional es un portaminas. En este sistema sólo son posibles dos acciones o entradas: pulsar o no pulsar, y sólo son posibles dos salidas: salir la mina o no hacer nada. El sistema es combinacional porque, siempre que se aplique una entrada, la respuesta del sistema sólo depende de esa entrada, es decir, depende de las combinaciones de las entradas actuales. En nuestro ejemplo del portaminas, siempre que se pulsa sale la mina mientras que si no se pulsa no pasa nada.

Existe un tipo más general de sistemas, llamados *secuenciales*, que además de entradas y salidas poseen unas variables adicionales llamadas variables internas o variables de estado (o bien simplemente estado). El estado hace que la salida del sistema dependa de las entradas anteriores además de la entrada actual. Físicamente, el estado va a representar una propiedad del sistema que, aunque no es observable directamente desde el exterior, va a determinar la salida que presente dicho sistema ante una entrada determinada. Es decir, para una misma entrada van a ser posibles distintas salidas, dependiendo del estado actual del sistema.

Un sistema secuencial se va a representar, tal y como se ve en la figura 1.3, por dos funciones combinacionales ( $F_1$  y  $F_2$ ) y un nuevo tipo de función o módulo ( $\Delta$ ) que se encarga de almacenar el valor de los estados. Las funciones  $F_1$  y  $F_2$  son aplicaciones entre el conjunto de todas las entradas ( $X$ ), todas las salidas ( $Y$ ) y todos los estados internos ( $S$ ). La primera función ( $F_1$ ) calcula el estado siguiente del sistema según el estado y la entrada actuales.

$$F_1 : S \times X \mapsto S$$

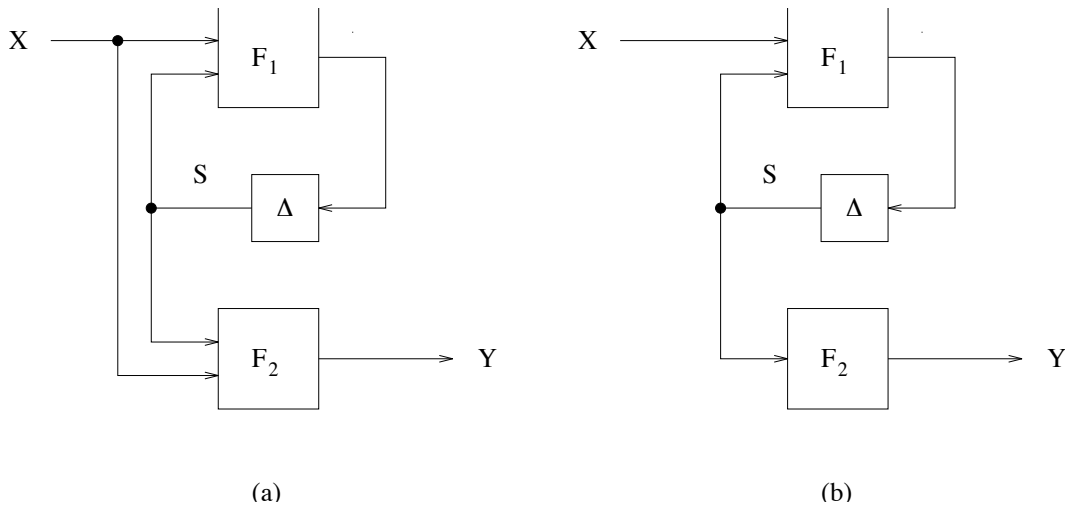


Figura 1.3: Definición de un autómata Mealy (a) y Moore (b).

Por otra parte,  $F_2$  calcula cual es la salida en función del estado y de la entrada también actuales. Es decir,

$$F_2 : S \times X \mapsto Y$$

Esta es la definición de un autómata Mealy. Existe otra forma equivalente de definir un sistema secuencial que es la representación mediante un autómata Moore. En este caso, la función  $F_2$  sólo necesita conocer el estado actual para calcular la salida, es decir,

$$F_2 : S \mapsto Y$$

Como se ha dicho, ambas formulaciones son equivalentes, y un autómata Mealy se puede transformar en un autómata Moore y viceversa.

Un ejemplo sencillo de sistema secuencial sería un bolígrafo en donde al pulsar un botón sale o entra la punta. En este caso también existen dos entradas diferentes: pulsar o no pulsar, y tres salidas: salir punta, entrar punta o no hacer nada. Siempre que no se pulse no pasará nada, pero la entrada pulsar produce salidas diferentes según el estado del bolígrafo: si la punta está fuera entonces entra, mientras que si la punta está dentro entonces sale. Por lo tanto, la misma acción o entrada (pulsar) produce dos salidas diferentes en función del estado interno del sistema (punta dentro o punta fuera).

La salida se puede decir que depende de la entrada y el estado actuales, o que depende de toda la historia previa (secuencia de entradas y el estado inicial en el que partió el sistema) que está almacenada en el estado interno del sistema. El nombre de sistema secuencial se debe a esa dependencia de la salida con la secuencia de entradas (el tipo de entradas y el orden en el que se aplican). Se puede deducir rápidamente, que un sistema combinacional se puede representar como un sistema secuencial con un único estado interno. Este tipo de sistemas que contienen estados internos son estudiados por una teoría más amplia denominada *Teoría de Autómatas*.



Cuadro 1.1: Todas las funciones binarias de 1 variable.

	$f_0$	$f_1$	$f_2$	$f_3$
$X$	0	$X$	$\overline{X}$	1
0	0	0	1	1
1	0	1	0	1

Cuadro 1.2: Todas las funciones binarias de 2 variables.

$X_1$	$X_0$	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$	$f_8$	$f_9$	$f_{10}$	$f_{11}$	$f_{12}$	$f_{13}$	$f_{14}$	$f_{15}$
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

## 1.4. CIRCUITOS LÓGICOS Y SÍMBOLOS

Tal y como vimos en la definición de función binaria, para una función de  $n$  variables existen un total de  $2^{2^n}$  posibles definiciones. Por lo tanto, para el caso de funciones de 1 variable existe un total de  $2^{2^1} = 2^2 = 4$  posibilidades, las cuales se pueden ver en la tabla 1.1.

Todas esas funciones unarias (de una sólo entrada) tienen un nombre. La primera de ellas,  $f_0$ , es la función cero, porque su salida siempre es 0. La segunda,  $f_1$ , es la función identidad, ya que la salida siempre es igual a la entrada. Físicamente equivale a un cable o línea de comunicación puesto que transmite la entrada sin ningún tipo de modificación. La tercera función,  $f_2$ , es la función NOT porque la salida es siempre la complementaria (el valor opuesto) a la entrada. La última es la función unidad, ya que siempre es 1 independientemente de la entrada.

En el caso de funciones con dos entradas existen un total de  $2^{2^2} = 2^4 = 16$  posibilidades, las cuales se pueden ver en la tabla 1.2.

No todas las funciones de dos entradas tienen nombre ni están implementadas a nivel de puertas lógicas. A continuación mencionaremos las que se suelen llamar puertas lógicas básicas. En la figura 1.4 se pueden ver los símbolos de algunas de estas puertas y sus tablas de verdad.

- La función 0 o cero ( $f_0$ ) siempre produce la salida 0, independientemente del valor que estén en sus entradas. Físicamente se implementa conectando la salida con el valor bajo de tensión (GND o low).
- La función 1 o unidad ( $f_{15}$ ) es la negada de la anterior, siempre está a 1 independientemente del valor de sus entradas. Físicamente se implementa conectando la salida con el valor alto de tensión ( $V_{CC}$  o high).

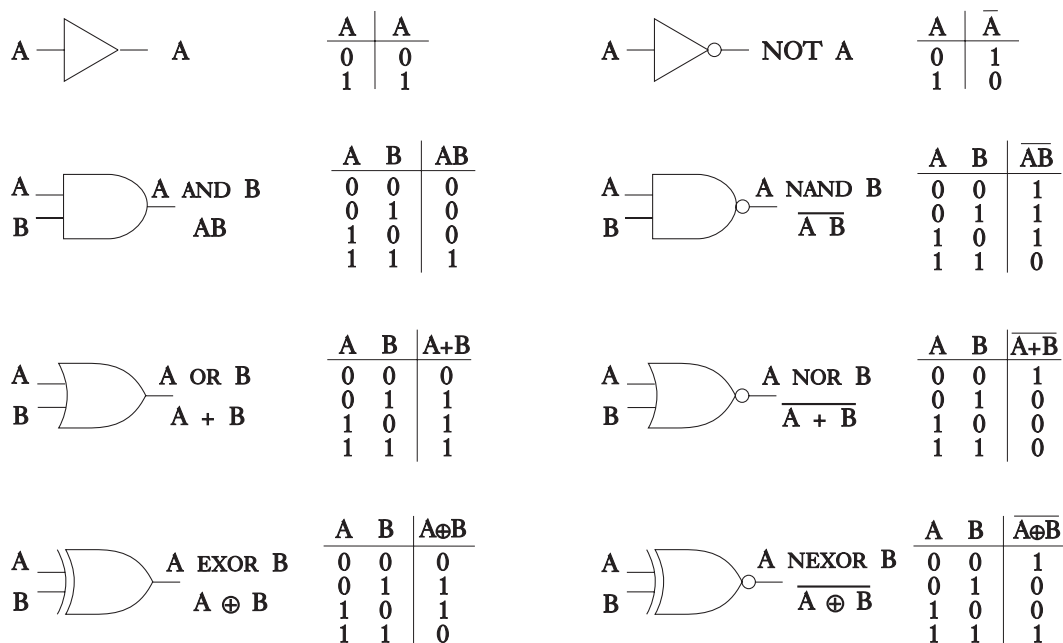


Figura 1.4: Símbolos de las puertas lógicas básicas.

- La función o puerta AND ( $AB$ ),  $f_8$ , ya es conocida. Es aquella cuya salida es 1 únicamente cuando todas las entradas son 1. Esta definición se puede generalizar para  $n$  variables.
- La puerta NAND ( $\overline{AB}$ ),  $f_7$ , equivale a una puerta AND seguida de una NOT (de ahí su nombre NOT AND). La salida es 0 únicamente cuando todas las entradas son 1.
- La función o puerta OR ( $A+B$ ),  $f_{14}$ , también es conocida. Es aquella cuya salida es 1 cuando alguna de las entradas es 1. También se puede generalizar para  $n$  variables.
- La puerta NOR ( $\overline{A+B}$ ),  $f_1$ , equivale a una puerta OR seguida de una NOT (NOT OR). La salida es 0 cuando alguna de las entradas es 1.
- La función o puerta EXOR ( $A \oplus B$ ),  $f_6$ , es aquella cuya salida es 1 cuando a la entrada hay un número impar de 1's, y es 0 en el caso contrario (un número par de 1's a la entrada). Si generalizamos a  $n$  variables tendremos una EXOR de  $n$  entradas. Con esta puerta se construyen los detectores de paridad (si el número de 1's de una variable es par o impar).
- La puerta NEXOR ( $\overline{A \oplus B}$ ),  $f_9$ , equivale a una puerta EXOR seguida de una NOT (NOT EXOR). Por lo tanto la salida es 0 cuando a la entrada hay un número impar de 1's, y es 1 en caso contrario. También se puede generalizar para  $n$  variables. La puerta NEXOR de 2 entradas también se llama IGUALDAD o EQUIVALENCIA ya que la salida es 1 cuando ambas entradas toman el mismo valor.

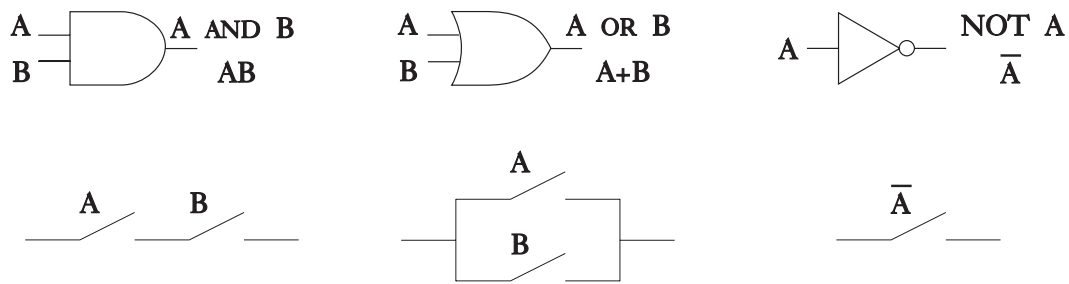


Figura 1.5: Una interpretación de las puertas lógicas básicas usando interruptores.

En la figura 1.5 se muestran los circuitos o puertas lógicas básicos que se utilizan en todos los sistemas digitales. Tal y como veremos en el tema siguiente, el Álgebra de Boole nos asegura que **cualquier** función digital binaria se puede implementar utilizando únicamente esas 3 puertas lógicas: AND, OR y NOT. La razón es muy sencilla: el Álgebra de Boole está definida según esas 3 operaciones, por lo que cualquier expresión booleana contiene, únicamente, esas 3 operaciones.

En la figura 1.5, además de los símbolos de las puertas lógicas básicas, se incluyen el equivalente de cada una en la teoría de la conmutación a base de interruptores. Un interruptor es un dispositivo con una entrada, una salida y una variable de control. Cuando la variable de control se activa (1) entonces la entrada y la salida están físicamente conectadas, mientras que si la variable de control no se activa (0) entonces no existe conexión entre la entrada y la salida del interruptor.

Como se puede ver en la figura 1.5, el comportamiento de una puerta AND de dos entradas es equivalente al funcionamiento de dos interruptores puestos uno a continuación del otro (en serie). Sólo existe conexión cuando las señales de control de ambos interruptores (A y B) están cerrados, es decir, las variables de control están activadas (son 1). En el caso de que una o las dos variables estén inactivas (0), entonces uno o los dos interruptores estará abierto y no existirá conexión. Esto se corresponde con la definición de puerta AND si asociamos la salida de la puerta (1 y 0) con la existencia o no de conexión.

Con la puerta OR de 2 entradas se puede seguir un esquema parecido, pero su comportamiento es equivalente al funcionamiento de dos interruptores puestos en paralelo (ver figura). En este caso, existe salida (conexión) si al menos una de las entradas está activa (un interruptor esté cerrado).

Por último, la puerta NOT equivale a un interruptor que funciona de forma contraria a la usual: se cierra cuando la variable de control no se activa (1) y se abre (no existe conexión) cuando la variable de control se activa (0).

En la teoría lógica, AND implica que han de cumplirse a la vez las dos premisas para que la proposición sea verdadera. Con OR es suficiente que se cumpla una sólo de las premisas para que la proposición sea verdadera. Por último, con la operación NOT la proposición es verdadera si la premisa es falsa, y viceversa. En la teoría de conjuntos, la operación AND es la intersección de conjuntos, la operación OR es la unión y NOT es el conjunto complementario.

Cuadro 1.3: Tablas de verdad de las puertas lógicas AND, OR y NOT.

$A$	$B$	$AB$	$A$	$B$	$A + B$	$A$	$\bar{A}$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Matemáticamente, a una variable activa se le asocia el valor lógico “1” y a una variable no activa el valor lógico “0”. Por otra parte, se denota a la función AND por  $(\cdot)$ , a la función OR por  $(+)$  y a la función NOT por  $(\bar{\quad})$  sobre la variable o bien con  $(')$ . Estos símbolos no implican que se realice una suma o multiplicación pues las variables no son numéricas. Cuando dos variables se ponen juntas se entiende una operación AND. Otras alternativas son el símbolo  $(\wedge)$  para AND y  $(\vee)$  para OR. Las Tablas de Verdad (tablas de valores) de cada una de estas funciones básicas se pueden ver en la tabla 1.3.

Físicamente, en electrónica digital se dispone de dos valores o niveles de tensión diferenciados: un valor alto H (high, 5 voltios,  $V_{CC}$  o alimentación) y un valor bajo L (low, 0 voltios, GND o tierra). Normalmente se asocia al nivel H el “1” lógico y al nivel L el “0” lógico y la lógica se dice positiva. La otra alternativa es asociar a H el “0” y a L el “1” y la lógica será negativa.

## 1.5. CIRCUITOS INTEGRADOS

La mayoría de los elementos que veremos se encuentran disponibles en el mercado en forma de circuitos integrados (CI). En concreto, se implementan en CI todos aquellos circuitos que, debido a su uso generalizado, son rentables comercialmente. La principal ventaja de los circuitos integrados, frente a una implementación tradicional basada en transistores discretos, es que facilitan y reducen el tiempo de diseño y de implementación y, por lo tanto, de los costes finales. Otras ventajas de los CI son:

- Alto grado de integración, llegándose a implementar millones de componentes en un chip de reducidas dimensiones.
- Reducción de coste, debido al alto grado de automatización existente en la fabricación de los CI y la producción en masa.
- La fiabilidad. Un CI posee mayor fiabilidad en cuanto a funcionamiento y duración que los transistores discretos.
- La velocidad de funcionamiento es mayor ya que el paso de la corriente depende de las longitudes de las interconexiones, muy pequeñas dentro del CI.
- Reducción de los posibles errores de montaje e interconexión de componentes.

- Reducción del tiempo de localización de averías, ya que sólo hay que buscarlas en las conexiones entre los CI y en los CI defectuosos. En este último caso bastaría con cambiar el CI por otro en buenas condiciones.

Obviamente también existen limitaciones. Las más importantes son que debido a sus reducidas dimensiones la potencia máxima que puede disipar un CI es pequeña, sólo se pueden implementar rangos reducidos de resistencias y condensadores, y es muy difícil implementar bobinas e inductancias en los CI.

Últimamente, y debido a todas las ventajas antes mencionadas, están cambiando las técnicas para el diseño de circuitos digitales. De tal forma que actualmente es muy extraño hacer diseños digitales basados en elementos discretos, es decir, transistores y resistencias. Esto es cierto salvo en algunas aplicaciones, como por ejemplo las que manejan grandes potencias. La tendencia actual es utilizar pocos CI pero muy complejos (con muchos transistores), entre los que están las PLA y las PAL, circuitos que veremos en el tema dedicado a la lógica programable.

Los CI se pueden implementar con diferentes técnicas o *tecnologías*, según sean los métodos de fabricación de los componentes. Las tecnologías más conocidas y usadas son las TTL y CMOS, aunque existen otras, tales como la ECL, DTL, Bipolar, NMOS, PMOS, etc. Cada una de ellas posee sus propias ventajas e inconvenientes que las hacen más adecuadas a ciertas aplicaciones. Las líneas de investigación actual apuntan a la fabricación de los diferentes componentes dentro de un mismo CI con distintas tecnologías.

En la *nomenclatura* de los circuitos TTL cada circuito integrado posee un código basado en un conjunto de números y letras de la forma  $74yyxxx$  (o  $54yyxxx$ ). Los dos primeros números se refieren a la tecnología y características del encapsulamiento, lo que determina el rango de temperaturas en donde pueden funcionar, máxima potencia de disipación, etc. El código  $74$  es para circuitos comerciales y el  $54$  es para los militares (más caros). El código  $yy$  es un conjunto de 1 ó 2 letras que indica el tipo de transistor, velocidad, consumo, etc. Así, por ejemplo,  $L$  indica bajo consumo,  $H$  es para alta velocidad y  $LS$  es para transistores Schottky ( $S$ ) de bajo consumo ( $L$ ). Por último, los números  $xxx$  especifican el tipo de circuito que está implementado en el CI. Así,  $32$  es el código de un CI con 4 puertas OR de 2 entradas cada una,  $00$  son 4 puertas NAND de 2 entradas,  $86$  son 4 puertas EXOR de 2 entradas y  $04$  son 6 puertas NOT. La nomenclatura de los CMOS es  $40xxx$ , donde  $40$  indica el tipo de tecnología (CMOS) y  $xxx$  indica el tipo de circuito implementado. Por ejemplo,  $02$  es el código de un chip CMOS que implementa 2 puertas NOR de cuatro entradas cada una,  $01$  es un chip con 4 puertas NOR de 2 entradas cada una.

Además de la tecnología, los CI se diferencian por el número de transistores o puertas lógicas equivalentes que implementan. Así se establecen 4 niveles de *integración* diferentes.

- Pequeña escala de integración, SSI (“Small Scale Integration”), cuando el CI posee un máximo de 12 puertas lógicas.

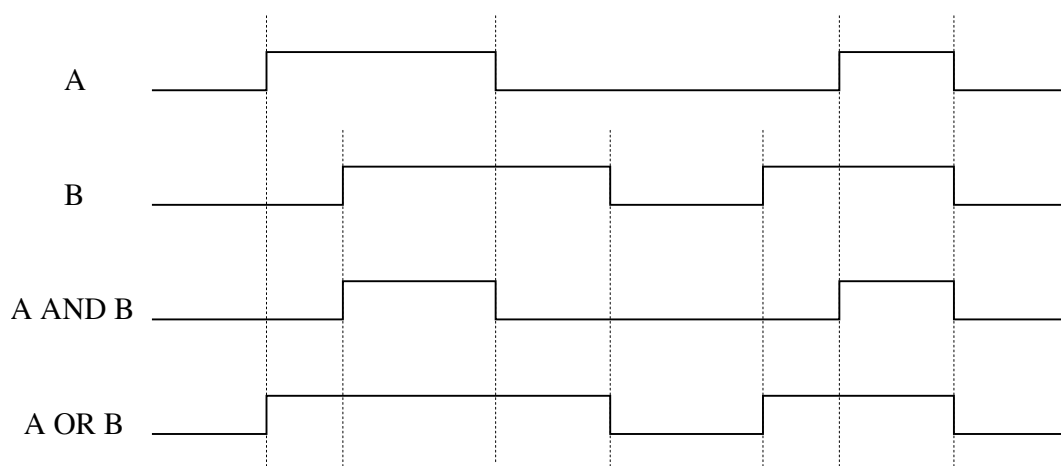


Figura 1.6: Funcionamiento de las puertas lógicas AND y OR.

- Media escala de integración, MSI (“Medium Scale Integration”), entre 12 y 100 puertas.
- Alta escala de integración, LSI (“Large Scale Integration”), entre 100 y aproximadamente 10.000 puertas.
- Muy alta escala de integración, VLSI (“Very Large Scale Integration”), cuando el CI posee del orden de millones de transistores.

En las prácticas de laboratorio sólo utilizaremos circuitos pequeños (SSI y MSI) y fabricados con las tecnologías TTL y CMOS. En la figura 1.9 se pueden ver algunos ejemplos de CI comerciales.

Físicamente, los circuitos integrados trabajan con valores de tensión. En los CI digitales binarios, los únicos valores de tensión son baja tensión (L o GND) y tensión alta (H o  $V_{CC}$ ). En la figura 1.6 se puede ver un ejemplo de como funcionan dos puertas lógicas (en este caso una AND y una OR). Como se puede ver, la tensión de las entradas cambia con el tiempo, lo cual produce que la salida de cada puerta también varíe. La salida de cada puerta sólo depende del valor de sus entradas en cada instante. La salida se calcula utilizando las tablas de verdad de cada una de las funciones (ver figura 1.4 o tabla 1.3).

En la práctica, como los CI están contruidos con elementos físicos (transistores), la salida no se genera instantáneamente, sino que presenta un pequeño retraso (que depende de la tecnología de fabricación aunque no suele superar unos pocos nanosegundos). Por ejemplo, la respuesta de un inversor ante una tensión de entrada podría ser la que se ve en la figura 1.7.

De todos modos, siempre se considerará, salvo que se indique lo contrario, que todas las puertas lógicas y CI son ideales y no poseen ningún tipo de retardo de generación de su salida. Esta suposición nos ayudará a entender mejor el funcionamiento de los circuitos que diseñemos al no tener que considerar, explícitamente, el tiempo que tarda en generarse cada señal.

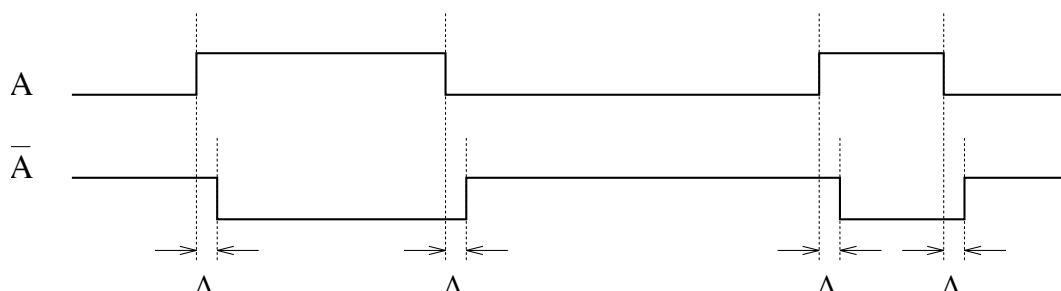


Figura 1.7: Retardo ( $\Delta$ ) en la generación de la salida de un inversor

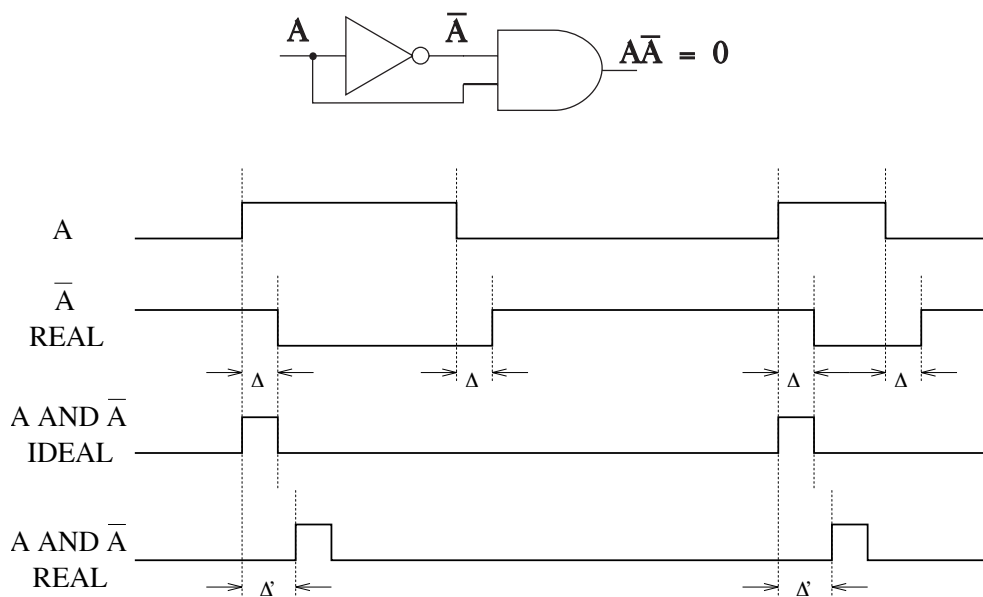


Figura 1.8: Circuito sencillo con azares.

Sin embargo, conviene tener presente que ese retardo existe y es real, lo cual provoca que, en la práctica, los circuitos se aparten ligeramente de su comportamiento ideal predicho por la teoría. Generalmente, en sistemas combinatoriales, el problema se reduce a que, durante un cierto tiempo, las salidas del sistema no serán correctas. Este periodo transitorio es el que necesita el sistema para generar todas las señales intermedias. En el caso de sistemas secuenciales el problema se resuelve haciendo que el sistema sea síncrono (que todas las variables de estado se modifiquen a la vez). En el caso de sistemas secuenciales asíncronos, el problema de los retardos es crucial y es necesario prestarle una atención y técnicas de diseño específicas para conseguir que el sistema funcione correctamente.

En la figura 1.8 se puede ver un ejemplo de los *azares* o salidas transitorias que se pueden generar en un circuito debido a los retardos de las puertas reales. En este circuito teóricamente la salida debería dar siempre cero ( $A\bar{A} = 0$ ), sin embargo podemos observar que durante un pequeño intervalo (que coincide con el tiempo que tarda el inversor en invertir su señal de entrada) la salida es 1. Es lo que se llama una espiga. El ancho de la espiga depende de la diferencia de retardos que van acumulando cada una de las señales al ir recorriendo las distintas puertas por las que debe pasar cada señal. En este caso, la

espiga se debe a que una de las entradas pasa por el inversor y la otra no. El ancho de la espiga es por lo tanto el retardo de una puerta NOT. Como se indica en la figura, el retardo de una puerta NOT no tiene por qué coincidir con el de una puerta AND.

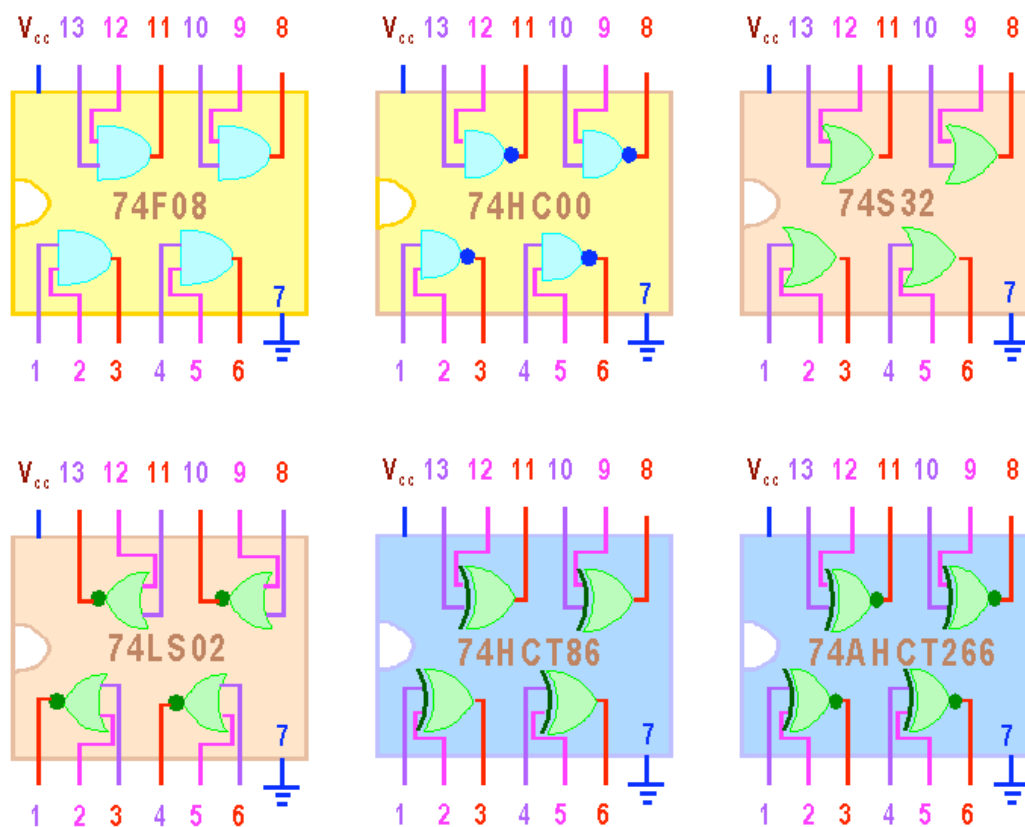


Figura 1.9: Ejemplos de circuitos integrados comerciales.