Este libro fue amablemente cedido por el Ing. Fernando Szklanny para ser utilizado por los alumnos de la materia Arquitectura de Computadoras 1109 de la Universidad Nacional de La Matanza. Solo se permite el uso de esta versión a los alumnos de esta materia.

Agradecemos al Ing. Szklanny su gentileza al ceder esta versión digital muy útil durante el aislamiento por Covid-19.

INTRODUCCIÓN A LOS SISTEMAS DIGITALES

ING. FERNANDO IGNACIO SZKLANNY

PRÓLOGO

La segunda edición de esta Introducción a los Sistemas Digitales, surgida dos años después de la versión inicial, incluye no sólo la revisión y corrección total de la misma, sino también el agregado de algunos temas complementarios a los publicados anteriormente.

Es así que se ha ampliado el capítulo 4, referido a funciones lógicas, con los métodos utilizados para la simplificación de dichas funciones. Asimismo, se ha agregado el capítulo 5, con una base introductoria a los circuitos combinatorios.

Planteado así, este trabajo aparece como un texto introductorio, pensado para aquellos que inician su camino por los circuitos y sistemas digitales, sin pretender de ninguna manera reemplazar la lectura de las muchas y muy buenas obras que existen sobre el tema. Tal como lo planteé en el prólogo de la primera edición, el objetivo era el de ofrecer algo que sirviese como texto básico para una materia introductoria al hardware de computadoras en carreras como licenciatura de sistemas, ingeniería de sistemas, o afines, para alumnos sin conocimientos de hardware ni de los recursos básicos que sirven como base al hardware de computadoras (sistema de numeración binaria, álgebra de Boole, lógica electrónica, etc.).

Quedan pendientes para una próxima, y espero que definitiva, edición, los temas referidos a circuitos secuenciales, así como el aspecto tecnológico de los circuitos integrados y la lógica electrónica.

Debo a esta altura replantear los agradecimientos a todos aquellos que insistieron en que el camino iniciado era el correcto. Por consiguiente, nuevamente quiero agradecer a mi colega, el Ing. Daniel Giullianelli, cuya insistencia me obligó a decidir de una vez la publicación de aquellos capítulos que hace ya un tiempo yacían encerrados en una computadora, y que, al utilizar este texto en su cátedra de Introducción a la Informática, me puso en la obligación de continuar con el compromiso planteado en la primera edición de esta modesta obra.

Debo agradecer también al Lic. Fernando Orthusteguy, por su completa revisión de la primera edición y sus comentarios y sugerencias para mejorar algunos de los temas desarrollados.

También debo reiterar mi agradecimiento a mis colaboradores docentes en las cátedras de Sistemas de Computación I en la Universidad Nacional de La Matanza, de Sistemas de Computación de la Universidad de Morón y de Técnicas Digitales de la Facultad de Ingeniería de la Universidad de Buenos Aires, por cuanto, además de sus continuos aportes a la metodología de la enseñanza de estos contenidos, fueron lo suficientemente críticos como para hacerme notar cuantas cosas podrían haber hecho mejor. Hago mención aquí, reparando la injusticia cometida en la primera edición, a mis colegas docentes Elio De María, Miriam Perez Berro, Silvia Perrone, Carlos Maidana, Hugo Tantignone, Juan Carlos Cervellera, Roberto Di Lorenzo, Roberto Landaburu, Francisco Vanni, Guillermo Beneitez, Carlos Rodríguez, Hugo Pagola, Miguel Martinez, Pablo Lena, Eduardo Iacobacci, Marcelo Volpi, Hugo Pagola, Facundo Caram, Veronica Tramanelli, Jorge Fiter, Pablo Pandolfo. Cada uno de ellos, de alguna manera o de otra, ha hecho algo para que esta segunda edición esté en camino.

Nuevamente corresponde agradecer a mi familia, por su apoyo y su paciencia, y en especial a mi hijo Ariel, que ahora, con catorce años, siguió trabajando en la ilustración de los capítulos que forman esta publicación con el mismo entusiasmo con el que lo hizo anteriormente.

Por último, debo reiterar el compromiso, ante quienes han intentado convencerme de la importancia de este trabajo, de intentar llegar a completar, en el plazo más corto posible, el objetivo inicialmente planteado

Fernando I. Szklanny
Marzo 2002

INTRODUCCIÓN A LOS SISTEMAS DIGITALES

ING. FERNANDO IGNACIO SZKLANNY

CAPÍTULO I:

CONCEPTOS FUNDAMENTALES I

CAPÍTULO 1: CONCEPTOS FUNDAMENTALES I

1.1.- Introducción

El incesante progreso de los sistemas de computación está asociado con la permanente necesidad de administrar información en forma cada vez más intensiva. Esta exigencia hace que aquellas actividades relacionadas con el manejo de la información requieran herramientas y medios que permitan simplificar y acelerar ese manejo. Operaciones y actividades que no hace mucho tiempo atrás se resolvían mediante elementos que hoy parecen primitivos (papel y lápiz, regla de cálculo, ábacos, calculadoras mecánicas, etc.) requieren de sistemas de computación como algo casi imprescindible.

No cabe duda de que los progresos tecnológicos, especialmente en el campo de la electrónica y la microelectrónica, así como en el de las técnicas de programación, han permitido la valorización de la computadora como herramienta auxiliar para muchas aplicaciones de la vida diaria.

Y tampoco cabe duda de que otro progreso tecnológico de estos últimos años, producido en el campo de las comunicaciones, ha vuelto a revolucionar la tecnología, al permitir la intercomunicación de los sistemas de transmisión de voz y de datos a nivel mundial, con una sencillez, para el usuario, casi imposible de imaginar hace no muchos años atrás.

1.2.- Magnitudes físicas

Todos estos progresos tienen que ver fundamentalmente con administración de información. Dicha información se obtiene, en última instancia, a partir de valores o datos que pueden obtenerse en función de diferentes procesos o procedimientos físicos. Al hablar de procedimientos físicos, se piensa habitualmente en lo que se conoce como medición. Desde el punto de vista físico, medir significa comparar, y de acuerdo con el tipo de datos a medir, esa comparación puede llevarse a cabo de distintas maneras. El concepto de medición se relaciona con el concepto físico de magnitud, y la comparación vinculada con esa medición, a su vez, se relaciona con la unidad correspondiente a esa magnitud.

Para intentar poner claridad sobre este planteo, se deberán establecer algunas definiciones básicas que permitan vincular estas consideraciones físicas con el objetivo fundamental de esta obra: la utilización de los sistemas de computación y su vinculación con el procesamiento electrónico de información. De acuerdo con la definición tradicional, se llama magnitud a todo aquello que es capaz de ser medido. Una magnitud física, por consiguiente, implica la posibilidad de determinar un valor, el que deberá determinarse a través de su comparación con la unidad correspondiente a dicha magnitud. Dicha unidad, en general, se encuentra definida en la forma de un patrón de medición, establecido en forma precisa y de carácter universal.

1.3.- Magnitudes continuas y discretas

Dentro del ámbito de la física, la inmensa mayoría de las magnitudes son de tipo **continuo**. Este concepto implica que, dentro de un rango de validez aceptado para dicha magnitud, la misma puede adoptar todos y cada uno de los valores existentes. Dicho de otra manera, entre dos valores cualesquiera determinados para esa magnitud, siempre podrá encontrarse algún otro valor intermedio. Dado que esta última definición es iterativa, se puede pensar que una magnitud es continua cuando, dentro de su rango de validez, es capaz de adoptar infinitos valores distintos. Si se representa una magnitud de estas características en un diagrama de tiempos, lo que se obtendrá,

desde el punto de vista matemático es una función continua F = f(t) tal como se observa en la figura 1.1.

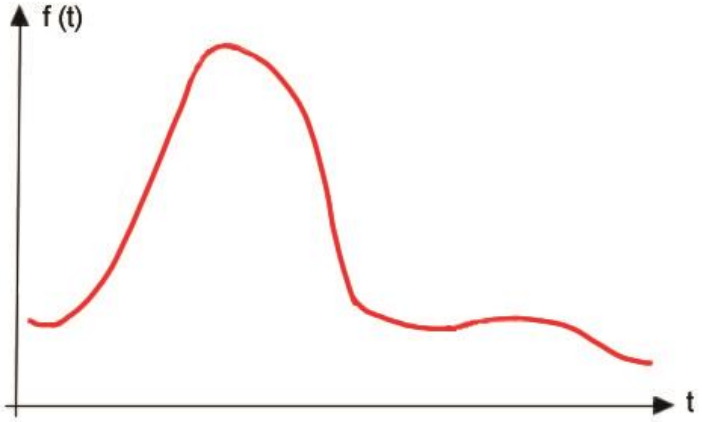


figura 1.1. – Representación de una función continua

No obstante, existen en el universo magnitudes cuya valoración no es continua. Por el contrario, en estos casos, los distintos valores que las componen están separados entre sí, por lo que no cabe la posibilidad de encontrar siempre un valor intermedio entre otros dos valores de la magnitud en cuestión. En estos casos se entenderá que la magnitud es **discreta** o **cuantificada**, y su representación temporal mostrará una función no continua, en la que solamente se podrán tener, sobre el eje de ordenadas, aquellos valores de la función que correspondan a su cuantificación. La figura 1.2 muestra un ejemplo de magnitud discreta. Surge de esta definición que el proceso de medida de una magnitud discreta es a través del conteo de sus valores permitidos. Una magnitud discreta se mide contando.

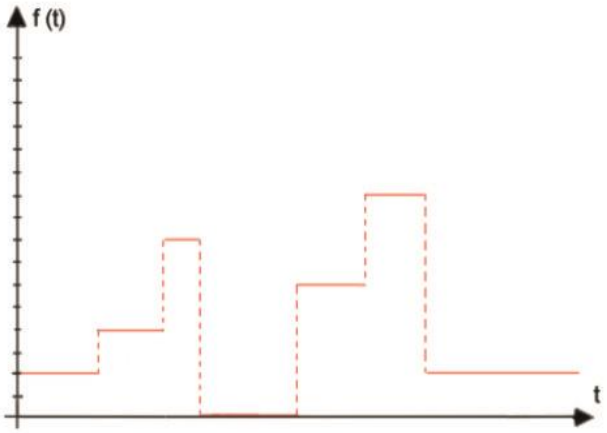


Figura 1.2. Una función discreta

La necesidad de medición de las diferentes magnitudes determina la existencia de métodos e instrumentos adecuados para cada una de ellas. Así, se utilizará la balanza para medir pesos, o se utilizará un recipiente graduado para medir volúmenes o capacidades. De la misma manera, se pueden establecer dispositivos que permitan medir velocidades, presiones, niveles sonoros, distancias, temperaturas, etc., por señalar solamente algunas de las magnitudes habitualmente medidas en la actividad diaria.

En muchos casos, la medición de determinadas magnitudes se realiza en forma indirecta, lo que significa la medición de una magnitud determinada a través de otra. Este concepto implica, por lo

general, la necesidad de una conversión entre la magnitud a medir y la que se utiliza como intermediaria, o una relación directa entre ambas, que permita obtener los resultados requeridos.

Se define como ***sensor*** a un elemento que permite reconocer una magnitud determinada, respondiendo a la misma en una forma conocida, generalmente una magnitud eléctrica proporcional a la magnitud física que se está pretendiendo medir. Esto significa una relación (que no necesariamente debe ser lineal) entre la magnitud medida por el sensor y la magnitud eléctrica entregada por el mismo.

La utilización de sensores para la medición de temperaturas (termocuplas, termoresistencias, etc.), presiones o fuerzas (celdas de carga), etc., permite obtener una forma sencilla de medida, adecuada a distintos tipos de instrumentos que completarán la medición en forma eléctrica o electrónica, permitiendo así la obtención de una información compatible con los modernos sistemas de procesamiento de datos.

1.4.- El concepto de lo analógico y lo digital. Conversión analógico-digital.

La utilización de sensores para medir magnitudes físicas es hay en día una práctica corriente, ya que la diversidad de sensores existentes, para casi todo tipo de magnitud, su precisión y, en muchos casos, su bajo precio y su compatibilidad con los sistemas de procesamiento de datos facilita en forma tal la medición que no se justifica la utilización de métodos más complejos aún cuando sean más precisos o exactos. La relación establecida entre la magnitud que se mide y la magnitud que entrega el sensor en su salida, al ser conocida, permite determinar perfectamente el valor que se está midiendo. Las magnitudes a medir son, en su inmensa mayoría, magnitudes continuas o ***analógicas,*** lo que implica, según lo ya expresado, magnitudes que se representan a través de una función continua. Las salidas de los sensores también son magnitudes continuas. En la generalidad de los casos, no obstante, resulta conveniente que la magnitud medida se represente como una función discreta o ***digital,*** dado que, según se verá oportunamente, los sistemas de procesamiento de datos operan con magnitudes discretas. Representar una magnitud continua (analógica) a través de una magnitud discreta (digital), significa la **cuantificación** de esa magnitud continua, convirtiéndola en una sucesión de pasos discretos. Esta cuantificación, obviamente, y según se puede ver en la figura 1.4, convierte a la magnitud continua en una serie de escalones que la representan. El **error de cuantificación** representa la diferencia entre el valor real de la magnitud y el valor medido en un momento dado. Este error será menor en la medida que la digitalización se realice con un mayor número de escalones o pasos, lo que significará pasos más chicos y, por consiguiente, una mayor aproximación al valor real de la magnitud.

La figura 1.3 permite ver la cuantificación de una magnitud arbitrariamente elegida, continua, utilizando solamente cuatro pasos de cuantificación. En la figura 1.4 se muestra la misma magnitud digitalizada a través de 16 pasos. Se puede intuir que, cuanto mayor el número de pasos dentro del rango de la magnitud digitalizada, más precisa será la conversión realizada.

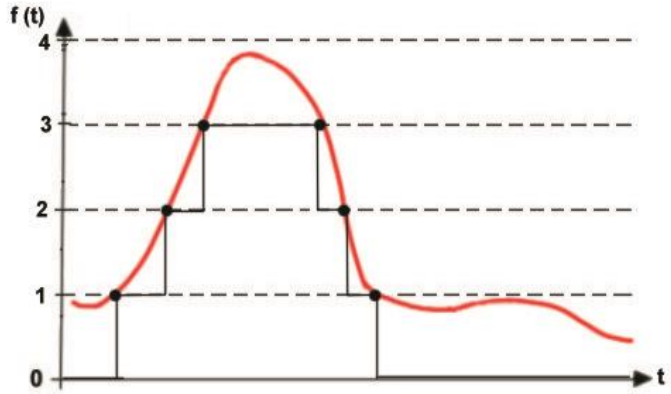


Figura 1.3: Cuatro pasos de digitalización

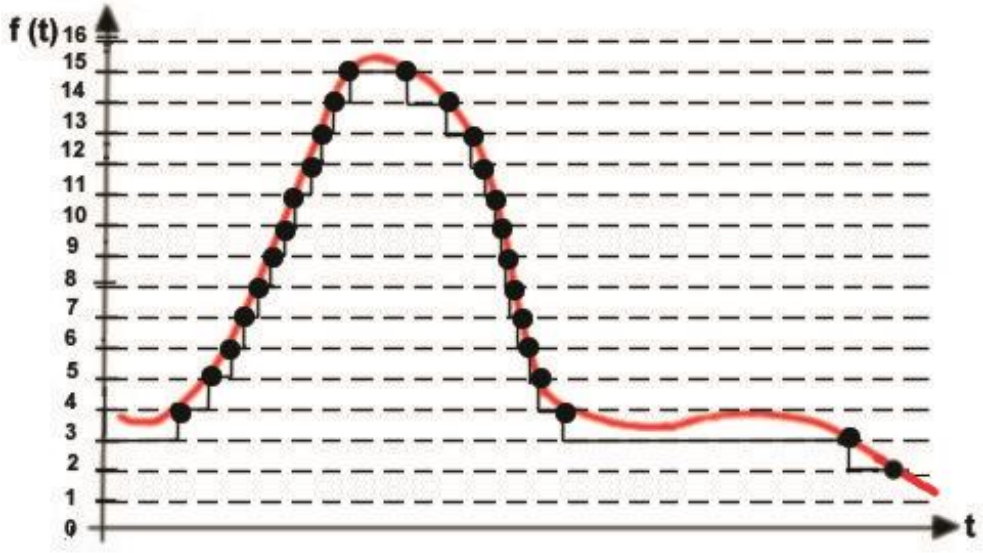


Figura 1.4: Cuatro veces más pasos de digitalización

El concepto anterior se conoce como ***conversión analógico-digital***. Se utiliza para representar mediciones de magnitudes continuas a través de valores numéricos, ya que, debe recordarse, se ha expresado previamente que las magnitudes discretas se miden contando. El procedimiento inverso, de conversión digital-analógica, permite reconstruir, con la aproximación requerida, una magnitud de tipo analógico a partir de una serie de valores discretos. En capítulos posteriores se verán las aplicaciones prácticas de estos dos procedimientos.

1.5.- Conceptos básicos de electricidad. Tensión, corriente, resistencia. Unidades. Circuitos eléctricos. Configuración serie y paralelo.

La utilización de sensores para la medición de diferentes magnitudes permite la representación de esas magnitudes a medir por otras, proporcionales (o no) a las magnitudes originales, y de mayor sencillez en su manejo. La representación entregada por un sensor es una magnitud eléctrica, lo que implica la conversión del valor adquirido por una **tensión** o una **corriente**. Como simple introducción a los circuitos eléctricos, se recordarán en el presente apartado algunos conceptos básicos. La profundización de los mismos puede obtenerse en las referencias bibliográficas correspondientes.

Un circuito eléctrico elemental está compuesto por un generador eléctrico y una carga. El generador eléctrico es un elemento, que puede tener distintas formas y tamaños, que entrega, entre sus extremos, una determinada **tensión** o **potencial eléctrico.** Este generador puede adoptar la forma de una batería eléctrica, del tipo de las que se utilizan para alimentar calculadoras, receptores de radio, teléfonos celulares, etc., elementos generadores de ***corriente continua***. Puede también adoptar otras formas, invisibles probablemente para la mayoría de los usuarios, quienes no ven el generador sino sus efectos, representados por los diversos tomacorrientes en los que se conectan lámparas, motores, artefactos eléctricos y electrónicos de uso diario y permanente. Estos generadores, que pueden ser de distinta naturaleza (térmicos, hidroeléctricos, nucleares, etc.), proveen la ***corriente alterna*** que se utiliza en aquellos casos en que se requiere una mayor provisión de **energía eléctrica**. El nombre de corriente alterna proviene de que la forma de onda de las magnitudes eléctricas existentes en el circuito adopta una variación, en el tiempo, que la hace cambiar de signo con respecto del nivel tomado como referencia. En la generalidad de los casos, dicha forma de onda es senoidal. (figura 1.5)

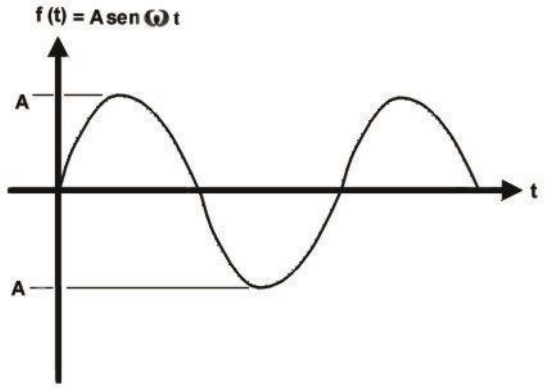


figura 1.5: Función senoidal

Cuando se conecta una **carga eléctrica** sobre un generador, el conjunto genera lo que se conoce como **corriente eléctrica**. Esta corriente eléctrica consiste en el movimiento de electrones a través del circuito. La relación entre la corriente eléctrica que circula por un circuito y la tensión generada por el generador está determinada por la ley de Ohm, la que define una relación directa entre las dos magnitudes:

$$V = I \cdot R$$

expresión en la que **V** representa la tensión eléctrica producida por el generador, **I** representa la corriente eléctrica en el circuito, y **R** representa la **resistencia eléctrica** de la carga conectada sobre el circuito. Se entiende por resistencia eléctrica a la capacidad de la carga de oponerse al paso de la corriente. En base al valor de resistencia que adoptan los distintos elementos cuando se hace circular corriente eléctrica a su través, los mismos se pueden clasificar en materiales conductores y aisladores. Un material es **conductor** cuando no se opone al paso de la corriente eléctrica, lo que, desde el punto de vista de la definición anterior, implica un elemento de baja resistencia. En cambio, se define como **aislante** o **dieléctrico** a un elemento cuya oposición al paso de la corriente eléctrica es importante, lo que significa un elevado valor de resistencia eléctrica. Más adelante, se analizará el concepto nuevamente sobre la base de la definición de la **energía eléctrica**. La figura 1.6 ilustra en forma elemental un circuito eléctrico de corriente continua, en el que se muestran los elementos integrantes del mismo.

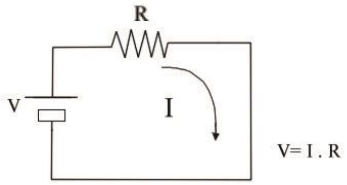


figura 1.6: Circuito eléctrico básico – Ley de Ohm

Cuando el circuito está alimentado por un generador de corriente alterna, la definición de la ley de Ohm se amplía, reemplazando el concepto de resistencia eléctrica por el de **impedancia**:

$$V = I. Z$$

siendo ahora Z la impedancia eléctrica de la carga conectada sobre el generador. Al igual que el concepto anterior, el de impedancia refleja la capacidad de la carga eléctrica de un circuito a oponerse al paso de la corriente. Es una magnitud que solamente tiene validez en circuitos de corriente alterna, en los cuales la relación entre la tensión y la corriente no solamente es una constante numérica debido a que el tipo de carga puede hacer que se produzca un defasaje temporal entre tensión y corriente.

Lo interesante es que los efectos que se obtienen de un circuito eléctrico no se determinan ni por la corriente ni por la tensión del circuito sino por un concepto derivado, como es el de **potencia eléctrica**. La potencia de un circuito eléctrico está relacionada con un concepto aún más importante: el de **energía eléctrica.** La energía eléctrica se determina por la capacidad que posee el circuito eléctrico de producir trabajo, por lo que su definición está directamente asociada con los conceptos de trabajo conocidos desde la mecánica y la física convencionales. Desde el punto de vista del cálculo, la potencia de un circuito eléctrico se determina fácilmente como el producto de la tensión aplicada por la corriente circulante, y por consiguiente, utilizando la ley de Ohm, se obtienen distintas expresiones que permiten calcular la potencia desarrollada en un circuito eléctrico:

$$W = V. I = I^{2}.R = V^{2}/R$$

Por otra parte, y dado que la potencia de un circuito se establece como la energía generada en la unidad de tiempo, es inmediato determinar que la energía eléctrica, en función de todo lo dicho, puede definirse como

$$E = W. t$$

siendo E la energía eléctrica, W la potencia determinada y t el tiempo durante el cual se mantiene esa potencia.

1.5.1.- Unidades.

Las distintas magnitudes que forman parte de un circuito eléctrico se expresan en base a unidades convencionales, definidas en los distintos sistemas normalizados de medidas y unidades. La tensión eléctrica se mide habitualmente en **Volts (V),** unidad conocida dado que la mayoría de los

generadores expresan su valor en dicha unidad (Una pila entrega una tensión de 1,5 V, una batería de automóvil alimenta al mismo con 12 V, la alimentación domiciliaria se realiza a un valor de 220 V, etc.). La corriente circulante en ese circuito se mide en **Amperes (A)**, y en ese caso, la resistencia o la impedancia eléctrica se miden en **Ohm (Ω).** Esto significa que si en el circuito de la figura 1.6 se aplica una tensión de 1 Volt sobre una carga de 1 Ohm se obtendrá una corriente de 1 Ampere. Por otra parte, cuando la tensión del circuito se mide en Volts y la corriente se mide en Amperes, la potencia del circuito se mide en **Watt (W).** En este caso, es habitual definir la energía eléctrica en términos de **Watt-hora**, unidad definida directamente a partir de las expresiones del apartado anterior. En la mayoría de los casos se utiliza un múltiplo de esa unidad, el **Kwh (Kilowatt-hora)** para definir la energía producida por un motor, un generador o una subestación eléctrica.

1.5.2.- Circuitos eléctricos elementales. Configuración serie y paralelo.

El esquema circuital de la figura 1.6 es un ejemplo de un circuito formado por sus tres elementos básicos: un generador, una carga eléctrica, y los elementos de conexión entre ambos. No obstante, es por demás frecuente que un mismo generador alimente de energía eléctrica a más de una carga en forma simultánea, caso en el cual la configuración del circuito se hace más compleja. En este caso, la energía eléctrica generada por el generador se distribuye entre todos los elementos pasivos que componen el circuito. Es el caso típico de una instalación domiciliaria, en la que la cual la empresa de energía alimenta en forma simultánea a todos los electrodomésticos que se encuentren conectados sobre la línea de tensión.

El caso ejemplificado involucra un circuito eléctrico configurado en **paralelo**, tal como se muestra en la figura 1.7. En la misma se observa que a todos los elementos conectados se les provee la misma tensión, por lo que, intuitivamente, puede llegar a determinarse que la corriente eléctrica provista por el generador se distribuye por las distintas ramas del circuito en forma inversamente proporcional a la resistencia (impedancia) de los elementos que lo forman.

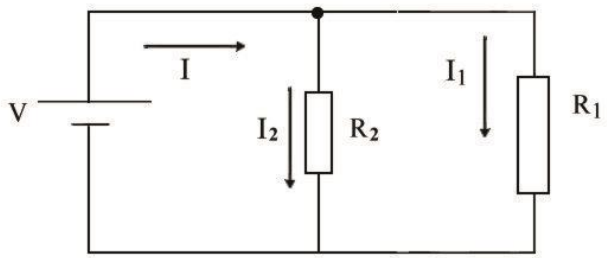


figura 1.7 : Un circuito en Paralelo

En otros casos, los elementos alimentados por la fuente de energía pueden estar conectados sobre la misma en otra forma, descripta en la figura 1.8. En este caso, todos los elementos pasivos comparten la línea de alimentación, por lo que la corriente eléctrica en todos los elementos del circuito es la misma. Este tipo de configuración se conoce como circuito **serie.**

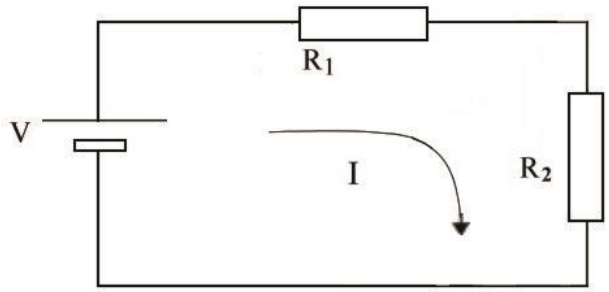


figura 1.8: Un circuito en serie

En otros casos, se puede lograr una combinación de ambos esquemas de conexión, obteniéndose en este caso circuitos de configuración serie-paralelo, como el que se representa en la figura 1.9.

Cualquier circuito eléctrico puede reducirse a una de estas formas básicas, en las cuales se satisfacen las reglas (tales como las leyes de Ohm, Kirchhoff, etc.), que determinan los balances de tensiones y corrientes que deben cumplir tanto las tensiones aplicadas como las corrientes circulantes por las distintas mallas.

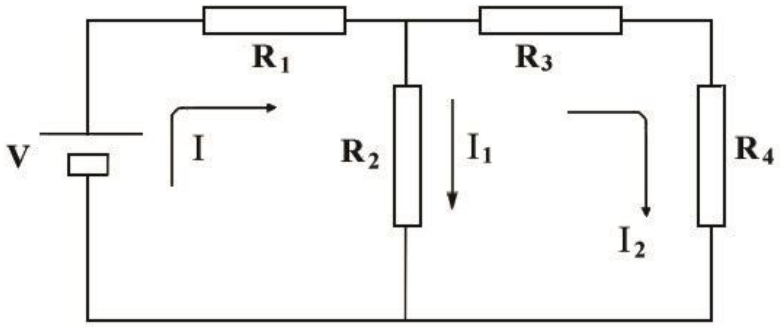


figura 1.9: Circuito serie - paralelo

1.6.- Magnitudes repetitivas. El concepto de período. El concepto de frecuencia. Unidades utilizadas para medir frecuencia y período.

Se ha definido con anterioridad el concepto de corriente alterna como el de una magnitud cuyo valor instantáneo varía, con respecto de un valor tomado como referencia, modificando no solamente su valor sino su signo a lo largo del tiempo. En el caso particular de la corriente alterna, se ha dicho e ilustrado que la variación mencionada se realiza en forma senoidal. Este es un ejemplo de una magnitud repetitiva o **periódica**, definida así debido a que su conducta en el tiempo se repite a intervalos regulares de tiempo. Se define como ***período*** de una función repetitiva al tiempo transcurrido entre dos pasos consecutivos de la magnitud por el mismo valor en el mismo sentido (figura 1.5). El período de una función repetitiva se mide en unidades de tiempo. Cada una de las repeticiones de la magnitud (o **señal**) periódica se conoce como **ciclo** de la magnitud. Queda claro que la cantidad de ciclos que se producen en un tiempo determinado depende directamente del período de la señal periódica. Se define como **frecuencia** de la magnitud

periódica a la cantidad de ciclos de la misma que se producen en la unidad de tiempo. Por consiguiente, la relación entre período y frecuencia es:

$$f = 1/T$$

expresión en la que **T** es el período de la señal y **f** es su frecuencia. Al ser la cantidad de ciclos una magnitud adimensional, la frecuencia se mide en inversa de la unidad de tiempo. Cuando el período de la magnitud se mide en segundos, la inversa del segundo (1/seg) se denomina **Hertz (Hz),** unidad habitualmente utilizada, como sus múltiplos, para la medición de frecuencias de magnitudes periódicas.

1.7.- Introducción a las magnitudes binarias.

Las magnitudes periódicas no siempre tienen comportamiento sinusoidal. Más aún, en el caso de magnitudes discretas, la cuantificación hace que la variación, aún repetitiva, sea de tipo escalonado, como se observa en el ejemplo de la figura 1.10.

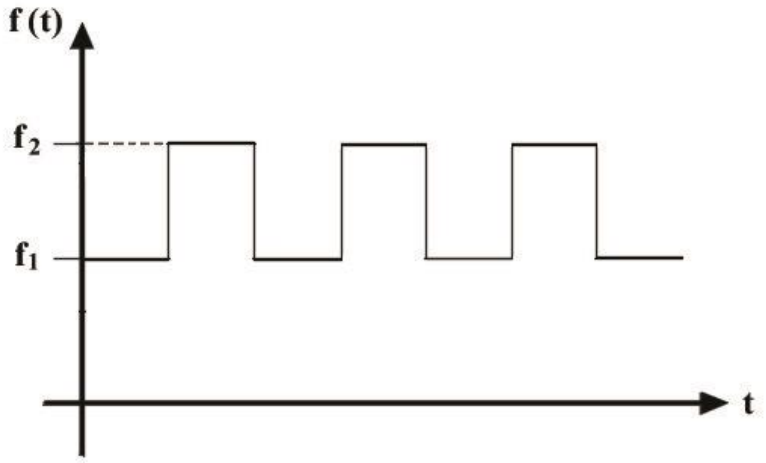


figura 1.10.: Función periódica discreta

En el caso más simple, puede pensarse en una magnitud que adopte solamente dos valores, que pueden ser uno positivo y otro negativo con respecto de la referencia adoptada (figura 1.11a) o bien, uno de ellos coincidente con la referencia siendo el otro de valor distinto a la misma - positivo o negativo (figura 1.11b).

En cualquiera de los casos, la magnitud representada adopta solamente dos valores diferentes, por lo que se la denomina **magnitud binaria.**

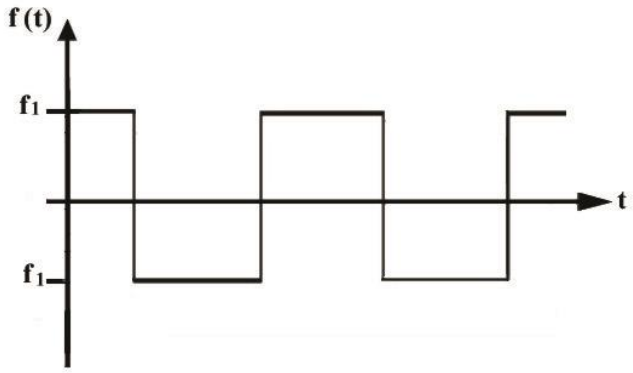


figura 1.11.a: Función binaria bipolar

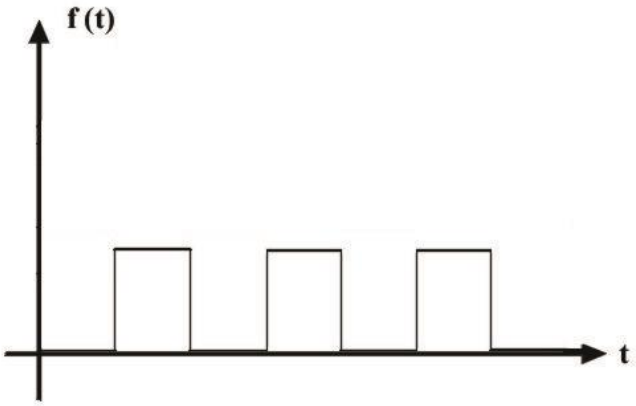


figura 1.11.b: Función binaria unipolar

En una magnitud binaria, como las representadas, se siguen definiendo los conceptos de frecuencia (cantidad de ciclos en la unidad de tiempo) y período (duración de cada uno de los ciclos de la magnitud). Pero surgen además algunos otros conceptos, que dependen de la relación entre los tiempos en que la magnitud se mantiene en cada uno de sus dos valores. Si se considera como **semiciclo positivo** a aquel tiempo en que la magnitud toma el valor mayor y como **semiciclo negativo** al tiempo en que se mantiene en el menor de sus valores, se suele decir que la señal es **cuadrada** si ambos semiciclos son de duración similar. La figura anterior ilustra un modelo de señal cuadrada.

En cambio, cuando la magnitud permanece en uno de sus dos valores durante un tiempo substancialmente mayor que el tiempo en que permanece en el otro valor se dice que la magnitud es **pulsante**. En este caso, generalmente uno de los dos valores se considera como **valor activo** de la magnitud, entendiéndose como tal al tiempo durante el cual la magnitud produce efectos sobre los elementos sobre los que actúa, tiempo que se denominará **pulso**. El parámetro t_p define el **ancho** del pulso, correspondiente al tiempo en el que el mismo se mantiene activo.

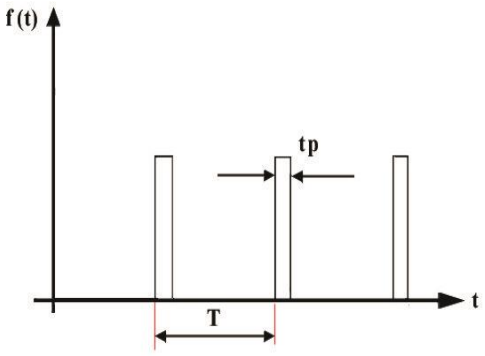


figura 1.12: Señal o magnitud pulsante

Se define como **ciclo de trabajo (duty cycle)** de una señal periódica a la relación entre tiempo en que la magnitud se mantiene en su estado activo y la duración total del ciclo, por lo que, de acuerdo con este concepto, una magnitud periódica binaria puede considerarse cuadrada si su ciclo de trabajo es de alrededor de 0,5 (o 50% si se lo mide en valores porcentuales), y pulsante si su ciclo de trabajo es cercano a 0 o a 1 (0% o 100%, respectivamente). La expresión que define el ciclo de trabajo es, de acuerdo con esta definición:

$$\tau = t_p/T$$

Las magnitudes pulsantes no siempre toman la forma de la figura 1.12. En efecto, en muchos casos, la misma activa a los elementos sobre los que actúa durante el momento en que su nivel es el más bajo de los dos. En un pulso negativo, el nivel activo es el menor de sus dos niveles, en tanto que el nivel inactivo es el nivel alto. El ciclo de trabajo sigue midiéndose como la relación entre el tiempo en que la magnitud está en su estado activo (ahora el nivel bajo) y el ancho total del pulso (o el período de la señal si la misma es periódica).

1.8.- Digitalización de magnitudes. El concepto de procesamiento digital.

Digitalizar una magnitud analógica significa convertirla en una magnitud digital. En realidad, y para expresarlo con mayor corrección, significa representar una magnitud analógica a través de otra magnitud, discreta, la que, por consiguiente, deberá adoptar su forma escalonada, en la mejor forma posible, a la forma de onda de la magnitud analógica original. En la práctica, digitalizar una magnitud analógica significa obtener una representación binaria de la misma, a través de dos pasos consecutivos: uno, la conversión de la magnitud en una magnitud discreta; el segundo, la representación de cada uno de los valores que adopta la magnitud discreta a través de un número binario.

Como se verá en el capítulo siguiente, la utilización de un sistema de representación binario, formado por solamente dos elementos diferentes, es en este caso una herramienta fundamental. Estos valores, a los que llamaremos ***bits***, corresponden uno a cada uno de los dos dígitos que constituyen el sistema binario de numeración. Los valores binarios así obtenidos servirán para su procesamiento a través de sistemas de computación, que utilizan esos valores para realizar con ellos todo tipo de operaciones, aritméticas, algebraicas, etc.

La figura 1.13 repite el ejemplo de la figura 1.4, en la que se muestra una magnitud analógica arbitraria.. La misma, de acuerdo con lo dicho en el correspondiente apartado, ha sido digitalizada

en cuatro y dieciséis niveles, lo que respectivamente requiere la utilización de dos y cuatro bits. Se indican asimismo los valores asignados a cada uno de los pasos de la función, y su representación binaria, la que quedará claramente justificada para el lector una vez que haya sido analizado el concepto de sistema posicional descripto en el capítulo 2.

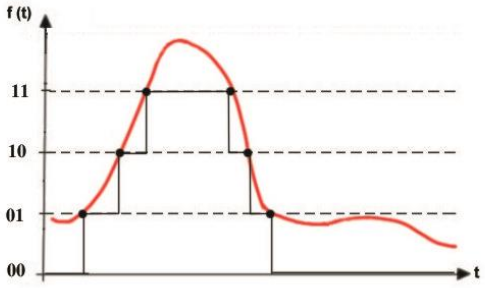


Figura 1.13.a: Nuevamente un ejemplo de digitalización en cuatro pasos (dos bits)

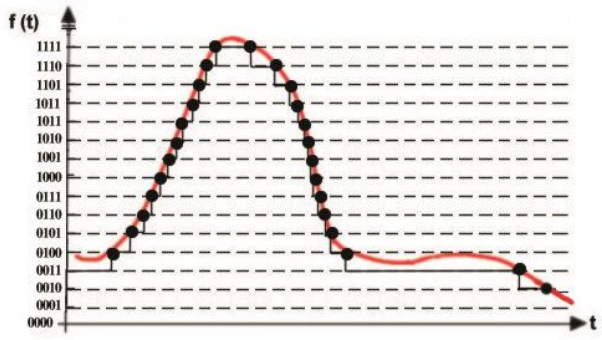


Figura 1,13.b: El mismo ejemplo en diez y seis pasos (cuatro bits)

INTRODUCCIÓN A LOS SISTEMAS DIGITALES

ING. FERNANDO IGNACIO SZKLANNY

CAPÍTULO II:

CONCEPTOS FUNDAMENTALES II:

SISTEMAS DE NÚMERACIÓN

CAPÍTULO 2 : SISTEMAS DE NUMERACIÓN

2.1.- Clasificación elemental: Sistemas posicionales y no posicionales. Definición de sistema posicional. Su expresión general. Concepto de base. Una antigua costumbre: el sistema de numeración decimal.

Un sistema de numeración es la herramienta requerida para la representación de cantidades numéricas. La costumbre, adquirida desde la infancia, de utilizar un único sistema de numeración, suele hacer que quien lo utiliza olvide que los conceptos que se utilizan a diario podrían adecuarse con facilidad a otros sistemas.

El mencionado sistema de numeración es el decimal, formado, como su nombre lo indica, por diez símbolos: los números del 0 al 9, conocidos como ***dígitos*.** Se define como dígito a cada uno de los símbolos diferentes que constituyen al sistema de numeración, y se define asimismo como ***base*** del sistema de numeración a la cantidad de dígitos que lo forman. Una vez agotada la cantidad de dígitos que forman al sistema de numeración, las cantidades mayores que la base se obtienen combinando en forma adecuada los diferentes dígitos del sistema. Esto hace que cada uno de esos dígitos adopte distintos valores según la posición que ocupe dentro del número representado. La existencia de este ***valor relativo*** o ***peso***, que define al dígito según su posición más allá de su valor absoluto, es la característica que permite definir al sistema decimal como un sistema numérico **posicional**.

En un sistema posicional, cada dígito adopta un valor relativo que depende de su posición dentro del número representado. En el número decimal 3434, por ejemplo, será distinto entre sí el valor de los dos dígitos 3, y lo mismo se podrá decir de los dos cuatros.

$$3434/_{10} = 3000 + 400 + 30 + 4$$

Para una representación más clara de este valor, el mismo podrá ser expresado como suma de potencias de la base 10:

$$3434/_{10} = 3.10^3 + 4.10^2 + 3.10^1 + 4.10^0$$

Este formato, que define el carácter posicional del sistema de numeración decimal, así como los diferentes valores que adoptan los dígitos a causa de su posición dentro del número o cantidad representada, se puede generalizar para cualquier sistema de numeración posicional.

En un sistema de numeración de base B, donde los dígitos del mismo se representan como a_i permite definir un número entero cualquiera N mediante la expresión

$$N_B = a_0.B^0 + a_1.B^1 + a_2.B^2 + ... + a_n.B^n ,$$

lo que, en forma abreviada, puede representarse mediante el clásico símbolo de sumatoria:

$$N_B = \sum_{i=0}^{n} a_i \cdot B^i$$

Esta expresión no establece límites o restricciones con respecto a la base del sistema de numeración a definir. Esto significa que si, por alguna razón especial, quisiéramos operar en un sistema de numeración de base 5, nos veríamos restringidos a utilizar únicamente los dígitos 0, 1, 2, 3 y 4, lo cual, de todos modos, nos permitiría expresar cantidades mediante la utilización exclusiva de esos dígitos.

Así, el mismo número anterior, 3434, pero ahora interpretado en este nuevo sistema de numeración, tendría el siguiente significado:

$$3434\ /_5 = 3.5^3 + 4.5^2 + 3.5^1 + 4.5^0$$

Debe notarse aquí que en la expresión anterior se ha utilizado un símbolo (el 5) que no forma parte del sistema de numeración original. El mismo sí forma parte del sistema de numeración decimal de uso diario. Por lo tanto, la expresión anterior ofrece la posibilidad inmediata de "leer" el número representado mediante la determinación de su valor decimal. Será simplemente necesario resolver la suma de potencias expresada, para averiguar el valor "real" del número así representado.

$$3434\ /_5 = 3.5^3 + 4.5^2 + 3.5^1 + 4.5^0 = 3.125 + 4.25 + 3.5 + 4 = 375 + 100 + 15 + 4 = 494\ /_{10}$$

Un sistema posicional, cualquiera sea la base que se utilice para el mismo, cumplirá con una serie de características o propiedades ya conocidas en el sistema decimal (debe recordarse que el sistema decimal es solamente un caso particular de sistema posicional). Así, los múltiplos de la base B estarán siempre terminados en 0, y las potencias de la base B se representarán con la unidad seguida de tantos ceros como sea el exponente al que estamos elevando dicha base.

La representación antedicha se puede extender también a los números fraccionarios, los que pueden expresarse como una suma de potencias de la base, pero ahora de exponente negativo. En efecto, el número 0,4573 no significa otra cosa que:

$$0,4573_{10} = \frac{4}{10} + \frac{5}{100} + \frac{7}{1000} + \frac{3}{10000} = 4.10^{-1} + 5.10^{-2} + 7.10^{-3} + 3.10^{-4}$$

Nuevamente puede extenderse la definición de un número menor que la unidad, expresado en una base cualquiera B, mediante el uso de la expresión

$$N_B = a_{-1}.B^{-1} + a_{-2}.B^{-2} + ... + a_{-m}.B^{-m}$$

o, en su forma reducida, mediante el uso del símbolo de sumatoria, expresándolo como

$$N_B = \sum_{i=-1}^{-m} a_i \cdot B^i$$

expresión en la que queda claramente establecido el carácter fraccionario del número N a través de los valores negativos por las que va transcurriendo la variable ***i***.

La representación genérica de un número N, formado por una parte entera y una fraccionaria, será la conjunción de ambas sumatorias, lo que nos permitirá completar la definición de un sistema posicional a través de la expresión matemática siguiente, que resume lo expresado anteriormente:

$$N_B = \sum_{i=-m}^{n} a_i \cdot B^i$$

No todos los sistemas de numeración utilizados son posicionales. De hecho, es habitual el uso de un sistema de numeración *no posicional*, como lo es el sistema romano. En este tipo de sistemas, las reglas de formación de las representaciones numéricas son diferentes, y, en ningún caso, se cumple la expresión anterior. Esto surge claramente de la manera de formar cantidades numéricas en el sistema romano, en el que cada uno de los símbolos utilizados tiene un valor absoluto, y la cantidad a representar se obtiene por suma o resta de los valores absolutos de cada símbolo. Por ejemplo, el número 3874 se obtiene a través de la suma de los valores absolutos de los símbolos que representan los números 1000, 500, 100, 50, 10, 5 y 1:

$$3874_{10} = \boldsymbol{MMMDCCCLXXIV}$$

Los valores absolutos de todos los símbolos se suman para formar el número representado, excepto en el caso en que un símbolo de menor valor absoluto se ubique a la izquierda de otro de mayor valor, en cuyo caso se resta, como sucede con la combinación ***IV*** para obtener el número 4.

2.2.- Sistemas de numeración para uso en computación. El sistema binario. Sistemas auxiliares: el octal y el hexadecimal. Terminología: Bit, byte, palabra (word).

Tal como se ha mencionado anteriormente, las computadoras digitales operan exclusivamente con magnitudes discretas, representadas a través de señales binarias. Esta representación, que facilita enormemente la implementación de los circuitos requeridos para tal fin, utiliza como herramienta fundamental al **sistema binario** de numeración. Este sistema, como su nombre lo indica, utiliza solamente dos símbolos, el 0 y el 1, por lo cual toda representación de cantidades numéricas deberá realizarse exclusivamente con esos dos elementos. Al contar con solamente dos dígitos, la representación binaria de cantidades requiere una gran cantidad de dígitos para números que en otro sistema, tal como el decimal, requerirán muchos menos dígitos para su representación. El sistema binario es un sistema posicional, por lo que cumple con la expresión general presentada en el apartado 2.l. Como ejemplo, considérese el número binario 11001100. Al desarrollar la suma de potencias correspondiente a la base en la que está expresado se obtiene:

$$11001100_2 = 0.2^0 + 0.2^1 + 1.2^2 + 1.2^3 + 0.2^4 + 0.2^5 + 1.2^6 + 1.2^7 = 4 + 8 + 64 + 128 = 204_{10}$$

Nótese que la serie de potencias ha permitido determinar en forma rápida el valor decimal del número binario representado, siendo este método válido para convertir al sistema decimal un valor representado en cualquier otro sistema. Es también válida la representación binaria de números fraccionarios:

$$0,11101_2 = 1.2^{-1} + 1.2^{-2} + 1.2^{-3} + 0.2^{-4} + 1.2^{-5} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} = \frac{16+8+4+1}{32} = \frac{29}{32} = 0,90625_{10}$$

A efectos de simplificar la terminología, la costumbre ha definido al dígito binario (es decir, al 0 o al 1) como un **bit**, contracción obtenida de la nomenclatura inglesa **BI**nary digi**T**. Asimismo se define como **byte** a un conjunto de 8 bits, con el cual, desde el punto de vista de la representación de cantidades, se logran definir todos los valores numéricos que van del 0 al 255. Se suele utilizar además el término **nibble** para definir un conjunto de cuatro bits. Estas definiciones particulares de conjuntos formados por un determinado número de bits se engloban dentro de un concepto

más general: el de **palabra (word),** término que se utiliza para definir un determinado número de bits con el cual una computadora puede operar en forma simultánea.

A efectos de simplificar la representación de números binarios, y como elementos auxiliares para la documentación o la programación, se utilizan habitualmente los sistemas de numeración **octal** (base 8) y **hexadecimal** (base 16). La utilización de estos dos sistemas se hace interesante por la facilidad de conversión de los valores representados en los mismos hacia y desde el sistema binario, como se verá oportunamente. El sistema de numeración octal incluye, obviamente, ocho dígitos, representados por los ocho primeros símbolos del sistema decimal: 0, 1, 2, 3, 4, 5, 6 y 7. En el caso del sistema hexadecimal, se trata de un sistema con mayor cantidad de dígitos que los símbolos que forman el sistema decimal, por lo que a los diez dígitos decimales se deberán agregar seis símbolos adicionales para completar el conjunto de 16 dígitos que forman al sistema. Por convención, se utilizan con este objetivo las primeras letras del alfabeto, por lo que en el sistema hexadecimal, el orden numérico será 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, **10,** valor este último que, de acuerdo con la definición de sistema posicional, representa la base del sistema de numeración, y que, por lo tanto, debe leerse como diez y seis.

2.3.- Conversión entre sistemas de numeración posicionales.

El significado de una misma cantidad numérica es diferente según la base en que se la represente. Esto tiene su obvia justificación en la diferente cantidad de símbolos que posee cada sistema de numeración para la representación de cantidades. A mayor base, serán menos los dígitos requeridos para una representación determinada.

Al hacer referencia a la representación de una cantidad, estamos haciendo referencia a los símbolos utilizados para dicha representación, pero no a la nomenclatura de la misma. Esto significa que cuando se expresa un número, la expresión que lo denomina tiene relación directa con su notación decimal, aún cuando el número en cuestión esté representado en otro sistema de numeración. Así, cuando por ejemplo se nombra al número "quinientos cincuenta y cinco", se está haciendo mención a una cantidad que, *en el sistema decimal y solamente en él*, estará representada por la cifra 555. Esto tiene como significado que esa misma cantidad, sin cambiar su nombre, cambiará su aspecto al ser representada en distintos sistemas de numeración, sean posicionales o no.

La metodología de conversión entre diferentes sistemas de numeración representación de una cantidad en distintos sistemas de numeración permite lograr la representación del mismo en cualquier sistema a partir de su representación decimal. Permite asimismo convertir al sistema decimal una cifra representada en algún otro sistema de numeración.

Estos dos procesos de conversión de números desde un sistema de numeración cualquiera al sistema decimal y viceversa, se basan de forma exclusiva en la expresión que define al sistema posicional. La expresión correspondiente permite resolver estas conversiones en forma sencilla según lo muestran los siguientes casos.

2.3.1.- Conversión de un número expresado en una base cualquiera a base decimal.

En apartados anteriores se ha planteado ya el método a utilizar cuando se desea determinar el valor decimal de un número expresado en una base diferente a la decimal. Como un nuevo ejemplo, la conversión del número 4535 expresado en base 6 a base 10 requiere, como se menciona en el apartado anterior, del desarrollo de la suma de potencias de la base aplicada al número y la base utilizadas. En consecuencia:

$$4535_6 = 5.6^0 + 3.6^1 + 5.6^2 + 4.6^3 = 5 + 18 + 180 + 864 = \mathbf{1067_{10}}$$

Este mismo criterio es válido para la conversión de números fraccionarios, para los cuales el concepto es similar. Por consiguiente, la expresión decimal correspondiente al número 0,2342 expresado en base 5, y llevado al sistema decimal sería:

$$0,2342_5 = 2.5^{-1} + 3.5^{-2} + 4.5^{-3} + 2.5^{-4}$$

El valor decimal de esta expresión se obtiene simplemente operando con los valores fraccionarios:

$$0,2342_5 = \frac{2}{5} + \frac{3}{25} + \frac{4}{125} + \frac{2}{625} = \frac{2.125 + 3.25 + 4.5 + 2}{625} = 0,5552_{10}$$

Para el caso en que los sistemas en que se expresan los números a convertir correspondan a bases mayores que 10, el procedimiento a utilizar será similar, con la simple precaución de recordar que en esos sistemas existen dígitos que en el sistema decimal se expresan con dos cifras:

$$2A3B_{16} = 11.16^0 + 3.16^1 + 10.16^2 + 2.16^3 = 11 + 48 + 2560 + 8192 = \mathbf{10811_{10}}$$

2.3.2.- Conversión de un número expresado en base 10 a otro sistema.

Un número cualquiera expresado en el sistema decimal puede convertirse a otro sistema posicional cualquiera, por medio de la expresión general de los sistemas posicionales vista anteriormente. El método no es tan simple como el visto para la conversión hacia base 10,. pero su fundamento teórico es el mismo: la expresión, como suma de potencias, de los números que se expresan en un sistema posicional.

Las demostraciones siguientes permitirán justificar el método de conversión utilizado tanto en el caso de números enteros como en el de fraccionarios.

Para el caso de un número entero, su expresión algebraica en la base 10 es:

$$N_{10} = b_0.10^0 + b_1.10^1 + b_2.10^2 + ... + b_n.10^n$$

Al convertirlo a una base genérica B, el mismo número, como ya ha sido dicho, se expresará a través de:

$$N_B = a_0.B^0 + a_1.B^1 + a_2.B^2 + ... + a_n.B^n$$

Siendo conocidos los coeficientes (dígitos) de la base 10, el problema radica en determinar los valores de los dígitos a_i que representan al numero en la nueva base. Dado que la segunda expresión del número es una ecuación con varias incógnitas, la única alternativa es averiguarlas una por una. A tal efecto, la expresión general permite averiguar en forma inmediata el valor del primer coeficiente, que está multiplicado por el valor 1. Esto se logra dividiendo el número N (en su expresión de base 10) por la base B a la que se lo debe convertir. De este proceder resulta la expresión siguiente:

$$\frac{N_{10}}{B} = \frac{1}{B}.(a_0. + a_1.B^1 + a_2.B^2 + ... + a_n.B^n) =$$

$$= \frac{a_0}{B} + a_1 + a_2.B^0 + a_3.B^2 + a_4.B^3 + ... + a_n.B^{n-1}$$

En esta segunda expresión puede observarse que el segundo miembro de la igualdad ha quedado formado por dos partes, una de ellas fraccionaria (a_0/B), en tanto que los demás términos que forman la expresión adoptan un valor entero. Esta expresión se asemeja, en su forma, a la definición de la operación de división entera. En efecto, la definición matemática de la división entera, la define como:

$$\frac{D}{d} = C + \frac{R}{d}$$

siendo **D** el dividendo, **d** el divisor, **C** el cociente y **R** el resto de la división.

Surge de aquí que el primer coeficiente de la representación del número N en la nueva base B se obtiene como el resto de la división entera del número N por la base B. Despejando de dicha expresión la parte fraccionaria obtenida en el segundo miembro de la misma, se obtiene un nuevo número entero:

$$\frac{N_{10}}{B} - \frac{a_0}{B} = a_1 + a_2.B^0 + a_3.B^2 + a_4.B^3 + ... + a_n.B^{n-1}$$

Dado que esta expresión es similar a la primera definición del número N, surge con claridad que con una nueva división por B se podrá obtener el segundo dígito del número en su nueva base:

$$\frac{1}{B}(\frac{N_{10}}{B} - \frac{a_0}{B}) = \frac{1}{B}(a_1 + a_2.B^0 + a_3.B^2 + a_4.B^3 + ... + a_n.B^{n-1})$$

En este caso, es a_1 el elemento que queda como parte fraccionaria de la división, el que por ende, resulta quedar definido a través del resto obtenido en la segunda división consecutiva del número N por la base B. Dado que con cada división y pasaje de la parte fraccionaria correspondiente al primer miembro de la igualdad se obtiene un resultado entero menor que el anterior, llegará un momento en que dicha parte entera sea cero. Ese es el momento en que se completa la conversión, obteniéndose el número N expresado en la nueva base como el número formado por todos los restos obtenidos, siendo el primer resto el que corresponde al dígito menos significativo (de menor peso) y el último al de mayor peso.

Ejemplo 2.1.: Se desea realizar la conversión del número 4567, expresado en base 10, a un sistema de numeración de base 6. El procedimiento involucra las siguientes operaciones:

- Dividir $\frac{4567}{6} = 761\frac{1}{6}$ *El primer resto es 1.*

- Volver a dividir el número entero resultante (761) por la base 6: $\frac{761}{6} = 126\frac{5}{6}$*. El nuevo resto es 5.*

- Dividir nuevamente: $\frac{126}{6} = 21$ *El tercer resto es 0.*

- Dividir nuevamente: $\frac{21}{6} = 3\frac{3}{6}$ *El cuarto resto es 3.*

- Una última división resultará en una parte entera nula y un resto de 3, que será el último resto obtenido. Por consiguiente, el resultado de la conversión es:

$$4567_{10} = 33051_6$$

La corrección de este resultado puede verificarse en forma rápida y sencilla, desarrollando la serie de potencias en la base 6 que permitirá retornar a la base 10 de origen:

$$33051_6 = 1.6^0 + 5.6^1 + 0.6^2 + 3.6^3 + 3.6^4 = 1 + 5.6 + 3.216 + 3.1296 = 4567$$

2.3.3.- Conversión de números fraccionarios expresados en base 10 a otra base.

Por aplicación del mismo criterio anterior, se puede justificar, utilizando únicamente la expresión general de un sistema posicional, el método a utilizar cuando el número que se desea convertir es fraccionario. En este caso, como ya es sabido, la expresión del número en el sistema decimal es:

$$\boldsymbol{N}_{10} = \boldsymbol{b}_{-1}.10^{-1} + \boldsymbol{b}_{-2}.10^{-2} + ... + \boldsymbol{b}_{-\boldsymbol{n}}.10^{-\boldsymbol{n}}$$

El mismo número, expresado en la base de destino **B**, tendrá un formato similar, desarrollado en potencias de **B:**

$$\boldsymbol{N}_{\boldsymbol{B}} = \boldsymbol{a}_{-1}.\boldsymbol{B}^{-1} + \boldsymbol{a}_{-2}.\boldsymbol{B}^{-2} + ... + \boldsymbol{a}_{-\boldsymbol{m}}.\boldsymbol{B}^{-\boldsymbol{m}}$$

En forma totalmente análoga al caso desarrollado en el apartado anterior, estamos en presencia de una expresión conocida, la del número N expresado en base 10, y otra en la cual se desconocen los coeficientes a_i. En este caso, también, para determinar esos coeficientes es necesario hacerlo uno por uno. Para esto, es fácil observar que en la expresión de destino, el primer coeficiente puede aislarse de los demás si se multiplica al número a convertir por la nueva base:

$$\boldsymbol{N}_{\boldsymbol{B}}.\boldsymbol{B} = \boldsymbol{a}_{-1}.\boldsymbol{B}^{0} + \boldsymbol{a}_{-2}.\boldsymbol{B}^{-1} + ... + \boldsymbol{a}_{-\boldsymbol{m}}.\boldsymbol{B}^{-\boldsymbol{m}+1}$$

La parte entera resultante de este producto corresponde al primer dígito (el más significativo) del número una vez convertido a la base B de destino.

El procedimiento es iterativo, por lo que al pasar el coeficiente ya determinado a_{-1} al primer miembro de la igualdad, se volverá a obtener un número enteramente fraccionario:

$$N_B.B - a_{-1}. = a_{-2}.B^{-1} + a_{-3}.B^{-2} ..... + a_{-m}.B^{-m+1}$$

Volviendo a multiplicar esta última expresión por la nueva base B se podrá determinar el valor del segundo dígito a_{-2}, y continuando con el mismo procedimiento se lograrán obtener todos los dígitos requeridos para la representación del número en la nueva base.

No obstante, existen algunas pequeñas diferencias metodológicas entre las conversiones de números enteros y el de números fraccionarios. En efecto, vimos que en el caso de números enteros la operación de conversión se completa cuando la parte entera obtenida durante las sucesivas divisiones llega a cero. En el caso de números fraccionarios, los sucesivos productos pueden llevar a situaciones diferentes, que básicamente se resumen en tres casos:

Caso 1.- Tras alguna de las multiplicaciones la parte fraccionaria del segundo miembro de la igualdad anterior se hace cero. En este caso se ha completado la conversión y el número así obtenido es un número racional en la nueva base. Ejemplo de este caso es el siguiente, en el que se trata de convertir el número decimal 0.625 a la base 2:

$$0,625.2 = 1,250; \longrightarrow a_{-1} = 1$$
$$0,250.2 = 0,500 \longrightarrow a_{-2} = 0$$
$$0,500.2 = 1,000 \longrightarrow a_{-3} = 1$$

Resulta de aquí que el resultado de la conversión es:

$$0,625_{10}=0,101_2$$

El número obtenido tiene una cantidad finita de cifras fraccionarias.

Caso 2.- Al realizar las sucesivas multiplicaciones, se determina que tras uno o más productos, la parte fraccionaria de dichas multiplicaciones se comienza a repetir: el número obtenido es periódico en la nueva base. En este caso, una vez determinado el período no es necesario seguir iterando. Como ejemplo, el mismo número anterior 0,625 se convierte a la base 5.

$$0,625.5 = 3,125 \longrightarrow a_{-1} = 3$$
$$0,125.5 = 0,625 \longrightarrow a_{-2} = 0$$
$$0,625.5 = 3,125 \longrightarrow a_{-3} = 3$$

y, como se observa, tras el segundo producto, se comienza a repetir la parte fraccionaria, por lo que de la conversión resulta:

$$0,625_{10} = 0,3030....._5$$

Caso 3.- No se puede determinar la aparición de un período ni la parte fraccionaria se logra hacer cero. En este caso, la conversión, en tanto no se demuestre lo contrario, estará dando como resultado un número irracional en la nueva base, el que, por lo tanto, tendrá infinitas cifras fraccionarias no periódicas.

Surge en este último caso una cuestión a determinar: en qué momento se puede dar el número por convertido. Cuando se produce una situación como la que se acaba de describir, el proceso de conversión debe interrumpirse con cierto criterio lógico. Este criterio puede basarse en la determinación de la exactitud con que se ha logrado convertir al número, de lo cual solo se podrá

tener una cierta idea si se conoce la relación entre las potencias negativas de 10 (la base original) y las potencias negativas de la base a la cual se está convirtiendo el número fraccionario. En este caso, la cantidad de dígitos decimales del número a convertir define la cantidad de cifras fraccionarias requeridas. Sirva el siguiente como ejemplo de lo planteado. Si se desea convertir el número $0{,}432_{10}$ al sistema de numeración de base 4, se procederá en la forma indicada anteriormente:

0,432 x 4 = 1,728	0,472 x 4 = 1,888
0,728 x 4 = 2,912	0,888 x 4 = 3,552
0,912 x 4 = 3,648	0,552 x 4 = 2,208
0,648 x 4 = 2,592	0,208 x 4 = 0,832
0,592 x 4 = 2,368	0,832 x 4 = 3,328
0,368 x 4 = 1,472	0,328 x 4 = 0,312

La operación de conversión, hasta el momento, ha dado como resultado el número $0{,}123221132030_4$. Puede suponerse que la conversión ha sido completada? Indudablemente, al no obtener un resto nulo en la parte fraccionaria, ni una expresión periódica, la misma no se ha completado. Es probable, sin embargo, que a los fines requeridos, la conversión pueda darse por terminada. Si se convierte el número obtenido a la base 10, se tendrá la siguiente expresión:

$$0{,}123221132030_4 = 1.4^{-1} + 2.4^{-2} + 3.4^{-3} + 2.4^{-4} + 2.4^{-5} + 1.4^{-6} + 1.4^{-7} + 3.4^{-8} + 2.4^{-9} + 3.4^{-11}$$

La pregunta planteada anteriormente podría revisarse en el sentido siguiente: son necesarios doce dígitos después de la coma decimal para satisfacer la conversión? Probablemente no sea así, y la respuesta se obtendrá del siguiente análisis. El número a convertir desde la base 10 estaba formado por tres cifras fraccionarias, siendo en consecuencia una cantidad entera, en este caso, de milésimos (10^{-3}). Podría suponerse que la primer potencia de la base a la que se está convirtiendo el número (en este caso 4), que tenga un valor menor que 0,001 dará una aproximación suficiente a la conversión. En este caso, la primer potencia negativa de 4 que es menor que 0,001 es 4^{-5} = 1/1024=0,0009765625.
Si se trunca la conversión en ese término, el resultado convertido sería:

$$0{,}12322_4 = 1.4^{-1} + 2.4^{-2} + 3.4^{-3} + 2.4^{-4} + 2.4^{-5} = 0{,}431640625$$

El resultado obtenido, comparado contra el número original, entrega en forma correcta las dos primeras cifras del número original. Esto es debido a que, si bien la potencia considerada es menor que la última cifra significativa del número a convertir, al estar afectado por un coeficiente (2, en este caso), su influencia se acentúa. Está claro, por otra parte, que cuantas más cifras fraccionarias se determinen, más exacta será la representación del número convertido. Agregar una posición más al valor anterior dará por resultado:

$$0{,}123221_4 = 1.4^{-1} + 2.4^{-2} + 3.4^{-3} + 2.4^{-4} + 2.4^{-5} + 1.4^{-6} = 0{,}431884765625$$

Este resultado, más satisfactorio que el anterior, se acerca más al valor original (al que, por otra parte, ya se ha dicho no se llegará nunca si no se logra encuadrar a la conversión en los dos casos antes mencionados). Agregar aún una posición más, permitirá llegar a:

$$0{,}1232211_4 = 1.4^{-1} + 2.4^{-2} + 3.4^{-3} + 2.4^{-4} + 2.4^{-5} + 1.4^{-6} + 1.4^{-7} = 0{,}43194580078$$

Nótese que la potencia de la base 4 en la que se ha truncado esta expresión (4^{-7}) es ahora menor que 0,0001, es decir, menor aún que la cifra posterior a la cantidad de dígitos originalmente utilizados

en la representación decimal del valor a convertir. En términos de precisión o exactitud, diremos que la conversión ha sido obtenida con error menor que un diezmilésimo ($\varepsilon < 0,0001$). Queda claro que haber interrumpido la conversión, como se ha hecho en el ejemplo, con doce términos posteriores a la coma decimal, solo hubiese aportado una exactitud mayor:

$$0,123221132030_4 = 1.4^{-1} + 2.4^{-2} + 3.4^{-3} + 2.4^{-4} + 2.4^{-5} + 1.4^{-6} + 1.4^{-7} + 3.4^{-8} + 2.4^{-9} + 3.4^{-11} =$$
$$= 0,43196940422$$

pero de todos modos, interrumpir antes (en 6 o 7 dígitos) hubiese dado, igualmente, un resultado satisfactorio para la mayoría de los casos.

2.3.4.- Conversión de un número desde un sistema de numeración a otro.

Cuando un número se halla expresado en un sistema que no es el decimal, y se lo desea convertir a otro sistema, que tampoco es el decimal, la dificultad mayor se ve reflejada en el hecho de que las operaciones deben realizarse en alguna de las dos bases, ambas distintas de diez. En estos casos la dificultad estriba en el hecho de que no es habitual realizar operaciones que involucren sistemas distintos del decimal: es una dificultad de orden práctico más que operativa. En consecuencia, el método habitualmente utilizado consiste en pasar a través del sistema de numeración decimal, al que se utiliza como intermediario. El número original a convertir se convierte al sistema decimal, utilizando los métodos apropiados según se trate de un número entero o fraccionario, y el número decimal así obtenido se convierte al nuevo sistema.

No obstante, en algunos casos suele permitirse la conversión directa entre un sistema y otro, **sin necesidad de pasar por el sistema decimal.** Esto sucede cuando una de las bases de los sistemas en juego es potencia directa de la otra, como ocurre entre los sistemas binario y octal ($8=2^3$), binario y hexadecimal ($16=2^4$), etc. En estos casos, la definición de sistema posicional permite encontrar un camino directo que convierta al número expresado en una de esas bases a la otra sin dificultad alguna.

Se plantea como ejemplo la conversión directa de un número **N** expresado en el sistema binario de numeración al sistema octal. Como ya se ha dicho, la expresión general que representa a dicho número en el sistema binario es:

$$\boldsymbol{N}_2 = \boldsymbol{a}_0.2^0 + \boldsymbol{a}_1.2^1 + \boldsymbol{a}_2.2^2 + \boldsymbol{a}_3.2^3 + \boldsymbol{a}_4.2^4 + \boldsymbol{a}_5.2^5 + \boldsymbol{a}_6.2^6 + \boldsymbol{a}_7.2^7 + \boldsymbol{a}_8.2^8 + \boldsymbol{a}_9.2^9 + \ldots\ldots$$

En esta expresión se conocen todos los coeficientes a_i, los que cumplen con la condición de valer solamente 0 o 1. Este mismo número entero **N**, expresado en el sistema octal al que se lo quiere convertir, tendrá una forma similar, pero ahora en potencias de ocho:

$$\boldsymbol{N}_8 = \boldsymbol{b}_0.8^0 + \boldsymbol{b}_1.8^1 + \boldsymbol{b}_2.8^2 + \boldsymbol{b}_3.8^3 + \ldots\ldots$$

En esta expresión, los coeficientes b_i cumplen con la condición de valer entre 0 y 7. En estas condiciones, existe una relación directa entre las dos expresiones del número N. En efecto, la expresión del número N en el sistema binario puede fácilmente convertirse al sistema octal, dado que en la misma aparecen (en la forma de potencias de 2) las sucesivas potencias de 8. Para visualizarlo, en la expresión binaria se agrupan los sucesivos términos de a tres (por ser $8=2^3$).

$$\boldsymbol{N}_2 = (\boldsymbol{a}_0.2^0 + \boldsymbol{a}_1.2^1 + \boldsymbol{a}_2.2^2) + (\boldsymbol{a}_3.2^3 + \boldsymbol{a}_4.2^4 + \boldsymbol{a}_5.2^5) + (\boldsymbol{a}_6.2^6 + \boldsymbol{a}_7.2^7 + \boldsymbol{a}_8.2^8) + \ldots\ldots$$

Las potencias de 8 se podrán explicitar sacando como factor común, en cada uno de los grupos, la potencia de 2 que corresponda:

$$N_2 = 2^0.(a_0.2^0 + a_1.2^1 + a_2.2^2) + 2^3.(a_3.2^0 + a_4.2^1 + a_5.2^2) + 2^6.(a_6.2^0 + a_7.2^1 + a_8.2^2) + ...$$

Dado que los subíndices incluidos en cada uno de los elementos de la expresión solo puede valer 0 o 1, queda claro que cada uno de los paréntesis adopta valores que van desde cero (cuando los tres coeficientes de cada expresión valen 0) hasta siete (cuando los tres elementos valen 1 simultáneamente). En consecuencia, cada uno de los términos que acompañan a las sucesivas potencias de 2^3 cumplen con el requisito de ser un dígito del sistema octal, y, por consiguiente, la regla que surge de esta comparación es automática: para convertir un número desde el sistema binario al octal, deben agruparse los sucesivos bits en grupos de a tres, reemplazando cada trío por su valor octal.

Este concepto se puede generalizar, planteando en consecuencia que:

Un número expresado en un sistema de numeración de base B_1 puede convertirse en forma directa a otro sistema de base B_2, si se cumple que $B_2 = B_1^n$, agrupando los dígitos de la base B_1 en grupos de n dígitos, y reemplazando el valor obtenido por el correspondiente en la nueva base.

De la misma manera, cuando la base del sistema de numeración en el que está expresado el número N es potencia de la base del sistema al que se quiere convertir ese número, se podrá realizar la conversión en forma directa, representando cada uno de los dígitos de la base original con tantos dígitos de la nueva base como corresponda a la relación de potencias entre ambas bases.

Ejemplo 2.2:

Convertir el número 11001110110011_2 al sistema hexadecimal.

Si se agrupan los dígitos binarios de a cuatro, se obtiene:

$11001110110011_2 = 11\ 0011\ 1011\ 0011 = 33B3_{16}$.

2.4.- Operaciones aritméticas.

Las operaciones aritméticas habitualmente utilizadas en el sistema de numeración decimal se aplican, en forma directa, en todo sistema de numeración posicional, no importa cual sea la base del mismo. Es cierto que, dada la costumbre de utilización del sistema decimal, cualquier operación realizada en otros sistemas de numeración resulte extraña en cuanto a su representación numérica, pero no menos cierto es, justamente, que lo único que se altera es la representación numérica del resultado y no el resultado en sí mismo. Esto es, seis más seis serán siempre doce, cualquiera sea la forma de representar el número doce (el que solamente se escribe 12 si se lo representa en el sistema decimal). Más allá de este planteo, las operaciones algebraicas habitualmente utilizadas (suma, producto, etc.), no son más que tablas de doble entrada, que se pueden aplicar en todo sistema posicional. Como ejemplo, se plantearán las tablas requeridas para sumar en los sistemas de numeración de bases 5 y 8:

B=5	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	**10**
2	2	3	4	**10**	**11**
3	3	4	**10**	**11**	**12**
4	4	**10**	**11**	**12**	**13**

B=8	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	**10**
2	2	3	4	5	6	7	**10**	**11**
3	3	4	5	6	7	**10**	**11**	**12**
4	4	5	6	7	**10**	**11**	**12**	**13**
5	5	6	7	**10**	**11**	**12**	**13**	**14**
6	6	7	**10**	**11**	**12**	**13**	**14**	**15**
7	7	**10**	**11**	**12**	**13**	**14**	**15**	**16**

Ejemplo 2.3.: Sumar los números 133242_5 y 324232_5.

Al realizar la suma, debe tenerse en cuenta que para la representación del resultado solo se dispone de los dígitos correspondientes a la base 5. Por consiguiente, al realizar la operación se deberá "pensar en decimal" y escribir los valores obtenidos con los dígitos disponibles. En consecuencia:

$$133242_5 + 324232_5 = 1013024_5$$

La corrección del resultado puede verificarse con sólo convertir los tres números al sistema decimal: $133242_5 = 5447_{10}$; $324232_5 = 11192_{10}$; $1013024_5 = 16639_{10}$, valor que corresponde al resultado de la suma.

De la misma manera se procede con las demás operaciones. La multiplicación, al igual que la suma, puede plantearse como una tabla, donde los productos difieren solamente en su representación numérica, y no en el valor del resultado. Nuevamente como ejemplo, se plantean las tablas de la multiplicación en los sistemas de numeración de bases 5 y 8.

B=5	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	**11**	**13**
3	0	3	**11**	**14**	**22**
4	0	4	**13**	**22**	**31**

B=8	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	**10**	**12**	**14**	**16**
3	0	3	6	**11**	**14**	**17**	**22**	**25**
4	0	4	**10**	**14**	**20**	**24**	**30**	**34**
5	0	5	**12**	**17**	**24**	**31**	**36**	**43**
6	0	6	**14**	**22**	**30**	**36**	**44**	**52**
7	0	7	**16**	**25**	**34**	**43**	**52**	**51**

2.5.- Complemento de un número con respecto de la base.

El concepto de **complemento** de un número es válido en cualquier sistema de numeración posicional. Sin embargo, y por razones que se justificarán en su oportunidad, este concepto se utiliza fundamentalmente en el sistema de numeración binaria para permitir la adecuada representación de números signados y para la realización de operaciones con este tipo de números.

Se define como complemento de un número con respecto a la potencia más cercana de la base como la diferencia entre dicha potencia de la base y el número que se está complementando.

Matemáticamente, esta definición se expresa mediante la igualdad

$$C_{A,B} = B^n - A$$

donde A es el número a complementar, que puede ser entero o fraccionario, B es la base del sistema de numeración en que está representado el número A y B^n es la potencia más cercana de dicha base que permita obtener el complemento como un número positivo. Esto es, el complemento de A con respecto de B se calcula siempre **por exceso**, lo que significa potencias de B que sean superiores a A. Dicho de otra manera, el complemento es un valor siempre positivo y no puede ser nulo.

Ejemplo 2.4: Calcular el complemento del número 432.
Considerando el número expresado en el sistema decimal de numeración, el cálculo del complemento, de acuerdo con la definición planteada, es:

$$C_{432,10} = 1000 - 432 = 568$$

habiéndose calculado el complemento, en este caso, con respecto a 10^3, primer potencia de la base que es mayor que el número que se desea complementar.

Ejemplo 2.5: Calcular el complemento del mismo número anterior, ahora expresado en el sistema de numeración octal.

Considerando la misma definición anterior, deberá buscarse la potencia de 8 que resulte mayor que 432_8 la que corresponde a $512=8^3$. En este caso, la diferencia a calcular para obtener el complemento será:

$$1000_8 - 432_8 = 346_8$$

En este caso, se ha realizado la resta directamente en el sistema octal de numeración, lo que puede verificarse volviendo a sumar el número obtenido como complemento al valor original utilizando la tabla de la operación de suma presentada en el apartado anterior.

En el sistema de numeración binaria, el cálculo del complemento se realiza en forma análoga a la de los ejemplos presentados. Dado que el sistema binario solamente posee dos dígitos, la operación es relativamente simple. El número a complementar se deberá restar de la potencia más cercana de 2.

Ejemplo 2.6: Calcular el complemento del número binario 100011010.

La operación a realizar para el cálculo del complemento solicitado es:

$$\begin{array}{r} 1000000000 \\ -\ 100011010 \\ \hline 11100110 \end{array}$$

Ejemplo 2.7: Calcular el complemento del número binario 10110110111

De la misma manera que en el caso anterior:

$$\begin{array}{r} 100000000000 \\ 10110110111 \\ \hline 01001001001 \end{array}$$

Ejemplo 2.8: Calcular el complemento del número binario 101001,0011

Se ha dicho, al plantear la definición de complemento, que el mismo puede definirse para cualquier número entero o fraccionario. La operación a realizar es la misma que en los casos anteriores, por lo que el complemento se obtendrá directamente mediante la operación siguiente:

$$\begin{array}{r} 1000000,0000 \\ -\ 101001,0011 \\ \hline 10110,1101 \end{array}$$

Por tratarse de un complemento que se calcula siempre restando un número (el número a complementar) de una potencia de la base en que dicho número se encuentra expresado, se lo suele denominar, en forma abreviada, **complemento a 10, lo que,** al hablar en el sistema decimal se debe interpretar como complemento a diez, y en el sistema binario, complemento a dos, nombres estos que pueden admitirse como sinónimos del de complemento a la base con el que definiéramos este concepto.

2.6.- Complemento con respecto a la base menos uno.

Con el mismo criterio con que se ha definido en el apartado anterior el complemento a la base, puede calcularse el complemento de una cantidad a cualquier otra cantidad numérica. Como caso válido para entender la extensión de este concepto, en la vida diaria suele utilizarse muy habitualmente el complemento, sin asignarle exactamente ese nombre. En efecto, lo que en una operación de pago de una compra se conoce como el "vuelto", no es otra cosa que el cálculo de un complemento. La diferencia con la definición matemática expresada anteriormente es que, en este caso particular, el "vuelto" es el complemento del monto pagado, pero con respecto de un valor que es la suma de los billetes utilizados para el pago, suma que no necesariamente es potencia de la base 10.

En algunas situaciones particulares, suele ser adecuado pensar en un **complemento a la potencia más cercana de la base menos uno,** el que se calcula según la definición siguiente:

$$\mathbf{C_{A,B-1} = C_{A,B} - 1 = B^n - A - 1}$$

lo que traducido al sistema decimal significará calcular el complemento con respecto a un número formado por tantos nueves como cifras significativas tiene el número a complementar. En el caso del sistema de numeración binaria, esta misma definición permitirá obtener el complemento como la resta del número a complementar de otro formado por tantos unos como cifras tiene el valor complementado.

Ejemplo 2.9: Calcular el complemento a la base menos 1 del número decimal 436:

De acuerdo con lo expresado, el complemento de 436 se obtiene restándolo de 999, por lo que resulta:

$$C_{436,B-1} = 999 - 436 = 563$$

Ejemplo 2.10: Calcular el complemento del número decimal 2375,275:

El criterio para calcular el complemento es el mismo que para números enteros, por lo que deberá realizarse la resta entre 9999 y el número a complementar:

$$C_{A,B-1} = 9999 - 2375,275 = 7624,725$$

La utilización del complemento a la base menos uno se hace más interesante en el sistema de numeración binaria, debido a la facilidad de cálculo del mismo. En efecto, como ya se ha dicho, el cálculo del complemento a la base menos uno en este sistema de numeración implica restar el número a complementar de otro número formado íntegramente por unos. Esta operación, según se verá en los ejemplos correspondientes, permite encontrar un método de cálculo muy sencillo, que requiere simplemente la inversión del número, reemplazando aquellos dígitos que originalmente son ceros por unos y viceversa.

Ejemplo 2.11: Calcular el complemento a la base menos uno del número binario 1011011101.

De acuerdo con la definición, se deberá restar el número del anterior a la potencia siguiente de la base

$$\begin{array}{r} 1111111111 \\ -\ 1011011101 \\ \hline 0100100010 \end{array}$$

2.7. Utilización del complemento en operaciones de resta.

El concepto de complemento recién enunciado puede ser utilizado para simplificar las operaciones de resta entre números. Si bien es cierto que la operación de resta es habitual, en lo que hace a su utilización cotidiana, resulta de cálculo algo más engorroso que la operación de suma. En especial, en la utilización de máquinas que operan en binario, como las computadoras, y especialmente en la primera época de las mismas, resultó más sencillo, por diversas circunstancias, convertir la resta en una operación en la que participase el concepto de complemento. Es de fácil demostración la forma de convertir una operación de resta en otra, de suma, en la que intervenga el complemento del sustrayendo de la resta.

Sea la operación de resta definida como:

$$\mathbf{R = S - T}$$

Esta expresión no sufre cambios si en el segundo miembro de la misma se suma y se resta un mismo número. En este caso es válido plantear que:

$$R = S \mathbf{+ B^n - B^n} - T$$

Se ha sumado y restado una potencia de la base utilizada, tal que sea mayor que S y que T. En este caso, reacomodando la expresión, se tiene:

$$\mathbf{R} = S + B^n - T - B^n = \mathbf{S + C_{T,B} - B^n}$$

La resta entre S y T se convierte, prácticamente, en la suma de S más el complemento de T. La operación se completará eliminando la potencia de la base que se utilizó para el cálculo del complemento. Si el resultado de la resta es positivo, este último paso será muy simple, dado que el número a eliminar es un uno seguido por una cantidad de ceros. Los ejemplos siguientes permiten interpretar esta definición, con aplicación a distintos sistemas de numeración.

Ejemplo 2.12: Sea restar los números 475 y 324, expresados en el sistema decimal de numeración.

Está claro que la resta entre dos números decimales es una operación sencilla de hacer, sin necesidad de incorporar conceptos "extraños" a la naturaleza de la resta. En este caso, la resta a calcular da como resultado el número 151.

Si se aplica la definición de complemento, se deberá realizar la siguiente secuencia de pasos:

1.- Calcular el complemento de 324:

$$1000 - 324 = 676$$

2.- Sumar el complemento así calculado con el número 475.

$$475 + 676 = 1\ 151$$

3.- Restar, del resultado obtenido, la potencia de la base utilizada para calcular el complemento.

$$1151 - 1000 = 151$$

Se podrá decir que esta secuencia de operaciones es más larga que el cálculo directo de la resta en la forma convencional, lo cual es cierto cuando se opera en el sistema decimal, debido a la costumbre ya adquirida desde siempre que permite calcular las restas metódicamente.

No obstante, se verá que, en el caso de las máquinas que operan con números en forma binaria, la utilización del complemento para convertir las restas en sumas ahorra circuitos lógicos, dado que las dos operaciones, suma y resta, se resuelven con la misma estructura circuital. Este ahorro fue fundamental e importante en las primeras etapas de la tecnología de calculadoras y computadoras, en las que la complejidad y el costo del hardware requerido para la implementación de las distintas operaciones, por muy simples que fueran, era parte importante dentro del sistema.

Ejemplo 2.13: Calcular, utilizando el complemento a la base, la resta entre 324 y 475.

La operación, realizada por los métodos convencionales, implica restar en el sentido posible (475-324) y asignar al resultado el signo negativo, dado que ese es el signo del mayor de los dos operandos.

Si se realiza la resta aplicando los conceptos anteriores, se podrá plantear una secuencia de pasos similar a la del ejemplo anterior:

1.- Calcular el complemento a la base de 475:

$$1000 - 475 = 525$$

2.- Sumar el complemento así calculado con el número 324.

$$324 + 525 = 849$$

3.- Restar, del resultado obtenido, la potencia de la base utilizada para calcular el complemento.

$$849 - 1000 = -151$$

Nótese que en este caso, la potencia de 10 utilizada para calcular el complemento del sustraendo no apareció en forma explícita como en el caso del ejemplo 2.12. Por el contrario, debió realizarse la

resta entre el resultado obtenido en el paso 2 y esa potencia de la base, lo que, en breve, significa recalcular el complemento del resultado obtenido para lograr el verdadero resultado de la operación.

En la práctica, lo que este ejemplo demuestra es que, en aquellos casos en que el resultado de una resta es negativo, lo que se obtiene al calcular la suma entre el minuendo y el complemento del sustraendo ($S + C_{T,B}$) es el complemento del resultado real de la resta.

Ejemplo 2.14: Realizar la operación del ejemplo 2.12, pero con los operandos expresados en el sistema binario de numeración.

Se supone conocido el procedimiento de conversión por medio del cual se obtienen:

$324_{10} = 101000100_2$; $475_{10} = 111011011_2$

Si se desea calcular la resta 475 - 324 solamente habrá que calcular el complemento del número 324, para lo cual ya se han visto los métodos apropiados.

Por consiguiente, los tres pasos planteados en los ejemplos anteriores son:

1.- Calcular el complemento de 324, ahora con respecto a la base 2 (los números están presentados en binario).:

$C_{324,2} = 010111100$

2.- Sumar el complemento calculado con el minuendo 475:

$$\begin{array}{r} 111011011 \\ +\underline{010111100} \\ 1010010111 \end{array}$$

3.- Eliminar la potencia de 2 con respecto de la cual se calcuóo el complemento, lo que significa, simplemente, descartar el bit que excede la cantidad de bits usados para la representación.

El resultado final es 010010111_2, lo que convertido nuevamente a decimal, representa el número 151.

Ejemplo 2.15: Restar, en binario, 324 - 475.

Se procede de la misma manera que en el caso anterior, calculando el complemento del sustraendo.

$324_{10} = 101000100_2$; $475_{10} = 111011011_2$
$C_{475,2} = 000100101$

La suma de 324 con el complemento de 475 resulta dar:

$$\begin{array}{rr} & 101000100 \\ + & \underline{000100101} \\ & 101101001 \end{array}$$

Para obtener el resultado real de la operación, debe complementarse el resultado obtenido (no hay ningún bit excediendo la longitud de los dos operandos). Dado que el resultado es menor que la potencia de dos respecto de la cual se calculó el complemento, debe interpretarse que dicho resultado es negativo.

$C_{101101001,2} = 010010111$, coincidente con el resultado del ejemplo anterior. El resultado de la resta es - 151.

2.8.- Representación de información numérica. El caso de los números con signo.

Se ha definido con anterioridad el concepto de **palabra** para hacer referencia a la cantidad (fija) de bits que se utilizan en un sistema de computación cuando se almacena información. Dado que la cantidad de combinaciones que pueden obtenerse con **n** bits es 2^n, cuando estos n bits se requieren para representar información numérica, queda claramente establecido el **rango** o alcance de los números a almacenar. Un total de n bits permitirá representar números que van desde 0 hasta $2^n - 1$, siendo todos ellos positivos. No es común que un sistema de computación opere solamente con números positivos. Sin embargo, en todos los ejemplos presentados hasta el momento se ha hecho referencia, en forma excluyente, a la representación y simbolización de números positivos. Obviamente, esto no significa que en otros sistemas de numeración se desconozcan los números negativos, ya que, con el mismo criterio utilizado en el sistema decimal, se podrá representar un número negativo anteponiendo el signo menos (-) al valor a representar.

En el caso particular de la utilización de computadoras o sistemas de cálculo que utilizan exclusivamente ceros y unos (el sistema binario) para la representación de la información a almacenar o procesar, debe adecuarse la representación de los números enteros, positivos y negativos, a la limitación que significa esa utilización de sólo dos dígitos.

Desde el punto de vista conceptual, puede considerarse que el signo de una cantidad es una magnitud discreta, que adopta solamente dos valores, positivo y negativo. Por ende, y sobre esa base, se puede establecer una relación vinculante entre los dos posibles valores del signo y los dos dígitos del sistema binario. Esto significa que la representación de un número signado se puede llevar a una representación enteramente binaria donde la caracterización del número (si es positivo o negativo) se haga a través del agregado de un **bit de signo** que represente, mediante una convención, si el número que se está expresando es positivo o negativo. Esta convención permitirá entonces representar cualquier número entero signado mediante un valor numérico binario, uno de cuyos bits, generalmente el más significativo, se considerará como indicativo del signo del número representado.

Esto significa que, para un sistema que almacene información considerando palabras de **n** bits, no cambiará la cantidad de combinaciones que podrán representarse, las que seguirán siendo 2^n, sino que deberá cambiar la forma de interpretar esas 2n combinaciones, las que dejarán de estar en el rango **$0..2^n-1$** para pasar a representar un rango diferente, dado que se "pierde" un bit para representar el signo. Además, la representación estará centrada en el valor 0, lo que significará que en el caso de números con signo, **n** bits representan los números válidos dentro del rango

$$\mathbf{0 .. \pm 2^{n-1}-1.}$$

En el caso de una máquina o sistema digital que almacena información en 8 bits, si la misma corresponde a información numérica siempre positiva, los 8 bits permitirán representar números en el rango **0..255**, en tanto que, si se trata de números con signo, el mismo número de bits representará valores en el rango **-127 .. +127**.

Por otra parte, la convención a utilizar no solamente debe permitir la adecuada diferenciación entre los números positivos y los negativos sino que, además, debe servir para la realización de operaciones aritméticas que incluyan dichos números signados. Esto significa que, si bien uno de los dígitos usados indicará el signo del número representado, al realizar una operación aritmética, dicho bit deberá interpretarse, para la operación a realizar, como uno más de los dígitos del número, sin hacer diferenciación alguna en la operación. Por ende, la posición que ocupa el bit del signo de cada uno de los operandos deberá coincidir con la del resultado, y, además, el bit de signo del resultado deberá reflejar correctamente, en valor y signo, la operación completada.

En las convenciones que se analizarán a continuación, habitualmente utilizadas en los sistemas digitales, se utilizará siempre la misma representación para los **números positivos**, los que se representan a través de su **valor absoluto y bit de signo en cero.** En el caso de los números negativos, se plantearán distintas alternativas de representación, basadas en los conceptos anteriormente analizados, dentro de las cuales se podrá observar que no todas ellas cumplen con las dos condiciones de permitir la representación de los números y la realización de las operaciones aritméticas requeridas.

2.8.1.- Representación en valor absoluto y signo.

En esta representación, como ya ha quedado establecido, los números positivos se representan por su valor absoluto y con el bit de signo en **cero**. La longitud de la palabra incluye al bit de signo.

Ejemplo 2.16: Representar el número +72, considerando que la información se almacena en ocho bits.

En este caso, el bit más significativo representará el signo del número, por lo que la representación de +72 es 01001000.

Los números negativos, por su parte, se representarán también a través de su valor absoluto, con el bit de signo en **uno**.

Ejemplo 2.17: Representar el número -72, considerando la misma estructura de ocho bits.

Al igual que en el caso anterior, el bit más significativo representará el signo, utilizándose los otros siete bits para representar el número mediante su valor absoluto, por lo que $-72_{10} = 11001000_2$.

Esta convención para la representación de números enteros es muy simple y cómoda de implementar, dado que entre los números positivos y los negativos solamente existe una diferencia en el bit de signo. No obstante, se puede observar fácilmente que la misma no tiene gran utilidad, dado que no cumple con una de las exigencias planteadas: no permite hacer operaciones aritméticas.

En efecto, con solamente sumar los dos números representados en los ejemplos anteriores, se podrá observar que la suma de +72 con -72, que debería dar cero, al utilizar la representación anterior da por resultado un valor no nulo:

+ 72	0 1001000
+ - 72	+ 1 1001000
0	1 1010000 que no es cero.

2.8.2.- Representación en complemento a la base y bit de signo.

El concepto de complemento permite resolver la limitación anterior, encontrando en consecuencia una forma de representar números negativos, permitiendo además utilizar esa representación para realizar operaciones aritméticas.

En efecto, el complemento de un número es único; existe una relación biunívoca entre un número dado y su complemento. Por consiguiente, utilizar el complemento para representar un número es una representación válida, que no produce conflicto en lo que hace a dicha representación.

Solamente habrá que demostrar que, una vez representado el número a través de su complemento, las operaciones aritméticas realizadas son válidas.

En este tipo de representación los números positivos se representan, al igual que en el caso anterior, mediante su valor absoluto, tomando el bit de signo como cero. En cambio, ***se utilizará el complemento únicamente para representar los números negativos*, siendo en este caso el bit de signo igual a uno**.

Ejemplo 2.18: Representar el número - 72, tomado con 8 bits, en representación de complemento a la base y signo.

Del ejemplo 2.8.1 se tiene la representación del número positivo 72:

+ 72 = 0 100 1000

La representación del número negativo -72 se obtendrá directamente complementando el número positivo anterior, incluido el bit de signo:

*- 72 = 1 0000 0000 - 0100 1000 = **1** 011 1000*

Al realizar la suma entre los dos números (+72 y -72) se verificará el resultado esperado: cero.

+ 72 = 0 100 1000
+ - 72 = 1 011 1000
10 000 0000

Debe notarse la aparición de un bit adicional, de arrastre (carry), que se agrega a los ocho bits originales con que se han representado los datos. La justificación de esta aparición es la misma ya planteada en el caso de las operaciones de resta a través de complemento. Al calcular el complemento del sustraendo para realizar la operación se ha debido sumar una potencia de la base (para permitir el cálculo de dicho complemento) que es la que ahora excede el resultado. Por consiguiente todo lo que hay que hacer es descartarlo. Esto se verá con mayor detalle al desarrollar las operaciones aritméticas con números signados en los apartados siguientes.

2.8.3.- Representación en complemento a la base menos uno y bit de signo.

Así como se ha utilizado la notación de complemento a base para representar números signados, puede utilizarse también el complemento a la base menos 1. En esta notación, igual que en los casos anteriores, la utilización del complemento queda reservada a la representación de los números negativos. Los números positivos, al igual que en las otras dos notaciones, se seguirán representando a través de su valor absoluto, con bit de signo en cero. Los números negativos, en este caso, se representarán a través de su complemento a la base menos uno y bit de signo en uno. La utilización de este tipo de esquema para representar números signados tiene una justificación relativamente simple: el cálculo del complemento a la base menos uno es más sencillo, ya que se lo puede calcular reemplazando, en el número a complementar, los ceros por unos y los unos por ceros.

Ejemplo 2.19: Representar el número - 72, tomado con 8 bits, en representación de complemento a la base menos uno y signo.

Del ejemplo 2.16 se tiene la representación del número positivo 72:

+ 72 = 0 100 1000

La representación del número negativo -72 se obtendrá, al igual que en el ejemplo anterior, complementando, ahora a la base menos uno, el número positivo, incluido el bit de signo:

- 72 = 1111 1111 - 0100 1000 = ***1*** *011 0111*

2.8.4.- Rangos de representación.

Ha quedado dicho ya que un conjunto de n bits, utilizado para representar números enteros, permite la representación de 2^n combinaciones, centradas en cero. Corresponde explicar ahora, más en detalle, cual es el rango de representación que se logra. Podrá observarse que, a diferencia del caso de números naturales, donde se utilizan efectivamente las 2^n combinaciones, en el caso de números enteros, habrá alguna combinación que deberá ser descartada.

Si ejemplificamos con ocho bits, queda claro que esta cantidad de bits involucra 256 combinaciones numéricas, las que servirán para representar, **si no se utiliza signo**, los números naturales que van desde 0 hasta 255. Si se trabaja con números signados, en cambio, de acuerdo con la definición, el rango de números positivos a representar será, cualquiera sea la convención a utilizar, el comprendido entre 0 y +127:

0	0 000 0000
.	. ...
.	. ...
+ 127	0 111 1111

Para el caso de los números negativos, la forma de representación será distinta según la convención a utilizar. No obstante se verá que el rango de representación es el mismo, y que solamente varía la forma numérica de representar los distintos valores.

En la representación **de valor absoluto y bit de signo**, los números a interpretar como negativos son aquellos que tienen su bit más significativo en uno. Esto haría que el rango representable sea el que va desde 1000 0000 hasta 1111 1111. Dado que el rango de representación de números negativos es el comprendido entre -1 y -127, queda claro que existen 127 números negativos a representar, pero 128 combinaciones binarias para establecer esa representación. Si se analiza el primero de los valores se observa que, de acuerdo con la convención establecida, el número signado 1000 0000 correspondería a -0. Dado que el cero, como cantidad, no tiene signo, esta combinación, en principio, no tiene significado numérico alguno, y, por consiguiente, el rango de validez de esta convención, para el caso de números negativos es:

-1	1 000 0001
.	. ...
.	. ...
-127	1 111 1111

En el caso de la representación en **complemento a la base y bit de signo** surge una situación similar. En este caso, recordando que los números negativos se representan a través de su complemento a la base, el rango de números negativos a representar es:

-1	1 111 1111	(complemento de 0 000 0001)
.	. ...	
.	. ...	
-127	1 000 0001	(complemento de 0 111 1111)

En este caso, la combinación restante, 1000 0000, puede plantearse como un elemento a utilizar. En efecto, al pretender complementar el valor numérico 1000 0000 (que representa un número negativo por su signo en 1), se vuelve a obtener el mismo valor 1000 0000. Este valor, ahora positivo, corresponde al número decimal 128, por lo que resulta que 1000 0000 es la representación binaria de –128, único valor formado por ocho bits que, complementado (ahora a 256) se convierte en sí mismo.

El rango de representación admitido para la convención de complemento a la base y bit de signo, para ocho bits, es entonces el rango que va desde +127 hasta –128.

La tercer representación posible para los números signados es la de **complemento a la base menos uno y bit de signo**, en la que los números negativos, por una razón fundamentalmente práctica (basada en la sencillez de cálculo de dicho complemento en el sistema binario), se representan a través de su complemento a la base menos uno.

También en este caso quedará una combinación sin utilizar. Su significado es el mismo de las otras dos convenciones, pero, por las características de la representación, no existirá posibilidad de confusión con respecto al significado de la misma.

Efectivamente, en la representación de números negativos a través del complemento a la base menos uno, el rango de representación válido para el caso de 8 bits será:

-1	1 111 1110	(complemento a la base menos 1 de 0 000 0001)
.	. ...	
.	. ...	
- 127	1 000 0000	(complemento a la base menos 1 de 0 111 1111)

La combinación no utilizada en esta convención corresponde al número binario 1 111 1111, que no tiene significado alguno.

Vale la pena insistir, una vez planteadas las distintas alternativas de representación, que este concepto se extiende a cualquier número de bits. Es decir, en todos los casos en que se representen números enteros y se utilicen n bits, al tomar uno de ellos como bit de signo, el rango disponible permitirá representar los números en el rango

$$\pm 2^{n-1} - 1$$

Debe quedar clara la diferencia con el caso de la representación de números naturales (solo positivos) utilizando la misma cantidad de bits, caso en el cual se podrán representar las combinaciones numéricas contenidas en el rango

$$0 .. 2^{n} - 1$$

2.9.- Operaciones con números enteros. Suma, resta, suma algebraica.

Las convenciones vistas en los apartados anteriores, que permiten la representación binaria de números enteros con signo, permiten además simplificar la realización de operaciones de suma y resta, al ofrecer una forma simple de realizarlas, sin tener que analizar especialmente los problemas de signos involucrados.

Con el objeto de simplificar estas operaciones, es de práctica plantear algunas reglas que, aplicadas normalmente, permitirán obtener los resultados adecuados, o, mejor dicho, interpretar adecuadamente los resultados obtenidos. Los sucesivos ejemplos a desarrollar aclararán el tema.

Estas reglas básicas consisten en plantear los siguientes conceptos:

- Para sumar o restar números con signo debe utilizarse alguna de las convenciones que representan los números negativos a través del complemento. Ya se ha visto que la representación en valor absoluto y bit de signo no sirve para realizar operaciones.

- Los bits de signo de los operandos forman parte de la operación. Esto significa que la cantidad de bits utilizados en la representación debe ser la misma para ambos operandos. De esta manera, los bits de signo quedarán encolumnados.

- Todo número negativo se representará a través de su complemento y bit de signo en uno.

- Las operaciones de resta se realizarán, como ya se ha visto, a través de la suma del complemento del sustraendo. ‘

- Al restar un número negativo de otro, no importa de qué signo, la aplicación de la regla de los signos llevará a convertir la resta en una suma. Al respecto de los complementos, esto corresponde dado que el número negativo se habrá obtenido complementando el positivo, y al restarlo, volver a complementar significará volver a obtener el valor positivo original.

De esta manera se obtendrá, en aquellas operaciones que entreguen resultado positivo, el resultado correcto de la operación. En el caso en que el resultado a entregar sea negativo, se obtendrá el complemento del resultado real, por lo que la interpretación del valor obtenido será tan simple como volver a complementar el resultado entregado por la operación que se haya realizado.

Ejemplo 2.20: Sumar los números positivos + 36 y +54. Representar los operandos con 8 bits incluido signo.

+ 36 = 0 010 0100 *+ 54 = 0 011 0110*

Al proceder a la suma se obtendrá:

$$\begin{array}{ll} & +\ 36 = 0\ 010\ 0100 \\ + & \underline{+\ 54 = 0\ 011\ 0110} \\ & 0\ 101\ 1010 \end{array}$$

resultado binario que corresponde al número positivo + 90.

Obsérvese que la columna de los bits de signo dio como resultado cero, lo que corresponde a la codificación del signo de un número positivo.

Ejemplo 2.21: Sumar los números negativos - 36 y - 54. Representar los operandos con 8 bits incluido signo.

Del ejemplo anterior se obtuvieron los valores binarios de los números positivos:

+ 36 = 0 010 0100 *+ 54 = 0 011 0110*

Al ser ambos negativos corresponderá representarlos mediante sus respectivos complementos:

- 36 = 1 101 1100 *- 54 = 1 100 1010*

los que deberán sumarse para obtener:

- 36 = 1 101 1100
+ - 54 = 1 100 1010
1 1 010 0110
bit de arrastre ***↑↑ bit de signo***

resultado binario que corresponde a un número negativo (su bit de signo es 1). Por consiguiente, para determinar el resultado de la operación deberá complementarse dicho resultado, obteniéndose así 0 101 1010, que resulta ser el mismo resultado de la operación del ejemplo anterior, es decir, 90. Por consiguiente, el resultado de la resta es -90.

Obsérvese en este último ejemplo, la aparición en el resultado, a la izquierda (más significativo) del bit de signo, de un bit adicional, al que se conoce como **bit de arrastre o carry.** La aparición de este bit no debe sorprender dado que surge directamente de la expresión que se ha deducido para permitir la realización de restas mediante complementos. Por consiguiente, todo lo que debe hacerse es ignorar la existencia de dicho bit.

Ejemplo 2.22: Sumar los números + 36 y - 54. Representar los operandos con 8 bits incluido signo.

De los ejemplos anteriores se obtuvieron los valores binarios de los números a utilizar, tanto positivos como negativos:

+ 36 = 0 010 0100 *+ 54 = 0 011 0110*
- 36 = 1 101 1100 *- 54 = 1 100 1010*

Para realizar la operación requerida, se deberán sumar:

+ 36 = 0 010 0100
+ - 54 = 1 100 1010
1 110 1110
bit de arrastre ***↑↑ bit de signo***

resultado binario que corresponde a un número negativo (su bit de signo es 1). Por consiguiente, al igual que en el ejemplo anterior, para determinar el resultado de la operación deberá complementarse dicho resultado, obteniéndose así 0 001 0010, de donde el resultado de la operación, como corresponde, es -18.

Nótese que en este caso no ha aparecido el bit de arrastre planteado en el caso anterior. Su desaparición no es casual, sino que se produce por las mismas causas vistas al analizar la resta entre números naturales. De su falta, por consiguiente, surge la necesidad de complementar el resultado obtenido en la suma para llegar al resultado correcto de la operación.

Ejemplo 2.23: Sumar los números - 36 y + 54. Representar los operandos con 8 bits incluido signo.

De los ejemplos anteriores se obtuvieron los valores binarios de los números a utilizar, tanto positivos como negativos:

+ 36 = 0 010 0100 *+ 54 = 0 011 0110*
- 36 = 1 101 1100 *- 54 = 1 100 1010*

Para realizar la operación requerida, se deberán sumar:

- 36 = 1 101 1100
\+ *+ 54 = 0 011 0110*
***1** 0 001 0010*
arrastre ↑

resultado binario que corresponde a un número positivo, dado que el bit de signo es cero. El resultado de la operación es + 18.

El bit de arrastre aparecido en este ejemplo sufrirá la misma consideración que en el caso anterior, lo que significa lisa y llanamente su desaparición debido, (si es que se busca una justificación) a la necesidad de cumplir con la expresión que permite convertir una resta entre dos números en una suma entre un número y el complemento del otro:

$$\mathbf{R} = S + B^n - T - B^n = \mathbf{S + C_{T,B} - B^n}$$

bit de arrastre ↑

Ejemplo 2.24: Un caso especial.

Sumar, en formato de 8 bits con signo incluido, los números + 72 y +60

Esta simple suma, de dos números positivos, encierra una situación que hasta ahora no había aparecido en los ejemplos anteriores.

Siguiendo el mismo método anterior, se expresan los dos operandos en 8 bits:

+ 72 = 0 100 1000 *+ 60 = 0 011 1100*

Al proceder a la suma, se tendrá:

+ 72 = 0 100 1000
\+ *+ 60 = 0 011 1100*
= 1 000 0100
BIT DE SIGNO NEGATIVO !!! ↑

El resultado de una suma de dos números que fueron definidos como positivos, (y así se observa de la representación binaria de los mismos) dio un resultado negativo. Este "problema", que no debe considerarse como tal, merece un análisis más detallado de la situación para determinar su causa.

Los dos números utilizados en el ejemplo, 60 y 72, son números que han sido correctamente representados en el formato de siete bits más signo. No obstante, surge el problema al obtener el resultado de la suma. La respuesta es casi inmediata: al utilizar 8 bits, incluyendo uno para signo, el rango admitido para representar números con signo, es + 127 → -127. Está claro que la suma de

+60 y +72 es +132, lo que lleva a un resultado que **no puede representarse en ocho bits incluyendo signo.** El total de los ocho bits representa correctamente al resultado de la operación: el número + 132. No obstante, no se está considerando signo dentro de esos ocho bits.

Esta situación, habitual en las operaciones realizadas con números signados, se conoce como **overflow** o **exceso (**para algunos autores **desborde** o **rebalse).** El significado de la expresión pretende graficar lo ocurrido**: el número a representar como resultado de la operación excede la capacidad de almacenamiento del campo numérico en que se lo quiere almacenar.**

En la realización de operaciones aritméticas por parte de las máquinas que utilizan la notación signada para sus operandos, el asunto es de fácil solución, dado que la situación planteada solamente puede producirse en aquellos casos en que el resultado de una operación exceda la capacidad de almacenamiento del campo. Esta situación solamente puede darse, y no siempre, en el caso de una suma entre dos operandos que tengan el mismo signo. En este caso, el signo del resultado se corresponde con el signo de los operandos (si los dos operandos fueron positivos el resultado debe necesariamente ser positivo) y el bit que está ocupando el lugar del bit de signo en el resultado es el bit más significativo del mismo.

Ejemplo 2.25: El mismo caso especial, llevado a la suma de dos números negativos.

Sumar, en formato de 8 bits con signo incluido, los números -72 y - 60

Siguiendo el mismo método anterior, se expresan los dos operandos en 8 bits:

+ 72 = 0 100 1000 *+ 60 = 0 011 1100*

y se obtienen los complementos para proceder a la suma:

- 72 = 1 011 1000 *- 60 = 1 100 0100*

Al proceder a la suma, se tendrá:

```
      - 72 =  1 011 1000
+     - 60 =  1 100 0100
      ------------------
      1 0 111 1100
```

BIT DE SIGNO POSITIVO !!! ↑

Esta situación deberá interpretarse de la misma manera que en el caso anterior. El resultado de la suma de dos números negativos no puede ser sino negativo. Por consiguiente, el resultado es negativo y está expresado en ocho bits. Al complementar dicho resultado se obtendrá 1 000 0100 = -132, que es el resultado correcto de la operación.

Hasta ahora se han planteado ejemplos en los que la operación a realizar siempre fue una suma. En algunos casos los sumandos eran positivos, en otros casos negativos. Una resta entre números signados puede considerarse incluida dentro de cualquiera de los ejemplos anteriores, dado que la aplicación correcta de las reglas relacionadas con los signos y la utilización de las pautas establecidas al principio de este apartado, permitirán resolver cualquier situación.

Ejemplo 2.9.7. - Qué ocurre con la resta entre números signados?

Realizar la resta entre + 42 y -39.

Queda claro, desde el punto de vista algebraico, que + 42 - (-39) = + 42 + 39 = + 81.

Por lo tanto, la solución de esta operación consiste simplemente en realizar la suma propuesta, habiendo aplicado previamente la regla de los signos.

De realizar la operación en forma sistemática, los operandos serán:

+ 42 = 0 010 1010 - 39 = 1 101 1001

Para realizar la operación deberá sumarse al número + 42 el complemento del número - 39. Por consiguiente:

$$\begin{array}{ll} & + 42 = 0\ 010\ 1010 \\ + & C_{-39} = 0\ 010\ 0111 \\ \hline & 0\ 101\ 0001 \end{array}$$

El resultado es, naturalmente, 81, dado que al realizar el complemento de -39 se ha vuelto a obtener el número positivo 39, que fue sumado a +42, convirtiendo la operación de resta en una suma de dos números positivos.

2.10.- Ampliación de la representación de números enteros. Formatos de representación extendida.

Como se ha visto en los apartados anteriores, la representación de números enteros en forma binaria se realiza mediante una cantidad determinada de bits, por lo que, de acuerdo con la cantidad de bits a utilizar se determina el rango de números que pueden representarse. Este formato, que se conoce como **representación de punto fijo**, hace referencia al punto decimal implícito permanentemente en la posición menos significativa del número representado.

Cuando se requiere representar números enteros de un valor mayor que el rango disponible por la cantidad de bits que se están utilizando, se puede multiplicar la cantidad de bits utilizados para permitir la representación de esos números que se encuentran fuera del rango. Por ejemplo, si la representación utilizada requiere 8 bits como formato standard de palabra, se pueden utilizar dos palabras de 8 bits para obtener una única palabra de 16 bits, llevando entonces el rango total de representación al rango 0..65535 para números naturales o al rango - 32767 .. + 32767 para números enteros. Esta forma de aumentar el rango de representación mediante la utilización conjunta de dos palabras se conoce como formato de **punto fijo de doble precisión.** El bit de signo de la palabra compuesta así obtenida será el bit más significativo de la palabra doble obtenida. De la misma manera se puede obtener un formato de triple precisión, vinculando tres palabras, cuádruple precisión, vinculando cuatro palabras, y, en general, se hablará de una representación de **punto fijo de múltiple precisión** cuando se requieran utilizar múltiples palabras en conjunto para almacenar un valor entero.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Figura 2.1.- Representación de un número natural de 8 bits (sin signo) en formato de punto fijo.

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_{9}	b_{8}	b_{7}	b_{6}	b_{5}	b_{4}	b_{3}	b_{2}	b_{1}	b_{0}

Figura 2.2.- Representación de un número natural en doble precisión (16 bit sin signo)

	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
b_{signo}	b_{6}	b_{5}	b_{4}	b_{3}	b_{2}	b_{1}	b_{0}

Figura 2.3.- Representación de un número entero de 8 bits (con signo) en formato de punto fijo.

	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
b_{signo}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_{9}	b_{8}	b_{7}	b_{6}	b_{5}	b_{4}	b_{3}	b_{2}	b_{1}	b_{0}

Figura 2.4.- Representación de un número natural en doble precisión (16 bit con signo)

2.11.- Representación exponencial. El concepto de punto flotante.

No siempre la utilización de formatos de punto fijo resuelve todos los problemas de representación de números. En efecto, si bien la representación de múltiple precisión permite, con ciertas limitaciones, representar cualquier número entero, quedan fuera de dicha representación los números fraccionarios. En efecto, hasta el momento no se ha hecho referencia, en ningún caso de los ejemplificados, a la existencia de números que no fueran enteros. Sin embargo, los números fraccionarios existen, y por lo tanto, requieren ser representados de alguna manera.

El formato a utilizar en la representación de números fraccionarios (aunque también sirve para la representación de números naturales o enteros) es el que se conoce como **representación en punto flotante**. El concepto surge en contraposición al de punto fijo, ya que la coma decimal no ocupa un lugar determinado dentro de la representación del número.

La representación numérica punto flotante es una herramienta conocida en diferentes aplicaciones matemáticas y físicas. Consiste en la **representación exponencial** de los números **reales**, lo que significa que una determinada cantidad se representa como el producto de una cifra, llamada **mantisa**, por una potencia de la **base** del sistema de numeración en la que se lo está representando. Es por eso que la representación se conoce, asimismo, como representación en **mantisa y exponente**.

Esta simbolización no depende del sistema de numeración utilizado para representar los números, aunque, en computadoras, es permanentemente utilizada para representar números que, por diferentes razones (ser números fraccionarios, o números enteros que requerirían una representación de muchos bits) no se pueden representar en los formatos simples o extendidos de punto fijo.

Un número N cualquiera, representado en una base B, se presentará como el producto

$$\mathbf{N = m \cdot B^{n}}$$

siendo **m** la mantisa, **B** la base del sistema de numeración en que se está trabajando, y **n** el exponente requerido para llegar al número **N**. Queda claro que esta expresión puede adoptar innumerables valores, ya que se obtendrán expresiones equivalentes si se procede a multiplicar y dividir el número por una misma potencia de la base B.
Como ejemplo, el número decimal 475,376 puede representarse en notación exponencial, entre otras, de las distintas maneras siguientes:

$475{,}376 = 475{,}376 \times 10^0 =$
$= 47{,}5376 \times 10^1 =$
$\mathbf{= 4{,}75376 \times 10^2 =}$ **(1)**
$\mathbf{= 0.475376 \times 10^3 =}$ **(2)**
$= 0{,}0475376 \times 10^4 =$
$= 0{,}00475376 \times 10^5 =$
$= 4753{,}76 \times 10^{-1} =$
$= 47537{,}6 \times 10^{-2} =$
$= 475376 \times 10^{-3} =$
...

Obsérvese que todas estas expresiones son equivalentes, y representan la misma cantidad. Definir cual de las diferentes representaciones es la que debe utilizarse es uno de los primeros conceptos que permitirán aprovechar el sistema de representación en punto flotante.

En el ejemplo se han resaltado dos de las posibles alternativas de representación. En la primera de ellas, la mantisa adopta un valor de un sólo dígito, lo que significa **10 > m > 1**. Es una forma de representación utilizada habitualmente en ambientes científicos, dado que la mantisa de un sólo dígito entero ofrece algunas ventajas. El tamaño de la magnitud a representar está definido por el exponente de la expresión. Observando dicho exponente se puede tener en claro si el número representado es menor que la unidad (exponente negativo) o mayor que la unidad (exponente positivo).

La segunda representación resaltada, tiene como característica que la parte entera de su mantisa es nula. Si bien esta misma característica se cumple en los dos ejemplos siguientes, la expresión (2) ofrece además una característica adicional: la parte fraccionaria tiene su primer cifra posterior al punto (coma) decimal no nula, lo que significa una mantisa en el rango **1 > m > 0,1.** Este tipo de expresión, en la que se utiliza una mantisa de valor entero nulo y cuya parte fraccionaria está entre 1 y la primer potencia negativa de la base, se conoce como **forma normalizada de la mantisa** y es la que se utiliza en las representaciones punto flotante más habituales.

Debe tenerse en cuenta que en el caso de números negativos, el signo se verá reflejado sobre el signo de la mantisa. El signo del exponente, en todos los casos, indica si el número que se representa es mayor o menor que la unidad.

Ejemplo 2.10.1.- Representar en forma exponencial, normalizada, los números 2540,025; 0,0234; - 16557, que están expresados en base 10.

En cada uno de los casos, el método para convertir un número a formato punto flotante consiste en expresar el número como una mantisa menor que la unidad multiplicado por una potencia de la base tal que se vuelva a obtener el número original.

- 2540,025 = 2540,025 x 10^0 se puede multiplicar y dividir la expresión por la potencia adecuada de 10. En este caso será 10000 la potencia a utilizar:
2540,025** = 2540,025 x 10^4 x 10^{-4} = **0,2540025 x 10^4
- 0,0234** = 0,0234 x 10^0 = 0,0234 x 10^1 x 10^{-1} = **0,234 x 10^{-1}
- 16657** = -16557 x 10^{-5} x 10^5 = - **0,16657 x 10^5

La representación exponencial de los números reales permite expresar con claridad valores que pueden ser muy grandes y que, por consiguiente, requieran muchas cifras enteras para poder ser expresados en una forma habitual, o por el contrario, valores muy pequeños, en los que la cantidad de ceros luego de la coma decimal y antes de la primer cifra significativa puede ser importante.

Pero además, esta forma de representar números permite realizar operaciones aritméticas sin inconvenientes, ya que solamente habrá que respetar algunas reglas sencillas que permitan operar con números expresados en este formato:

- Para sumar o restar números expresados en punto flotante se requerirá que los números tengan el mismo exponente. En ese caso, se sumarán las mantisas. Si luego de realizada la suma o la resta, la mantisa no queda expresada en su forma normalizada, se podrá multiplicar y dividir el resultado por la potencia de la base que corresponda para lograr la normalización del número.

Ejemplo 2.10.2.: Sumar los números 0,245 x 10^{12} y 0,475 x 10^{12}.

Dado que los operandos tienen el mismo exponente, todo lo que debe hacerse es sumar las mantisas.

0,245 x 10^{12} + 0,475 x 10^{12} = 0,71 x 10^{12}

Como la mantisa se mantuvo menor que la unidad, el resultado está expresado en su forma normalizada y no debe ser modificado.

Ejemplo 2.10.3.: Sumar los números 0,3474 x 10^5 y 0,5647 x 10^7

En este caso, los números no tienen el mismo exponente. Para poder realizar la conversión deben llevarse los dos operandos a tener el mismo exponente. Esto puede realizarse modificando cualquiera de los dos operandos. Si por ejemplo se opera sobre el segundo, se tendrá:

$$0,5647 \times 10^7 = 0,5647 \times 10^5 \times 10^2 = 56,47 \times 10^5$$

Ahora sí se puede realizar la operación, la que resultará:

$$0,3474 \times 10^5 + 56,47 \times 10^5 = 56,8174 \times 10^5 = 0,568174 \times 10^7$$

Donde el último paso de la operación sirvió para volver a normalizar el resultado.

- Para multiplicar números en punto flotante se multiplican las mantisas entre sí. El exponente del resultado se obtiene sumando los exponentes de los operandos. Nuevamente se puede proceder a una normalización del número si la mantisa no hubiese quedado normalizada.

Ejemplo 2.10.4.: Multiplicar los números 0,4567 x 10^{12} y - 0,1432 x 10^{-6}

El producto de estos dos números da por resultado

$$- 0,4567 \times 0,1432 \times 10^{12-6} = - 0.06539944 \times 10^6 = - 0,6539944 \times 10^5$$

Donde nuevamente el último paso consistió en normalizar la mantisa.

- Para dividir números expresados en punto flotante se dividen las mantisas entre sí. El exponente del resultado se obtiene restando los exponentes de los operandos. Nuevamente se puede proceder a una normalización del número si la mantisa no hubiese quedado normalizada.

2.11.1.- Representación de números reales. Formatos de representación en punto flotante.

Nótese que hasta el momento, la representación de números reales en formato de mantisa y exponente se ha ejemplificado con números decimales, es decir, expresados en el sistema de numeración decimal. En es caso de las máquinas calculadoras o computadoras, la representación punto flotante se realizará cuando la magnitud a representar no permita una cómoda representación en punto fijo. En estos casos, se expresarán las magnitudes en un formato de punto flotante, es decir, mantisa, base y exponente, en el que la base a considerar será, salvo situaciones especiales, dos.

En el caso de la representación binaria de números expresados en punto flotante, se tendrá también un número expresado con una mantisa (que será positiva o negativa según lo sea el número representado), y un exponente, que será positivo o negativo según que el número a representar sea mayor o menor que la unidad. Dado que tanto la mantisa como el exponente deben indicar signos, se utilizará, para la representación, alguna de las convenciones normalmente utilizadas, y ya definidas, para números enteros. En la mayoría de los casos, se utiliza la representación correspondiente al complemento a la base. Esto es, la mantisa correspondiente a un número negativo se representará a través de su complemento a dos, y lo mismo ocurrirá con un exponente negativo (correspondiente a un número fraccionario).

Para expresar un número en punto flotante se requiere una cantidad determinada de bits para representar la mantisa, incluyendo un bit de signo, y otra determinada cantidad de bits para representar el exponente, incluyendo asimismo un bit para representar el signo. La base del número se da por sobreentendida, por lo que no hay necesidad de representarla. En una primera época, los diferentes fabricantes de computadoras utilizaban configuraciones propias, y por lo tanto, arbitrarias, para la representación de números en punto flotante. Así surgían dos inconvenientes: la incompatibilidad entre formatos correspondientes a distintas máquinas, y la consecuente dificultad de quienes debían interpretar la información, dado que se requería conocer la convención utilizada por cada uno de los fabricantes.

Un ejemplo de notación en punto flotante puede verse en la figura 2.5, en la que se representa un número binario arbitrariamente con 24 bits:

mantisa

ms	m14	m13	m12	m11	m10	m9	m8	m7	m6	m5	m4	m3	m2	m1	m0
b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9

Exponente

es	e6	e5	e4	e3	e2	e1	e0
b8	b7	b6	b5	b4	b3	b2	b1

Fig. 2.5.- Formato punto flotante, 24 bits, 8 para exponente, 16 para mantisa

Los siete bits menos significativos representan el exponente del número en convención de complemento a la base, siendo el octavo bit (b8) el signo del exponente. Asimismo, los quince bist siguientes representan la mantisa, en complemento a dos, normalizada, siendo el bit más significativo el de signo de la mantisa, y por ende, del número.

Con esta cantidad de bits, el exponente varía entre + 127 y - 127. Dado que la mantisa está normalizada, su primer dígito (m_{14}), deberá ser siempre uno, por lo que el rango de la mantisa está limitado a 0,5 .. 1. (en realidad no es 1 sino $(2^{15} - 1)/2^{15}$, lo que puede considerarse como 1 con una buena aproximación.

Por consiguiente, el rango de representación permitido en un formato de punto flotante como el que antecede va desde $\mathbf{2^{127}}$ hasta $\mathbf{2^{-128}}$. Esto puede verificarse rápidamente, dado que el límite superior es aquel en que el exponente es el mayor posible (127) y la mantisa también (1), siendo el límite inferior el que corresponde al menor exponente (-127) y a la menor mantisa (0,5, = 2^{-1}).

Ejemplo 2.11.1.- Representar en el formato punto flotante de la figura anterior el número 4532,125

El procedimiento requiere convertir el número (que está expresado en base 10) a un formato binario, del tipo $\boldsymbol{m \times 2^e}$*, en el que* ***m*** *es la mantisa (normalizada) y* ***e*** *es el exponente al que se debe elevar la base* ***2*** *para lograr la representación del número.*

La conversión a binario se hace de acuerdo con los métodos ya vistos con anterioridad en este capítulo, por lo que se obtendrá:

$4532_{/10} = 1000110110100$ $\qquad\qquad$ $0{,}125_{/10} = 0{,}001_{/2}$

La conversión del número decimal da por resultado:

$$\boldsymbol{1000110110100{,}001}$$

A los efectos de convertir el número a punto flotante, se deberá proceder a multiplicar y dividir el valor obtenido por una potencia de la base 2 que lleve la parte entera del número a cero:

$$\boldsymbol{1000110110100{,}001 = 1000110110100{,}001 \times 2^{13} / 2^{13} =}$$

$$\boldsymbol{0{,}1000110110100001 \times 2^{13}}$$

Nótese que el exponente 13 coincide con la cantidad de dígitos enteros que forman el número a convertir. La representación punto flotante, de acuerdo con el formato elegido, será la siguiente:

signo *mantisa*

0	***1***	***0***	***0***	***0***	***1***	***1***	***0***	***1***	***1***	***0***	***1***	***0***	***0***	***0***	***0***
b24	*b23*	*b22*	*b21*	*b20*	*b19*	*b18*	*b17*	*b16*	*b15*	*b14*	*b13*	*b12*	*b11*	*b10*	*b9*

signo *exponente*

0	***0***	***0***	***0***	***1***	***1***	***0***	***1***
b8	*b7*	*b6*	*b5*	*b4*	*b3*	*b2*	*b1*

Puede apreciarse que, en la representación obtenida, se ha debido omitir el último bit (el menos significativo), debido a la cantidad de bits disponibles para la representación según el formato elegido. Esta pérdida de precisión puede parecer no significativa, dada la posición ocupada por los bits descartados. No obstante, debe tenerse en cuenta que, al estar afectada la mantisa en cuestión por una potencia de la base que puede ser muy grande, esa pérdida de precisión en la conversión puede traducirse en una diferencia importante entre el valor a convertir y el valor obtenido tras la conversión.

Es por eso que en la representación de números en punto flotante, se le debe dar importancia a la cantidad de bits que forman la mantisa, aumentando dicha cantidad si fuese necesario.

Una forma de lograr, aún en forma ínfima, ese aumento, surge de plantear nuevamente el concepto de **normalización** de la mantisa. Según se ha dicho, este concepto implica que el primer dígito (bit) después de la coma decimal debe ser significativo. En el caso de representaciones binarias, el concepto de un bit significativo equivale a decir que si la mantisa de un número representado en punto flotante está normalizada, el primer bit fraccionario del número vale **uno**. Dado que este

precepto se cumple siempre, puede evitarse la representación de dicho 1, interpretándolo en forma implícita. Esto permitirá representar el número con un bit más que la capacidad asignada, entendiéndose que el primer bit siempre vale 1. De esta manera se logrará duplicar la precisión, al incorporar un bit más, en el extremo menos significativo, en la representación punto flotante. Para el número del ejemplo anterior, su representación, en el caso de considerar implícito el primer bit será:

signo mantisa

0	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9

exponente

signo

↓

0	**0**	**0**	**0**	**1**	**1**	**0**	**1**
b8	b7	b6	b5	b4	b3	b2	b1

Obsérvese que el primer bit representado es ahora cero, lo que no parece coincidir con el concepto de mantisa normalizada, pero debe tenerse en cuenta, de acuerdo con esta nueva convención, que el primer bit representado no es el primer bit del número, el que es necesariamente uno. Obsérvese además que al operar de esta forma, se incorpora al final del número el bit 1 que había sido descartado en la representación anterior por falta de capacidad para incluirlo.

Cuando se pretende representar números negativos en formato de punto flotante, siguen siendo válidas los conceptos anteriores, con algunas consideraciones adicionales. Al haberse planteado la representación en formato de complemento a la base, con bit de signo, se obtiene como efecto el representar los números negativos a través de su mantisa complementada y con el bit de signo en **uno**. En este caso, la normalización de la mantisa involucra que el primer bit de la misma deberá ser **cero** (complemento del primer uno) Esto permite plantear la idea de que, en todos los casos, el primer bit de la mantisa es complementario del bit de signo, por lo que, aún utilizándolo en forma implícita, su valor es conocido sea el número representado positivo o negativo.

2.11.2.- Variantes en la forma de representación del exponente.

Así como se han planteado distintas alternativas en la representación de las mantisas, también existen distintas formas de representar los exponentes de los números representados en punto flotante. Una primera alternativa, ya planteada, es la de representar el exponente también en su forma de complemento a la base. Este formato, si bien es válido, requiere la utilización de otro bit de signo, ofreciendo además el inconveniente de la doble representación del cero, ya tratado en apartados anteriores. Por consiguiente, se suele utilizar un formato distinto para la representación de los exponentes, que permita evitar estos inconvenientes.

La representación de exponentes en **exceso n,** siendo n un número arbitrario (que suele ser potencia de dos o un valor una unidad menor que potencia de dos), convierte a todos los exponentes en números positivos. Esto significa que el rango de representación de los exponentes irá desde 0 hasta 2^n-1, representando con ese rango tanto exponentes negativos como positivos.

Un ejemplo aclarará este planteo. Si se utilizaran 8 bits para el rango de exponentes, los mismos representan exponentes en el rango - 127 .. + 127. Si se utiliza la convención ya vista de complemento y signo, los exponentes positivos serían todos aquellos en el rango binario 00000000 01111111 (0 .. 127). En tanto que los negativos corresponderán al rango binario 10000001 .. 11111111 (- 127 .. -1).

Utilizar una representación de exponentes en **exceso 127** significará sumarle ese valor a cada uno de los exponentes. En este caso, la representación del exponente -127 será 0000 0000, en tanto que la representación del exponente +127 será 1111 1110 (el número binario 254).

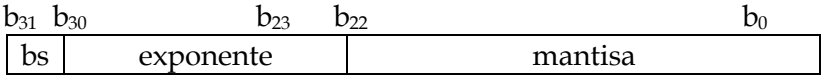
Si se hubiese utilizado una representación en **exceso 128**, el mismo rango de exponentes estará representado por los números binarios 1 .. 255 (0000 0001 .. 1111 1111). Al plantear este tipo de simbolización, desaparece el bit de signo como tal. El signo del exponente está dado por el valor numérico del mismo en relación al rango total de exponentes a representar: La primera mitad corresponde a los negativos, la segunda mitad a los positivos, el conjunto está centrado sobre el exponente cero.

Es importante tener en cuenta que, a los efectos de la correcta interpretación del número representado, debe ser conocida la convención utilizada. Si bien durante mucho tiempo, no había acuerdo entre los diferentes fabricantes de computadoras acerca del formato a utilizar para la representación de números flotantes (en capítulos posteriores quedará clara la relación entre el formato de representación de números y la forma de almacenarlos en una computadora), finalmente se logró formalizar una norma de representación, la que fue homologada por el Institute of Electrical and Electronics Engineers (IEEE).

2.11.3.- El formato de representación punto flotante IEEE 754.

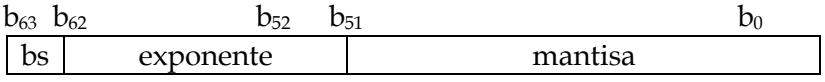
La representación formalizada como norma por el IEEE hacia fines de la década de los sesenta permitió, de alguna manera, uniformar la confusa situación reinante en las representaciones de números en formato punto flotante. Esta norma define tres formatos diferentes de punto flotante, con destino a diferentes aplicaciones.

El formato básico o de **simple precisión** utiliza 32 bits, distribuidos de acuerdo con la figura. El bit más significativo es el signo del número representado, en la convención habitual: bit de signo cero para los números positivos. Los ocho bits siguientes corresponden al exponente, representado en exceso 127. Posteriormente, los 23 bits restantes corresponden a la mantisa, que se representa en forma normalizada, con el primer bit implícito, lo que significa que en realidad la mantisa incluye 24 bits: el primero invisible (implícito) y los demás visibles.



Este formato de representación permite expresar números en el rango de 2^{-126} a 2^{+127}, lo que, en la práctica, y para hacer referencia a cantidades decimales, habla de números en el rango 10^{-38} .. 10^{+38}.

Si este rango de representación no es suficiente, la norma prevé un segundo formato, de doble precisión, que utiliza 64 bits para la representación numérica. En este caso se utiliza un bit para signo, 11 para el exponente, el que se representa en notación exceso 1023, y los otros 52 bits se usan para representar la mantisa.



Con once bits, el exponente permite representar números en el rango binario -1022 .. + 1023, lo que, llevado a potencias de 10, representará números en el rango que va desde 10^{-308} (308 ceros luego de la coma decimal y antes de la primer cifra significativa) hasta 10^{+308} (un 1 seguido de 308 ceros).

INTRODUCCIÓN A LOS SISTEMAS DIGITALES

ING. FERNANDO IGNACIO SZKLANNY

CAPÍTULO III:

CONCEPTOS FUNDAMENTALES III:

CÓDIGOS BINARIOS

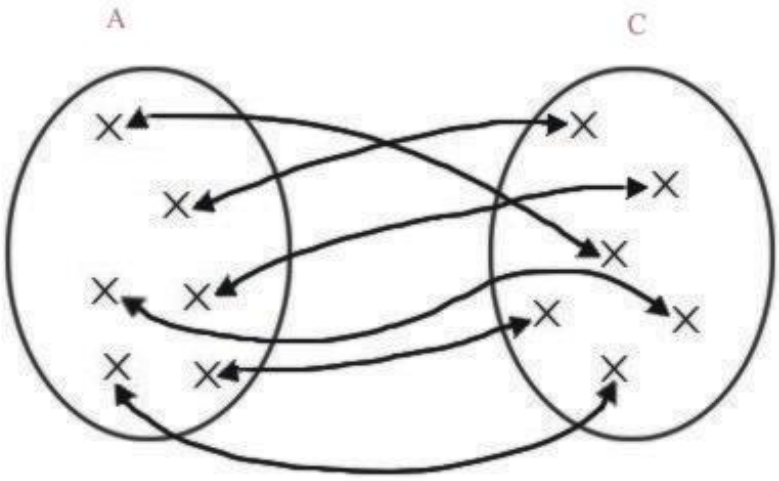
CAPÍTULO 3: CÓDIGOS BINARIOS.

3.1.- El concepto de código.

El concepto de **código** se utiliza en forma amplia y en muchas aplicaciones de uso corriente. Una definición de este concepto podría vincularlo con la identificación de letras, símbolos, números u otros elementos a través de un conjunto de símbolos especiales. De hecho, esta representación o codificación se puede observar permanentemente en distintas aplicaciones: el código de barras con que se individualizan las distintas mercaderías en un comercio, la identificación individual a través de un número único y personal (como la identificación de una cuenta bancaria o el número de un documento), etc.

Dicho de otra manera, existe un conjunto de elementos al que debe darse alguna forma de representación o identificación individual, y existe un segundo conjunto de elementos (el código en sí) que identifica a ese primer conjunto. Para que esta identificación sea factible, es necesario que cada uno de los elementos a codificar tenga su representación única y exclusiva. Para los matemáticos, este tipo de condición sería lo que se llama una **relación biunívoca**. Desde el mismo punto de vista matemático, un código podrá definirse como una relación entre dos conjuntos, uno, el de los elementos o ítems a codificar; el otro, el conjunto de los elementos utilizados como código. Que esta relación sea biunívoca implica dos condiciones: que para cada elemento del primer conjunto haya uno y solo un elemento en el segundo conjunto (un mismo elemento a codificar no puede tener dos codificaciones distintas), y que cada elemento del segundo conjunto se vincule con uno y solo un elemento del primer conjunto (no se puede asignar un mismo código a dos elementos distintos a codificar).

Estas dos condiciones se observan en la figura 3.1. Surge de la misma que los dos conjuntos utilizados en la definición matemática del código deberán tener la misma cantidad de elementos.



3.1.- Una definición matemática de un código

El concepto de código se ha utilizado y se utiliza largamente en distintas aplicaciones. En el campo de las comunicaciones, el desarrollo de la telegrafía implicó la utilización de un código basado en señales eléctricas de dos duraciones distintas, a los que se conoce como **puntos** y **rayas**. La sucesión de puntos y rayas del código Morse permitió la comunicación, a través del telégrafo, con alcance mundial, al representar las distintas letras del alfabeto con combinaciones de dos elementos. La figura 3.2 representa el conjunto de combinaciones de puntos y rayas del código Morse.

Fig. 3.2. - El hoy histórico y ya no usado código Morse

También en el campo de la electricidad, y, posteriormente, de la electrónica, se utilizaba en forma intensiva el concepto de código para reconocer los valores de los elementos pasivos de los circuitos. La utilización de un código de colores para la identificación de resistencias permite codificar los diez dígitos decimales mediante la utilización de diez colores diferentes. De acuerdo con lo que se puede observar en la figura 3.3, un valor de resistencia se identifica con un conjunto de barras de diferentes colores pintado sobre el cuerpo de la resistencia. El conjunto de barras determina el valor del elemento resistivo. Este código tiene algunas características especiales, como ser su carácter posicional, dado que no solamente el color es importante para identificar la resistencia, sino también la posición de la franja de un color determinado dentro del conjunto de franjas que definen al valor final..

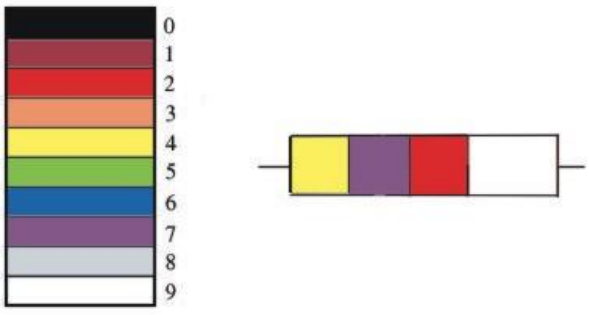


Fig. 3.3 Código de colores para la definición de resistencias eléctricas

Yendo a usos un poco más habituales, puede plantearse como un código la relación que identifica a una persona con el número de su documento de identidad (cada habitante de un país tiene un único número de identidad, no hay un mismo número asignado a dos habitantes diferentes), el código numérico o de letras y números que identifica a un automóvil (vulgarmente, su número de patente), la clave de identificación de un individuo ante la oficina recaudadora de impuestos, etc. En algunos de estos casos, los códigos planteados están formados solamente por números. En otros casos, por letras y números, por colores, o por distintos tipos de elementos que permitirán obtener una amplia gama de códigos de distintas y variadas aplicaciones.

3.2.- Códigos binarios. Conceptos fundamentales. Módulo.

Con el desarrollo de los sistemas de computación, que trabajan con magnitudes binarias, surgió la necesidad de codificar distintos tipos de datos para permitir su procesamiento en ese tipo de sistemas. Se definen como códigos binarios a aquellos códigos cuyos elementos están todos formados por solamente dos elementos: el **cero** y el **uno**. Planteado de otra forma, se puede entender un código binario como un código, utilizado para representar distintos tipos de elementos, que está basado en la utilización del sistema binario de numeración, **con el cual no debe ser confundido**. Dependiendo de la cantidad de unos y ceros que se utilicen para formar cada uno de los elementos del código se determina la cantidad de componentes del código, y, por lo tanto, la cantidad de elementos distintos que se pueden codificar. Se define como **módulo** de un código a la cantidad de elementos que dicho código permite representar.

Esta cantidad de combinaciones dependerá exclusivamente de la cantidad de bits que se han utilizado para la creación del código, o, en ciertos casos, de las reglas de generación definidas para el mismo. De acuerdo con lo visto en el capítulo anterior puede afirmarse que en un código formado por **n** bits el máximo número de combinaciones que pueden lograrse es $\mathbf{2^n}$. Se verá oportunamente que en algunas aplicaciones no se utilizan todas las combinaciones que la cantidad de bits involucrados permite obtener sino que solamente se utilizan algunas de esas combinaciones. En este caso, aún si se define alguna limitación particular del código en lo que hace a la cantidad de combinaciones a utilizar, el concepto de módulo no cambiará dado que como se ha enunciado, el modulo determina la cantidad de combinaciones a representar y no la cantidad total de combinaciones que se pueden formar con la cantidad de bits utilizados.

> ***Módulo de un código:*** *Cantidad de elementos que el código permite representar.*

Para generar un código binario de módulo dado, solamente hace falta seleccionar tantas combinaciones de ceros y unos como elementos se pretendan codificar, y asignar cada una de las combinaciones a cada uno de los elementos a codificar. De esta manera, se cumplirá la definición básica de código, y se habrá definido una codificación binaria que permita la representación de los elementos requeridos. En el ejemplo siguiente se puede observar un código arbitrario de módulo 8, que permite representar, por consiguiente, ocho elementos diferentes. Se puede observar que todas las combinaciones que forman este código tienen la misma cantidad de bits (cuatro en este caso). Obsérvese la diferencia con la representación del código Morse de la figura 3.2, en el que los distintos elementos del código tienen diferente cantidad de puntos y rayas (o de unos y ceros, si se quisiese hacer una analogía).

Se dirá que un código binario es de **longitud de palabra fija** cuando todas las combinaciones de dicho código tienen la misma cantidad de bits. En tanto que se dirá que un código binario es de **longitud de palabra variable** cuando sus diferentes combinaciones utilizan diferente cantidad de

bits en su representación. Los ejemplos que se analizarán en este capítulo serán todos códigos de longitud de palabra fija.

Ejemplo 3.1.
Un código cualquiera de módulo 8

$a \rightarrow 0011$
$b \rightarrow 0101$
$c \rightarrow 0110$
$d \rightarrow 1010$
$e \rightarrow 1001$
$f \rightarrow 1100$
$g \rightarrow 0000$
$h \rightarrow 1111$

De acuerdo con lo dicho en el párrafo anterior, bastará con elegir una cantidad de combinaciones binarias diferentes para permitir la obtención de un código binario de módulo dado. En este aspecto, una posibilidad es la de seleccionar las **n** primeras combinaciones del sistema binario, en orden correlativo, para obtener un código de módulo **n**.

La secuencia siguiente permitirá representar entonces un código binario de módulo 16. Obsérvese que de acuerdo con lo especificado, se ha formado un código binario de cuatro bits, ya que las combinaciones de 4 bits permiten obtener 16 elementos.

0	0000	4	0100	8	1000	12	1100
1	0001	5	0101	9	1001	13	1101
2	0010	6	0110	10	1010	14	1110
3	0011	7	0111	11	1011	15	1111

La numeración asignada a los diferentes elementos que forman el código no corresponde solamente al orden de los mismos dentro de la recta numérica. Puede ser también la asignación de los dieciséis elementos del código binario a los 16 elementos que se necesita codificar. Si lo que se desea codificar son números, el anterior es un ejemplo de un código binario para simbolizar los dieciséis primeros números de la secuencia numérica. Debe notarse que las correspondientes representaciones cumplen con el carácter posicional del sistema binario, dado que cada uno de los elementos a codificar está representado por su correspondiente valor binario.

3.3.- Códigos decimales.

Tal como ha quedado dicho en el capítulo anterior, el ser humano es, a todos los efectos numéricos, decimal por naturaleza. La costumbre de trabajar con los diez dígitos decimales es absoluta y única, por lo que no puede pretenderse aquí que se produzca un cambio en esa costumbre. No obstante, y dado que la necesidad de operar en sistemas binarios requiere de métodos de representación que puedan ser utilizados por computadoras y otros sistemas de procesamiento de datos, se hace imprescindible alguna forma de representación o codificación de esos números que permita, por un lado, tener una representación única para cada dígito decimal y, por otro lado, permitir que los sistemas de cálculo utilicen esas mismas representaciones para realizar las operaciones aritméticas necesarias. El lector podrá preguntarse si no basta con utilizar directamente el sistema de numeración binario para resolver el problema, ya que, de acuerdo con lo visto, este sistema de numeración permite representar cantidades y operar con ellas sin inconvenientes. La respuesta es sencilla: en muchos casos (calculadoras, por ejemplo), no resulta

cómodo realizar las conversiones hacia y desde el sistema binario. Debe recordarse que en estos casos, los operandos se ingresan dígito por dígito, por lo que sería mucho más cómoda una representación que permita seguir el manejo de la información dígito por dígito hasta la finalización del procesamiento.

Los **códigos decimales** (también llamado a veces **códigos numéricos**) son códigos utilizados para representar cantidades numéricas. Su utilización se limita a la representación de los diez dígitos decimales. Son, por lo tanto, códigos binarios de **módulo diez**. Por su designación en lengua inglesa se los suele conocer como **código BCD** ***(Binary Coded Decimal***).

Su utilización es muy sencilla, ya que solo consiste en representar cada uno de los dígitos decimales por su correspondiente codificación binaria. Así, y para ejemplificar utilizando el código anterior, se podrá representar el número 4572 mediante la representación individual de cada uno de sus dígitos:

4572 = 0100 0101 0111 0010
4 5 7 2

Es importante resaltar nuevamente que la codificación binaria de una cantidad no tiene relación alguna con su representación en el sistema binario. Esta afirmación es válida cualquiera sea el código binario a utilizar para dicha representación.

3.3.1.- Códigos pesados.

La forma más simple de obtener un código BCD consiste en volver al sistema binario, y adoptar como representación de los diez dígitos decimales las diez primeras combinaciones del mismo:

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

Este código sigue cumpliendo con el carácter posicional del sistema binario dado que proviene de aquel. Por consiguiente, no es siquiera necesario recordar la relación individual entre cada elemento del código y el elemento codificado, dado que la misma se puede obtener directamente a través de la suma de los valores relativos de los unos que aparecen en cada una de las columnas. Planteado de otra forma, se puede decir que en el ejemplo planteado cada una de las columnas tiene un valor relativo, al que se designará como **peso** de la columna, y que será el valor que adopte siempre un uno ubicado en esa columna.. En el caso en que las columnas de un código tengan peso, el código en cuestión se considerará como un **código pesado**.

El peso de cada columna en un **código pesado** es el valor numérico que adopta un uno ubicado en una columna determinada del código. El valor numérico del elemento representado se obtendrá como suma de los pesos de las columnas en las que haya un uno. Por corresponderse directamente con el sistema de numeración binario, las columnas del código del ejemplo anterior tienen pesos (de izquierda a derecha) de 8, 4, 2 y 1, respectivamente. En la práctica, los códigos pesados se designan con los valores relativos, o pesos, que los definen, por lo que el ejemplo anterior se conoce como **código 8421**.

Pueden plantearse diferentes códigos decimales pesados, si se definen apropiadamente los pesos de las columnas que forman al código. El concepto básico es que pueden definirse códigos pesados mediante selección de los pesos de las distintas columnas siempre y cuando dichos pesos permitan la representación de los diez dígitos decimales. Esto es, la suma de algunos o todos los pesos de las columnas binarias del código deberá permitir obtener todos los dígitos del cero al nueve. A continuación se ejemplifican algunos códigos que cumplen con estas características.

	6321		**643-2**	**6311**		**5221**	
0	0000		0000	0000		0000	
1	0001		0011	0001	(0010)	0001	
2	0010		0101	0011		0010	(0100)
3	0100	(0011)	0010	0100		0011	(0101)
4	0101		0100	0101	(0110)	0110	
5	0110		0111	0111		1000	
6	1000	(0111)	1000	1000		1001	
7	1001		0110	1001	(1010)	1010	(1100)
8	1010		1101	1011		1011	(1101)
9	1100	(1011)	1010	1100		1110	

De los ejemplos anteriores puede observarse, primeramente, que de acuerdo con las combinaciones de pesos seleccionados para formar el código, los dígitos decimales representados adoptan representaciones totalmente diferentes entre sí y, además, diferentes al valor binario real del dígito representado. Por otra parte puede verse que en algunos casos, un mismo dígito puede representarse de más de una forma, lo que, en principio, aparenta ser una violación a la regla establecida (una única codificación para cada elemento a codificar). Lo que debe quedar claro en esta situación es que una vez elegida una determinada combinación de ceros y unos para representar un dígito, esa será la única codificación válida y no se deberá tener en consideración ninguna otra suma de pesos que represente ese mismo dígito.

La situación aludida surge en casi todos los códigos ejemplificados anteriormente. Incluso, dos de los códigos utilizados tienen dos columnas con el mismo peso. Esto permite afirmar que en varios casos existen dos combinaciones distintas que permiten representar el mismo dígito, y que simplemente se obtienen desplazando un uno a la otra columna de igual peso, si estuviera en cero.

3.3.2.- Códigos sin peso.

No todos los códigos son pesados. En algunos casos, en que la asignación se basa en otros criterios (aún el de la total arbitrariedad en la asignación), un uno ubicado en una dada columna no tiene un valor relacionado con dicha columna. En este caso sí es necesario conocer la relación individual entre cada uno de los elementos a representar y cada uno de los elementos del código. En este caso se hablará de **códigos no pesados** o códigos **sin peso**.

Este tipo de códigos puede crearse para distintas aplicaciones o por diferentes causas, algunas de las cuales se detallarán en los próximos apartados. En todo caso, se trata de códigos particulares, cuya utilización se definirá en función de la necesidad correspondiente. El siguiente ejemplo muestra un código BCD no pesado cuya concepción fue totalmente arbitraria.

0	0000	5	1101
1	1111	6	0100
2	0001	7	1011
3	1110	8	1000
4	0010	9	0111

El número representado anteriormente, 4572, planteado sobre la base de este último código, se verá como

4572 = 0010 1101 1011 0001

3.3.3.- Algunas características de los códigos numéricos. Códigos progresivos. Códigos cerrados. Ejemplos: Código Johnson. Código Gray.

La selección del código más apropiado para una aplicación determinada puede depender de distintos factores. En muchos casos la utilización del código 8421 o de algún otro código pesado es suficiente para el objetivo que se busca. En otros casos se suelen requerir códigos que presenten alguna propiedad especial adecuada a la necesidad de codificación. Las distintas características que pueden presentar los códigos binarios se resumen a continuación. Al mismo tiempo, se intentará ejemplificar la utilización de estas propiedades.

El código del ejemplo siguiente ofrece una característica singular. Al avanzar a lo largo del código, las combinaciones consecutivas se diferencian entre sí por un solo bit. Se define como **código progresivo** a todo código en el cual ***cada combinación difiere de la que le sigue en un solo bit***. La utilización de este tipo de códigos y algún método sencillo de generación de los mismos serán temas a tratar en capítulos posteriores.

0	0000
1	0001
2	0011
3	0010
4	0110
5	1110
6	1010
7	1011
8	1001
9	1000

Surge de alguna manera una condición necesaria para poder definir un código como progresivo: el concepto de código ordenado. Cada elemento del código debe tener una relación de orden con el elemento anterior y con el posterior.

El código del ejemplo cumple con la condición mencionada para un código progresivo: al pasar de una combinación a la siguiente sólo se modifica un bit, y esta situación se repite en todos los casos. No solamente sucede al avanzar desde una combinación cualquiera del código hacia la siguiente. En este caso, lo mismo sucede al pasar de la última combinación del código (la que corresponde al dígito 9) a la primera (la que corresponde al cero). En caso de que el código presente una diferencia de un bit también entre sus combinaciones extremas se lo denomina **código cerrado**.

Si un código, como el del ejemplo, es progresivo y cerrado, el mismo puede verse como un conjunto de combinaciones entre todas las cuales se cumple el principio de diferir con la que le sigue en un solo bit. En este caso, podrá obtenerse otro código a partir del anterior eligiendo cualquier otra combinación como combinación inicial. El código que así se obtenga seguirá siendo cerrado y progresivo, como se ve en el ejemplo siguiente.

0	1011	3	0000	6	0010	9	1010
1	1001	4	0001	7	0110		

2	1000	5	0011	8	1110

Debe aclararse que las condiciones de ser progresivo y cerrado no necesariamente deben coexistir en un código. Sin embargo, en la mayoría de los códigos, si se determina que el mismo será progresivo se suele buscar que además sea cerrado.

Un primer ejemplo de este tipo de código es el que se muestra en la tabla siguiente. El código representado se conoce como **código Johnson**, y si bien no es, si se analiza con todo rigor, un código decimal, podrá utilizárselo como tal en determinadas condiciones. En efecto, un código Johnson es un código sin peso que puede estar formado por **n** bits, para entregar un total de **2.n** combinaciones diferentes. Esto permite considerar como caso particular de un código decimal al código Johnson de módulo 10. Se trata de un código que cumple con las condiciones de ser progresivo y cerrado, y, además, fácil de generar.

0	00000
1	00001
2	00011
3	00111
4	01111
5	11111
6	11110
7	11100
8	11000
9	10000

Obsérvese que el código ha sido obtenido a partir de una combinación de ceros, los que se han ido reemplazando por unos, a razón de un uno en cada combinación, hasta llegar a un elemento formado por unos en su totalidad. En este momento, se repite la secuencia pero ahora reemplazando los unos por ceros hasta volver a la situación inicial. Cabe acotar que el reemplazo de ceros por unos y posteriormente de unos por ceros, que en el ejemplo se realizó desde la derecha, puede realizarse también de izquierda a derecha, sin que esto modifique el código así creado.

Otro código que reúne algunas de estas características se conoce como **código Gray**. Al igual que en el caso del código Johnson, no se trata rigurosamente de un código decimal. En este caso lo que se analiza es un código de módulo $\mathbf{2^n}$, siendo **n** la cantidad de bits que lo forman. De un código Gray de cuatro bits, módulo 16, podrá alcanzarse, mediante algunos procedimientos sencillos, un código decimal que siga teniendo las características del código del cual proviene. El código Gray es un código progresivo y cerrado. Pero además tiene una característica que le es propia. Un código Gray formado por **n** bits puede ser generado, en forma recurrente, a partir de un código formado por **n-1** bits. Este planteo permite ir obteniendo los códigos de Gray de módulos mayores a partir de los de módulo menor. El ejemplo siguiente permitirá llegar al código Gray de módulo 16 a partir de la nada. En efecto, si se plantea el significado de un bit a través de la idea de código, surge claramente que un bit permite codificar (o representar) dos eventos:

0
1

Puede decirse que el anterior es un código Gray de un bit: módulo 2, progresivo y cerrado. El método para obtener el código de 2 bits, módulo 4, consiste en **reflejar** el código anterior, entendiéndose por reflejar el proceso de agregar a las combinaciones anteriores las mismas

combinaciones pero rebatidas como si se viesen en un espejo. Para completar, y a los efectos de lograr las cuatro combinaciones diferentes, se agrega un segundo bit, el que valdrá **cero** de un lado del eje de simetría y **uno** del otro lado del mismo. El código de módulo 4 se muestra en la tabla siguiente.

00
01
11
10

De la misma manera se logrará el código de tres bits y módulo 8:

000
001
011
010
110
111
101
100

El paso al código de módulo 16 es análogo: se repiten los ocho elementos anteriores, se reflejan con respecto al eje de simetría formado luego del octavo elemento del código y se agrega la cuarta variable.

0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

De la misma manera se procederá hasta llegar al módulo que se requiera. Esta propiedad del código Gray hace que se lo defina como un **código reflejado**, lo que significa que a cada combinación del código le corresponde otra, simétrica de la misma con respecto del eje central, de la cual difiere en un solo bit. Esta característica del código es la que permite obtener códigos de módulo distinto a 2^n que sigan respetando la estructura del Gray: progresivo, cerrado y reflejado. En efecto, si se requiere reducir el módulo del código Gray (por ejemplo para obtener un código decimal), la forma de lograrlo es por medio de la eliminación de combinaciones. Si por cada combinación eliminada en el código se elimina también su simétrica, se obtendrá un código de módulo menor, que seguirá siendo reflejado. Por ejemplo, para obtener un código Gray de módulo 14 bastará con eliminar la primer y la última combinación del código anterior. Otra alternativa

sería eliminar las dos combinaciones centrales, o cualquier otro par de combinaciones simétricas. El ejemplo siguiente muestra dos posibilidades de obtener un código Gray de módulo 10.

En el primer caso se han suprimido las seis combinaciones centrales, tres a cada lado del eje de simetría. En el segundo caso se han eliminado las tres combinaciones iniciales y las tres combinaciones finales, simétricas con aquellas.

0	0000
1	0001
2	0011
3	0010
4	0110
	~~0111~~
	~~0101~~
	~~0100~~
	~~1100~~
	~~1101~~
	~~1111~~
5	1110
6	1010
7	1011
8	1001
9	1000

	~~0000~~
	~~0001~~
	~~0011~~
0	0010
1	0110
2	0111
3	0101
4	0100
5	1100
6	1101
7	1111
8	1110
9	1010
	~~1011~~
	~~1001~~
	~~1000~~

En ambos casos se han obtenido códigos de módulo 10, aptos, por consiguiente, para representar los diez dígitos decimales. Estos dos códigos decimales se observan, sin la inclusión de los elementos eliminados, en las tablas siguientes:

0	0000
1	0001
2	0011
3	0010
4	0110
5	1110
6	1010
7	1011
8	1001
9	1000

0	0010
1	0110
2	0111
3	0101
4	0100
5	1100
6	1101
7	1111
8	1110
9	1010

3.3.4.- Códigos autocomplementados.

La codificación de los dígitos decimales mediante el sistema binario no solamente se utiliza para una representación de cantidades numéricas. Existen códigos numéricos que, además, permiten realizar operaciones aritméticas, tal como si se estuviese operando en el sistema decimal. No obstante, y dado que se está utilizando un sistema numérico binario requiriéndose de resultados

decimales, estos resultados no se obtienen directamente sino que en la generalidad de los casos deben realizarse algunas correcciones que permitan obtener el resultado correcto.

No todos los códigos BCD permiten realizar operaciones aritméticas. Solamente algunos de ellos, en los cuales pueda encontrarse una regla coherente para corregir los resultados intermedios obtenidos, permitirán su utilización con tal objetivo. Entre estos pocos códigos adecuados para realizar operaciones aritméticas se encuentran dos códigos pesados, el 8421 y el 2421 (conocido como **código Aiken**) y el código no pesado conocido como **exceso 3.**

En estos dos últimos casos se trata de códigos que tienen una propiedad particular: la de ser **autocomplementados**. En el caso del código Aiken se trata de un código pesado, de pesos **2421**, en el que se aprovechan las dos columnas de igual peso, y la columna de peso 4, para seleccionar adecuadamente las representaciones de los diez dígitos. A través de esta forma particular de selección, se podrá observar que a partir de una combinación cualquiera del código se podrá obtener la representación de su complemento a 9 de la misma manera en que se calcula el complemento a la base menos uno en el sistema binario. Esto significa que, en este código en particular, el cálculo del complemento a 9 (la base menos uno, ya que el código representa dígitos decimales) se realiza como si se estuviese operando en el sistema binario de numeración, invirtiendo unos por ceros y ceros por unos.

	2421		**Exceso 3**
0	0000	0	0011
1	0001	1	0100
2	0010	2	0101
3	0011	3	0110
4	0100	4	0111
5	1011	5	1000
6	1100	6	1001
7	1101	7	1010
8	1110	8	1011
9	1111	9	1100

Otro código que también presenta la característica de ser autocomplementado es el código que se conoce como **Exceso 3** (XS3). Este código, ahora sin peso, se obtiene a partir del 8421 sumando tres unidades a la representación binaria de cada uno de los dígitos representados. Esto significa que el cero (0) se representa con la combinación de ceros y unos que en binario puro (o en 8421) representaría al 3, el 1 se representa con la combinación binaria que correspondería al 4 y así siguiendo hasta llegar al 9, cuya representación es la combinación que en binario natural representa al 12. No debe entenderse que el código representa números entre el 3 y el 12, sino que utiliza las combinaciones binarias correspondientes al rango 3 - 12 para representar los 10 dígitos decimales.

3.4.- Operaciones aritméticas utilizando códigos BCD.

La utilización de códigos binarios para representar los dígitos decimales no solamente implica la posibilidad de dicha representación sino también, en muchos casos, la necesidad de resolver operaciones aritméticas con operandos decimales representados en algún código binario. No todos los códigos son aptos para realizar operaciones aritméticas con cantidades codificadas: es necesario encontrar aquellos códigos en los que las operaciones se resuelvan en una forma sistemática y sencilla. Dos de los códigos habitualmente utilizados para la realización de operaciones son el código 8421 y el código Exceso 3. En ambos casos, las operaciones de suma se realizan dígito a

dígito como si se estuviese operando en el sistema decimal. Se verá inmediatamente que esta forma de operar es sencilla pero requiere de una corrección que, en ambos casos, se puede deducir en forma sencilla y sistemática para obtener el resultado definitivo de la operación.

La necesidad de introducir correcciones al realizar una operación de suma surge del planteo siguiente: un código BCD es un código en el que cada uno de los diez dígitos decimales se representa con una combinación de cuatro dígitos binarios. Al realizar la suma de dos dígitos decimales (números del 0 al 9) se puede obtener un resultado que va desde 0 hasta 18. La representación de los resultados mayores que nueve requieren de dos dígitos decimales, representados cada uno de ellos por cuatro bits. No obstante una combinación de cuatro bits, en el sistema binario de numeración, permite representar las cantidades que van desde 0 hasta 15. Esta diferencia implica la existencia de seis combinaciones binarias de cuatro bits (las que en el sistema binario representarían a los números 10 al 15) que no existen en el código BCD. Por consiguiente, si el resultado de una suma cae dentro de ese rango del 10 en adelante, se deben convertir los resultados obtenidos para que los mismos aparezcan expresados en BCD y no en binario natural.

3.4.1.- Operaciones aritméticas utilizando el código 8421.

En las dos tablas siguientes se observa la relación entre los primeros 18 números decimales expresados en el sistema binario natural (columna a) y en el código 8421 (columna b).

	Sistema binario	**Código 8421**
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	**1010**	0001 0000
11	**1011**	0001 0001
12	**1100**	0001 0010
13	**1101**	0001 0011
14	**1110**	0001 0100
15	**1111**	0001 0101
16	1 0000	0001 0110
17	1 0001	0001 0111
18	1 0010	0001 1000

De la tabla surge la identidad entre las dos representaciones de los números 0 al 9, tanto en el sistema binario natural como en el código 8421. Surge además que a partir del número decimal 10, las representaciones dejan de ser iguales. En efecto, entre la representación codificada del número decimal 9 (1001) y la del número siguiente (0001 0000) aparecen, en el sistema binario, seis combinaciones que no pertenecen al código 8421. Estas seis combinaciones, resaltadas en la tabla, deberán ser evitadas como resultado de una operación de suma de dos cantidades codificadas en 8421. Queda claro que no se puede evitar que el resultado de una suma de dos dígitos decimales sea un número entre 10 y 18. Lo que habrá que hacer es expresar ese resultado correctamente, lo que implicará saltear las seis combinaciones para obtener la expresión correcta del resultado. Esto

significa que en cualquier suma de dos dígitos expresados en BCD en la que se obtenga un resultado mayor que 9, deberá realizarse una corrección consistente en sumarle 6 al resultado obtenido. Esto, en la práctica, significará saltear las seis combinaciones inexistentes del código 8421 y, en definitiva, obtener el resultado correcto, codificado en forma correcta.

Por lo tanto, al realizar la suma de dos números cualesquiera, representados según un código 8421, la operación de suma se realizará columna por columna, tal como si se estuviese operando en el sistema decimal. Una vez obtenidos los resultados de cada columna, se procederá a analizar la pertenencia de cada uno de los valores obtenidos al código en cuestión, corrigiendo los valores que no correspondan.

Ejemplo 3.2:

Sumar los números 497 + 662, los que se codifican en 8421.

Se procederá a la representación de cada uno de los dos operandos, dígito por dígito:

```
   11
  497    =    0100  1001  0111
  662    =    0110  0110  0010
 1159         1010  1111  1001
```

Nótese que los valores obtenidos en las columnas de las decenas y de las centenas representan, en el sistema binario, los dígitos 15 y 10, respectivamente. Nótese también que, en la operación realizada en el sistema decimal, esas dos columnas producen arrastre hacia la columna de mayor peso (indicados los arrastres en la parte superior de cada una de las columnas. Para obtener los resultados correctos en código 8421, deberá "provocarse" la aparición de dichos arrastres, lo que significará sumar un 6 en cada una de las dos columnas a corregir, sin afectar las columnas correctas. (la de las unidades en este caso) La operación se completa llevando los arrastres a las columnas correspondientes, para finalmente obtener el resultado correcto.

```
          1010   1111   1001
+         0110   0110   0000
        1 0000 1 0101   1001
+           1
0001     0001   0101   1001    = 1159
```

3.4.2.- Operaciones aritméticas utilizando el código Exceso 3.

No solamente el código 8421 permite realizar, correcciones mediante, operaciones en el sistema decimal utilizando representación binaria. También el código Exceso 3 permite la realización de dichas operaciones, con un criterio similar al anterior: la corrección de los resultados obtenidos a través de la suma o la resta de un determinado valor binario que permita obtener el resultado correcto. Este código no sólo ofrece la ventaja de ser autocomplementado, como ya se ha dicho, sino que, como se verá a continuación, exige corrección en todos los casos, a diferencia del código 8421.

Al estar los valores codificados representados en binario puro más tres, cuando se realice la suma de dos operandos representados en este código, el resultado obtenido se obtendrá en binario puro más seis. Este resultado (binario más seis) podrá interpretarse de dos formas diferentes:

- Si el resultado obtenido en la suma es menor o igual que nueve, el resultado correcto, expresado en exceso 3, se obtendrá restando tres al valor obtenido en la suma.

- Si en cambio el resultado de la suma es mayor que nueve, la expresión resultante, por efecto de los seis elementos de cuatro bits que no forman parte del código, estará representada en binario natural, por lo que, para volverlo a exceso 3, se requerirá la suma del valor tres al resultado obtenido.

En definitiva, cualquiera sea el valor obtenido en la suma, deberá realizarse la correspondiente corrección, sumando o restando tres unidades en cada una de las columnas según que el resultado obtenido en dicha columna sea mayor que nueve o no.En los capítulos correspondientes, se procederá a la implementación de los circuitos que permitan realizar las operaciones aquí detalladas, y que se ejemplifican a continuación.

Ejemplo 3.3:

Realizar la suma del apartado anterior, expresando los operandos en exceso 3.

	497	=	*0111*	*1100*	*1010*
+	*662*	=	*1001*	*1001*	*0101*
	1 159	=	*10000*	*10101*	*1111*

Nótese que en la columna de las unidades, el resultado no dio mayor que nueve, lo que se puede detectar a través de la inexistencia de arrastre (el resultado es de cuatro bits). En este caso, el resultado obtenido resultó estar expresado en exceso 6, por lo que se lo puede volver a exceso 3 restando el valor tres en dicha columna.

En las otras dos columnas, los resultados dieron cinco bits, excediendo la representación del código. Más aún, si se eliminaran en cada una de las columnas los bits más significativos de los respectivos resultados, los que deben ser considerados como arrastres hacia las columnas siguientes,. se obtendría adecuadamente en la columna de las decenas el valor 5, pero representado en binario natural. Corresponde entonces realizar las correcciones en las columnas de decenas y centenas, sumando tres a los valores obtenidos:

	497	=	*0111*	*1100*	*1010*
+	*662*	=	*1001*	*1001*	*0101*
	1 159	=	*1 0000*	*10101*	*1111*
			+ 0011	*0011*	*+ 0011*	*- 0011*
			0100	*0011*	*1000*	*1100*
			+		*1*	
			0100	*0100*	*1000*	*1100*

1 1 5 9

Nótese que en la operación realizada se produjeron dos arrastres, una desde la columna de las decenas a la de las centenas, y otro desde la columna de las centenas a la de las unidades de mil (inexistente en los operandos). En ambos casos esos arrastres se trasladaron a las columnas siguientes y, en el caso de las unidades de mil, se procedió también a corregir, mediante la suma de tres, para poder expresar el número íntegramente en el código utilizado.

3.5.- Códigos alfanuméricos.

Tal como ya se ha dicho, no solamente se utilizan codificaciones binarias para la representación de números. La necesidad de almacenar y trasmitir otro tipo de información, incluyendo textos, gráficos y otros, hace prácticamente habitual la necesidad de convertir a formato binario, entre otros, textos formados por letras, símbolos de puntuación y todo tipo de caracteres habituales en la expresión escrita. En este sentido, la necesidad de codificar esa cantidad de caracteres requerirá de códigos con mayor número de combinaciones, y, por ende, con mayor cantidad de bits en cada una de las mismas.

Un primer planteo, muy simplificado, permitiría pensar en un código alfanumérico como un código que permite representar un alfabeto completo. Si se toma en cuenta que dicho alfabeto es de 26 letras en el idioma inglés y de 27 en el español, surge claramente que la necesidad mínima es de cinco bits para generar dicho código, dando un máximo de 32 combinaciones.

Resulta obvio que, cuando se trata de trasmitir texto, no es suficiente con poder codificar las letras del alfabeto, dado que el texto requiere además el uso de símbolos de puntuación y, eventualmente, números, lo que permite deducir que cinco bits son pocos. De hecho, las actuales normas de codificación alfabética incluyen siete, ocho y hasta diez y seis bits.

No obstante, existieron aplicaciones que utilizaron códigos pretendidamente de cinco bits, como era el caso de las comunicaciones vía teleimpresoras (teletipo) cuyo sistema de comunicación se basaba en la lectura y generación de cinta de papel perforada, continua, en la cual la información aparecía codificada con cinco bits. El secreto para la utilización de dicho código era la existencia, en el teclado de las teleimpresoras, de dos teclas ("letras" y "números"), que tenían asignada una codificación particular y cuya función era la de definir una de dos interpretaciones para cada uno de los elementos del código. De acuerdo con esto, todos los códigos que aparecen con posterioridad al código de "letras" corresponden al alfabeto definido, en tanto que esos mismos códigos, si aparecen con posterioridad al código de "símbolos" representan números o símbolos de puntuación.

3.5.1.- Ejemplos de utilización. Código ASCII. Código EBCDIC.

El código ASCII (American Standard Code for Information Interchange) es un código normalizado de siete bits, cuyas 128 combinaciones permiten generar un código que represente dos alfabetos (mayúsculas y minúsculas), los diez dígitos decimales, un conjunto de símbolos de puntuación y caracteres no imprimibles o de control. El mismo se ha representado en la figura 3.4.

Este código, en su versión de siete bits, considera básicamente los símbolos alfabéticos del idioma inglés, lo que resulta inconveniente para aquellos otros idiomas (como el español, el francés o el portugués) requieren otros símbolos literales. Para resolver las necesidades de esos otros idiomas, el código ASCII extendido representa 256 combinaciones para lo cual utiliza ocho bits en lugar de siete. Esta cantidad de bits es coherente con la estructura habitual de datos dentro de un sistema de cómputos, el que almacena o procesa su información en conjuntos de ocho bits o múltiplos de dicha cantidad.

Cabe aquí aclarar que existen distintas variantes del código ASCII extendido, las que dependen de las necesidades particulares de los usuarios. De hecho, la figura 3.5 presenta la representación de los 128 caracteres agregados al código ASCII cuando se lo representa con ocho bits en lugar de siete, generados por la computadora en la que se ha escrito este texto.

En computadores de mayor envergadura, como los llamados *mainframes,* suelen utilizarse otros códigos alfanuméricos para el mismo fin: representar letras, números y símbolos de puntuación, en definitiva, texto imprimible. Uno de los códigos utilizados habitualmente se conoce como EBCDIC, código de 8 bits comúnmente utilizado como alternativa al código ASCII. Debe tenerse en cuenta aquí que la utilización de códigos diferentes para la representación de la misma información convierte a las máquinas que los utilizan en incompatibles desde el punto de vista de la trasferencia de datos en forma directa. Esto es, para poder trasferir datos representados en uno de los códigos mencionados hacia una máquina que utiliza otro, deberá previamente realizarse un proceso de conversión de dichos códigos.

En los últimos años, y acompañando el avance de las necesidades tecnológicas, se desarrolló un código binario, conocido como Unicode, el que, al utilizar mayor cantidad de bits por palabra, permite, de acuerdo con sus objetivos, representar todo aquel elemento de un lenguaje que requiera representación. Esto incluye letras, números, ideogramas, y hasta los pequeños gráficos conocidos como "emojis" que se utilizan a diario en comunicaciones informales y formales. La versión actualizada de este código utiliza palabras de 32 bits para llegar (hoy, mañana no se sabe) a más de 136,000 combinaciones.

Claramente, sería ilógico tratar de reproducir este código en las páginas de este texto, por lo que se invita a los interesados en acceder al sitio correspondiente a la organización responsable del código: https://www.unicode.org.

Dec	Hex	Oct	Car.	Dec	Hex	Oct	Car	Dec	Hex	Oct	Car	Dec	Hex	Oct	Car
0	00	000	NUL (null)	32	20	040	SPACE	64	40	100	@	96	60	140	`
1	01	001	SOH (start of heading)	33	21	041	!	65	41	101	A	97	61	141	a
2	02	002	STX (start of text)	34	22	042	"	66	42	102	B	98	62	142	b
3	03	003	ETX (end of text)	35	23	043	#	67	43	103	C	99	63	143	c
4	04	004	EOT (end of transmission)	36	24	044	$	68	44	104	D	100	64	144	d
5	05	005	ENQ (enquiry)	37	25	045	%	69	45	105	E	101	65	145	e
6	06	006	ACK (acknowledge)	38	26	046	&	70	46	106	F	102	66	146	f
7	07	007	BEL (bell)	39	27	047	'	71	47	107	G	103	67	147	g
8	08	010	BS (backspace)	40	28	050	(	72	48	110	H	104	68	150	h
9	09	011	TAB (horizontal tab)	41	29	051	)	73	49	111	I	105	69	151	i
10	0A	012	LF (NL line feed)	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	0B	013	VT (vertical tab)	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	0C	014	FF (NP form feed)	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	0D	015	CR (carriage return)	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	0E	016	SO (shift out)	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	0F	017	SI (shift in)	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE (data link escape)	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1 (device control 1)	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2 (device control 2)	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3 (device control 3)	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4 (device control 4)	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN (synchronous idle)	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB (end of trans. block)	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN (cancel)	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM (end of medium)	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB (substitute)	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC (escape)	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS (file separator)	60	3C	074	<	92	5C	134	\	124	7C	174	|
29	1D	035	GS (group separator)	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS (record separator)	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US (unit separator)	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

Fig. 3.4: El código ASCII de siete bits

Dec	Hex	Oct	Car	Dec	Hex	Oct	Car	Dec	Hex	Oct	Car	Dec	Hex	Oct	Car
128	80	200	Ç	160	A0	240	á	192	C0	300	+	224	E0	340	Ó
129	81	201	ü	161	A1	241	í	193	C1	301	-	225	E1	341	ß
130	82	202	é	162	A2	242	ó	194	C2	302	-	226	E2	342	Ô
131	83	203	â	163	A3	243	ú	195	C3	303	+	227	E3	343	Ò
132	84	204	ä	164	A4	244	ñ	196	C4	304	-	228	E4	344	õ
133	85	205	à	165	A5	245	Ñ	197	C5	305	+	229	E5	345	Đ
134	86	206	å	166	A6	246	ª	198	C6	306	ã	230	E6	346	µ
135	87	207	ç	167	A7	247	º	199	C7	307	Ã	231	E7	347	þ
136	88	210	ê	168	A8	250	¿	200	C8	310	+	232	E8	350	Þ
137	89	211	ë	169	A9	251	¹	201	C9	311	+	233	E9	351	Ú
138	8A	212	è	170	AA	252	¬	202	CA	312	-	234	EA	352	Û
139	8B	213	ï	171	AB	253	½	203	CB	313	-	235	EB	353	Ù
140	8C	214	î	172	AC	254	¼	204	CC	314	¦	236	EC	354	ý
141	8D	215	ì	173	AD	255	¡	205	CD	315	-	237	ED	355	Ý
142	8E	216	Ä	174	AE	256	«	206	CE	316	+	238	EE	356	¯
143	8F	217	Å	175	AF	257	»	207	CF	317	¤	239	EF	357	´
141	90	220	ì	176	B0	260	_	208	D0	320	ð	240	F0	360	-
145	91	221	æ	177	B1	261	_	209	D1	321	Ð	241	F1	361	±
146	92	222	Æ	178	B2	262	_	210	D2	322	Ê	242	F2	362	_
147	93	223	ô	179	B3	263	¦	211	D3	323	Ë	243	F3	363	¾
148	94	224	ö	180	B4	264	¦	212	D4	324	È	244	F4	364	¶
149	95	225	ò	181	B5	265	Á	213	D5	325	i	245	F5	365	§
150	96	226	û	182	B6	266	Â	214	D6	326	Í	246	F6	366	÷
151	97	227	ù	183	B7	267	À	215	D7	327	Î	247	F7	367	¸
152	98	230	ÿ	184	B8	270	©	216	D8	330	Ï	248	F8	370	°
153	99	231	Ö	185	B9	271	¦	217	D9	331	+	249	F9	371	¨
154	9A	232	Ü	186	BA	272	¦	218	DA	332	+	250	FA	372	·
155	9B	233	ø	187	BB	273	+	219	DB	333	_	251	FB	373	¹
156	9C	234	£	188	BC	274	+	220	DC	334	_	252	FC	374	³
157	9D	235	Ø	189	BD	275	¢	221	DD	335	¦	253	FD	375	²
158	9E	236	×	190	BE	276	¥	222	DE	336	Ì	254	FE	376	_
159	9F	237	ƒ	191	BF	277	+	223	DF	337	_	255	FF	377	(Espacio)

Figura 3.5 Una versión del código ASCII extendido a 8 bits.

3.6.- Distancia entre elementos de un código. Distancia mínima de un código. Paridad.

Los diversos códigos analizados hasta el momento, tanto numéricos como alfanuméricos, cumplen con una diversidad de funciones que han sido analizadas en los diferentes apartados de este capítulo. El concepto de codificación, no obstante, incluye algunas otras posibilidades de utilización, que en ciertos casos llegan a ser tan importantes como la representación que implican. En el caso de los códigos binarios, uno de los problemas principales que suele surgir en las diferentes utilizaciones de los mismos tiene que ver con la confiablidad y seguridad de dicha codificación. Dicho en otros términos, no solamente es importante codificar la información siguiendo las reglas del código que se haya elegido, sino que, además, debe tenerse la certeza de que dicho código permitirá la recuperación de la información en el momento que sea necesario.

En muchos casos, la formación de un código no contempla más objetivos que la representación para la cual ese código ha sido pensado. En otros casos, por el contrario, además de la representación, se requieren contemplar otros objetivos, entre lo cuales, como se ha dicho, la seguridad de la información representada es uno de los más importantes. Los conceptos a analizar en los próximos apartados tienen que ver con la seguridad de la información codificada mediante códigos binarios.

3.6.1.- Distancia entre elementos de un código.

Como ya se ha dicho, la formación de un código binario solamente requiere la relación biunívoca entre los elementos a codificar y las combinaciones binarias elegidas, sin exigir ningún otro criterio, salvo, claro está, la adecuada elección de la cantidad de bits a utilizar. Por consiguiente, la generación de los diversos códigos a utilizar puede surgir de muy variadas maneras, las que tendrán que ver con las características requeridas.

Se define como **distancia** entre dos elementos cualesquiera de un código a la cantidad de bits que deben modificarse para llegar desde uno de esos elementos al otro.

Este concepto, muy simple, permite determinar, dadas dos combinaciones cualesquiera de un código, la cantidad de alteraciones que, producidas simultáneamente en una de las palabras que forman el código, la convertirán en otra palabra diferente pero también perteneciente al mismo código. Al plantearse el concepto de distancia entre dos combinaciones cualesquiera, queda claro que el valor puede ser diferente, según los pares de combinaciones elegidos. Así, en el caso del código 8421, se pueden observar combinaciones que difieren en un solo bit (distancia 1), combinaciones que difieren en dos bits (entre el 5 y el 6, por ejemplo, cambian dos bits), otras que difieren en tres bits y hasta un caso en el que la distancia es cuatro. Nótese que el concepto de distancia entre palabras de un código no necesariamente tiene relación con que dichas combinaciones sean consecutivas. Puede tomarse como ejemplo la situación que vincula a los elementos 7 y 8 de dicho código, que, siendo consecutivos, ofrecen cuatro cambios de bits para llegar de uno al otro.

Se define como **distancia mínima** de un código, o simplemente como **distancia de un código**, a la menor de todas las distancias que puedan encontrarse analizando los distintos pares de combinaciones que forman el código en cuestión.

De la tabla que sigue, figura 3.6, puede observarse que el código 8421 ofrece varias combinaciones que difieren en un sólo bit, por lo que la distancia entre las mismas es uno, de donde resulta que la distancia del código 8421 es también uno.

Elemento a codificar	Representación 8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Fig. 3.6.- Otra vez el código 8421

3.6.2.- Paridad. Paridad par e impar.

Al analizar el concepto de seguridad de la información codificada en binario se debe entender dicho concepto de seguridad relacionándolo con el concepto de **error**. La información binaria que se utiliza dentro de cualquier sistema de computación o que se trasmite a través de algún sistema de comunicaciones está sujeta a la probabilidad de que por razones propias o externas al sistema en cuestión se produzcan modificaciones en la información binaria. Estas interferencias, internas o externas pueden hacer que un bit modifique su valor por lo que un cero se convertirá en uno y viceversa, no habiendo, en este caso, ninguna otra alternativa posible.

La necesidad de evitar la propagación de errores en la información trasmitida o almacenada requiere aumentar la capacidad de los códigos utilizados al respecto, para lo cual se utiliza, como elemento básico a tal efecto, el concepto de **paridad**. El mismo implica la modificación de la información codificada mediante el agregado de uno o más bits adicionales, los que cumplirán con la misión de aumentar la distancia del código y de identificar, de alguna manera, las combinaciones válidas (sin errores) de aquellas invalidadas por la existencia de algún error.

Un bit de paridad es un bit que se agrega a todas las combinaciones de un código con el objeto de que las mismas presenten el mismo aspecto en cuanto a la cantidad de unos y ceros que las forman. Así, un código está definido con **paridad par en los unos** si se configuran todas las palabras de un código con un número par de bits con valor de uno. De la misma manera, si todas las palabras del código presentan un número impar de bits en uno se determina que dicho código tiene **paridad impar en los unos.** La paridad de un código puede definirse también sobre los bits de valor cero, por lo que pueden plantearse tanto códigos de paridad par o impar en ceros. Es importante acotar que en el caso de los códigos de longitud de palabra fija, definir la paridad sobre los unos inmediatamente define también la paridad de los ceros.

Debe quedar claro que el concepto de paridad no requiere que las combinaciones que forman el código tengan todas exactamente la misma cantidad de unos, sino que solamente requiere que todas las combinaciones tengan un número par de unos (si se ha definido al código con paridad par) o impar de unos (en el otro caso).

El agregado de este bit de paridad implica el aumento de la distancia original del código en una unidad, por lo que un código de distancia d pasará a tener distancia $d+1$ una vez incorporado el bit de paridad. Este incremento de distancia permitirá que el código en cuestión se convierta en un

código hábil para la detección de errores, según se verá en el apartado siguiente. Si se incorpora un bit de paridad al código 8421 planteado anteriormente se lo convierte en un código de distancia 2:

Código sin paridad (distancia 1)	Código con paridad par (distancia 2)	Código con paridad impar (distancia 2)
0	00000	00001
1	00011	00010
2	00101	00100
3	00110	00111
4	01001	01000
5	01010	01011
6	01100	01101
7	01111	01110
8	10001	10000
9	10010	10011

3.7.- Detección y corrección de errores en información binaria.

Detectar errores en una información binaria significa reconocer la eventual modificación de uno o más bits de una palabra durante su almacenamiento o transmisión. La generación de errores en un sistema binario es, obviamente, una situación accidental, que no siempre puede prevenirse pero que sí puede detectarse o, llegado el caso, incluso corregirse, lo que significaría, en el primer caso, determinar que la información recuperada no es la misma que se almacenó o trasmitió, sin posibilidad de determinar el o los bits equivocados. En el caso de tener la posibilidad de corrección, lo que termina ocurriendo es que, más allá de los errores producidos, existen métodos que permitirán recuperar la información original, tal como se la generó antes de la ocurrencia de dichos errores. Para lograr tales objetivos, la información que se pretende proteger (mediante la detección o corrección de errores) deberá estar representada en un código cuya distancia sea adecuada para tales fines.

Como ya ha sido dicho, que un código sea de distancia ***d*** implica que entre dos cualesquiera de sus combinaciones difieren como mínimo ***d*** bits. Un error, que implica un bit cuyo valor original se ha modificado, resulta ser una combinación binaria que difiere en un bit con respecto a la combinación a partir de la cual se generó el error. Desde el concepto de distancia, se puede decir que la ocurrencia de un error en una combinación binaria de un código presenta distancia 1 con respecto a la combinación original desde la que se produjo el error.

El desarrollo de códigos binarios que permitan detectar o corregir errores implica, por consiguiente, pensar en códigos de distancia mayor que uno. De lo expresado en el apartado anterior surge que el agregado de bits de paridad es la forma de incrementar la distancia de un código con el objeto de hacerlo apto para su utilización en la detección y corrección de errores.

Se puede encontrar una relación directa entre la distancia de un código y la cantidad de errores que dicho código puede detectar y corregir. En la medida que el código tenga mayor distancia, mayor será su capacidad de detección y corrección, por lo que, en función de la probabilidad de ocurrencia de errores, el diseñador del código deberá elegir la distancia más apropiada para una determinada codificación.

La relación mencionada puede expresarse algebraicamente a través de la desigualdad:

$$\mathbf{D > d+k}$$

Expresión en la que **D** es la distancia mínima del código a utilizar, **d** es la cantidad de errores que se desean detectar y **k** la cantidad de errores que se desean corregir. Debe tenerse en cuenta además que cada error que se desea corregir debe previamente ser detectado, por lo que se requiere cumplir como condición adicional que **d ≥ k.** Esta condición adicional implica como ejemplo que para obtener un código que permita detectar y corregir un error en una combinación del mismo su distancia debe ser **tres.**

3.8.- Códigos para corrección de errores. Ejemplo: la corrección de errores mediante códigos de Hamming.

Como ejemplo de lo expresado anteriormente se considerará el caso, bastante habitual, de pretender un código que permita detectar y corregir un único error en una combinación del mismo. De acuerdo con lo dicho, se requerirá un código de distancia 3. El planteo a seguir implica determinar la forma en que puede llevarse a distancia 3 un código cualquiera de distancia 1, como pueden ser el 8421 en el caso de códigos numéricos o el ASCII en el caso de códigos alfabéticos.

Uno de los métodos más utilizados para resolver este planteo es el que en 1950 desarrollara Richard W. Hamming (1915-1998). La teoría matemática planteada permite convertir cualquier código de distancia unitaria, no importa la cantidad de bits que lo forman, en un código de distancia tres. El método plantea la formulación de ecuaciones parciales de paridad par de acuerdo con determinadas pautas.

A tal efecto, el método plantea que el número **p** de bits de paridad requeridos para una cantidad **d** de bits de datos (y por ende la cantidad de ecuaciones a plantear) se determina a través de la siguiente inecuación:

$$d + p < 2^p$$

De donde surge una relación simple que permite calcular la cantidad de bits de paridad requeridos, según sea el tamaño de la palabra a codificar:

Cant. Máx. bits en la palabra	**Cantidad bits de paridad**	**Total bits del código corrector de errores**
4	3	7
11	4	15
26	5	31

Conocida la cantidad de bits de paridad a incluir, resta definir cuales serán los cálculos de paridad a realizar, para lo cual el método plantea que cada bit de paridad es el resultante de una ecuación de paridad par, que se determinará incluyendo determinados bits de datos con la condición fundamental de generar un sistema de ecuaciones de paridad independientes, lo que, en el más puro sentido matemático, significa que no puede obtenerse ninguna de las ecuaciones a partir de las demás. Restará definir la forma de construir las ecuaciones de paridad para que cumplan con este requisito.

Una forma más o menos sistemática de resolver esta última cuestión consiste en acomodar la palabra ya codificada intercalando los bits de datos y los de paridad, de forma tal que estos últimos ocupen posiciones predeterminadas: **aquellas cuya posición es potencia de dos**.

Para una mejor comprensión del tema se planteará un ejemplo basado en una palabra de cuatro bits que puede corresponder, por ejemplo a una palabra de algún código BCD.

Si la palabra original está constituida por los bits ABCD, se deberán agregar, según lo ya mencionado, tres bits de paridad par, PQR, con los cuales se constituirá una palabra única de siete bits. La distribución de los bits en esta palabra será:

P1 P2 X3 **P4** X5 X6 X7

Donde P1 P2 P4 = PQR y X3 X5 X6 X7 = ABCD.

Nótese que, habiendo numerado los siete bits en orden creciente de izquierda a derecha, las posiciones 1, 2 y 4 (potencias de dos) corresponderán a los bits de paridad. Las tres ecuaciones de paridad requeridas resultarán de acomodar los cuatro bits de información en forma arbitraria, de modo de obtener tres ecuaciones independientes.

En realidad, la forma arbitraria puede no ser tal si se acepta una sistematización para distribuir los bits de datos. Obsérvese la distribución de las ecuaciones siguientes:

P4 = Paridad par (X5,X6,X7)

P2 = Paridad par (X3,X6,X7)

P1 = Paridad par(X3,X5,X7)

Cada uno de los bits que originalmente formaban la palabra de datos fue distribuida en las ecuaciones de paridad, de forma tal que el subíndice indicador de su posición coincida con la suma de los subíndices correspondientes a los bits de paridad en cuyas ecuaciones se lo incluyó. Como ejemplo, el bit X3 fue acomodado en las ecuaciones correspondientes a P1 y P2, el bit X5, en las que corresponden a P1 y P4, el bit X6, en las que corresponden a P2 y P4, y el bit X7 en las correspondientes a P1, P2 y P4 (la suma de los tres subíndices da siete).

Esta distribución asegura la independencia de las ecuaciones. Además, y como se verá a continuación, genera la capacidad de determinar, en caso de producirse un error, cual fue el bit erróneo. Al poder determinar cual fue el bit equivocado, su simple inversión permitirá la corrección del error.

El sistema de detección de posibles errores y su corrección se basara en el análisis de la palabra completa, planteando nuevamente las ecuaciones de paridad, pero ahora para verificar que se cumplan las condiciones siguientes:

Paridad (P4,X5,X6,X7) = par

Paridad (P2,X3,X6,X7) = par

Paridad (P1,X3,X5,X7) = par

Estas expresiones son las mismas planteadas anteriormente para la determinación de los tres bits de paridad, por lo que en una situación normal las verificaciones correspondientes deberían dar que efectivamente se cumplen las paridades parciales mencionadas. En una situación de error, en cambio, alguno de los bits que forman la palabra (sea alguno de los bits de datos originales o

alguno de los bits de paridad generados para proteger a dichos datos) se modifica. Por consiguiente, la verificación de estas últimas ecuaciones no dará paridad par en todas ellas.

Del análisis de las ecuaciones que no verifiquen la paridad par surgirá con claridad cual es el bit (de haber sido uno solo) cuyo valor fue alterado. En efecto, dado que la distribución de los distintos bits en las ecuaciones de paridad es único para cada uno de ellos, el bit en error alterará únicamente aquellas ecuaciones en las que fuera incluido, sin tocar las demás. Si replanteamos las ultimas expresiones como un nuevo calculo de paridad, podríamos expresarlas de la forma siguiente:

E4 = Paridad (P4,X5,X6,X7) = cero, si no hay errores en P4,X5,X6,X7

E2 = Paridad (P2,X3,X6,X7) = cero, si no hay errores en P2,X3,X6,X7

E1 = Paridad (P1,X3,X5,X7) = cero, si no hay errores en P1,X3,X5,X7

En caso de haber errores, una o mas de las ecuaciones no responderá al cálculo de paridad par. Por consiguiente, se podrá determinar, en función de cuales hayan sido las ecuaciones en error, cual fue el bit que lo provocó. Los resultados pueden resumirse en la tabla siguiente:

E4	**E2**	**E1**	**Bit en error**
0	0	0	Ninguno
0	0	1	P1
0	1	0	P2
0	1	1	X3
1	0	0	P4
1	0	1	X5
1	1	0	X6
1	1	1	X7

El método desarrollado puede ampliarse para cualquier longitud de palabra de datos, con solo considerar la cantidad de bits necesarios en cada caso, según la expresión oportunamente planteada.

En los ejemplos que siguen se verá la forma de aplicar los criterios de Hamming a una palabra de cuatro bits, y a una de siete bits que podría representar un dato codificado en ASCII.

Ejemplo 3.4:

Se desea codificar según el método de Hamming la palabra de cuatro bits 0101, con el objeto de permitir la detección y corrección de un eventual error.

El método implica el agregado de tres bits de paridad, a la palabra original, de forma de obtener una palabra de siete bits con la siguiente forma:

***P1 P2** X3 **P4** X5 X6 X7*

Para nuestro ejemplo:

P1 P2 0 P4 1 0 1

De las expresiones correspondientes, se determinan:

P4 = Paridad par (X5,X6,X7) = Paridad par (1,0,1) = 0

P2 = Paridad par (X3,X6,X7) = Paridad par (0,0,1) = 1

P1 = Paridad par (X3,X5,X7) = Paridad par (0.1,1) = 0

Por lo tanto, la palabra completa, incluyendo los bits de paridad será:

***P1 P2** X3 **P4** X5 X6 X7 = 0 1 0 0 1 0 1*

Ejemplo 3.5:

Se desea codificar según el método de Hamming la palabra de siete bits 1001001 (en el código ASCII, corresponde a la letra A), con el objeto de permitir la detección y corrección de un eventual error.

Siguiendo el planteo del ejemplo anterior, deberán agregarse cuatro bits de paridad, los que ocuparán las posiciones cuyo subíndice es potencia de dos, configurando las ecuaciones de paridad en la forma planteada anteriormente, de forma de obtener una palabra de once bits con la siguiente forma:

***P1 P2** X3 **P4** X5 X6 X7 P8 X9 X10 X11*

Para nuestro ejemplo:

P1 P2 1 P4 0 0 1 P8 0 0 1

De las expresiones correspondientes, se determinan:

P8 = Paridad par (X9,X10,X11) = Paridad par (0,0,1) = 1
P4 = Paridad par (X5,X6,X7) = Paridad par (0,0,1) = 1
P2 = Paridad par (X3,X6,X7,X10,X11) = Paridad par (1,0,1,0,1) = 1
P1 = Paridad par (X3,X5,X7,X9,X11) = Paridad par (1,0.1,0,1) = 1

Por lo tanto, la palabra completa, incluyendo los bits de paridad será:

***P1 P2** X3 **P4** X5 X6 X7 P8 X9 X10 X11= 1 1 1 1 0 0 1 1 0 0 1*

Ejemplo 3.6:

Se ha recibido la palabra de doce bits (código ASCII extendido) 0011 0101 0100

Se desea determinar cual fue la palabra originalmente generada, si la misma se planteó de acuerdo con los criterios de Hamming.

En el caso de doce bits, de acuerdo con lo visto anteriormente, la palabra original es una palabra de cuatro bits a la que se codificó con cuatro bits de paridad. El formato de la palabra, en forma similar al caso anterior, es:

P1 P2 *X3* ***P4*** *X5 X6 X7 P8 X9 X10 X11 X12*

Las ecuaciones de generación de paridad son:

P8 = Paridad par (X9,X10,X11,X12)
P4 = Paridad par (X5,X6,X7,X12)
P2 = Paridad par (X3,X6,X7,X10,X11)
P1 = Paridad par (X3,X5,X7,X9,X11)

Por lo que las condiciones a verificar son:

E8 = Paridad par (P8,X9,X10,X11,X12)
E4 = Paridad par (P4,X5,X6,X7,X12)
E2 = Paridad par (P2,X3,X6,X7,X10,X11)
E1 = Paridad par (P1,X3,X5,X7,X9,X11)

Para nuestro ejemplo: (0011 0101 0100)

E8 = Paridad par (P8,X9,X10,X11,X12) = Paridad par (1,0,1,0,0) = 0
E4 = Paridad par (P4,X5,X6,X7,X12) = Paridad par (0,1,0,1,0) = 0
E2 = Paridad par (P2,X3,X6,X7,X10,X11) = Paridad par (0,1,1,0,1,0) = 1
E1 = Paridad par (P1,X3,X5,X7,X9,X11)= Paridad par (0,1,0,0,0,0) = 1

Por consiguiente, dado que la palabra formada por E8 E4 E2 E1 vale 0011, se determina que el dato obtenido tiene un error en el bit X3, el que, en consecuencia, valdrá cero. Eliminando los bits de paridad, y corrigiendo el bit erróneo se obtiene el dato original: 0010 0100.

INTRODUCCIÓN A LOS SISTEMAS DIGITALES

ING. FERNANDO IGNACIO SZKLANNY

CAPÍTULO IV:

HERRAMIENTAS BÁSICAS:

EL ÁLGEBRA DE BOOLE

CAPÍTULO 4: EL ALGEBRA DE BOOLE

4.1. Introducción.

Los métodos de análisis y síntesis de circuitos lógicos que se analizarán a lo largo de la presente obra tienen, como se ha dicho en capítulos anteriores, dos bases fundamentales. Una de ellas, la representación binaria de la información ya ha sido analizada en los capítulos anteriores. En este capítulo se desarrollarán los conceptos asociados con el álgebra de Boole, herramienta matemática desarrollada por George **Boole** en el siglo XIX. A través de esta herramienta se podrán definir elementos y operaciones binarias, las que a su vez permitirán, utilizando los habituales elementos de conmutación, obtener circuitos de comportamiento binario que permitan implementar las llamadas funciones lógicas: representaciones circuitales de expresiones binarias. La combinación de los conceptos binarios descriptos en los capítulos anteriores, las funciones booleanas objeto de este capítulo y los circuitos de conmutación a describir posteriormente permitirán llegar a los distintos tipos de circuitos lógicos que, hoy por hoy, son fundamento básico de todos aquellos sistemas digitales utilizados en tantos campos de aplicación.

4.2. El álgebra de Boole. Definición. Postulados básicos. Leyes y teoremas.

El álgebra de Boole es un esquema de razonamiento lógico, para cuya definición se requiere de un conjunto de **postulados** o **axiomas** básicos, y de una serie de **leyes** o **teoremas** que complementan dichos postulados. Los postulados del álgebra de Boole, al igual que los del álgebra convencional, son verdades que no requerirán demostración. En cambio, cuando se habla de teoremas, se habla de enunciados que requieren demostración, basada esta en los axiomas fundamentales o en otros teoremas previamente demostrados.

El álgebra de Boole es un álgebra de **elementos**, entre los cuales se establecen **relaciones** y **operaciones**. Si dichas relaciones y operaciones permiten plantear un conjunto **cerrado** de postulados independientes, se podrá admitir a dicho conjunto de postulados como definición del álgebra. Se dirá que el conjunto de postulados es cerrado cuando no haya contradicciones entre los mismos. Planteada como un álgebra de proposiciones, la teoría algebraica de Boole se convirtió en un fundamento importante para aplicaciones lógicas. Los desarrollos matemáticos realizados por Claude **Shannon**, entre otros, hicieron del álgebra de Boole y de sus derivados uno de los pilares de la computación digital.

Existen diferentes grupos de postulados para definir al álgebra de Boole. En efecto, de todas las definiciones que puedan llegar a plantearse, podrán definirse algunas como postulados, aceptarlas como tales, y demostrar otro conjunto de esas definiciones como

teoremas a ser demostrados a partir de aquellos. El conjunto de postulados que se utilizarán en esta obra fue definido por Edward **Huntington**[1] a principios de este siglo**.** En dichos postulados se establecen la definición del álgebra en cuanto al universo en que se la define, las relaciones requeridas entre los elementos que la forman, y las operaciones a realizar entre dichos elementos.

El primer postulado permitirá establecer una definición del álgebra:

Postulado 1: Definición.

Un álgebra de Boole es un sistema algebraico cerrado, definido por un conjunto C, formado por al menos dos elementos a y b, entre los cuales deberá cumplirse una relación de equivalencia, y dentro del cual se definirán dos operaciones, suma lógica y producto lógico, cerradas en dicho conjunto C.

Si se deseara expresar matemáticamente el enunciado anterior, resultaría una expresión simbólica del siguiente estilo:

$$\exists \boldsymbol{C} \wedge \exists (\boldsymbol{a}, \boldsymbol{b}, ..) \in \boldsymbol{C} \,/\, \boldsymbol{aRb} \wedge \exists (\boldsymbol{a} + \boldsymbol{b}) \in \boldsymbol{C} \wedge \exists (\boldsymbol{a}.\boldsymbol{b}) \in \boldsymbol{C}$$

En la misma se utilizan los clásicos operadores de la lógica simbólica para enunciar la existencia de un conjunto C en el que se definirán los elementos y operaciones del álgebra booleana. Se establece, además, la existencia de al menos dos elementos, llamados genéricamente ***a*** y ***b***, entre los cuales se requiere la existencia de una relación de equivalencia. Para recordar este concepto, debe enunciarse la existencia de una relación de equivalencia entre elementos cuando se cumplen el principio de identidad, el de simetría y el de transitividad:

aRa	Principio de identidad
$aRb \Rightarrow bRa$	Principio de simetría
$aRb \wedge bRc \Rightarrow aRc$	Principio de transitividad

Los elementos integrantes del conjunto están sujetos a dos operaciones lógicas, las que se denominarán como **suma (+)** y **producto (.)**, respectivamente. A los efectos de evitar confusiones con las operaciones del álgebra tradicional, se utilizan en forma habitual los nombres de **suma lógica** y **producto lógico.** Las características de las dos operaciones surgen a partir de los tres siguientes postulados de Huntington. Los mismos se definen como pares de postulados, correspondientes en cada par uno a la suma lógica y el otro al producto lógico, operaciones que, según surge de lo expresado, tienen propiedades similares.

Algunos de estos postulados son idénticos a los que, en el álgebra convencional, corresponden a las operaciones de igual nombre. Esto, obviamente, no significa que las

[1] Edward Huntington. Sets of Independent Postulates for the Algebra of Logic. 1904.
New sets of independent postulates for the Algebra of Logic. 1933.

operaciones aritméticas de suma y producto sean iguales, equivalentes o similares a sus homónimas del álgebra de Boole.

Postulado 2: Postulado de Conmutatividad.

Las operaciones de suma y producto lógicos son conmutativas.

Con este simple enunciado se plantea la primera propiedad de las operaciones del álgebra de Boole. De acuerdo con lo dicho, se deberá dividir este enunciado en dos, uno para cada una de las operaciones:

***Postulado 2a**: La suma lógica es una operación que admite la propiedad conmutativa.*

***Postulado 2b:** El producto lógico es una operación que admite la propiedad conmutativa.*

Expresados simbólicamente, resultan las siguientes:

$$\forall(a,b)\exists C \rightarrow a+b=b+a \quad \text{Postulado 2.a}$$
$$\forall(a,b)\exists C \rightarrow a.b=b.a \quad \text{Postulado 2.b}$$

Postulado 3: Postulado de asociatividad.

Las operaciones de suma y producto lógicos son asociativas.

.

También con un simple enunciado se define una nueva propiedad de ambas operaciones lógicas, similar a la que se cumple en la suma y en el producto en el álgebra convencional. Respetando la individualidad de las dos operaciones se tiene:

***Postulado 3a**: La suma lógica admite la propiedad asociativa de los sumandos.*

***Postulado 3b**: El producto lógico admite la propiedad asociativa de los factores.*

Expresados en forma simbólica, los postulados admiten las siguientes expresiones:

$$\forall(a,b,c)\in C \rightarrow (a+b)+c=a+(b+c) \quad \text{Postulado 3.a}$$
$$\forall(a,b,c)\in C \rightarrow (a.b).c=a.(b.c) \quad \text{Postulado 3.b}$$

Postulado 4: Postulado de distributividad.

La suma y el producto lógico son respectivamente distributivos uno respecto del otro.

Separadamente para ambas operaciones, se obtendrá:

***Postulado 4a**: En el álgebra de Boole, la suma es una operación distributiva respecto del producto.*

***Postulado 4b**: En el álgebra de Boole, el producto es distributivo respecto de la suma.*

Matemáticamente:

$\forall(a,b,c) \in C \rightarrow (a+b).c = a.c + b.c$	Postulado 4.a
$\forall(a,b,c) \in C \rightarrow (a.b) + c = (a+c).(b+c)$	Postulado 4.b

Nótese la primera diferencia entre las operaciones lógicas y las del álgebra convencional, en el cual no existe equivalente al segundo postulado del par.

Los postulados restantes definen algunas propiedades de los elementos que forman el conjunto C. En particular, los mismos permiten definir algunos elementos esenciales para la existencia del Álgebra. Así pueden definirse los siguientes elementos.

Postulado 5: Existencia del elemento neutro.

***Postulado 5a:** Para la operación de suma puede definirse un elemento neutro N_1 tal que la suma lógica de cualquier elemento del conjunto y el neutro, dé por resultado el elemento de origen.*

***Postulado 5b**: Para la operación de producto puede definirse un elemento N_2, tal que el producto lógico de cualquier elemento del conjunto y el neutro, dé por resultado el elemento de origen.*

Las expresiones algebraicas correspondientes a este postulado son las siguientes:

$\forall a \in C \exists N1 / a + N1 = a$	Postulado 5.a
$\forall a \in C \exists N2 / a.N2 = a$	Postulado 5.b

Estas definiciones, y las propiedades de los respectivos neutros, son idénticas a las características del cero y el uno en el álgebra convencional, por lo que es absolutamente habitual definir a los dos neutros, respectivamente, con los nombres y símbolos que corresponden al 0 y al 1, en vez de N_1 y N_2.

Postulado 6: Existencia del elemento opuesto.

Existe para cada elemento del conjunto, un elemento opuesto al mismo tal que:

Postulado 6a: *La suma del elemento mencionado con su elemento opuesto da por resultado el neutro del producto.*

Postulado 6b*: El producto del elemento mencionado con su opuesto da por resultado el neutro de la suma.*

Este postulado enuncia la existencia de un elemento adicional que cumple con la propiedad de permitir la obtención del neutro de la suma a través de una operación de producto, así como la obtención del neutro del producto a través de una operación de suma. Esta característica es totalmente novedosa, si se pretende seguir comparando la estructura del álgebra booleana con la del álgebra convencional.

$\forall a \in C \exists \bar{a} / a + \bar{a} = N2$	Postulado 6.a
$\wedge / a.\bar{a} = N1$	Postulado 6.b

4.3. El álgebra de Boole como un álgebra binaria.

Los postulados anteriores forman un conjunto de afirmaciones consistente, en el sentido de no establecer contradicciones entre los mismos. Asimismo, se puede verificar la independencia de dichos postulados, dado que no hay ninguno que pueda deducirse a través de combinaciones de otros postulados del grupo detallado. A partir de los mismos, en consecuencia, surge la posibilidad de definir en forma completa las operaciones de suma y producto incluidas en la estructura algebraica.

Para permitir dicha definición, y para acomodarla al uso habitual, faltará un sólo paso, el que permitirá simplificar la estructura lógica para adecuarla, además, a los usos y necesidades de los circuitos y elementos de conmutación. Se procederá a la definición de un álgebra de Boole binaria, lo que se logrará mediante la modificación del enunciado del primer postulado de Huntington. En efecto, si en vez de enunciar la existencia de un conjunto formado al menos por dos elementos, como lo expresa dicho postulado, se define un conjunto formado solamente por dos elementos, de los restantes postulados surgirán claramente cuales son los elementos que conforman un álgebra de Boole binaria. Surge claramente que si, para cada una de las operaciones booleanas se define un elemento neutro, **estos dos, el cero y el uno, deberán ser los dos únicos elementos de dicho conjunto.**

Surge de todos modos otra cuestión. Dado que el sexto postulado requiere la existencia de un elemento opuesto para cada uno de los elementos que forman el conjunto, no podrá mantenerse la condición de un álgebra de Boole binaria salvo que se defina

expresamente que **el cero es el elemento opuesto al uno** y que **el uno es el elemento opuesto al cero.**

Puede verificarse rápidamente que este planteo no se contradice con los postulados, por lo que su validez no está en dudas.

A partir de estas definiciones, pueden obtenerse, con el uso de los postulados enunciados, las tablas que permitan resolver las dos operaciones de suma y producto lógicos.

Como el álgebra bajo análisis es binaria, la operación de suma lógica se reduce a solamente cuatro alternativas, las que surgen directamente de los postulados enunciados.

0 + 0 = 0 1 + 0 = 1 0 + 1 = 1 1 + 1 = 1

Con criterios similares, la operación de producto lógico se reduce a otras cuatro alternativas:

0 . 0 = 0 0 . 1 = 0 1 . 0 = 0 1 . 1 = 1

En general, estas operaciones se expresan en la forma de tablas, las que permiten ver mejor los resultados de las mismas, en función de los valores de los elementos operandos.

Planteada de esta forma, la operación suma lógica se obtendrá a través de la siguiente tabla binaria:

a	b	**a+b**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

en tanto que la operación de producto lógico se obtendrá en la forma siguiente:

a	b	**a.b**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Llegado a este punto, debe hacerse una aclaración. El lector notará que la tabla de verdad de la operación de producto lógico es idéntica, en sus resultados, a lo que se obtendría en una operación de multiplicación aritmética entre los números cero y uno. No ocurre así con la operación de suma lógica, en la que uno de los resultados obtenidos es distinto al que corresponde en aplicaciones aritméticas, sean estas

expresadas en el sistema de numeración decimal o en el sistema binario. Debe quedar perfectamente en claro, entonces, **que estas dos operaciones son totalmente distintas y nada tienen que ver con las operaciones aritméticas que fueran analizadas en el capítulo 2 del presente.**

4.4. Teoremas o leyes del álgebra de Boole.

A partir de los postulados del Algebra booleana, se pueden demostrar un conjunto de leyes o teoremas, que completan la enunciación de las propiedades de los elementos del álgebra y sus operaciones. Como ya ha sido expresado, los teoremas a enunciar requerirán una demostración, la que, en todos los casos, surge a partir de los postulados ya enunciados.

No es objeto de este texto introducirse en la demostración matemática rigurosa de estos teoremas o leyes, la que, por otra parte, puede encontrarse en la bibliografía relacionada con temas de lógica matemática. El lector podrá verificar la validez de estos teoremas a través de los distintos elementos y herramientas que se desarrollarán a lo largo de este capítulo y los siguientes.

4.4.1. Leyes de De Morgan.

Las leyes conocidas como de De Morgan vinculan las dos operaciones lógicas enunciadas en los postulados con la definición de elemento opuesto. En concreto, enuncian una propiedad particular del opuesto de una suma lógica y del opuesto de un producto lógico.

Las leyes pueden enunciarse de la siguiente forma:

El opuesto de una suma lógica de varios elementos booleanos se obtiene a través del producto lógico de los opuestos de cada elemento individual.

El opuesto de un producto lógico de varios elementos booleanos se obtiene a través de la suma lógica de los opuestos de cada elemento individual.

Expresadas estas dos leyes en forma algebraica resultan:

$$\forall a, b \in C \rightarrow \overline{(a+b)} = \overline{a}.\overline{b} \wedge$$
$$\overline{a.b} = \overline{a} + \overline{b}$$

4.4.2. Ley de la doble negación.

El elemento opuesto de cualquier elemento del álgebra da por resultado el elemento original.

Esta ley es aplicable a cualquier elemento u operación algebraica dentro del esquema de Boole. Si se parte de un elemento, o de un conjunto de elementos vinculados entre sí por uno o más operadores lógicos, obtener el opuesto de su opuesto dará por resultado el elemento o conjunto de elementos original. Las aplicaciones más importantes de esta ley surgen cuando la misma se combina con otros postulados o leyes, como las leyes de De Morgan ya mencionadas. Algunas aplicaciones específicas de esta ley se detallarán en apartados posteriores.

4.4.3. Ley de absorción.

Para cualquier par de elementos del conjunto booleano se verifica:

$$a + a.b = a \wedge a.(a + b) = a$$

La verificación del cumplimiento de este teorema es sencillo, dado que simplemente se basa en la aplicación de los postulados relacionados con las propiedades distributivas que una operación booleana satisface respecto de la otra.

4.4.4. Leyes de unicidad.

La suma lógica (o el producto lógico) de un elemento consigo mismo da por resultado el mismo elemento. Expresado esto en términos algebraicos:

$$\forall a \in C \rightarrow a + a = a \wedge a.a = a$$

Estos enunciados son solamente algunos de los teoremas o leyes que surgen en el álgebra de Boole. Más detalles acerca de los mismos, su función y su utilización se desarrollarán en los apartados posteriores de este capítulo.

4.5. Otros ejemplos de álgebra de Boole.

4.5.1. El álgebra de conjuntos como caso particular de un álgebra de Boole. Ejemplos y comparaciones.

El álgebra de conjuntos es un esquema lógico que se apoya en el concepto de **conjunto**. Un conjunto se define como una agrupación de elementos, entre los cuales pueden establecerse distintas relaciones. La definición de estas relaciones, de los operadores lógicos que los vinculan y de la estructura algebraica en la que se pretende desarrollar el planteo permite definir un conjunto de postulados que son, en esencia, los mismos postulados del álgebra de Boole, orientados hacia la estructura mencionada.

Como ya se ha dicho, un conjunto es una agrupación de elementos. No se hace referencia alguna al tipo de elementos o a sus características particulares. Lo único que se establece es la pertenencia de los elementos en cuestión a uno o más conjuntos. Con el objeto de confirmar la característica booleana del álgebra de conjuntos puede verificarse el cumplimiento de los postulados de Huntington. De hecho, el primer postulado requiere la existencia de dos operaciones dentro de la estructura, las que, en el caso del álgebra de conjuntos, son las de **unión** e **intersección**.

Se define la operación de unión entre dos conjuntos A y B al conjunto que se obtiene con todos los elementos de A y todos los elementos de B. (fig. 4.1).

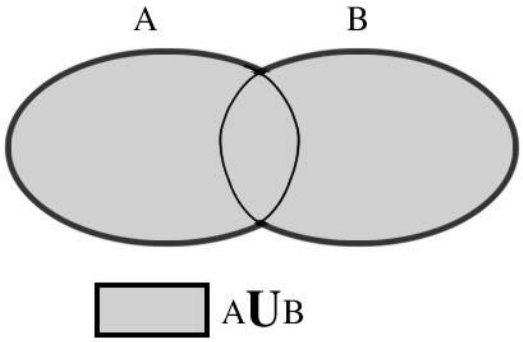


Fig. 4.1.: Operación de unión entre conjuntos

Asimismo, se define como intersección entre dos conjuntos A y B al conjunto que se obtiene con los elementos que pertenecen simultáneamente a ambos conjuntos A y B. (fig. 4.2).

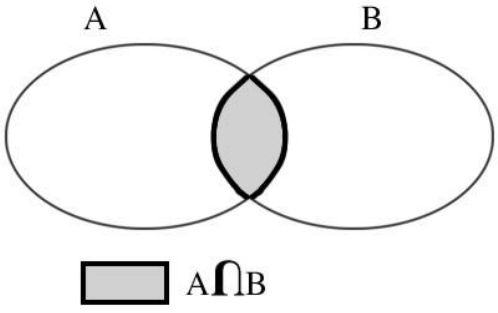


Fig. 4.2.: Operación de intersección entre conjuntos

De la simple observación visual puede deducirse que las operaciones de unión e intersección entre conjuntos satisfacen los postulados 2, 3 y 4 del álgebra de Boole.

Postulado 2: *Las operaciones de unión y la intersección entre conjuntos son conmutativas.*

$$A \cup B = B \cup A \wedge A \cap B = B \cap A \forall A, B$$

Postulado 3: *Las operaciones de unión y la intersección entre conjuntos son asociativas.*

$$(A \cup B) \cup C = A \cup (B \cup C) \forall A, B, C$$

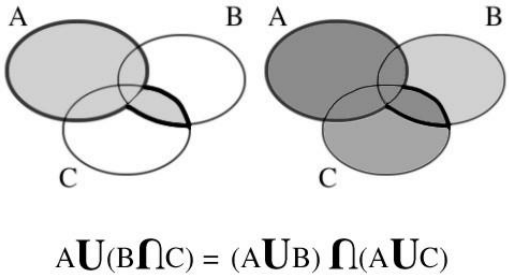
Postulado 4: *Las operaciones de unión y la intersección entre conjuntos son operaciones respectivamente distributivas entre sí.*

Las figuras 4.3. ilustran la verificación del postulado 4, que se expresa algebraicamente

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \forall A, B, C$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \forall A, B, C$$

según las siguientes ecuaciones:



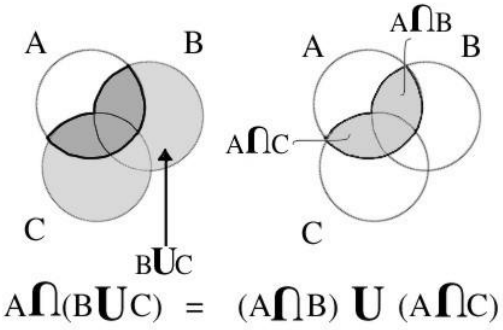


Fig. 4.3.: Propiedades distributivas entre la unión y la intersección de conjuntos

Las definiciones de los elementos neutros y del elemento opuesto completan los requerimientos de dichos postulados. Los elementos neutros están representados por el llamado conjunto vacío, elemento neutro de la operación de unión, y por el conjunto universal, elemento neutro de la operación de intersección.

Postulado 5: Existencia del elemento neutro.

Postulado 5a: *Para la operación de unión puede definirse un conjunto neutro N_1, llamado conjunto vacío, tal que la unión de cualquier conjunto con el neutro, dé por resultado el conjunto de origen.*

Postulado 5b*: Para la operación de intersección puede definirse un elemento N_2, o conjunto universal, tal que la intersección de conjunto con el neutro dé por resultado el conjunto de origen.*

La expresión lógica correspondiente se enuncia a continuación:

$$\forall A \exists \phi / A \cup \phi = A \wedge \exists \cup / A \cap \bigcup = A$$

Asimismo, el llamado complemento es la representación, dentro del álgebra de conjuntos, del elemento opuesto del álgebra de Boole.

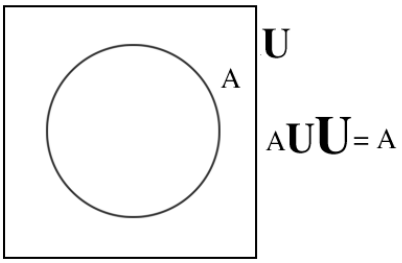


Fig. 4.4. La existencia del elemento neutro

Postulado 6: Existencia del elemento opuesto.

Existe para cada el conjunto, un conjunto complemento del mismo tal que:

Postulado 6a: *La unión del conjunto mencionado con su complemento da por resultado el conjunto universal.*

Postulado 6b: La intersección del conjunto *mencionado con su complemento da por resultado un conjunto vacío.*

$$\forall A \exists \overline{A} / A \cup \overline{A} = \bigcup \wedge A \cap \overline{A} = \overline{\Phi}$$

Nuevamente se plantea a continuación, en simbología lógica, el enunciado de estos postulados:

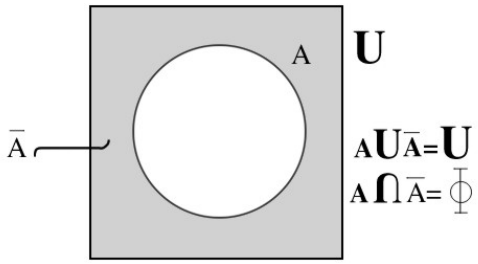


Fig. 4.5. La existencia del elemento opuesto (complemento) en el álgebra de conjuntos

La verificación de los postulados booleanos en una estructura como la del álgebra de conjuntos, por medio de los clásicos diagramas de Venn, así como la verificación de los teoremas, permite, en definitiva, plantear la existencia de una relación directa entre los enunciados del Álgebra de Boole y el Álgebra de Conjuntos, la que puede resumirse en la tabla siguiente:

BOOLE	**CONJUNTOS**
Elementos	Conjuntos
Suma lógica	Producto lógico
Elemento opuesto	Complemento
Neutro de la suma (cero)	Conjunto vacío
Neutro del producto (uno)	Conjunto universal.

En apartados posteriores surgirá la utilidad del Álgebra de Conjuntos para el desarrollo de métodos y conceptos asociados con las funciones lógicas y la electrónica digital, objeto final de esta obra.

4.5.2. El álgebra proposicional como caso particular de un álgebra de Boole. Ejemplos y consideraciones. La proposición como elemento binario. Proposiciones simples y compuestas.

El álgebra proposicional es un esquema lógico que se basa en el análisis de elementos conocidos como **proposiciones**. Una proposición puede definirse como cualquier expresión de la cual se pueda afirmar que es cierta o es falsa. Esta definición involucra

ya el carácter binario de las proposiciones, por lo que, para determinar si se trata de un álgebra de Boole, habrá que verificar si el análisis proposicional y las operaciones que vinculan a las mismas satisfacen los postulados de Huntington enunciados anteriormente.

Para realizar dicho análisis puede plantearse el álgebra proposicional como un espacio algebraico en el cual las proposiciones se vinculan entre sí a través de dos operadores, conocidos como **disyunción** ($\vee$; léase O) y **conjunción** ($\wedge$; léase Y). Estos dos operadores permiten vincular dos o más proposiciones para obtener otras proposiciones, cuyos grados de certeza o falsedad se definirán en función de los valores de las proposiciones originales y de los operadores utilizados.

Puede así plantearse una diferenciación entre las llamadas proposiciones simples y compuestas. Serán **proposiciones simples** aquellas cuyo grado de verdad o falsedad depende exclusivamente de la proposición enunciada, en tanto se definirán como **proposiciones compuestas** aquellas otras proposiciones en las que el grado de certeza o falsedad de la proposición obtenida dependa de las proposiciones más simples que la forman.

El planteo de los postulados del Algebra de Boole aplicados al Álgebra de proposiciones, permite enunciar:

Postulado 1: Definición.

Un álgebra de proposiciones es un sistema algebraico cerrado, definido por un conjunto C, formado por al menos dos proposiciones lógicas a y b, entre los cuales deberá cumplirse una relación de equivalencia, y dentro del cual se definirán dos operaciones, llamadas disyunción y conjunción, cerradas en dicho conjunto C.

Postulado 2: Postulado de Conmutatividad.

Las operaciones de disyunción y conjunción entre proposiciones son conmutativas.

Postulado 3: Postulado de asociatividad.

Las operaciones de disyunción y conjunción entre proposiciones son asociativas.

Postulado 4: Postulado de distributividad.

La disyunción y la conjunción entre proposiciones son operaciones respectivamente distributivas una respecto de la otra.

Postulado 5: Existencia del elemento neutro.

Postulado 5a: *Para la operación de disyunción puede definirse un elemento neutro N_1 tal que la disyunción entre cualquier proposición y el elemento neutro, dé por resultado la proposición de origen.*

Postulado 5b*: Para la operación de conjunción puede definirse un elemento N_2, tal que la conjunción entre cualquier proposición y el elemento neutro, dé por resultado la proposición de origen.*

Postulado 6: Existencia del elemento opuesto.

Existe para cada proposición dentro del conjunto, una proposición opuesta, tal que:

Postulado 6a: *La disyunción de la proposición y su opuesto da por resultado el elemento neutro de la conjunción.*

Postulado 6b*: La conjunción de la proposición y su opuesto da por resultado el neutro de la disyunción.*

Es habitual, en el caso del álgebra de proposiciones, pensar en las siguientes simplificaciones:

La existencia del elemento opuesto puede plantearse como la existencia de una proposición negada, la que se obtiene de la proposición inicial agregando a la misma, donde sea gramaticalmente correcto, el adverbio no.

La existencia del elemento neutro de la operación de disyunción se considera equivalente a una proposición siempre falsa. Algunos autores denominan contradicción a este elemento neutro.

Asimismo, el elemento neutro de una operación de conjunción es el equivalente a una proposición siempre cierta.

En resumen, puede plantearse que existe una relación directa entre las tres variantes presentadas, las que permiten ampliar la tabla del apartado anterior para enunciar la siguiente tabla de analogías.

BOOLE	**CONJUNTOS**	**PROPOSICIONES**
Elementos	Conjuntos	Proposiciones
Suma lógica (+)	Unión($\cup$)	Disyunción ($\vee$)
Producto lógico (.)	Intersección($\cap$)	Conjunción ($\wedge$)
Elemento opuesto	Complemento	Negación (no)
Neutro de la suma (0)		Conjunto vacío (Φ) Falsedad (F)
Neutro del producto (1)		Conjunto universal. (U) Certeza (V)

4.6. Del álgebra de proposiciones a las funciones lógicas. Variables y funciones lógicas. Operadores lógicos.

La estructura del álgebra de proposiciones, y fundamentalmente los conceptos ya definidos de proposiciones simples y compuestas, permiten hacer una analogía con expresiones del álgebra numérica, en la que el concepto de variable (sea independiente o dependiente) lleva inmediatamente al de función. Así, en el álgebra proposicional, queda claro que las proposiciones llamadas compuestas dependen directamente, en lo

que hace a sus valores lógicos de verdad o falsedad, de los valores que adopten las variables simples que las constituyen. A partir de esta relación pueden definirse inmediatamente los conceptos de variable y función lógicas.

Podrá definirse como **variable lógica** a cualquier proposición cuyo grado de verdad o falsedad no dependa de otras proposiciones que la constituyan. Desde este punto de vista, una variable lógica es una proposición simple.

Se definirá entonces como **función lógica** a toda aquella proposición cuyo valor de verdad dependa del valor de verdad o falsedad de otras proposiciones. En otras palabras, una función lógica es una proposición compuesta, ya que este tipo de proposiciones depende de otras proposiciones para determinar si, ante una determinada combinación de valores de verdad o falsedad de aquellas, adopta un valor de verdadero o falso. Queda claramente establecido que una función lógica es, por naturaleza, binaria.

Cuando se enuncia una proposición compuesta en lenguaje coloquial, el mismo enunciado de la misma permite encontrar la relación existente entre las proposiciones simples y la proposición compuesta formada por aquellas. En un enunciado del tipo:

Iré al cine si no llueve y si encuentro alguien que me acompañe,

Queda clara la dependencia entre la acción propuesta, la de ir al cine, condicionada por dos situaciones que podrían tomarse como independientes entre sí e independientes de cualquier otra situación. Se podrá enunciar la proposición anterior como una función lógica (dado que se trata de un enunciado de carácter binario) de dos variables lógicas (también binarias). En este caso, podrá definirse las variables lógicas (proposiciones simples):

A: No llueve **y** B: Encuentro alguien que me acompaña.

La proposición compuesta o función lógica enunciada resultará F=F(A,B), resultando claro, además, en virtud del enunciado planteado, que la vinculación planteada es del tipo conjuntivo (ambas proposiciones simples vinculadas por un operador Y de conjunción).

4.7. Representación de funciones lógicas mediante tablas de verdad. Expresión algebraica de las funciones lógicas.

La representación de una función lógica se hace más evidente mediante la utilización de una tabla de verdad. Se entiende como tal a una representación tabular, del tipo de las utilizadas anteriormente para la definición de las operaciones básicas, en la que se representan, por una parte, todas las variables independientes de la función lógica y, por otro lado, la función o variable dependiente, indicando, para cada una de las combinaciones de verdad y falsedad de esas variables independientes, el valor de verdad de la función. Esta presentación resultará en una tabla de 2^n combinaciones (renglones en la tabla), siendo n el número de variables independientes de la función. Se

ejemplifican a continuación las tablas de verdad para combinaciones de dos y de tres variables independientes.

a	b	**F(a,b)**
F	F	
F	V	
V	F	
V	V	

a	b	c	**F(a,b,c)**
F	F	F	
F	F	V	
F	V	F	
F	V	V	
V	F	F	
V	F	V	
V	V	F	
V	V	V	

Pueden observarse algunas características de estas tablas, más allá de que las mismas no se hayan completado. Para una dada función lógica, la última columna deberá representar el valor de verdad o falsedad de la misma en función de la verdad o falsedad de todas las proposiciones simples que la constituyen. Los distintos renglones de la tabla de verdad indican todas las combinaciones de verdad y falsedad que esas proposiciones simples pueden adoptar.

Ordenadas en la forma precedente, se observa una importante similitud entre las tablas de verdad del álgebra proposicional y el ordenamiento binario de los números naturales, expresados respectivamente en 2 y en 3 bits.

A todos los efectos prácticos, y habiendo planteado el problema de las funciones lógicas como un tema similar al de las funciones del álgebra numérica, podrá admitirse, apoyados por la tabla del apartado anterior, expresar la situación de falsedad de una proposición mediante un 0 lógico, y la verdad de las mismas como un 1 lógico. En este planteo, las tablas de verdad pasarán a tomar las formas siguientes:

a	b	**F(a,b)**
0	0	
0	1	
1	0	
1	1	

a	b	c	**F(a,b,c)**
0	0	0	
0	0	1	

0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Cualquier función lógica puede llevarse a una tabla de verdad. Si la función se expresa en un formato coloquial, el enunciado deberá descomponerse de modo de ubicar todas las proposiciones simples (variables lógicas) que definen la proposición compuesta (función lógica). A partir de esa descomposición se deberá analizar qué proposiciones simples (o que combinación de las proposiciones simples) hacen cierta la función, indicando dicha situación en la tabla correspondiente.

Ejemplo 4.1.: Representar la tabla de verdad para la proposición compuesta siguiente:

Podré salir de vacaciones si termino de rendir los exámenes del año y no tengo trabajo, o si, teniendo trabajo, tengo exámenes para rendir y las fechas no coinciden con el período de licencia.

Si se analiza sintácticamente esta proposición, surge que la posibilidad de salir de vacaciones depende de las siguientes condiciones:

A: Tengo exámenes para rendir.
B: Tengo trabajo.
C: Las fechas de exámenes coinciden con el período de licencia.

Puede decirse que las vacaciones planteadas son función de las tres condiciones (variables independientes) enunciadas, en las siguientes alternativas, que deben cumplirse en forma simultanea.

- Independientemente de las fechas de exámenes y su coincidencia con el periodo de licencia (C), no tener exámenes para rendir (no A) y no tener trabajo (no B).

O alternativamente

- Tener exámenes (A), tener trabajo (B), y no coincidir las fechas de exámenes con el periodo de licencia (no C)

La tabla de verdad para la función planteada es la siguiente:

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

1	*1*	*1*	***0***

Del ejemplo anterior surge una observación adicional. Se ha planteado la posibilidad de representar una función lógica a través de una tabla de verdad. En dicha tabla de verdad se ha expresado, para la función representada, como 1 cada una de las situaciones que corresponden a la veracidad de la función, y como 0, aquellas combinaciones de las variables lógicas.

En el mismo ejemplo se han enunciado, además, las alternativas, en lo que hace a las condiciones de verdad o falsedad de las variables independientes, que hacen que la función planteada sea cierta. Este planteo se completa, desde un punto de vista algebraico, encontrando una forma de expresar esa relación como una ecuación lógica. En este punto se puede plantear la expresión lógica de la función expresada como:

$$F(A,B,C) = (\overline{A} \wedge \overline{B}) \vee (A \wedge B \wedge \overline{C})$$

Es de hacer notar que esta función lógica esta formada por dos términos, uno de los cuales solamente toma en cuenta dos de las tres variables de la función en tanto que el segundo incluye las tres variables. El motivo de esta diferencia surge de la simple lectura del enunciado original.

Es de hacer notar asimismo que, en homenaje a la sencillez, la expresión habitual de una función lógica utiliza los símbolos del álgebra de Boole antes que los del álgebra proposicional, lo que convierte a la expresión anterior en:

$$F(A,B,C) = \overline{A}.\overline{B} + A.B.\overline{C}$$

Lo que resulta más claro y simple para su lectura.

4.8. Funciones equivalentes.

Ha quedado planteada en el apartado anterior la representación de funciones lógicas a través de tablas de verdad, que permiten expresar en forma completa la vinculación entre las variables lógicas y las funciones dependientes de aquellas. Una función lógica queda representada en forma unívoca por una tabla de verdad, en la que para cada una de las posibles combinaciones de verdad y falsedad de las variables de la misma se podrá ver si la función es verdadera o falsa.
Una misma tabla de verdad, no obstante, puede representar distintas funciones. Para expresarlo mejor, una misma función puede ser enunciada de distintas formas, y todas ellas, por tratarse de distintas representaciones de la misma función, estarán representadas por la misma tabla de verdad.

*Se definen como **funciones lógicas equivalentes** a aquellas funciones lógicas que responden a una misma tabla de verdad.*

4.8.1. Formas canónicas de una función lógica. Representación de una función lógica mediante suma de productos (primera forma canónica). Términos mínimos.

El concepto enunciado implica que no hay una única representación de una función lógica. Es más, hay una buena cantidad de formas de expresar y representar una función lógica de **n** variables. La representación que surge directamente de cada elemento de una tabla de verdad es una representación de simultaneidad o **conjunción**. En efecto, un uno en una combinación de variables, implica la condición que deben satisfacer simultáneamente las variables de la función para que la misma sea verdadera. Si en esa tabla de verdad existe más de un uno, implicará que existen distintas combinaciones de variables que hacen que la función sea cierta, lo que, a su vez, determinará una condición **disyuntiva**.

Llevada esta definición al lenguaje de las operaciones booleanas, surge que la representación más apropiada para definir una función lógica a partir de su tabla de verdad es la de una **suma de productos** lógicos, donde cada uno de los productos (conjunciones) responde a cada una de las combinaciones de variables que hacen que la función sea verdadera, y donde la suma representa a las diferentes combinaciones que, en su conjunto, hacen que la función sea cierta.

En cada uno de los términos de la suma aparecen, entonces, **todas** las variables que forman la función lógica representada, en su forma cierta o negada, dando origen a la representación de la función en lo que se conoce como **primera forma normal o canónica.**

*Una función se representa en su **primer forma canónica** cuando se la representa a través de una expresión suma de productos, en la que cada producto incluye **todas** las variables que componen la función.*

Para el caso de la función representada en el ejemplo 4.1., la tabla de verdad en él representada permite ver que la misma tiene tres combinaciones de sus variables independientes para la cual se hace cierta. Estas tres combinaciones corresponden a los dos primeros renglones de la tabla de verdad, así como al sexto. El significado de estas tres combinaciones se hace evidente al leer los conceptos involucrados en cada renglón:

- No tengo exámenes para rendir, no tengo trabajo, el período de exámenes no coincide con mi licencia (primer renglón).

- No tengo exámenes para rendir, no tengo trabajo, el período de exámenes sí coincide con mi licencia (segundo renglón)
- Tengo exámenes para rendir, tengo trabajo, el período de exámenes no coincide con mi licencia (sexto renglón).

Resulta interesante releer en este momento la proposición compuesta originalmente citada en el ejemplo. Si se compara aquella expresión con las tres condiciones recién planteadas, surge que se está diciendo lo mismo, pero en este caso, en una forma más compleja. En efecto, si se unen las dos primeras condiciones se puede ver que la coincidencia o no de la licencia con las fechas de exámenes es independiente de la posibilidad de tener vacaciones en caso de no tener exámenes.

Una expresión algebraica de la función, tal como se deduce de la tabla de verdad, será la siguiente:

$$F(A,B,C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.B.\overline{C}$$

La que, se observa, resulta más compleja que la original, aún conservando la misma tabla de verdad. Obsérvese que, según lo expresado, la función adopta la forma de una suma lógica, en la que aparecen términos en cada uno de los cuales, a su vez, aparecen todas las variables de la función, ciertas o negadas. Por razones que resultarán evidentes más adelante, cada uno de los términos de la suma se conoce como **término mínimo** (algunos autores hablan de **minitérminos**) de la función.

*Un **término mínimo** de una función lógica es cada uno de los productos de una función que incluyen a todas las variables de la misma y que la hacen cierta.*

Para una definición algo más simple, un término mínimo es la representación de un uno de la función a partir de todas las variables de la misma. A partir de este concepto se puede definir una función en su primer forma canónica simplemente como una función expresada como suma de términos mínimos.

La cantidad de términos mínimos que puede contener una función no es un numero fijo. De hecho, todo depende de la cantidad de combinaciones de las variables que hagan que esa función sea cierta. El ejemplo 4.1 muestra una función que tiene tres términos mínimos. Lo que sí puede determinarse es el rango dentro del cual puede variar la cantidad de esos términos mínimos. Ese rango está asociado directamente con la cantidad de variables que forman la función.

Una función de n variables presentará una tabla de verdad de 2^n combinaciones posibles. De aquí surge inmediatamente que el número de términos mínimos en una función de n variables puede variar entre ninguno (una función que es siempre falsa) hasta todos (2^n) en el caso de una función siempre cierta.

La tabla siguiente representa los ocho posibles términos mínimos de una función de tres variables A, B, C.

$$\overline{A}.\overline{B}.\overline{C}$$
$$\overline{A}.\overline{B}.C$$
$$\overline{A}.B.\overline{C}$$
$$\overline{A}.B.C$$
$$A.\overline{B}.\overline{C}$$
$$A.\overline{B}.C$$
$$A.B.\overline{C}$$
$$A.B.C$$

Ejemplo 4.2.- Se requiere expresar en su primer forma canónica la función

$$F(A,B,C,D) = A.\overline{B} + C.D + \overline{A}.B.\overline{C}$$

Se observa que la función se encuentra expresada en forma de suma de productos, aun cuando los productos no incluyen todas las variables de la función.

Convertir esta función a su primer forma canónica implica agregar en cada término las variables que no se encuentran en el mismo, de modo de no modificar la expresión original. Esto puede realizarse mediante la aplicación de los postulados del álgebra de Boole, en especial de aquellos l que enuncian las características de los elementos neutro y opuesto:

$$A + \overline{A} = 1$$
$$A.1 = A$$

Obsérvese esta primera utilización binaria de los postulados varias veces repetidos en apartados anteriores mediante el uso de otra simbología.

El uso de la segunda expresión permite aceptar que no se modifica la función si se realiza el producto lógico de cada término por el elemento neutro 1, el que, en función de las variables faltantes, se expresará en cada termino en la forma adecuada.

$$F(A,B,C,D) = A.\overline{B}.(C + \overline{C}).(D + \overline{D}) + C.D.(A + \overline{A}).(B + \overline{B}) + \overline{A}.B.\overline{C}.(D + \overline{D})$$

La aplicación de la propiedad distributiva en aquellos casos en que aparecen productos lógicos, lleva a la siguiente expresión completa de la función original.

$$F(A,B,C,D) = A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.\overline{D} + A.\overline{B}.C.D + \overline{A}.\overline{B}.C.D + \overline{A}.B.C.D +$$
$$+ A.\overline{B}.C.D + A.B.C.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D$$

Se completa el ejemplo con la representación de la mencionada función a través de su tabla de verdad.

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

Obsérvese que al desarrollar la suma canónica de productos ha aparecido algún termino representado en mas de una ocasión. Estos términos repetidos dejaran de ser tales por la simple aplicación del teorema de unicidad del álgebra de Boole, enunciado oportunamente.

4.8.2. Representación de una función lógica mediante producto de sumas (segunda forma canónica). Términos máximos.

A partir del hecho de que una tabla de verdad permite ver los ceros y los unos de la función (esto es, las situaciones en que dicha función es falsa o verdadera), se puede plantear que en una tabla de verdad no solamente se define la función expresada por la misma, sino que, simultáneamente, se define también su función opuesta (**negada**). En efecto, todas aquellas combinaciones de la función representada en las que la misma adopte el valor **cero** serán aquellas en que la función opuesta o negada adopte el valor **uno**. Desde este punto de vista, la misma tabla de verdad a utilizar para representar una función como suma de productos permitirá también representar como suma de productos su función negada. Dado que la ley de doble negación vista en apartados anteriores permite obtener una función a partir de su opuesta, surge inmediatamente que la negación de la función opuesta y la subsiguiente aplicación de las leyes de De Morgan a la expresión obtenida entrega como resultado una nueva expresión de la función lógica original, en este caso representada como un **producto de sumas**, en el que cada factor del producto (cada una de las sumas) incluye como sumandos a todas las variables de la función, ciertas o negadas. Una función expresada de esta manera se dice representada en su **segunda forma normal o canónica**.

*Una función se representa en su **segunda forma canónica** cuando se la representa a través de una expresión producto de sumas, en la que cada factor incluye **todas** las variables que componen la función.*

Al igual que en el caso de la primer forma canónica, surge un nombre particular para cada uno de los factores que constituyen esta segunda forma. Cada factor, por razones que también se harán evidentes más adelante, se conoce como un **término máximo** (o **maxitérmino**) de la función representada. Se define entonces un término máximo de una función como cada una de las sumas de todas las variables de la función que hacen que la misma sea falsa (Nótese el carácter dual de ambas definiciones). De esta manera, una función expresada en su segunda forma canónica se encuentra expresada como producto de términos máximos. Para que la función, expresada de esta forma, sea cierta deberán ser ciertos simultáneamente todos los factores. Uno de ellos que no resulte cierto implicará que el producto, y por ende la función, sea falsa. Es por eso que se suele decir que la segunda forma normal representa la función a través de sus ceros.

Volviendo al ejemplo 4.2, queda claro que para aquellas combinaciones de entradas para las que la función F(A,B,C,D) es falsa, será cierta su función opuesta (negada).

Si se lee dicha función a partir de las combinaciones de la tabla de verdad que valen cero para la función original, se tendrá la siguiente tabla de verdad:

A B C D	*F(A,B,C,D)*	*~F(A,B,C,D)*
0 0 0 0	*0*	*1*
0 0 0 1	*0*	*1*
0 0 1 0	*0*	*1*
0 0 1 1	*1*	*0*
0 1 0 0	*1*	*0*
0 1 0 1	*1*	*0*
0 1 1 0	*0*	*1*
0 1 1 1	*1*	*0*
1 0 0 0	*1*	*0*
1 0 0 1	*1*	*0*
1 0 1 0	*1*	*0*
1 0 1 1	*1*	*0*
1 1 0 0	*0*	*1*
1 1 0 1	*0*	*1*
1 1 1 0	*0*	*1*
1 1 1 1	*1*	*0*

Lo que permite expresar la función opuesta como suma de términos mínimos:

$$\overline{F}(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.B.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.B.\overline{C}.\overline{D}$$

La aplicación del teorema de doble negación permitirá representar la función F(A,B,C,D) en la forma siguiente:

$$F(A,B,C,D) = \overline{\overline{F}} = (\overline{\overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.B.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.B.\overline{C}.\overline{D}})$$

La aplicación posterior de las leyes de De Morgan permitirán, en dos pasos, llegar a la expresión correspondiente a F(A,B,C,D) en la forma de un producto de sumas canónico.

$$F(A,B,C,D) = (A+B+C+D).(A+B+C+\overline{D}).(A+B+\overline{C}+D).(A+\overline{B}+\overline{C}+D).(\overline{A}+\overline{B}+C+D).$$
$$(\overline{A}+\overline{B}+C+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+D)$$

Expresión que define a la función ejemplificada en su segunda forma canónica.

4.8.3. Lógica de dos niveles.

Ambas representaciones canónicas desarrolladas utilizan solamente dos niveles de operaciones lógicas para la expresión buscada, por lo que en forma genérica se las conoce como **representaciones en dos niveles lógicos**. Esto significa, para decirlo en

forma simple, que, desde la variable independiente hasta la función final, nunca se encontrarán más de dos operadores lógicos. Esta definición es válida si no se considera la negación como un operador lógico. En el capítulo correspondiente se analizarán las ventajas derivadas de la representación en dos niveles por sobre otros tipos de implementaciones.

INTRODUCCIÓN A LOS SISTEMAS DIGITALES

ING. FERNANDO IGNACIO SZKLANNY

CAPÍTULO V:

CIRCUITOS COMBINATORIOS

CAPÍTULO 5: CIRCUITOS COMBINATORIOS

5.1.- De las funciones lógicas a la lógica electrónica.

Los conceptos lógicos desarrollados en el capítulo anterior pueden convertirse en elementos de aplicación práctica, a través de su implementación mediante elementos circuitales basados en los conceptos que se desarrollarán más adelante. En efecto, la utilización del transistor como elemento conmutador es el camino más fácil hacia la construcción de circuitos binarios, los que podrán basarse en los conceptos del álgebra booleana, concretando así el concepto de circuito lógico como denominación de aquellos circuitos que permitan desarrollar funciones lógicas, en el sentido ya desarrollado, y llevar a la práctica su comportamiento.

Un **circuito lógico** es una construcción circuital (no necesariamente electrónica) que permite representar funciones lógicas a través de elementos físicos que tienen un comportamiento binario. Esto significa que a través del estado de ciertos elementos pueden representarse los dos estados lógicos (en este caso tomado el concepto como sinónimo de binario) de una variable lógica, teniendo a la vez otros elementos cuyos estados representarán a las correspondientes funciones o variables compuestas.

Desde este punto de vista, y sin entrar en detalles constructivos, se puede visualizar un circuito lógico como un conjunto de elementos representativos de variables lógicas independientes y de las funciones lógicas que de ellas dependen, y en el cual dicha dependencia se representa a través de las magnitudes físicas que en el circuito vinculan a dichos elementos. La implementación más sencilla, desde el punto de vista eléctrico, permite representar una variable independiente con una llave o interruptor, cuyos dos estados (abierto y cerrado) representan los estados lógicos de 0 y 1 de una variable independiente. En el mismo sentido, los dos estados lógicos de una función lógica podrán representarse, por ejemplo, con los dos estados de encendido y apagado de una lámpara o indicador de cualquier tipo, el que, conectado con dicha llave, formará un circuito eléctrico como el de la Figura 5.1. Las funciones lógicas que se puedan construir con llaves y lámparas se representarán a través de las magnitudes eléctricas de tensión y corriente, para lo cual se requerirá completar el circuito con el elemento generador de tensión que corresponda.

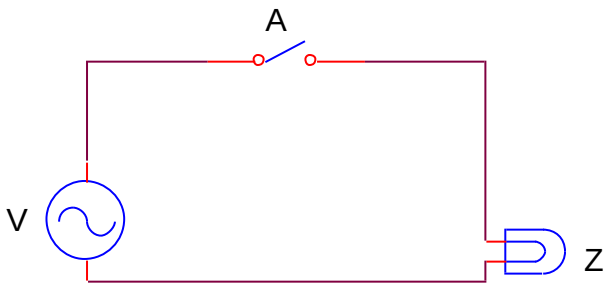


Figura 5.1. - Circuito con una llave y una lámpara

En el circuito de la figura 5.1, se representa una función muy simple de una única variable, en la que el estado de la lámpara (encendido o apagado) depende en forma directa del estado de la llave (cerrado o abierto). Desde el punto de vista eléctrico, el concepto es claro:

La lámpara está encendida siempre que la llave esté cerrada.

La interpretación lógica del funcionamiento del circuito mencionado, sin embargo, ofrece alternativas. En efecto, en este circuito coexisten una proposición simple y una proposición compuesta que pueden enunciarse de distintas formas. Para el caso de la llave, las dos posibles proposiciones pueden ser:

A = La llave está abierta A = La llave está cerrada.

Para el caso de la lámpara indicadora, las dos proposiciones involucradas son

Z = La lámpara está encendida Z = La lámpara está apagada.

En ambos casos, tanto en el caso de la lámpara como en el de la llave, las dos posibles sentencias involucradas son opuestas entre sí. Esto permite enunciar la función lógica anterior de varias maneras distintas, todas equivalentes:

La lámpara está apagada siempre que la llave esté abierta.

La lámpara está apagada siempre que la llave no esté cerrada.

La lámpara está encendida siempre que la llave no esté abierta.

además de las otras que pudiesen enunciarse por la negativa de cada una de estas cuatro.

La expresión lógica representada por el circuito, en consecuencia, dependerá de una convención, que deberá adoptarse para definir cual de las proposiciones se utiliza, en cada caso, para ser representada por el circuito considerado. Si se adoptan como proposiciones representadas las proposiciones *A = La llave está cerrada* y *Z = La lámpara está encendida*, la función representada por el circuito será Z = A. En cambio, si se adopta, por ejemplo*: Z = La lámpara está apagada*, la función representada será Z = no A. Nótese que no cambia el circuito, sino solamente la selección de las proposiciones representadas por cada uno de los estados de llave e indicador, para obtener una función lógica distinta.

5.2.- Representación física de los operadores lógicos. Compuertas. Compuertas básicas Y, O, NO.

El circuito de la figura 5.1 define la vinculación básica entre una variable independiente (representada por la llave o interruptor) y una variable dependiente o función, representada a su vez por la lámpara o indicador. Las funciones lógicas presentadas en capítulos anteriores tienen que ver con diversas variables lógicas, vinculadas entre sí por diferentes operadores lógicos, a partir de los cuales se obtendrá la función lógica requerida. Surge entonces como necesidad la de definir elementos que permitan representar las operaciones lógicas enunciadas en aquellos capítulos, con lo que se logrará la representación de diferentes proposiciones compuestas.

Se definirá como **compuerta lógica** a todo aquel circuito que permita representar un operador lógico. Esta definición implica la existencia de distintos tipos de circuitos o compuertas para representar, como mínimo, las dos operaciones básicas del álgebra de Boole (suma y producto lógicos). Estas compuertas se pueden implementar, desde el punto de vista eléctrico, en forma sencilla a partir del circuito de la figura 5.1, analizando los conceptos vinculados con dichas operaciones lógicas y determinando su relación con los elementos representativos utilizados en aquel.

El circuito de la figura 5.2 muestra una de dichas operaciones lógicas implementadas sobre la misma base de interruptores e indicadores. En este circuito sólo se tendrá la lámpara encendida si ambos interruptores están cerrados en forma simultánea. Cualquier otra combinación de llaves hará que la lámpara esté apagada.

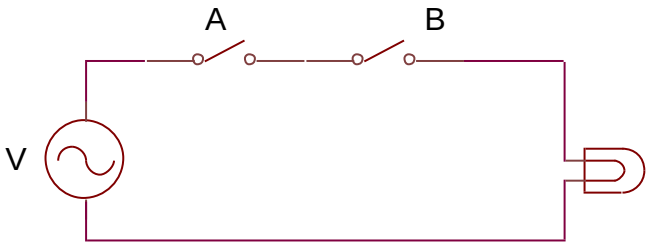


Figura 5.2.- Un circuito lógico Y

Si se aceptan como válidas las proposiciones:

A = El interruptor está cerrado.
B = El interruptor está cerrado.
Z = La lámpara está encendida.

surgirá claramente que el circuito de la figura muestra una operación de producto lógico, representada por la expresión

$$Z(A,B) = A \cdot B$$

En efecto, de acuerdo con lo dicho las dos llaves deberán estar cerradas ($A=V \wedge B=V$) para que la lámpara esté encendida ($Z=V$). Cualquier otra combinación de llaves hará que la lámpara esté apagada ($Z=F$)

Planteada como tabla de verdad, la actuación del circuito resultará en la siguiente:

A	**B**	**Z = A.B**
F	F	F
F	V	F
V	F	F
V	V	V

Análogamente, el circuito de la figura 5.3 representa otra alternativa circuital, en la que bastará que una de las dos llaves interruptoras esté cerrada (una, otra o ambas), para que la lámpara ilumine. Solamente en el caso en que ambas llaves estén abiertas, el indicador luminoso permanecerá apagado.

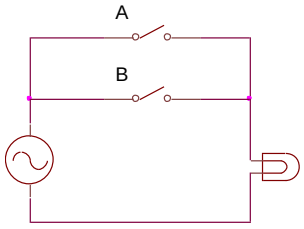


Figura 5.3.- Un circuito lógico O

Si se aceptan los criterios del ejemplo anterior, surge claramente que el circuito eléctrico, tomado como elemento lógico, representa una función O, cuya tabla, ya conocida, es la siguiente:

A	B	Z=A+B
F	F	F
F	V	V
V	F	V
V	V	V

El concepto de estos dos ejemplos puede ampliarse a cualquier número de variables de entrada, cada una de las cuales deberá representarse a través de un interruptor que, cerrado, representará la variable en su estado de certeza, en tanto que abierto la representará en su estado de falsedad.

Un tercer tipo de circuito lógico es el que permite representar el opuesto de una variable lógica cualquiera. Este tipo de circuito, conocido como circuito inversor o negador, entregará a su salida un cero lógico cuando la variable de entrada valga uno, y entregará un uno cuando la variable lógica valga cero.

A partir de estos esquemas circuitales, además, puede implementarse cualquier función lógica de dos niveles, sea suma de productos o producto de sumas, a través de circuitos constituidos con llaves combinadas de forma de implementar las funciones de suma y producto lógico.

Así como se plantea la posibilidad de implementar compuertas con llaves, mediante circuitos eléctricos, puede lograrse la implementación circuital de compuertas mediante otras técnicas. En el capítulo correspondiente se analizan las estructuras electrónicas que permiten la construcción de compuertas y circuitos lógicos más complejos mediante la utilización de distintos tipos de transistores y otros componentes electrónicos. Estos circuitos darán origen a lo que se suele llamar **lógica electrónica**. Será esta la tecnología utilizada a lo largo de este texto como herramienta para la descripción y construcción de circuitos lógicos. No obstante, se pueden obtener implementaciones de circuitos lógicos (y por ende de compuertas) utilizando técnicas de tipo hidráulico, neumático, etc.

Para permitir uniformar la representación de circuitos lógicos, sin importar el tipo de tecnología a utilizar, a partir de ahora se los identificará mediante símbolos convencionales, los que se presentan en la figura 5.4. En la misma aparecen dos tipos de símbolos correspondientes a las compuertas representativas de las operaciones lógicas básicas (Y, O, NO) que se definieron en el capítulo 4. Para cada una de las compuertas se han representado los símbolos tradicionalmente utilizados para su identificación, así como la simbología propuesta por la norma IEEE/ANSI 91-1984.

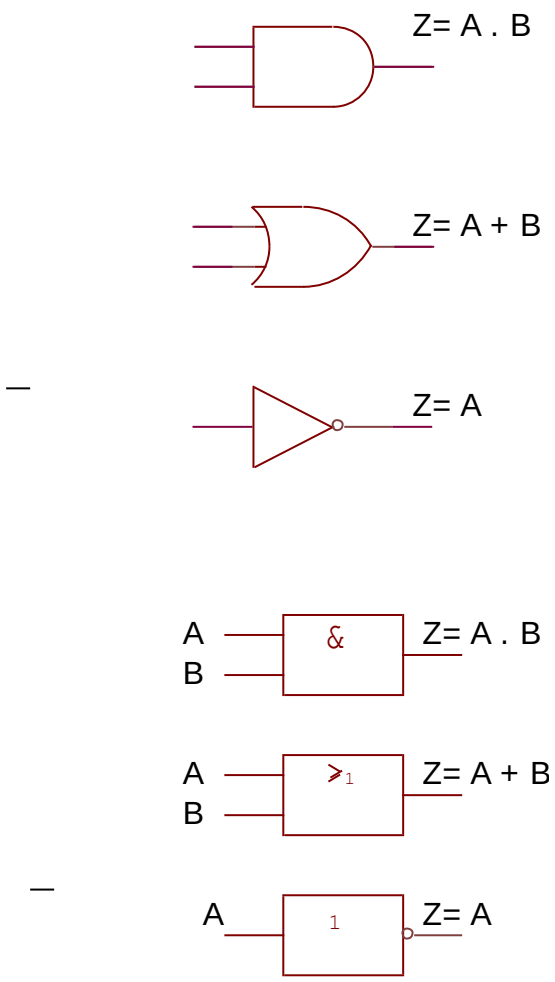


Figura 5.4.- Símbolos de las compuertas lógicas. Dos convenciones

5.3.- Convenciones lógicas.

La representación simbólica de las compuertas refleja su carácter de circuito eléctrico o electrónico. En el mismo cada terminal de entrada representa una dada variable lógica. La salida será, por lo tanto, función de todas las entradas que llegan a la compuerta, representando el nivel de la misma los valores de verdad y falsedad de la función lógica implementada por la compuerta. Al interconectar compuertas para obtener funciones más complejas se podrá evaluar la tabla de verdad de la función lógica a través de los niveles de salida del circuito implementado.

Este planteo significa que existe una relación entre los valores de verdad y falsedad de una función lógica y los respectivos valores de las magnitudes eléctricas (tensión o corriente) que se tienen en los diversos puntos de un circuito lógico. Dado que las compuertas son una representación física de una función lógica, lo primero que deberá establecerse es una convención que relacione los valores de certeza (1) y falsedad (0) de la función lógica representada con alguna de las magnitudes físicas involucradas. La magnitud típicamente utilizada es la tensión en cada uno de los terminales de entrada y salida. Como ya se ha visto en el capítulo correspondiente, al ser los elementos circuitales con que se trabaja de naturaleza

binaria, los mismos admitirán a su entrada, y entregarán en su salida dos posibles niveles de tensión. [1]

A partir de esta consideración se pueden establecer dos posibles convenciones que permitan asociar los estados lógicos de una función lógica y sus variables con los niveles de tensión utilizados en el circuito físico para representar dicha función. Se define como **lógica positiva** (Figura 5.5a) a la representación en la cual el estado lógico de verdad (1) se representa por el más alto de los niveles de tensión, en tanto que el cero (falso) se representa por el menor de los dos niveles de tensión. Se define en cambio como **lógica negativa** a aquella representación que asigna al cero el mayor valor de tensión y al uno el menor valor de tensión en el terminal correspondiente (Figura 5.5b). No importan los valores absolutos de tensión utilizados; solo importa cual de los dos valores existentes en el circuito se asignan respectivamente al cero y al uno.

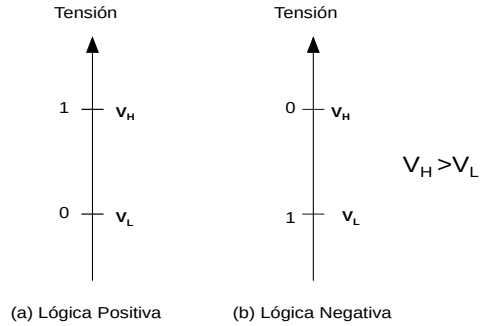


Figura 5.5.- Lógica positiva y negativa

En el transcurso de este texto, como se hace en la mayoría de las aplicaciones, se utilizará la convención de lógica positiva, con lo que se asociará directamente el mayor valor de tensión en un terminal de entrada o salida con una variable cierta, y el menor valor de esa tensión con la variable falsa (o negada).

5.4.- Implementación de funciones lógicas mediante la utilización de compuertas.

Una función, expresada en su primer forma canónica, o simplificada a partir de aquella, se representa en la forma conocida como de **suma de productos**. La estructura lógica de un circuito que represente dicha forma requerirá de un primer nivel de compuertas Y, que recibirán las entradas de la función. Las salidas de las compuertas Y se llevaran a las entradas de una compuerta O, cuya salida representará la función lógica implementada.

Si la función se representa en su segunda forma canónica, o a través de una simplificación por sus ceros, se obtendrá una estructura **producto de sumas**, lo que requerirá de un primer nivel de compuertas O cuyas salidas se conecten todas a una única compuerta Y, la que entregará la salida de la función lógica.

En ambos casos se trata de implementaciones en lógica de dos niveles, concepto que ya ha sido definido en el capítulo 4, y cuya aplicación se verá en el siguiente ejemplo.

[1] Al analizar posteriormente las tecnologías de circuitos integrados se verá que en realidad cada nivel de tensión es una banda, limitada por un valor máximo y un valor mínimo para cada nivel lógico.

Ejemplo 5.1.- Implementar la función lógica expresada como

$F(A,B,C,D) = \Sigma m\ (0,1,3,4,8,9,11,12)$

Se analizarán distintas implementaciones de esta función, sea a partir de sus dos formas canónicas, sin simplificar, o a partir de las funciones mínimas en sus dos formas suma de productos y producto de sumas.

Para la implementación de la función a partir de su primer forma canónica, se expresa directamente la función como suma de términos mínimos.

La expresión correspondiente es la siguiente:

$$F(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.B.\overline{C}.\overline{D}$$

Por consiguiente, se requerirán ocho compuertas, de cuatro entradas cada una, y una compuerta O de ocho entradas para permitir la implementación de la función mencionada. El circuito correspondiente se muestra en la figura 5.6.

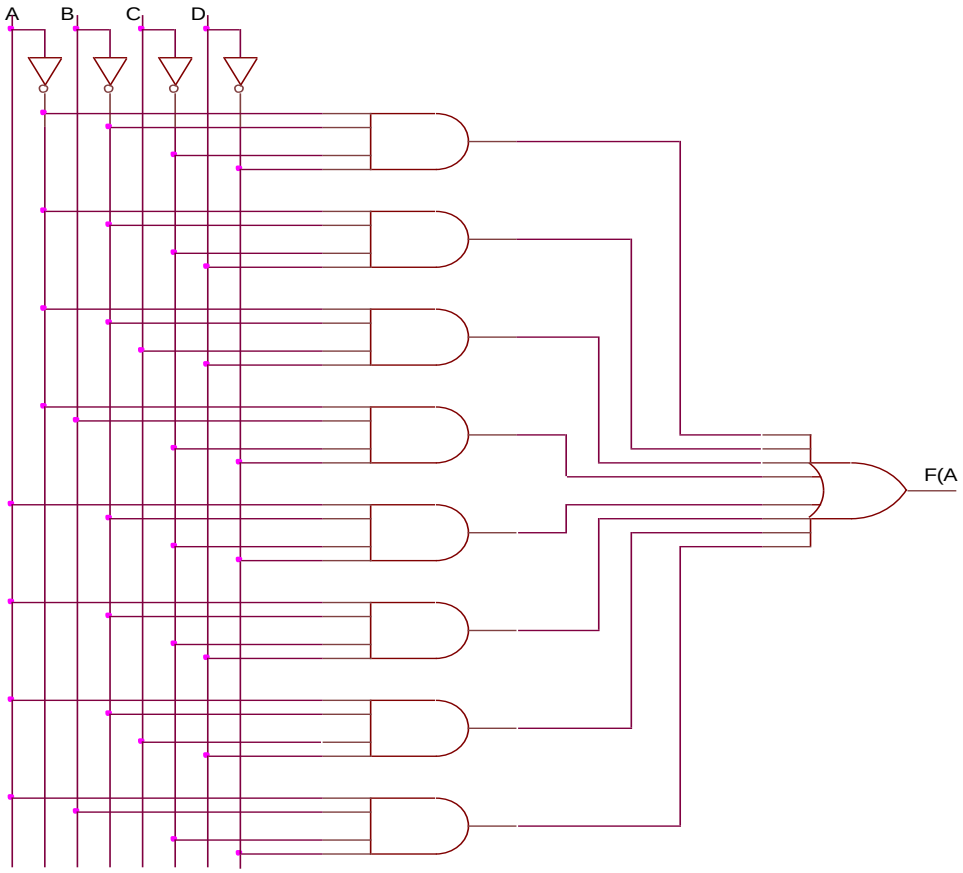


figura 5.6.- Implementación de la función en su primera forma canónica.

Si se procede a la simplificación de la función antes de su implementación, se obtendrá la siguiente función lógica,

$$F(A,B,C,D) = \overline{C}.\overline{D} + \overline{B}.D$$

Como se observa en este caso, la minimización de la función requerirá de solo dos compuertas Y de dos entradas y una compuerta O, también de dos entradas, lo que da lugar a una lógica mucho más simple y por ende más económica. La misma se muestra en la figura 5.7.

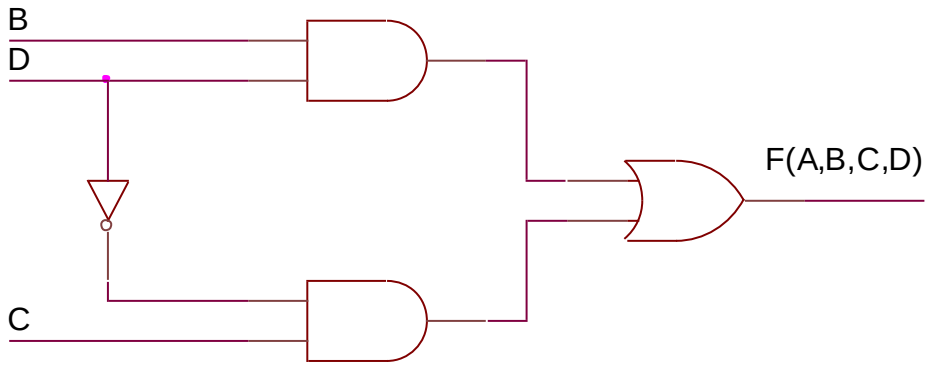


Figura 5.7.- La misma función, simplificada como suma de productos.

Si se pretende implementar la función a partir de sus ceros, la tabla de verdad de la misma muestra ocho combinaciones de las entradas para las cuales la función vale cero. En este caso, la implementación circuital resulta de una complejidad circuital similar a las anteriores, por cuanto se requieren ocho compuertas O de cuatro entradas cada una y una compuerta Y de ocho entradas, según se aprecia en la figura 5.8. La ecuación correspondiente a la función implementada es la siguiente:

$$f(A,B,C,D) = (A+B+\overline{C}+D)(A+\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)(\overline{A}+B+C+D).$$
$$(\overline{A}+B+\overline{C}+D)(\overline{A}+B+\overline{C}+\overline{D})(\overline{A}+\overline{B}+C+\overline{D})(\overline{A}+\overline{B}+\overline{C}+\overline{D})$$

Por lo que el circuito correspondiente será el de la Figura 5.8.

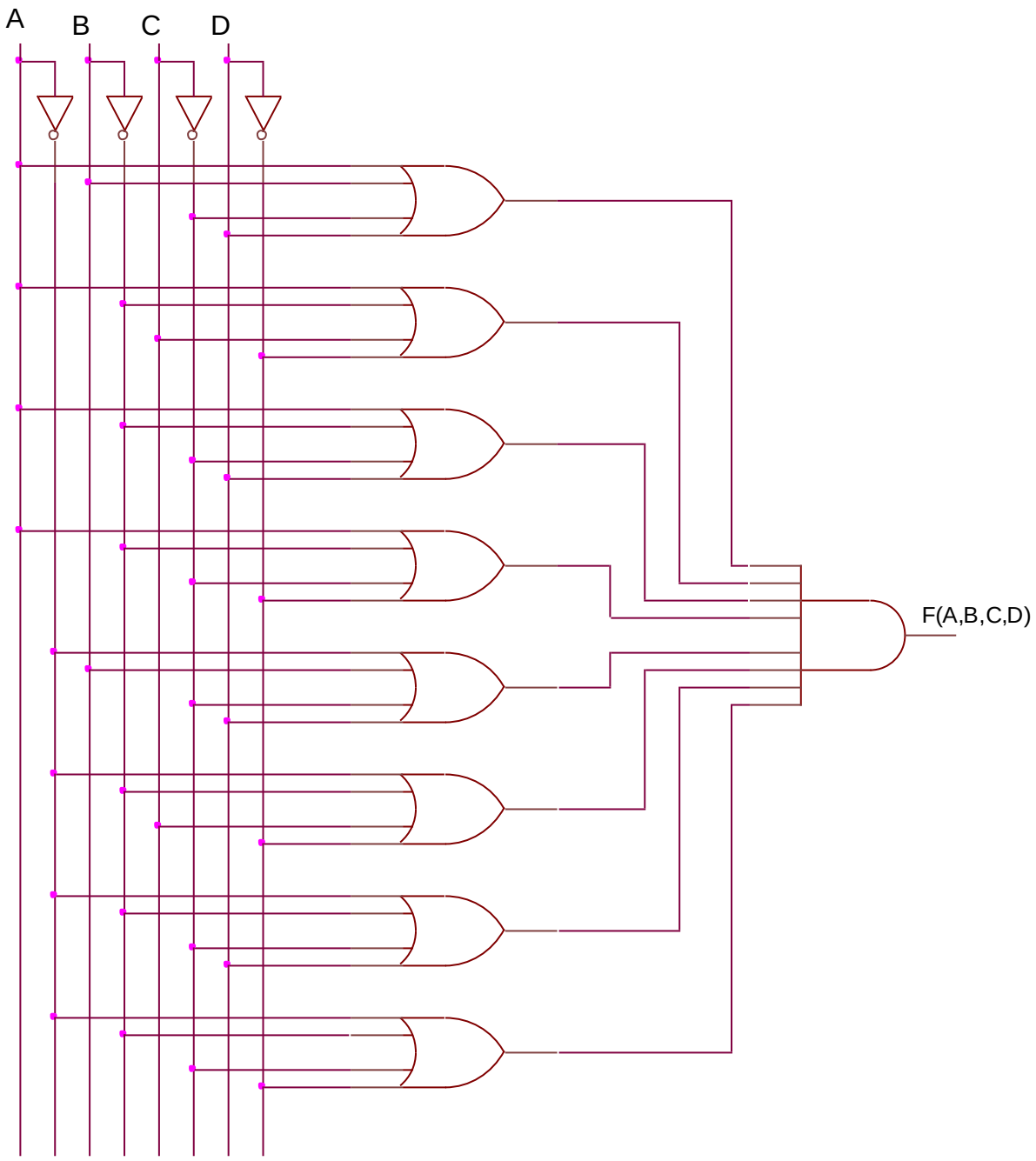


Figura 5.8.- La misma función, en su segunda forma canónica.

No obstante, si se simplifica la función así expresada, se obtiene un producto de sumas mínimo que, también en este caso, resultará en una expresión mucho más sencilla que la forma canónica planteada anteriormente:

$$F(A,B,C,D) = (\overline{B} + \overline{D}).(\overline{C} + D)$$

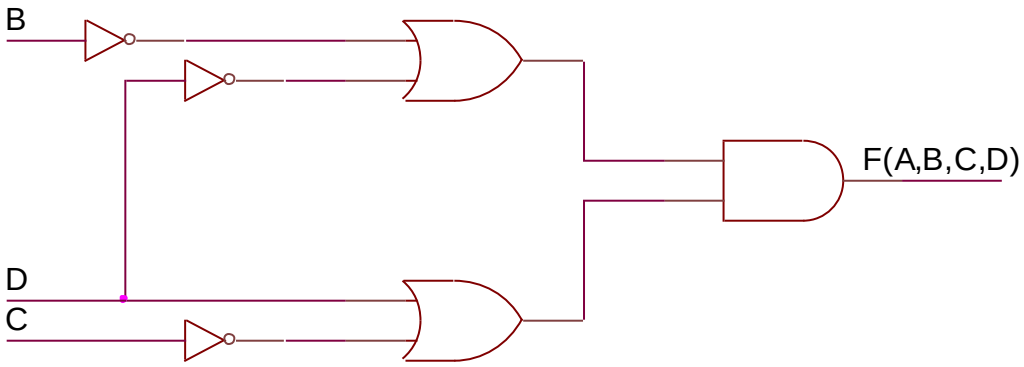


Figura 5.9.- Cuarta alternativa, la función simplificada como producto de sumas.

5.5.- Lógica de dos niveles.

La implementación de funciones lógicas mediante circuitos de dos niveles, ya sean de compuertas Y-O, O-Y, NAND o NOR, tiene, en muchos casos, una ventaja fundamental sobre otras implementaciones con compuertas. Si bien los conceptos asociados con las características funcionales y los parámetros físicos de las compuertas se analizarán en capítulos posteriores, puede decirse en este punto que al ser las compuertas elementos físicos, requerirán para su funcionamiento un determinado tiempo, en el que reaccionarán a los cambios de sus entradas. Ese tiempo, llamado "tiempo de propagación" de una compuerta, mide la velocidad de reacción de la compuerta a un cambio en alguna entrada. Es evidente que cuantas más compuertas se interpongan en el camino entre una entrada de un circuito lógico y la salida del mismo, más tiempo le llevará a la salida del circuito reaccionar al cambio de dicha entrada. Es por eso que, ante diferentes alternativas de diseño de un circuito con compuertas, la implementación en dos niveles es, salvo casos muy particulares, la más ventajosa si se trata de obtener un circuito que responda a los cambios de sus entradas a la mayor velocidad posible.

5.6.- Compuertas NAND y NOR. Sus tablas de verdad.

De la combinación de las operaciones básicas del Álgebra de Boole pueden obtenerse nuevas funciones lógicas que permitan ampliar las posibles herramientas circuitales a utilizar en la implementación de circuitos lógicos. La aplicación básica de las leyes de De Morgan permite crear, sobre la base de las funciones básicas Y y O, dos nuevas expresiones lógicas, cuyas tablas de verdad se presentan a continuación, y a las que se denomina habitualmente NAND (por NOT AND, negación de la función AND) y NOR (por NOT OR, negación de la función OR).

A	B	Z=A nor B	Z=A nand B
F	F	V	V
F	V	F	V
V	F	F	V
V	V	F	F

Estas compuertas tienen una ventaja interesante en lo que hace a su utilización en la implementación de circuitos, dado que, según se verá, permiten implementar con sencillez funciones lógicas con dos niveles de compuertas utilizando un único tipo de compuertas aun en aquellos casos en que la lógica de dos niveles requiere compuertas Y y O para la

implementación de las funciones en sus formas de suma de productos o producto de sumas. En los apartados siguientes se muestran los símbolos de las compuertas mencionadas, indicándose también la metodología de utilización de las mismas en la implementación de las funciones básicas del álgebra de Boole.

5.6.1.- Implementación de las operaciones básicas mediante el uso de compuertas NAND.

Dado que una compuerta NAND representa la expresión negada de una función Y, surge inmediatamente, según la ley de doble negación, que puede obtenerse una compuerta Y negando la salida de una compuerta NAND, según se aprecia en la figura 5.10.

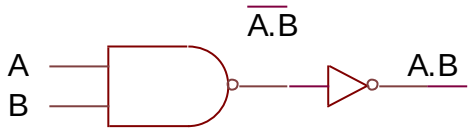


Figura 5.10 – Una compuerta Y utilizando NAND e inversores.

De la misma manera, y a partir de las leyes de De Morgan, puede verse que la expresión de salida de una compuerta NAND equivale a una función lógica O, de acuerdo con lo que se muestra en la figura 5.11.
Figura 11

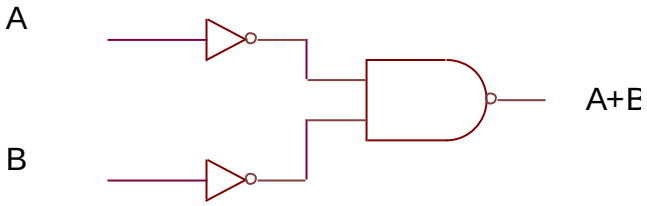


Figura 5.11.- Una compuerta O utilizando NAND e inversores.

En ambas representaciones puede utilizarse una compuerta NAND para la implementación de las compuertas inversoras. Dado que a partir de la tabla de verdad de la misma surge que cuando las entradas de la compuerta NAND adoptan el mismo valor lógico, la salida entrega el valor opuesto, queda claro que puede obtenerse una compuerta inversora con solo unir entre sí todas las entradas de la compuerta NAND.

5.6.2.- Implementación de las operaciones básicas mediante compuertas NOR.

Con el mismo criterio anterior, las compuertas Y y O pueden implementarse mediante la utilización de compuertas NOR. Dado que una compuerta NOR representa la expresión negada de una función O, surge inmediatamente, según la ley de doble negación, que puede obtenerse una compuerta O negando la salida de una compuerta NOR, según se aprecia en la figura 5.12.

Figura 5.12 – Una compuerta O utilizando compuertas NOR e inversores.

De la misma manera, y a partir de las leyes de De Morgan, puede verse que la expresión de salida de una compuerta NOR equivale a una función lógica Y, de acuerdo con lo que se muestra en la figura 5.13.
Figura 13

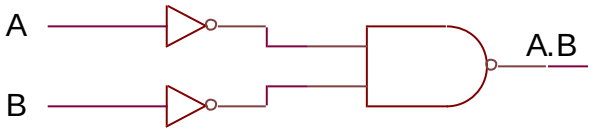


Figura 5.13.- Una compuerta Y utilizando compuertas NOR e inversores.

Por supuesto, y tal como se ha planteado en el caso de las compuertas NAND, también puede utilizarse una compuerta NOR para la implementación de compuertas inversoras. Ya que a partir de la tabla de verdad de la misma surge que cuando las entradas de la compuerta NOR adoptan el mismo valor lógico, la salida entrega el valor opuesto, queda claro que puede obtenerse una compuerta inversora con solo unir entre sí todas las entradas de una compuerta NOR.

5.7.- Implementación de funciones lógicas con un único tipo de compuertas.

De acuerdo con lo visto, una función como la del ejemplo 5.1 puede implementarse, sea en su forma canónica o simplificada, con dos niveles de compuertas NAND, de acuerdo con lo que se observa en la figura 5.14, en la que puede observarse que el primer nivel de compuertas NAND reemplaza al nivel de compuertas Y, en tanto que la compuerta NAND de salida reemplaza a la compuerta Y de salida de las figuras anteriores. Se observa en la misma figura que los inversores asociados con las compuertas Y y O quedan cancelados por estar en secuencia según la ley de doble negación.

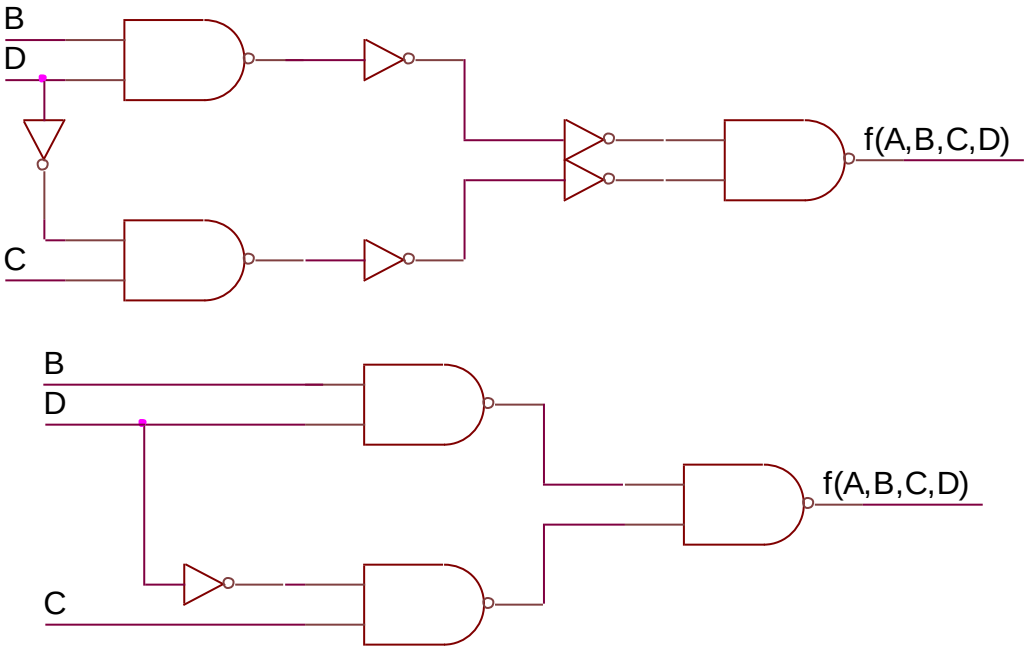


Figura 5.14. Implementación NAND- NAND de la función del ejemplo 5.1.

De la misma manera, si se quiere plantear la implementación a partir de la segunda forma canónica, podrá llegarse a una implementación con dos niveles de compuertas NOR, tal como se observa en la figura 5.15.

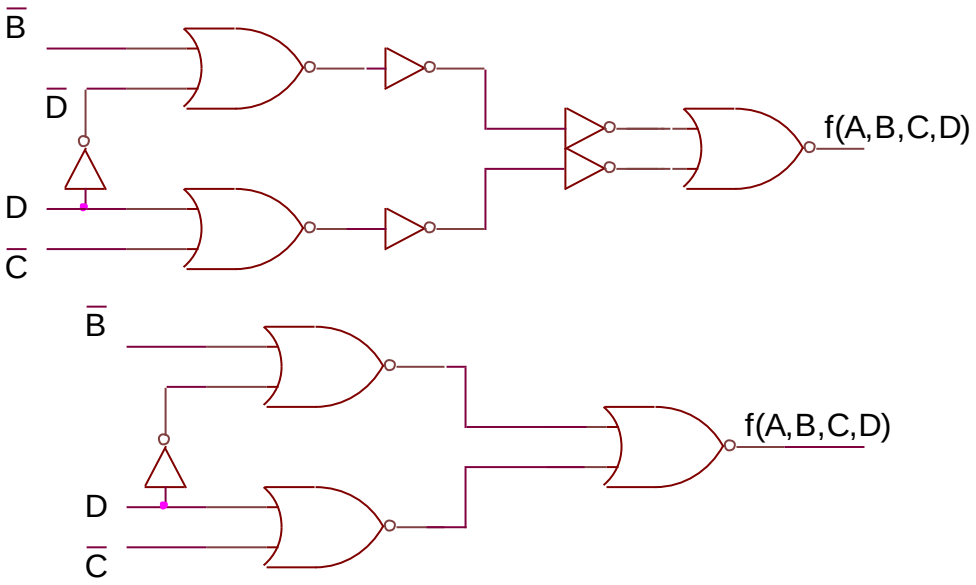


Figura 5.15.- Implementación NOR-NOR de la misma función.

Cabe aquí establecer una regla práctica que permita decidir qué tipo de compuertas deben seleccionarse. Esta regla dice que:

Una función definida como suma de productos tiene su mejor implementación cuando se la implementa con dos niveles de compuertas NAND, como alternativa a la implementación Y-O.

En tanto que una función expresada como producto de sumas tiene su mejor implementación en dos niveles de compuertas NOR, como alternativa a la implementación O-Y.

5.8.- Circuitos combinatorios.

Al establecer la relación entre la lógica electrónica y el álgebra proposicional, hemos definido una compuerta como cualquier circuito que permita representar físicamente una operación booleana. Desde este punto de vista, una función lógica se puede representar por medio de un conjunto de compuertas interconectadas de forma tal que se cumpla la tabla de verdad a la que responde dicha función lógica. Ese conjunto de compuertas determina lo que genéricamente se conoce como un circuito lógico. El mismo mostrará en sus salidas, para cada combinación de unos y ceros en sus entradas, el valor lógico de cero o uno que indique la tabla de verdad. Debido a esta razón, un circuito lógico que representa una tabla de verdad se define como un circuito combinatorio. En este tipo de circuito, el valor de las salidas del mismo, en cada instante de la vida del circuito, sólo depende de los valores que adopten las entradas en ese mismo momento de tiempo. La representación general de un circuito combinatorio es la que se muestra en la Figura 5.16.

Dado que en cada momento del circuito sus salidas Z_j dependerán exclusivamente de sus entradas X_i podrá expresarse el funcionamiento del circuito mediante la relación $Z_j = f(X_i)$ para todo momento t^n, donde t^n representa un momento determinado cualquiera en la vida del circuito.

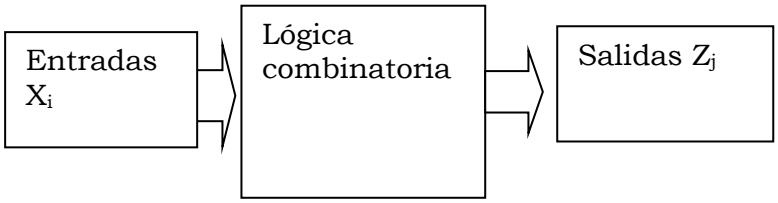


Figura 5.16.- Representación general de un circuito combinatorio.

Se define como circuito combinatorio a aquel circuito lógico cuyas salidas, en cada momento, responden exclusivamente al valor de sus entradas en ese mismo momento.

Cabe acotar que, como se podrá ver en capítulos posteriores de esta obra, no todos los circuitos lógicos cumplen con este requisito. De hecho, en los capítulos 7 y posteriores se analizará un conjunto de circuitos, denominados circuitos secuenciales, cuyas características de funcionamiento son totalmente diferentes a las de los circuitos combinatorios en análisis.

Para la implementación de un circuito combinatorio, se deben plantear una serie de pasos que permitan obtener dicho circuito a partir de los requerimientos funcionales que se pretenden para el mismo. Estos pasos incluyen, como primero, el planteo de la tabla de verdad a la que debe responder el circuito. La tabla así obtenida debe expresarse en la forma de una función lógica, la que podrá estar dada en alguna de sus formas canónicas o, preferentemente, deberá estar simplificada para poder representar la expresión mínima de la función lógica. Una vez obtenida esa función lógica, corresponderá elegir los elementos para su implementación, los que, a esta altura de los conocimientos adquiridos, no podrán ser otra cosa que distintos tipos de compuertas que permitan la implementación de la función en dos niveles de compuertas.

En los apartados futuros, se irán desarrollando ejemplos de aplicación en los cuales se aplicaran, en la medida de lo necesario, los métodos de simplificación de funciones lógicas vistos en los apartados anteriores. Los circuitos correspondientes se implementarán en cada caso en la forma que resulte más conveniente para cada aplicación.

Estos circuitos, de diferente utilidad dentro de los sistemas digitales, pueden implementarse en diferentes formas, tanto mediante la utilización de compuertas como a través de dispositivos electrónicos (bloques funcionales) más complejos cuyas descripciones y utilización se verán a lo largo de capítulos posteriores.

Dentro de estas aplicaciones se ejemplificarán en el presente apartado aquellos circuitos que tienen que ver con los conceptos desarrollados en los capítulos anteriores, referidos a operaciones aritméticas, representación de información mediante códigos binarios, etc. Como un último ejemplo, se desarrollarán los circuitos que se conocen como multiplexores o selectores, una de cuyas aplicaciones importantes es la utilización de los mismos como bloques funcionales complejos tendientes a implementar circuitos combinatorios de diferente complejidad.

5.9.- Circuitos aritméticos.

Los circuitos que realizan operaciones aritméticas tienen una gran importancia en las estructuras de las computadoras, de las calculadoras electrónicas, y, en general, de los circuitos y sistemas digitales. El cálculo aritmético debe implementarse dentro de dichos sistemas como una herramienta fundamental para otras operaciones, por lo que se hacen necesarios aquellos circuitos que permitan la realización, al menos, de las operaciones básicas. Estos circuitos

pueden ser tan simples como un circuito que sume dos números binarios en una representación de punto fijo, o tan complejos como aquellos que permitan realizar operaciones con valores representados en punto flotante.

5.9.1.- Circuitos sumadores para números de un bit.

Para comenzar con un ejemplo sencillo, se planteará el diseño de un circuito que permita la suma **aritmética** de dos números de un bit.

En este momento de la tecnología, en que los sistemas de cómputo suman palabras de gran cantidad de bits en forma paralela (simultanea), no parece resultar del todo evidente la utilidad de este circuito. No obstante, la sencillez del mismo permite que se lo utilice como ejemplo básico de un circuito combinatorio, el que se irá complicando a lo largo de la ejemplificación hasta convertirlo en la base de implementación de un sumador de números de mayor cantidad de bits.

El circuito sumador más sencillo de implementar es aquel que permite sumar dos variables binarias, consideradas como números de un bit cada uno. En este caso, y tal como se ha visto en el capitulo correspondiente, las alternativas posibles para la suma son cuatro:

0 + 0 = 0 0 + 1 = 1 + 0 = 1 1 + 1 = 10

La tabla de verdad correspondiente a estas expresiones es la siguiente:

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

En esta tabla de verdad, A y B son los dos números de un bit que se desean sumar, siendo C y S las variables de salida del resultado de la suma. Como se verá más adelante, la salida C representa el arrastre (“carry”) de una suma hacia la columna de mayor peso.

Las expresiones lógicas de las dos funciones surgen directamente de la misma tabla:

$$S = \overline{A}.B + A.\overline{B} = A \oplus B;$$
$$C = A.B$$

y el circuito correspondiente, denominado **semisumador** para dos números de un bit *(one bit half adder*), es el que se muestra en la figura 5.17.

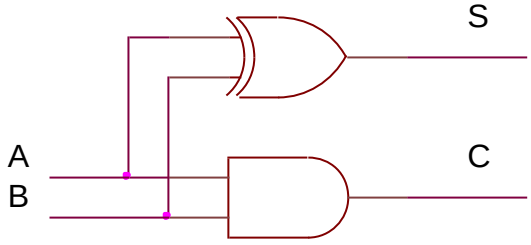


Figura 5.17.- Semisumador para dos números de un bit.

Dado que normalmente se requerirán sumas con más de un digito en cada operando, se hace necesario ampliar el alcance de este circuito. A tal efecto, el llamado **sumador completo** para números de un bit (*one bit full adder*) permite sumar tres operandos de un bit cada uno, identificados como A, B y C, siendo las salidas del circuito las mismas dos anteriores. En este caso, A y B podrían considerarse como dos operandos a sumar, a los que debe agregarse un arrastre proveniente de una columna anterior, el que podría estar representado por C.

Este circuito se obtiene a partir de la tabla de verdad que sigue, en la que S1 y S0 representan las salidas obtenidas de la suma aritmética de A, B y C:

A	**B**	**C**	**S1**	**S0**
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

De la misma surgen las ecuaciones correspondientes a S1 y S0, las que una vez reducidas, son las siguientes:

$$S1 = A.B + A.C + B.C$$
$$S0 = \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.C = A \oplus B \oplus C$$

de donde se obtiene el circuito siguiente como implementación de un circuito sumador completo para números de un bit.

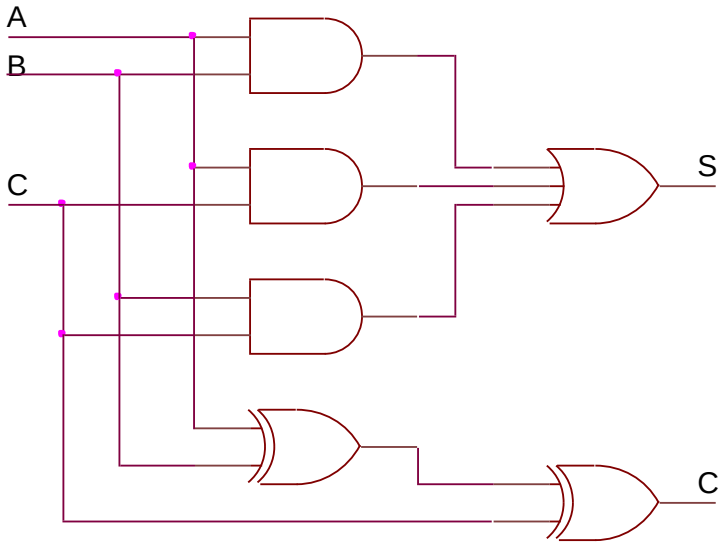


Figura 5.18.- Sumador completo para números de un bit.

5.9.2.- Sumador paralelo para números de más de un bit.

Es indudable que los sumadores vistos en el apartado anterior no resuelven la necesidad de permitir sumar números de mayor cantidad de bits, lo que, en cualquier sistema de cálculo, es una necesidad imperiosa.

Por consiguiente, habrá que plantear el diseño de un circuito que permita la suma de dos números de n bits, entregando el resultado como un número de n+1 bits. En un sumador paralelo, los dos números de n bits ingresan como entradas a un circuito que, por consiguiente, requerirá 2n entradas, entregando el resultado a través de n+1 salidas.

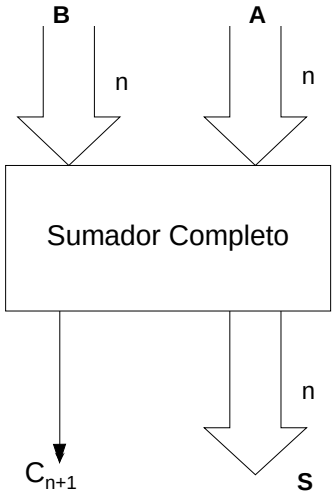


Figura 5.19. Diagrama en bloques de un sumador paralelo de n bits.

El sumador paralelo tiene la característica, como ventaja, de poder diseñarse como un circuito de dos niveles lógicos, lo que significa, en cuanto a su funcionamiento, la mayor velocidad de respuesta posible. Esta velocidad de respuesta, además, es independiente de la cantidad de bits que conforman los operandos a sumar. Como contrapartida, tiene la desventaja de la cantidad de entradas requeridas para resolver la suma, la que, como se ha dicho, es el doble del número de bits de los operandos.

Dado que no en todos los casos el requerimiento de velocidad no es el más importante, el diseñador de un sistema que requiera un circuito sumador deberá determinar si prioriza la velocidad de respuesta del circuito o su sencillez, para lo cual se desarrollarán, en los apartados siguientes, algunas alternativas a este tipo de planteo.

Ejemplo 5.2.- Sumador paralelo para dos números de dos bits.

Un sumador paralelo, que permita sumar dos números de dos bits cada uno, requerirá, de acuerdo con el diagrama en bloques de la figura 5.19, cuatro entradas y tres salidas, con lo que podrá responder a la tabla de verdad siguiente:

A1	A0	B1	B0	S2	S1	S0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1

0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

De la tabla de verdad planteada se obtienen las funciones lógicas a las que deberá responder el circuito:

$$S2 = \Sigma m\ (7,10,11,13,14,15)$$
$$S1 = \Sigma m\ (2,3,5,6,8,9,12,15)$$
$$S0 = \Sigma m\ (1,3,4,6,9,11,12,14)$$

Estas tres funciones, simplificadas, dan origen a las siguientes funciones mínimas:

$$S2 = A1.B1 + A0.B1.B0 + A1.A0.B0$$
$$S1 = A0.B0 \oplus A1 \oplus B1$$
$$S0 = A1 \oplus A0 \oplus B1 \oplus B0$$

Con lo cual el sumador paralelo que permita sumar dos números de dos bits cada uno se implementará a través de un circuito como el de la figura 5.20.a, esquematizado en la forma de bloque funcional en la figura 5.20.b

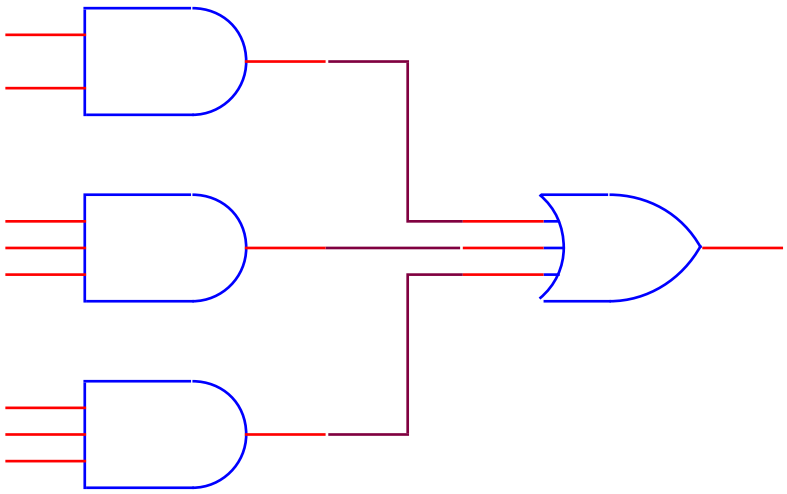


Figura 5.20.- Un sumador paralelo para números binarios de dos bits.

5.9.3.- Sumador serie para números binarios de n bits.

Se han mencionado en apartados anteriores las ventajas y desventajas de los circuitos planteados en dos niveles lógicos. En el caso de los circuitos sumadores, a la ventaja de su velocidad de operación, independiente de la cantidad de bits a sumar, se contrapone la desventaja derivada de la complejidad cada vez mayor a medida que va aumentando el tamaño de los sumandos.

Cuando la velocidad de respuesta no es tan importante como la sencillez del circuito, puede pensarse en utilizar elementos sencillos, tales como los sumadores para números de un bit ya presentados, como bloques funcionales que permitan, mediante su interconexión, obtener un resultado equivalente al del sumador paralelo.

La figura 5.21 permite ver la forma de implementar un circuito sumador para dos números de dos bits, equivalente al del ejemplo anterior, utilizando dos sumadores para números de un bit, concretamente un semisumador para la columna de las unidades y un sumador completo para la columna de mayor peso. En este caso, la salida de mayor peso del semisumador se utiliza como arrastre hacia la columna siguiente, tal como se procede cuando se suma con papel y lápiz.

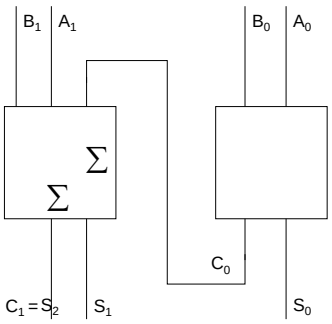


Figura 5.21.- Suma de dos números de dos bits, utilizando sumadores de un bit.

El semisumador correspondiente a la columna de las unidades suma los bits A0 y B0 de los dos operandos, entregando a su salida la suma aritmética S0 correspondiente a dichos bits. La salida C0 de ese semisumador, correspondiente al arrastre de la columna de las unidades, se inyecta al sumador completo junto con los bits A1 y B1 de los dos operandos, con lo que este sumador completo toma en cuenta dicho arrastre anterior al realizar la suma, entregando en sus salidas S1 y S2 los restantes bits de la suma S = A+B.

Este planteo puede extenderse hacia sumandos de cualquier cantidad de bits, simplemente mediante la interconexión en cascada o en serie de las salidas de arrastre de cada sumador con una de las entradas del sumador siguiente. Esto se muestra como ejemplo en la figura 5.22, en la que mediante el uso de cuatro bloques de un bit cada uno se procede a sumar dos sumandos de cuatro bits cada uno.

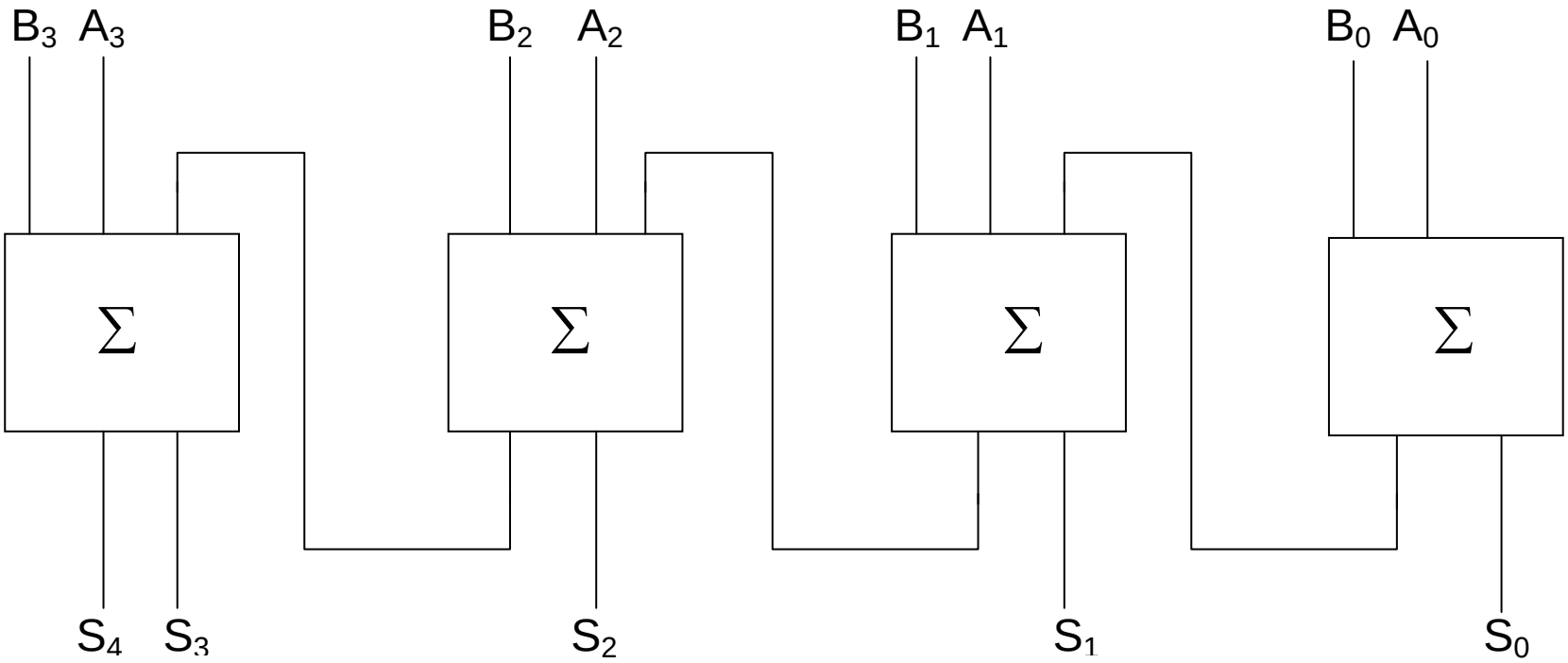


Figura 5.22.- Sumador serie para dos números de cuatro bits, utilizando sumadores de un bit.

En la figura 5.23 se plantea una solución diferente, para el mismo sumador de dos números de cuatro bits, en la que se utiliza como bloque funcional un sumador para números de dos bits, del tipo de los planteados en el ejemplo 5.2.

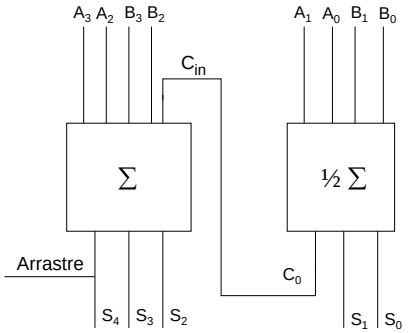


Figura 5.23.- Sumador serie para dos números de cuatro bits, utilizando sumadores de dos bits.

Nótese que en este caso, el segundo sumador es algo diferente al anteriormente visto, debido a que para poder hacer la suma planteada se requiere vincular ambos sumadores por medio de las correspondientes salidas y entradas de arrastre.

Llegados a este punto, cabe analizar las diferencias, ventajas y desventajas de cada una de las soluciones posibles para este último ejemplo.

Si se requiere sumar dos números de n bits, la primera solución planteada es la que corresponde al sumador paralelo. En este caso, se tendrá un circuito, de dos niveles lógicos, pero con 2n entradas y n+1 salidas. Para un ejemplo de sumandos de ocho bits debe pensarse en 16 entradas y nueve salidas. La dificultad evidente pasa por el método de diseño de un circuito de 16 entradas. Si bien no es imposible desarrollarlo, es probable que la implementación con compuertas resulte en un circuito complejo. Como contrapartida, el circuito responderá a los cambios en sus entradas en un tiempo equivalente a la suma de los tiempos de propagación de dos compuertas.

Si el mismo circuito sumador para dos números de ocho bits se resuelve con sumadores de un bit, se obtendrá un circuito mucho más simple en su implementación, pero, dado que los bloques sumadores se interrelacionan entre sí a través de las salidas de arrastre, el tiempo de propagación se verá aumentado en ocho veces, siendo el tiempo total de propagación, en consecuencia, el equivalente a la suma de los tiempos de propagación de 16 compuertas.

En el caso de la implementación con sumadores para números de dos bits, la complejidad de cada bloque funcional aumenta con respecto al caso anterior, pero esta mayor complejidad de los bloques individuales se compensa con el menor tiempo total de propagación, equivalente en este caso al tiempo de propagación de cuatro bloques encadenados. Siendo los bloques implementaciones de dos niveles, el tiempo total de propagación será el de ocho compuertas.

5.9.4.- Implementación serie paralelo. Acelerador de arrastre.

Si se pretende lograr una mejor eficiencia en los circuitos sumadores planteados en el apartado anterior, puede pensarse en alguna alternativa que disminuya los tiempos de propagación

elevados de los circuitos sumadores serie, sin llegar a la complejidad de los circuitos sumadores en paralelo.

Una de las alternativas posibles pasa por intentar evitar que cada uno de los bloques del sumador serie deba esperar a que el anterior calcule el arrastre de su columna y lo trasfiera. Esta reducción de tiempos se podrá lograr si cada una de las etapas del sumador es capaz de determinar cual es el arrastre que vendría desde la etapa anterior sin esperarlo. Esto requerirá que el diseño de cada etapa incluya la lógica correspondiente al cálculo del arrastre de la etapa anterior, y por lo tanto, hará que cada una de las etapas del sumador aumente en dos el número de sus entradas con respecto de las entradas de la etapa anterior. Este esquema se plantea en la figura 5.24, donde se observa que para sumar dos números de cuatro bits cada uno se tienen cuatro sumadores, en los que cada etapa sumadora requiere una sola salida, la que corresponde a la suma de los sumandos de esa columna. Solamente la última etapa tendrá una salida adicional, la que corresponde al quinto bit de la suma de dos números de cuatro bits cada uno Mientras tanto, la cantidad de entradas de cada etapa aumenta desde dos en la primer columna hasta ocho en el último bloque sumador.

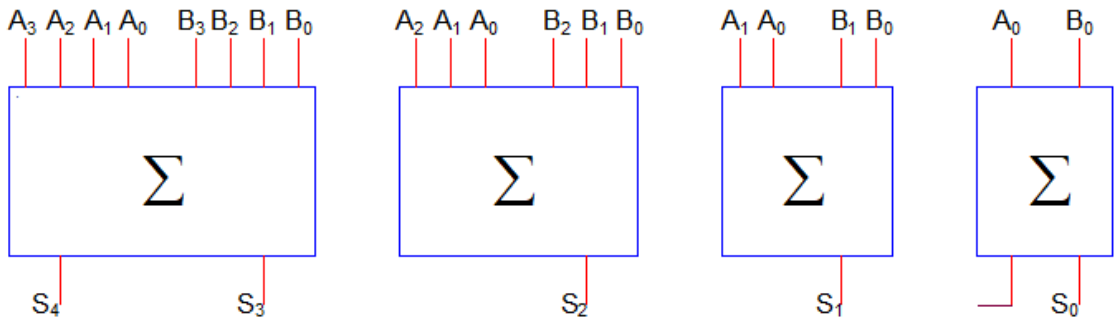


Figura 5.24.- Circuito sumador para dos números de cuatro bits con cálculo simultáneo de arrastres.

Si este mismo esquema se plantea para dos números de ocho bits cada uno, se ve claramente que la cantidad de entradas de cada etapa aumenta desde dos en el primer bloque (el de la columna de peso 2^0), hasta diez y seis en el bloque que debe sumar la columna de peso 2^7. Dado que esta cantidad de entradas coincide con la que presenta el sumador paralelo para dos números de ocho bits, debe notarse que la diferencia con el sumador paralelo es la de tener, en este caso, solamente dos salidas, las correspondientes a S7 y S8, contra nueve salidas del sumador paralelo propiamente dicho.

Esto implica una probable reducción de la complejidad del circuito de cada una de las etapas con respecto del sumador paralelo, permitiendo un tiempo de propagación equivalente al de aquel. No obstante, la complejidad de cada etapa es bastante mayor que la de las etapas que forman un sumador serie, por lo que en ciertos casos, especialmente en aquellos en los que se requiere una solución de compromiso entre velocidad y complejidad circuital, se debe buscar una alternativa intermedia. En este caso, el planteo pasa por calcular los arrastres en forma anticipada solamente en algunas de las etapas, hasta llegar a un nivel de complejidad circuital aceptable.

Otra solución se puede obtener a partir de una generalización de las expresiones lógicas que representan las salidas de arrastre generadas por cada una de las etapas del sumador, planteándolas en función del arrastre producido por dicha etapa y del eventual arrastre proveniente de etapas de menor peso. En este caso, los circuitos sumadores conocidos como de **arrastre anticipado** (*look ahead carry*), permiten implementar funciones de generación y de propagación de los arrastres entre las distintas etapas, permitiendo así una sustancial mejora de la eficiencia del circuito.

5.9.5.- Circuitos sumadores para números decimales codificados.

Los ejemplos anteriores hacen referencia, en todos los casos, a la suma de números expresados en el sistema binario de numeración. Si bien en muchas aplicaciones los sistemas de cálculo y cómputo operan directamente con números expresados en binario natural, no es menos cierto que existen situaciones en las que se requiere la suma de números decimales codificados en algún código binario. En el capítulo correspondiente se ha analizado la metodología de operación para los códigos habitualmente utilizados para realizar operaciones aritméticas con números decimales, por lo que en este punto se plantearan simplemente los circuitos por medio de los cuales pueden llevarse a cabo dichas operaciones.

A tal efecto, se tomará como bloque funcional básico un circuito sumador completo para dos sumandos de cuatro bits cada uno, el que podrá adoptar la forma de un sumador serie completo, paralelo completo, paralelo con acelerador de arrastre o cualquier otra forma que se considere adecuada para su implementación. Dicho circuito, tomado como bloque funcional, se representa en la figura 5.25.

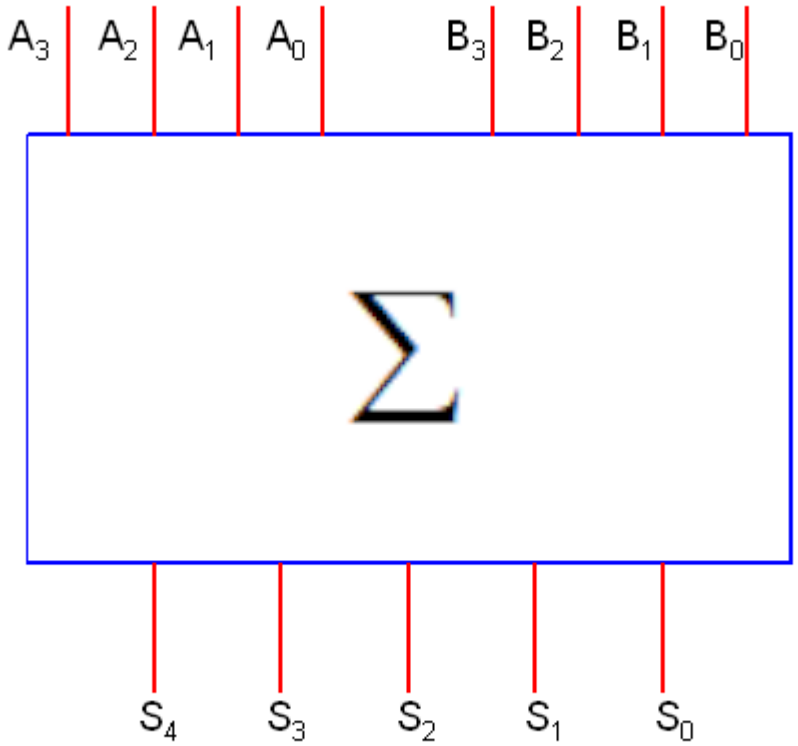


Figura 5.25.- Un bloque funcional para sumar dos números de cuatro bits cada uno.

Este bloque funcional se toma como base, en el ejemplo siguiente, para realizar la suma de dos números decimales de un digito cada uno, expresados en código 8421. Debe recordarse aquí nuevamente que esto significa que los dos operandos solamente podrán estar en el rango 0..9, en tanto que el resultado adoptará cualquier valor entre 0 y 19 (para el caso de sumar 9 más 9 más un eventual bit de arrastre de una columna decimal de menor peso), debiendo aparecer también codificado en el mismo código binario que aquellos.

En este caso, y según ya se planteó en el apartado correspondiente, la suma de dos números decimales codificados en el código 8421 debe corregirse, sumando el número 6 al resultado obtenido cuando dicho resultado supere el valor de 9. Cuando el resultado de la suma está entre 0 y 9 no se requiere corrección, o lo que es equivalente, la corrección pasa por sumar cero al resultado obtenido.

La solución a este planteo pasa por implementar una función, que determine, en función del valor de la salida del sumador, si el resultado entregado por el mismo debe corregirse o no.

La tabla de verdad siguiente muestra la expresión lógica de dicha función de detección de la necesidad de corregir, sobre la base de las siguientes premisas:

- *Se debe sumar seis al resultado cuando dicho resultado es mayor que nueve.*
- *No se debe sumar nada cuando el resultado está entre cero y nueve.*
- *El valor de la salida del sumador puede estar entre cero y 19 (9+9+1).*
- *No pueden obtenerse valores de salida mayores que 19, por lo que dichos valores, para la implementación, se considerarán como combinaciones prohibidas.*

De estas premisas surge la tabla de verdad en cuestión, en la que se muestra la salida E, que indica si el resultado de la suma debe corregirse o no, en función de dicho resultado:

S4	**S3**	**S2**	**S1**	**S0**	**E**
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	X
1	0	1	0	1	X
1	0	1	1	0	X
1	0	1	1	1	X
1	1	0	0	0	X
1	1	0	0	1	X
1	1	0	1	0	X
1	1	0	1	1	X
1	1	1	0	0	X
1	1	1	0	1	X

1	1	1	1	0	X
1	1	1	1	1	X

De la simplificación de esta tabla por cualquiera de los métodos conocidos, se obtiene como expresión del circuito de detección de error:

$$E\ (S4,S3,S2,S1,S0) = S4 + S3.S2 + S3.S1$$

Lo que puede representarse en el circuito de la figura 5.26:

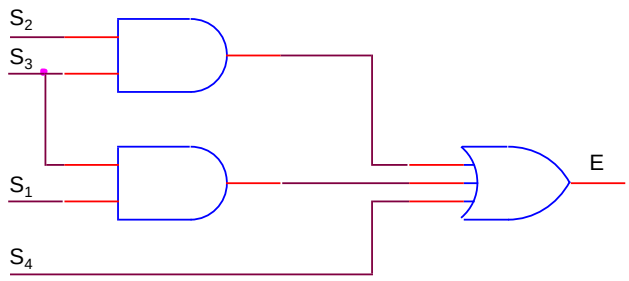


Figura 5.26.- Circuito que permite detectar cuando debe corregirse el valor de la suma obtenida a la salida del sumador de cuatro bits.

Para obtener el circuito final del sumador que permita la suma de dos dígitos decimales expresados en BCD 8421, falta interconectar al sumador con el circuito detector recién obtenido y con otro sumador que haga la corrección, en forma controlada por este circuito detector.

Puede observarse que, en relación con la salida E del circuito de detección, la necesidad de corregir se puede plantear de la siguiente forma:

Cuando E = 0, debe sumarse 0000 al número de cuatro bits obtenido en la salida del sumador. En tanto, cuando E =1, lo que debe sumarse es 0110.

Esto podrá resolverse, en cualquiera de los dos casos, con un sumador para dos números de cuatro bits, conectado a la salida del anterior y controlado por la función E. En este segundo sumador, puede plantearse que el valor del operando corrector, que se sumará al resultado expresado por el primer sumador, es 0EE0.

No termina aquí la cuestión. Dado que la suma original, en el primer sumador, tiene un rango que va desde cero hasta 19, el resultado entregado por ese primer sumador puede tener un cero o un uno en el bit mas significativo.

Si el resultado es de cuatro bits, con cero en el más significativo y con salida cero en el circuito de detección, (lo que significa un resultado entre 0 y 9 inclusive), no habrá corrección, y la salida del segundo sumador coincidirá con el del primero.

Si el resultado es de cuatro bits, pero el circuito de detección entrega un uno en su salida, debe entenderse que ese resultado está en el rango 10 – 15, con lo que, cuando se realice la corrección del caso, se obtendrá un arrastre (resultado en cinco bits) en el segundo sumador, representado por S4´.

Si, por último, el resultado entregado por el primer sumador ya tiene su bit más significativo en uno, el resultado obtenido está entre 16 y 19, por lo que ya ese primer sumador está informando el arrastre hacia la columna de mayor peso mediante la señal S4. Al realizarse la corrección en el segundo sumador no habrá arrastre.

Este planteo tiene que ver con el arrastre de salida del circuito, el que debe eventualmente llevarse como entrada a otro sumador similar que sume los dígitos de mayor peso de los números a sumar.

Por consiguiente, y para completar el esquema planteado, se muestra en la figura 5.27 el esquema completo de un sumador que permita sumar dos números decimales de un dígito cada uno, expresados en código BCD 8421. El elemento indicado como **corrector** representa, en forma de bloque funcional, al circuito de detección indicado en la figura 5.26.

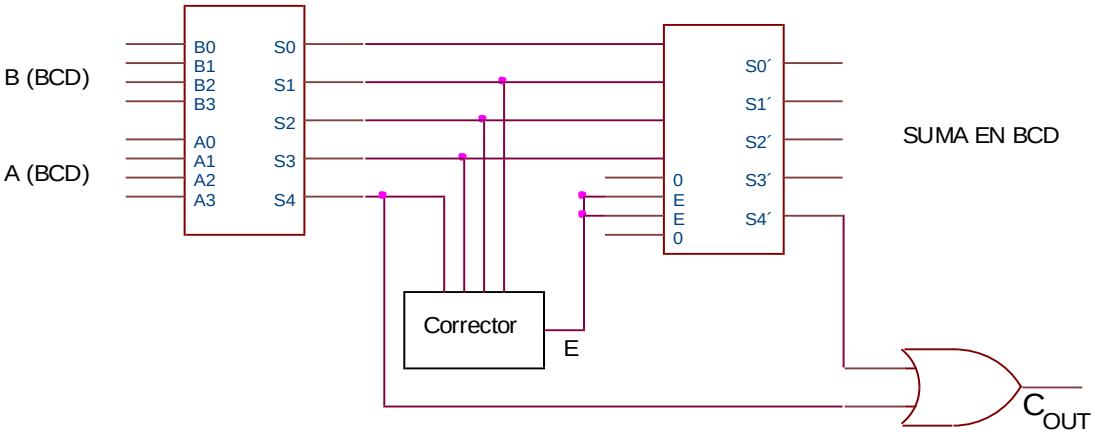


Figura 5.27.- Un sumador completo para números de un dígito codificados en 8421.

A nivel de la tecnología actual, el circuito de la figura 5.27, basado a su vez en el sumador binario de la figura 5.25, podrá considerarse como bloque funcional básico, por lo que varios de estos bloques podrán interconectarse entre sí, con el objeto de permitir sumas de números decimales de varios dígitos. En la figura 5.28 se ejemplifica este caso para dos números decimales de dos dígitos cada uno. En la misma, y con el objeto de simplificar la figura, los conjuntos de líneas de entrada y salida se representan en forma abreviada.

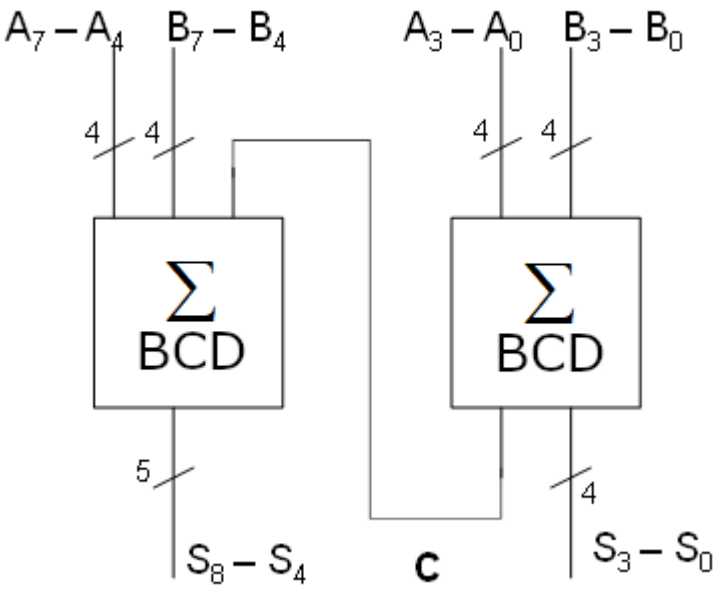


Figura 5.28.- Sumador BCD para números de dos dígitos decimales.

5.9.6.- Circuitos restadores.

En forma similar al planteo anterior, pueden requerirse circuitos que permitan restar dos números binarios. El esquema de estos circuitos es similar al de los circuitos sumadores, dado que las diferencias básicas consistirán en la interpretación de sus valores de salida. La tabla de verdad de un circuito restador para restar números de un bit será la siguiente, interpretando que el circuito resuelve la operación A – B. En este caso, el bit de salida C representa el caso en que, por ser A menor que B, se debe pedir una unidad de mayor peso a la columna correspondiente (*borrow*).

Por consiguiente, la tabla de verdad es la siguiente:

A	**B**	**C**	**S**
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Las ecuaciones correspondientes son:

$$S = \overline{A}.B + A.\overline{B};$$
$$C = \overline{A}.B$$

por lo que resultan ser muy similares a las del circuito semisumador de un bit. El circuito correspondiente se esquematiza en la figura 5.29.

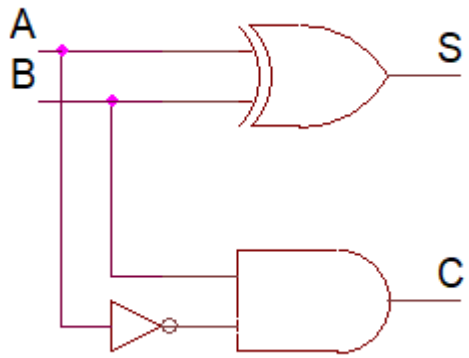


Figura 5.29.- Circuito restador para dos números de un bit.

5.9.7.- Implementación de un circuito sumador - restador elemental.

En virtud de lo planteado en el capítulo referido a operaciones aritméticas, es factible pensar en la utilización de un mismo circuito lógico que permita realizar tanto las operaciones de suma como las de resta. En efecto, el concepto de complemento de un número con respecto a la base, o a la base menos uno, permite plantear un circuito que sume o reste, considerando básicamente la idea de que restar dos números coincide con el sumar el primer número con el complemento del segundo. Bastará entonces presentar en la entrada correspondiente al segundo operando el operando en sí o su complemento, según lo determine la operación a realizar.

La figura 5.30 plantea un diagrama en bloques de un circuito que permita sumar o restar dos números de cuatro bits cada uno. La señal de control que indica si se deben sumar o restar los operandos A y B, indica en definitiva si se debe tomar en cuenta el número B o su complemento. El circuito entregará el resultado en notación complementada, por lo que deberá tomarse en cuenta el signo del resultado para interpretarlo correctamente.

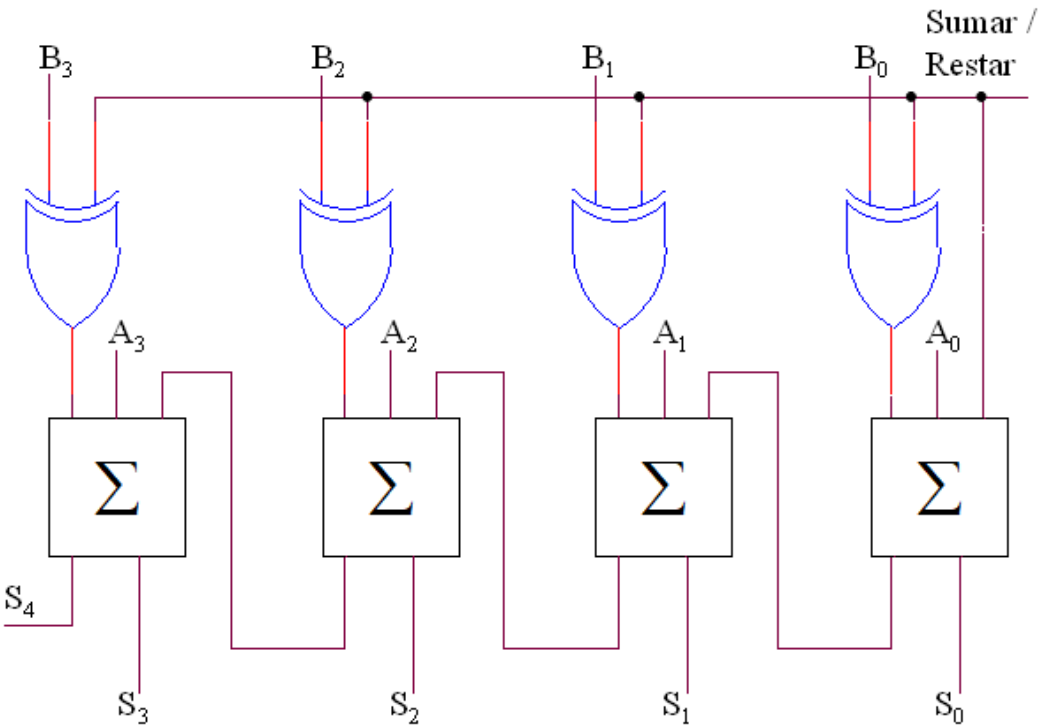


Figura 5.30.- Sumador – restador para números de cuatro bits.

5.10.- Circuitos comparadores.

Se define como circuito **comparador** a aquel circuito que, ante dos valores numéricos A y B presentados en sus entradas, permite determinar si ambos números son iguales entre sí o no. Los diferentes tipos de comparador permiten distintas variantes de este concepto, dado que en el caso más simple, la salida del comparador solamente indica que A y B son iguales o distintos. Este tipo de circuito suele conocerse también como **detector de igualdad**, dado que la única relación que podrá analizarse es la igualdad o desigualdad de sus entradas, independientemente de sus valores lógicos.

En casos más complejos, las salidas del comparador permitirán plantear las alternativas de mayor, menor o igual, así como combinaciones de las mismas, tales como A ≥ B, etc.

Un circuito detector de igualdad para dos números de un bit cada uno detecta la situación en la cual ambos números son cero o uno. Sin necesidad de plantear la tabla de verdad, surge como expresión de la salida:

$$S = \overline{A}.\overline{B} + A.B$$

lo que corresponde a la salida de una compuerta XOR negada. El circuito correspondiente es el de la figura 5.31.

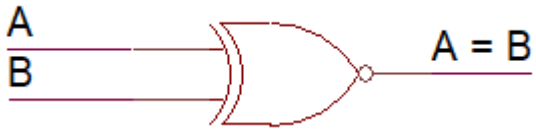


Figura 5.31.- Detector de igualdad (comparador) para dos números de un bit.

Si lo que se pretende es un circuito que detecte, para dos números de un bit, las relaciones de igual, mayor o menor, el circuito tendrá tres salidas y responderá a la siguiente tabla de verdad:

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

Como se observa, las tres salidas son mutuamente excluyentes, en el sentido de que no pueden darse dos salidas simultáneamente en uno. El circuito correspondiente es el de la figura 5.32, en el que se incluye además una representación en bloques del mismo.

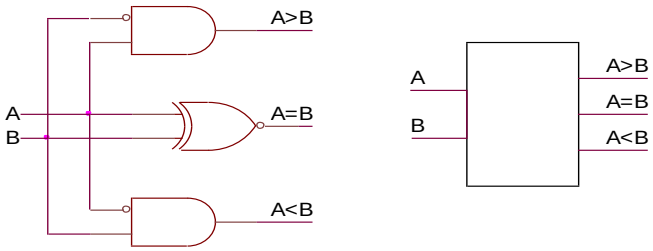


Figura 5.32.- Un comparador por mayor, menor o igual para números de un bit

En la generalidad de los casos, como ya se ha dicho reiteradamente, lo que se necesita es el manejo de una cantidad dada de bits en forma simultanea. Por consiguiente, en el caso de la

comparación de datos numéricos, lo que se requiere es un circuito que permita comparar dos números formados cada uno de ellos por n bits.

Si lo que se requiere es simplemente verificar que ambos números sean iguales o no, el planteo es muy simple, dado que surge claramente que dos números son iguales cuando todos los pares de bits ubicados en las mismas posiciones de ambos números son iguales. Esto significa que, por ejemplo, para dos números de cuatro bits A y B, los mismos serán iguales cuando se cumplan simultáneamente las cuatro igualdades $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ y $A_0 = B_0$.

En estas condiciones, el circuito que determine la igualdad de A y B podrá implementarse a través del producto lógico de cuatro detectores de igualdad como los de la figura 5.32. El esquema correspondiente es el de la figura 5.33.

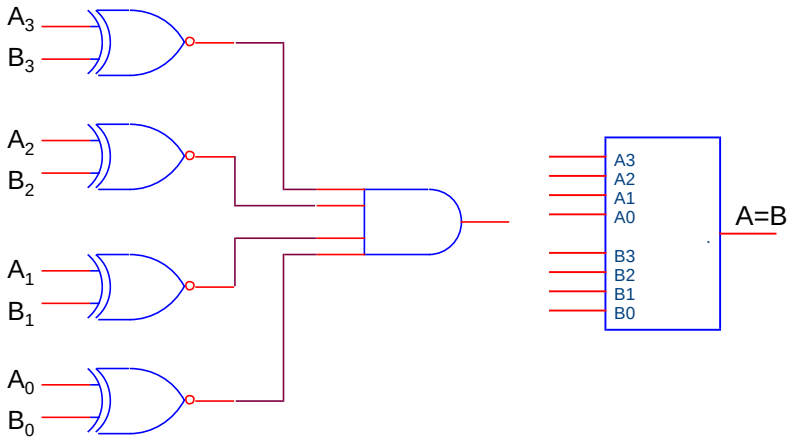


Figura 5.33.- Detector de igualdad para dos números de cuatro bits.

Si en lugar de detectar solamente la igualdad entre los dos números se pretende generar un circuito que permita compararlos en el sentido amplio, se deberá plantear una tabla de verdad para cada una de las tres funciones A>B, A=B, A<B.

Si ambos números A y B están formados por n bits, la correspondiente tabla de verdad tendrá 2n entradas, dando lugar a lo que puede denominarse un comparador paralelo o simultaneo.

Queda claro que en la medida en que aumente la cantidad de bits de A y de B, la tabla de verdad resultará más compleja, pero, cualquiera sea la cantidad de bits que se requiera comparar, se obtendrá un circuito planteado en dos niveles lógicos, lo que implicará una comparación en alta velocidad.

Ejemplo 5.3.- Circuito comparador completo para dos números de dos bits.

Se requiere implementar un circuito que permita comparar dos números A y B, formados por dos dígitos cada uno, indicando, con tres salidas diferenciadas, las condiciones de A>B, A=B, A<B.

El diseño de este circuito requiere la implementación de la correspondiente tabla de verdad, donde A_1A_0 forman el numero A, donde B_1B_0 forman el numero B, y donde las tres salidas Z_3, Z_2, Z_1 corresponden respectivamente a las condiciones a detectar, con lo que Z_1 = 1 si A>B, Z_2 será 1 si A=B, y Z_3 será 1 si A<B.

La tabla de verdad se plantea a continuación:

A_1	A_0	B_1	B_0	Z_1 A>B	Z_2 A=B	Z_3 A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Las tres funciones se simplifican por cualquiera de los métodos conocidos, dando lugar al siguiente conjunto de ecuaciones. Nótese que la representación de la función Z_2 no puede lograrse a través de la forma convencional suma de productos, sino que se obtiene como un producto de comparadores de un bit.

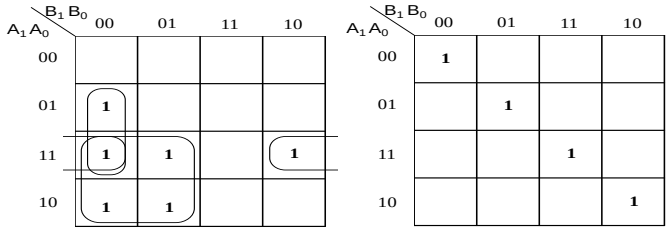


Figura 5.34.- Diagramas de Karnaugh para la simplificación de Z_1, Z_2, Z_3.

Las expresiones finales para este comparador son las siguientes:

$$Z_1 = A_1.\overline{B_1} + A_0.\overline{B_1}.\overline{B_0} + A_1.A_0\,\overline{B_0}$$

$$Z_2 = (\overline{A_1 \oplus B_1}).(\overline{A_0 \oplus B_0})$$

$$Z_3 = \overline{A_1}.B_1 + \overline{A_1}.\overline{A_0}.B_0 + \overline{A_0}.B_1.B_0$$

Siendo el circuito correspondiente el de la figura 5.35.

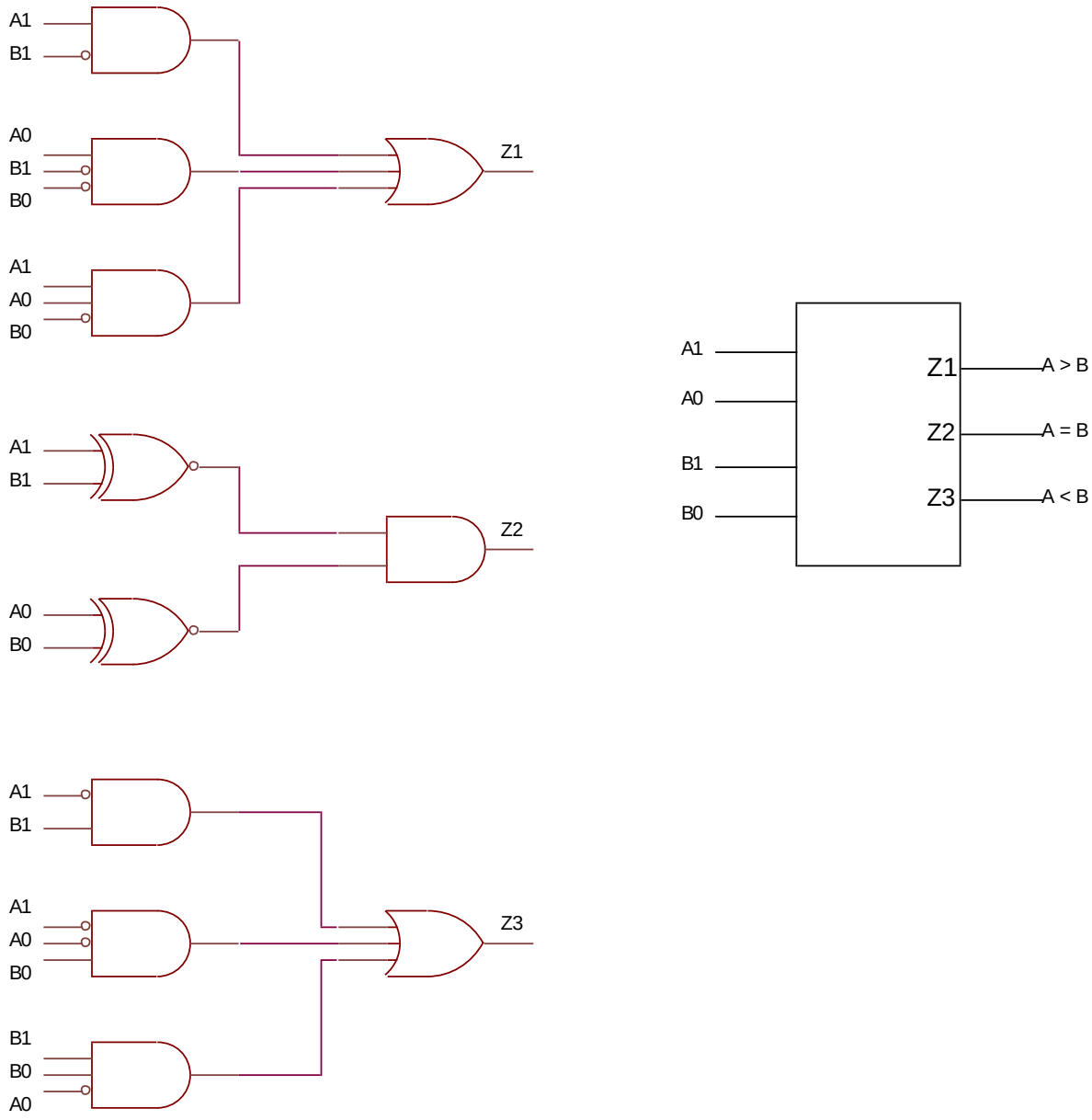


Figura 5.35.- Comparador paralelo para dos números de dos bits.

Como ya ha sido dicho, el inconveniente del circuito comparador paralelo es el de su complejidad cada vez mayor cuanto más grandes sean los números a comparar. Este inconveniente puede obviarse, a costa de una pérdida en la velocidad de comparación, planteando un esquema de comparadores encadenados (en serie o en cascada), en forma similar a lo que se ha visto en el caso de los circuitos sumadores.

En un comparador en cascada, el criterio es el de considerar que, dados dos números A y B, se puede determinar cuando A>B o cuando A<B comparando los dos bits de mayor peso. Siendo A_n y B_n los dos bits más significativos de A y B, surge claramente que la única forma en que A puede ser mayor que B es que A_n sea mayor que B_n. Análogamente, A será menor que B cuando A_n sea menor que B_n. Solamente si A_n y B_n son iguales, habrá que analizar qué ocurre con los

bits de menor peso. Si siendo $A_n = B_n$, A será mayor que B si $A_{n-1} > B_{n-1}$, y A será menor que B si $A_{n-1} > B_{n-1}$. Otra vez, si los bits siguientes a los más significativos también son iguales, habrá que comparar los siguientes, y así repetitivamente hasta llegar a los bits menos significativos de ambos números. El circuito correspondiente, para dos números de cuatro bits, se muestra en la figura 5.36.

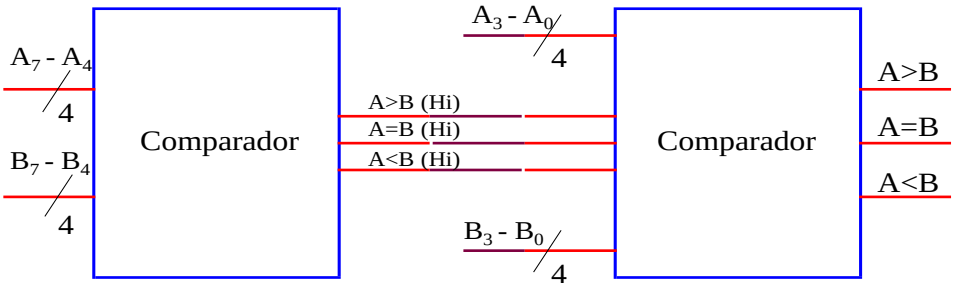


Figura 5.36.- Circuito comparador en cascada para dos números de cuatro bits.

5.11.- Circuitos codificadores, conversores de código y decodificadores.

Los circuitos que se plantean en este apartado tienen relación directa con los conceptos vistos en el capítulo correspondiente a los códigos binarios. En efecto, y dado que en aquel momento se planteó el concepto de codificación binaria como el de la representación de distintos elementos mediante conjuntos de unos y ceros, surge inmediatamente la posibilidad de desarrollar distintos tipos de circuitos lógicos relacionados con dicho concepto.

Se pueden, por ende, describir los siguientes tipos de circuitos lógicos relacionados con códigos binarios:

Un circuito **codificador** recibe información binaria no codificada y la convierte en un código determinado. Por información no codificada puede entenderse cualquier tipo de datos binarios que no reúne las características definidas en su momento para un código binario.

Un ejemplo típico se presenta en la entrada de datos de una computadora, a través de un teclado, cuya estructura de filas y columnas genera señales eléctricas que indican cual es la tecla que se oprimió, mediante la indicación de la correspondiente intersección fila columna. Un circuito codificador convertirá esas intersecciones en los correspondientes elementos binarios (ceros y unos) de un código alfanumérico que permita la posterior representación e interpretación de los datos ingresados.

Un circuito **conversor de códigos**, o **transcodificador**, recibe información binaria representada en un cierto código A, entregando a su salida la información en otro código B, distinto al anterior pero representativo de la misma información de entrada.

En el caso de un circuito **decodificador**, la información codificada a su entrada se entrega a la salida sin codificación alguna. En un ejemplo habitual, un decodificador puede convertir información numérica representada, por ejemplo, en código decimal 8421, en una salida de 10

elementos individuales, que identifiquen cada combinación binaria mediante el encendido de una lámpara.

En todos los casos se trata de circuitos combinatorios, dado que cada vez que se produzca una dada combinación a la entrada del circuito, la salida será siempre la misma.

Por consiguiente, todos estos circuitos pueden representarse mediante tablas de verdad e implementarse mediante los métodos desarrollados en apartados anteriores.

5.12.- Circuitos para la detección y corrección de errores.

5.12.1.- Circuitos generadores de paridad.

Con el objeto de proteger la información binaria en un sistema sujeto a errores, se implementan procedimientos para la detección y corrección de errores, para lo cual se utilizan los conceptos de paridad desarrollados en el capítulo 3. En este punto, se plantean como ejemplos los circuitos requeridos para generar bits de paridad, así como los circuitos utilizados para la detección y corrección de errores.

Se entiende como circuito generador de paridad a aquel circuito que es capaz de entregar un bit de paridad, par o impar, para una determinada cantidad de bits de datos que se presentan a su entrada.

Ejemplo 5.4.- Circuito para la generación de un bit de paridad par en palabras de dos bits.

Se plantea el circuito básico de generación de paridad, que permite agregar un bit de paridad a un par de bits, tomados como entradas del circuito generador.

La correspondiente tabla de verdad es la siguiente:

A	*B*	*P(A,B)*
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

Como se observa, la tabla obtenida corresponde a la expresión ya conocida de una compuerta XOR (O excluyente), por lo que puede plantearse, como regla general, que dicha compuerta es un elemento básico para la generación de bits de paridad par.

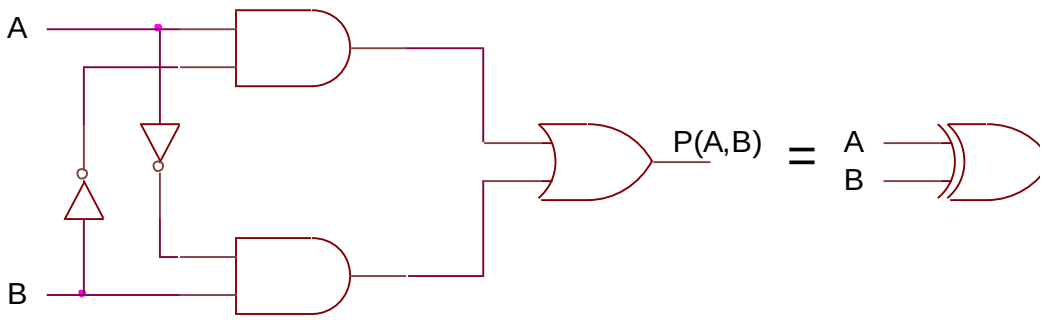


Figura 5.37.- Un generador de paridad par para dos bits de datos.

Este circuito permite su generalización a cualquier numero de bits de datos, simplemente mediante la utilización de tantas compuertas XOR como sea necesario. El ejemplo siguiente se plantea para palabras de mayor cantidad de bits.

Ejemplo 5.5.- Generador de paridad para cuatro bits de datos.

Se ejemplifica para un conjunto de elementos expresados en cuatro bits, expresados en binario puro, para el cual a continuación se representa la tabla de verdad correspondiente

ABCD	***Paridad par P(A,B,C,D)***
0000	*0*
0001	*1*
0010	*1*
0011	*0*
0100	*1*
0101	*0*
0110	*0*
0111	*1*
1000	*1*
1001	*0*
1010	*0*
1011	*1*
1100	*0*
1101	*1*
1110	*1*
1111	*0*

La ecuación correspondiente a la tabla de verdad planteada resulta ser

$$P(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.C.D + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.C.D + A.B.\overline{C}.D + A.B.C.\overline{D}$$

o bien, en forma abreviada,

$$P(A,B,C,D) = \Sigma m(1,2,4,7,8,11,13,14)$$

La expresión se vuelca en un mapa K para su simplificación, el que se ilustra en la figura 538.

AB \ CD	00	01	11	10
00		**1**		**1**
01	**1**		**1**	
11		**1**		**1**
10	**1**		**1**	

Figura 5.38.- Diagrama de Karnaugh para la función P(A,B,C,D)

Del diagrama puede observarse que la misma no puede simplificarse como lógica de dos niveles, lo que es coherente con lo afirmado en el ejemplo anterior. La simplificación de la función anterior por medio de compuertas XOR lleva a:

$$P(A,B,C,D) = A \oplus B \oplus C \oplus D$$

expresión cuyo circuito lógico se refleja en la figura 5.39.
FIGURA 39

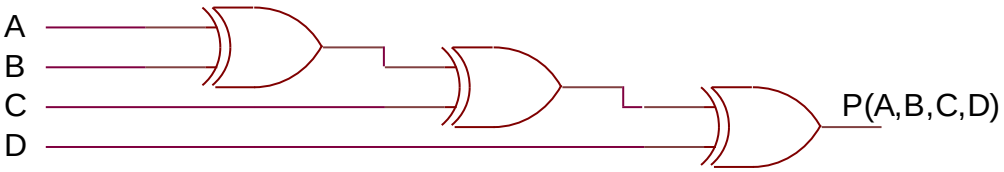


Figura 5.39.- Generador de paridad para una palabra de cuatro bits.

5.12.2.- Circuitos para detección de errores.

De acuerdo con lo planteado, la detección de errores en información binaria solamente puede analizarse cuando la mencionada información ha sido previamente codificada con el agregado de un bit de paridad. Por consiguiente, y en esa situación, lo que deberá hacerse para verificar la seguridad de la información previamente protegida, será verificar si la paridad completa de la palabra (incluyendo el bit de paridad previamente agregado) sigue siendo la misma que cuando se la codificó. Para plantearlo con mayor claridad, si una palabra de cuatro bits se protege con un bit de paridad par (tal como en el ejemplo 5.5), la verificación deberá realizarse analizando si los cinco bits de la palabra codificada sigue siendo par.

El circuito que permite verificar la corrección de la palabra de cuatro bits codificada con paridad con paridad par será, entonces, el mismo circuito que permitiría calcular la paridad par de la palabra de cinco bits. El circuito lógico correspondiente al detector de errores para una palabra de cinco bits (cuatro de datos más paridad) se muestra en la figura 5.40.
FIGURA 40

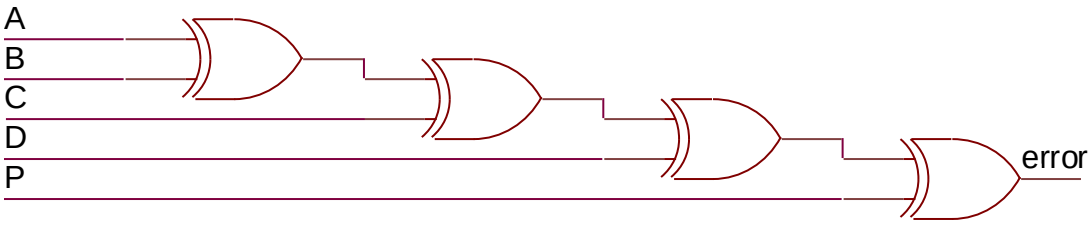


Figura 5.40.- Detector de errores para cinco bits (cuatro más paridad)

5.12.3.- Circuitos para la corrección de errores.

Cuando se requieren corregir errores en una palabra representada en un código corrector, debe plantearse primero la detección de los eventuales errores existentes. Una vez detectada la existencia de error, deberá determinarse cuales son los bits que han sufrido el error, para, por último, realizar la corrección.
El esquema general que puede plantearse para el sistema de detección y corrección se plantea, a nivel de diagrama de bloques, en la figura 5.41. En la misma se observan las distintas etapas correspondientes a la generación de los bits de paridad, la verificación de los mismos, la determinación del bit errado y la corrección del error.

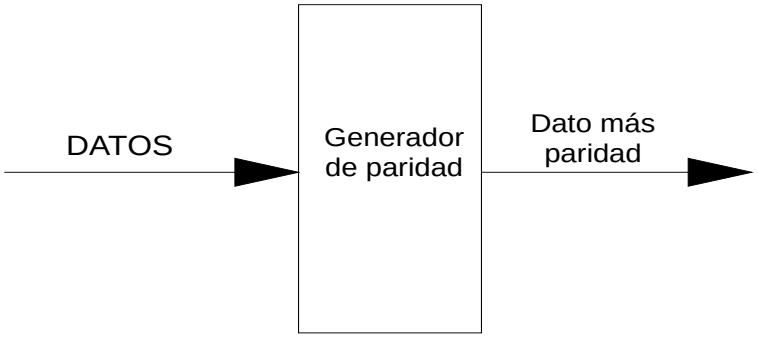


Figura 5.41.- Diagrama en bloques de un sistema de generación de paridad y corrección de errores.

Como ejemplo de aplicación, se planteará el procedimiento requerido para corregir información codificada según el método de Hamming, desarrollado en el apartado 3.8 del capítulo anterior. A tal efecto, debe recordarse que el método mencionado permite codificar información, de cualquier número de bits, mediante el agregado de bits de paridad parcial, con el objeto de llevar un código de distancia 1 a distancia 3, lo que permitirá posteriormente corregir un error. En el ejemplo siguiente se procederá a la implementación de los circuitos que permitan, consecutivamente, generar los bits de paridad, verificarlos y eventualmente corregirlos. Se utilizará a tal fin el mismo valor binario del ejemplo 3.4.

Ejemplo 5.6: Implementar el circuito que permita codificar según el método de Hamming la palabra de datos 0101.

Repitiendo el planteo del ejemplo 3.4, la palabra original, de cuatro bits, requiere el agregado de tres bits de paridad, para adoptar la forma

P1 P2 X3 P4 X5 X6 X7

Lo que en este caso, y con los valores dados a los bits de datos, será:

P1 P2 0 P4 1 0 1

Las expresiones de paridad para P1, P2 y P4, de acuerdo con lo ya visto, pueden representarse en términos de compuertas XOR:

$$P4 = X5 \oplus X6 \oplus X7$$
$$P2 = X3 \oplus X6 \oplus X7$$
$$P1 = X3 \oplus X5 \oplus X7$$

lo que lleva al circuito generador de paridad de la figura 5.41.

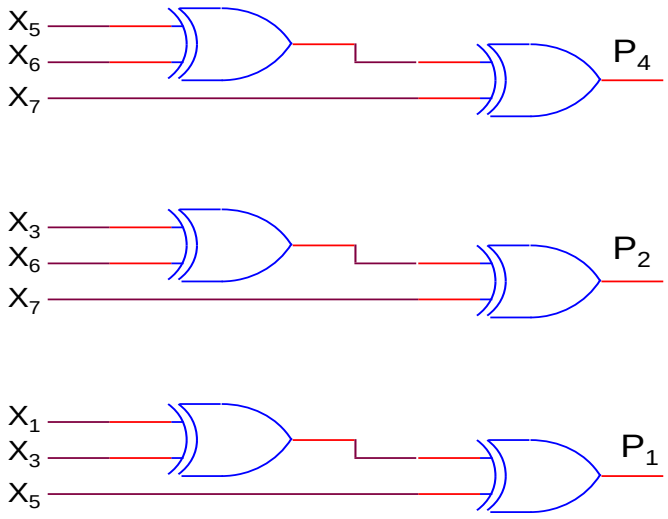


Figura 5.42.- Un circuito generador de paridad según Hamming para cuatro bits de datos.

Una vez generados los bits de paridad, la palabra de 7 bits puede verificarse para detectar errores, por medio de las expresiones generales del método de Hamming, las que, para el caso de cuatro bits de datos y tres de paridad, se enuncian en base a las siguientes expresiones:

$$E4 = P4 \oplus X5 \oplus X6 \oplus X7 = 0$$
$$E2 = P2 \oplus X3 \oplus X6 \oplus X7 = 0$$
$$E1 = P1 \oplus X3 \oplus X5 \oplus X7 = 0$$

El circuito correspondiente al detector de errores es el de la figura 5.43, en el que se incluye una compuerta O que incluya las tres posibles condiciones de error.

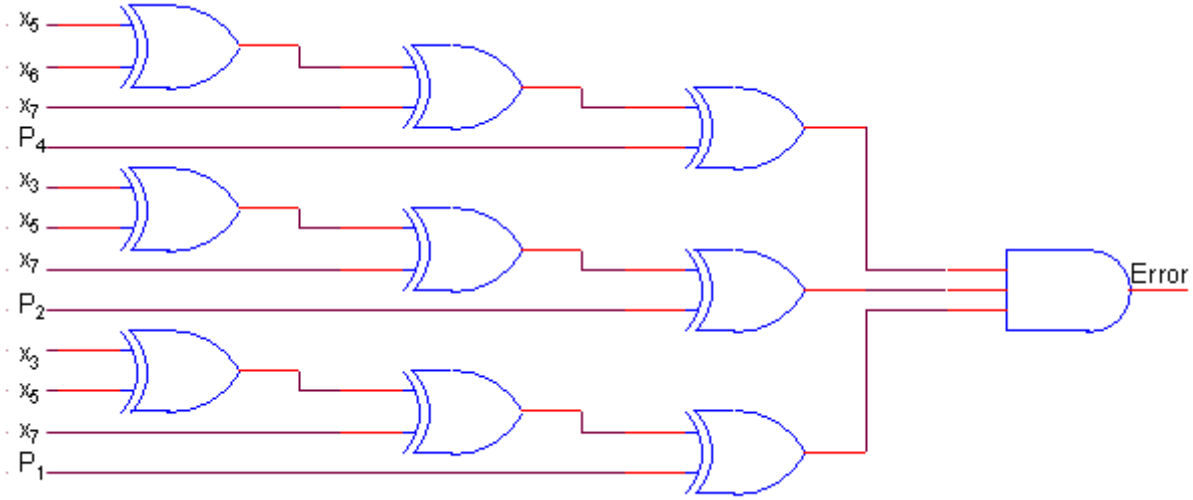


Figura 5.43.- Detector de error para información codificada según Hamming.

Dado que el interés por el uso del método de Hamming proviene de la posibilidad de corregir los errores producidos, el paso siguiente corresponde a la determinación del bit erróneo. En función de lo detallado en el capítulo 3, la diferente distribución de los bits de datos y de paridad en las ecuaciones de Hamming permite determinar claramente el bit errado, para lo cual se deben decodificar las distintas alternativas obtenidas en los valores de E4, E2 y E1.

Una vez determinado el bit en error, el mismo debe ser corregido, lo que simplemente implica su inversión para obtener el valor correcto. Para invertir un bit solamente cuando el mismo contiene error se utilizará una compuerta XOR por cada bit a corregir, las que serán controladas por las respectivas señales indicadoras de que se produjo error en dicho bit. La salida del circuito corrector permitirá descartar los bits de paridad y recuperar la palabra de datos original.

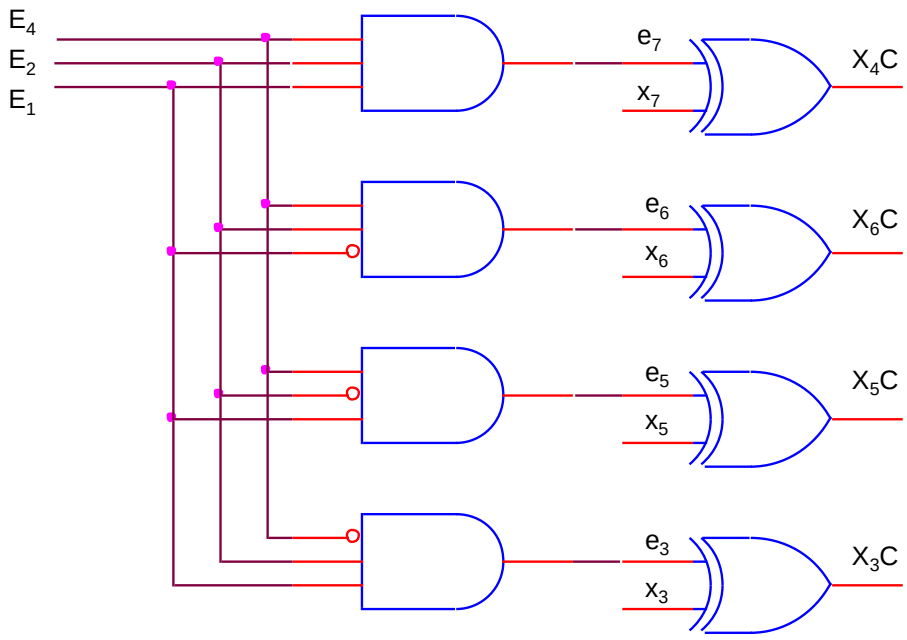


Figura 5.44.- Determinación del bit errado y su corrección.

Debe tenerse en cuenta aquí que, dado que el método de Hamming permite detectar y corregir un único error, el bloque de corrección solamente funciona cuando ese error se produjo en uno de los bits de datos. Si se produce error pero el mismo afecta a un bit de paridad, se presupone que los bits de datos no contienen errores, por lo que, al descartar los bits de paridad, se lo hace sin necesidad de corrección previa.

5.13.- Circuitos selectores (multiplexores)

Un circuito **selector**, también conocido como **multiplexor**, tiene como característica fundamental la de distinguir dos grupos de entradas binarias con características diferentes. Un primer grupo de entradas corresponde a las llamadas **entradas de datos**, por las que se ingresa información binaria, la que se conectará con la única salida del circuito selector, en función del valor binario que asuma el otro grupo de entradas, a las que se conocen como **entradas de control** o selección.

La figura 5.45 representa un esquema del circuito a nivel de bloque funcional, en tanto que la figura 5.46 permite comparar su funcionamiento con el de una llave selectora rotativa, en la que la posición del elemento rotativo conecta una y solo una de las entradas con la salida.

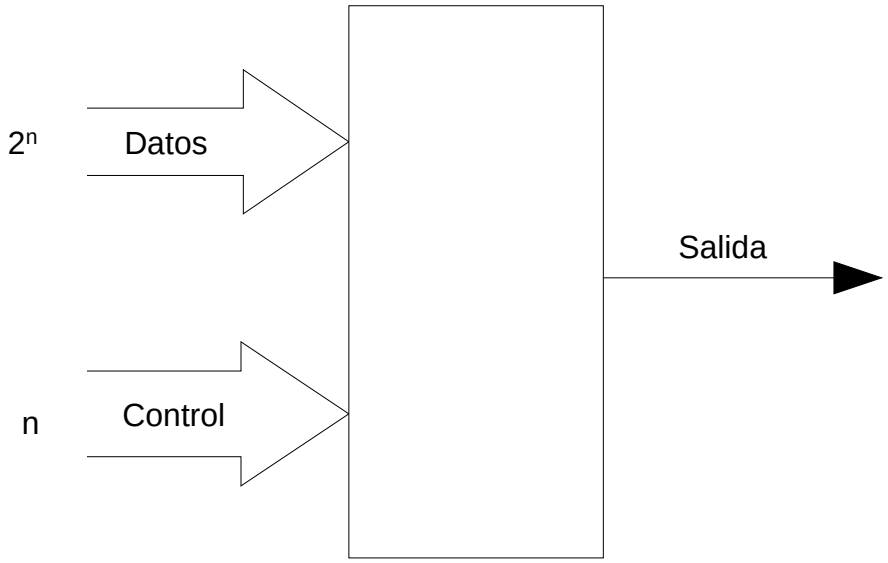


Figura 5.45.- Diagrama funcional de un circuito selector.

En el modelo propuesto, el valor binario que adoptan (como un conjunto), las entradas de control define cual es la entrada que se conecta con la salida Z. A tal fin, modificar ese valor binario se corresponde con el hecho de cambiar la posición de la llave selectora.

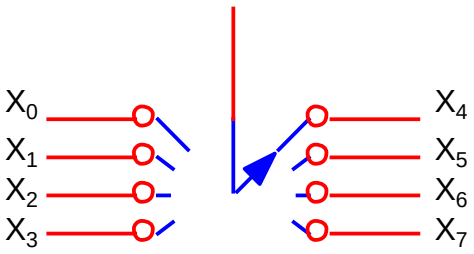


Figura 5.46.- Un modelo de llave selectora mecánica.

En un circuito selector existe una relación directa entre la cantidad de entradas de control y la cantidad de entradas de datos. En efecto, dado que el valor colocado en las entradas de control debe interpretarse como un número binario, si el circuito tiene n entradas de control se podrán obtener 2^n combinaciones de dichas entradas de control. El objetivo del circuito es lograr que con cada una de esas 2^n combinaciones de las entradas de control se logre la conexión con la salida de una entrada de datos distinta en cada caso.

Se define como **orden** o **rango** del selector a la cantidad de entradas de control que el circuito ofrece. Un selector de primer rango ofrece una entrada de control, permitiendo seleccionar entre dos entradas de datos. El selector de segundo rango tendrá dos entradas de control y cuatro de datos, y así siguiendo, el selector de orden n tendrá n entradas de control, vinculadas con 2^n entradas de datos diferentes.

Dado que el circuito selector es un circuito combinatorio, se puede plantear la salida del circuito como una función lógica tanto de sus entradas de datos como de control. Esta función lógica, a su vez, se puede representar en una tabla de verdad, la que normalmente se representa en forma simplificada, tal como se muestra en la siguiente, representativa de un circuito multiplexor de tercer orden.

C2	**C1**	**C0**	**Z**
0	0	0	X0
0	0	1	X1
0	1	0	X2
0	1	1	X3
1	0	0	X4
1	0	1	X5
1	1	0	X6
1	1	1	X7

El diagrama circuital correspondiente se presenta en la figura 5.47, y corresponde a la representación lógica de la función

$$Z = \overline{C_2}.\overline{C_1}.\overline{C_0}.X0 + \overline{C_2}.\overline{C_1}.C_0.X1 + \overline{C_2}.C_1.\overline{C_0}.X2 + \overline{C_2}.C_1.C_0.X3 +$$
$$+ C_2.\overline{C_1}.\overline{C_0}X4 + C_2.\overline{C_1}.C_0.X5 + C_2.C_1.\overline{C_0}.X6 + C_2.C_1.C_0.X7$$

la que representa el comportamiento lógico del circuito selector.

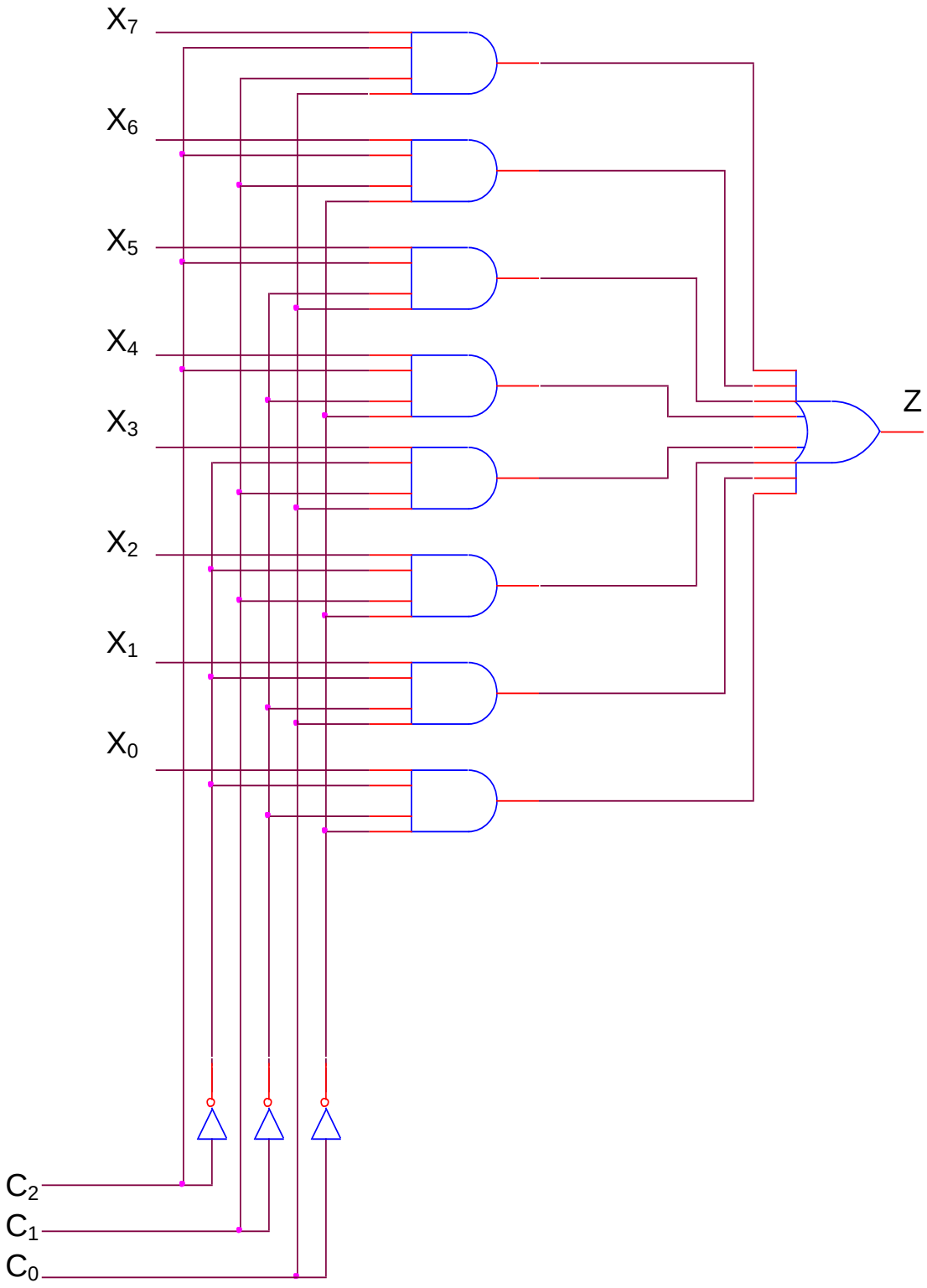


Figura 5.47.- Un circuito selector de tercer orden

El circuito multiplexor presentado puede complementarse con un circuito análogo, pero que tenga una sola entrada, n entradas de control y 2^n salidas, el que se conoce como circuito **deselector** o **demultiplexor**. (Suele también conocerse como circuito decodificador, pero dado que este último concepto es mucho más amplio, se prefiere, en este texto, no utilizar esa nomenclatura).

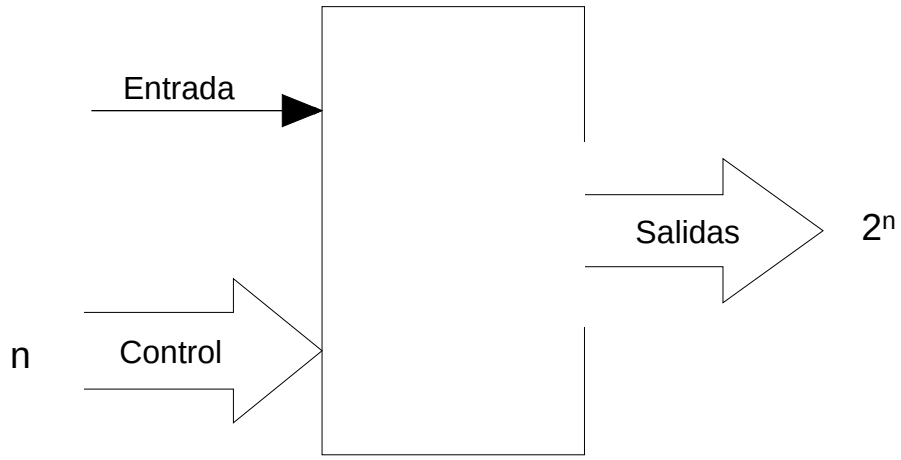


Figura 5.48.- Un circuito deselector, como bloque funcional

La figura 5.48 muestra el esquema funcional de un circuito demultiplexor de orden n, en tanto que la figura 5.49 ilustra el diagrama circuital de un demultiplexor de segundo orden, cuya tabla de verdad es la siguiente:

C1	**C0**	**Z3**	**Z2**	**Z1**	**Z0**
0	0	X	X	X	**D**
0	1	X	X	**D**	X
1	0	X	**D**	X	X
1	1	**D**	X	X	X

Esta tabla de verdad, que se corresponde con las siguientes ecuaciones lógicas, se representa a nivel circuital en la figura 5. 49.

$$Z3 = C_1.C_0.D$$
$$Z2 = C_1.\overline{C_0}.D$$
$$Z1 = \overline{C_1}.C_0.D$$
$$Z0 = \overline{C}_1.\overline{C}_0.D$$

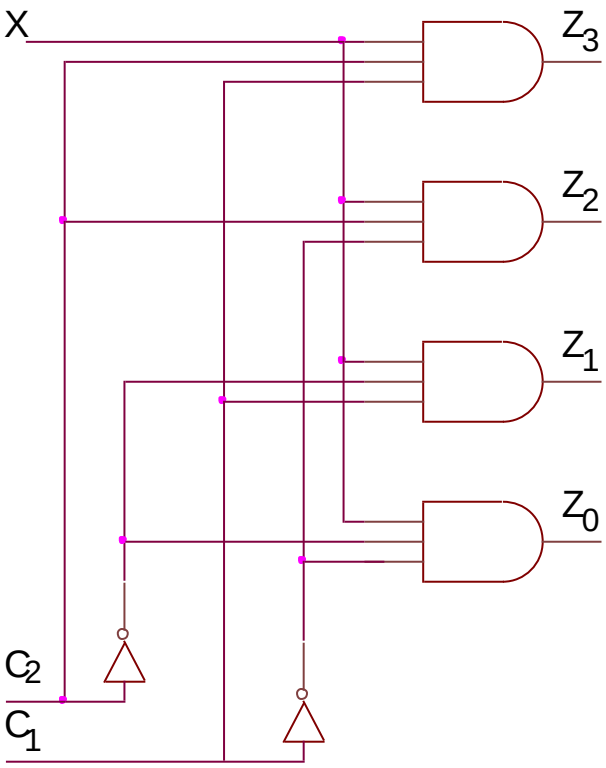


Figura 5.49.- Circuito demultiplexor de segundo orden.