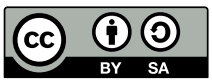
Conceptos elementales de computadores

Sergio Barrachina Mir Germán León Navarro
José Vicente Martí Avilés

Área de Arquitectura y Tecnología de Computadores
Universitat Jaume I



Índice general

Índice general III

Índice de figuras V

Índice de cuadros IX

1 Introducción **1**
1.1. Introducción al computador 1
1.2. Concepto de arquitectura de un computador 2
1.3. Arquitectura de Von Neumann 3
1.4. Niveles estructurales de Bell y Newell 4

2 Representación de la información **7**
2.1. Representación e interpretación 7
2.2. Representación de números enteros positivos 8
2.3. Representación con signo 12
2.4. Representación de números reales 15
2.5. Representación de caracteres alfanuméricos 20
2.6. Sistemas de numeración octal y hexadecimal 21

3 Álgebra de Boole **23**
3.1. Términos del álgebra de Boole 23
3.2. Representación de funciones booleanas 24
3.3. Simplificación de funciones: Mapas de Karnaugh 26
3.4. Introducción a las puertas lógicas 30

4 Circuitos combinacionales **33**
4.1. Definición y clasificación de circuitos 33
4.2. Análisis de circuitos combinacionales 35
4.3. Síntesis de circuitos combinacionales 35
4.4. Circuitos combinacionales integrados (MSI) 41

5 Circuitos secuenciales **55**
5.1. Introducción 55
5.2. Biestables 57
5.3. Descripción funcional de un circuito secuencial 62
5.4. Análisis de circuitos secuenciales 66
5.5. Síntesis de circuitos secuenciales 69
5.6. Registros 76
5.7. Contadores 77

6 Memorias **83**
6.1. Características de las memorias 83
6.2. Jerarquía de memorias 87
6.3. La memoria principal del computador 88
6.4. Asociación de memorias 92

7 Unidad Central de Proceso **97**
7.1. Estructura de la Unidad Central de Proceso 97
7.2. Ciclo de trabajo 98
7.3. Excepciones e interrupciones 99
7.4. Formato de instrucción 99
7.5. Lógica cableada y lógica microprogramada 101
7.6. RISC y CISC 102
7.7. El microprocesador R2000 103

8 Unidad de Entrada/Salida **109**
8.1. Introducción 109
8.2. Comunicación física 111
8.3. Métodos de sincronización 113

Bibliografía **115**

Índice de figuras

1.1. Componentes de un computador 3

2.1. Representación e interpretación de la información 8
2.2. Ejemplo del método de divisiones sucesivas 10
2.3. Ejemplo del método de substracciones sucesivas 10
2.4. Código Gray o binario reflejado 11
2.5. Rango de la representación en exceso Z 15
2.6. Representación del $12,375$ en binario con coma fija 16
2.7. Representación de los formatos IEEE 754 de simple y doble precisión . . 19
2.8. Rango de representación en coma flotante 19
2.9. Representación del formato IEEE 754 'reducido' 20

3.1. Obtención de las formas canónicas de una función 26
3.2. Ejemplo de mapa de Karnaugh de una función de dos variables: (a) tabla de verdad, (b) mapa de Karnaugh, (c) minitérminos y (d) maxitérminos 27
3.3. Ejemplos de simplificación de funciones lógicas utilizando mapas de Karnaugh 29
3.4. Principales puertas lógicas 31

4.1. Circuito mal formado: cortocircuito 34
4.2. Circuitos combinacionales mal formados: realimentación 35
4.3. Ejemplo de análisis de circuitos combinacionales: circuito a analizar . . 35
4.4. Ejemplo de análisis de circuitos combinacionales: tablas de verdad y funciones algebraicas 36
4.5. Sumador de 2 bits 38
4.6. Mapas de Karnaugh de las salidas del sumador de 2 bits: R2 (a), R1 (b) y R0 (c) 38
4.7. Asociación de puertas NAND de 2 entradas para obtener una puerta NAND de 3. La figura (a) representa la forma correcta de hacerlo y (b) la incorrecta 40
4.8. Implementación de R2 sólo con puertas NAND 40
4.9. Implementación de R1 sólo con puertas NAND 41
4.10. Implementación de R0 sólo con puertas NAND 42
4.11. Decodificador $n \times 2^n$ 42
4.12. Tabla de verdad y circuito de un decodificador 1×2 con salidas activas a nivel alto 43
4.13. Tabla de verdad de un decodificador 1×2 con salidas activas a nivel alto y entrada *Enable* activa a nivel alto (a) o a nivel bajo (b) 44

4.14. Tabla de verdad de un decodificador 1×2 con salidas activas a nivel bajo y entrada *Enable* activa a nivel alto (a) o a nivel bajo (b) 44
4.15. Tabla de verdad y esquemático del decodificador 3×8 74xx138 44
4.16. Implementación de la función F utilizando un decodificador con salidas activas a nivel alto y tomando los unos de la función (a) o tomando los ceros de la función (b); utilizando un decodificador con salidas activas a nivel bajo y tomando los unos de la función (c) o tomando los ceros de la función (d) 46
4.17. Obtención de un decodificador 2×4 utilizando decodificadores 1×2 . . 47
4.18. Obtención de un decodificador 3×8 utilizando decodificadores 1×2 y 2×4 48
4.19. Codificador $8 : 3$ con la entrada 6 activa 48
4.20. Multiplexor 50
4.21. Ejemplo de asociación de multiplexores 51
4.22. Implementación de una función de 3 variables utilizando un multiplexor $8 : 1$ 52
4.23. Implementación de una función de 3 variables utilizando un multiplexor $4 : 1$ y como máximo un inversor 53
4.24. Demultiplexor $1 : 2^n$ 53

5.1. Señal de reloj 56
5.2. Esquema básico de un circuito secuencial síncrono 56
5.3. Esquema de un biestable RS asíncrono con puertas NOR 58
5.4. Símbolo de un biestable RS asíncrono 58
5.5. Símbolos de un biestable RS síncrono activo por flanco de bajada (a) y por flanco de subida (b) 59
5.6. Tabla de funcionamiento y símbolo de un biestable JK síncrono activo por flanco de subida 59
5.7. Cronograma de un hipotético biestable JK asíncrono 60
5.8. Cronograma de un biestable JK síncrono activo por flanco de subida . . 60
5.9. Tabla de funcionamiento y símbolo de un biestable T síncrono activo por flanco de subida 61
5.10. Implementación de un biestable T utilizando un JK 61
5.11. Tabla de funcionamiento y símbolo de un biestable D activo por flanco de subida 61
5.12. Implementación de un biestable D utilizando un JK y una puerta inversora 62
5.13. Implementación de un biestable D utilizando un RS y una puerta inversora 62
5.14. Biestable JK con entradas *Preset* y *Reset* 62
5.15. Transición entre estados en la máquina de Moore 64
5.16. Transición entre estados en la máquina de Mealy 64
5.17. Tabla de estados de una máquina de Moore 65
5.18. Tabla de estados alternativa de una máquina de Moore 65
5.19. Tabla de estados de una máquina de Mealy 65
5.20. Ejemplo de análisis de css: Circuito secuencial síncrono 67
5.21. Ejemplo de análisis de css: Máquina de Moore 68
5.22. Ejemplo de análisis de css: Máquina de Mealy 69
5.23. Ejemplo de funcionamiento del sumador serie propuesto 71
5.24. Máquina de Mealy del sumador serie propuesto 71
5.25. Circuito que implementa el sumador serie propuesto 72

5.26. Ejemplo de funcionamiento del detector de secuencia propuesto 73
5.27. Máquina de Moore del detector de secuencia propuesto 73
5.28. Circuito que implementa el detector de secuencia propuesto 75
5.29. Registro de almacenamiento 76
5.30. Registro de desplazamiento a la derecha con entrada serie y salida serie 77
5.31. Registro de desplazamiento de entrada serie y salida paralelo 77
5.32. Máquina de Moore de un contador binario ascendente de 2 bits 78
5.33. Tabla de estados (a), de transiciones (b) y expresiones algebraicas de $Q_1(t+1)$ y $Q_0(t+1)$ (c) 79
5.34. Contador asíncrono binario natural descendente de 3bits 79
5.35. Contador asíncrono binario natural ascendente de 3bits 80

6.1. Jerarquía de memorias 87
6.2. Estructura interna de una ROM $2^n \times 1$ 89
6.3. Esquema de una memoria RAM $2^n \times m$ 89
6.4. Estructura interna de una memoria RAM $2^n \times m$ 90
6.5. Unificación de los buses de datos 92
6.6. Conversión de las señales de control 92
6.7. Estructura interna de una memoria RAM $2^n \times m$ 93
6.8. Memoria de $1K \times 8$ utilizando memorias $1K \times 4$ 94
6.9. Memoria de $2K \times 8$ utilizando memorias $1K \times 8$ 95

8.1. Estructuración en niveles de la Entrada/Salida 110

Índice de cuadros

2.1. Representación en binario natural 8
2.2. Código bi-quinario 12
2.3. Código BCD 12
2.4. Signo-magnitud 13
2.5. Complemento a uno 13
2.6. Complemento a dos 14
2.7. Representación en notación científica 17
2.8. Caracteres ASCII imprimibles 21
2.9. Sistemas octal y hexadecimal 22

3.1. Operaciones + y · 24
3.2. Tabla de verdad de $F(x, y, z) = xz + \bar{y}z + \bar{x}\,\bar{y}$ 25
3.3. Minitérminos y maxitérminos 25

4.1. Tabla de verdad del sumador de 2 bits 38
4.2. Relación entre minitérminos y maxitérminos y las salidas de un decodificador 45
4.3. Combinaciones posibles de decodificadores y puertas lógicas para la implementación de funciones lógicas 45
4.4. Función lógica que se desea implementar con un decodificador y una puerta lógica 46
4.5. Tabla de verdad de un codificador 4 : 2 sin prioridad 49
4.6. Tabla de verdad de un codificador 4 : 2 con prioridad 49

5.1. Tabla de transiciones de un biestable RS 57
5.2. Tabla de funcionamiento de un biestable RS 58
5.3. Ejemplo de análisis de css: Tabla de funcionamiento 67
5.4. Ejemplo de análisis de css: Tabla de transiciones (Moore) 68
5.5. Ejemplo de análisis de css: Tabla de transiciones (Mealy) 69
5.6. Tabla de estados del sumador serie propuesto 71
5.7. Tabla de transiciones del sumador serie propuesto 71
5.8. Tabla de funcionamiento del sumador serie propuesto 72
5.9. Tabla de estados del detector de secuencia propuesto 74
5.10. Tabla de transiciones del detector de secuencia propuesto 74
5.11. Tabla de funcionamiento del detector de secuencia propuesto 74
5.12. Tabla de obtención del estado futuro de un biestable T (a), JK (b) y RS (c) 75
5.13. Tabla de funcionamiento para cualquier tipo de biestables del detector de secuencia propuesto 76

5.14. Funcionamiento de un registro de desplazamiento a la derecha de 3 bits con entrada serie (E) y salida serie ($Q_2(t)$) 77
5.15. Tabla de funcionamiento de un contador asíncrono binario natural descendente de 3 bits 80
5.16. Tabla de funcionamiento de un contador asíncrono binario natural ascendente de 3 bits 81

6.1. Principales características de una memoria 84

Agradecimientos

Deseamos hacer constar nuestro agradecimiento a todos aquellos que con su apoyo, ayuda y comprensión hicieron posible la reedición de libro *Conceptos elementales de computadores* en el año 2000 y, en especial, a Rafael Mayo Gual y a Federico Prat Villar por su desinteresada colaboración en la revisión de varios de los capítulos y por sus valiosas sugerencias.

Esta nueva edición corrige las erratas que durante su uso como material docente en la asignatura de *Introducción a los Computadores* de la Universitat Jaume I fueron detectadas y reportadas por los siguientes estudiantes: Adrián Escrig Orenga, Antonio Calero Alcarria, Blanca Galve Simó, David Vicente Consuegra, Diego Fernández Pallarés, Ernesto Lázaro Pérez Espinosa, Ester García Benítez, Francisco José Lafuente Pérez, Javier Ramón Gil, José Manuel Gutierro Felip, Miguel Ángel Clavero Buj, Pau Conejero Alberola y Samuel Ibáñez Gómez. A todos ellos, nuestro más sincero agradecimiento.

Estamos seguros de que a pesar de todas las revisiones anteriores, habremos sido capaces de incorporar nuevas erratas, por lo que asumimos éstas como enteramente nuestras.

Por otro lado, confiamos en que el tiempo invertido en la reedición del presente libro permita al lector una mejor comprensión de la materia en él tratada, ayudándole en su formación. Si esto es así, daremos este tiempo como bien empleado.

Capítulo 1

Introducción

Este capítulo proporciona una breve introducción al concepto de computador y a los componentes que lo forman. También está pensado como una guía del presente libro, señalando en qué capítulos se desarrollan los conceptos aquí introducidos.

1.1. Introducción al computador

Un computador puede definirse de forma genérica como un aparato o máquina destinado a procesar información, entendiéndose por proceso las sucesivas fases, manipulaciones o transformaciones de la información destinadas a resolver un problema determinado [Mig94]. Se llama computadores electrónicos a aquellos que representan la información mediante señales eléctricas y utilizan dispositivos electrónicos para su procesamiento. Dentro de los computadores electrónicos se distinguen dos tipos de computadores según el sistema utilizado para la representación de la información:

- Computadores analógicos
- Computadores digitales

En los *computadores analógicos* la información está representada mediante un sistema analógico. Es decir, las señales son interpretadas como funciones continuas (en un determinado instante el valor de la función es un valor cualquiera y la evolución de la función con el tiempo es continua). La información representable mediante este tipo de funciones sólo puede ser de tipo numérico. Debido a esto, los computadores analógicos únicamente pueden procesar información numérica, por lo que generalmente reciben el nombre de *calculadoras analógicas*.

En los *computadores digitales*, por el contrario, la información está representada mediante un sistema digital binario, es decir, un sistema que sólo diferencia entre dos estados posibles, generalmente llamados '1' y '0'.

La unidad básica de expresión de información de un computador digital es, por tanto, un dígito binario. Esta unidad básica recibe el nombre de *bit* (*binary digit*).

Puesto que con un solo bit únicamente se puede expresar una de entre dos alternativas, por regla general, se suelen utilizar cadenas formadas por varios de estos

bits. Normalmente las cadenas de bits utilizadas son de una determinada longitud. Dependiendo de su longitud una de estas cadenas recibe el nombre de: *cuarteto* (*nibble*) si está formada por 4 bits; *octeto* (*byte*) si está formada por 8 bits; una *palabra* (*word*) si está formada por 16 bits; y *doble palabra* si está formada por 32 bits. En el Capítulo 2 se verá con más detalle cómo representar la información utilizando el sistema binario.

El gran desarrollo que han tenido, y están teniendo, las técnicas digitales se debe a las ventajas que su uso representa en el tratamiento de la información frente a otras técnicas como las analógicas. Estas ventajas se pueden resumir en [GGM92]:

- Precisión y versatilidad en el procesamiento de la información.
- Alta fiabilidad en la construcción de componentes electrónicos de conmutación.
- Capacidad de almacenamiento de información.
- Posibilidad de detección y corrección de errores en la transmisión de la información.

Debido a estas ventajas y a la importancia de los computadores digitales este libro se centrará en el estudio de los mismos.

Para que un computador digital pueda procesar información, se le ha de indicar la secuencia de pasos que ha de seguir para su procesamiento. Esta secuencia de pasos constituye la programación del mismo y al más bajo nivel se realiza en un lenguaje que éste puede entender. Este lenguaje, llamado *lenguaje máquina*, está formado por un conjunto detallado de instrucciones, que recibe el nombre de *juego de instrucciones de máquina*, donde cada instrucción está representada por una determinada cadena de bits. Cada una de estas instrucciones contiene información sobre la *operación* a realizar, los *operandos* —o la dirección de éstos— de la misma y la dirección del resultado.

1.2. Concepto de arquitectura de un computador

La arquitectura de un computador [Mig94, Sta97] se refiere a aquello que un programador en lenguaje máquina necesita saber de un computador. Comprende lo siguiente:

- Juego de instrucciones de máquina.
- Tipos y formatos de los operandos de las instrucciones de máquina.
- Mapas de memoria y de entrada/salida.
- Modelo de ejecución.

La arquitectura de un computador es una abstracción del mismo ya que define su comportamiento pero no sus características internas. En este sentido conviene diferenciar los siguientes términos:

Arquitectura Especifica lo que puede hacer el computador.

Organización o estructura interna Especifica cómo lo hace.

Tecnología Especifica los elementos físicos concretos con los que está construido el computador.

1.3. Arquitectura de Von Neumann

La arquitectura propuesta por Von Neumann en 1945 durante el desarrollo del EDVAC (*Electronic Discrete Variable Automatic Computer*) es la utilizada prácticamente en todos los computadores actuales.

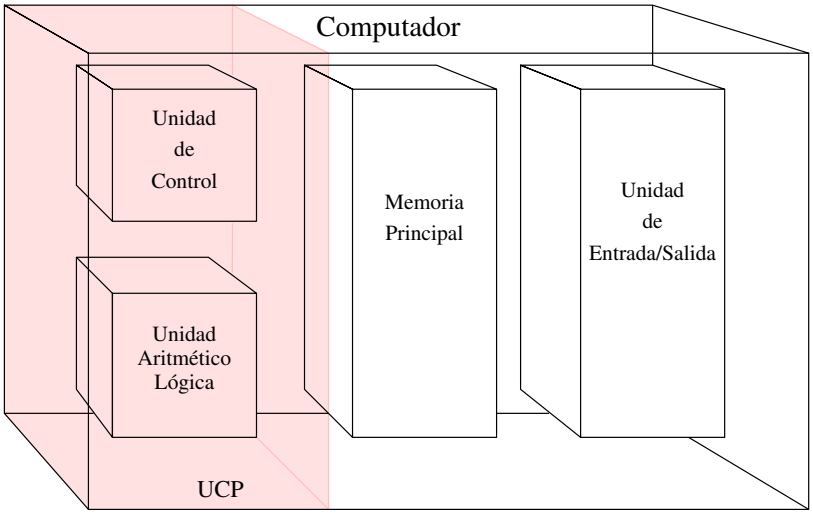


Figura 1.1: Componentes de un computador

Esta arquitectura describe el funcionamiento de un computador a partir de tres componentes (véase Figura 1.1): la *unidad central de proceso*, la *memoria principal* y la *unidad de entrada/salida*. Esta organización es independiente de la tecnología empleada para la construcción de un computador. Es decir, cualquier parte de cualquier computador, antiguo o moderno, puede clasificarse como perteneciente a una de estos tres componentes [Mig94]:

La arquitectura de Von Neumann fue llevada a la práctica en 1952 con la construcción y puesta en marcha del IAS —desarrollado por Von Neumann y otros en el Instituto Princeton para Estudios Avanzados (*Princeton Institute for Advanced Studies*). El IAS [Sta97] es, por tanto, el primer prototipo de los computadores de propósito general actuales.

Unidad central de proceso o procesador (UCP o *CPU*) Se encarga de coordinar el funcionamiento de todos los componentes de un computador. La unidad central de proceso está a su vez formada por una *unidad de control* y una *unidad aritmético lógica*. La UCP se verá en el Capítulo 7.

Memoria principal Se emplea para almacenar tanto los datos como las instrucciones de un computador. No es el único medio de almacenamiento presente en un computador, ya que la unidad central de proceso posee una serie de registros en los que se puede almacenar información y además, a través de la unidad de entrada/salida, el ordenador puede utilizar dispositivos periféricos de almacenamiento como pueden ser discos duros, disquetes, etc.. Tanto la unidad central de proceso como la unidad de entrada/salida pueden leer y escribir en la memoria principal. La memoria de un computador se verá en el Capítulo 6.

Unidad de entrada/salida Esta unidad gestiona la transferencia de información entre el computador y el exterior por medio de unas unidades que reciben el nombre de *periféricos*. Un computador puede poseer periféricos de entrada: teclado, ratón, etc.; de salida: monitor, impresora, etc. y de entrada/salida: unidad de disco duro, unidad de disquete, etc.

En resumen, la unidad central de proceso recibe instrucciones y datos de la memoria principal, la unidad de entrada/salida escribe y lee datos de la memoria principal. Además, la unidad de control envía señales que determinan el comportamiento de la unidad aritmético lógica, de la memoria principal y de la unidad de entrada/salida. Toda esta comunicación entre los distintos componentes se realiza a través de una serie de caminos llamados *buses*. Un bus es un conjunto de líneas que transmiten un determinado tipo de información. Según el tipo de información que transmiten, se puede separar el *bus del sistema* en tres partes [Hyd96]:

El bus de datos Utilizado para la transferencia de información entre los distintos componentes de un computador. El ancho del bus de datos (número de líneas del mismo) determina la cantidad de bits que pueden transferirse a la vez entre los distintos componentes del computador.

El bus de direcciones Se utiliza para determinar la posición de memoria o de entrada/salida en la que se encuentra la información a la que se desea acceder. El ancho del bus de direcciones determina la cantidad de posiciones de memoria distintas a las que puede acceder un procesador (si n es el número de líneas del bus de direcciones, se podrán direccionar hasta 2^n posiciones distintas).

El bus de control Es un conjunto de señales que controlan cómo se realiza la comunicación entre el procesador y el resto de componentes del sistema. Se verá la utilidad del bus de control con un ejemplo: El procesador envía información a la memoria y recibe información de la memoria utilizando un mismo bus de datos. Esto plantea el siguiente dilema, ¿está enviando o está recibiendo? Hay una línea en el bus de control, $R/\overline{W}$, que determina el sentido de la transferencia.

Además de esta organización, un computador de arquitectura de Von Neumann presenta las siguientes características:

- Tanto los datos como las instrucciones se almacenan en una única memoria.
- El contenido de la memoria es direccionable por posición. No hay distinción entre el acceso a un dato o a una instrucción.
- Ejecución en forma secuencial de las instrucciones.

1.4. Niveles estructurales de Bell y Newell

El *hardware* de un computador son los componentes físicos, tangibles, del mismo. El *software* está formado por aquellos programas que *dan vida* a un computador.

Tanto el *hardware* como el *software* están formados por una serie de niveles jerárquicos, donde cada nivel esconde detalles del nivel inferior [PH94]. Gracias a este principio de abstracción, tanto los diseñadores de hardware como los desarrolladores de software pueden hacer frente a la complejidad de los sistemas computacionales.

En cuanto al hardware de un computador, Bell y Newell proponen una división en niveles estructurales de tal forma que los elementos utilizados en cada nivel son los conjuntos o sistemas construidos en el nivel inferior [Mig94]. Es decir, para cada nivel se parte de una serie de elementos de los que se conoce su funcionamiento pero de los que no interesa saber cómo han sido construidos. Cada nivel se diferencia por un lenguaje con el que se describe su estructura y funcionamiento, por unas leyes de comportamiento y por unas reglas o métodos de diseño propios.

El primer nivel es el **nivel de componentes**. Partiendo de los materiales de fabricación: semiconductores de tipo n y p, metales, etc., se construyen: diodos, transistores, resistencias, condensadores, etc.

El segundo nivel es el **nivel de circuito electrónico**. Utilizando como primitivas los componentes del primer nivel se diseñan puertas lógicas, biestables, osciladores, etc.

El tercer nivel corresponde al **nivel de circuito digital**. Este nivel se divide en dos subniveles: el de **circuitos combinacionales** y el de **circuitos secuenciales**. En el primer subnivel se utilizan puertas lógicas para la construcción de sistemas combinacionales: multiplexores, decodificadores, etc. En el subnivel secuencial se utilizan además circuitos de memoria tipo biestable y se generan: contadores, registros, etc. Las puertas lógicas se describen en el Capítulo 3, el subnivel de circuitos combinacionales se estudia en el Capítulo 4 y el subnivel de circuitos secuenciales en el Capítulo 5.

El cuarto nivel es el **nivel de transferencia entre registros** (RT, *Registry Transference*). Los elementos constructivos son los buses, registros, bloques combinacionales, memorias, etc. Los sistemas formados serán bloques funcionales del computador o este mismo. Este cuarto nivel está tratado en los Capítulos 6 y 7.

El quinto y último nivel corresponde al **nivel de intercambio procesador memoria** (PMS, *Processor Memory Switch*). Éste es el nivel correspondiente al mayor grado de abstracción posible y proporciona una descripción detallada del funcionamiento del computador.

Capítulo 2

Representación de la información

Los sistemas digitales presentan una serie de ventajas sobre los analógicos que los hacen interesantes para ciertas aplicaciones, entre sus ventajas, destacan su simplicidad de diseño y su inmunidad al ruido eléctrico. Sin embargo, para que un sistema digital pueda procesar información, es necesario primero representarla en forma de '0's y '1's. Este capítulo muestra cómo se representan e interpretan distintos tipos de información en un sistema digital.

2.1. Representación e interpretación

Para expresar la información en un lenguaje dado y posteriormente poder recuperarla, es necesario definir dos operaciones: *representación* e *interpretación*.

La **representación** es la operación por la que se expresa una determinada información como una combinación particular de símbolos de un determinado lenguaje. Así por ejemplo, el número *trece* se representa mediante los símbolos 1 y 3; y el color *azul* mediante 'a', 'z', 'u' y 'l'.

La **interpretación** es la operación inversa a la anterior. Es decir, aquella operación que a partir de una combinación de símbolos obtiene la información originalmente representada. Así por ejemplo, los símbolos 13 se interpretan como el número 'trece'. Para que la interpretación sea correcta es necesario conocer el código que fue utilizado para representar la información que se desea interpretar. Por poner un ejemplo, el número 100 puede significar cien o cuatro dependiendo del sistema de numeración empleado.

En resumen, la representación permite expresar cualquier **información** (conceptos) en forma de **datos** (símbolos) utilizando un **código** (normas) determinado. La interpretación permite recuperar la **información** original conocido el **código**, a partir de un conjunto de **datos**. La Figura 2.1 representa gráficamente estas relaciones.

Es importante recordar que para poder interpretar correctamente una determinada información, es necesario conocer el código con el que ésta ha sido representada.

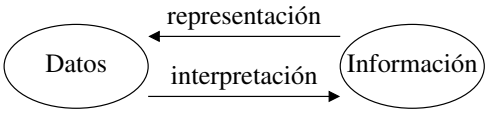


Figura 2.1: Representación e interpretación de la información

Tipos de información

La información básica a representar en un computador digital puede clasificarse en: información de tipo *numérico* e información de tipo *alfanúmerico*.

Se entiende por información numérica a aquella que expresa cantidades cuantificables. La información numérica puede estar formada por números de distinta naturaleza: números enteros positivos, números enteros con signo y números reales. Los sistemas de representación en binario de números enteros positivos, números enteros con signo y números reales se ven en las Secciones 2.2, 2.3 y 2.4, respectivamente.

Se denomina información alfanumérica a aquella información de tipo texto y que está formada por caracteres que pueden representar letras, números o símbolos especiales. La representación de información alfanumérica se ve en la Sección 2.5.

2.2. Representación de números enteros positivos

En este apartado se describen los siguientes sistemas de codificación de números enteros positivos: el binario natural, el código Gray, el código 2 entre 5 y el BCD.

Binario natural

Este código o sistema de representación es el resultado de aplicar el mismo sistema empleado para la numeración en decimal (que posee diez símbolos distintos: '0'–'9') pero utilizando sólo dos símbolos: el '0' y el '1'. El procedimiento por el que se obtiene el siguiente número a uno dado en ambos sistemas es el siguiente (se empieza con el dígito situado más a la derecha del número): se incrementa el dígito en una unidad sustituyendo el símbolo actual por el siguiente símbolo de entre los disponibles, si no quedan símbolos, se sustituye el símbolo actual por el '0' y se repite el procedimiento con el dígito situado a la izquierda del actual.

La Tabla 2.1 muestra los números del 0 al 31 representados en *binario natural*.

0 - 0	8 - 1000	16 - 10000	24 - 11000
1 - 1	9 - 1001	17 - 10001	25 - 11001
2 - 10	10 - 1010	18 - 10010	26 - 11010
3 - 11	11 - 1011	19 - 10011	27 - 11011
4 - 100	12 - 1100	20 - 10100	28 - 11100
5 - 101	13 - 1101	21 - 10101	29 - 11101
6 - 110	14 - 1110	22 - 10110	30 - 11110
7 - 111	15 - 1111	23 - 10111	31 - 11111

Cuadro 2.1: Representación en binario natural

El sistema binario de numeración es, al igual que el decimal, un sistema de numeración posicional: el valor de un dígito depende de la posición que éste ocupa dentro del número. Esto quiere decir que un número como el 435 es interpretado como $400 + 30 + 5 = 4 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$.

El bit situado más a la izquierda de un número binario recibe el nombre de *bit más significativo* (*MSB*) y el situado más a la derecha recibe el de *bit menos significativo* (*LSB*).

En general, dado un número expresado como una cadena de dígitos de longitud n y base b: $x_{n-1} \cdots x_1 x_{0_b}$, éste puede interpretarse de la siguiente forma:

$$x_{n-1} \cdots x_1 x_{0_b} = x_{n-1} b^{n-1} + \cdots + x_1 b^1 + x_0 b^0$$

Que particularizando para números binarios (base igual a 2), queda:

$$x_{n-1} \cdots x_1 x_{0_2} = x_{n-1} 2^{n-1} + \cdots + x_1 2^1 + x_0 2^0 \tag{2.1}$$

Por lo tanto, para interpretar un número en binario natural bastará con aplicar la fórmula anterior. Así, por ejemplo:

$$11010_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 16 + 8 + 0 + 2 + 0 = 26_{10},$$

como puede comprobarse si se consulta la Tabla 2.1.

Teniendo en cuenta la Ecuación 2.1 se pueden deducir fácilmente las siguientes igualdades que facilitan la conversión entre binario y decimal:

$$(1\overbrace{0\ldots0}^{n})_2 = 2^n \quad (\overbrace{1\ldots1}^{n})_2 = 2^n - 1 \tag{2.2}$$

Métodos de representación en binario natural

Para *representar* un número en binario natural se pueden utilizar varios métodos. Dos de estos métodos son el de divisiones sucesivas y el de substracciones sucesivas. Estos métodos se explican a continuación.

Método de divisiones sucesivas

Este método puede utilizarse para la conversión entre dos bases cualesquiera (teniendo en cuenta que las operaciones necesarias —divisiones— se han de realizar en la aritmética de la base de partida). A continuación se detallará para el caso particular de la conversión de un número en base decimal a binaria.

Para representar un número decimal en binario utilizando este método se han de realizar los siguientes pasos:

1. Se divide el número por dos y se anota el resto.
2. Mientras el cociente de la última división realizada sea mayor a 1, éste se vuelve a dividir por dos y se anota el resto obtenido.
3. Una vez se han realizado las divisiones necesarias para que el último cociente sea igual a 1, el número binario esta formado por: el bit más significativo es igual al cociente de la última división (un 1), el siguiente bit (de izquierda a derecha) al resto de la última división, el siguiente al de la anterior, y así sucesivamente hasta llegar al resto de la primera división que será el bit menos significativo.

La Figura 2.2 muestra el procedimiento seguido para convertir el número 26 a binario.

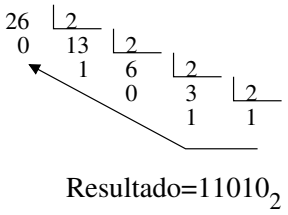


Figura 2.2: Ejemplo del método de divisiones sucesivas

Método de substracciones sucesivas

Para representar un número decimal en binario utilizando este método se han de realizar los siguientes pasos:

1. Se localiza la mayor potencia de 2 menor o igual al número a representar, 2^x.
2. Se resta del número a representar la potencia obtenida y se anota un 1.
3. Para cada potencia de 2, desde 2^{x-1} hasta 2^0, se intenta restar dicha potencia del último resultado mayor o igual a cero y si el resultado es mayor o igual a cero se anota un 1, en caso contrario se anota un 0.
4. Finalmente, la representación en binario natural la constituyen los 1s y 0s anotados en el orden en el que éstos se obtuvieron.

La Figura 2.3 muestra el procedimiento seguido para convertir el número 26 a binario.

1. Buscar la potencia de 2 más grande menor o igual que 26:
   $2^4 = 16 < 26 < 32 = 2^5 \Rightarrow$ la potencia buscada es 16
2. Restar sucesivamente:
$$\begin{aligned} 26 - 16 &= 10 \rightarrow 1 \\ 10 - 8 &= 2 \rightarrow 1 \\ 2 - 4 &= -2 \rightarrow 0 \\ 2 - 2 &= 0 \rightarrow 1 \\ 0 - 1 &= -1 \rightarrow 0 \end{aligned}$$
3. Leer de arriba a abajo:
   El número 26 en binario natural es: 11010_2

Figura 2.3: Ejemplo del método de substracciones sucesivas

Rango de representación

Se denomina *rango de representación* al conjunto de números que pueden ser representados por un sistema de representación. El rango de representación de un sistema se suele indicar por medio del intervalo comprendido entre el menor y el mayor número representables.

En principio, en binario natural puede representarse cualquier número entero positivo: cuanto mayor sea el número, mayor será el número de bits necesarios para su representación.

Sin embargo, si el número de bits disponibles está limitado, también lo estará el número de enteros representables en dicho sistema.

Así, el rango de representación del binario natural con n bits vendrá dado por los números enteros comprendidos entre el menor y el mayor número representables utilizando n bits. El menor número será aquel que tenga sus n bits a cero: $0\ldots0_2$ (0 en decimal); y el mayor el que tenga sus n bits a uno: $1\ldots1_2$ (2^n-1 en decimal, tal y como se vio en la Ecuación 2.2). Por lo tanto, el rango de representación del binario natural con n bits es:

$$[0.,2^n-1]$$

Código Gray (o binario reflejado)

La representación en binario natural no es la más adecuada para cualquier uso o aplicación. Es por ello que existen codificaciones distintas que presentan ciertas cualidades que las hacen idóneas para ciertas aplicaciones. El *código Gray* es una de estas codificaciones.

El código Gray se caracteriza por que la representación de un número y la del siguiente, y por tanto también la del anterior, tan sólo se diferencian en un bit. La Figura 2.4 muestra el código Gray correspondiente a los números del 0 al 7.

Este código cumple, además, otra propiedad: al dividir un secuencia de n bits por la mitad, las mitades obtenidas son simétricas a excepción del bit más significativo. De ahí que también reciba el nombre de *binario reflejado*.

0		00		000
1	→	01		001
		11		011
		10	→	010
				110
				111
				101
				100

Figura 2.4: Código Gray o binario reflejado

Código 2 entre 5

El *código 2 entre 5* representa cada dígito decimal mediante una combinación única de cinco bits. De estos cinco bits, sólo dos de ellos valen 1 (véase Tabla 2.2). Este código se suele utilizar para facilitar la detección de errores en la transmisión de información.

Código BCD

El *código BCD* (*Binary Coded Decimal*) emplea cuatro bits para expresar en binario natural cada una de las diez cifras del sistema decimal (véase Tabla 2.3).

0 - 00011	5 - 01100
1 - 00101	6 - 10001
2 - 00110	7 - 10010
3 - 01001	8 - 10100
4 - 01010	9 - 11000

Cuadro 2.2: Código bi-quinario

0 - 0000	5 - 0101
1 - 0001	6 - 0110
2 - 0010	7 - 0111
3 - 0011	8 - 1000
4 - 0100	9 - 1001

Cuadro 2.3: Código BCD

La representación de un número decimal en código BCD consiste en la sustitución de cada cifra del número decimal por su representación en BCD (e.g. el número decimal 435 se representa en BCD como $0100\ 0011\ 0101_{BCD}$).

Este código es utilizado principalmente en calculadoras electrónicas (al ser la aritmética BCD similar a la decimal) y en *displays* de 7 segmentos (para indicar el número a representar).

2.3. Representación con signo

En la sección anterior se han visto algunos de los sistemas de representación utilizados para la codificación de números enteros positivos. En esta se ven los sistemas de representación de números enteros (positivos y negativos).

Signo-Magnitud

La representación en *signo-magnitud* es el sistema de representación de números enteros más sencillo. En ésta, el bit más significativo indica el signo del número (0 positivo y 1 negativo); y los bits restantes el valor absoluto del mismo (en binario natural). La Tabla 2.4 muestra los números enteros del -7 al 7 representados en signo-magnitud.

Puesto que el rango de representación se divide en partes iguales entre los números positivos y negativos quedando $n-1$ bits para representar el valor absoluto del número, el rango de representación en signo-magnitud es:

$$[-\underbrace{1\ldots1}_{n-1}..\underbrace{1\ldots1}_{n-1}] = [-(2^{n-1}-1).,2^{n-1}-1]$$

Esta representación presenta los siguientes inconvenientes: se utilizan dos combinaciones distintas para representar el 0 y las operaciones aritméticas han de tener en cuenta el signo de los operandos.

0 - 0000	-0 - 1000
1 - 0001	-1 - 1001
2 - 0010	-2 - 1010
3 - 0011	-3 - 1011
4 - 0100	-4 - 1100
5 - 0101	-5 - 1101
6 - 0110	-6 - 1110
7 - 0111	-7 - 1111

Cuadro 2.4: Signo-magnitud

Complemento a uno y a dos

Los sistemas de representación en complemento obtienen la representación de un número negativo restando del total de combinaciones posibles con n bits (2^n) el número a representar y una cantidad fija. Dependiendo del valor de esta cantidad fija se distinguirá entre: *complemento a uno* y *complemento a dos*.

Complemento a uno

También denominado *complemento a la base menos uno*. Consiste en la representación de los números enteros de la siguiente forma: los números positivos se representan en signo-magnitud y los negativos en forma de complemento según la ecuación:

$$Ca1(x, n) = 2^n - 1 - x, \tag{2.3}$$

donde x es el valor absoluto del número negativo que se quiere representar y n el número de bits disponibles. La Tabla 2.5 muestra la representación de los números del -7 al 7 en complemento a uno.

0 - 0000	-7 - 1000
1 - 0001	-6 - 1001
2 - 0010	-5 - 1010
3 - 0011	-4 - 1011
4 - 0100	-3 - 1100
5 - 0101	-2 - 1101
6 - 0110	-1 - 1110
7 - 0111	-0 - 1111

Cuadro 2.5: Complemento a uno

En la práctica, para representar un número **negativo** en complemento uno, en lugar de utilizar la Ecuación 2.3, se realizan los siguientes pasos: (i) se representa en binario natural el valor absoluto del número y (ii) se intercambian los 1s por 0s y los 0s por 1s.

Para interpretar un número negativo en complemento a uno se realiza la misma secuencia de operaciones pero en sentido contrario.

En cuanto a las operaciones aritméticas binarias, éstas son más sencillas de realizar que en signo-magnitud. En particular, la suma de dos números en complemento a uno es igual a su suma binaria, más el acarreo en el caso de que se produzca. El resultado obtenido está expresado directamente en complemento a uno.

Como desventaja del complemento a uno, señalar que también presenta duplicidad en el cero.

Por último, su rango de representación con n bits es, al igual que en signo-magnitud:

$$[-(2^{n-1}-1).,2^{n-1}-1]$$

Complemento a dos

También denominado *complemento a la base*. Consiste en la representación de los números enteros de la siguiente forma: los números positivos se representan en binario natural y los negativos en forma de complemento según la ecuación:

$$Ca2(x,n) = 2^n - x, \tag{2.4}$$

donde x es el valor absoluto del número negativo que se quiere representar y n el número de bits disponibles. La Tabla 2.6 se muestra la representación en complemento a 2 de los números del -8 al 7. Como puede observarse, no hay duplicidad en el cero, al contrario de lo que ocurre en signo-magnitud y en complemento a uno.

0 - 0000	-8 - 1000
1 - 0001	-7 - 1001
2 - 0010	-6 - 1010
3 - 0011	-5 - 1011
4 - 0100	-4 - 1100
5 - 0101	-3 - 1101
6 - 0110	-2 - 1110
7 - 0111	-1 - 1111

Cuadro 2.6: Complemento a dos

En la práctica, para representar un número **negativo** en complemento dos, en lugar de utilizar la Ecuación 2.4, se realizan los siguientes pasos: (i) se representa el número en complemento a uno y (ii) se suma un 1 (si el bit más significativo del número resultante es 0 en lugar de 1, el número original está fuera del rango de representación del complemento a 2 y por lo tanto no se puede representar correctamente con n bits).

Para interpretar un número negativo en complemento a dos se realiza la misma secuencia de operaciones pero en sentido contrario.

En cuanto a las operaciones aritméticas binarias, éstas son más sencillas de realizar que en signo-magnitud. En particular, la suma de dos números en complemento a dos es igual a la suma binaria de los mismos. El resultado obtenido está expresado directamente en complemento a dos.

Por último, el rango de representación en complemento a dos con n bits es:

$$[-(2^{n-1}).,2^{n-1}-1]$$

Exceso Z

Los sistemas de representación de números enteros con signo vistos hasta ahora no respetan el orden de los números representados.

En la *representación en exceso Z* el orden de los números a representar coincide con el orden de su representación (es decir, para todo número mayor a otro, la representación en binario del primero es mayor a la representación en binario del segundo).

Para representar un número en exceso Z se suma a la cantidad a representar un valor fijo, Z, y el resultado se representa en binario natural. De esta forma, el rango de representación en exceso Z es:

$$[-Z.,2^n - 1 - Z],$$

donde se puede observar que coincide con el de representación en binario natural desplazado Z unidades hacia los números negativos (véase Figura 2.5).

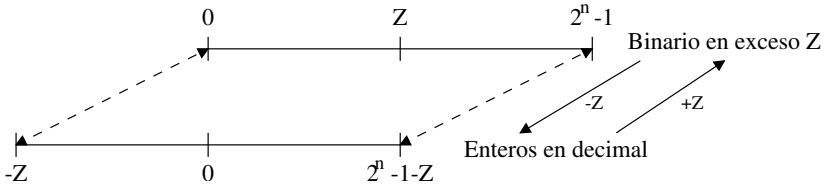


Figura 2.5: Rango de la representación en exceso Z

En cuanto a la realización de operaciones aritméticas binarias, al sumar dos números representados en exceso Z, se obtiene un número en exceso 2Z, por lo que es necesario restar Z para obtener el resultado correcto; y al restar dos números representados en exceso Z, se obtiene un número en exceso 0 y por lo tanto, es necesario añadir Z para obtener el resultado correcto. (Excepto cuando $Z = 2^{n-1}$.)

Por último, si $Z = 2^{n-1}$, la representación en exceso coincide con la representación en complemento a dos con el bit de signo invertido.

2.4. Representación de números reales

Para completar las posibilidades de representación numérica, esta sección trata la representación de números reales con coma fija y flotante.

Representación en binario con coma fija

Un número real está formado por dos partes: una entera (a la izquierda de la coma decimal) y una fraccionaria (el número real menos la parte entera). Para representar un número real en binario con coma fija bastará con representar en binario natural la parte entera y la fraccionaria de dicho número. Puesto que en las secciones anteriores se ha visto como representar un número entero en binario, se trata a continuación sólo la representación de la parte fraccionaria.

El método utilizado para representar en binario la parte fraccionaria es el siguiente:

1. Se multiplica la parte fraccionaria por dos y si el resultado es mayor a la unidad, se anota un 1, en caso contrario, se anota un 0.
2. Se elimina la parte entera del último resultado obtenido.
3. Se repiten los pasos 1 y 2 hasta que la parte fraccionaria desaparezca, los bits disponibles para la representación del número se agoten o se vea que se trata de un número periódico.
4. Los 1s y 0s anotados (en el mismo orden en el que se han obtenido) forman la parte fraccionaria (de mayor a menor peso).

La Figura 2.6 muestra la obtención de la representación con coma fija del número $12,375$ utilizando el método descrito.

Parte entera: 12
$12 = 1100_2$
Parte fraccionaria: $0,375$

$$\begin{aligned} 0,375 \times 2 &= 0{,}750 \rightarrow 0 \\ 0,750 \times 2 &= 1{,}5 \rightarrow 1 \\ 0,5 \times 2 &= 1{,}0 \rightarrow 1 \\ 0 \times 2 &= 0{,}0 \rightarrow \text{Fin} \end{aligned}$$

$0,375 = 0,011_2$
Resultado
$12,375 = 1100,011_2$

Figura 2.6: Representación del $12,375$ en binario con coma fija

La representación de números reales con coma fija presenta un problema: cantidades muy grandes o muy pequeñas en las que no todos los dígitos son significativos requieren un elevado número de bits tan sólo para indicar el orden de magnitud de la cantidad a representar. La representación con *coma flotante* soluciona este problema.

Representación en binario con coma flotante

La representación en binario con coma flotante se basa en la notación científica que a su vez consiste en la representación de un número como:

$$N = m \times b^e,$$

donde m es un número real generalmente fraccionario que corresponde a la parte significativa del número representado y que recibe el nombre de *mantisa* y la potencia b^e indica el orden de magnitud del número representado.

Cuando b coincide con la base en la que está representada la mantisa, el exponente e indica cuántas posiciones ha de desplazarse la coma en ésta para obtener el valor representado: si e es positivo la coma ha de desplazarse $|e|$ posiciones hacia la derecha, por el contrario, si e es negativo la coma ha de desplazarse $|e|$ posiciones

hacia la izquierda. En la Tabla 2.7 se muestran una serie de números representados tanto en decimal como en notación científica.

Decimal	**Notación científica**
211,00	$2{,}11 \times 10^{2}$
0,0000023	$2{,}3 \times 10^{-6}$
-1.246,7	$-1{,}2467 \times 10^{3}$
-0,0000000000017	$-1{,}7 \times 10^{-12}$
23.454.000.000,00	$2{,}3454 \times 10^{10}$

Cuadro 2.7: Representación en notación científica

Al igual que se utiliza la notación científica para la representación de números decimales, ésta puede utilizarse para la representación de números binarios.

De esta forma, las siguientes representaciones binarias son equivalentes:

$$\begin{array}{rr} 1100100000000000_2 & 0,11001_2 \times 2^{16} \\ -0,00000000010101_2 & -0,10101_2 \times 2^{-9} \end{array}$$

Como se puede ver, un número binario se expresa en coma flotante en la forma:

$$N = m \times 2^{e},$$

donde m y e son números enteros que representan ala mantisa y al exponente respectivamente. (Aunque la base habitualmente es 2 no es necesario que lo sea, puede ser una potencia de 2.)

Al expresar cantidades en notación científica decimal, se suele utilizar la notación científica *normalizada*. Por normalizada se entiende que la coma decimal de la mantisa se sitúa a la **derecha** de la primera cifra significativa, es decir, para la mantisa se verifica que: $10,0 < mantisa \leq 1,0$.

De igual forma, al representar un número en binario con coma flotante, también se suele utilizar una representación normalizada ya que esto facilita las operaciones sobre las cantidades así representadas. Existen varias formas de normalización, una de ellas consiste en que la coma binaria de la mantisa se sitúe a la **izquierda** del primer bit significativo (distinto de 0), es decir, la mantisa binaria verifica que: $1,0 < mantisa \leq 0,0$.

Un número real se puede representar en coma flotante normalizada en la forma:

$$N = [-]0,M \times 2^{e}$$

Por lo tanto, la información que es necesario almacenar será: la parte significativa de la mantisa M (y su signo) y el exponente e.

Tanto la parte significativa de la mantisa como el exponente pueden ser representados utilizando alguno de los formatos de representación de números enteros ya vistos. La parte significativa de la mantisa, $[-]M$, puede considerarse como un número entero (aunque la mantisa completa $0,M$ es en realidad un número fraccionario) y suele utilizarse el sistema de representación en signo-magnitud para su representación. El exponente suele representarse mediante el sistema de representación en exceso Z para facilitar las comparaciones.

Al utilizar la representación en coma flotante normalizada, el primer bit de la parte significativa de la mantisa, M, será, por definición, siempre un 1. Puesto que el primer bit es siempre un 1 se puede expresar la mantisa como $0,1M$, o como $1,M$ si la normalización se realiza colocando la coma binaria a la derecha del primer 1 en lugar de a la izquierda. Estas formas de expresar la mantisa reciben el nombre de mantisa con *bit implícito* fraccionario y entero, respectivamente.

Las tres formas en las que puede expresarse la mantisa son, por tanto:

- Mantisa sin bit implícito: $[-]0,M$.
- Mantisa con bit implícito fraccionario: $[-]0,1M$.
- Mantisa con bit implícito entero: $[-]1,M$.

M es en los tres casos la cadena de bits que es necesario almacenar en la representación en coma flotante.

IEEE 754

Uno de los formatos de representación en coma flotante más empleados es el definido por el IEEE (*Institute of Electrical and Electronics Engineers*) con el número de orden 754: el IEEE 754 ('i e cubo siete cinco cuatro').

Esta norma propone dos formatos: uno de *simple precisión*, con 32 bits, y otro de *doble precisión*, con 64 bits. Las especificaciones comunes a ambos son:

- Mantisa fraccionaria normalizada con bit implícito entero expresada en signo-magnitud ($[-]1,M$).
- Exponente expresado en exceso $2^{q-1}-1$, donde q es el número de bits del exponente.

Con estas especificaciones, el número binario representado mediante este formato puede expresarse como:

$$N = (-1)^S \cdot 1,M \times 2^{E-Z},$$

donde S es un bit que representa el signo (el término $(-1)^S$ será negativo, -1, cuando $S = 1$ y positivo, $+1$, cuando $S = 0$); M es la parte significativa de la mantisa; E el exponente expresado en exceso Z ($Z = 127$ en el formato de simple precisión y $Z = 1023$ en el de doble precisión). Utilizando la expresión anterior se puede interpretar cualquier valor representado coma flotante.

Por lo tanto, los n bits disponibles para la representación del número se dividen en tres *campos*: uno para almacenar el bit de signo (S), otro para almacenar el exponente en exceso $2^{q-1}-1$ (E) y un tercero para almacenar la parte significativa de la mantisa (M). La Figura 2.7 muestra la disposición de estos campos en los formatos IEEE 754 de simple y doble precisión.

El orden en el que se presentan los campos, así como los sistemas de representación elegidos para la representación de la mantisa (signo-magnitud separando el bit de signo) y del exponente (en exceso $2^{q-1}-1$) fueron elegidos de tal forma que la comparación de cantidades representadas en estos formatos sea fácil.

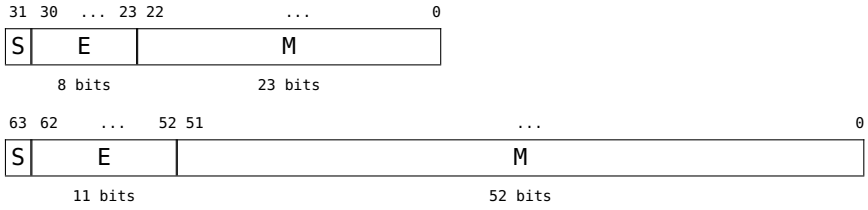


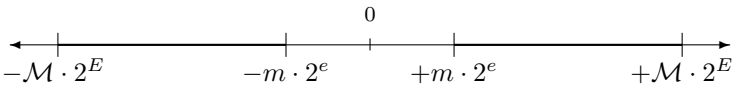
Figura 2.7: Representación de los formatos IEEE 754 de simple y doble precisión

Rango y resolución del estándar IEEE 754

Un número binario representado mediante este formato puede expresarse como:

$$N = (-1)^S \cdot 1, M \times 2^{E-(2^{q-1}-1)}$$

El rango de representación en coma flotante está comprendido entre el menor y el mayor número representable en dicho formato. Pero además, si la representación utiliza bit implícito, como la mantisa nunca puede valer 0, un rango de números alrededor del 0 no pueden ser representados. La Figura 2.8 ilustra el rango de representación en coma flotante.



$-\mathcal{M}$: máximo valor negativo de la mantisa
$+\mathcal{M}$: máximo valor positivo de la mantisa
$-m$: mínimo valor negativo de la mantisa
$+m$: mínimo valor positivo de la mantisa
E : máximo valor del exponente
e : mínimo valor del exponente

Figura 2.8: Rango de representación en coma flotante

Se denomina *resolución* de un sistema de representación a la distancia que éste presenta entre cantidades consecutivas representadas en dicho sistema. En los sistemas de representación de números con coma fija, la resolución es *uniforme*, y en el caso de números enteros, además, igual a la unidad.

Por el contrario, en un sistema de representación con coma flotante la resolución varía a lo largo de su rango de representación. Ésta será pequeña cuando en las proximidades del 0 y se irá haciendo cada vez mayor a medida que se consideren números cada vez más alejados del 0.

Casos especiales del estándar IEEE 754

Para representar números y resultados no representables en un formato de coma flotante, el estándar IEEE 754 contempla las siguientes combinaciones de los campos E y M como casos especiales denominados *desnormalizados* [Mig94]:

- $\boldsymbol{E = 2^q - 1}$ **y** $\boldsymbol{M \neq 0}$. Se emplea para indicar que el resultado de una operación no tiene sentido. Por ejemplo, el resultado de 0/0.

- $\boldsymbol{E = 2^q - 1}$ **y** $\boldsymbol{M = 0}$. Se emplea para indicar un valor infinito positivo o negativo (dependiendo del valor del bit S).
- $\boldsymbol{E = 0}$ **y** $\boldsymbol{M = 0}$. Se emplea para representar el cero.
- $\boldsymbol{E = 0}$ **y** $\boldsymbol{M \neq 0}$. Se emplea para representar números muy pequeños de forma desnormalizada y cubrir el hueco dejado en torno al cero. El número se calcula en este caso como: $(-1)^S \cdot 0, M \times 2^{-(2^{q-1}-1)}$.

IEEE 754 reducido

Basándose en las características de los formatos IEEE 754, se propone, para la realización de ejercicios, un formato IEEE 754 'reducido' representado en la Figura 2.9. Al utilizar sólo 16 bits, los cálculos necesarios para la representación e interpretación de números en dicho formato son considerablemente más sencillos que si se utilizara cualquiera de los formatos *verdaderos*.

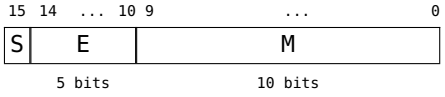


Figura 2.9: Representación del formato IEEE 754 'reducido'

2.5. Representación de caracteres alfanuméricos

En cuanto a la representación de la información tipo texto escrito, ésta se realiza codificando, mediante un conjunto de bits, cada uno de los caracteres que componen dicha información.

Las características que distinguen a los diversos tipos de representaciones alfanuméricas existentes son: el número de bits utilizados para almacenar cada uno de los caracteres —como ya se ha visto, a mayor número de bits mayor será el número de caracteres distintos que se podrán representar— y el código utilizado para representar cada carácter.

El sistema de representación alfanumérico más empleado en la actualidad es el ASCII (*American Standard Code for Interchange of Information*). El código ASCII utiliza 7 bits y posee 128 caracteres alfanuméricos.

La Tabla 2.8 muestra los códigos ASCII correspondientes a los caracteres alfanuméricos imprimibles (los códigos del 0 al 31 y el 127 no están representados).

Existen varios códigos de representación de caracteres alfanuméricos que utilizan 8 bits para proporcionar un mayor número de caracteres que el del código ASCII. Estos caracteres extra (127) se utilizan para la representación de símbolos gráficos, matemáticos y letras no anglosajonas ('ñ', letras acentuadas, ...) que el código ASCII no contempla.

Entre las extensiones del código ASCII más difundidas figuran el *ASCII extendido* utilizado por el sistema operativo DOS y el *ISO Latín 1* que se ha convertido en un estándar más extendido que el anterior al ser utilizado por varios sistemas

32		48	0	64	@	80	P	96	‘	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	’	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o		

Cuadro 2.8: Caracteres ASCII imprimibles

operativos actuales y por aplicaciones de Internet (navegadores, lectores de correo, etc.).

2.6. Sistemas de numeración octal y hexadecimal

Al ser la base del sistema binario una base tan pequeña, el número de dígitos necesarios para representar números grandes crece con rapidez. Debido a esto, trabajar directamente en binario es una tarea muy engorrosa. Para poner un ejemplo, el número 1444 que en decimal se representa con 4 dígitos, necesita 11 dígitos cuando se representa en binario: 10110100100_2.

Para facilitar el trabajo con cantidades binarias, se utilizan sistemas de numeración donde cada dígito en dicho sistema equivale a un determinado número de dígitos binarios consecutivos[1]. Los dos sistemas de numeración más empleados con esta finalidad son: el *octal* y el *hexadecimal*.

El sistema octal es un sistema de numeración de base 8 y por lo tanto dispone de 8 símbolos distintos. Para la representación de estos símbolos se utilizan los números del 0 al 7.

La conversión entre las bases binaria y octal es muy sencilla, basta con agrupar de tres en tres los dígitos binarios y sustituir cada grupo de tres bits por el dígito octal correspondiente empezando por el bit menos significativo. Por ejemplo, el número binario 10110100100_2 se representa en octal como 2644_8:

$$\underbrace{010}_{2}\underbrace{110}_{6}\underbrace{100}_{4}\underbrace{100}_{4}$$

[1]Esta relación entre sistemas de numeración sólo es posible cuando la base de uno es potencia exacta de la del otro.

El sistema hexadecimal es un sistema de numeración de base 16 y por lo tanto dispone de 16 símbolos distintos. Para la representación de estos símbolos se utilizan los números del 0 al 9 y las primeras 6 letras del alfabeto (A, B, C, D, E y F).

La conversión entre las bases binaria y hexadecimal es muy sencilla, basta con agrupar de cuatro en cuatro los dígitos binarios y sustituir cada grupo de cuatro bits por el dígito hexadecimal correspondiente empezando por el bit menos significativo. Por ejemplo, el número binario 10110100100_2 se representa en hexadecimal como $5A4_{16}$:

$$\underbrace{0101}_{5}\underbrace{1010}_{A}\underbrace{0100}_{4}$$

En la Tabla 2.9 se muestran las equivalencias entre las bases decimal, binaria, octal y hexadecimal para los números del 0 al 15.

dec	bin	oct	hex	dec	bin	oct	hex
0	0000	0	0	8	1000	10	8
1	0001	1	1	9	1001	11	9
2	0010	2	2	10	1010	12	A
3	0011	3	3	11	1011	13	B
4	0100	4	4	12	1100	14	C
5	0101	5	5	13	1101	15	D
6	0110	6	6	14	1110	16	E
7	0111	7	7	15	1111	17	F

Cuadro 2.9: Sistemas octal y hexadecimal

CAPÍTULO 3

Álgebra de Boole

El álgebra de Boole es una estructura matemática generada por un conjunto y dos operaciones. Ésta fue introducida por el matemático inglés George Boole en 1854 para el análisis de la lógica.

En 1938, Claude E. Shannon al aplicar el álgebra de Boole, particularizada para el caso de un conjunto con dos elementos, al estudio de circuitos eléctricos de conmutación —circuitos con dos estados posibles—, sentó las bases teóricas para el diseño de los actuales circuitos digitales.

En este capítulo se introducen los conceptos teóricos necesarios para el diseño de circuitos digitales.

3.1. Términos del álgebra de Boole

Definición 3.1 *Un conjunto $\mathcal{A}$ dotado con dos operaciones algebraicas más $(+)$ y por $(\cdot)$ es un álgebra de Boole si y sólo si se verifican los siguientes postulados:*

1. *El conjunto $\mathcal{A}$ es cerrado respecto a las dos operaciones:*
$$\forall x, y \in \mathcal{A} \Rightarrow x + y \in \mathcal{A} \wedge x \cdot y \in \mathcal{A}$$

2. *Ambas operaciones son conmutativas:*
$$\forall x, y \in \mathcal{A} \Rightarrow x + y = y + x \wedge x \cdot y = y \cdot x$$

3. *Cada una de las operaciones es distributiva con respecto a la otra:*
$$\forall x, y, z \in \mathcal{A} \Rightarrow x \cdot (y + z) = (x \cdot y) + (x \cdot z) \wedge x + (y \cdot z) = (x + y) \cdot (x + z)$$

4. *Existe el elemento identidad para las dos operaciones:*
$$\forall x \in \mathcal{A} \Rightarrow \exists 0 \mid x + 0 = x \wedge \exists 1 \mid x \cdot 1 = x$$

5. *Existe el elemento complementario:*
$$\forall x \in \mathcal{A} \Rightarrow \exists \bar{x} \mid x + \bar{x} = 1 \wedge x \cdot \bar{x} = 0$$

Si dado un conjunto $\mathcal{A} = \{0, 1\}$, se definen las operaciones + y $\cdot$ tal y como se muestran en la Tabla 3.1, éste constituye un álgebra de Boole. Este álgebra de Boole particularizada recibe el nombre de álgebra de Boole bivalente o de conmutación y es la que se utiliza para el diseño de los circuitos digitales.

$0 + 0 = 0$	$0 \cdot 0 = 0$
$0 + 1 = 1$	$0 \cdot 1 = 0$
$1 + 0 = 1$	$1 \cdot 0 = 0$
$1 + 1 = 1$	$1 \cdot 1 = 1$

Cuadro 3.1: Operaciones + y ·

Leyes de De Morgan

El álgebra de Boole cumple una serie de propiedades que conviene resaltar:

$$\forall x, y \in \mathcal{A} \Rightarrow$$

$$\overline{x + y} = \bar{x} \cdot \bar{y} \quad (1^{\text{a}} \text{ Ley de De Morgan}) \tag{3.1}$$

$$\overline{x \cdot y} = \bar{x} + \bar{y} \quad (2^{\text{a}} \text{ Ley de De Morgan}) \tag{3.2}$$

Estas dos propiedades son conocidas como las leyes de De Morgan. Además, también es interesante tener en cuenta la siguiente propiedad:

$$\forall x \in \mathcal{A} \Rightarrow \bar{\bar{x}} = x$$

3.2. Representación de funciones booleanas

Definición 3.2 *Una* variable booleana*, x, es un símbolo utilizado para representar indistintamente cualquiera de los elementos de un conjunto $\mathcal{A}$ sobre el que se ha definido un álgebra de Boole.*

Definición 3.3 *Una* función booleana *o* función lógica *es una aplicación de $\mathcal{A}^n$ en $\mathcal{A}$ que asocia a cada tupla de $\mathcal{A}^n$ un elemento de $\mathcal{A}$:*

$$\begin{array}{rccc} \mathrm{F}: & \mathcal{A}^n & \longrightarrow & \mathcal{A} \\ & (x_1, x_2, \ldots, x_n) & & x \end{array}$$

Una función booleana se puede representar mediante una *expresión algebraica* o mediante una *tabla de verdad*.

Se denomina expresión booleana de una función $\mathrm{F}(x_1, x_2, \ldots, x_n)$ a la representación algebraica de la misma utilizando las operaciones $+$, $\cdot$ y el complementario. Un ejemplo de expresión booleana es:

$$\mathrm{F}(x, y, z) = x \cdot z + \bar{y} \cdot z + \bar{x} \cdot \bar{y}$$

No es necesario indicar explícitamente la operación $\cdot$, por lo que la función anterior puede escribirse también en la forma:

$$\mathrm{F}(x, y, z) = xz + \bar{y}z + \bar{x}\,\bar{y}$$

Otra forma de representar una función booleana es mediante su tabla de verdad. La tabla de verdad de un función booleana recoge, por un lado, todas las posibles combinaciones de las variables x_1, x_2, …, x_n y por otro, el valor de dicha función para cada una de estas combinaciones. La Tabla 3.2 presenta la tabla de verdad de la función $xz + \bar{y}z + \bar{x}\,\bar{y}$.

Se dice que dos expresiones booleanas son *equivalentes* si y solo si dan lugar a la misma tabla de verdad.

En la siguiente sección se muestra cómo realizar el paso contrario, es decir, pasar de una tabla de verdad a una expresión algebraica equivalente.

x	y	z	$\mathrm{F}(x,y,z)$
0	0	0	$\mathrm{F}(0,0,0)$
0	0	1	$\mathrm{F}(0,0,1)$
0	1	0	$\mathrm{F}(0,1,0)$
0	1	1	$\mathrm{F}(0,1,1)$
1	0	0	$\mathrm{F}(1,0,0)$
1	0	1	$\mathrm{F}(1,0,1)$
1	1	0	$\mathrm{F}(1,1,0)$
1	1	1	$\mathrm{F}(1,1,1)$

$\Rightarrow$

x	y	z	$\mathrm{F}(x,y,z)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Cuadro 3.2: Tabla de verdad de $\mathrm{F}(x,y,z) = xz + \bar{y}z + \bar{x}\,\bar{y}$

Expresión canónica de una función booleana

Definición 3.4 *Se llama* término canónico *de una función lógica a todo producto o suma en el cual aparecen todas las variables de las que depende esa función. A los términos producto se les llama* minitérminos *y a los términos suma,* maxitérminos.

La Tabla 3.3 muestra todos los minitérminos y maxitérminos posibles con tres variables.

x	y	z	Minitérminos		Maxitérminos	
0	0	0	$\bar{x}\,\bar{y}\,\bar{z}$	m_0	$x+y+z$	M_0
0	0	1	$\bar{x}\,\bar{y}z$	m_1	$x+y+\bar{z}$	M_1
0	1	0	$\bar{x}y\bar{z}$	m_2	$x+\bar{y}+z$	M_2
0	1	1	$\bar{x}yz$	m_3	$x+\bar{y}+\bar{z}$	M_3
1	0	0	$x\bar{y}\,\bar{z}$	m_4	$\bar{x}+y+z$	M_4
1	0	1	$x\bar{y}z$	m_5	$\bar{x}+y+\bar{z}$	M_5
1	1	0	$xy\bar{z}$	m_6	$\bar{x}+\bar{y}+z$	M_6
1	1	1	xyz	m_7	$\bar{x}+\bar{y}+\bar{z}$	M_7

Cuadro 3.3: Minitérminos y maxitérminos

Teorema 3.1 *Cualquier función booleana puede expresarse como una suma de minitérminos o producto de maxitérminos.*

Demostración 3.1 *Por el* teorema de expansión de Shannon*, cualquier función booleana puede descomponerse como:*

a) $\mathrm{F}(x_1,x_2,\ldots,x_n) = x_1 \cdot \mathrm{F}(1,x_2,\ldots,x_n) + \overline{x_1} \cdot \mathrm{F}(0,x_2,\ldots,x_n)$

b) $\mathrm{F}(x_1,x_2,\ldots,x_n) = (x_1 + \mathrm{F}(0,x_2,\ldots,x_n)) \cdot (\overline{x_1} + \mathrm{F}(1,x_2,\ldots,x_n))$

Si se aplica reiteradamente este teorema sobre las expresiones anteriores, se llega a:

a) $\mathrm{F}(x_1,x_2,\ldots,x_n) = \sum_{i=0}^{2^n-1} m_i \cdot F(i)$

b) $\mathrm{F}(x_1,x_2,\ldots,x_n) = \prod_{i=0}^{2^n-1} (M_i + F(i))$

donde $\mathrm{F}(i) = \mathrm{F}(a_{i_1}, a_{i_2}, \ldots, a_{i_n})$*, siendo* $a_{i_1}a_{i_2}\cdots a_{i_n}$ *la representación en binario natural de* i*.*

Por lo que una $\mathrm{F}(x_1, x_2, \ldots, x_n)$ *genérica se puede expresar como una suma de minitérminos o como un producto de maxitérminos, como se quería demostrar.*

Se denominan *formas canónicas* de una función booleana a la expresión de la misma como suma de minitérminos o como producto de maxitérminos.

Como se puede observar en la demostración anterior, los minitérminos m_i que intervienen en la forma canónica de una función booleana son aquellos para los cuales $\mathrm{F}(x_1, x_2, \ldots, x_n)|_{x_1x_2\cdots x_n=i}$ es '1'. De igual forma, los maxitérminos M_i que intervienen en la forma canónica alternativa de una función booleana son aquellos para los cuales $\mathrm{F}(x_1, x_2, \ldots, x_n)|_{x_1x_2\cdots x_n=i}$ es '0'. La Figura 3.1 presenta la forma de obtener rápidamente las *formas canónicas* de una determinada función $\mathrm{F}(x, y, x)$.

x	y	z	$\mathrm{F}(x,y,z)$		Minitérminos	Maxitérminos
0	0	0	1		m_0	
0	0	1	1		m_1	
0	1	0	0			M_2
0	1	1	0	$\Rightarrow$		M_3
1	0	0	0			M_4
1	0	1	1		m_5	
1	1	0	0			M_6
1	1	1	1		m_7	

$$\begin{aligned} F(x,y,z) &= m_0 + m_1 + m_5 + m_7 = \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z + x\bar{y}z + xyz \\ &= M_2M_3M_4M_6 = (x + \bar{y} + z)(x + \bar{y} + \bar{z})(\bar{x} + y + z)(\bar{x} + \bar{y} + z) \end{aligned}$$

Figura 3.1: Obtención de las formas canónicas de una función

3.3. Simplificación de funciones: Mapas de Karnaugh

Con el fin de emplear la menor cantidad posible de puertas lógicas a la hora de implementar una función booleana, resulta rentable intentar simplificarla. Para ello existen, a priori, tres posibilidades:

Reducción algebraica Aplicación de las propiedades del álgebra de Boole para simplificar la expresión algebraica. Este método presenta el inconveniente de no ser sistemático, por lo que no hay garantías de que las soluciones obtenidas sean las más simples.

Mapas de Karnaugh Sistema de simplificación gráfico y sistemático. Es útil cuando el número de variables es reducido.

Método de Quine-McCluskey Se utiliza cuando las funciones poseen un gran número de variables. Permite la sistematización total del método y para su aplicación se suele recurrir a un sistema computador. Este método escapa a los objetivos del presente texto.

El método de reducción algebraica se basa en intentar aplicar reiteradamente las siguientes propiedades:

$$\begin{aligned} x + x &= x \\ x + \bar{x} &= 1 \\ xx &= x \\ x\bar{x} &= 0 \end{aligned}$$

Para ver la forma de aplicar estas propiedades, supóngase la expresión $xy + \bar{x}y$. Ésta se puede expresar como $y(x + \bar{x})$, y de esta forma, aplicando la segunda propiedad de las vistas anteriormente, se deduce que: $xy + \bar{x}y = y$.

Desgraciadamente, dependiendo del orden en el que se apliquen estas propiedades, se puede llegar a distintas expresiones irreducibles a partir de una misma función, por lo que no se puede tener la seguridad de que una determinada expresión por el hecho de ser irreducible sea la mínima posible. El método de los mapas de Karnaugh permite realizar este tipo de simplificaciones de una forma sistemática, de tal forma, que la expresión que se obtiene sea siempre la más sencilla posible.

Este método fue introducido inicialmente por E.W. Veitch en 1952 y modificado ligeramente por M. Karnaugh (de ahí su nombre actual) en 1953 y se basa en la utilización de los mapas de Karnaugh para la reducción de funciones.

Un mapa de Karnaugh es básicamente una tabla de verdad de la función a simplificar, pero con la particularidad de que los términos canónicos adyacentes (combinables mediante las reglas de simplificación mostradas anteriormente) ocupan posiciones físicamente contiguas, con lo que es fácil identificar aquellos términos canónicos que pueden simplificarse con otros, ya que, con esta disposición, cada término canónico sólo puede simplificarse con sus vecinos (se entiende por vecinos los que están arriba, abajo, a la derecha o a la izquierda del término en cuestión). Adicionalmente, en los mapas de 3, 4 y más variables, las celdas situadas en un borde se consideran adyacentes a las celdas situadas en el borde contrario. Es decir, las celdas del borde superior se consideran contiguas a las del borde inferior y las del borde derecho a las del izquierdo.

x	y	F(x,y)
0	0	1
0	1	1
1	0	1
1	1	0

(a)

$x\backslash y$	0	1
0	1	1
1	1	0

(b)

$x\backslash y$	0	1
0	m_0	m_1
1	m_2	

(c)

$x\backslash y$	0	1
0		
1		M_3

(d)

Figura 3.2: Ejemplo de mapa de Karnaugh de una función de dos variables: (a) tabla de verdad, (b) mapa de Karnaugh, (c) minitérminos y (d) maxitérminos

La Figura 3.2b muestra el mapa de Karnaugh de una función de dos variables. Cada una de las celdas del mapa de Karnaugh indica el valor que toma la función cuando la entrada x es la indicada por su fila, y la entrada y la indicada por su columna. Al disponer los valores de la función de esta forma, los términos canónicos que pueden agruparse para su simplificación aparecen contiguos unos a otros.

Una vez representado el mapa de Karnaugh de una función, el siguiente paso es la agrupación de los términos canónicos para la simplificación de la función. Los

grupos deben constar de una cantidad de términos canónicos que sea potencia de dos $(1, 2, 4, 8, 16, \ldots)$ y definir figuras rectangulares (condición de adyacencia de los términos canónicos). Si el grupo está formado por un término canónico, no se simplifica nada. Si se agrupan dos términos canónicos, se puede eliminar una variable en la expresión algebraica del grupo. Si se agrupan cuatro términos se pueden eliminar dos variables. Y así sucesivamente.

La simplificación de una función booleana se realiza agrupando o bien los minitérminos (Figura 3.2c) o bien los maxitérminos (Figura 3.2d) de la función. Las expresiones simplificadas quedarán, por tanto, como una suma de productos (resultantes de la simplificación de los minitérminos) o como un producto de sumas (resultantes de la simplificación de los maxitérminos).

Para que la simplificación obtenida sea equivalente a la función original, se han de respetar las siguientes normas:

- Cada término canónico (minitérminos si se agrupan los minitérminos y maxitérminos si se agrupan los maxitérminos) ha de formar parte por lo menos de un grupo, pudiendo formar parte de más de un grupo.
- La figuras formadas en el mapa por los grupos de términos deben ser cuadrados o rectángulos.

Para conseguir la distribución óptima de grupos se han de seguir las siguientes reglas:

- Crear el menor número de grupos posibles, cada uno de ellos formado por el mayor número de términos posible.
- No considerar como grupos aquellos en los que todos sus términos forman parte de otros grupos.

Estas reglas pueden resumirse en la siguiente táctica de agrupación de términos: tomar todos los unos que no pueden combinarse con ningún otro; luego los grupos de dos que no pueden pertenecer a ningún grupo más grande, luego los de cuatro y así sucesivamente.

La Figura 3.3 muestra una serie de ejemplos de simplificación de funciones lógicas utilizando mapas de Karnaugh y agrupando por minitérminos (los 6 ejemplos en la parte superior de la figura) y por maxitérminos (los 2 ejemplos restantes).

Una vez agrupados de forma óptima los términos canónicos, hay que determinar qué expresión algebraica representa a cada uno de ellos. Si se agrupan minitérminos contiguos, la suma de éstos se puede simplificar en un único producto de variables de la función (negadas o no). Las variables que forman este producto son aquellas cuyo valor es el mismo para todos los minitérminos del grupo y aparecen en el producto tal cual si valen '1' y negadas si valen '0'. Por ejemplo, en el mapa de Karnaugh situado en la esquina superior izquierda de la Figura 3.3, el grupo formado por los minitérminos m_4, m_5, m_7 y m_6, tienen en común que para todos ellos $w = 0$ y $x = 1$, por lo que dicha agrupación se puede expresar mediante el producto $\bar{w}x$.

Si, por el contrario, se agrupan maxitérminos contiguos, el producto de éstos se puede simplificar en una única suma de variables de la función (negadas o no). Las variables que forman esta suma son aquellas cuyo valor es el mismo para todos los maxitérminos del grupo y aparecen en la suma tal cual si valen '0' y negadas si valen '1'. Por ejemplo, en el mapa de Karnaugh situado en la esquina inferior izquierda de

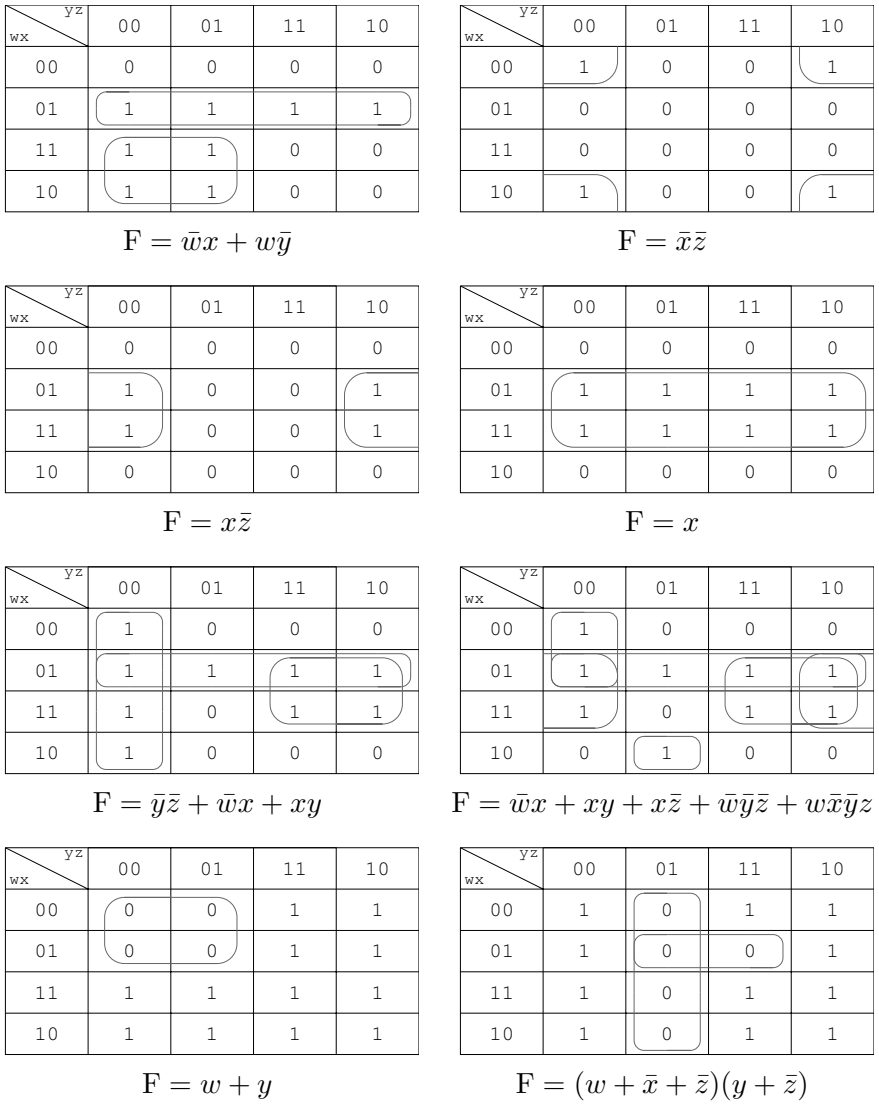


Figura 3.3: Ejemplos de simplificación de funciones lógicas utilizando mapas de Karnaugh

la Figura 3.3, el grupo formado por los maxitérminos M_0, M_1, M_4 y M_5, tienen en común que para todos ellos $w = 0$ y $y = 0$, por lo que dicha agrupación se puede expresar mediante la suma $w + y$.

Funciones incompletas

En ocasiones, la tabla de verdad de una determinada función no está completamente especificada debido a que ciertas combinaciones de entrada no pueden darse o a que, en el caso de que se den, no importa cuál sea el valor en la salida de la función lógica. A este tipo de funciones lógicas se les denomina funciones incompletas. Por ejemplo, sería el caso de un conversor de BCD a Exceso 5. Los valores de entrada correspondientes a los números binarios mayores que 9 no tendrían combinación asignada, ya que el BCD carece de esos valores. En esas posiciones de la

tabla de verdad de la función se suele colocar una equis 'X' indicando que el valor de la función para esa combinación de entrada es indiferente. Consecuentemente, en el mapa de Karnaugh tomaremos cada una de estas equis como mejor convenga, sin que sea necesario que estén todas incluidas en los grupos seleccionados; ni siquiera que alguna de ellas lo esté.

3.4. Introducción a las puertas lógicas

Todo el aparato matemático desarrollado hasta el momento a partir del álgebra de Boole está orientado a la creación de circuitos electrónicos digitales cuya salida sea una función booleana de sus entradas. Para la construcción de dichos circuitos, se pueden emplear componentes elementales denominados *puertas lógicas*.

La tecnología empleada actualmente consiste en utilizar voltajes bien diferenciados para representar el '1' lógico y el '0' lógico. De esta forma, el nivel lógico alto, asimilado al '1', corresponde a un determinado rango de voltajes y el nivel lógico bajo, asimilado al '0', a otro rango distinto. Una puerta lógica genera en su salida un valor de voltaje correspondiente a uno u otro rango en función de los valores que se encuentren presentes en su o sus entradas en ese instante.

Pueden fabricarse puertas lógicas que implementen cualquier función lógica, pero en la práctica, resulta más rentable construir solamente algunos tipos básicos de puertas e implementar las funciones lógicas complejas mediante la combinación de varias de estas puertas. Dichas puertas podría pensarse, en principio, que fuesen exclusivamente aquellas equivalentes a los operadores que definen el álgebra de Boole, es decir, la suma, el producto y el complemento. En efecto, existen puertas lógicas que implementan dichas funciones y se denominan OR, AND y NOT, respectivamente. Pero además, resulta útil disponer de tipos de puertas adicionales. Estas puertas son: las XOR y XNOR porque son costosas de construir mediante la combinación de las tres anteriores y se usan con relativa frecuencia; y las NAND y NOR por ser funcionalmente completas (esto quiere decir que cualquier función lógica puede implementarse utilizando exclusivamente puertas NAND o puertas NOR) y por ser las más simples de construir y por lo tanto más baratas.

La Figura 3.4 muestra los símbolos utilizados para la representación de las puertas lógicas mencionadas, las funciones lógicas que éstas realizan y sus correspondientes tablas de verdad.

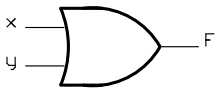
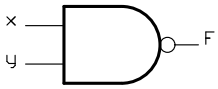
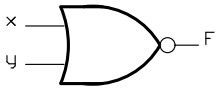
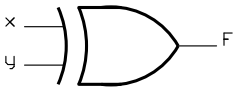
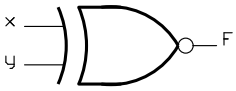
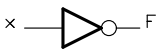
Nombre	Símbolo	Función	Tabla de Verdad
AND		$F = xy$	x y | $F(x,y)$ 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | $F(x,y)$ 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NAND		$F = \overline{xy}$	x y | $F(x,y)$ 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{x + y}$	x y | $F(x,y)$ 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = \bar{x}y + x\bar{y} = x \oplus y$	x y | $F(x,y)$ 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
XNOR		$F = \bar{x}\bar{y} + xy = \overline{x \oplus y}$	x y | $F(x,y)$ 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1
NOT		$F = \bar{x}$	x | $F(x)$ 0 | 1 1 | 0
DRIVER		$F = x$	x | $F(x)$ 0 | 0 1 | 1

Figura 3.4: Principales puertas lógicas

CAPÍTULO 4

Circuitos combinacionales

El objetivo de este capítulo es estudiar los circuitos digitales ya que éstos constituyen la base de las diferentes unidades funcionales de un ordenador. De esta forma, se obtendrá una visión global y completa del funcionamiento interno de un computador.

4.1. Definición y clasificación de circuitos

Los circuitos digitales son circuitos electrónicos, que interpretan la tensión que hay en sus entradas como valores lógicos (0 ó 1), y generan una determinada tensión en sus salidas para representar los correspondientes valores lógicos. El sistema de representación consiste en asignar a cada valor lógico un rango de tensión eléctrica. Así, el funcionamiento de un circuito digital puede especificarse como una función que asigna valores lógicos a las salidas en función de los valores lógicos presentes en las entradas.

Existen básicamente dos tipos de circuitos digitales: los circuitos **combinacionales** y los **secuenciales**. La diferencia fundamental entre ellos es la capacidad de memorizar información: los circuitos combinacionales no son capaces de memorizar información mientras que los secuenciales sí. Se dice que un circuito es combinacional si "el valor de sus salidas en un instante dado, depende únicamente del valor de sus entradas en ese mismo instante". Este capítulo trata solamente de los circuitos combinacionales dejando los circuitos secuenciales para el siguiente capítulo.

Descripción de un circuito

Existe una amplia variedad de técnicas de diseño y de implementación de circuitos. Estas técnicas, que permiten construir físicamente un circuito, se denominan **tecnologías**. Cada tecnología asume la descripción completa de un circuito atendiendo a los aspectos físicos y funcionales concretos de la misma.

En este capítulo se consideran dos tipos de descripción:

Descripción comportamental o funcional. Esta descripción es independiente de la tecnología empleada, y tan sólo determina el funcionamiento del circuito, sin tener en cuenta los aspectos relacionados con la implementación física. La definición funcional de un circuito la constituyen las funciones lógicas que relacionan sus salidas con sus entradas. Así, se puede dar la descripción funcional del circuito proporcionando su tabla de verdad o su expresión algebraica.

Descripción estructural. Se basa en la descripción del circuito por medio de una estructura de interconexiones de circuitos básicos. Los circuitos básicos que se pueden utilizar dependen de la tecnología usada para la construcción del circuito.

Para que una estructura formada por componentes interconectados constituya un circuito combinacional, debe cumplir las siguientes condiciones:

1. No se pueden conectar dos salidas de un circuito entre sí. Esto, a nivel lógico, no tiene sentido, ya que no se sabe cuál es el valor lógico asociado al punto común cuando los valores lógicos de las salidas conectadas entre sí difieren (véase Figura 4.1).

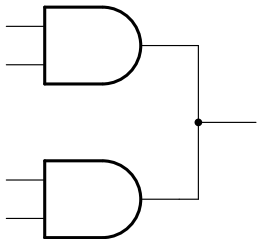


Figura 4.1: Circuito mal formado: cortocircuito

Como este tipo de conexiones provocan en algunos casos cortocircuitos que deterioran los componentes, hay tecnologías que permiten estas conexiones asociándoles una función lógica. Existen dos posibilidades (sólo una de ellas puede darse para una misma tecnología): que la conexión de varias salidas se comporte como una puerta AND o como una puerta OR, se hablará entonces de que la tecnología proporciona ANDs cableadas o ORs cableadas, respectivamente.

2. En los circuitos combinacionales no pueden existir realimentaciones. La realimentación de un circuito se produce cuando una de sus salidas influye en una o en varias de sus entradas. Dicho con otras palabras, cuando una o varias de las entradas de un circuito están en función de alguna de las salidas del mismo. La Figura 4.2 muestra dos ejemplos de circuitos combinacionales mal formados debido a la existencia de realimentación.

   Sin embargo, la realimentación implica una memorización del valor de las salidas, y ésta será utilizada para la construcción de los circuitos secuenciales que se tratan en el siguiente capítulo.

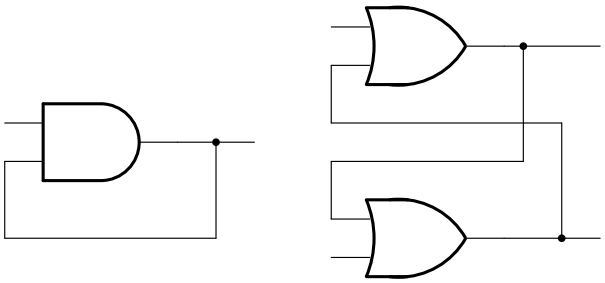


Figura 4.2: Circuitos combinacionales mal formados: realimentación

4.2. Análisis de circuitos combinacionales

El análisis de un circuito combinacional consiste en averiguar su funcionamiento a partir de una descripción estructural del mismo. Para conseguir la tabla de verdad del circuito, éste se divide en tantas partes como circuitos simples lo forman, elaborándose tablas de verdad parciales para cada una de esas partes, respetando el orden de dependencia entre ellas. La composición de estas tablas de verdad parciales proporciona la tabla de verdad del circuito.

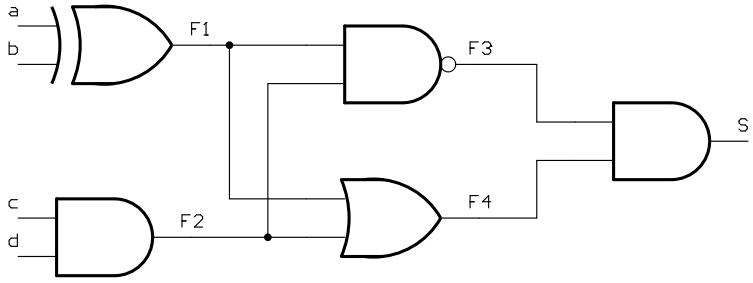


Figura 4.3: Ejemplo de análisis de circuitos combinacionales: circuito a analizar

Dado el ejemplo de la Figura 4.3, se puede observar que la función S depende de las funciones $F3$ y $F4$, que a su vez están en función de $F1$ y $F2$, y que éstas dependen directamente de las entradas del circuito a, b, c y d. Entonces, el orden de dependencia implica la obtención en primer lugar de las tablas de verdad de $F1$ y $F2$, luego las de $F3$ y $F4$ y finalmente la de S. En definitiva, el análisis del circuito propuesto es el presentado en la Figura 4.4.

4.3. Síntesis de circuitos combinacionales

La síntesis de circuitos combinaciones consiste en la implementación de los mismos partiendo de las especificaciones de los mismos. Las fases de la síntesis son:

Especificación. Existen básicamente dos formas de especificación: utilizando lenguajes de descripción de circuitos (Abel, VHDL) y mediante la representación del esquemático del circuito. La representación de esquemáticos suele realizarse utilizando sistemas de desarrollo que permiten describir gráficamente (con librerías de circuitos elementales) la forma estructural del circuito.

a	b	c	d	F1	F2	F3	F4	S
0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	0	0
0	0	1	1	0	1	1	1	1
0	1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1	1
0	1	1	0	1	0	1	1	1
0	1	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1	1
1	0	0	1	1	0	1	1	1
1	0	1	0	1	0	1	1	1
1	0	1	1	1	1	0	1	0
1	1	0	0	0	0	1	0	0
1	1	0	1	0	0	1	0	0
1	1	1	0	0	0	1	0	0
1	1	1	1	0	1	1	1	1

$$
\begin{array}{lcll}
F1 & = & XOR(a,b) & = & a \oplus b \\
F2 & = & AND(c,d) & = & c \cdot d \\
F3 & = & NAND(F1,F2) & = & \overline{F1 \cdot F2} \\
F4 & = & OR(F1,F2) & = & F1 + F2 \\
S & = & AND(F3,F4) & = & F3 \cdot F4
\end{array}
$$

Figura 4.4: Ejemplo de análisis de circuitos combinacionales: tablas de verdad y funciones algebraicas

Asimismo, la especificación puede consistir en un enunciado que describa el comportamiento del circuito.

Habrá casos en los que, si la especificación del circuito es muy detallada, no sea necesario realizar la descripción funcional del mismo y se pueda pasar directamente a la descripción estructural.

Descripción funcional. Dada la especificación de un circuito, el siguiente paso es la descripción funcional del mismo. En esta fase, el funcionamiento del circuito queda completamente especificado independientemente de cuál sea la tecnología que se vaya a emplear para su construcción.

Descripción estructural. Partiendo de la descripción funcional del circuito se realiza la descripción estructural del mismo. Los circuitos elementales utilizados en esta fase dependerán de la tecnología que se vaya a emplear en la fabricación del circuito.

De entre de los diversos aspectos dependientes de la tecnología, aquí se va a tratar únicamente el de la escala de integración proporcionada. Existen cuatro niveles de integración, a saber:

Nivel de integración	Número de puertas	Funciones integradas
Escala de integración pequeña (SSI - *Small Scale Integration*)	1–10	Puertas lógicas, biestables
Escala de integración mediana (MSI - *Medium Scale Integration*)	10–100	Decodificadores, multiplexores, registros
Escala de integración grande (LSI - *Large Scale Integration*)	100–1.000	Memorias
Escala de integración muy grande (VLSI - *Very Large Scale Integration*)	> 1.000	Memorias, microprocesadores

Este capítulo trata la síntesis de circuitos combinacionales de SSI (puertas lógicas) y MSI (circuitos combinacionales integrados).

Implementación física. En esta fase se obtiene una descripción completa del circuito. La fabricación del circuito forma parte de esta fase.

Para el caso concreto de síntesis de circuitos combinacionales utilizando puertas lógicas (SSI) se siguen los siguientes pasos:

1. Especificación del **número de entradas y salidas** del circuito a partir de la descripción funcional expresada en el enunciado del problema.

2. Obtención de las **tablas de verdad** de las diferentes funciones del circuito (una por cada salida) especificadas en el enunciado del problema. Finalizado este paso, se dispone de la descripción funcional del circuito.

3. Simplificación de las **funciones lógicas** con la finalidad de diseñar un circuito lo más sencillo posible (con el menor número de componentes).

4. Conversión de las expresiones algebraicas en forma de productos negados (NANDs) o sumas negadas (NORs). Aunque este paso no es necesario, en la práctica es bastante interesante ya que en un mismo circuito integrado suelen encapsularse varias puertas lógicas del mismo tipo y al utilizar en el diseño un único tipo de puerta lógica se puede aprovechar mejor el material disponible.

5. Obtención de la **descripción estructural**, es decir, la representación de las puertas que componen el circuito y de las interconexiones existentes entre ellas.

La secuencia de pasos anterior sólo es válida para la síntesis con puertas lógicas. Cuando se estudien los circuitos MSI (Sección 4.4) se verá cómo se pueden generar funciones lógicas utilizando estos circuitos, siendo diferentes los pasos 3 al 5.

Para exponer con mayor claridad los diferentes pasos de que consta la síntesis de circuitos combinacionales utilizando puertas lógicas, se resuelve a continuación un problema ejemplo.

Enunciado

Diseñar un sumador de dos números de dos bits utilizando únicamente puertas NAND.

Solución

1. Especificación de las entradas y salidas del circuito

En el ejemplo propuesto se tiene como entradas dos números de dos bits cada uno. Si llamamos A al primero y B al segundo, se necesitarán 4 entradas para especificar los operandos: A1 y A0 para representar el primer operando y B1 y B0 para el segundo.

Cuando un conjunto de señales representa un número, éstas se denominan con el mismo nombre y se utiliza un número de orden para diferenciarlas entre sí (este número indica el peso de cada una dentro del conjunto).

En cuanto a la salida, para determinar el número de señales necesarias, se ha de averiguar el rango de valores que puede tomar la misma. Este rango viene dado por el resultado más pequeño y el más grande que se puedan dar. Puesto que las entradas son números de 2 bits, el valor más pequeño a representar es 0 $(0 + 0)$ y el más grande 6 $(3 + 3)$. Como para la representación en binario del número 6 son necesarios 3 bits, el circuito debe poseer tres salidas (las denominaremos R2, R1 y R0).

La Figura 4.5 muestra la descripción funcional del sumador en el que se observan las entradas y salidas del mismo.

2. Obtención de la tabla de verdad

La tabla de verdad recoge el comportamiento del circuito (valores que deben tomar sus salidas) para todas las combinaciones posibles de sus entradas. En nuestro caso, los valores de entrada posibles son los correspondientes a todas las posibles

Figura 4.5: Sumador de 2 bits

combinaciones de sumas de dos números de 2 bits. La salida será la suma de los valores indicados en cada una de dichas combinaciones. Así pues, la tabla de verdad puede expresarse en la forma representada en la Tabla 4.1.

Operandos	A1	A0	B1	B0	R2	R1	R0	Resultado
0+0	0	0	0	0	0	0	0	0
0+1	0	0	0	1	0	0	1	1
0+2	0	0	1	0	0	1	0	2
0+3	0	0	1	1	0	1	1	3
1+0	0	1	0	0	0	0	1	1
1+1	0	1	0	1	0	1	0	2
1+2	0	1	1	0	0	1	1	3
1+3	0	1	1	1	1	0	0	4
2+0	1	0	0	0	0	1	0	2
2+1	1	0	0	1	0	1	1	3
2+2	1	0	1	0	1	0	0	4
2+3	1	0	1	1	1	0	1	5
3+0	1	1	0	0	0	1	1	3
3+1	1	1	0	1	1	0	0	4
3+2	1	1	1	0	1	0	1	5
3+3	1	1	1	1	1	1	0	6

Cuadro 4.1: Tabla de verdad del sumador de 2 bits

3. Simplificación de las funciones por el método de Karnaugh

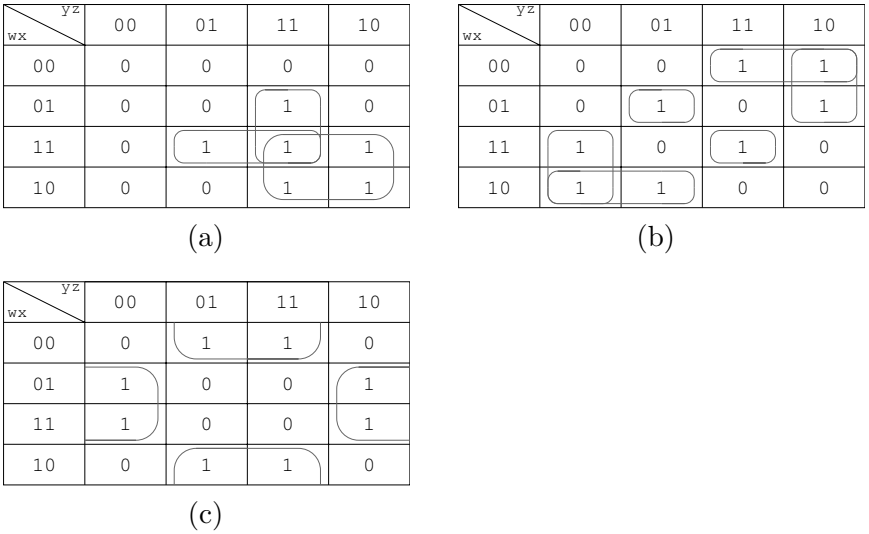


Figura 4.6: Mapas de Karnaugh de las salidas del sumador de 2 bits: R2 (a), R1 (b) y R0 (c)

La Figura 4.6 muestra los mapas de Karnaugh correspondientes a las funciones de salida: R2, R1 y R0. Si se realiza la simplificación de dichas funciones agrupando por unos se llega a las siguientes expresiones algebraicas:

$$\begin{aligned} R2(A1, A0, B1, B0) &= A1 \cdot B1 + B1 \cdot B0 \cdot A0 + A1 \cdot A0 \cdot B0 \\ R1(A1, A0, B1, B0) &= \overline{A1} \cdot \overline{A0} \cdot B1 + B1 \cdot \overline{B0} \cdot \overline{A1} + \overline{A1} \cdot A0 \cdot \overline{B1} \cdot B0 \\ &+ A1 \cdot A0 \cdot B1 \cdot B0 + A1 \cdot \overline{B1} \cdot \overline{B0} + A1 \cdot \overline{A0} \cdot \overline{B1} \\ R0(A1, A0, B1, B0) &= A0 \oplus B0 = \overline{A0} \cdot B0 + A0 \cdot \overline{B0} \end{aligned}$$

4. (OPCIONAL) Conversión de las expresiones algebraicas en forma de productos negados (NANDs) o sumas negadas (NORs)

Esta conversión se realiza de una de las siguientes formas dependiendo de si se desea utilizar únicamente puertas NAND o NOR:

- Para obtener un circuito que sólo utilice puertas NAND, se parte de una expresión algebraica simplificada en forma de suma de productos ($\sum p_i$), esta suma se complementa dos veces y se aplica la ley de De Morgan por la cual la negación de una suma de términos es igual al producto de los términos negados, obteniendo de esta forma una expresión algebraica basada solamente en productos negados que es directamente implementable con puertas NAND. De una manera más formal:

$$F = \sum p_i = \overline{\overline{\sum p_i}} = \overline{\prod \overline{p_i}}$$

- Para obtener un circuito que sólo utilice puertas NOR, se parte de una expresión algebraica simplificada en forma de producto de sumas ($\prod s_i$), este producto se complementa dos veces y luego se aplica la ley de De Morgan por la cual la negación de un producto de términos es igual a la suma de los términos negados, obteniendo de esta forma una expresión algebraica basada solamente en sumas negadas que es directamente implementable con puertas NOR. De una manera más formal:

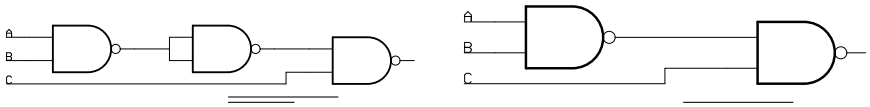
$$F = \prod s_i = \overline{\overline{\prod s_i}} = \overline{\sum \overline{s_i}}$$

En general, el enunciado de un problema de diseño suele especificar, además del tipo y número de puertas a utilizar, el número de entradas de cada una de las puertas a utilizar. Cuando éste sea el caso, hay que tener en cuenta que tanto la NAND como la NOR **no** son asociativas. Para ver esto, conviene recordar primero que las operaciones AND ($\cdot$) y OR ($+$) son asociativas ya que verifican:

$$a + b + c = a + (b + c) = (a + b) + c$$

$$a \cdot b \cdot c = a \cdot (b \cdot c) = (a \cdot b) \cdot c$$

Como ejemplo de cómo se puede resolver el problema de la asociación con puertas NAND o NOR, la Figura 4.7a representa la forma en la que se puede construir una



$\overline{A \cdot B \cdot C} = \overline{\overline{A \cdot B} \cdot C}$ (a) Correcta

$\overline{A \cdot B \cdot C} \neq \overline{\overline{A \cdot B} \cdot C}$ (b) Incorrecta

Figura 4.7: Asociación de puertas NAND de 2 entradas para obtener una puerta NAND de 3. La figura (a) representa la forma correcta de hacerlo y (b) la incorrecta

puerta NAND de 3 entradas utilizando puertas NAND de 2 entradas. Obsérvese que la Figura 4.7b muestra la forma incorrecta de hacerlo.

En el presente ejemplo, se pide que se implemente el circuito utilizando sólo puertas NAND pero no se especifica un número máximo de entradas por puerta, por lo que se pueden emplear puertas de cualquier número de entradas. Por lo tanto, la conversión de cada una de las funciones de salida a puertas NAND se realizaría de la siguiente forma:

La salida $R2$ puede expresarse como:

$$\begin{aligned} R2(A1, A0, B1, B0) &= A1 \cdot B1 + B1 \cdot B0 \cdot A0 + A1 \cdot A0 \cdot B0 \\ &= \overline{\overline{A1 \cdot B1 + B1 \cdot B0 \cdot A0 + A1 \cdot A0 \cdot B0}} \\ &= \overline{\overline{A1 \cdot B1} \cdot \overline{B1 \cdot B0 \cdot A0} \cdot \overline{A1 \cdot A0 \cdot B0}} \end{aligned}$$

La implementación de R2 se muestra en la Figura 4.8.

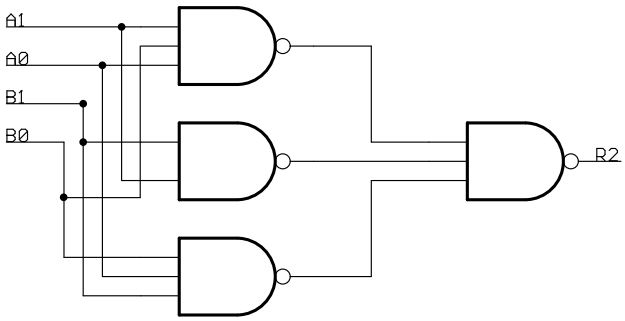


Figura 4.8: Implementación de R2 sólo con puertas NAND

La salida $R1$ puede expresarse como:

$$\begin{aligned} R1(A1, A0, B1, B0) &= \overline{A1} \cdot \overline{A0} \cdot B1 + B1 \cdot \overline{B0} \cdot \overline{A1} + \overline{A1} \cdot A0 \cdot \overline{B1} \cdot B0 \\ &\quad + A1 \cdot A0 \cdot B1 \cdot B0 + A1 \cdot \overline{B1} \cdot \overline{B0} + A1 \cdot \overline{A0} \cdot \overline{B1} \\ &= \overline{\overline{\overline{A1} \cdot \overline{A0} \cdot B1 + B1 \cdot \overline{B0} \cdot \overline{A1} + \overline{A1} \cdot A0 \cdot \overline{B1} \cdot B0} \\ &\quad \overline{+A1 \cdot A0 \cdot B1 \cdot B0 + A1 \cdot \overline{B1} \cdot \overline{B0} + A1 \cdot \overline{A0} \cdot \overline{B1}} \\ &= \overline{\overline{\overline{A1} \cdot \overline{A0} \cdot B1 \cdot \overline{B1 \cdot \overline{B0} \cdot \overline{A1}} \cdot \overline{\overline{A1} \cdot A0 \cdot \overline{B1} \cdot B0}} \\ &\quad \overline{\cdot \overline{A1 \cdot A0 \cdot B1 \cdot B0} \cdot \overline{A1 \cdot \overline{B1} \cdot \overline{B0}} \cdot \overline{A1 \cdot \overline{A0} \cdot \overline{B1}}} \end{aligned}$$

La implementación de R1 se muestra en la Figura 4.9.

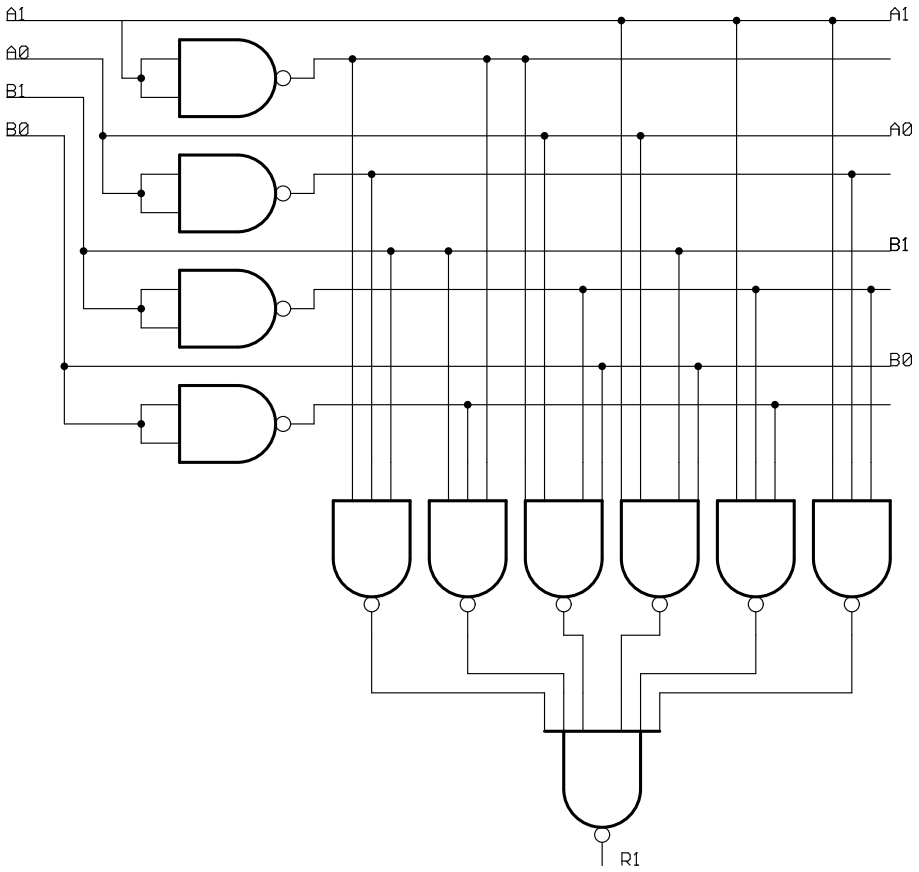


Figura 4.9: Implementación de R1 sólo con puertas NAND

La salida $R0$ puede expresarse como:

$$\begin{aligned} R0(A1, A0, B1, B0) &= \overline{A0} \cdot B0 + A0 \cdot \overline{B0} \\ &= \overline{\overline{\overline{A0} \cdot B0 + A0 \cdot \overline{B0}}} \\ &= \overline{\overline{\overline{A0} \cdot B0} \cdot \overline{A0 \cdot \overline{B0}}} \end{aligned}$$

La implementación de R0 se muestra en la Figura 4.10.

4.4. Circuitos combinacionales integrados (MSI)

Con el objetivo de simplificar la labor del diseñador de circuitos y de proporcionar una serie de dispositivos de amplia utilización e indudable utilidad, resulta habitual encontrar circuitos combinacionales que realizan funciones específicas y que están formados por unas decenas de puertas lógicas. La escala de integración utilizada para la construcción de estos dispositivos es la MSI (*Medium Scale Integration*) descrita en la sección anterior. Estos circuitos MSI incluyen, entre otros: decodificadores, codificadores, multiplexores y demultiplexores.

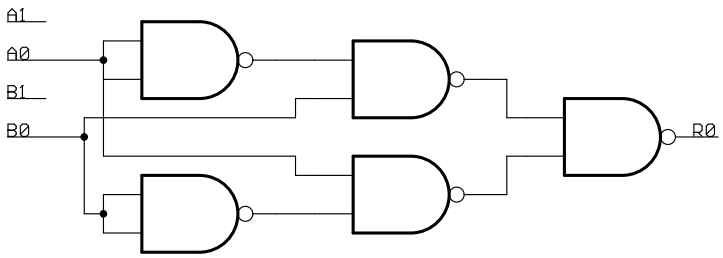


Figura 4.10: Implementación de R0 sólo con puertas NAND

Como estos dispositivos se fabrican comercialmente, están preparados para que puedan realizarse asociaciones de varios de ellos para formar un dispositivo más grande. Para ello, además de las entradas y salidas habituales, estos circuitos suelen poseer una entrada especial llamada de habilitación (*Enable*), que permite habilitar o no el circuito.

Decodificadores

Un circuito decodificador es aquel que, dado un código a su entrada, activa únicamente una de sus salidas, en concreto, aquella especificada por dicho código. Normalmente, el sistema utilizado para la codificación es el binario natural.

Puesto que un decodificador tiene n entradas y 2^n salidas, la notación habitualmente empleada para referirse al mismo es $n \times 2^n$ (así por ejemplo, 3×8, 1×2, 2×4, ...). La Figura 4.11 representa el esquema de un decodificador genérico $n \times 2^n$.

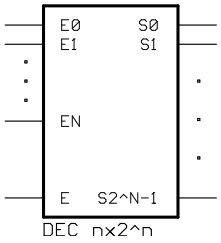


Figura 4.11: Decodificador $n \times 2^n$

Una de las aplicaciones más comunes de los decodificadores es la selección o activación de un dispositivo entre varios. Si se asigna un código binario a cada uno de los dispositivos y se conecta cada una de las salidas de un decodificador $n \times 2^n$ a la señal de habilitación (*Enable*) de cada uno de ellos, puede activarse o seleccionarse uno entre 2^n dispositivos. De esta forma, para activar uno de los dispositivos, basta con introducir en las entradas del decodificador el código de n bits correspondiente al dispositivo que se desea activar. Un ejemplo de este tipo de aplicación lo constituye la selección de dispositivos de memoria que se estudiará con mayor detalle en el Capítulo 6.

Al describir el funcionamiento del decodificador se ha dicho que éste *activa* una de sus salidas. Se podría dar por supuesto que *activar* significa colocar un "uno lógico" en la salida en cuestión, pero en realidad, la acción de activar implica diferenciar

la salida en cuestión del resto, por lo que activar podría ser tanto poner un "uno lógico" como un "cero lógico" sin más que poner en el resto de salidas el valor contrario. Cuando las salidas se activan colocando un "uno lógico" se dice que el decodificador tiene salidas activas a nivel alto. Por el contrario, cuando las salidas se activan colocando un "cero lógico", se dice que el decodificador tiene salidas activas a nivel bajo.

Para representar gráficamente si una señal es activa a nivel alto o bajo existen dos convenios: el primero, es el de complementar el nombre asociado a la señal (p.ej. $\overline{señal}$); el segundo, consiste en poner un pequeño círculo en cualquiera de los extremos de la línea que representa dicha señal.

Como se ha comentado anteriormente, casi todos los circuitos combinacionales integrados poseen una entrada de control (*Enable*) para habilitar el funcionamiento de dicho circuito. En el caso de un decodificador, cuando esta señal está *activa*, el circuito funciona según las especificaciones. En caso contrario, todas las salidas del circuito se desactivan. La señal *Enable* también puede ser activa a nivel alto o a nivel bajo, independientemente de que si las salidas del circuito en cuestión son activas a nivel alto o bajo.

De acuerdo a lo visto hasta ahora, el decodificar representado en la Figura 4.11 tiene: n entradas (`E0` – `EN`), 2^n salidas activas a nivel alto (`S0` – `S2^N-1`) y una entrada *Enable* (`E`) activa a nivel alto.

Para comprender mejor el funcionamiento de los decodificadores, se presentan a continuación la tabla de verdad de un decodificador 1×2 (se muestran varias combinaciones posibles de salidas y entrada *Enable* activas a nivel alto o bajo) y de un decodificador 3×8.

Decodificador 1×2 con salidas activas a nivel alto sin entrada *Enable*

La Figura 4.12 muestra la tabla de verdad de un decodificador 1×2 con salidas activas a nivel alto (no posee entrada *Enable*) y un circuito combinacional que lo implementa.

E0	S1	S0
0	0	1
1	1	0

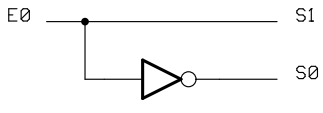


Figura 4.12: Tabla de verdad y circuito de un decodificador 1×2 con salidas activas a nivel alto

Decodificador 1×2 con salidas activas a nivel alto y entrada *Enable*

La Figura 4.13a representa la tabla de verdad de un decodificador 1×2 con salidas activas a nivel alto y entrada *Enable* a nivel alto. La Figura 4.13b representa la tabla de verdad del mismo decodificador pero con la entrada *Enable* a nivel bajo.

Decodificador 1×2 con salidas activas a nivel bajo y entrada *Enable*

La Figura 4.14a representa la tabla de verdad de un decodificador 1×2 con salidas activas a nivel bajo y entrada *Enable* a nivel alto. La Figura 4.14b representa la tabla de verdad del mismo decodificador pero con la entrada *Enable* a nivel bajo.

Decodificador 3×8 con salidas activas a nivel bajo y entrada *Enable*

Un ejemplo real de decodificador 3×8 lo constituye el encapsulado 74138 que posee salidas activas a nivel bajo y 3 entradas de habilitación. La Figura 4.15 re-

E	E0	S1	S0
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0

(a)

$\overline{E}$	E0	S1	S0
0	0	0	1
0	1	1	0
1	0	0	0
1	1	0	0

(b)

Figura 4.13: Tabla de verdad de un decodificador 1×2 con salidas activas a nivel alto y entrada *Enable* activa a nivel alto (a) o a nivel bajo (b)

E	E0	$\overline{S1}$	$\overline{S0}$
0	0	1	1
0	1	1	1
1	0	1	0
1	1	0	1

(a)

$\overline{E}$	E0	$\overline{S1}$	$\overline{S0}$
0	0	1	0
0	1	0	1
1	0	1	1
1	1	1	1

(b)

Figura 4.14: Tabla de verdad de un decodificador 1×2 con salidas activas a nivel bajo y entrada *Enable* activa a nivel alto (a) o a nivel bajo (b)

presenta la tabla de verdad y esquemático del mismo tal y como se presenta en las hojas de características del mismo.

Entradas					Salidas							
Enable		Selección										
G1	$\overline{G2}^*$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

H=Nivel alto (1), L=Nivel bajo (0), * $\overline{G2} = \overline{G2A} + \overline{G2B}$

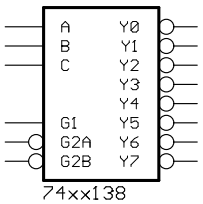


Figura 4.15: Tabla de verdad y esquemático del decodificador 3×8 74xx138

Generación de funciones lógicas utilizando decodificadores

Para cada una de las salidas (S_i) de un decodificador $n \times 2^n$, sólo existe una combinación de las entradas que haga que ésta se active, tomando el valor "uno

Salidas activas a nivel alto	$m_i = S_i$	$M_i = \overline{S_i}$
Salidas activas a nivel bajo	$m_i = \overline{S_i}$	$M_i = S_i$

Cuadro 4.2: Relación entre minitérminos y maxitérminos y las salidas de un decodificador

lógico" (si las salidas son activas a nivel alto) o "cero lógico" (si las salidas son activas a nivel bajo).

Debido a ello, cada función de salida (S_i) constituye un minitérmino o un maxitérmino de una función de n variables (donde n es el número de entradas del decodificador). En definitiva, si las salidas son activas a nivel alto, las 2^n salidas del decodificador representan directamente los 2^n minitérminos posibles ($S_i = m_i$) y si son activas a nivel bajo, las salidas representan directamente los 2^n maxitérminos posibles ($S_i = M_i$). La Tabla 4.2 muestra un resumen de la relación entre maxitérminos y minitérminos y las salidas de un decodificador dependiendo de si el decodificador tiene salidas activas a nivel alto o bajo.

Se ha visto en el capítulo anterior que el paso de la tabla de verdad de una función a una expresión algebraica se puede realizar tomando los unos de la función (resultando una suma de productos) o tomando los ceros de la función (resultando un producto de sumas). Teniendo en cuenta esta consideración, se puede generar cualquier función lógica de n variables utilizando un decodificador $n \times 2^n$ y una puerta lógica con menos de 2^n entradas. El tipo de puerta lógica a emplear dependerá del nivel de activación de las salidas del decodificador y de si se toman los unos o los ceros de la función. La Tabla 4.3 muestra el tipo de puerta que se utiliza en cada caso.

	Unos de la función	Ceros de la función
Salidas activas a nivel alto	$\mathrm{F} = \sum m_i = \sum S_i = \mathrm{OR}(S_i)$	$\mathrm{F} = \prod M_i = \prod \overline{S_i} = \overline{\sum Si} = \mathrm{NOR}(S_i)$
Salidas activas a nivel bajo	$\mathrm{F} = \sum m_i = \sum \overline{S_i} = \overline{\prod S_i} = \mathrm{NAND}(S_i)$	$\mathrm{F} = \prod M_i = \prod S_i = \mathrm{AND}(S_i)$

Cuadro 4.3: Combinaciones posibles de decodificadores y puertas lógicas para la implementación de funciones lógicas

De esta forma, es posible implementar una función de cuatro formas distintas. A continuación se muestran, a modo de ejemplo, las posibles implementaciones de una función F cuya tabla de verdad se muestra en la Tabla 4.4. Esta función, puede expresarse tomando los ceros de la misma como $\mathrm{F}(A, B, C) = \prod (0, 3, 5, 6)$, o bien, tomando los unos, como $\mathrm{F}(A, B, C) = \sum (1, 2, 4, 7)$.

Para implementar esta función utilizando un decodificador y una puerta lógica se debe elegir en primer lugar un decodificador con tantas entradas como variables tenga la función, es decir, puesto que la función es de tres variables, un decodificador 3×8; en segundo lugar se debe escoger la puerta lógica que se debe utilizar en función del nivel de activación de las salidas y de si se resuelve por unos o por ceros la función. La Figura 4.16 muestra las posibles soluciones (las entradas de habilitación $G1$, $G2A$ y $G2B$ funcionan tal y como se han descrito anteriormente para el decodificador 74138).

A	B	C	F(A,B,C)
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

Cuadro 4.4: Función lógica que se desea implementar con un decodificador y una puerta lógica

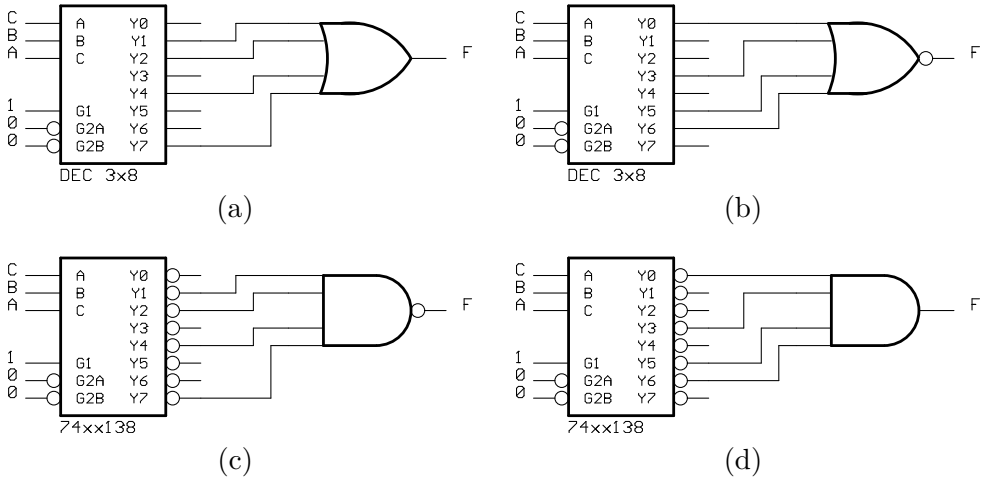


Figura 4.16: Implementación de la función F utilizando un decodificador con salidas activas a nivel alto y tomando los unos de la función (a) o tomando los ceros de la función (b); utilizando un decodificador con salidas activas a nivel bajo y tomando los unos de la función (c) o tomando los ceros de la función (d)

Asociación de Decodificadores

La asociación de decodificadores consiste en la construcción de un decodificador utilizando varios decodificadores de menor tamaño. El proceso es el siguiente:

1. Se utilizan tantos decodificadores como sean necesarios para suministrar todas las salidas del decodificador que se pretende diseñar. Las entradas de los mismos se conectan en paralelo a las entradas de menor peso del decodificador resultante.

2. Se emplea un decodificador que tenga como entradas las entradas de mayor peso no utilizadas hasta ahora del decodificador que se desea construir y las salidas de este último decodificador se conectan a las entradas *Enable* de los decodificadores definidos en el apartado anterior. La función de este decodificador será pues la de seleccionar a uno del resto de decodificadores. Como

entrada *Enable* del circuito final se puede utilizar la de este último decodificador.

Ejemplo 1: Diseñar un decodificador 2×4 utilizando sólo decodificadores 1×2

Se emplean dos decodificadores 1×2 para poder proporcionar las 4 salidas del decodificador resultante, conectando la entrada de cada decodificador a la entrada de control del decodificador 2×4 resultante de menor peso. También se utiliza otro decodificador para que habilite sólo uno de los dos anteriores. La entrada de este decodificador será la entrada de control de mayor peso del decodificador 2×4. La Figura 4.17 muestra el circuito resultante.

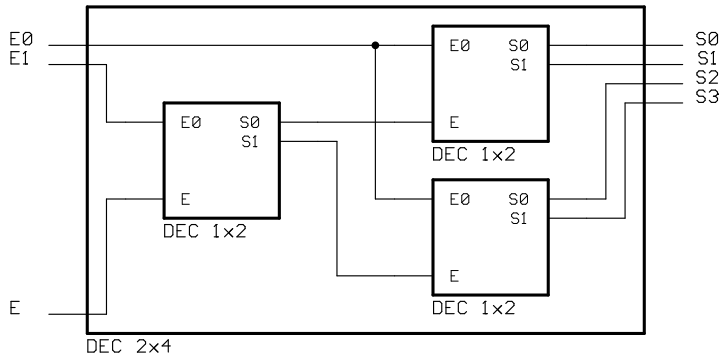


Figura 4.17: Obtención de un decodificador 2×4 utilizando decodificadores 1×2

Ejemplo 2: Diseñar un decodificador 3×8 utilizando decodificadores 1×2 y 2×4

La Figura 4.18 muestra una posible solución.

También se podría construir un decodificador 3×8 utilizando sólo decodificadores 1×2. Esta asociación es similar a la anterior, consiste en la sustitución del decodificador 2×4 de la Figura 4.18 por una asociación de decodificadores 1×2 construida como en el ejemplo anterior.

Codificadores

Un codificador realiza la función inversa al decodificador. En sus entradas tiene información no codificada y en sus salidas presenta la misma información pero codificada. La ventaja de codificar la información estriba en que la misma información se expresa de una forma más compacta (es decir, utilizando menos bits). Por ejemplo, en el caso de un teclado de 32 teclas, el código de la tecla pulsada se puede representar con 5 bits en lugar de utilizar las 32 líneas asociadas a cada una de las teclas.

Los codificadores se nombran con el número de entradas, dos puntos y el número de salidas que poseen. Habitualmente, tienen un número de entradas que es potencia exacta de dos por lo cual la notación será $2^n : n$, siendo n el número de salidas que posee el codificador (número mínimo de bits que son necesarios para expresar el número de orden de una de sus entradas).

El código empleado habitualmente por los codificadores es el binario natural. Es decir, suponiendo que sólo se activa una entrada en cada instante de tiempo, la salida del codificador será el código binario asociado a la entrada que ha sido

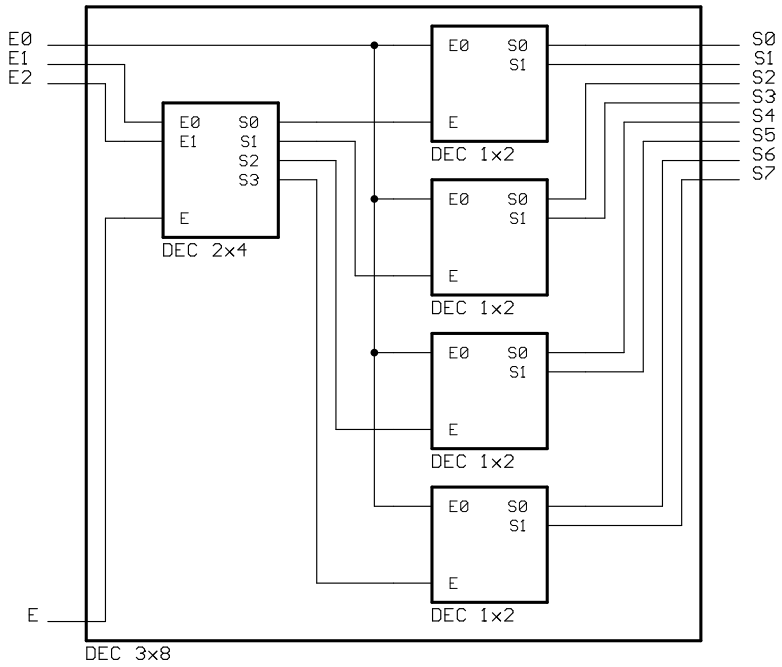


Figura 4.18: Obtención de un decodificador 3×8 utilizando decodificadores 1×2 y 2×4

activada. Por ejemplo, si en un codificador $8:3$, se activa la entrada 6, en la salida se tendrá el número 6 en binario natural (110) (véase Figura 4.19).

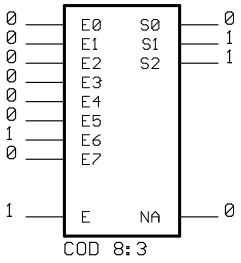


Figura 4.19: Codificador $8:3$ con la entrada 6 activa

Como se puede observar, un codificador posee, además de las salidas de datos, una salida adicional, NA. Esta salida tiene como función indicar si ninguna de las entradas está activa en un momento dado (es decir, cuando ninguna entrada esté activa, se activará la salida NA).

Codificador sin prioridad

La definición de codificador dada hasta ahora, corresponde, en realidad, a la de un **codificador sin prioridad**. Éstos proporcionan en sus salidas, funciones incompletas que sólo están definidas cuando está activa una o ninguna de las entradas en un mismo instante de tiempo. O lo que es lo mismo, las salidas de datos están indefinidas cuando dos o más entradas están activas simultáneamente.

En detalle, el comportamiento de un codificador sin prioridad es el siguiente: si ninguna entrada está activa se activa la salida NA y el resto de salidas están indefinidas (este es el comportamiento teórico aunque es usual en la práctica asignarles para ese caso el valor "cero lógico"). Si se activa una de sus entradas, las salidas adoptarán el valor correspondiente al código de la misma y la salida NA tomará el valor "cero lógico'. La Tabla 4.5 muestra la tabla de verdad de un codificador sin prioridad 4 : 2.

E_3	E_2	E_1	E_0	$S1$	$S0$	NA
0	0	0	0	x	x	1
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
1	0	0	0	1	1	0

Cuadro 4.5: Tabla de verdad de un codificador 4 : 2 sin prioridad

Codificador con prioridad

Los codificadores sin prioridad tienen poca utilidad en la práctica debido a que es bastante difícil asegurar que sólo se active una única entrada en un instante dado. Los *codificadores con prioridad* definen funciones completas que contemplan aquellos casos en los que se activan varias entradas al mismo tiempo. Normalmente, el valor que se codifica en la salida cuando varias entradas se activan al mismo tiempo es el de la entrada activa de mayor peso (de ahí el nombre de con prioridad).

Por ejemplo, un codificador 4 : 2 con prioridad funciona de la siguiente forma: si la entrada 3 (la más prioritaria) está activa, independientemente de si otra entrada está activa, las salidas codifican el valor 3. Para que las salidas muestren el valor 2 la entrada 3 no debe estar activa y la entrada 2 sí, sin importar entonces, si las entradas 1 y 0 están o no activas. Y así sucesivamente. La Tabla 4.6 muestra la tabla de verdad de un codificador 4 : 2 con prioridad. (Una 'x' en la especificación de una entrada se utiliza para indicar que la salida es la misma para cualquier valor de esa entrada.)

E_3	E_2	E_1	E_0	$S1$	$S0$	NA
0	0	0	0	x	x	1
0	0	0	1	0	0	0
0	0	1	x	0	1	0
0	1	x	x	1	0	0
1	x	x	x	1	1	0

Cuadro 4.6: Tabla de verdad de un codificador 4 : 2 con prioridad

Multiplexores

Un multiplexor es un dispositivo que posee una sola salida, 2^n entradas de datos y n entradas de control. Este dispositivo *traslada* a su única salida el valor lógico

presente en la entrada cuyo código se encuentra en las entradas de control. Este comportamiento permite emplear los multiplexores como selectores, donde el valor presente en las entradas de control permite seleccionar la entrada de datos cuyo valor se desea transferir a la salida.

El esquema, por tanto, será el mostrado en la Figura 4.20.

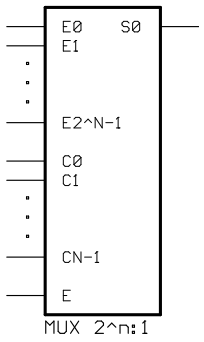


Figura 4.20: Multiplexor

Un multiplexor puede utilizarse, por ejemplo, cuando una impresora se comparte por varios ordenadores. Los ordenadores se conectarían a las entradas del multiplexor y la impresora a la salida del mismo. De esta forma, la combinación presente en las entradas de control determinan qué ordenador envía datos a la impresora en un momento dado.

Asociación de multiplexores

La asociación de multiplexores consiste en la construcción de un multiplexor utilizando multiplexores de menor tamaño (menor número de entradas). Para realizar esta asociación se necesita un potencia exacta de dos de multiplexores iguales entre sí más uno con tantas entradas como multiplexores iguales hay en el grupo anterior, es decir, de 2^n entradas.

Las entradas de datos de los 2^n multiplexores iguales constituirán las entradas de datos del multiplexor resultante. Las entradas de control de todos estos multiplexores se conectarán en paralelo entre sí y constituirán las entradas de control de menor peso del multiplexor que estamos construyendo.

Se empleará el multiplexor de 2^n entradas de datos para seleccionar una salida de entre todas las de los 2^n multiplexores anteriores. Dichas salidas se conectarán en orden a las entradas de datos de este multiplexor. Es decir, la salida del multiplexor que tenga la entrada principal 0, se conectará a la entrada 0, y la salida del multiplexor que tenga la entrada principal de mayor peso, se conectará a la entrada de mayor peso de este multiplexor. Las entradas de control de este multiplexor serán las entradas de control de mayor peso del multiplexor resultante.

La Figura 4.21 muestra un ejemplo de construcción de un multiplexor $4:1$ mediante la asociación de multiplexores $2:1$.

Generación de funciones lógicas con multiplexores

Existen dos formas para generar cualquier función lógica de n variables con multiplexores:

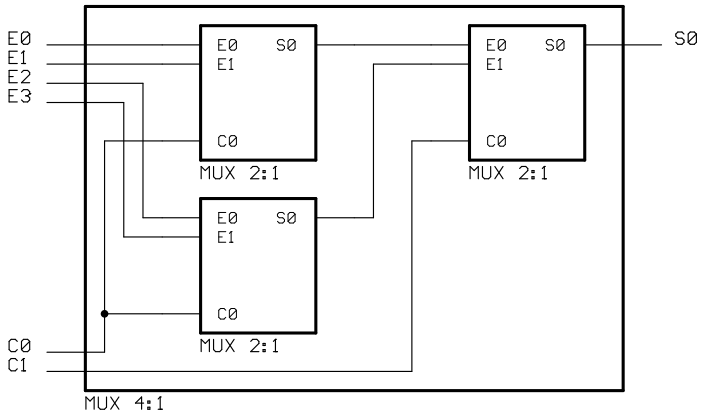


Figura 4.21: Ejemplo de asociación de multiplexores

- Utilizando un multiplexor $2^n : 1$.
- Utilizando un multiplexor $2^{n-1} : 1$ y un inversor.

La segunda opción es más barata, ya que requiere un multiplexor de menor tamaño.

Para llevar a cabo la generación mediante primera forma se procede como sigue: se conectan las entradas correspondientes a las variables de la función a las entradas de control del multiplexor y en las entradas de datos se introducen los correspondientes valores constantes (0 ó 1) de la tabla de verdad de la función. El funcionamiento consiste en que cada combinación de las entradas de control selecciona la entrada correspondiente del multiplexor. Si en dicha entrada está presente el valor de salida de la función para la combinación de entrada seleccionada en las entradas de control en ese instante, la salida del multiplexor es directamente la función que se deseaba implementar.

La generación mediante la segunda forma tiene una filosofía similar. En lugar de tener valores constantes en las entradas del multiplexor, se tienen funciones de una sola variable. El coste máximo de una función de una variable es un inversor ya que tan sólo existen cuatro posibles funciones de una variable: siempre 0, siempre 1, valor igual al de entrada y valor complementario al de entrada. Resumiendo:

x	$f(x)=0$	$f(x)=1$	$f(x)=x$	$f(x)=\bar{x}$
0	0	1	0	1
1	0	1	1	0

La implementación mediante esta segunda forma consistirá en:

1. Conectar $n-1$ variables de entrada a las entradas de control. Esto significa que se ha de elegir una de las variables que no se conectará a las entradas de control del multiplexor. A esta variable le llamaremos x para simplificar la descripción del método. La elección de esa variable la consideraremos arbitraria, es decir, no se van considerar argumentos para elegir una u otra.

2. Obtener las 2^{n-1} funciones dependientes de x. Para ello, se confecciona una tabla de verdad de la función en donde la dependencia de la variable x se expresa en la columna de los valores de salida. Para cada combinación de las $n-1$ variables del apartado anterior, x puede tomar los valores 0 o 1. A su vez, los posibles valores de F para esas dos combinaciones de entrada pueden ser cuatro (0, 1, x, $\bar{x}$). Así pues, la tabla de verdad resultante tendrá $n-1$ variables de entrada y su salida tomará valores entre los cuatro posibles anteriores.

3. Conectar las funciones anteriores a las entradas correspondientes del multiplexor. Las 2^{n-1} funciones de una variable x obtenidas en la tabla de verdad del apartado anterior serán la que se deben conectar, respetando el orden correspondiente, en la entradas del multiplexor 2^{n-1} : 1 empleado para la implementación.

A continuación, se ilustran ambos métodos de implementación de una función de n variables empleando multiplexores mediante un ejemplo. Sea la función F:

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

El primer método consiste en copiar los valores de la tabla de verdad en las entradas del multiplexor y conectar en las entradas de control las variables. Hay que tener en cuenta los pesos de las variables de control y su relación con el orden de las entradas. Así, el circuito resultante es el mostrado en la Figura 4.22.

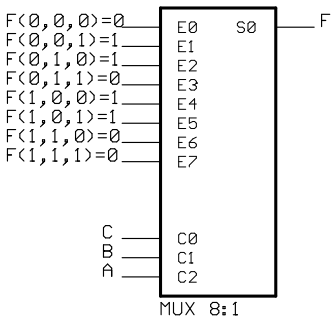


Figura 4.22: Implementación de una función de 3 variables utilizando un multiplexor 8 : 1

A continuación, se muestra la resolución utilizando el segundo método. Primero se elige una de las tres variables como independiente. Si la elección recae sobre la variable C, la tabla de verdad en función de C queda:

A	B	$C=0$	$C=1$	F
0	0	0	1	C
0	1	1	0	$\overline{C}$
1	0	1	1	1
1	1	0	0	0

El circuito resultante será, por tanto, el mostrado en la Figura 4.23.

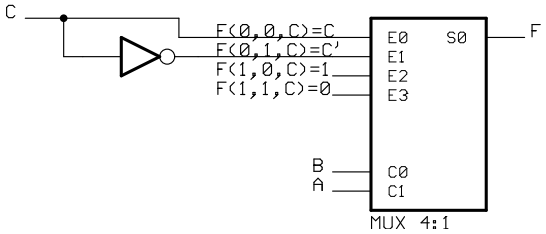


Figura 4.23: Implementación de una función de 3 variables utilizando un multiplexor 4 : 1 y como máximo un inversor

Demultiplexores

Un demultiplexor realiza la función inversa al multiplexor. Éstos son de utilidad por ejemplo en el caso de disponer de un único ordenador y varias impresoras de diferentes características. Un demultiplexor permitiría la selección de una de las impresoras conectando la salida del ordenador con la entrada de la impresora correspondiente. Las entradas de control del demultiplexor determinarían qué impresora está conectada en un momento dado.

Un circuito combinacional que realice la función de demultiplexor será aquel que tenga n entradas de selección o de control, una entrada a demultiplexar y 2^n salidas. Las entradas de control seleccionan una salida de forma que se le asignará el mismo valor que haya presente en la entrada en ese instante manteniendo desactivadas (valor lógico cero) el resto de salidas. Así, el esquema de un demultiplexor es el mostrado en la Figura 4.24.

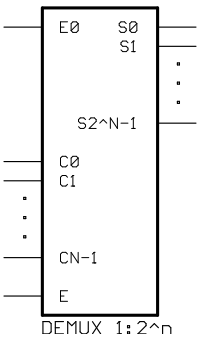


Figura 4.24: Demultiplexor $1 : 2^n$

A continuación, se compara el comportamiento de un demultiplexor con el de un decodificador con salidas activas a nivel alto y *Enable* activa a nivel alto con

el objetivo de demostrar la equivalencia entre ambos cuando se utiliza la entrada *Enable* del decodificador como la entrada a demultiplexar.

Primero se estudia el caso de que la entrada del demultiplexor valga 0. Los valores de las salidas del demultiplexor serán todos 0, ya que todas las salidas excepto la seleccionada valen 0 y el valor de la salida seleccionada es el mismo que el de la entrada, que en este caso vale 0. En un decodificador como el propuesto, si la entrada de habilitación está desactivada (vale 0) todas las salidas están desactivadas (valen 0). Es evidente, pues, que ambos funcionan de la misma forma para este caso.

En segundo lugar, qué pasa cuando la entrada del demultiplexor vale 1, sus salidas valen entonces todas 0 salvo la seleccionada que vale 1 (el valor presente en la entrada). En un decodificador como el propuesto, si la entrada de habilitación está activada (vale 1), entonces todas las salidas están desactivadas (valen 0) excepto la seleccionada por la entradas de control, que valdrá 1. También en este caso, el comportamiento del demultiplexor coincide con el del decodificador, con lo cual se ha demostrado que ambos dispositivos son equivalentes.

CAPÍTULO 5

Circuitos secuenciales

5.1. Introducción

La característica más importante de un circuito secuencial es su capacidad de memorizar información. De esta forma, un circuito secuencial modifica su comportamiento (salidas) ante un conjunto de estímulos (entradas) teniendo en cuenta situaciones anteriores (estados), al contrario que, en los circuitos combinacionales donde el valor de las salidas depende exclusivamente del valor de sus entradas en ese instante.

En un circuito secuencial se pueden encontrar dos tipos de funciones: las *funciones de salida* y las *funciones de transición*, ambas dependientes tanto de las entradas como del estado actual del circuito. Las funciones de transición son aquellas que permiten determinar el estado futuro del circuito. Dicho de otra forma, las funciones de transición definen el estado futuro dependiendo de las entradas en ese instante y del estado actual.

Las funciones de salida de un circuito combinacional se expresan de una de las siguientes formas:

$$\begin{aligned} S_i(t) &= F_i\left(E_1(t), \ldots, E_n(t), Q_1(t), \ldots, Q_p(t)\right), \forall i \in [1..m], \text{o} \\ S_i(t+1) &= F_i\left(E_1(t), \ldots, E_n(t), Q_1(t), \ldots, Q_p(t)\right), \forall i \in [1..m] \end{aligned}$$

Y las funciones de transición como:

$$Q_i(t+1) = F_i\left(E_1(t), \ldots, E_n(t), Q_1(t), \ldots, Q_p(t)\right), \forall i \in [1..p]$$

Estructuralmente, un circuito secuencial está definido por un circuito combinacional que determina todas las funciones (salidas y de transición) y un circuito de memoria que permite almacenar información. Este circuito de memoria está formado por una composición de circuitos básicos capaces de almacenar un bit cada uno de ellos. Estos circuitos básicos reciben el nombre de biestables y se estudian en el siguiente apartado.

Los conceptos de estado actual y estado futuro, hacen referencia al estado del circuito secuencial en un instante de tiempo arbitrario al que llamaremos t y al estado del circuito en el siguiente instante de tiempo $t+1$, respectivamente. Según cómo se distinga el instante t del $t+1$ se puede hablar de dos tipos de circuitos secuenciales: asíncronos y síncronos. La diferencia entre ambos estriba en cuándo se puede modificar el contenido de la memoria.

Circuito secuencial asíncrono. El estado actual puede cambiar cuando se produce un cambio en las entradas del circuito.

Circuito secuencial síncrono. Estos circuitos utilizan una señal denominada *de reloj* que marca su pauta de trabajo. La señal de reloj es una señal periódica de periodo T, es decir, los valores que toma durante un tiempo T se repiten en el siguiente periodo de tiempo y así indefinidamente. En concreto, el valor de la señal cambia de valor (de 0 a 1 o viceversa) cada $T/2$ (véase Figura 5.1). Un circuito secuencial síncrono utiliza un determinado cambio en el valor de esta señal para marcar el instante en el cual se puede cambiar de estado. Si el cambio de nivel utilizado para sincronizar el circuito es de 1 a 0, el circuito secuencial se dice que es *activo por flanco de bajada*, y si es de 0 a 1, se dice que es *activo por flanco de subida*.

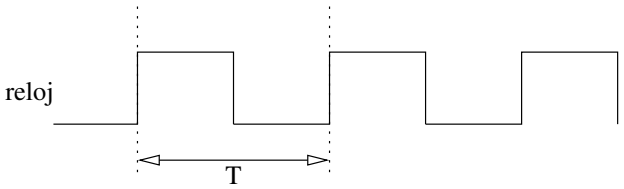


Figura 5.1: Señal de reloj

En definitiva, el esquema básico de un circuito secuencial síncrono es el presentado en la Figura 5.2.

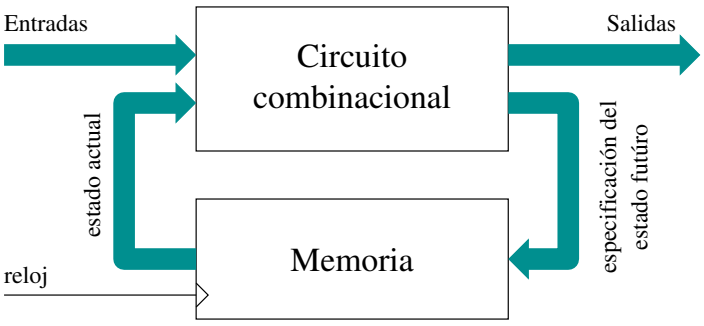


Figura 5.2: Esquema básico de un circuito secuencial síncrono

Este capítulo se centra en el estudio de los circuitos secuenciales síncronos.

5.2. Biestables

Un biestable es el circuito secuencial más elemental que existe. Sólo puede tener dos estados: 0 (almacena el valor 0) ó 1 (almacena el valor 1). La función de salida de un biestable, por lo tanto, equivale al estado almacenado. Como se ha comentado anteriormente, los biestables son los componentes básicos del circuito de memoria de los circuitos secuenciales.

Existen cuatros tipos de biestables estándar: RS, JK, T y D. El nombre de cada biestable se corresponde con las entradas que posee: tanto los biestables RS como los JK poseen dos entradas, mientras que los T y D solamente cuentan con una. En las siguientes secciones se detallan las características funcionales de cada uno de ellos.

Biestable RS

Posee dos entradas: R, inicial de *Reset* que significa poner a cero, y S, inicial de *Set* que significa poner a uno. Su funcionamiento, por lo tanto, será el siguiente: cuando se ponga a 1 la entrada R, el valor almacenado en el biestable RS pasará a ser 0, y cuando se ponga a 1 la entrada S, el valor almacenado pasará a ser 1. Mientras las dos entradas permanezcan a 0, el valor almacenado en el biestable se mantendrá. Por último, si las dos entradas se ponen a 1 simultáneamente, el valor de la salida es inestable y además puede ser que el biestable deje de funcionar correctamente a partir de entonces. En definitiva, se define la función de transición del biestable RS como la mostrada en la Tabla 5.1.

R	S	$Q(t)$	$Q(t+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	?
1	1	1	?

Cuadro 5.1: Tabla de transiciones de un biestable RS

Para expresar el comportamiento de un biestable, normalmente se reescribe la anterior tabla de transiciones para convertirla en la denominada tabla de funcionamiento. Esta tabla indica para cada combinación de las entradas, el valor del estado futuro $Q(t+1)$ en función del estado actual $Q(t)$. La Tabla 5.2 muestra la tabla de funcionamiento de un biestable RS.

Biestable RS asíncrono

El biestable RS es el único tipo de biestables que puede funcionar como circuito secuencial asíncrono. La Figura 5.3 muestra una implementación de un biestable RS asíncrono utilizando puertas NOR y la Figura 5.4 presenta el símbolo utilizado para representar a este biestable. Como se puede observar en sendas figuras, aunque

R	S	$Q(t+1)$
0	0	$Q(t)$
0	1	1
1	0	0
1	1	X

Cuadro 5.2: Tabla de funcionamiento de un biestable RS

la salida que indica el estado del biestable es la etiquetada como Q, generalmente también se suele proporcionar la salida complementada $\overline{Q}$ (referenciada como `Q'`). La salida de este biestable (su estado) cambiará en el instante en que cambien sus entradas.

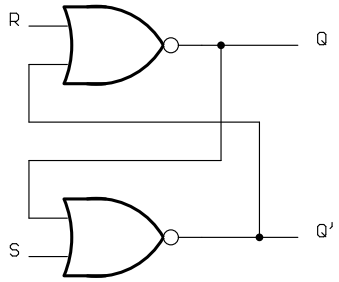


Figura 5.3: Esquema de un biestable RS asíncrono con puertas NOR

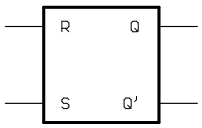
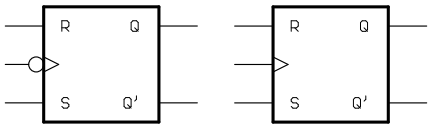


Figura 5.4: Símbolo de un biestable RS asíncrono

Biestable RS síncrono

El biestable RS síncrono se distingue del asíncrono en que utiliza una señal de reloj para determinar cuándo se puede producir la transición del estado actual al siguiente. Dependiendo de si el biestable es activo por flanco de bajada o de subida, el instante en el que puede cambiar, es aquel en el que la señal de reloj pasa de 1 a 0 o de 0 a 1, respectivamente.

La Figuras 5.5a y 5.5b muestran los símbolos utilizados para representar los biestables RS síncronos activos por flanco de bajada y de subida, respectivamente. Como se puede observar en dichas figuras, la entrada de reloj se marca con un pequeño triángulo y el flanco de activación se indica de la siguiente forma: con un pequeño circulo en la entrada de la señal de reloj cuando es por flanco de bajada, y sin nada, cuando es de subida.



(a) (b)

Figura 5.5: Símbolos de un biestable RS síncrono activo por flanco de bajada (a) y por flanco de subida (b)

Biestable JK

El funcionamiento del biestable JK es similar al biestable RS (considerando $J \equiv S$ y $K \equiv R$) pero define la función de transición para la combinación de entrada $J = K = 1$ como el valor complementario del estado actual. La Figura 5.6 muestra la tabla de funcionamiento de un biestable JK y el símbolo empleado para su representación.

J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

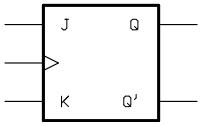


Figura 5.6: Tabla de funcionamiento y símbolo de un biestable JK síncrono activo por flanco de subida

No existe una versión asíncrona del biestable JK debido a que, sin un mecanismo de sincronización, la salida para la combinación de entradas $J = K = 1$ es inestable. Esta inestabilidad consiste en que el tiempo durante el cual se mantiene el mismo valor a la salida del biestable no está definido por la tabla de funcionamiento del mismo, sino por su implementación física (es decir, depende del tiempo de retardo de sus componentes internos).

En la Figura 5.7 se ilustra cómo sería el funcionamiento de un hipotético biestable JK asíncrono. Como se puede observar en dicho cronograma, cuando las entradas del biestable valen simultáneamente 1, la salida del biestable (el valor de su estado) varía constantemente entre 0 y 1 y viceversa, dependiendo el periodo de dicha variación únicamente del tiempo de propagación de los componentes que formen dicho biestable. Puesto que esto implica que no se puede conocer el estado actual del biestable en un momento dado, este biestable solamente tiene sentido como biestable síncrono.

La Figura 5.8 presenta un cronograma de un biestable JK síncrono activo por flanco de subida en el que las entradas J y K presentan el mismo comportamiento que en el cronograma anterior. Como se puede observar, la variación de la salida está gobernada por los flancos de subida de la señal de reloj (en particular, cuando $J = K = 1$ éstos determinan el tiempo de permanencia en cada estado).

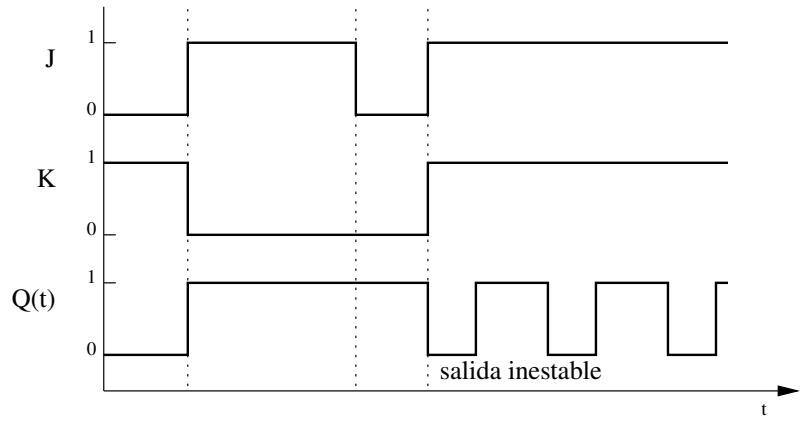


Figura 5.7: Cronograma de un hipotético biestable JK asíncrono

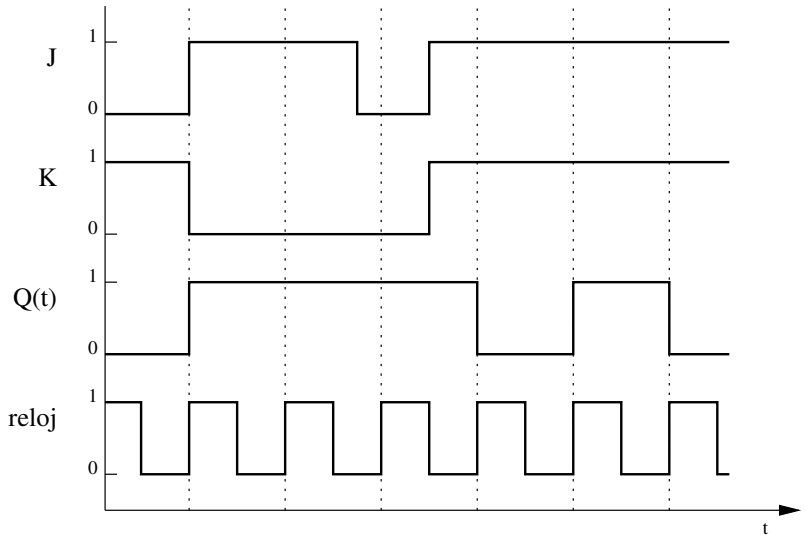


Figura 5.8: Cronograma de un biestable JK síncrono activo por flanco de subida

Biestable T

El biestable T, al igual que el biestable D que se verá en la siguiente sección, posee una única entrada. El nombre de este biestable viene de la palabra inglesa *Toggle* que significa *conmutar*. Su funcionamiento cuando su entrada vale 1 será justamente el de conmutar el valor del estado actual (de 0 a 1 o viceversa), cuando su entrada vale cero el valor del estado actual se mantiene. La Figura 5.9 muestra la tabla de funcionamiento de un biestable T y el símbolo utilizado para su representación.

Se puede entender el biestable tipo T como una variante del biestable JK, ya que si la entrada T se conecta a las entradas J y K (T=J=K), dicho circuito se comporta como un biestable tipo T, como se puede ver en la Figura 5.10.

T	$Q(t+1)$
0	$Q(t)$
1	$\overline{Q(t)}$

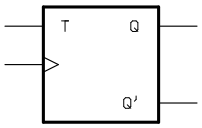


Figura 5.9: Tabla de funcionamiento y símbolo de un biestable T síncrono activo por flanco de subida

T	$Q(t+1)$	J	K
0	$Q(t)$	0	0
1	$\overline{Q(t)}$	1	1

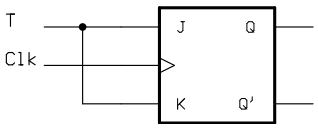


Figura 5.10: Implementación de un biestable T utilizando un JK

Por la misma razón que la del biestable JK, no tiene sentido una versión asíncrona del biestable T.

Biestable D

El nombre de este biestable viene de la palabra inglesa *Delay* que significa *retardo*. Su funcionamiento como síncrono consiste en colocar en su salida el valor de la entrada que había en el último flanco de la señal de reloj. La Figura 5.11 muestra la tabla de funcionamiento de un biestable D y el símbolo utilizado para su representación.

D	$Q(t+1)$
0	0
1	1

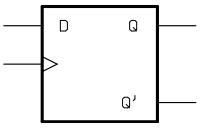


Figura 5.11: Tabla de funcionamiento y símbolo de un biestable D activo por flanco de subida

El biestable D asíncrono no existe puesto que su funcionamiento consistiría en tener en todo momento en su salida el mismo valor que hubiera en su entrada. Es decir, un biestable D asíncrono sería igual a un segmento de conductor o a un *driver* por lo que no tiene sentido como tal.

Este tipo de biestable es el más simple que existe, y representa el concepto más puro de memoria. De hecho, se tomará el biestable D como el circuito más típico para implementar la memoria de los circuitos secuenciales.

Un biestable D se puede construir utilizando un biestable tipo JK o RS y una puerta inversora. Así, para realizar un biestable D con un JK, se conecta la entrada D directamente a J y a través de un inversor a K, como se puede observar en la Figura 5.12.

La implementación de un biestable D utilizando un RS, es similar a la realizada con un JK. La entrada D se conecta directamente a S y a través de un inversor a R (véase Figura 5.13).

D	$Q(t+1)$	J	K
0	0	0	1
1	1	1	0

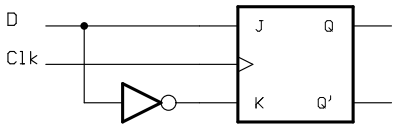


Figura 5.12: Implementación de un biestable D utilizando un JK y una puerta inversora

D	$Q(t+1)$	R	S
0	0	1	0
1	1	0	1

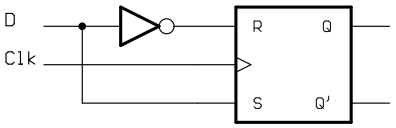


Figura 5.13: Implementación de un biestable D utilizando un RS y una puerta inversora

Entradas asíncronas: *Preset* y *Reset*

En los circuitos secuenciales síncronos existe la posibilidad de tener entradas asíncronas. Estas entradas funcionan independientemente de la señal de reloj, es decir, cuando está activa una entrada asíncrona afecta inmediatamente al valor del biestable y anula el efecto de las entradas síncronas. Existen dos posibles entradas asíncronas: *Preset* (P) y *Reset* (R). La Figura 5.14 muestra un biestable JK con señales *Preset* y *Reset*.

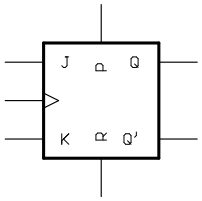


Figura 5.14: Biestable JK con entradas *Preset* y *Reset*

En cuanto se activa la entrada P, el estado actual del biestable pasa a ser 1 independientemente de la señal de reloj y de las entradas del biestable. De forma similar, en cuanto se activa la entrada R, el estado actual del biestable pasa a ser 0 de forma inmediata. Si ambas entradas se activan simultáneamente, el estado del biestable está indeterminado.

Una aplicación usual de estas entradas es la fijación del estado inicial.

5.3. Descripción funcional de un circuito secuencial

La descripción funcional de un circuito, ya sea secuencial como combinacional, se puede realizar, tal y como se ha visto en el capítulo anterior, mediante la especificación de las funciones lógicas que éste implementa. En el caso de un circuito secuencial, su descripción funcional se obtiene mediante la especificación de las funciones de salida y de transición.

Puesto que la interpretación de la tabla de verdad de un circuito secuencial no es obvia, se utilizan otras descripciones adicionales para representar el funcionamiento de un circuito secuencial:

1. Descripción de la máquina de Mealy o Moore.
2. Descripción de la tabla de estados.
3. Descripción de la tabla de transiciones.
4. Descripción de la tabla de funcionamiento.

La primera descripción resulta más intuitiva desde el punto de vista humano, mientras que la última está directamente relacionada con la implementación del circuito secuencial. De esta forma, cuando se desea analizar un circuito secuencial, primero se tiene que obtener la cuarta descripción, luego la tercera, la segunda y por último la primera. Por el contrario, cuando lo que se desea es diseñar un circuito secuencial partiendo de unas especificaciones, primero se tiene que obtener la primera, luego la segunda, la tercera y por último la cuarta.

Las siguientes secciones introducen teóricamente cada una de estas descripciones para después poder realizar el análisis y síntesis de varios circuitos secuenciales.

Máquinas de Moore/Mealy

Las máquinas de Moore y de Mealy describen gráficamente el funcionamiento de los circuitos secuenciales síncronos. La máquina de Moore se diferencia de la de Mealy en la forma en la que se especifican las funciones de salida.

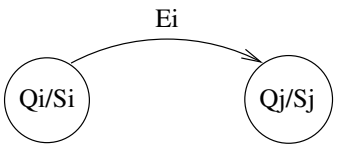
La descripción de la máquina de estados se basa en grafos dirigidos. Un grafo dirigido está constituido por nodos y por enlaces etiquetados que van de un nodo a otro. Un nodo se representa con un círculo, y un enlace se representa con una flecha que parte de un nodo y acaba en otro.

Los nodos representan los estados posibles del circuito. A los nodos se les dan nombres para diferenciar unos nodos de otros. Este nombre, por regla general, es una palabra o frase corta que indica el estado en el que se encuentra la máquina. En descripciones posteriores este nombre se codificará en una combinación particular de 0s y 1s.

Los enlaces representan las funciones de transición entre estados y cada uno de ellos se etiqueta con la combinación de entradas que provocan la transición que éste representa.

La interpretación de una máquina de estados es la siguiente: en un instante t, la máquina se encuentra en un estado $Q(t) = Q_i$ (representado por su correspondiente nodo), en el siguiente instante de tiempo $t + 1$, la máquina cambia a otro estado $Q(t + 1)$ (que puede ser igual al anterior), este nuevo estado $Q(t + 1) = Q_j$ se corresponde con el nodo al que llega el enlace que parte del nodo Q_i y está etiquetado con el valor actual de las entradas.

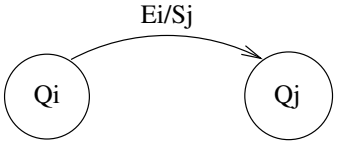
En la máquina de Moore el valor de la salida se indica en cada nodo y representa el valor que tiene la salida cuando se está en el estado representado por ese nodo. De esta forma, los nodos de una máquina de Moore se etiquetan como Q_i/S_i (estado en el instante t y salida en el instante t). La Figura 5.15 representa una transición entre estados en la máquina de Moore.



$E_i = E(t)$
$Q_i = Q(t)$
$S_i = S(t)$
$Q_j = Q(t+1)$
$S_j = S(t+1)$

Figura 5.15: Transición entre estados en la máquina de Moore

En la máquina de Mealy, a diferencia de la de Moore, el valor de la salida se indica con una etiqueta en el enlace entre dos nodos y representa el valor que tendrá la salida cuando se llegue al nuevo estado, es decir, $S(t+1)$. De esta forma, los enlaces en una máquina de Mealy se etiquetan como E_i/S_j (entrada en el instante t y salida en el instante $t+1$). La Figura 5.16 representa una transición entre estados en la máquina de Mealy.



$E_i = E(t)$
$Q_i = Q(t)$
$Q_j = Q(t+1)$
$S_j = S(t+1)$

Figura 5.16: Transición entre estados en la máquina de Mealy

Tabla de estados

La *tabla de estados* es una transcripción de la máquina en forma de tabla. En ella, se expresa cuáles serán el estado y las salidas futuras para cada combinación de los valores de entrada partiendo de cada posible estado actual. Existen dos formatos de tabla de estados posibles, dependiendo de si la máquina de partida es de Moore o de Mealy.

La tabla de estados de la máquina de Moore se confecciona de la siguiente forma: cada columna se identifica con una de las combinaciones de las entradas principales y cada fila con uno de los estados posibles (nodos) de la máquina de Moore y el valor de salida correspondiente a ese estado (separados por una /). Cada celda del interior de la tabla consigna el estado al que evolucionará la máquina en el instante $t+1$ cuando la máquina se encuentre en el estado indicado por su fila y se produzca la combinación de entrada indicada por su columna (véase Figura 5.17).

Existe una definición alternativa en la que en el interior de la tabla, cada vez que se referencia el estado futuro, se consigna también la salida correspondiente a dicho estado (véase Figura 5.18).

La tabla de estados de la máquina de Mealy se confecciona de la siguiente forma: cada columna se identifica con una de las combinaciones de las entradas principales y cada fila con uno de los estados posibles (nodos) de la máquina de Mealy. Cada celda del interior de la tabla consigna el estado al que evolucionará la máquina en el instante $t+1$ y la salida que proporcionará la máquina (separados por una /)

Estados\Entradas	$\cdots$	E_i	$\cdots$
$\vdots$		$\downarrow$	
Q_i/S_i	$\rightarrow$	Q_j	
$\vdots$			

Figura 5.17: Tabla de estados de una máquina de Moore

Estados\Entradas	$\cdots$	E_i	$\cdots$
$\vdots$		$\downarrow$	
Q_i/S_i	$\rightarrow$	Q_j/S_j	
$\vdots$			

Figura 5.18: Tabla de estados alternativa de una máquina de Moore

cuando estando en el estado actual (indicado por su fila) se produzca la combinación de entrada indicada por su columna (véase Figura 5.19).

Estados\Entradas	$\cdots$	E_i	$\cdots$
$\vdots$		$\downarrow$	
Q_i	$\rightarrow$	Q_j/S_j	
$\vdots$			

Figura 5.19: Tabla de estados de una máquina de Mealy

Tabla de transiciones

La *tabla de transiciones* se obtiene, partiendo de la tabla de estados, al codificar en binario los posibles estados (hasta ahora, cada estado tenía un nombre que lo identificaba). Dependiendo de la codificación empleada, el circuito final puede ser más simple o más fiable. No se van a tratar aquí los distintos tipos de codificación existentes y simplemente se va a procurar que la codificación elegida utilice el menor

número de bits posibles. Esto es interesante, ya que cada uno de los bits utilizados para indicar el estado se almacena en un biestable, por lo que a menor número de bits, menos biestables serán necesarios para la implementación del circuito secuencial.

El nombre de tabla de transiciones responde al hecho de que cuando se recorre la tabla, los datos en ella consignados indican cuáles son los valores que almacenan los biestables y sus transiciones cuando se pasa de un estado al estado futuro.

En resumen, la tabla de transiciones será idéntica a la tabla de estados con la salvedad de que cada estado se expresa con un conjunto de bits en lugar de por su nombre.

Tabla de funcionamiento

La tabla de funcionamiento se obtiene al transformar la tabla de transiciones en una tabla de verdad de las funciones de transición y de salida, donde:

- las variables de entrada las forman las entradas principales E y la codificación del estado actual $Q(t)$ de la máquina (salidas de los biestables) y
- las salidas de la misma son las funciones de transición que especifican el estado futuro $Q(t+1)$ y las funciones de salida ($S(t)$ o $S(t+1)$, según sea una máquina de Moore o de Mealy, respectivamente).

Además, como *salidas* de la tabla de funcionamiento también se indican los valores que han de tomar las entradas de los biestables para que estos realicen las transiciones indicadas. Cuando los biestables utilizados sean del tipo D, las entradas de los mismos coinciden con los valores de $Q(t+1)$ (ya que $D = Q(t+1)$); cuando los biestables sean de otro tipo (RS, JK o T), se tendrán que confeccionar las funciones particulares para las entradas de los mismos. La forma de obtener estas funciones se verá en la sección 5.5.

5.4. Análisis de circuitos secuenciales

El análisis de un circuito consiste en la obtención de una descripción funcional del mismo a partir de su descripción estructural. En el caso de los circuitos secuenciales, la descripción funcional que interesa obtener (por ser más intuitiva desde el punto de vista humano) es la de máquina ya sea ésta de Moore o de Mealy. Para llegar a esta descripción, es necesario primero obtener la tabla de funcionamiento del circuito, después su tabla de transiciones, después, opcionalmente, su tabla de estados. A partir de la tabla de estados o de la de transiciones se puede construir la máquina de Moore o de Mealy que describe el funcionamiento del circuito.

El proceso de obtención de la tabla de funcionamiento es similar al de la obtención de la tabla de verdad de un circuito combinacional, teniendo en cuenta que existen funciones y variables internas ($Q_i(t+1)$ y $Q_i(t)$, respectivamente) que definen las funciones de transición. Las variables internas ($Q_i(t)$) son las salidas de los biestables. Las funciones internas ($Q_i(t+1)$), si los biestables son de tipo D, son las entradas de los biestables; si los biestables son de otro tipo, $Q_i(t+1)$ se obtiene como una función de $Q_i(t)$ y las entradas del biestable correspondiente.

La tabla de transiciones, la tabla de estados, en su caso, y la máquina de Moore o de Mealy se siguen sin demasiada dificultad a partir de la tabla de funcionamiento.

Como ejemplo de análisis de circuitos secuenciales, y para poder mostrar los pasos que intervienen en dicho proceso, se propone el análisis del circuito representado en la Figura 5.20.

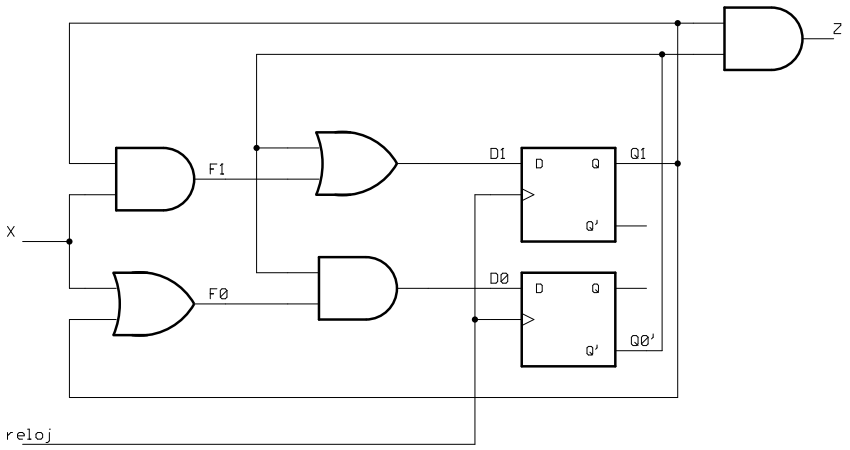


Figura 5.20: Ejemplo de análisis de css: Circuito secuencial síncrono

El primer paso en el análisis del anterior circuito es la obtención de la tabla de funcionamiento del mismo.

X	$Q_1(t)$	$Q_0(t)$	$F1$	$F0$	$Q_1(t+1)$	$Q_0(t+1)$	$Z(t)$	$Z(t+1)$
0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	0	0	0
0	1	0	0	1	1	1	1	0
0	1	1	0	1	0	0	0	0
1	0	0	0	1	1	1	0	0
1	0	1	0	1	0	0	0	0
1	1	0	1	1	1	1	1	0
1	1	1	1	1	1	0	0	1

Cuadro 5.3: Ejemplo de análisis de css: Tabla de funcionamiento

Las entradas de dicha tabla (véase la Tabla 5.3) son las entradas principales del circuito X y el valor del estado actual $Q(t)$ que está formado por las funciones $Q_1(t)$ y $Q_0(t)$ (las salidas de los dos biestables tipo D).

Como funciones intermedias y para simplificar el cálculo del resto de funciones se han obtenido los valores que toman $F1$ y $F0$ en función de las anteriores entradas.

Las salidas de la tabla de funcionamiento son: el estado futuro $Q(t+1)$ y la salida del circuito. El estado futuro lo determinan las funciones $Q_1(t+1)$ y $Q_0(t+1)$ que, al ser los biestables de tipo D, son las funciones de entrada de los mismos: D_1 y D_0.

Por último, dependiendo del tipo de máquina al que se quiera llegar, existen dos posibilidades a la hora de obtener las funciones de salida. Si se desea obtener un autómata de Moore como resultado del análisis, es suficiente con calcular $Z(t)$, pero si se desea obtener un autómata de Mealy, es necesario calcular $Z(t+1)$. Esta

última, se obtiene por deducción a partir de la forma algebraica de $Z(t)$. Así, para este ejemplo, $Z(t+1)$ se obtendría como:

$$Z(t) = Q_1(t) \cdot \overline{Q_0(t)} \Rightarrow Z(t+1) = Q_1(t+1) \cdot \overline{Q_0(t+1)}$$

En este ejemplo, aparecen $Z(t)$ y $Z(t+1)$ en la tabla de funcionamiento porque se va a obtener tanto la máquina de Mealy como la de Moore. Se verá en primer lugar como se realiza el resto del análisis cuando se trata de una máquina de Moore.

El siguiente paso en el análisis es la obtención de la tabla de transiciones a partir de la tabla de funcionamiento (siendo $Z(t)$ la salida que se va a utilizar). La tabla de transiciones (véase Tabla 5.4) es simplemente una redistribución de los contenidos de la tabla de funcionamiento. En concreto: los posibles estados actuales $Q_1(t)Q_0(t)$ y la salida asociada $Z(t)$ son los encabezados de las filas; los posibles valores de la entrada X son los encabezados de las columnas; y cada celda contiene el estado futuro $Q_1(t+1)Q_0(t+1)$.

$Q_1(t)Q_0(t)/Z(t) \setminus X$	0	1
00/0	10	11
01/0	00	00
10/1	11	11
11/0	00	10

Cuadro 5.4: Ejemplo de análisis de css: Tabla de transiciones (Moore)

En el análisis de un circuito secuencial se puede obviar la obtención de la tabla de estados considerando que los valores codificados son el nombre de cada estado.

Así, la máquina de Moore se puede construir directamente a partir de la tabla de transiciones. La Figura 5.21 muestra la máquina de Moore resultante con la que el análisis del circuito secuencial síncrono propuesto se ha completado.

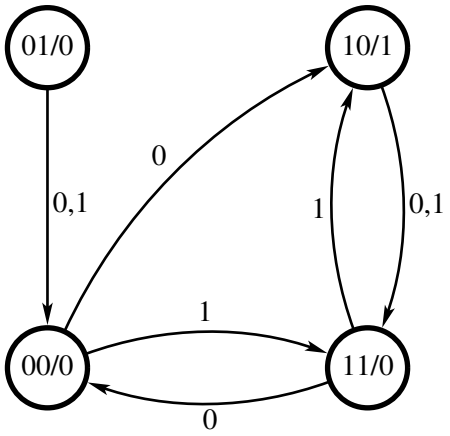


Figura 5.21: Ejemplo de análisis de css: Máquina de Moore

En cuanto al análisis de la máquina de Mealy, partiendo de la tabla de funcionamiento presentada en la Tabla 5.3 (siendo $Z(t+1)$ la salida que se va a utilizar), el siguiente paso consiste en la obtención de la tabla de transiciones.

La tabla de transiciones (véase Tabla 5.5) es simplemente, al igual que en el análisis para Moore, una redistribución de los contenidos de la tabla de funcionamiento. En concreto: los posibles estados actuales $Q_1(t)Q_0(t)$ son los encabezados de las filas; los posibles valores de la entrada X son los encabezados de las columnas; y cada celda contiene el estado $Q_1(t+1)Q_0(t+1)$ y salida $Z(t+1)$ futuros.

$Q_1(t)Q_0(t)$ \ X	0	1
00	10/1	11/0
01	00/0	00/0
10	11/0	11/0
11	00/0	10/1

Cuadro 5.5: Ejemplo de análisis de css: Tabla de transiciones (Mealy)

Igual que para el caso de la máquina de Moore, la obtención de la tabla de estados se puede obviar si se considera que los valores codificados son el nombre de cada estado.

Así, la máquina de Mealy se puede construir directamente a partir de la tabla de transiciones. La Figura 5.22 muestra la máquina de Mealy resultante con la que el análisis del circuito secuencial síncrono propuesto se ha completado.

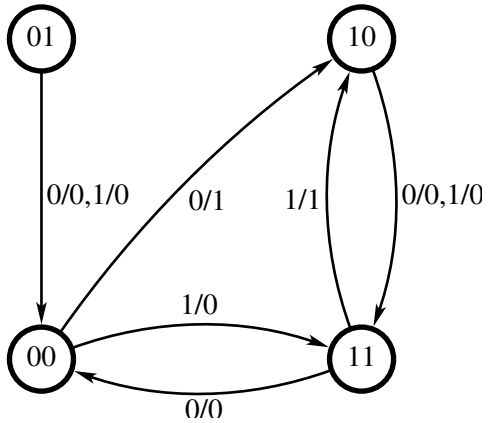


Figura 5.22: Ejemplo de análisis de css: Máquina de Mealy

5.5. Síntesis de circuitos secuenciales

La síntesis de circuitos secuenciales consiste en la obtención de una descripción estructural de un circuito (conjunto de componentes elementales interconectados entre sí) a partir de unas especificación funcional del mismo. Como consecuencia de esto, el primer paso para realizar una síntesis consistirá en definir adecuadamente el funcionamiento del circuito y a continuación, a partir de la misma, seguir un procedimiento sistemático que culmine con la obtención de la descripción estructural del mismo. La secuencia de pasos a seguir será la siguiente:

1. Deducir el número de entradas y salidas que debe poseer el circuito a partir de la descripción funcional expresada en el enunciado del problema.

2. Determinar si se trata de un circuito combinacional o secuencial. La caracterización de los circuitos secuenciales viene determinada por la necesidad de almacenar información para determinar las salidas.

3. Determinar el método de síntesis que se va a utilizar: Moore o Mealy. Aunque la definición y desarrollo por Mealy es más general, en algunos casos resulta más sencillo utilizar Moore. La elección está determinada por la dependencia o no de las salidas con respecto a las entradas principales del circuito. Si las salidas no dependen directamente de las entradas principales (es decir, sólo dependen del estado actual) entonces el método más apropiado es Moore. En caso contrario, se debe utilizar Mealy. Como en algunos casos, es difícil determinar a partir de la descripción funcional si las salidas dependen o no de las entradas principales, se resuelve en principio la máquina de Mealy y antes de continuar la síntesis del circuito, se analiza a partir de la máquina si las salidas dependen o no de las entradas. Si ése es el caso, se continuará la síntesis utilizando el método de Mealy, en caso contrario, se utilizará el de Moore.

En una máquina de Mealy, las salidas no dependen de las entradas cuando se cumple, para todos los estados, que todos los enlaces que llegan a cada estado tienen etiquetada la misma salida.

4. Obtener la máquina de Moore o de Mealy, la tabla de estados equivalente y, a partir de ésta, la tabla de transiciones.

5. Realizar la elección de los biestables que se utilizarán para la implementación del circuito y, en función de los tipos de éstos, obtener la tabla de funcionamiento del circuito.

6. Sintetizar todas las funciones lógicas definidas en la tabla de funcionamiento obteniendo el correspondiente circuito combinacional.

7. Obtener la descripción estructural del circuito secuencial utilizando el circuito combinacional descrito y la descripción estructural de la memoria (biestables) (véase de nuevo la Figura 5.2).

Para ilustrar la síntesis de circuitos secuenciales, se realizan a continuación dos ejemplos de síntesis: el primero, utilizando el método de Mealy y el segundo, el de Moore.

Síntesis por el método de Mealy

Enunciado del problema

Diseñar un sumador serie de dos operandos, considerando que los operandos se envían bit a bit de menor a mayor peso (se introduce un nuevo bit del operando en cada ciclo de reloj).

Caracterización del circuito

El circuito tiene dos entradas (el bit a sumar de cada uno de los operandos) y una salida (el resultado de la suma). Para determinar el valor de la salida es necesario considerar, además de la suma de los dos operandos, el posible acarreo producido en la suma anterior. El circuito tiene que ser capaz, por tanto, de memorizar si el resultado de la última operación provocó un acarreo o no. Es decir, el circuito es

secuencial y posee dos estados a los que se puede denominar como: *Sin Acarreo* y *Con Acarreo*.

Para clarificar cuál es el funcionamiento esperado del circuito, en la Figura 5.23 se presenta como ejemplo la suma de los números 10 (001010_2) y 44 (101100_2).

	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$	
Operando 1	0	1	0	1	0	0	10
Operando 2	0	0	1	1	0	1	44
Salida (suma)	0	1	1	0	1	1	54
Estado	*Sin Acarreo*	*Sin Acarreo*	*Sin Acarreo*	*Con Acarreo*	*Sin Acarreo*	*Sin Acarreo*	

Figura 5.23: Ejemplo de funcionamiento del sumador serie propuesto

Este circuito se debe sintetizar empleando el método de Mealy porque las salidas dependen del estado actual (condición de acarreo) y además del valor que presentan las entradas (los operandos).

Máquina de Mealy

El funcionamiento del circuito se puede expresar mediante la máquina de Mealy representada en la Figura 5.24.

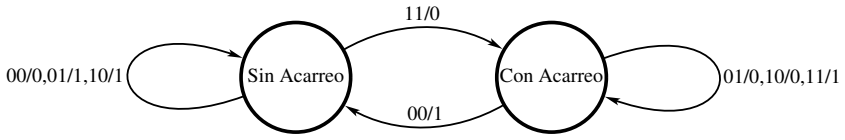


Figura 5.24: Máquina de Mealy del sumador serie propuesto

Si se denomina A a la entrada correspondiente al primer operando y B a la entrada correspondiente al segundo operando, la tabla de estados correspondiente es la representada en la Tabla 5.6.

Estados \ AB	00	01	10	11
Sin Acarreo	*Sin Acarreo*/0	*Sin Acarreo*/1	*Sin Acarreo*/1	*Con Acarreo*/0
Con Acarreo	*Sin Acarreo*/1	*Con Acarreo*/0	*Con Acarreo*/0	*Con Acarreo*/1

Cuadro 5.6: Tabla de estados del sumador serie propuesto

Para obtener la tabla de transiciones a partir de la de estados, se ha de asignar una codificación para la representación en binario de todos los estados posibles. Una posible codificación es la siguiente: el estado *Sin Acarreo* se representa por un 0 y el estado *Con Acarreo* se representa por un 1. Utilizando esta codificación, la tabla de transiciones es la representada en la Tabla 5.7.

Estado \ AB	00	01	10	11
0	0/0	0/1	0/1	1/0
1	0/1	1/0	1/0	1/1

Cuadro 5.7: Tabla de transiciones del sumador serie propuesto

Se desea utilizar para el diseño del circuito secuencial biestables tipo D, por lo que la tabla de funcionamiento (véase Tabla 5.8) se obtiene sin más que redistribuir el contenido de la tabla de transiciones.

A	B	$Q(t)$	$Q(t+1)$	$S(t+1)$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Cuadro 5.8: Tabla de funcionamiento del sumador serie propuesto

Una vez obtenida la tabla de funcionamiento, únicamente resta la construcción del circuito combinacional que la implemente. Para ello, se utiliza un decodificador 3×8 con salidas activas a nivel alto. Puesto que tanto $Q(t+1)$ como $S(t+1)$ poseen cuatro ceros y cuatro unos, es indistinto resolver dichas funciones por ceros o por unos. El circuito propuesto (véase Figura 5.25), resuelve $Q(t+1)$ por unos y $S(t+1)$ por ceros. Como se puede observar en dicho circuito, al ser la salida $S(t+1)$ la salida en el instante futuro, es necesario utilizar un biestable D para retrasarla y obtener la salida en el instante actual (véase de nuevo la Figura 5.25).

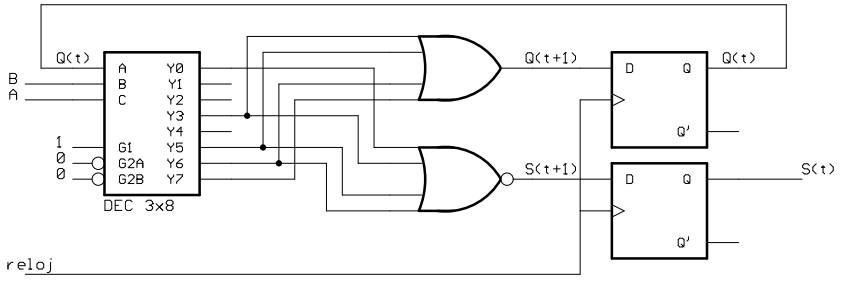


Figura 5.25: Circuito que implementa el sumador serie propuesto

Síntesis por el método de Moore

Enunciado del problema

Diseñar un circuito secuencial síncrono que detecte una secuencia de tres o más "unos" consecutivos.

Caracterización del circuito

El circuito posee una entrada A y una salida. Es un circuito secuencial ya que tiene que memorizar el número de "unos" consecutivos que se han visto en la entrada para poder indicar cuando se han visto 3 o más "unos" consecutivos. Son necesarios 4 estados para poder contemplar todas las situaciones posibles, a saber:

No Llevo No se ha detectado ningún "uno". Es decir, la última entrada fue "cero".

Llevo 1 Se ha detectado un "uno". Es decir, la última entrada fue "uno" y la anterior "cero".

Llevo 2 Se ha detectado una secuencia de dos "unos" consecutivos. Es decir, las dos últimas entradas fueron "unos" y la anterior a éstas fue "cero".

Llevo $\geq$ 3 Se ha detectado una secuencia de tres o más "unos" consecutivos. Es decir, las tres últimas entradas fueron "unos".

Para clarificar cuál es el funcionamiento esperado de este circuito, en la Figura 5.26 se presenta un ejemplo del funcionamiento del mismo ante la siguiente entrada: 011011110.

	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$	$t+6$	$t+7$	$t+8$
Entrada	0	1	1	0	1	1	1	1	0
Salida	0	0	0	0	0	0	1	1	0
Estado	*No Llevo*	*Llevo 1*	*Llevo 2*	*No Llevo*	*Llevo 1*	*Llevo 2*	*Llevo* $\geq$ *3*	*Llevo* $\geq$ *3*	*No Llevo*

Figura 5.26: Ejemplo de funcionamiento del detector de secuencia propuesto

Como se puede observar en el ejemplo y por el enunciado, cuando el circuito está en el estado "Llevo $\geq$ 3" la salida es "uno" y cuando está en cualquier otro estado es "cero", independientemente de cuál sea el valor de la entrada. Debido a esto, se deduce que el método óptimo para su resolución es el de Moore.

Máquina de Moore

El funcionamiento del circuito se puede expresar mediante la máquina de Moore representada en la Figura 5.27.

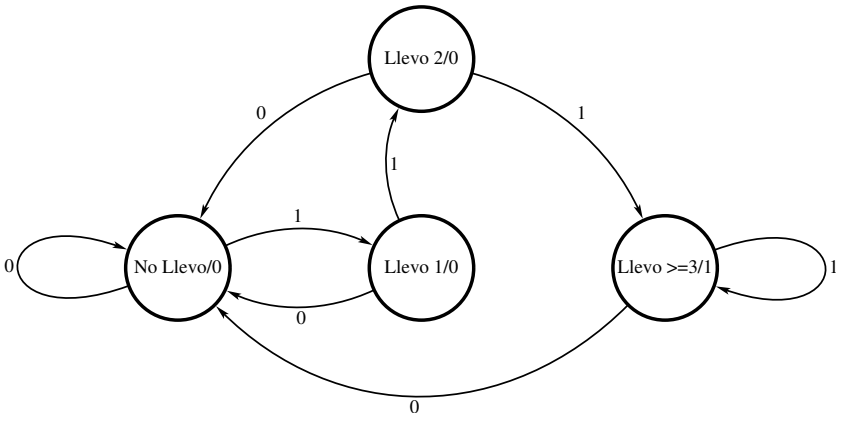


Figura 5.27: Máquina de Moore del detector de secuencia propuesto

Dada la máquina Moore, se obtiene la tabla de estados representada en la Tabla 5.9.

Para construir la tabla de transiciones es necesario codificar en binario los estados. La codificación elegida es la siguiente: *No Llevo*= 00, *Llevo 1*= 01, *Llevo 2*= 10 y *Llevo* $\geq$ *3*= 11. Utilizando está codificación, la tabla de transiciones queda tal y como se representa en la Tabla 5.10.

Estado/$S(t) \setminus A$	0	1
No Llevo/0	*No Llevo*	*Llevo 1*
Llevo 1/0	*No Llevo*	*Llevo 2*
Llevo 2/0	*No Llevo*	*Llevo* $\geq$ *3*
Llevo $\geq$ *3*/1	*No Llevo*	*Llevo* $\geq$ *3*

Cuadro 5.9: Tabla de estados del detector de secuencia propuesto

$Q_1(t)Q_0(t)/S(t) \setminus A$	0	1
00/0	00	01
01/0	00	10
10/0	00	11
11/1	00	11

Cuadro 5.10: Tabla de transiciones del detector de secuencia propuesto

Si se utilizan biestables tipo D para la implementación del circuito, la tabla de funcionamiento consiste en la reorganización de la información contenida en la tabla de transiciones (véase Tabla 5.11).

A	$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$	$S(t)$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	1	1

Cuadro 5.11: Tabla de funcionamiento del detector de secuencia propuesto

Una vez obtenida la tabla de funcionamiento, el último paso es la construcción del circuito que lo implementa. La Figura 5.28 muestra una posible implementación de dicho circuito.

Si en lugar de utilizar biestables tipo D, se desean utilizar biestables de otro tipo, es necesario obtener las funciones de entrada de los mismos. A modo de ejemplo, se va a añadir a la anterior tabla de funcionamiento las entradas necesarias para utilizar biestables tipo T, JK o RS.

La forma de obtener estas entradas es la siguiente: para cada combinación de la tabla de funcionamiento, las entradas del biestable elegido han de ser tales que dado el estado actual $Q(t)$ provoquen una transición al estado futuro $Q(t+1)$. Es decir, interesa conocer para toda posible transición de $Q(t)$ a $Q(t+1)$, qué combinaciones de entradas de un biestable en concreto provocan esa transición.

Si el biestable es de tipo T, para que el estado futuro sea igual al estado actual, la entrada T debe valer 0, y para que el estado futuro sea distinto al estado actual

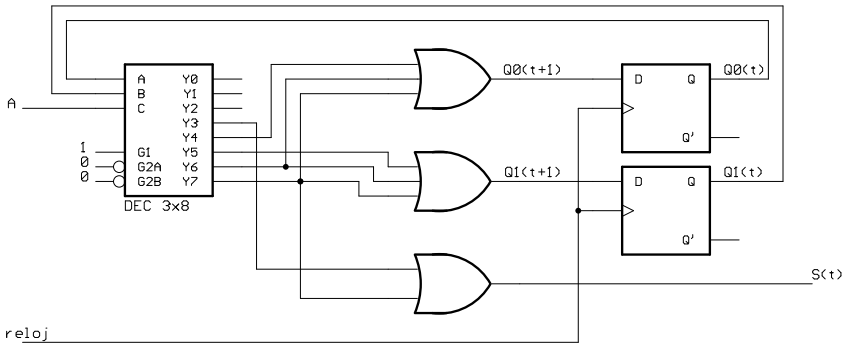


Figura 5.28: Circuito que implementa el detector de secuencia propuesto

la entrada T debe valer 1, por lo que la tabla de obtención del estado futuro de un biestable T es la representada en la Tabla 5.12a.

Si el biestable es de tipo JK, para que dado el estado actual igual a 0, el futuro sea igual a 0, existen dos opciones: $J = 0, K = 0$ (mantener el estado actual) o $J = 0, K = 1$ (hacer 0 el estado futuro); para que dado el estado actual igual a 0, el futuro sea igual a 1, existen dos posibilidades: $J = 1, K = 1$ (invertir el estado actual) o $J = 1, K = 0$ (hacer 1 el estado futuro); para que dado el estado actual igual a 1, el futuro sea igual a 0, existen dos opciones: $J = 1, K = 1$ (invertir el estado actual) o $J = 0, K = 1$ (hacer 0 el estado futuro); y por último, para que dado el estado actual igual a 1, el futuro sea igual a 1, existen, a su vez, dos posibilidades: $J = 0, K = 0$ (mantener el estado actual) o $J = 1, K = 0$ (hacer 1 el estado futuro). La Tabla 5.12b muestra la tabla de obtención del estado futuro de un biestable JK.

Finalmente, si el biestable es de tipo RS, para que dado el estado actual igual a 0, el futuro sea igual a 0, existen dos opciones: $R = 0, S = 0$ (mantener el estado actual) o $R = 1, S = 0$ (hacer 0 el estado futuro); para que dado el estado actual igual a 0, el futuro sea igual a 1, tan sólo existe la siguiente combinación: $R = 0, S = 1$ (hacer 1 el estado futuro); para que dado el estado actual igual a 1, el futuro sea igual a 0, tan sólo existe la siguiente combinación: $R = 1, S = 0$ (hacer 0 el estado futuro); y por último, para que dado el estado actual igual a 1, el futuro sea igual a 1, existen, a su vez, dos posibilidades: $R = 0, S = 0$ (mantener el estado actual) o $R = 0, S = 1$ (hacer 1 el estado futuro). La Tabla 5.12c muestra la tabla de obtención del estado futuro de un biestable RS.

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

(a)

$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(b)

$Q(t)$	$Q(t+1)$	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

(c)

Cuadro 5.12: Tabla de obtención del estado futuro de un biestable T (a), JK (b) y RS (c)

Teniendo en cuenta las tablas de obtención de los biestables T, JK y RS se puede

construir una tabla de funcionamiento en la que se especifiquen cuáles deben ser las entradas de los mismos para poder implementar el circuito utilizando cualquiera de estos circuitos. La Tabla 5.13 muestra los valores de las entradas para todos los tipos de biestables.

A	$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$	T_1	T_0	J_1K_1	J_0K_0	R_1S_1	R_0S_1	$S(t)$
0	0	0	0	0	0	0	0X	0X	X0	X0	0
0	0	1	0	0	0	1	0X	X1	X0	10	0
0	1	0	0	0	1	0	X1	0X	10	X0	0
0	1	1	0	0	1	1	X1	X1	10	10	1
1	0	0	0	1	0	1	0X	1X	X0	01	0
1	0	1	1	0	1	1	1X	X1	01	10	0
1	1	0	1	1	0	1	X0	1X	0X	01	0
1	1	1	1	1	0	0	X0	X0	0X	0X	1

Cuadro 5.13: Tabla de funcionamiento para cualquier tipo de biestables del detector de secuencia propuesto

5.6. Registros

Un registro es un circuito formado por un conjunto de biestables tipo D, cuya función principal es el almacenamiento de información. Se distinguen dos tipos de registros: de almacenamiento y de desplazamiento.

Registros de almacenamiento

Los registros de almacenamiento son aquellos que tienen tantas entradas y tantas salidas como biestables posee el registro. De esta forma, cada biestable tiene su propia entrada y su propia salida. La señal de reloj es común a todos los biestables. También se puede denominar como registro de entrada paralelo y salida paralelo. La Figura 5.29 representa la estructura básica de un registro de almacenamiento de n bits.

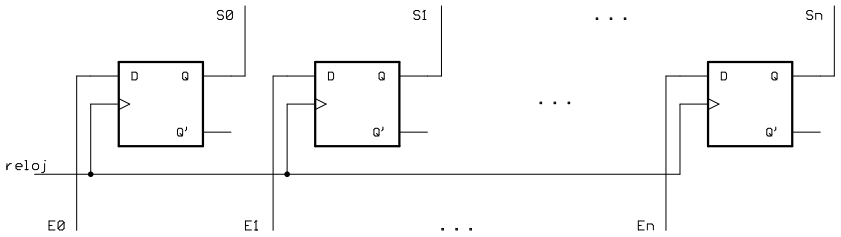


Figura 5.29: Registro de almacenamiento

Registros de desplazamiento

Los registros de desplazamiento tienen la misma finalidad que los de almacenamiento, pero permiten que la información se introduzca o extraiga en serie.

Si el registro es de entrada serie, con cada ciclo de reloj se introduce en él un nuevo dato. Si el registro es de salida serie, con cada ciclo de reloj, el registro suministra un nuevo dato. Esto implica que el contenido de los biestables ha de desplazarse de unos biestables a otros. Un desplazamiento puede implicar que el valor de cada biestable se transfiera al biestable de su derecha (desplazamiento a la derecha) o al biestable de su izquierda (desplazamiento a la izquierda).

La Figura 5.30 muestra la estructura básica de un registro de desplazamiento a la derecha con entrada serie y salida serie.

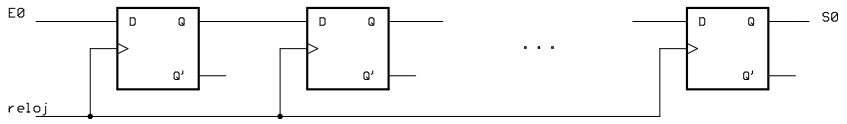


Figura 5.30: Registro de desplazamiento a la derecha con entrada serie y salida serie

La Tabla 5.14 ilustra el funcionamiento de un registro de desplazamiento a la derecha de 3 bits con entrada serie y salida serie.

reloj	E	$Q_0(t)$	$Q_1(t)$	$Q_2(t)$
—	1	0	0	0
↑	1	1	0	0
—	0	1	0	0
↑	0	0	1	0
—	1	0	1	0
↑	1	1	0	1

Cuadro 5.14: Funcionamiento de un registro de desplazamiento a la derecha de 3 bits con entrada serie (E) y salida serie ($Q_2(t)$)

Existe también la posibilidad de tener registros de entrada serie y salida paralelo empleados para conversiones serie-paralelo cuya estructura básica es la mostrada en la Figura 5.31.

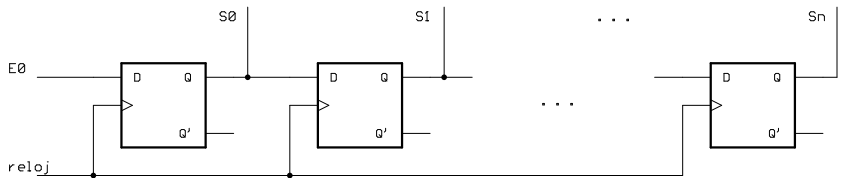


Figura 5.31: Registro de desplazamiento de entrada serie y salida paralelo

5.7. Contadores

Un contador es un circuito secuencial que en cada flanco de la señal de reloj presenta a su salida el siguiente valor dentro de un código cíclico. Por ejemplo, un

contador de código Gray de 2 bits presentaría a su salida la secuencia 00, 01, 11, 10, 00, ...; un contador de binario natural de 2 bits: 00, 01, 10, 11, 00, ...; un contador de BCD (4 bits): 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 0000, ..., etc.

Se pueden distinguir dos tipos de contadores: ascendentes y descendentes. Los contadores ascendentes son aquellos para los que el valor de su salida es siempre una unidad mayor al valor anterior (salvo cuando vuelve a empezar la cuenta). Todos los ejemplos anteriores son ejemplos de contadores ascendentes. Los contadores descendentes realizan la cuenta al revés de los contadores ascendentes, es decir, el valor que presentan a su salida es siempre menor al valor anterior (excepto cuando vuelve a empezar la cuenta).

La Figura 5.32 muestra un ejemplo de contador ascendente: un contador de binario natural ascendente de dos bits. Como se puede ver, se ha elegido la máquina de Moore para representar el contador, esto es así, debido a que normalmente es más sencillo describir un contador utilizando la máquina de Moore.

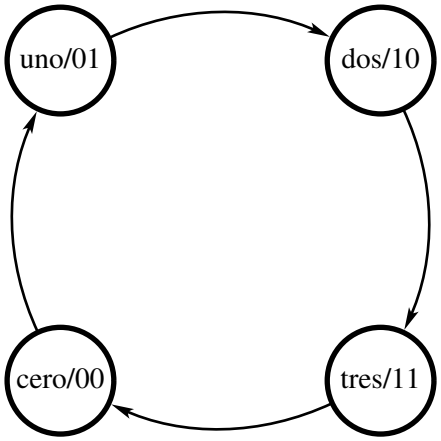


Figura 5.32: Máquina de Moore de un contador binario ascendente de 2 bits

La síntesis de este contador se realiza de la forma usual, es decir, a partir de la máquina de estados, se extrae la tabla de estados, de ésta, la de transiciones y, de esta última, la tabla de funcionamiento. La única salvedad está en que, como el circuito no tiene entradas, el número de columnas de las tablas de estados y de transiciones para indicar los estados futuros será igual a uno. La Figura 5.33 muestra las tablas de estados y de transiciones del contador, así como las expresiones algebraicas de $Q_1(t+1)$ y de $Q_0(t+1)$ (las salidas del contador son directamente el código del estado en el que se encuentra la máquina, es decir, las salidas de los biestables).

Contadores asíncronos

Los contadores asíncronos reciben su nombre no por no utilizar una señal de reloj (como es el caso de los circuitos secuenciales asíncronos), sino porque no se utiliza una señal de reloj común para todos los biestables. Generalmente, los biestables que no utilizan la señal de reloj externa, utilizan como señal de reloj, o derivada, la salida de otro biestable.

La utilización de estos contadores está muy extendida debido a su simplicidad.

Estado/$S_1(t)S_2(t)$	
0/00	1
1/01	2
2/10	3
3/11	0
(a)	

$Q_1(t)Q_0(t)/S_1(t)S_2(t)$	
00/00	01
01/01	10
10/10	11
11/11	00
(b)	

$$Q_1(t+1) = Q_1(t) \oplus Q_0(t)$$
$$Q_0(t+1) = \overline{Q_0(t)}$$
(c)

Figura 5.33: Tabla de estados (a), de transiciones (b) y expresiones algebraicas de $Q_1(t+1)$ y $Q_0(t+1)$ (c)

Como ejemplos de análisis de contadores asíncronos (no se va a ver la síntesis) se va a estudiar el funcionamiento de un contador asíncrono binario natural descendente de 3 bits y un contador asíncrono binario natural ascendente de 3 bits.

A) Contador asíncrono binario natural descendente de 3 bits

El circuito de este contador es el representado en la Figura 5.34. Como se puede observar en ésta, los tres biestables T tienen un "uno" a su entrada por lo que con cada flanco de subida de sus respectivas señales de reloj, éstos conmutarán el valor de su estado. El biestable de menor peso, lo hará con cada periodo de la señal de reloj externa. El biestable de peso 1, lo hará cuando la salida del biestable 0 pase de 0 a 1 (puesto que la salida del biestable 0 está conectada a su entrada de reloj), por lo que cambia de estado cada 2 periodos de la señal de reloj externa. Por último, el biestable de mayor peso, conmutará su estado cuando la salida del biestable 1 pase de 0 a 1 (puesto que la salida del biestable 1 está conectada a su entrada de reloj), por lo que cambia de estado cada 4 periodos de la señal de reloj externa.

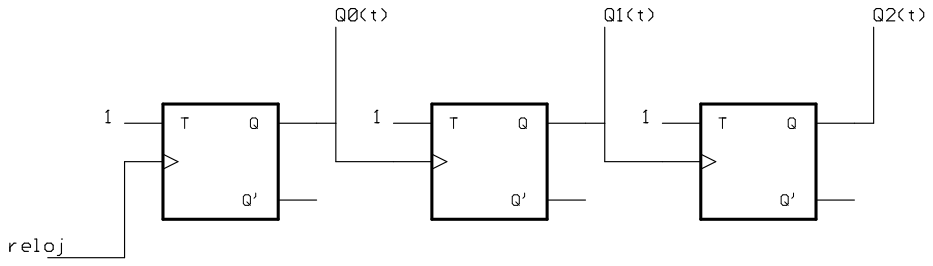


Figura 5.34: Contador asíncrono binario natural descendente de 3bits

La Tabla 5.15 muestra la tabla de funcionamiento de este contador. La forma de construir esta tabla es la siguiente: se parte del estado 0 (todos los biestables a 0) y se calcula el estado futuro (primero $Q_0(t+1)$ que depende de la señal de reloj externa, después $Q_1(t)$ que depende de la variación entre $Q_0(t)$ y $Q_0(t+1)$, y por último, $Q_2(t)$ que depende de la variación entre $Q_1(t)$ y $Q_1(t+1)$); una vez calculado, el estado futuro se pone en la siguiente fila de la tabla como estado actual y así sucesivamente hasta completar la tabla.

$Q_2(t)$	$Q_1(t)$	$Q_0(t)$	$Q_2(t+1)$	$Q_1(t+1)$	$Q_0(t+1)$
0	0	0	1	1 ↑	1 ↑
1	1	1	1	1 —	0 ↓
1	1	0	1	0 ↓	1 ↑
1	0	1	1	0 —	0 ↓
1	0	0	0	1 ↑	1 ↑
0	1	1	0	1 —	0 ↓
0	1	0	0	0 ↓	1 ↑
0	0	1	0	0 —	0 ↓

Cuadro 5.15: Tabla de funcionamiento de un contador asíncrono binario natural descendente de 3 bits

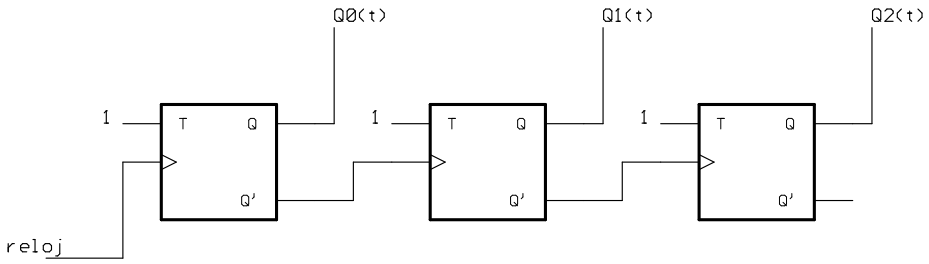


Figura 5.35: Contador asíncrono binario natural ascendente de 3bits

B) Contador asíncrono binario natural ascendente de 3 bits

El circuito de este contador es el representado en la Figura 5.35. Como se puede observar en ésta, el esquema es básicamente el del contador asíncrono descendente. La diferencia entre ambos está en que las señales de reloj de los biestables 1 y 2, ahora son las salidas completadas de los biestables 0 y 1, respectivamente, en lugar de las salidas sin complementar. Es decir, la conmutación de los biestables 1 y 2 se produce cuando los biestables 0 y 1, respectivamente, cambian de 1 a 0 y no de de 0 a 1 como ocurría en el ejemplo anterior.

La Tabla 5.16 muestra la tabla de funcionamiento de este contador. La forma de construir esta tabla es igual a la presentada para el contador descendente, salvo que ahora los flancos de bajada de las señales $Q_1(t)$ y $Q_0(t)$ (en realidad, los flancos de subida de las señales $\overline{Q_1(t)}$ y $\overline{Q_0(t)}$) son los que provocan los cambios en $Q_2(t)$ y en $Q_1(t)$.

$Q_2(t)$	$Q_1(t)$	$Q_0(t)$	$Q_2(t+1)$	$Q_1(t+1)$	$Q_0(t+1)$
0	0	0	0	0 —	1 ↑
0	0	1	0	1 ↑	0 ↓
0	1	0	0	1 —	1 ↑
0	1	1	1	0 ↓	0 ↓
1	0	0	1	0 —	1 ↑
1	0	1	1	1 ↑	0 ↓
1	1	0	1	1 —	1 ↑
1	1	1	0	0 ↓	0 ↓

Cuadro 5.16: Tabla de funcionamiento de un contador asíncrono binario natural ascendente de 3 bits

CAPÍTULO 6

Memorias

La memoria es una de las unidades funcionales de un computador. Su función es la de almacenar tanto datos como programas. Para que esta información pueda ser utilizada, la memoria ha de permitir el acceso a la misma, ya sea para su consulta o para su modificación. La consulta a la información almacenada en una memoria se realiza por medio de operaciones de lectura y la modificación, por medio de operaciones de escritura.

En el capítulo anterior se vieron circuitos capaces de memorizar el estado en el que se encuentran. Una de las aplicaciones de dichos circuitos era la creación de dispositivos capaces de almacenar información: los registros. Una memoria equivale conceptualmente a un conjunto o colección de estos registros.

6.1. Características de las memorias

La Tabla 6.1 muestra un resumen de las principales características de una memoria. Éstas se explican a continuación.

Unidad de transferencia. Es el número de bits que pueden ser leídos o escritos en una memoria de forma simultánea. Cuando la unidad de transferencia es relativamente pequeña (8, 16, 32, 64), recibe el nombre de *palabra*. Cuando la unidad de transferencia es mayor, recibe el nombre de *bloque*.

Capacidad. Es la cantidad de información que la memoria puede almacenar. Se suele expresar como el número de bytes que posee (p.ej. un disco duro de $17 Gbytes$). Si la memoria está organizada en palabras, también se puede proporcionar su capacidad de la forma: número de palabras $\times$ tamaño de palabra (p.ej. una RAM de $32M \times 64$ ($256 Mbytes$)).

Tipo de material. Para poder utilizar un determinado material como elemento de memoria, debe cumplir las siguientes propiedades:

- Debe presentar al menos dos estados diferenciados y caracterizados por una magnitud física concreta (carga eléctrica, corriente, campo magnético, reflexión, etc.).

Unidad de transferencia	**Método de acceso**
Palabra	Acceso secuencial
Bloque	Acceso directo
Capacidad	Acceso aleatorio
Número de palabras	Acceso asociativo
Tamaño de palabra	**Prestaciones**
Tipo de material	Tiempo de acceso
Semiconductor	Tiempo de ciclo
Superficie magnética	Ratio de transferencia
Superficie óptica	**Localización**
Tiempo de latencia	Interna (principal)
Dinámica	Externa (secundaria)
Estática	
Duradera	
Permanente	

Cuadro 6.1: Principales características de una memoria

- Debe ser posible determinar el estado en que se encuentra en un instante dado.

Una memoria construida de un material que cumpla las anteriores propiedades puede utilizarse para la lectura de la información almacenada en ella. Si, además, se desea poder escribir en la misma, el material empleado debe ser tal que el paso de uno de los estados al otro sea posible mediante la aplicación de una energía externa. Además, para poder escribir repetidas veces, los cambios de un estado a otro deberán ser reversibles y no producir un deterioro apreciable del material.

Algunos de los materiales utilizados en la actualidad para la construcción de memorias son: *semiconductores*, *superficies magnéticas* y *superficies ópticas*.

Tiempo de latencia. Es el tiempo durante el cual la memoria es capaz de mantener la información que contiene. En la siguiente lista se presentan, ordenados de menor a mayor tiempo de conservación, los tipos de memoria que existen:

Dinámicas La información se degrada paulatinamente aunque la memoria esté alimentada, lo que hace necesario refrescarla de forma periódica. Un ejemplo de memoria con refresco lo constituyen las memorias DRAM (*Dynamic RAM* — RAM dinámica—), donde se utiliza la carga de un condensador para indicar el valor lógico de un bit. Como la carga de un condensador desaparece con el tiempo, éste necesita ser recargado de forma periódica. La memoria principal de los computadores suele ser de este tipo debido a su bajo coste y alta capacidad de integración.

Estáticas La información permanece en la memoria únicamente mientras ésta recibe alimentación eléctrica. A este tipo pertenecen las memorias que utilizan dispositivos semiconductores como medio de almacenamiento. Debido a que esta memoria es muy rápida, la memoria caché de los computadores suele ser de este tipo.

Duraderas La información permanece en la memoria sin necesidad de una fuente de alimentación que la mantenga hasta que se borra intencionadamente. A este tipo pertenecen los dispositivos de memoria que utilizan soportes magnéticos como medio de almacenamiento (p.ej. discos, cintas, CD-RWs, etc.).

Permanentes La información no puede modificarse una vez grabada. Estas memorias se denominan también de sólo lectura y se designan con las siglas ROM (*Read-Only Memory* —memoria de sólo lectura—). A efectos prácticos, se considerarán pertenecientes a este grupo la memorias PROM (*Programmable Read-Only Memory* —se puede escribir una sola vez sobre la misma—), EPROM (*Erasable Programmable Read-Only Memory* —es un tipo especial de PROM que puede ser borrada completamente exponiéndola a luz ultravioleta; una vez borrada, se puede volver a escribir en ella—), EEPROM (*Electrically Erasable Programmable Read-Only Memory* —idéntica a la EPROM, salvo que se borra eléctricamente—), memoria Flash (también llamada Flash EEPROM, se puede borrar y reprogramar la información en bloques mientras que la EEPROM sólo puede hacerlo palabra a palabra; al poder reescribir por bloques, la reprogramación de una memoria Flash es mucho más rápida que la de una EEPROM). También pertenecen a este grupo los CD-ROMs, DVD-ROMs, etc.

El término Flash BIOS indica que la BIOS de un computador personal está almacenada en una memoria Flash.

Método de acceso. Según la forma de acceder o localizar la información dentro de una memoria, se distingue entre:

Acceso secuencial El acceso a la información en este tipo de memorias se realiza de forma secuencial. Tanto la información propiamente dicha, que se organiza en unidades llamadas *registros* (*records*), como la información de direccionamiento, necesaria para separar un registro del siguiente permitiendo la identificación de los mismos, se almacenan en la memoria.

Disponen de un único mecanismo de lectura/escritura que ha de recorrer ordenadamente la información hasta que localiza el registro buscado. Por lo tanto, el tiempo de acceso a la información es *variable*, ya que depende de la distancia entre la posición del registro solicitado y la posición inicial del mecanismo de lectura/escritura.

Las unidades de cinta magnética son memorias de acceso secuencial.

Acceso directo o pseudo-aleatorio La información se organiza en bloques individuales o registros que poseen una dirección única basada en la posición física que ocupan.

Disponen de un único mecanismo de lectura/escritura que accede a la información en dos pasos: en el primero, el mecanismo se desplaza directamente a la posición física donde está el bloque que contiene la información buscada; y en el segundo, se recorre éste de forma secuencial hasta encontrar la información solicitada. Por lo tanto, el tiempo de acceso de las mismas, al igual que en el caso de las memorias de acceso secuencial, es variable, ya que depende de la posición inicial del mecanismo de lectura/escritura con respecto a la posición de los datos a los que se accede.

Los discos duros y las unidades de disquete son memorias de acceso directo o pseudo-aleatorio.

Acceso aleatorio Cada una de las posiciones de memoria tiene su propio mecanismo de lectura/escritura incorporado. De esta forma, una operación de lectura/escritura no depende de la secuencia de lecturas/escrituras previas y se realiza en un tiempo que no depende de la dirección de memoria accedida. Por lo tanto, el tiempo de acceso en este tipo de memorias es *constante*.

Las memorias ROM y RAM (véase Sección 6.3) son memorias de acceso aleatorio.

Acceso asociativo Es un tipo de memoria de acceso aleatorio que permite comparar ciertos bits de una palabra de forma simultánea con todas las palabras almacenadas en la misma. Es decir, se accede a una posición de memoria no por su dirección, sino por su contenido. El mecanismo de lectura/escritura, al igual que en las memorias de acceso aleatorio, está asociado a cada posición de memoria y el tiempo de acceso es constante e independiente tanto de la dirección a la que se accede como del patrón de accesos anteriores. Debido a la forma de acceder a la información, algunas memorias caché son de este tipo.

Prestaciones. Las prestaciones de una memoria se definen basándose en los siguientes tres parámetros: *tiempo de acceso*, *tiempo de ciclo* y *ratio de transferencia*:

Tiempo de acceso Para las memorias de acceso aleatorio, se denomina tiempo de acceso al tiempo con el que se realiza de forma completa una operación de lectura o escritura. Para las memorias de acceso no aleatorio, se denomina tiempo de acceso sólo al tiempo utilizado por el mecanismo de lectura/escritura para alcanzar la posición deseada.

Tiempo de ciclo de memoria Se aplica principalmente a las memorias de acceso aleatorio y consiste en el tiempo de acceso más cualquier tiempo adicional que sea requerido para que otra operación de lectura/escritura pueda iniciarse sobre la misma memoria.

Ratio de transferencia Es la cantidad de bits por unidad de tiempo que puede transferirse desde o hacia la memoria. Para una memoria de acceso aleatorio, este ratio se calcula como:

$$R = \frac{1}{\text{Tiempo de ciclo de memoria}}$$

Para una memoria de acceso no aleatorio, se obtiene de la siguiente forma:

$$R = \frac{n}{T_n - T_a},$$

donde n es el número de bits, T_n el tiempo medio de lectura de n bits y T_a el tiempo medio de acceso.

Localización. La memoria, como unidad funcional de un computador, puede ser, según su localización, *interna* o *externa*. Se tiende a asociar la memoria interna únicamente con la memoria principal del computador, aunque, en realidad, también son parte de la memoria interna los registros, la memoria caché, etc. La memoria externa la constituyen los dispositivos periféricos de almacenamiento (p.ej. discos duros, CD-ROMs, disquetes, cintas magnéticas, etc.); generalmente, se accede a éstos por medio de la unidad de entrada/salida.

6.2. Jerarquía de memorias

Vista la variedad de características que presentan las memorias, ¿cómo debería de ser la memoria de un computador? Es decir, ¿cuánta memoria?, ¿de qué velocidad?, ¿de qué coste?

La respuesta a estas cuestiones viene dada por la especificación de las siguientes características: capacidad, tiempo de acceso y coste por bit de la memoria que se quiere emplear.

Idealmente, esta memoria, debería tener una gran capacidad de almacenamiento —se podrían tratar problemas de gran envergadura—, un tiempo de acceso pequeño —se podría procesar la información a gran velocidad— y un coste reducido —el coste de la memoria no debería encarecer en demasía el precio final del computador.

Desgraciadamente, entre estas tres características están relacionadas, y a lo largo del tiempo y para diversas tecnologías, siempre se ha cumplido que:

- A menor tiempo de acceso, mayor coste por bit.
- A mayor capacidad, menor coste por bit.
- A mayor capacidad, mayor tiempo de acceso.

Puesto que la situación ideal sugerida con anterioridad no puede lograrse utilizando un único componente de memoria, se ha llegado a la siguiente solución: organizar distintos tipos de memorias en lo que se conoce como la *jerarquía de memorias* de un computador. Una memoria estará más abajo en esta jerarquía cuando(véase Figura 6.1):

- Mayor sea su capacidad
- Mayor sea el tiempo de acceso a su información.
- Menor sea su coste por bit.
- Menor sea la frecuencia de acceso a la información que almacena.

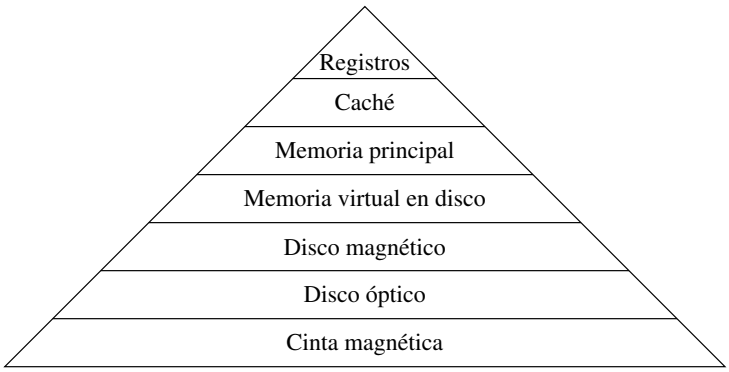


Figura 6.1: Jerarquía de memorias

De esta forma, memorias más rápidas, más pequeñas y más caras, se complementan con memorias más lentas, de mayor capacidad y más baratas. El éxito de organizar distintas memorias de esta forma, radica en la reducción del tiempo medio de acceso (cuanto más veces se requiera una determinada información, esta información se irá trasladando desde memorias situadas en niveles inferiores a memorias situadas en niveles superiores; conforme la frecuencia de acceso a esta información se reduzca, irá descendiendo en la escala jerárquica a memorias situadas en niveles inferiores), a la vez que se permite una gran capacidad de almacenamiento a unos costes razonables.

6.3. La memoria principal del computador

Dentro de la *memoria interna* de un computador, destaca por su importancia la llamada *memoria principal*. Hoy en día, la memoria principal es de tipo semiconductor. Esta sección trata los dos grupos de memoria de tipo semiconductor más comunes: la memoria ROM y la RAM.

Memoria ROM

La memoria ROM (*Read Only Memory*) es una memoria de acceso aleatorio de sólo lectura y, como su propio nombre sugiere, contiene una información permanente.

Las memorias ROM suelen utilizarse para la generación de funciones lógicas y para proporcionar subrutinas de bibliotecas con funciones básicas del sistema.

En un computador, se utiliza memoria ROM cuando se desea que su contenido esté disponible permanentemente (es decir, sin que sea necesario leerlo previamente desde un dispositivo de almacenamiento externo). Éste es el caso de la BIOS, que contiene el código necesario para controlar los dispositivos de entrada/salida de un computador personal.

Desde el punto de vista funcional, una memoria ROM es un circuito combinacional cuya entrada es la dirección de memoria que se desea leer y cuya salida es el dato almacenado en la dirección especificada. Para indicar la dirección que se desea leer, se utilizan n entradas ($A_{n-1} \cdots A_0$ —***A****ddress*—) y, para proporcionar el dato leído, m salidas ($D_{m-1} \cdots D_0$ —***D****ata*—).

Puesto que el contenido de cada una de las posiciones de memoria es fijo, se puede decir que una ROM de 2^n palabras de 1 bit (ROM $2^n \times 1$), equivale a un multiplexor $2^n : 1$ con cada una de sus entradas de datos conectadas permanentemente a un nivel lógico determinado; las entradas de control del multiplexor permiten direccionar cuál de sus entradas proporciona el valor de la salida.

La Figura 6.2 muestra la estructura interna de una ROM $2^n \times 1$. Como puede verse, las entradas de dirección, $A_{n-1} \cdots A_0$, de la memoria están conectadas a las entradas de control del multiplexor, $C_{n-1} \cdots C_0$, y la salida del multiplexor, S_0, es la salida de datos de la memoria, D_0. La entrada de habilitación del multiplexor (*Enable*) se utiliza para habilitar o no la memoria.

Para la construcción de una memoria ROM $2^n \times m$ basta con utilizar m multiplexores de la siguiente forma: las entradas de dirección, $A_{n-1} \cdots A_0$ se conectan con las entradas de control de todos los multiplexores. Así, las salidas de los multiplexores forman las salidas de datos, $D_{m-1} \cdots D_0$, de esta memoria.

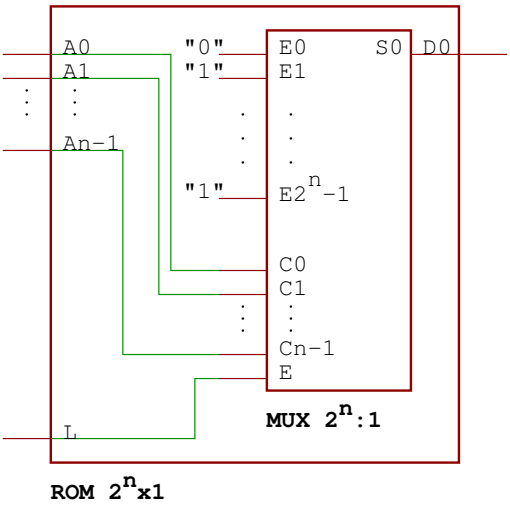


Figura 6.2: Estructura interna de una ROM $2^n \times 1$

Memoria RAM

La memoria RAM (*Random Access Memory*) es una memoria de acceso aleatorio de lectura/escritura. Es decir, la información que contiene una memoria RAM puede ser modificada.

Además, la memoria RAM es volátil —necesita ser alimentada para mantener la información— y puede ser dinámica (DRAM —*Dynamic RAM*—), o estática (SRAM —*Static RAM*—).

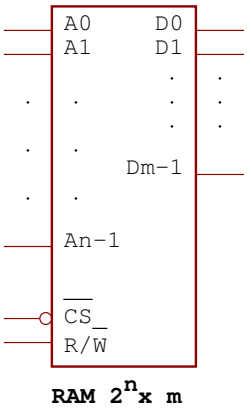


Figura 6.3: Esquema de una memoria RAM $2^n \times m$

Desde el punto de vista funcional, una memoria RAM de 2^n palabras de m bits (RAM $2^n \times m$), posee las siguientes entradas y salidas: una entrada de selección de la operación a realizar, lectura o escritura, $R/\overline{W}$ ($Read/\overline{Write}$); n entradas de dirección, $A_{n-1} \cdots A_0$, mediante las cuales se indica la dirección de memoria a la que se desea acceder, ya sea para leer o para escribir en ella; m líneas de datos, $D_{m-1} \cdots D_0$, que son tanto de entrada como de salida (de entrada si se escribe un dato y de salida

si se lee); y una entrada de selección de la memoria, $\overline{CS}$ ($\overline{Chip\ Select}$), utilizada para activar la memoria. La Figura 6.3 presenta el esquema de una memoria RAM con las entradas y salidas mencionadas.

En cuanto al diseño de una memoria RAM, ésta se puede estructurar en dos bloques: una matriz de *celdas de memoria*, donde cada celda de memoria almacena uno de los bits de la memoria, y un *circuito de control* encargado de la selección de aquellas celdas de memoria que intervienen en la operación en curso. Las memorias DRAM, además, disponen de un circuito de refresco encargado de la regeneración de la información que contienen las celdas de memoria.

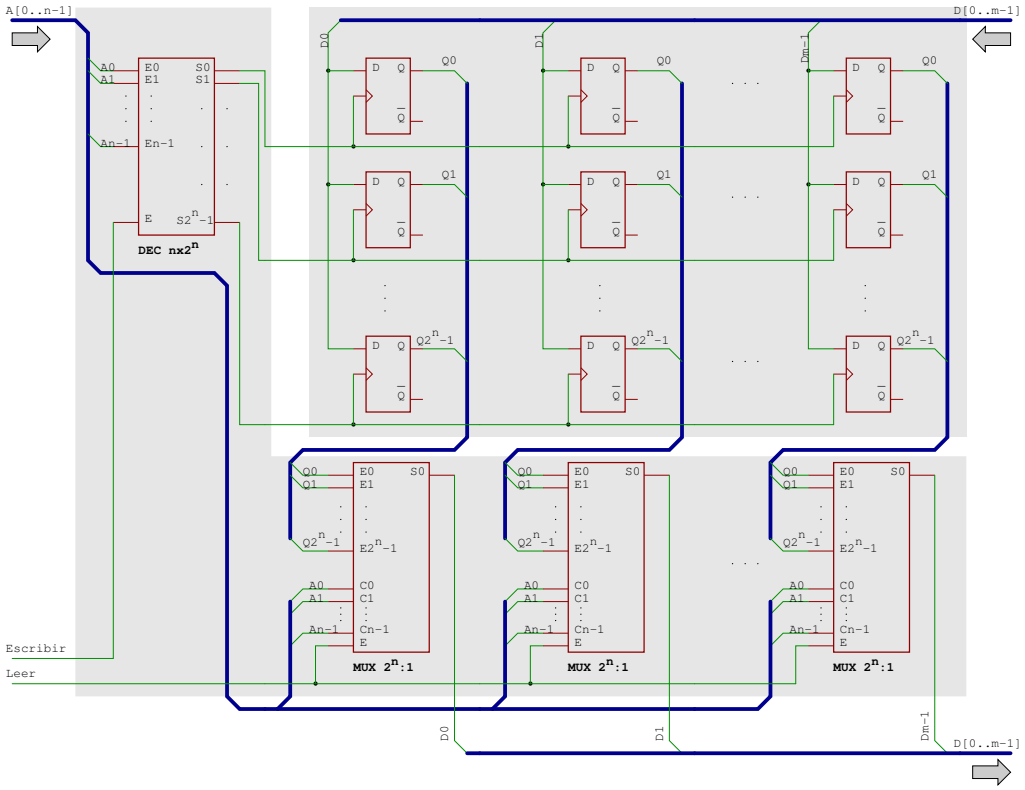


Figura 6.4: Estructura interna de una memoria RAM $2^n \times m$

La Figura 6.4 muestra una posible implementación de una memoria RAM $2^n \times m$ estática. Esta implementación presenta las siguientes entradas y salidas: n entradas de dirección ($A_{n-1} \cdots A_0$), m entradas de datos ($D_{m-1} \cdots D_0$ —parte superior del diagrama—), 2 líneas de control (*Escritura* y *Lectura*) y m salidas de datos ($D_{m-1} \cdots D_0$ —parte inferior del diagrama—).

Las líneas de *Escritura* y *Lectura* controlan el funcionamiento de la memoria. Siendo las combinaciones válidas de las mismas: *Escritura* y *Lectura* a 0 (memoria no seleccionada), *Escritura* a 1 y *Lectura* a 0 (modo de escritura) y *Escritura* a 0 y *Lectura* a 1 (modo de lectura).

Para realizar una lectura de dicha memoria, se indica la dirección que se desea leer en las entradas de dirección ($A_{n-1} \cdots A_0$) y se selecciona la operación de lectura (*Lectura* = 1 y *Escritura* = 0). El proceso por el que se lee el dato es el siguiente. Como la entrada *Lectura* está conectada a las entradas *Enable* de los m multiple-

xores, cuando se pone a '1' la entrada *Lectura*, éstos se activan. Entonces, la salida de cada uno de ellos, toma el valor de sus entradas $E_{A_{n-1}\cdots A_0}$. Estas entradas están conectadas a las salidas de los biestables D situados en la fila $A_{n-1}\cdots A_0$ y puesto las salidas de los multiplexores son la salida de datos de la memoria ($D_{m-1}\cdots D_0$ —parte inferior del diagrama—), la salida de la memoria es, por tanto, el dato almacenado en los biestables situados en la fila $A_{n-1}\cdots A_0$.

Para realizar una escritura en dicha memoria, se indica el dato que se desea escribir ($D_{m-1}\cdots D_0$ —parte superior del diagrama—), la dirección de memoria en la que se quiere almacenar dicho dato y se selecciona la operación de escritura ($Escritura = 1$ y $Lectura = 0$). El proceso por el que se almacena el dato es el siguiente. Como la entrada *Escritura* está conectada a la entrada *Enable* del decodificador, cuando se pone a '1' la entrada *Escritura*, éste se activa. Entonces, la salida $S_{A_{n-1}\cdots A_0}$ del decodificador pasa de '0' a '1' provocando un flanco de subida en la entrada de reloj de los biestables D situados en la fila $A_{n-1}\cdots A_0$. Entonces, cada uno de dichos biestables almacena el valor proporcionado en su entrada. Como las entradas de los biestables son las entradas de datos de la memoria, el dato de entrada se almacena (*escribe*) en los biestables D de la fila $A_{m-1}\cdots A_0$.

El circuito propuesto, para que tuviera las mismas entradas y salidas presentadas en el esquema de memoria RAM de la Figura 6.3, debería tener un único bus de datos $D_{m-1}\cdots D_0$ de entrada y salida y las señales de control deberían ser $R/\overline{W}$ y $\overline{CS}$.

Unificación de los buses de entrada y salida

Para que un mismo bus de datos se pueda utilizar alternativamente como bus de entrada o de salida de datos, no basta con unir eléctricamente éste simultáneamente a la entrada y a la salida de datos, ya que, de esta forma, se daría lugar a circuitos mal formados.

Es necesario, por tanto, utilizar un circuito que aisle el bus de la entrada o de la salida dependiendo del modo de funcionamiento de la memoria. Así, al realizar una operación de lectura, el bus debería estar conectado con la salida y aislado de la entrada de datos y, por el contrario, al realizar una operación de escritura, el bus debería estar conectado con la entrada y aislado de la salida de datos. Es más, si la memoria no está seleccionada, es conveniente que el bus quede aislado tanto de su entrada como de su salida de datos.

Aislar el bus de datos de una memoria no utilizada, permite que varias memorias puedan conectarse directamente a un mismo bus de datos.

La Figura 6.5 muestra cómo se pueden unificar los buses de entrada y salida utilizando *drivers* triestado controlados mediante las señales de *Escritura* y *Lectura*.

Un *driver* triestado es un *driver* que puede presentar en su salida, además de los valores '0' y '1', un estado llamado de *alta impedancia*. Cuando su salida es de alta impedancia, no hay conexión eléctrica entre su entrada y su salida.

Las señales de control $\overline{CS}$ y $R/\overline{W}$

La conversión entre las señales $\overline{CS}$ y $R/\overline{W}$ a *Escritura* y *Lectura* es sencilla sin más que tener en cuenta la relación existente entre las mismas. La Figura 6.6 presenta esta relación en forma de tabla de verdad (donde las señales $R/\overline{W}$ y $\overline{CS}$ son las entradas de la tabla y las señales *Escritura* y *Lectura* son las salidas). La misma figura presenta un circuito que implementa dicha tabla de verdad.

La Figura 6.7 representa el esquema completo de una memoria RAM (partiendo del esquema anterior) con un bus bidireccional de datos y entradas de control $\overline{CS}$ y $R/\overline{W}$.

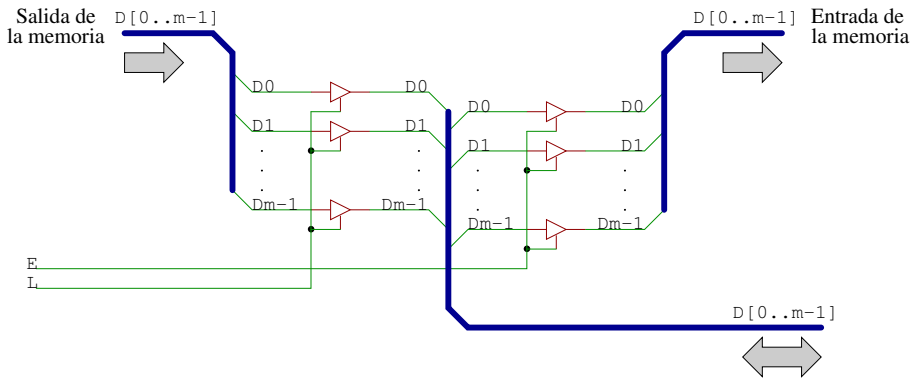


Figura 6.5: Unificación de los buses de datos

$\overline{CS}$	$R/\overline{W}$	*Escritura*	*Lectura*
0	0	1	0
0	1	0	1
1	0	0	0
1	1	0	0

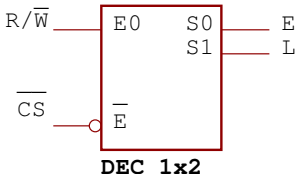


Figura 6.6: Conversión de las señales de control

6.4. Asociación de memorias

La asociación de memorias permite conseguir memorias de una determinada capacidad utilizando memorias de menor capacidad.

En las dos secciones siguientes se verá cómo asociar memorias para incrementar el tamaño de palabra (extensión del tamaño de palabra) y cómo asociar memorias para incrementar el número de palabras (extensión del número de palabras).

Para aumentar tanto el ancho de palabra como el número de palabras, basta con primero asociar las memorias para obtener nuevas memorias con el ancho de palabra deseado y, posteriormente, asociar estas últimas para conseguir el número de palabras requerido.

Extensión del tamaño de palabra

Consiste en obtener una memoria de un cierto tamaño de palabra, x, mediante la asociación de memorias de un tamaño de palabra, m, menor al deseado. Las memorias con las que se realiza la asociación poseen el mismo número de palabras que la memoria a implementar.

Para ello, basta con utilizar tantas memorias ($\lceil \frac{x}{m} \rceil$) como sean necesarias para que la la suma de sus líneas de datos sea igual (o superior) al número de líneas de datos de la memoria final.

Una vez determinado el número de memorias necesarias, el bus de datos de la primera de ellas se asigna a las m primeras líneas del bus de datos resultante, el de

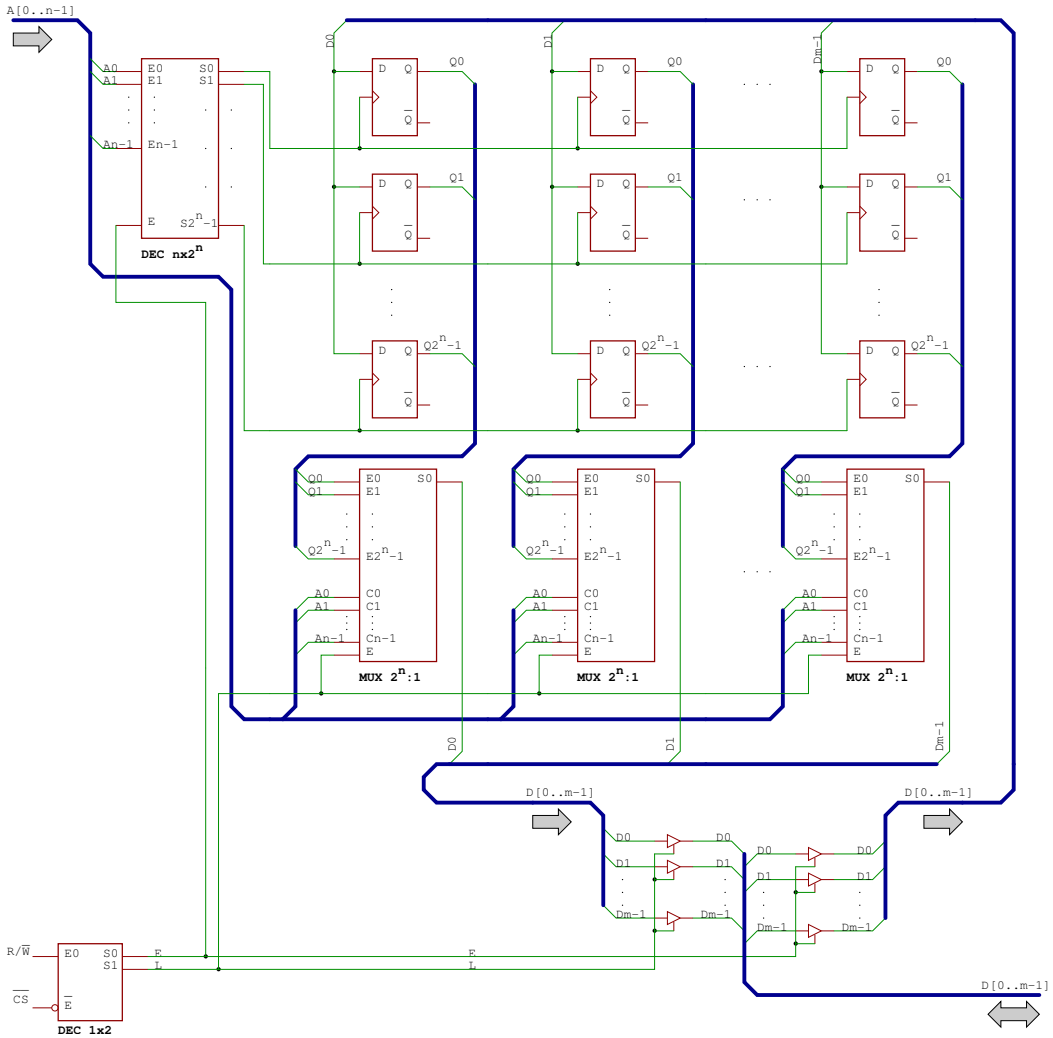


Figura 6.7: Estructura interna de una memoria RAM $2^n \times m$

la segunda a las m segundas líneas, y así sucesivamente, hasta completar el número total x de líneas del bus de datos deseado.

Puesto que el bus de datos de la nueva memoria está formado por los buses de datos de las memorias utilizadas, cualquier operación sobre esta memoria, ya sea de lectura o de escritura, ha de realizarse de forma simultánea en todas las memorias que la forman. Para que así sea, basta con conectar el bus de direcciones y las entradas de control $\overline{CS}$ y $R/\overline{W}$ de la nueva memoria a las entradas de direcciones y a las entradas $\overline{CS}$ y $R/\overline{W}$, respectivamente, de todas las memorias.

La Figura 6.8 muestra la asociación de memorias de $1K \times 4$ para obtener una memoria de $1K \times 8$.

Extensión del número de palabras

Consiste en obtener una memoria con un determinado número de palabras asociando memorias con un número de palabras inferior al deseado. Las memorias a asociar poseen el mismo tamaño de palabra que la memoria a implementar.

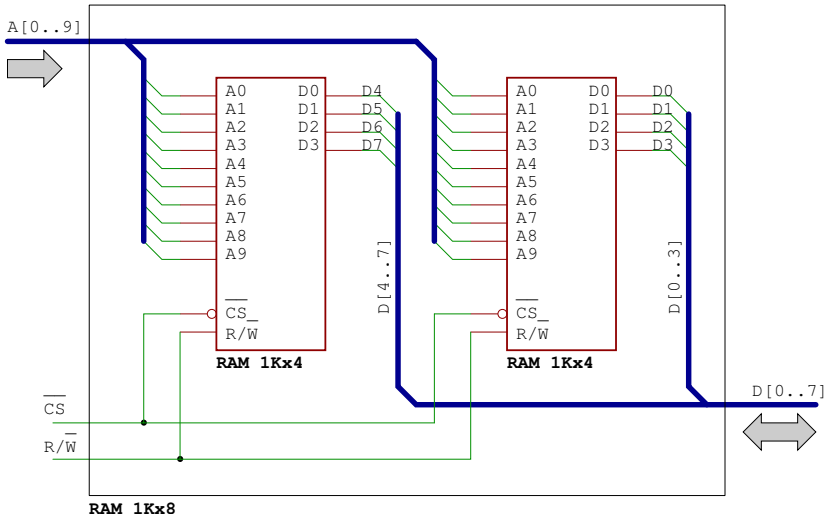


Figura 6.8: Memoria de $1K \times 8$ utilizando memorias $1K \times 4$

Para ello, basta con repartir el conjunto de direcciones total entre todas las memorias disponibles, de tal forma que cada una de las memorias proporcione un subrango de direcciones del total.

Puesto que la posición a leer o escribir puede pertenecer a cualquiera de las memorias, éstas deberán estar conectadas al bus de datos global. Por el mismo motivo, la entrada de selección de operación ($R/\overline{W}$) debe estar conectada a la entrada $R/\overline{W}$ de todas las memorias. Únicamente una de las memorias deberá estar seleccionada en un momento dado.

El direccionamiento de una posición de memoria se realiza, por lo tanto, en dos niveles: en un primer nivel, se selecciona aquella memoria que contiene el subrango de direcciones dentro del cual se encuentra la posición de memoria deseada; y en un segundo nivel, se indica la dirección dentro de la memoria seleccionada en la que se encuentra el dato.

Para la implementación del primer nivel, se puede utilizar un circuito que, tomando como entrada tantas líneas de mayor peso del bus de direcciones como sean necesarias, active la memoria correspondiente (y desactive el resto). Además, este circuito no deberá activar ninguna memoria si la entrada $\overline{CS}$ del conjunto no está activa.

Para la implementación del segundo nivel, selección de la posición dentro de una de las memorias, basta con asignar las líneas del bus de direcciones que no han intervenido en el proceso de selección de dicha memoria a su bus de direcciones.

La Figura 6.9 muestra la asociación de memorias $1K \times 8$ para obtener una memoria de $2K \times 8$.

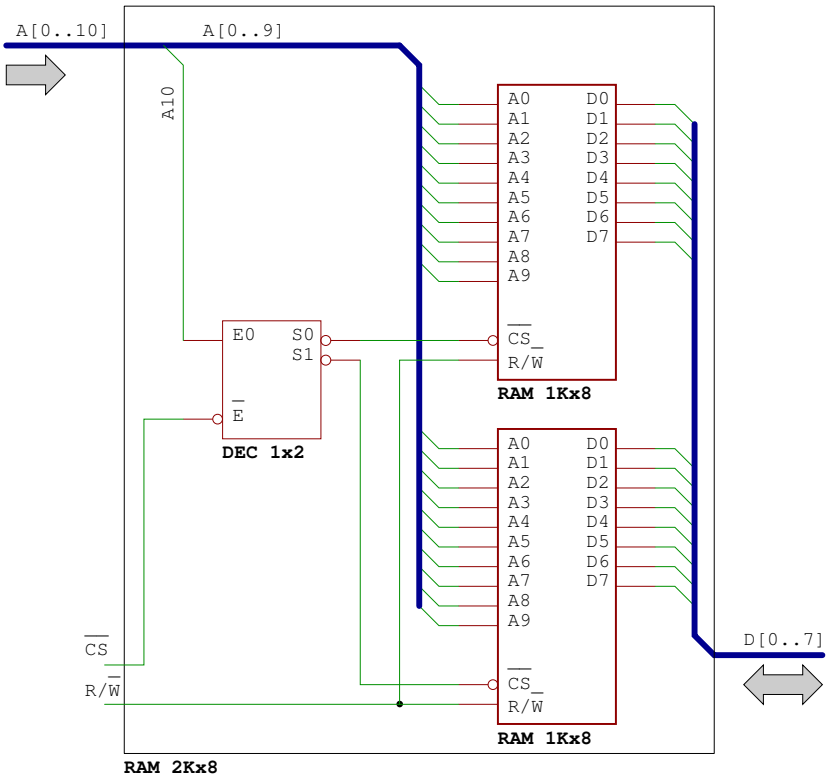


Figura 6.9: Memoria de $2K \times 8$ utilizando memorias $1K \times 8$

CAPÍTULO 7

Unidad Central de Proceso

La función de la Unidad Central de Proceso (UCP o *CPU*) es la de ejecutar los programas almacenados en la memoria principal del computador, leyendo las instrucciones, decodificándolas y realizando las acciones asociadas para cada una de ellas[Tan92]. En definitiva, es quien gobierna y coordina el funcionamiento de todos las unidades funcionales de un computador.

7.1. Estructura de la Unidad Central de Proceso

La UCP está estructurada en los siguientes bloques funcionales:

Unidad de control (UC o *CU*) Se encarga de interpretar las instrucciones y de la activación de las señales de control del resto de dispositivos del computador (incluidos los de la UCP) para provocar el funcionamiento ordenado del computador.

Unidad Aritmético Lógica (UAL o *ALU*) Un computador, tal y como se vió en el Capítulo 1, es capaz de procesar información realizando una serie de operaciones sobre la misma. La UAL es la encargada de realizar estas operaciones, que básicamente son de dos tipos: aritméticas (suma, resta, multiplicación, ...) y lógicas (AND, OR, NOT, ...).

Registros Se utilizan para el almacenamiento temporal de información y constituyen la memoria más rápida de la que dispone un computador (recuérdese que los registros están en la cúspide de la jerarquía de memorias —véase Capitulo 6—). Los registros pueden clasificarse según su función en:

Registros de propósito general. La información almacenada en los mismos no tiene un significado a priori. Estos registros pueden servir para almacenar datos exclusivamente, direcciones de memoria exclusivamente o datos y direcciones de memoria indistintamente.

Registros específicos. Estos registros realizan una función determinada, es decir, la información que almacenan tiene un significado concreto para

el funcionamiento de la UCP. Registros específicos son, por ejemplo, el *Contador de Programa* (*PC - Program Counter*), el *Registro de Estado* (*SR - State Register*), el *Puntero de Pila* (*SP - Stack Pointer*), ...

Tamaño de un procesador

Como se vió en el Capítulo 1, la comunicación de la UCP con el resto de los componentes de un computador se realiza por medio de una serie de buses, que en conjunto reciben el nombre de *bus del sistema*. De entre estos buses, el bus de datos, es quien determina la unidad de transferencia (tamaño de palabra) entre la UCP y el resto de las unidades funcionales de un computador.

Por otro lado, la UCP posee a su vez un bus interno cuyo ancho (número de líneas) viene determinado por el tamaño de palabra de sus registros.

El tamaño de ambos buses, de datos y de registros, que no tienen por qué ser iguales, determinan la capacidad de tratamiento de información de un procesador. Se suele adoptar la convención de que sea el ancho del bus de datos el que se utilice como descriptor del tamaño de una UCP [Hyd96].

Ruta de datos

Tradicionalmente, los circuitos capaces de realizar las operaciones aritméticas y lógicas se agrupan en la UAL, que constituye un bloque funcional independiente. Esto presenta el inconveniente de que la UC tiene que enviar a la UAL los operandos y la operación a realizar con los mismos y esperar a que ésta, cuando la operación termine, le devuelva el resultado de la operación.

Para acelerar la realización de las operaciones aritméticas y lógicas, y especialmente con la aparición de la filosofía RISC (véase la Sección 7.6), se tiende a la descentralización de los circuitos encargados de estas operaciones para integrarlos en lo que se conoce como la *ruta de datos* de la UCP.

La ruta de datos está formada por los circuitos combinacionales y secuenciales necesarios para ejecutar las distintas instrucciones de máquina del procesador. Dentro de la ruta de datos se incluyen, si una determinada instrucción requiere el cálculo de una operación aritmética o lógica, los circuitos necesarios para realizar dicha operación.

7.2. Ciclo de trabajo

Como se ha visto anteriormente, dentro de la UCP, la unidad de control se encarga de la generación de las señales de control adecuadas para que el computador ejecute una serie de instrucciones máquina (programa). Para ello, la UC realiza un conjunto de acciones que se pueden expresar en forma de un ciclo que se repite indefinidamente y que recibe el nombre de *ciclo de trabajo*. El ciclo de trabajo de un computador está compuesto por las siguientes fases:

1. **Búsqueda de la instrucción**. Esta fase consiste en la lectura de memoria de la instrucción que se tiene que ejecutar. La dirección de memoria de la instrucción que se tiene que ejecutar, está almacenada en el PC (contador de programa). Una vez se ha leído la instrucción, el PC se incrementa para apuntar a la dirección de memoria en la que se encuentra la siguiente instrucción.

2. **Interpretación de la instrucción**. En esta fase se determina qué líneas de control de la UC han de ser activadas y el orden en el que éstas deben activarse para que se lleven a cabo todas las acciones que conlleva la ejecución de la instrucción leída en la fase anterior.

3. **Ejecución de la instrucción**. Durante esta fase se realizan las siguientes acciones: se recuperan los operandos que requiere la instrucción (si ésta requiere operandos); se activan las señales de control en el orden que se determinó en la fase anterior; y se almacena el resultado de la ejecución de la instrucción (si la instrucción genera algún resultado).

Una vez completada la última fase, la unidad de control comienza otro ciclo de trabajo, y así indefinidamente.

7.3. Excepciones e interrupciones

Pese a que el computador ejecuta la secuencia anterior de forma indefinida, hay ocasiones en las que es interesante la ruptura de dicha secuencia.

Las *excepciones* e *interrupciones* son eventos que, sin ser ni bifurcaciones ni saltos del programa, cambian el flujo normal de ejecución de instrucciones. Una excepción es un evento inesperado generado en la misma unidad central de proceso; por ejemplo, un desbordamiento tras una operación aritmética. Una interrupción es un evento que también causa un cambio inesperado en el flujo de control pero que proviene del exterior de la UCP, es decir, no es generada por la misma UCP. Las interrupciones son utilizadas por los dispositivos de E/S para comunicarse con la UCP. Cualquiera de estas situaciones, tanto las que provocan excepciones como las que provocan interrupciones, requieren que el procesador las atienda de forma inmediata.

Muchas arquitecturas y autores no distinguen entre excepciones e interrupciones, utilizando habitualmente el término antiguo de interrupción para referirse a ambas.

Cuando se produce una interrupción, la UCP interrumpe el programa en curso, ejecuta una rutina de tratamiento de dicha interrupción y después retoma la ejecución del programa interrumpido de tal forma que todo vuelve a estar como antes de que se atendiera la interrupción. En otras palabras, el programa interrumpido no se ve afectado por la interrupción.

Puede ocurrir que mientras se está atendiendo a una interrupción (o una excepción) se produzca otra. Para determinar qué interrupciones pueden ser interrumpidas por otras, las interrupciones se organizan por niveles de prioridad, y se utiliza un registro especial para almacenar el nivel de prioridad de la interrupción en curso y evitar, así, atender interrupciones de menor o igual prioridad.

7.4. Formato de instrucción

Las instrucciones que puede ejecutar un procesador tienen que ser expresadas en el lenguaje que entienden los computadores digitales, es decir, como cadenas de ceros y unos. En principio, la correspondencia entre cada una de las instrucciones máquina de un procesador y una determinada cadena de ceros y unos podría ser arbitraria, siempre que se garantizara que instrucciones distintas utilizan cadenas diferentes. En la práctica, proceder de esta forma, dificultaría el diseño de la parte de la UC encargada de la *decodificación de la instrucción*, por lo que es conveniente codificar las instrucciones de tal forma que este circuito sea lo más sencillo posible.

Para facilitar la decodificación de las instrucciones, se divide en campos (cadenas contiguas de bits) el número de bits destinados a almacenar una instrucción y en cada uno de estos campos se almacena un determinado tipo de información. La información que contiene cada campo puede ser:

- La operación a realizar.
- Los operandos fuente o la dirección de los mismos.
- La dirección del operando destino (donde se almacena el resultado).

Como se puede ver, existen dos tipos básicos de campos: de operación (que especifican la operación a realizar) y de dirección (que especifican la ubicación de los operandos). En cuanto a la dirección de los operandos, ésta se puede especificar de diversas maneras dando lugar a los distintos modos de direccionamiento de un procesador. Los modos de direccionamiento se describen con más detalle en el siguiente apartado.

La organización de los campos —el orden y el tamaño de los mismos— de un conjunto de instrucciones se especifica mediante los denominados *formatos de instrucción*. Como no todas las instrucciones requieren el mismo tipo de información, un mismo procesador suele presentar formatos de instrucción distintos para optimizar el espacio destinado al almacenamiento de las instrucciones en función de los requerimientos de las mismas. Más adelante, en la Sección 7.7, se muestran los formatos de instrucción de un procesador real.

Modos de direccionamiento

La ubicación de los operandos —datos o direcciones de memoria— de una instrucción es parte de la información indicada en la misma. Los operandos pueden estar almacenados en:

- la propia instrucción,
- un registro, o
- en la memoria principal.

Así pues, para especificar el operando que se quiere utilizar, bastaría con codificar en la instrucción el operando directamente, o bien el nombre del registro, o bien la dirección de memoria donde se encuentra el operando. Sin embargo, parece conveniente disponer de otras formas más elaboradas de indicar la dirección de los operandos, debido a:

- Ahorro de espacio: cuanto más cortas sean las instrucciones, menos espacio ocuparán en memoria, por lo que será aconsejable que la forma de indicar la dirección de los operandos ocupe el menor espacio posible.
- Acceso a estructuras de datos: el manejo de estructuras de datos complejas (matrices, tablas, colas, listas, etc.) se simplifica con el empleo de formas suplementarias de indicar la dirección de los operandos.

- Código reubicable: si la dirección de los operandos sólo se pudiera expresar como una dirección de memoria fija, cada vez que se ejecutara un determinado programa, éste buscaría los operandos en las mismas direcciones de memoria, por lo que tanto el programa como sus datos habrían de utilizar siempre las mismas direcciones de memoria. ¿Qué pasaría entonces con el resto de programas que puede ejecutar el computador? ¿También tienen direcciones de memoria reservadas para ellos? ¿Cuántos programas distintos podría ejecutar un computador sin que estos se solaparan? ¿De cuánta memoria dispone el computador? Queda claro que es conveniente poder indicar la dirección de los operandos de tal forma que un programa pueda ejecutarse sin depender de la zona de memoria en la que se encuentre.

Se denomina *modos de direccionamiento* a las distintas formas en las que se puede indicar la *dirección efectiva* de los operandos de una instrucción, es decir, dónde se encuentran realmente los operandos (la dirección efectiva no tiene por qué ser una dirección de memoria). Los principales modos de direccionamiento se presentan a continuación:

Inmediato La dirección efectiva del operando es la instrucción actual. Es decir, la instrucción contiene el valor del operando.

Directo La dirección efectiva del operando es una dirección de memoria. La instrucción proporciona la dirección de memoria del operando.

Indirecto La dirección efectiva del operando es la dirección de memoria indicada en una dirección de memoria. La instrucción proporciona la dirección de memoria, no del operando, sino de la dirección de memoria del operando.

Directo a registro La dirección efectiva del operando es un registro. La instrucción indica en qué registro está el operando.

Indirecto con registro La dirección efectiva del operando es la dirección de memoria indicada por un registro. La instrucción indica qué registro contiene la dirección de memoria del operando.

Con desplazamiento Son aquellos en los que hay que realizar una operación para calcular la dirección efectiva:

Relativo al *PC* La dirección efectiva del operando es el resultado de sumar la dirección de memoria almacenada en el registro PC y un desplazamiento (positivo o negativo). La instrucción proporciona el valor del desplazamiento.

Indexado La dirección efectiva del operando es el resultado de sumar una dirección de memoria y el desplazamiento contenido en un registro. La instrucción proporciona la dirección de memoria y el nombre del registro que contiene el desplazamiento.

7.5. Lógica cableada y lógica microprogramada

Para el diseño de la UC se pueden utilizar dos metodologías que reciben los nombres de lógica cableada y lógica microprogramada, respectivamente. La principales características de ambas son:

Lógica cableada La UC se diseña como un circuito digital convencional utilizando, por ejemplo, puertas lógicas. Las características que poseen las UC diseñadas utilizando la lógica cableada se citan a continuación:

- Su diseño y puesta a punto son laboriosos y costosos.
- Para su modificación suele ser necesario volver a diseñarlas por completo.
- En igualdad de condiciones, son más rápidas que las diseñadas utilizando la técnica alternativa: la lógica microprogramada. Debido a esta característica, las UC de los grandes computadores suelen estar diseñadas utilizando lógica cableada.

Lógica microprogramada La UC se diseña utilizando una memoria de sólo lectura (ROM) en la que se almacena un conjunto de *microinstrucciones*. Cada microinstrucción, en el caso más general, está formada por tantos bits como señales de control posee la UC y el valor de éstos indica qué valor ha de tomar cada una de estas señales en un determinado período de reloj o fase. Por lo tanto, en lugar de utilizar puertas lógicas para la generación de las señales de control, se utiliza una memoria que contiene la secuencia de microinstrucciones (*microprograma*) que forman cada una de las instrucciones de máquina.

El término *microprograma* fue acuñado por M. V. Wilkes (años 50), y la técnica fue propuesta como una forma organizada y sistemática de diseño de la UC que evitaba la complejidad de la lógica cableada. Debido a los requerimientos de una memoria rápida y barata no se utilizó hasta 1964, siendo IBM el primero con los modelos de gama baja del System/360 [Sta97].

Las características que poseen las UC diseñadas utilizando la lógica microprogramada se citan a continuación:

- Su diseño y puesta a punto son más sencillos y económicos que los de la lógica cableada.
- Su modificación en la fase de diseño es relativamente sencilla. Es más, algunos procesadores permiten su modificación no sólo en la fase de diseño sino posteriormente (una vez fabricado y funcionando como parte de un computador) mediante la reescritura de su microcódigo.
- En igualdad de condiciones, son más lentas que las diseñadas utilizando lógica cableada.

El ámbito de aplicación de las unidades de control microprogramadas es el de los computadores de tamaño medio, ya que en los muy pequeños la estructura que necesita la microprogramación es demasiado costosa y en los grandes resulta demasiado lenta.

7.6. RISC y CISC

El término RISC se utiliza normalmente para referirse a un *Computador de Juego de Instrucciones Reducido* (*Reduced Instruction Set Computer*). IBM fue el pionero en la utilización de muchas de las técnicas RISC (aunque no utilizó dicho término) en su proyecto 801. Las ideas RISC provienen del computador CDC 6600, de varios proyectos de la Universidad de Berkeley (RISC I, RISC II y SOAR) y de la Universidad de Standford (proyecto MIPS).

Las principales características de los procesadores RISC son: pocas instrucciones y éstas con modos de direccionamiento sencillos; instrucciones del mismo tamaño; ejecución de una instrucción por ciclo —utilizando técnicas de segmentación; muchos registros —las operaciones suelen realizarse sobre registros, de hecho, algunos

procesadores RISC sólo acceden a memoria para recuperar o almacenar datos—; y generalmente se diseñan utilizando lógica cableada.

Pero a pesar de estas características que lo definen, el término RISC hace referencia más a una filosofía que a un conjunto de criterios de diseño. La filosofía de diseño RISC se plantea si es necesaria la implementación en el procesador de *hardware* capaz de realizar instrucciones complejas que puedan operar directamente con datos en la memoria, o si estas operaciones han de realizarse mediante *software* basado en un conjunto de instrucciones de máquina más sencillos que sólo accedan a memoria para la carga o almacenamiento de datos. El resultado, es un *chip* más sencillo que el correspondería a un equivalente CISC. La idea que subyace detrás de todo esto es que la necesidad de ejecutar un mayor número de instrucciones de máquina para realizar operaciones complejas normalmente se compensará por la mayor velocidad con la que las instrucciones pueden ejecutarse y por las mayores posibilidades de optimización de código del que dispondrán los compiladores. Por otro lado, algunas de las instrucciones más complejas que proporcionan los procesadores CISC son raramente utilizadas, por lo que la rapidez con la que éstas se realicen no influirá significativamente en el rendimiento global del procesador.

El término CISC se utiliza habitualmente para referirse a un *Computador de Juego de Instrucciones Complejo* (*Complex Instruction Set Computer*). Los computadores CISC no poseen un conjunto de particularidades de diseño que los caractericen, como es el caso en los RISC, pero muchos de los procesadores CISC actuales poseen las siguientes características: tantas instrucciones de máquina como se requieran (el objetivo último es el mapeo de cada una de las instrucciones de un lenguaje de alto nivel en una instrucción de máquina equivalente) y éstas, además, pueden utilizar un elevado número de modos de direccionamiento distintos; instrucciones de distintos tamaños (se persigue la reducción del tiempo necesario para la recuperación de las instrucciones y de los operandos que éstas requieren haciendo que aquellas instrucciones más frecuentes tengan el tamaño mas reducido posible); poseen pocos registros (ya que pueden realizar operaciones directamente sobre memoria); y generalmente se diseñan utilizando lógica microprogramada.

La filosofía CISC defiende el aumento de la velocidad con la que el procesador puede realizar operaciones complejas mediante el uso de *hardware*. Por desgracia, cuantas más instrucciones de máquina y modos de direccionamiento, más compleja se vuelve la UC, y por lo tanto, más difícil es aumentar la frecuencia de reloj de la misma. Para simplificar en la medida de lo posible el diseño de la UC y permitir el aumento de la frecuencia de reloj de la misma, se termina acudiendo a la microprogramación.

7.7. El microprocesador R2000

Características generales

El microprocesador R2000 es un procesador RISC de 32 bits. Como procesador RISC que es, posee las siguientes características:

- Juego de instrucciones reducido (alrededor de 64 instrucciones).
- Todas las instrucciones tienen la misma longitud (32 bits). Utiliza tres formatos de instrucción distintos: R, I y J.

- Pocos modos de direccionamiento: inmediato, de registro, con desplazamiento relativo al *PC* y con desplazamiento indexado.
- Arquitectura de carga/almacenamiento: las únicas instrucciones que acceden a memoria son las de lectura y escritura de datos.
- 31 registros de propósito general etiquetados como $1 al $31. Además, posee un registro adicional de sólo lectura cableado a "cero", $0, para simplificar la realización de ciertas operaciones.
- Su memoria está organizada por octetos, pudiendo direccionar hasta 4 gigaoctetos (2^{32} octetos).

Formatos de instrucción

Puesto que todas las instrucciones del R2000 ocupan 32 bits, es necesario utilizar distintos formatos de instrucción que distribuyan este espacio de la forma más adecuada al tipo y número de operandos utilizados por cada instrucción.

Los tres formatos de instrucción que posee el R2000 son:

- *Formato R.* Se utiliza para aquellas instrucciones cuyos operandos son registros y datos inmediatos pequeños.
- *Formato I.* Se utiliza para aquellas instrucciones que requieren datos inmediatos con un rango de valores mayor al proporcionado por el formato R.
- *Formato J.* Se utiliza para las instrucciones de salto.

Todas las instrucciones, independientemente de su formato, poseen un campo llamado *código de operación* que abarca los 6 bits de mayor peso de la instrucción. El contenido de este campo determina, además de la instrucción en sí, el formato de instrucción de la misma.

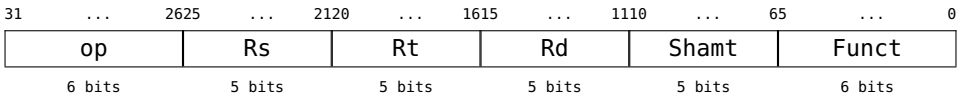
Formato R

El formato R se utiliza para aquellas instrucciones que requieren tres operandos (dos operandos fuente y un destino). El modo de direccionamiento de los tres es el directo a registro.

Puesto que el R2000 posee 32 registros, son suficientes 5 bits para la codificación de los mismos. Por lo tanto, este formato de instrucción requiere 3 campos de 5 bits cada uno para la codificación de qué tres registros se utilizan para almacenar los dos operandos fuente y el destino.

Los 11 bits restantes, se reparten entre dos campos. El primero de ellos (5 bits) se utiliza como un dato inmediato en algunas de las instrucciones que utilizan este formato y el segundo (6 bits) se utiliza para especificar una variante de la operación indicada por el campo *código de operación*.

En definitiva, el formato R presenta la siguiente disposición:



Donde,

- *op*: es el *código de operación*.
- *Rs*: es el primer *registro fuente*.
- *Rt*: es el segundo *registro fuente* (*transformable*).
- *Rd*: es el *registro destino*.
- *Shamt*: es el número de bits a desplazar[1] (*shift amount*).
- *Funct*: selecciona una variante de la operación indicada por el campo *código de operación*.

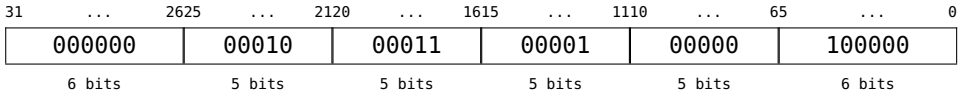
Las instrucciones que utilizan este formato realizan operaciones de uno de estos tipos:

$$Rd = \text{operación}(Rs, Rt) \text{ ó } Rd = \text{operación}(Shamt, Rt)$$

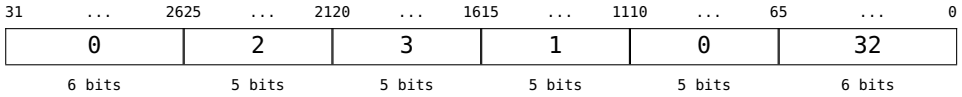
Como ejemplos de instrucciones que utilizan el formato R se van a ver: `add Rd,Rs,Rt` y `sll Rd, Rt, Shamt`.

La primera de ellas, `add Rd,Rs,Rt`, suma las palabras (de 32 bits) almacenadas en los registros `Rs` y `Rt` y almacena el resultado en el registro `Rd`. Para ver un caso concreto, la instrucción `add $1,$2,$3`, que suma los datos almacenados en los registros `$2` y `$3` y almacena el resultado en el registro `$1`, se codifica de la siguiente forma:

`add Rd,Rs,Rt`
`add $1,$2,$3`

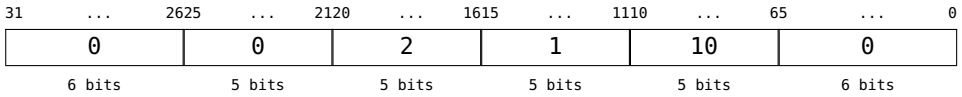


que si se expresa en decimal:



La segunda de las instrucciones dadas como ejemplo del formato R, `sll Rd, Rt, Shamt`, desplaza el contenido del registro `Rt` tantos bits a la izquierda como indica el campo `Shamt` y almacena el resultado en el registro `Rd`. La instrucción `sll $1,$2,10`, que desplaza 10 bits a la izquierda el contenido del registro `$2` y almacena el resultado en el registro `$1`, se codifica de la siguiente forma:

`sll $1,$2,10`

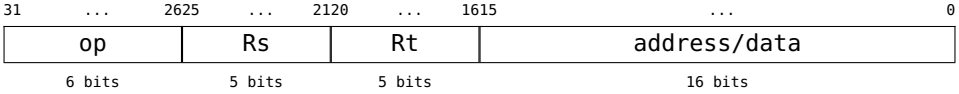


[1] Las operaciones de desplazamiento de bits desplazan los bits de *Rt* un determinado número de posiciones (hacia la izquierda o hacia la derecha). El número de posiciones a desplazar se puede indicar por medio de un registro, *Rs*, o por medio del campo `Shamt` (de ahí su nombre).

Formato I

Como se ha visto en la descripción del formato R, las instrucciones de dicho formato disponen de un campo de 5 bits (*Shamt*) en el que se puede indicar un dato inmediato. Si el valor almacenado en este campo es un número sin signo, el rango de valores posibles del mismo va desde 0 hasta 31. Para ciertas operaciones, es conveniente poder especificar uno de los operandos en la misma instrucción, sin embargo, el rango de valores proporcionado por el campo `Shamt` del formato R es demasiado pequeño. Puesto que se pretende que todas las instrucciones tengan el mismo tamaño, una solución al problema pasa por la definición de otro formato que disponga de un menor número de campos y proporcione uno de ellos con un mayor número de bits destinado a almacenar en el mismo, un dato inmediato. El formato I elimina los campos `Rd`, `Shamt` y `Funct` y en su lugar proporciona un único campo de 16 bits que se denomina `address/data`.

La asignación de campos en el formato I queda:



Donde,

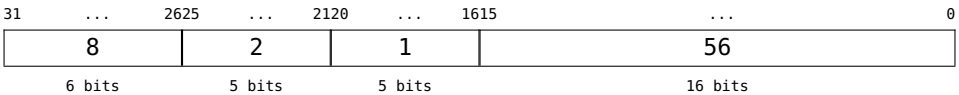
- *op*: *código de operación.*
- *Rs*: primer *registro fuente.*
- *Rt*: *registro transformable* (fuente o destino dependiendo de la instrucción).
- *address/data*: dirección de memoria o dato inmediato.

Las instrucciones que utilizan este formato son:

1. Instrucciones de 3 operandos, de los cuales uno es un dato inmediato, *data*, (modo de direccionamiento inmediato) y los otros dos son registros.
2. Instrucciones de 3 operandos, de los cuales, dos son registros y el tercero es la dirección de memoria de una instrucción. Esta dirección se expresa como relativa al PC: *address**4+PC (modo de direccionamiento relativo a PC).
3. Instrucciones de 2 operandos, uno de los cuales es *Rt*, y el otro está en la memoria principal. La dirección de éste es relativa al *Rs*: address+*Rs* (modo de direccionamiento indexado).

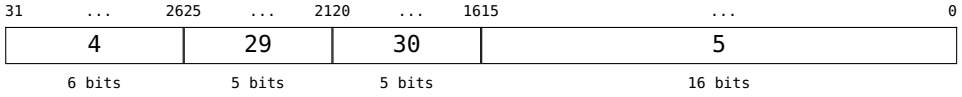
Como ejemplo del primer grupo (3 operandos: dos registros y un dato inmediato), está la instrucción `addi Rt,Rs,Imm`. Esta instrucción suma el contenido del registro `Rs` y el dato inmediato `Imm` y almacena el resultado en el registro `Rt`.

```
addi Rt,Rs,Imm
addi $1,$2,56
```



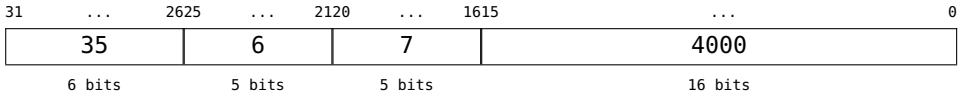
Como ejemplo del segundo grupo (3 operandos: dos registros y una dirección inmediata) está `beq Rs,Rt,label` (***b**ranch if* eq*ual*, bifurcar si igual), que compara el contenido de los registros `Rs` y `Rt` y en el caso de que éstos sean iguales, rompe la secuencia de ejecución de instrucciones y salta a la instrucción contenida en la dirección de memoria indicada por la expresión `label`$*4+$`PC`. La instrucción `beq $29,$30,5`, que compara el contenido de los registros `$29` y `$30` y salta 5 instrucciones hacia adelante si son iguales, se codifica como:

`beq Rs,Rt,label`
`beq $29,$30,5`

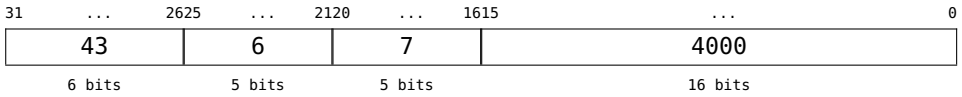


Como ejemplos del tercer grupo de instrucciones (2 operandos: registro y dato inmediato) están las instrucciones `lw Rt,Imm(Rs)` y `sw Rt,Imm(Rs)`. `lw Rt,Imm(Rs)` lee un dato de la dirección de memoria $Imm+(Rs)$ y lo almacena en `Rt` y `sw Rt,Imm(Rs)` almacena el dato contenido en `Rt` en la dirección de memoria $Imm + (Rs)$.

`lw Rt,Imm(Rs)`
`lw $7,4000($6)`

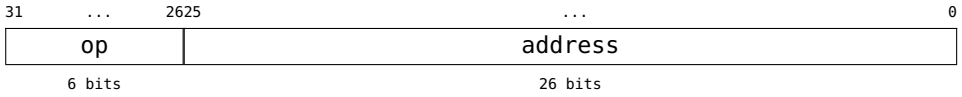


`sw Rt,Imm(Rs)`
`sw $7,4000($6)`



Formato J

Pertenecen a este formato las instrucciones de salto incondicional (`j` y `jal`). Estas instrucciones provocan un salto en la ejecución del programa. La dirección de este salto se indica en la propia instrucción de forma absoluta en un campo de 26 bits denominado `address`. Así, la distribución de campos de este formato es la siguiente:



Puesto que la dirección de salto hace referencia a una instrucción, y la dirección de memoria en la que puede encontrarse una instrucción ha de ser múltiplo de 4 en el R2000, la dirección de memoria a la que se salta viene dada por la expresión $address \times 4$. Es decir, al ejecutar una instrucción de salto, se escribe en el contador de programa el siguiente valor:

Como se puede observar, debido a los 6 bits utilizados para la identificación de la instrucción, las instrucciones de salto del R2000 sólo pueden indicar un destino situado dentro de los primeros $2 * 28$ bytes del mapa de memoria.

CAPÍTULO 8

Unidad de Entrada/Salida

8.1. Introducción

El objetivo de la unidad de Entrada/Salida (E/S) es la conexión entre el resto de unidades funcionales de un computador, primordialmente entre la Unidad Central de Proceso (CPU) y los dispositivos periféricos. Esta conexión presenta una serie de características [Mig94]:

- Los periféricos presentan velocidades de transmisión que van desde unos pocos bytes por segundo hasta millones de bytes por segundo.
- El ancho de palabra de los periféricos suele ser de un byte, lo que no coincide con el ancho de palabra de la mayoría de los computadores actuales.
- Algunos periféricos son sólo de lectura, otros sólo de escritura y otros permiten tanto la lectura como la escritura.

El problema de intercambio de información entre CPU y periféricos es bastante complejo, abarcando [Mig94]:

- El establecimiento de un mecanismo para transmitir la información, para lo que se debe tener en cuenta: el direccionamiento o selección del periférico, la forma de establecer el camino para el envío de datos, la posible conversión serie/paralelo, la conversión de códigos, etc.
- El establecimiento de un mecanismo de control que determine el origen y el destino de la información, la cantidad a transmitir, los códigos de protección, etc.

Estos mecanismos se reparten entre el controlador del periférico, la unidad de control de la CPU y los programas de entrada/salida.

Por todo ello, conviene diferenciar entre lo que es: transferencia elemental de información, transferencia de bloque y operación de entrada/salida. La *transferencia*

elemental tiene por objeto el envío o la recepción de una única unidad de información, ya sea esta un dato o una palabra de control. La *transferencia de bloque* se encarga de enviar o recibir un bloque de información, como puede ser un sector de un disco o un registro de una cinta. La *operación de E/S* tiene por objeto el proceso global de transferencia de un conjunto de datos, garantizando que ésta se produzca correctamente.

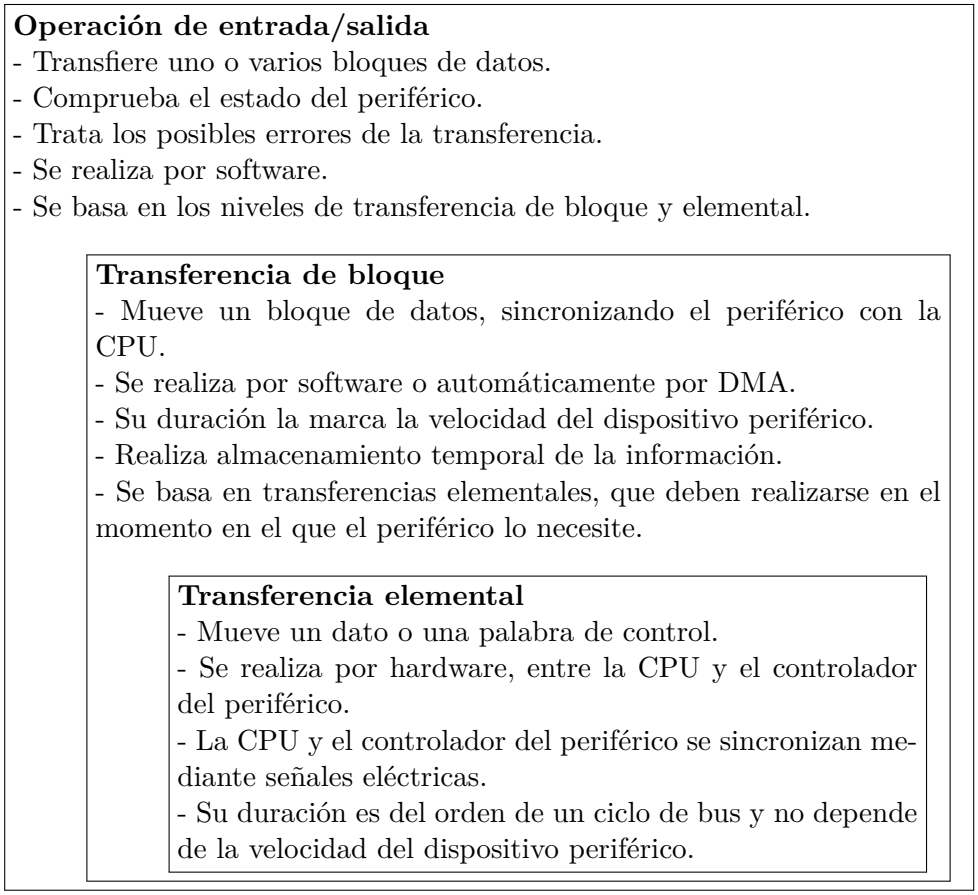


Figura 8.1: Estructuración en niveles de la Entrada/Salida

La Figura 8.1 representa las funciones más importantes de los niveles en los que se estructura la Entrada/Salida [Mig94]. Como complemento a esta funciones, se puede destacar lo siguiente:

- Transferencia elemental:
  - Comunicación física entre el periférico y la CPU para la transmisión de un bit, un octeto o una palabra. Hay que considerar los distintos mecanismos físicos de comunicación, así como las señales necesarias para realizarla.
  - Control de los periféricos. Este control incluye la interrogación y modificación de su estado (encendido/apagado, disponibilidad, etc.). Para ello, la CPU debe ser capaz de leer y gobernar una serie de señales de control del periférico.
- Transferencia de bloque:

  - Cuenta de los octetos o palabras transmitidas. Cuando se realizan transferencias múltiples es fundamental llevar esta cuenta para poder finalizarlas en el instante adecuado.
  - Sincronización de la CPU y el periférico. La CPU suele ser más rápida que el periférico, por lo que hay que establecer un mecanismo para que ésta sepa cuándo puede enviar o debe recibir un dato.
  - Detección de errores mediante códigos de paridad o polinomiales, y en caso necesario, repetición de la transferencia.

- Operación de entrada/salida:

  - Almacenamiento temporal de la información. Por razones de seguridad, la transferencia de los datos no suele hacerse sobre la zona de memoria del programa que la solicitó, sino sobre un *buffer* temporal.
  - Conversión de códigos. Suele ser necesaria ya que los datos están representados en el periférico de una forma distinta a como lo están en la CPU.
  - Conversión serie/paralelo. El distinto ancho de palabra de la CPU y el periférico hace necesario realizar operaciones de paralelización y serialización.

8.2. Comunicación física

Para analizar la comunicación entre la CPU y un periférico es conveniente modelizar este último como si estuviera formado por: un *dispositivo*, que comprende una parte mecánica y otra electrónica, y un *controlador*, que es la parte encargada de la comunicación con la CPU. A título de ejemplo, una unidad de disco está compuesta por los siguientes elementos:

- Dispositivo, formado por:

  - Motor que hace girar el disco a una velocidad constante.
  - Motor para posicionar el brazo.
  - Electrónica que genera las formas de corriente para grabar y que es capaz de interpretar las corrientes de lectura.
  - Electrónica de accionamiento de los motores.

- Controlador, que contiene, entre otras cosas:

  - Registro de datos.
  - Registro de control.
  - Registro de estado.

El registro de datos del controlador se emplea, a modo de buzón, para realizar las transferencias de datos con la CPU. El registro de control es el que contiene la información necesaria para realizar los accesos a la información contenida en el disco. El registro de estado, como su nombre indica, mantiene el estado actual del periférico.

La comunicación entre CPU y periférico no está gobernada por entero por la CPU puesto que algunas de las señales de control, necesarias para la transferencia, las ha de producir el controlador del periférico.

Existen dos formas de realizar una transferencia elemental: por ejecución de una instrucción (E/S programada) y por acceso directo a memoria. Éstas se tratan en las siguientes secciones.

Entrada/Salida programada

Cuando se utiliza E/S programada, la transferencia se realiza ejecutando una instrucción de E/S de la CPU. (La sincronización entre CPU y periférico se tratará más adelante.) Las operaciones de E/S son fundamentalmente operaciones de transferencia tipo `lw`, donde un operando es la dirección de memoria donde se encuentra el periférico y el otro es la posición de memoria donde se desea depositar la información. Se diferencia de la instrucción `lw` típica en que se debe contar con la siguiente información relativa al periférico:

Información de dirección. Esta información permite elegir el registro del periférico sobre el que se realiza la transferencia.

Tipo de operación. Especifica si la operación es de lectura o de escritura.

Temporización. Es necesario sincronizar la comunicación entre CPU y periférico.

La comunicación mediante este método se realiza utilizando las siguientes señales:

Señales de dirección. Describen la ubicación del puerto de E/S, dentro del mapa de memoria del computador, asociado con el periférico que se desea acceder. Es necesario, pues, que el controlador del periférico posea la capacidad de decodificar la dirección presente en el bus de direcciones (o el fragmento del mismo al que tenga acceso) y determinar si la dirección que hay en él, corresponde al dispositivo que este controlador tiene asociado. Existen dos tipos de mapas de memoria en cuanto a los periféricos se refiere: comunes y separados. En el caso de los separados, se usan señales específicas para indicar que se está accediendo a un periférico.

Señales de datos. Normalmente, las señales de datos se transmiten a través de un bus bidireccional, que es el propio bus de datos del computador. Es necesario que el controlador del periférico se conecte mediante puertas triestado a dicho bus.

Señales de control. Especifican el tipo de transferencia (si es de entrada o de salida) y establecen su temporización. La temporización puede ser síncrona o asíncrona. En la temporización síncrona, una señal de reloj marca el ritmo de transferencia, mientras que en la asíncrona, una señal adicional maneja al periférico e indica a la CPU cuándo puede leer o debe escribir un nuevo dato.

Acceso directo a memoria

Cuando la transferencia se realiza por acceso directo a memoria, es el controlador del periférico quien lleva todo el peso de la transferencia, comunicándose directamente con la memoria del computador sin necesidad de la intervención de la CPU. Para ello, la CPU debe proporcionar al principio de la transferencia la siguiente información al controlador del periférico:

- Dirección de memoria principal donde transferir la información.
- Dirección del periférico.
- Tipo de operación (lectura o escritura).
- Cantidad de información a transferir.

Existen dos formas de realizar esta operación:

- Mediante el uso de *memorias multipuerto*. El computador está dotado de un tipo especial de memoria que permite dos o más accesos simultáneos a la misma.
- Por *robo de ciclo*. El controlador del periférico aprovecha aquellos instantes en que la CPU no usa el bus para realizar la transferencia.

8.3. Métodos de sincronización

Se tratan aquí los mecanismos existentes para la sincronización entre CPU y periférico. Existen dos formas básicas de sincronizar el inicio de la transferencia de información entre la CPU y el periférico: *polling* o encuesta e interrupciones.

El método de *polling* o encuesta consiste en que la CPU consulta periódicamente todos los registros de estado de todos los periféricos que tiene conectados. Si alguno de ellos requiere su atención, procede a realizar la transferencia de información con el mismo. Este método es muy sencillo de implementar, ya que todo el peso recae sobre la CPU, pero resulta extremadamente lento cuando el número de periféricos es elevado y no es eficaz cuando las transferencias entre CPU y periféricos se realizan ocasionalmente.

El método de interrupciones consiste en la utilización de una serie de líneas de interrupción que debe poseer la CPU. Cuando se activan estas líneas, la CPU *aparca* la ejecución del programa en curso para atender a los periféricos que la soliciten. Es decir, al producirse una petición de interrupción, la CPU debe almacenar el estado actual y atender la interrupción solicitada, además, cuando ésta finalice, debe recuperar el estado almacenado para poder continuar con la ejecución del programa en el mismo punto en el que éste fue interrumpido. Es más, se ha de permitir incluso una interrupción cuando se está atendiendo otra, estableciéndose para ello unos niveles de prioridad. Estos niveles permiten que ciertas interrupciones tengan mayor prioridad que otras, es decir, cuando se atiende una interrupción, el nivel de la misma se almacena en un registro de estado de la CPU, para de esta forma, sólo atender a aquellas interrupciones con un nivel de prioridad mayor a la actual. La ventaja del método de interrupciones estriba en que la CPU atiende a los periféricos sólo cuando éstos lo requieren y, además, en el mismo instante en que éstos están preparados.

En resumen, el método de interrupciones es más complejo de implementar que el de *polling*, ya que requiere hardware específico, pero resulta mucho más eficiente que este último.

Bibliografía

[GGM92] García Sánchez, J. E., Gil Tomas, D. A. y Martínez Iniesta, M. (1992). *Circuitos y sistemas digitales.* Tebar Flores. ISBN 84-7360-125-4.

[Hyd96] Hyde, Randall (1996). *The art of assembly language programming.* University of California, Riverside. College of Engineering. Departament of Computer Science.
URL `http://webster.cs.ucr.edu/Page_asm/ArtOfAsm.html`

[Mig94] de Miguel Anasagasti, Pedro (1994). *Fundamentos de los computadores.* Paraninfo. ISBN 84-283-1790-9.

[PH94] Patterson, David A. y Hennessy, John L. (1994). *Organización y diseño de computadores. La interfaz hardware/software.* McGraw-Hill. ISBN 84-481-1829-4.

[Sta97] Stallings, Williams (1997). *Organización y arquitectura de computadores. Diseño para optimizar prestaciones.* Prentice Hall. ISBN 84-896-6024-7.

[Tan92] Tanenbaum, Andrew S. (1992). *Organización de computadoras. Un enfoque estructurado.* Prentice-Hall Hispanoamericana, S.A., segunda edición. ISBN 968-880-238-7.