Universidad Nacional Autónoma de México
Facultad de Contaduría y Administración
Sistema Universidad Abierta y Educación a Distancia

Licenciatura en Informática

Arquitectura de Computadoras

Apunte electrónico

SUAyED

COLABORADORES

DIRECTOR DE LA FCA
Dr. Juan Alberto Adam Siade

SECRETARIO GENERAL
L.C. y E.F. Leonel Sebastián Chavarría

– – – –

COORDINACIÓN GENERAL
Mtra. Gabriela Montero Montiel
Jefe de la División SUAyED-FCA-UNAM

COORDINACIÓN ACADÉMICA
Mtro. Francisco Hernández Mendoza
FCA-UNAM

– – –

AUTOR
Ing. Tomás García González

DISEÑO INSTRUCCIONAL
Lic. Dayanira Granados Pérez

CORRECCIÓN DE ESTILO
Mtro. Francisco Vladimir Aceves Gaytán

DISEÑO DE PORTADAS
L.CG. Ricardo Alberto Báez Caballero
Mtra. Marlene Olga Ramírez Chavero
L.DP. Ethel Alejandra Butrón Gutiérrez

DISEÑO EDITORIAL
Mtra. Marlene Olga Ramírez Chavero

OBJETIVO GENERAL

Al finalizar el curso, el alumno conocerá el fundamento teórico para comprender el funcionamiento de las computadoras digitales y contará con los elementos prácticos para analizar y diseñar los subsistemas lógicos que componen a éstas.

TEMARIO OFICIAL
(64 horas)

	Horas
1. Introducción	6
2. Sistemas de Numeración	8
3. Códigos	8
4. Álgebra de Boole	8
5. Circuitos combinatorios o combinacionales	10
6. Circuitos secuenciales	10
7. Memorias	8
8. Unidades funcionales	6

INTRODUCCIÓN

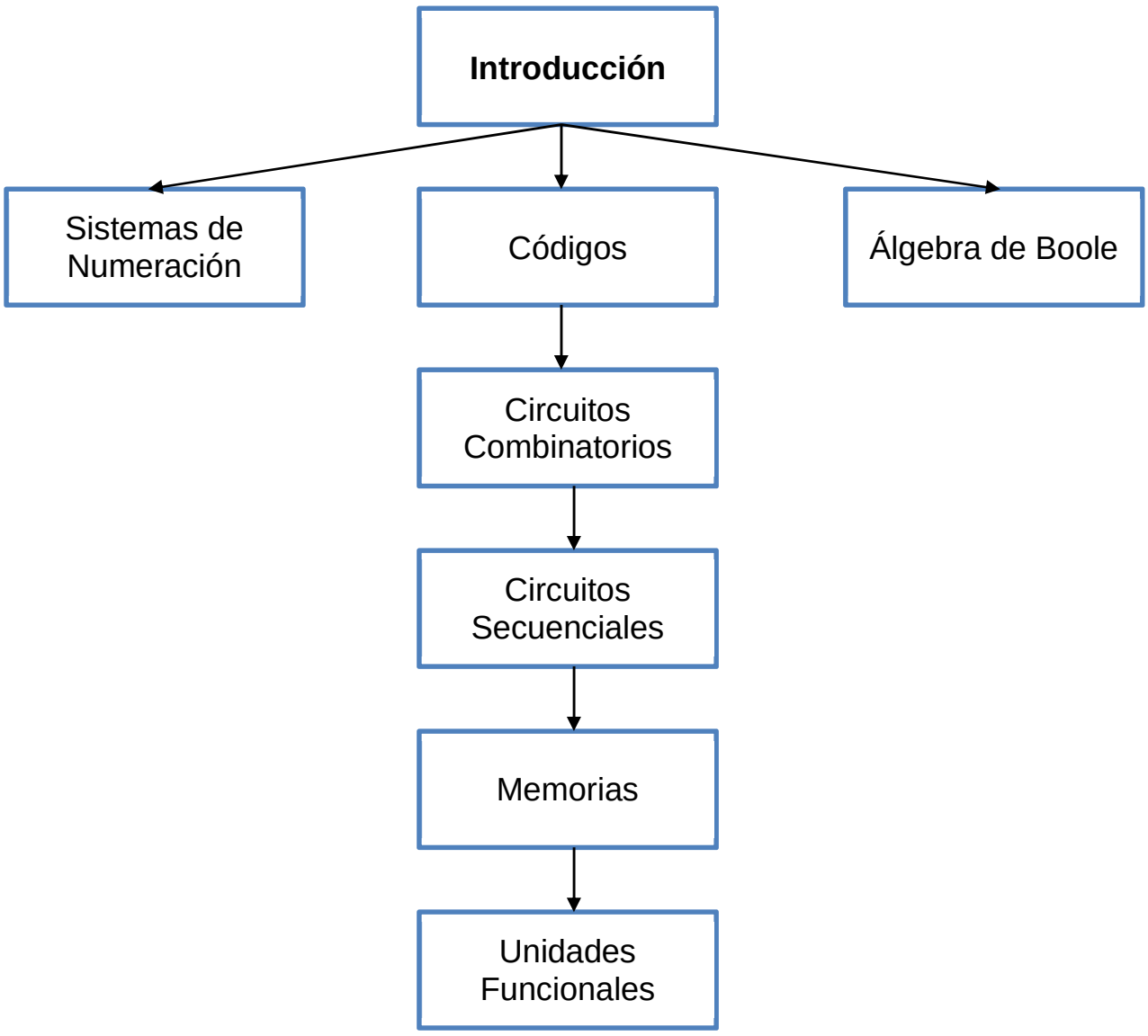
En la actualidad, la arquitectura (organización interna) de las computadoras digitales constituye un importante tema de estudio para los profesionales de la informática y sin duda alguna, su utilización e importancia aumentarán en el futuro. En esta asignatura trataremos la arquitectura básica de una computadora digital, el funcionamiento de cada uno de sus componentes y la interrelación entre sí de dichos componentes, tanto desde la parte conceptual (codificación, sistemas numéricos y álgebra de Boole) como de las partes físicas que operan mediante estos conceptos.

La unidad **uno** es introductoria y hace una presentación del modelo planteado por John von Neumann en el cual se presentan los conceptos básicos de la arquitectura de computadoras como ahora la conocemos y utilizamos, y que incluyen el uso de memoria para guardar los programas. La unidad **dos** trata de los sistemas numéricos y básicamente del uso del sistema binario y sus representaciones octal y hexadecimal. Siendo el sistema binario el utilizado por los circuitos electrónicos de una computadora, se revisan las formas de procesamiento de información numérica en este sistema. La unidad **tres** trata de la codificación o forma de representación y manejo de información, tanto numérica como alfanumérica. Se tratan asimismo los

códigos de detección de error. En la unidad **cuatro** se revisan los conceptos de lógica booleana así como de la implementación de funciones binarias. Su importancia radica en que las funciones booleanas permiten el diseño y construcción de circuitos mediante elementos electrónicos. Las unidades anteriores, de la 1 a la 4, constituyen la parte conceptual de la materia. Las unidades siguientes, la parte de aplicación, desde donde se pueden crear los módulos elementales de una computadora.

En la unidad **cinco** se presentan los diversos circuitos combinatorios, los que no incluyen la variable tiempo para su funcionamiento y a partir de los cuales se pueden crear las partes reconocibles de una computadora: memorias y unidades de control y proceso de datos. La unidad **seis**, trata de los circuitos lógicos secuenciales, en donde se involucran tiempos de proceso y señales de sincronía, así como los elementos básicos de memoria creados a partir de *flip flops*. La unidad **siete** presenta los principales tipos de memoria así como el funcionamiento, tiempos de acceso y control de las mismas. Finalmente en la unidad **ocho** se relacionan los elementos vistos hasta la unidad siete para integrar un modelo de computadora con sus partes básicas y subsistemas lógicos.

ESTRUCTURA CONCEPTUAL



Licenciatura: Informática

Unidad 1.

Introducción

OBJETIVO PARTICULAR

Al finalizar la unidad, el alumno identificará la estructura básica de las computadoras, su organización y los elementos básicos de un microprocesador.

TEMARIO DETALLADO
(6 horas)

1. Introducción

- 1.1. Estructura básica de las computadoras
- 1.2. Organización de un microcomputador (Estructura de von Neumann)
- 1.3. El Microprocesador
  - 1.3.1. Bus de direcciones
  - 1.3.2. Bus de datos
  - 1.3.3. Bus de control
  - 1.3.4. Unidad de Control
  - 1.3.5. Unidad lógica aritmética
  - 1.3.6. Registros

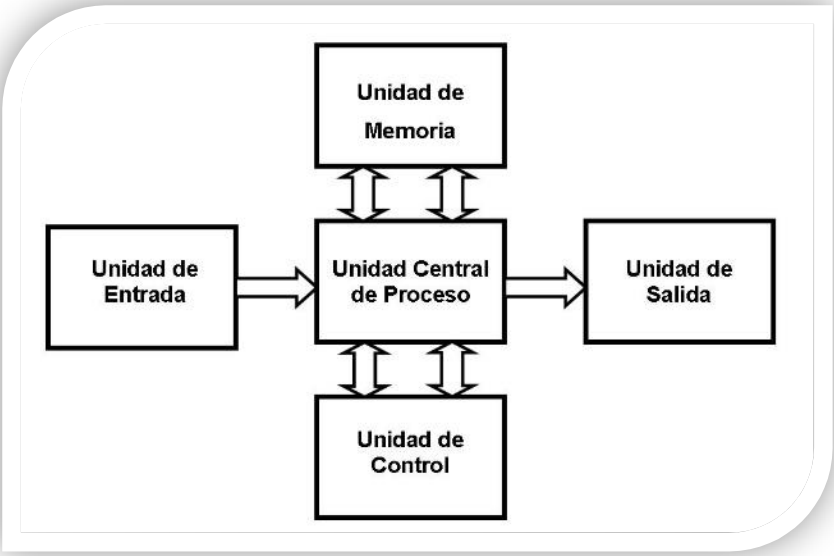
INTRODUCCIÓN

En la actualidad, la arquitectura (organización interna) de las Computadoras Digitales constituye un importante tema de estudio y una práctica cotidiana fundamental. En esta unidad explicaremos la arquitectura básica de una computadora digital, el funcionamiento de cada uno de sus componentes y la interrelación entre sí de dichos componentes.

1.1. Estructura básica de las computadoras

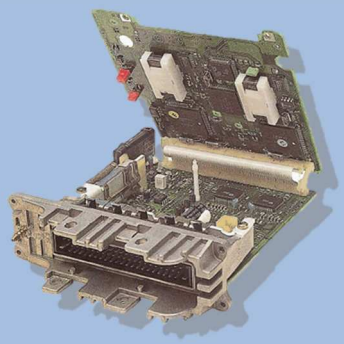
Al hablar de computadoras hoy en día se está haciendo referencia a máquinas electrónicas-mecánicas, esto es, máquinas cuyas funciones se efectúan utilizando circuitos electrónicos analógicos y/o digitales.

Definición de computadora: Una computadora es una máquina electro-mecánica que procesa información digital. Un modelo básico de una computadora consta de las siguientes partes: Unidad de Entrada, Unidad de Salida, Unidad de Central de Proceso, Unidad de Memoria y de la Unidad de Control interconectadas por buses unidireccionales y bidireccionales.



Modelo básico de una computadora

La Unidad de Entrada 	**La unidad de entrada suministra los datos y las instrucciones a la computadora. Para cada tipo de problema que se requiera resolver, solamente se necesita alimentar a la computadora con un nuevo conjunto de datos e instrucciones.**
La Unidad Central de Proceso 	Una vez introducidos en la computadora los datos y las instrucciones, esta unidad realiza con ellos diferentes cálculos, por ejemplo: operaciones aritméticas (adición, sustracción, multiplicación y división), operaciones lógicas (OR, AND y NOT) y operaciones de desplazamiento a la izquierda o a la derecha, así como también operaciones de comparación entre otras. La unidad Central de Proceso puede funcionar a velocidades más altas que la unidad de memoria, a pesar de que tiene que realizar varios pasos para completar una adición. Los resultados de las operaciones aritméticas se introducen de nuevo en la unidad de memoria para su almacenamiento, o estos resultados pueden transferirse posteriormente a una unidad de salida, que puede ser una cinta magnética, disco magnético, impresora o monitor, etc.

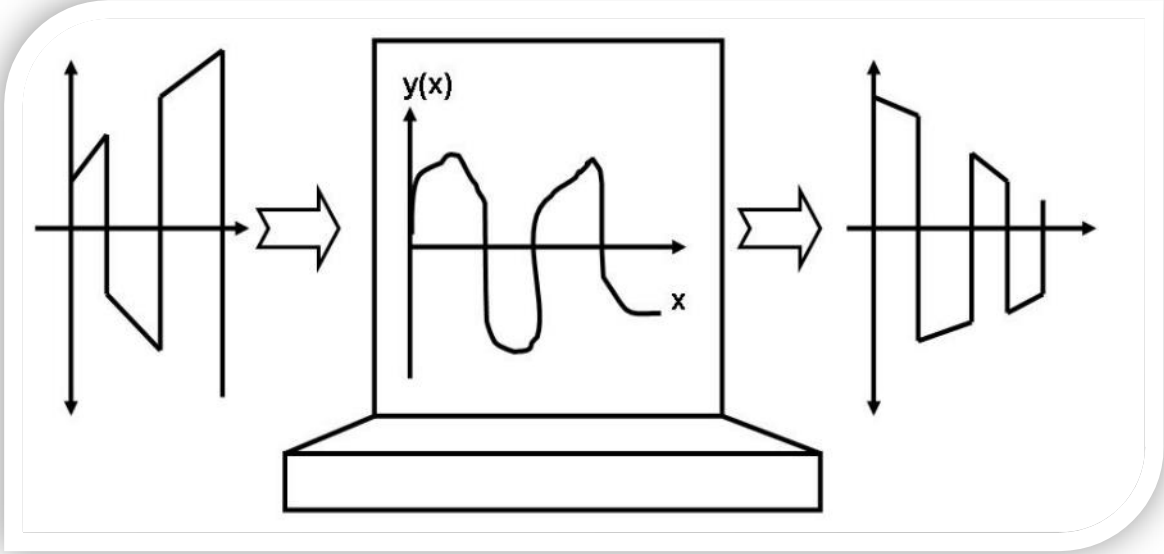
Unidad de Memoria 	Generalmente, las instrucciones (programa) y los datos se almacenan en la memoria interna de la computadora (la Unidad de Memoria), la cual se diferencia del dispositivo de entrada (que también puede ser una memoria) por su velocidad de operación. La memoria interna se diseña para almacenar y manipular cantidades relativamente pequeñas de datos, pero a velocidades muy rápidas. La velocidad de la computadora básica está limitada, por la velocidad de la memoria interna.
Unidad de control 	Esta unidad controla todas las operaciones de la computadora. Interpreta las instrucciones contenidas en la memoria y dice a las otras unidades dónde han de obtener los datos, dónde han de suministrarlos y qué operaciones han de realizar.
Unidad de salida 	La unidad de salida muestra los datos en un Tubo de Rayos Catódicos (TRC) o pantalla de cristal liquido (LCD) como dispositivo de salida. Conforme el (programador) usuario introduce algún programa vía el teclado, cada carácter se visualiza simultáneamente en la pantalla. Además de la pantalla, la computadora cuenta con otros dispositivos de salida como son el diskette, disco duro, memoria USB, cinta magnética, disco duro, la red o impresiones en papel.

Tipos de Computadoras

A partir de este modelo básico de computadora se han diseñado y construido dos tipos de computadoras:

- Computadoras analógicas
- Computadoras digitales.

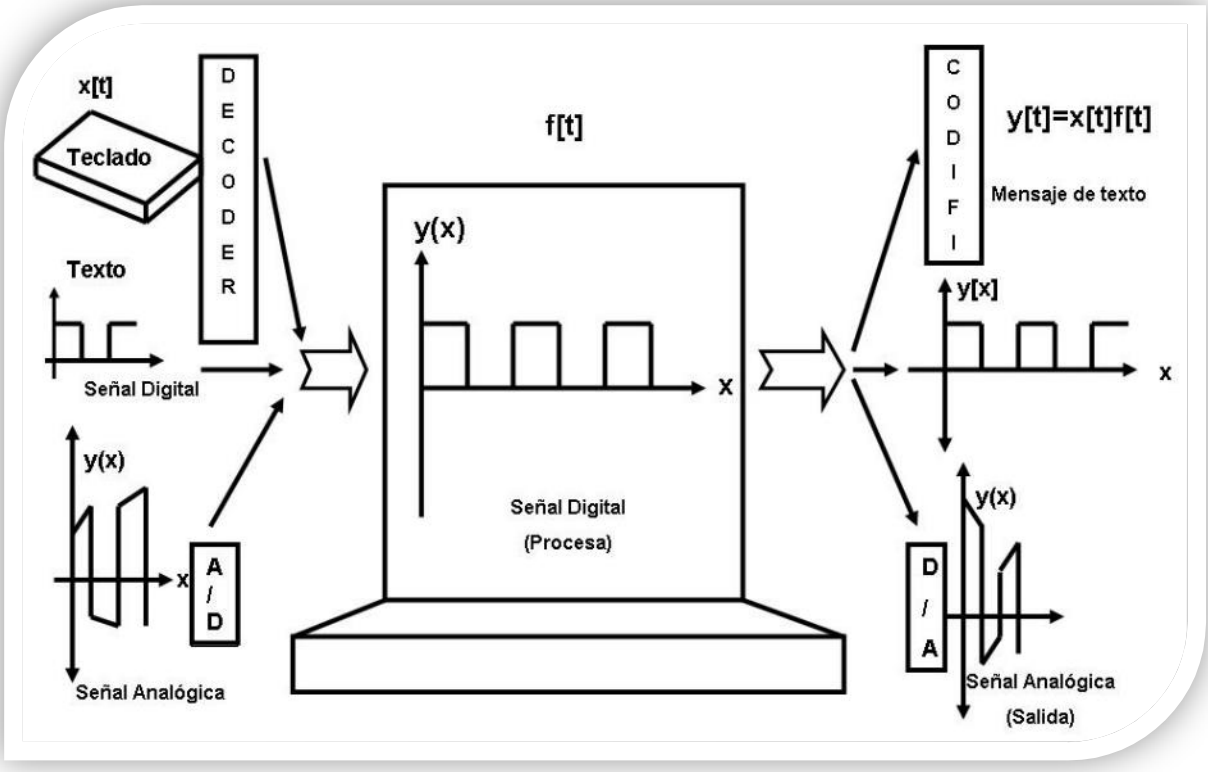
Una *computadora analógica* trabaja internamente con señales electrónicas continuas o analógicas de tal manera que mide las magnitudes de las cantidades en un circuito eléctrico que se coloca para imitar (en paralelo con, o análogo) a la ecuación del fenómeno físico investigado, la señal de entrada (por procesar) es una señal analógica y la señal de salida es una señal analógica representada en forma gráfica en un Tubo de Rayos Catódicos.



Esquema de una computadora analógica

El principal inconveniente de las computadoras analógicas es que sólo pueden alcanzar velocidades de resolución superiores a algunos centenares de ciclos por segundo o Hertz (o Hertzios). No obstante, esta reducida velocidad no significa necesariamente que la computadora analógica sea un dispositivo ineficaz. Además, este tipo de computadoras ocupan un espacio demasiado grande y su costo de mantenimiento es muy alto. La ventaja principal es su exactitud, que es mucho mejor que la de una computadora digital.

Una *computadora digital* trabaja internamente con señales electrónicas digitales o discretas, la señal de entrada (por procesar) puede ser: una señal analógica, una señal digital o un mensaje de texto, y la señal de salida puede ser una señal analógica, una señal digital y/o un mensaje de texto modificada por un programa específico previamente almacenado en la unidad de Memoria.



Esquema de una computadora digital

Las computadoras digitales son más compactas, más baratas, ocupan menos espacio, su costo de mantenimiento es muy bajo y su característica principal es su velocidad. Esta velocidad se logra debido a que están construidas con componentes electrónicos de alta velocidad. Estos componentes se utilizan para formar circuitos que realizan funciones más complejas y que operan con señales discretas (de dos niveles: nivel lógico “1” y el nivel lógico “0”). Dichas computadoras realizan a altas velocidades las operaciones aritméticas y lógicas basadas en el sistema de numeración de base 2 (ver Unidad 2). Esta característica binaria permite la utilización del álgebra de Boole en el diseño y construcción de algunos componentes que constituyen una computadora digital.

Las necesidades modernas de computación han llevado a incrementar el uso de combinaciones de computadoras analógicas y digitales, conocidas como computadoras híbridas. En esta asignatura no se considerarán las computadoras analógicas ni las híbridas, sino únicamente las digitales.

Clasificación de Computadoras digitales

Las computadoras digitales pueden clasificarse en dos categorías:

Computadoras para propósitos generales

- Una *computadora para propósitos generales* se diseña de modo que pueda ser programada para resolver una amplia variedad de problemas administrativos, de medicina, de ingeniería, etc. Las computadoras de propósito general pueden estudiar, en unos cuantos minutos, un problema de medicina, hacer una contabilidad financiera, estudiar el diseño de un proyecto de ingeniería o de jugar ajedrez con el operador (el usuario).

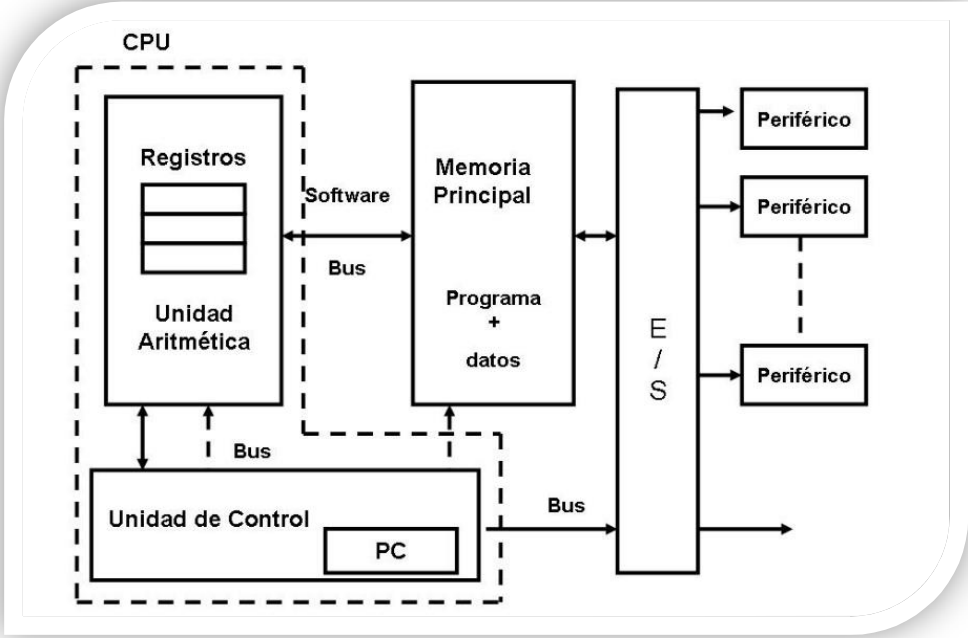
Computadoras para propósitos específicos

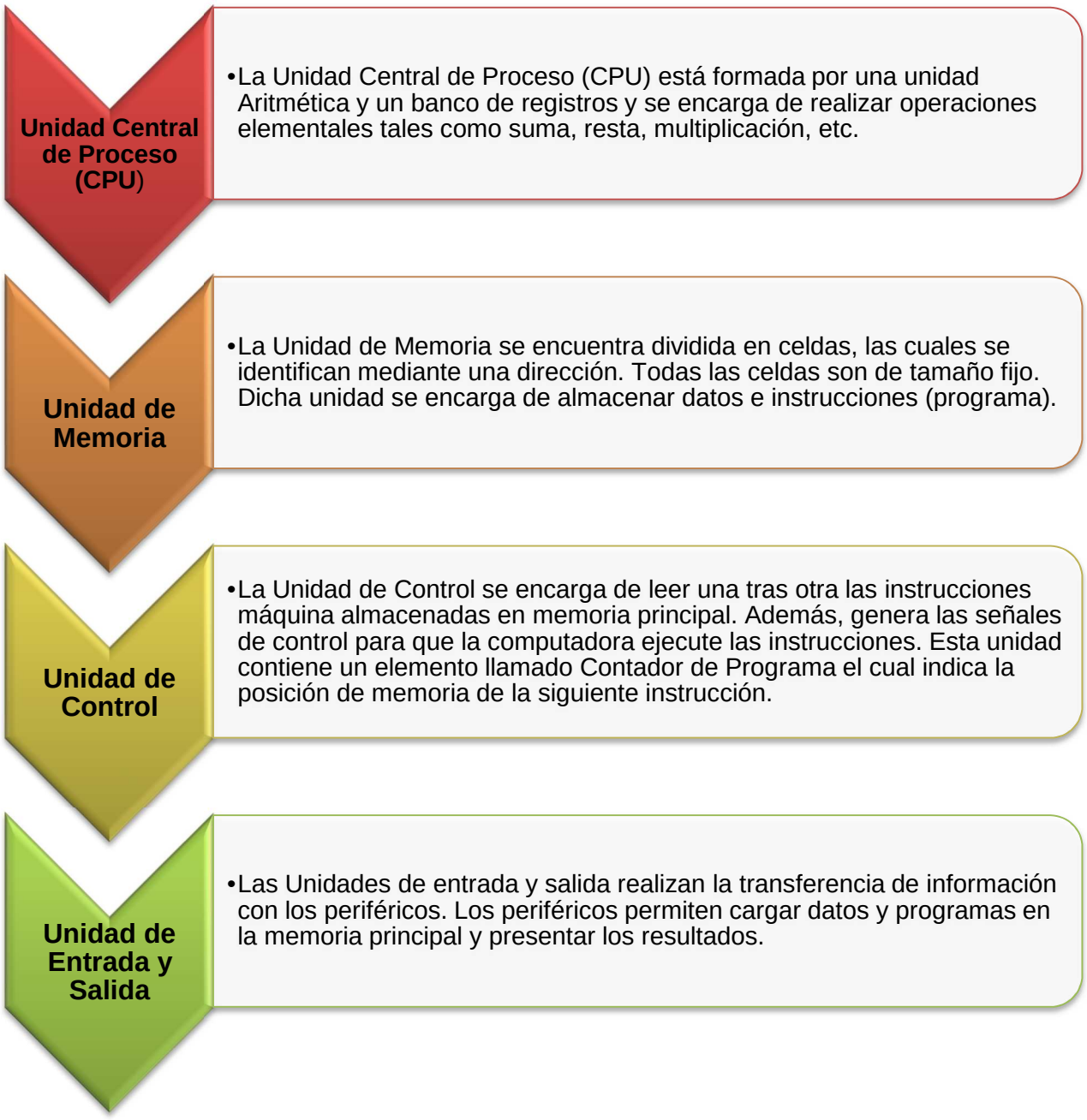
- Una *computadora de propósitos específicos* está diseñada en torno a un problema específico y es optimizada para tratar solamente con ese tipo de problema. Generalmente este tipo de computadora es de menor tamaño, más barata y más eficaz para la realización de esa función específica. Ejemplos típicos pueden ser el control de un avión, procesadores de los dispositivos de comunicación móviles o el control de una videocámara.

Ambos tipos de computadoras (de propósito general y de propósito específico) tienen básicamente la misma estructura. La diferencia reside en las unidades específicas empleadas para introducir los datos en la computadora y para proporcionar la información al exterior. A partir de este momento, haremos referencia a las computadoras digitales de propósito general.

1.2. Organización de un microcomputador (Estructura de von Neumann)

La estructura von Neumann es el modelo básico de arquitectura usado en la mayoría de las computadoras digitales actuales. Las dos principales características de esta estructura son: el uso del sistema de numeración binario y el concepto de “programa almacenado”. La estructura von Neumann está formada por:





Todas las unidades están conectadas por medio de un bus (conjunto de líneas y/o alambres por las cuales se transfiere información de cualquier dispositivo a otro) unidireccionales (un sólo sentido) o bidireccionales (en ambos sentidos) cuyo objetivo es hacer que las instrucciones, datos y señales de control circulen entre las distintas unidades de la computadora.

La estructura de von Neumann utiliza el modelo de “programa almacenado” y dicho modelo presenta las siguientes ventajas:

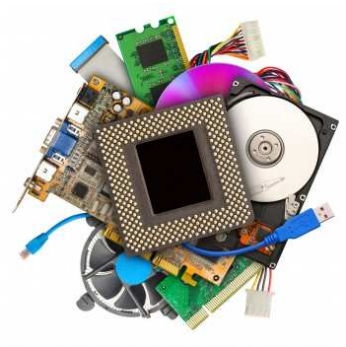
1. Se pueden ejecutar diversos programas.
2. Tiene gran velocidad de ejecución.
3. Se pueden construir programas automodificables, intérpretes, compiladores, etc.

Como se mencionó anteriormente, una de las principales aportaciones de la estructura de von Neumann es el concepto de “programa almacenado” el cual se explicará a continuación.

Las computadoras con este tipo de estructura resuelven un problema en una operación de dos fases: compilación y ejecución. Durante la fase de compilación se lee una serie de instrucciones introducidas (programa fuente), se traducen a lenguaje de máquina y se almacenan en la memoria principal. Cada instrucción se almacena en una palabra (o varias palabras, según se requiera), como una instrucción única. Durante la fase de ejecución, cada instrucción se llama en secuencia desde la unidad de almacenamiento y se retiene temporalmente en el registro de instrucción mientras se ejecuta. Esta operación de dos fases, en la cual el programa fuente se traduce y se almacena (compilación) y luego se ejecuta (ejecución) de manera automática y secuencial, se conoce como concepto de programa almacenado.

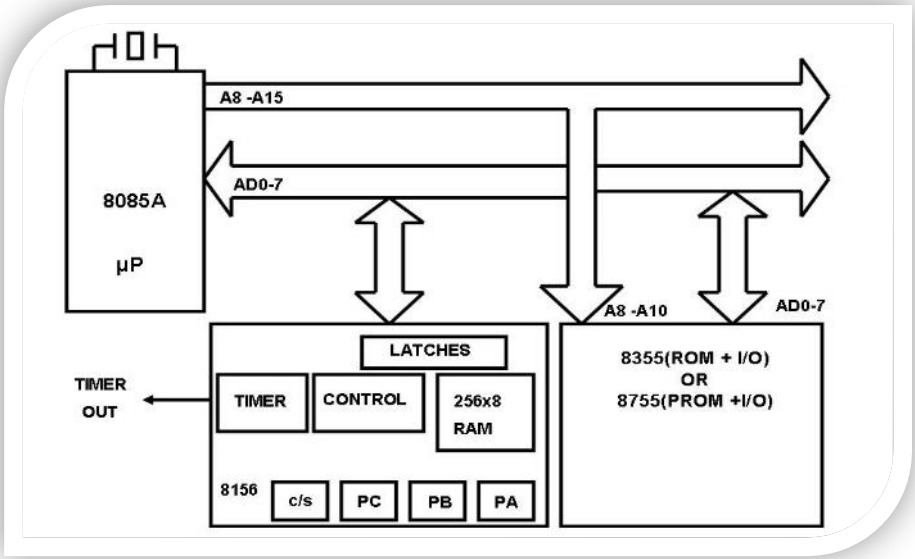
El concepto de programa almacenado permitió la lectura (almacenamiento) de un programa dentro de la memoria de la computadora y después la ejecución de las instrucciones del mismo sin tener que volverlas a escribir. Una computadora con la capacidad de “*programa almacenado*” podría ser utilizada para varias aplicaciones tan solo cargando y ejecutando el programa apropiado.

1.3. El Microprocesador



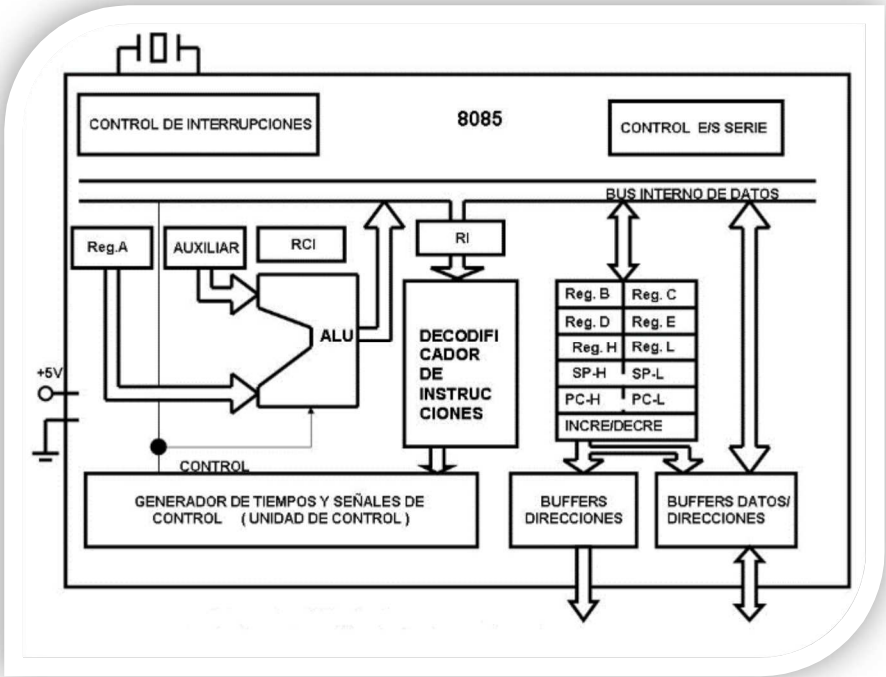
El *microprocesador* es una Unidad Central de procesos (CPU) implementada en uno o más circuitos integrados utilizando diversas tecnologías (MOS, Bipolar, etc.). La mayoría de los microprocesadores vienen implementados en un solo circuito integrado (C.I.) o “chip” de 40 o más terminales.

Se define como *microcomputadora* a la computadora digital en la que la CPU está formada por un microprocesador y sus componentes auxiliares por circuitos integrados que forman la familia del microprocesador (memoria, interfazs, decodificadores, *buffers*, puertos de entrada/salida, etc.). En la siguiente figura se ilustra un diagrama de una computadora digital basada en el microprocesador 8085A.



Sistema mínimo con base en el microprocesador 8085A

El microprocesador o CPU es la parte principal de la computadora digital y está compuesta por: Buses de Direcciones, Datos y de Control, la Unidad de Control, Unidad Lógica Aritmética (ALU, Arithmetic Logic Unit), un banco de registros, registro de banderas, etc., como lo muestra la figura, los cuales se explicarán a continuación.



Arquitectura de µP 8085

1.3.1. Bus de direcciones

El bus de direcciones es utilizado por el microprocesador para direccionar o llamar a las diversas posiciones de memoria. A través de estas líneas, el microprocesador puede acceder a la información sólo con llevar a las líneas citadas la posición que contiene la palabra por extraer. A continuación se dará la orden correspondiente de lectura, y la palabra en cuestión pasará al interior del microprocesador a través del Bus de Datos.

1.3.2. Bus de datos

La función del Bus de datos es mover o enviar los operandos o los resultados de los cálculos hacia la ALU para ser procesados desde o hacia la unidad de Memoria o hacia un dispositivo de entrada salida (puerto).

1.3.3. Bus de control

El bus de control controla las operaciones de entrada/salida tales como leer una localidad de memoria, escribir hacia una localidad de memoria, leer a partir de un periférico (puerto), escribir hacia un periférico.

1.3.4. Unidad de Control

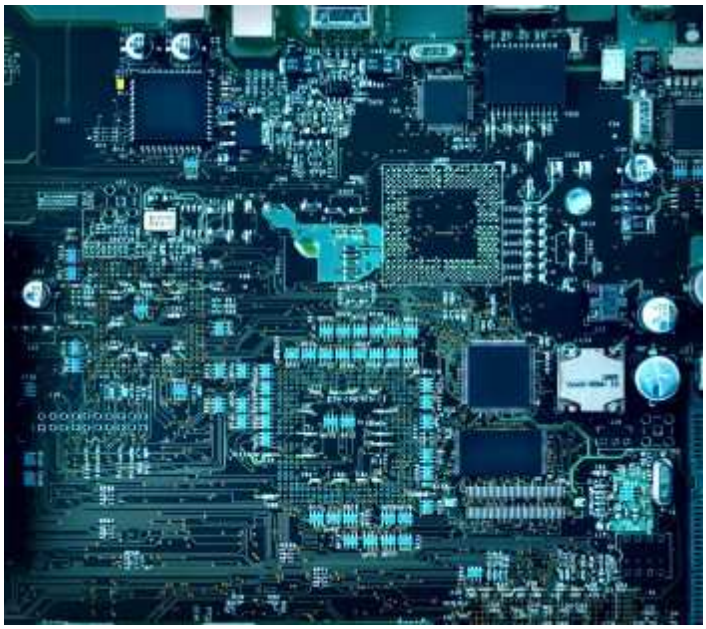
Todas las acciones dentro de una computadora deben estar sincronizadas y seguir las instrucciones de un programa. La Unidad de Control recibe las instrucciones codificadas en binario desde la memoria, y decide cuándo, cómo y qué operaciones se deben ejecutar para realizar cada instrucción e indica cuál es la que se debe ejecutar a continuación. Se considera a la Unidad de Control como el cerebro de la computadora.

→ **Desde el punto de vista de la ingeniería del software** los requerimientos parten del mismo análisis inicial del sistema. Es una etapa temprana dentro del desarrollo del sistema que se enfoca en la obtención, análisis, especificación y validación de los requerimientos para el software.

Dentro de la ingeniería de software, entendemos como **requerimientos** a "las declaraciones que identifican atributos, capacidades, características y/o cualidades que necesita cumplir un **sistema** para que tenga valor y utilidad para el usuario". [1] Es decir, un requerimiento muestra todos los elementos que son necesarios para la construcción del sistema.

→ **Desde el punto de vista del software**, un requerimiento es una serie de elementos básicos necesarios para que las aplicaciones funcionen de manera correcta, estos pueden ser, cantidad de memoria de la computadora, sistema operativo, tipo de procesador, entre otros.

1.3.5. Unidad lógica aritmética



Esta unidad es la que ejecuta realmente el trabajo de procesamiento aritmético y lógico. La ALU puede recibir datos y efectuar con ellos operaciones aritméticas, lógicas, de comparación y desplazamiento, entre otras. La ALU tiene dos registros asociados a ella para almacenar los datos y sobre los que va a realizar las operaciones: El registro acumulador y el registro auxiliar, cada uno de ellos de 8 bits. El registro principal de las ALU es el

[1] Definición de requerimiento, Diccionario web de informática, disponible en línea: http://www.alegsa.com.ar/Dic/requerimientos.php, consultado el 19/03/2011.

registro Acumulador, debido que al comienzo de una operación, el acumulador contiene uno de los operandos y al final de la operación, contiene el resultado.

Código del material 5708

1.3.6. Registros

La CPU o el microprocesador cuenta con dos tipos de registros: registros de propósito general y los registros de propósito especial. Los registros de propósito general se utilizan para el manejo de los datos, y los registros de propósito especial tienen funciones ya definidas.

Registros de propósito general

El banco de registros de propósito general está formado por los registros: B, C, D, E, H y L los cuales son de longitud de 8 bits. Estos registros tienen la característica que se pueden unir por pares para formar registros de 16 bits llamados: BC, DE y HL. El registro W-Z (o INCRE/DECRE) no es accesible por programa y tan sólo se utiliza por la unidad de control para la ejecución interna de instrucciones.

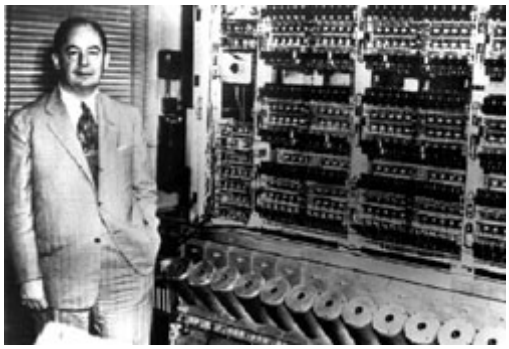
Registros de propósito especial

Los registros de propósito especial son: El Stack Pointer, el Registro de Banderas, Contador de Instrucción, Registro de Instrucción, etc. y sus funciones son:

El registro Stack Pointer	•Permite la gestión de interrupciones y subrutinas.
El registro de banderas	•Tiene información del resultado de la última operación. A la información contenida se le da el nombre de bandera y entre las cuales se pueden mencionar las banderas de Cero, Paridad, Signo, Acarreo, etc.
El contador de Instrucciones	•Apunta a la próxima instrucción que se debe ejecutar.
El registro de Instrucción	•Almacena cada instrucción conforme se llama a partir de la Unidad de Memoria e inicia su ejecución.

RESUMEN

En esta unidad revisamos la estructura y organización básica de las computadoras digitales, precisando que aunque la diferencia entre ambos conceptos es relativa, se puede diferenciarlas. La organización se refiere a la distribución y conexiones entre las diferentes unidades funcionales, mientras que la arquitectura se refiere al modelo de estructura que se sigue para integrar una computadora, en función de sus requerimientos y funciones. El modelo más utilizado es el de von Neumann el cual permitió el salto para un funcionamiento eficiente de los recursos informáticos. Las aportaciones fundamentales de este modelo es el uso de la codificación binaria de datos, el uso de memorias para almacenar, compilar y ejecutar secuencias de instrucciones y el concepto de programa almacenado el cual puede ser utilizado en diversas tareas posteriores, de esta manera los programas se pueden aplicar en múltiples procesos con solo cambiar los datos de entrada.



Von Neumann y su modelo de organización de las computadoras

El modelo anterior ha permitido el rápido desarrollo de las computadoras. La computadora es un dispositivo electromecánico que recibe datos, los procesa y entrega resultados de información procesada. Las partes básicas de una computadora son:

La unidad central de procesamiento o CPU, que es la que controla el funcionamiento y realiza las tareas de procesamiento. Su elemento esencial es el microprocesador.

Unidad central de procesamiento (CPU)

Unidad que controla el funcionamiento y realiza las tareas de procesamiento. Su elemento esencial es el microprocesador.

Memoria

Son los dispositivos electrónicos o electromecánicos que almacenan instrucciones, datos o direcciones en forma binaria.

Unidades de entrada y salida

Son los dispositivos mediante los cuales se ingresa o recoge la información.

Buses

Son los canales por los que circula la información en la computadora.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo	Páginas
Quiroga (2010)	1 Anexos	1-24 180-192
Mano (1986)	1 Anexos	1-4 372-377
Stallings (2006)	1 Anexos	8-26 438-484
Tanenbaum (2000)	1	16-19

Mano, Morris. (1986). *Lógica Digital y diseño de computadores.* México: Prentice Hall.

Quiroga, Patricia. (2010). *Arquitectura de computadoras.* México: Alfaomega.

Stallings, Williams. (2006). *Organización y arquitectura de computadores.* Madrid: Prentice Hall.

Tanenbaum, Andrew S. (2000). *Organización de computadoras. Un enfoque estructurado*. México: Prentice Hall.

Licenciatura: Informática

Unidad 2.

Sistemas de numeración

OBJETIVO PARTICULAR

Al finalizar la unidad, el alumno reconocerá los fundamentos teóricos de los sistemas numéricos, las conversiones de bases y operaciones con sistemas numéricos.

TEMARIO DETALLADO
(8 horas)

2. Sistemas de numeración

- 2.1. Sistemas numéricos posicionales
  - 2.1.1. Concepto y ejemplos de sistemas numéricos
- 2.2. Conversión entre bases
  - 2.2.1. Sistema decimal
  - 2.2.2. Sistema binario
  - 2.2.3. Sistema octal
  - 2.2.4. Sistema hexadecimal
  - 2.2.5. Sistemas de base “n”
- 2.3. Aritmética binaria y en diferentes bases
  - 2.3.1. Operaciones aritméticas
  - 2.3.2. Representación de números enteros y Reales en punto flotante
    - 2.3.2.1. Complementos a la base y a la base disminuida (a r y ar-1)
    - 2.3.2.2. Magnitud y signo
  - 2.3.3. Operaciones aritméticas con números con signo

INTRODUCCIÓN

Para la mayoría de nosotros el sistema numérico base 10 es algo “natural”, sin embargo si se establecen reglas de construcción basadas en otros dígitos, la posibilidad de contar con otras secuencias numéricas y sistemas numéricos, es posible. Las computadoras utilizan el sistema numérico binario. A diferencia del sistema decimal, el binario sólo utiliza dos dígitos: 0 y 1. Entender cómo se realizan las operaciones básicas aritméticas nos ayudará a entender cómo las computadoras procesan uno de los dos tipos de datos con los que las alimentamos, los numéricos.

El adquirir habilidad para el manejo de los sistemas binario, octal y hexadecimal, al igual que cuando se aprende a hablar otra lengua, no sólo significa cambiar la forma de expresar con diferentes señales o símbolos el mismo concepto, idea o entidad, sino adquirir la forma de construcción mental de dichas entidades y sus relaciones. Significa aprender a pensar de manera diferente, con diferente estructura y lógica. El lenguaje no sólo nombra la realidad sino que la ordena, interpreta y finalmente la transforma. Obtener la habilidad para manejar cantidades binarias significa “pensar” o por lo menos estructurar procesos como lo hace la computadora. De esta manera, si nuestra labor profesional está ligada a las computadoras, el entender la forma como interpretan y representan datos es una forma de poder comunicarnos y en consecuencia actuar sobre ellas.

En esta unidad abordamos de manera general qué son los sistemas numéricos. Todos tienen ciertas características que los hacen distinguirse como tales, es decir cumplen con algunas reglas de validación y estructura que los hicieron útiles a las culturas que los crearon. Podemos decir que por convención, es decir por acuerdo entre un grupo de personas, se reconoce a un mismo signo para una misma entidad o idea por todos aceptada. Con esto se crea la representación de ideas mediante símbolos. Los primeros elementos que le permitieron al hombre identificar “cantidades” diferentes fueron los dedos y de ahí la palabra dígito. La abstracción de la necesidad de medir

dio origen a los números, pues estos ya no están asociados necesariamente a las tareas de medición, son abstracciones. Uno de estos sistemas es el binario y es el que emplean las computadoras en sus cálculos. Aunque el sistema numérico que utilizamos generalmente es el decimal, necesitamos comprender las reglas de conversión entre ambos y los sistemas que se relacionan directamente con el binario como son el octal y el hexadecimal.

2.1. Sistemas numéricos posicionales

2.1.1. Concepto y ejemplos de sistemas numéricos

Un sistema numérico es un conjunto de símbolos y reglas de asociación con los que se pueden generar cantidades válidas para el conjunto definido por el sistema.

Cada sistema de numeración se caracteriza por su base, de manera que para la mayoría de nosotros, el sistema numérico base 10 es algo “natural”; sin embargo, no es el único que existe.

El sistema de base n más ampliamente usado para el diseño y construcción de pequeños sistemas digitales hasta una computadora digital es el sistema binario, pero para la programación de dichos sistemas digitales se utilizan los sistemas binario, octal, decimal y hexadecimal.

2.2. Conversiones entre bases

La forma más comúnmente usada para realizar la conversión entre diferentes bases es utilizando el sistema posicional. En el sistema posicional, el valor significativo asignado a cada dígito es una cantidad que está en función de su posición.

En el sistema posicional, un número N se representa en cualquier base n por la ecuación

$$N = d_{p-1}n^{p-1} + d_{p-2}n^{p-2} + \cdots + d_0 n^0 + d_{-(q-1)}n^{-(q-1)} + d_{-(q-2)}n^{-(q-2)}$$

o en su forma compacta

$$N = \sum_{i=0}^{p-1} d_i n^i + \sum_{j=1}^{q} d_j n^{-(i)}$$

Donde:

- *d* son los dígitos (coeficientes) del número
- *n* la base del sistema
- *p* el número de dígitos enteros
- *q* el número de dígitos fraccionarios

En un número cualquiera, al dígito entero que se encuentre más a la derecha se le da el nombre de "**menos significativo**" y el que se encuentre más a la izquierda el de "**más significativo**". En los dígitos fraccionales esta consideración sigue siendo válida.

La tabla de equivalencias entre diferentes sistemas de numeración, nos presenta una forma de relacionar el sistema posicional en cualquier base “n”, donde n = 2, 8, 10 y 16.

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

Equivalencias entre diferentes sistemas de numeración

Entre los sistemas de numeración más utilizados en la informática se encuentran los sistemas de numeración Decimal, Binario, Octal y Hexadecimal.

2.2.1. Sistema decimal

Emplea 10 diferentes signos (0, 1, 2, 3, 4, 5, 6, 7, 8 y 9).

2.2.2. Sistema binario

Emplea dos signos (0 y 1)

2.2.3. Sistema Octal

Emplea 8 dígitos (0, 1, 2, 3, 4, 5, 6 y 7)

2.2.4. Sistema hexadecimal

Emplea 15 signos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F)

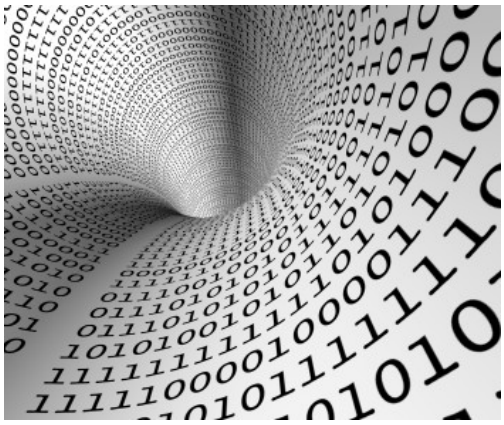
Para profundizar más sobre el tema Sistemas de numeración, descarga y estudia la presentación *Sistemas Numéricos* **(ANEXO B)**.

2.2.5. Sistemas de base “n”

Es el sistema de numeración considerado.

2.3. Aritmética binaria y en diferentes bases

En una computadora digital, las operaciones aritméticas se realizan en el sistema binario porque el diseño y construcción de circuitos lógicos (ver Unidad 5, temas 5.5 y 5.6) para realizar aritmética binaria es mucho más sencilla que para la aritmética decimal.



2.3.1. Operaciones aritméticas

Las operaciones aritméticas básicas que se efectúan con los números en base decimal, también se pueden llevar a cabo en los sistemas de numeración de base n. consulta el **(ANEXO C)**. *Aritmética binaria*, en el cual se explica cómo se realiza la suma, resta, multiplicación y división en los sistemas de numeración binaria, octal y hexadecimal.

2.3.2. Representación de números enteros y Reales en punto flotante

Los complementos se usan en las computadoras digitales para simplificar la operación de sustracción y para manipulaciones lógicas. Los complementos para cada sistema de base n son:

- El complemento a la base (n), y
- El complemento a la base disminuida (n-1)

2.3.2.1. Complementos a la base y a la base disminuida (a r y ar-1)

Dado un número positivo N en base *n* con una parte entera de *p* dígitos, el complemento de n de N se define como n^p - N para N ≠ 0 y 0 para N = 0. A continuación presentamos algunos ejemplos.

El complemento de 10 del número $(23)_{10}$ es $10^2 – 23 = 77$ (con $p = 2$).
El complemento de 10 del número $(0.37)_{10}$ es $1 – 0.37 = 0.63$.

El complemento de 2 de (1001) es

$$(2^4)_{10} - (1001)_2 = (10000)_2 – 1001 = 00111 \text{ (con } p = 4).$$

El complemento de 2 del número $(0.0101)_2$ es

$$(1)_2 – (0.0101)_2 = (0.1011)_2.$$

Como se mencionó anteriormente, el complemento a la base se utiliza para facilitar la operación de substracción. Para obtener el complemento a una base n de un número

se obtiene restando a la “base menos uno” cada uno de los dígitos del número que se va a convertir y sumándole uno al resultado de las restas. El acarreo final se ignora.

Ejemplo. Realizar la substracción decimal

Solución.

29 -12 = 17

El complemento a diez de 2 es 9-2 =7.

El complemento a diez de 1 es 9-1 =8.

Por lo tanto, el complemento a diez de 12 es 87 + 1 = 88, por lo tanto:

```
   29
+  88
  ____
  117
```

Ignorar el acarreo

Finalmente, obtenemos

```
   29
−  12
  ____
   17
```

De esta manera se realiza la resta decimal empleando el “complemento diez” de dos números en base decimal.

Complemento a la base disminuida (*n-1*)

Dado un número positivo N en base n con una parte entera de *p* dígitos y una parte fraccionaria de q dígitos, el complemento de n-1 de N se define como $n^{p}-n^{-q} – N$. A continuación presentamos algunos ejemplos.

El complemento de 9 del número $(327)_{10}$ es

$$10^3 - 1 - 327 = 672 \text{ (con } p = 3\text{), y}$$

$$10^{-m} = 10^0 = 1 \text{ (con } q = 0\text{)}$$

El complemento de 9 del número $(0.173)_{10}$ es

$$1 - 10^3 - 0.173 = 0.826 \text{(con } q = 3\text{), y}$$

$$10^p = 10^0 = 1 \text{ (con } p = 0\text{)}$$

El complemento de 1 del número $(101100)_2$ es

$$(10^6 - 1)_{10} - (101100)_2 = (111111 - 101100)_2 = (010011)_2 \text{ (con } p = 6\text{)}$$

El complemento de 1 del número $(0.0110)_2$ es

$$(1 - 2^{-4})_{10} - (0.0110)_2 - (0.1111 - 0.0110)_2 = (0.1001)_2$$

A continuación presentamos más ejemplos del Complemento a la base n aplicados a diferentes bases.

Complemento a la base 2

En nuestro caso, un caso muy interesante el complemento a la base (n = 2) aplicado a la resta binaria, la cual se realiza utilizando el complemento a dos de un número binario. El complemento a dos de un número binario se obtiene restando a 1 cada uno de los dígitos del número y sumándole 1 al resultado. Para el sistema de numeración binario, esto mismo se logra cambiando directamente los unos por ceros y los ceros por unos y sumando uno al resultado.

Ejemplo. Obtener el complemento a 2 del número $(0000\ 0101)_2$

Solución.

0000 0101 Complemento a 1 ⟶ 1111 1010

\+ 1

Complemento a 2 ⟶ 1111 1011

Ejemplo. Convertir el número 0111 1111 $(+127)_{10}$ a negativo

Solución

Número original = 01111 1111

Complemento a uno = 1000 0000

\+ 1

Complemento a dos = 1000 0001 = -127

Ejemplo. Encontrar el valor absoluto del número (1000 0010) 2

Solución

Número original = 1000 0010

Complemento a uno = 0111 1101

\+ 1

Complemento a dos = 0111 1110 = $(+126)_{10}$

Ejemplo. Obtener el complemento a dos del número $(101001)_2$

Solución.

Número original = 101001

Complemento a uno = 010110

\+ 1

Complemento a dos = 010111

Complemento a la base 8

El complemento a 8 de un número octal se obtiene restando cada uno de los dígitos del número 7 sumándole 1 al resultado.

Ejemplo. Obtener el complemento a 8 del número $(027)_8$, es decir, (obtener su negativo).

Solución.

377 350

- 027 + 1

---

350 –complemento a 7 351 ---------complemento a 8

Luego realizamos los siguiente $(351)_8 = (- 122)_8$, es decir, $(351)_8$ es el valor absoluto de $(-122)_8$.

Ejemplo. Obtener el complemento a 8 del número $(256)_8$, es decir, (obtener su valor absoluto).

Solución

377	121
- 256	+ 1
----------	________
121 –complemento a 7	122 ---------complemento a 8

Luego realizamos que $(256)_8 = (-122)_8$, es decir, $(122)_8$ es el valor absoluto de $(256)_8$.

Nota: A partir de los ejemplos anteriores observamos que del complemento de un número positivo se obtiene su negativo y del complemento de un número negativo se obtiene su valor positivo.

Complemento a la base 16

El complemento a 16 de una cantidad hexadecimal se obtiene restando cada uno de los dígitos de la cantidad a F_{16} y sumándole 1 al resultado.

Ejemplo. Obtener el complemento a 16 del número del número $(27)_{16}$ (Obtener su negativo)

Solución

FF	D8
- 27	+ 1
---------	--------
D8 Complemento a 15	D9 - Complemento a 16

posteriormente tenemos que $(-27)_{16} = (D9)_{16}$.

Ejemplo. Obtener el complemento a 16 del número del número $(D4)_{16}$ (Obtener su negativo).

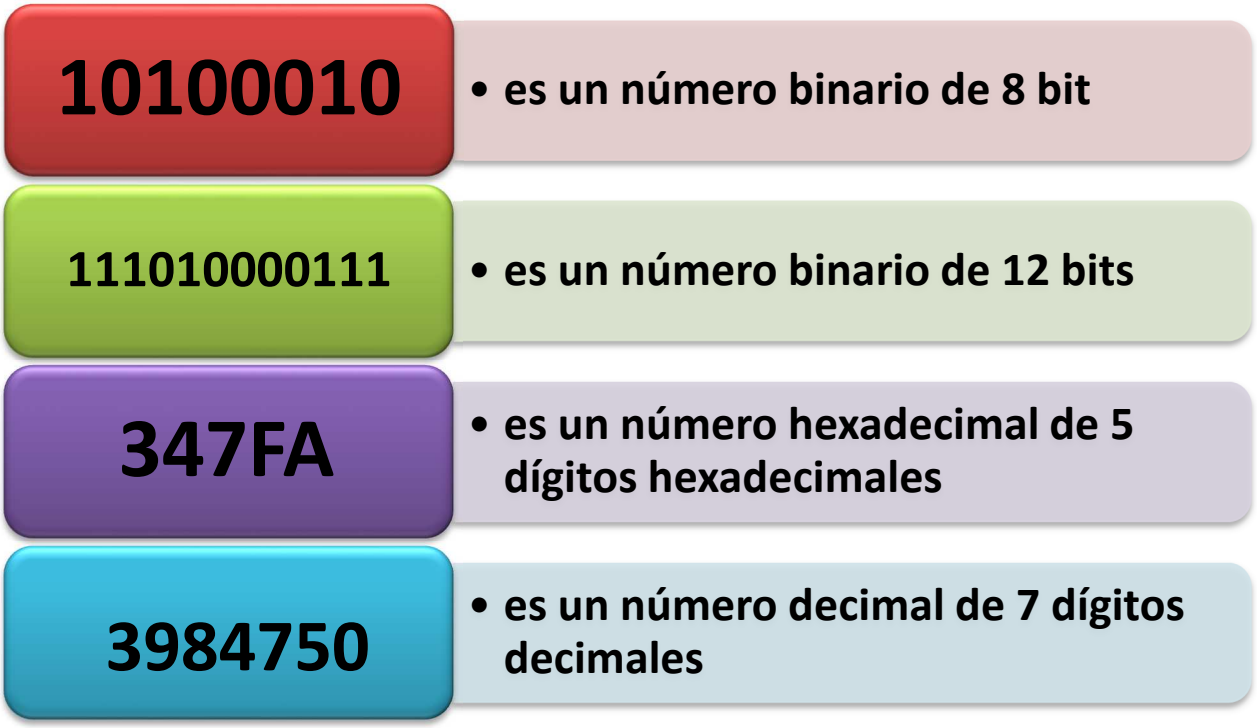
Solución.

FF		2B	
- D4		+ 1	
---		---	
2B	Complemento a 15	2C	- Complemento a 16

Finalmente realizamos $(D4)_{16} = (-2C)_{16}$

2.3.2.2. Magnitud y signo

Para la representación de los números binarios con signo debemos considerar dos conceptos: magnitud y signo. Para el caso de la magnitud podemos asociar dos ideas: la magnitud como una propiedad física que es medible (por ejemplo: longitud, superficie, volumen, peso tiempo, etc.) y la magnitud como una característica de dimensión o tamaño. Para nuestro caso, como los números binarios no están asociados a magnitudes físicas, emplearemos la magnitud como un concepto de tamaño. De manera práctica podemos decir, de cuántos bits o dígitos binarios está formada una secuencia de caracteres binarios. Por ejemplo:



2.3.3. Operaciones aritméticas con números con signo

Una computadora digital que procesa únicamente números positivos no es muy útil. La mayoría de las computadoras digitales trabajan con números signados (números positivos y números negativos). Las computadoras utilizan el método de complemento a dos para representar los números con signo (números signados).

El problema es conocer cuándo un número es negativo y cuándo es positivo. Una manera práctica que se emplea al diseñar una computadora digital es utilizar al bit más significativo del número para representar el signo. Un “1” en esa posición representa al signo negativo y un “0” representa al signo positivo.

0 xxx xxxx	representa un número positivo de 7 bits
1 xxx xxxx	representa un número negativo de 7 bits

La ventaja de usar este esquema para representar números signados es que no se necesitan circuitos digitales especiales para efectuar operaciones aritméticas. Únicamente se requiere una atención especial en la lógica de la programación con el bit de signo.

Para analizar esta representación de números signados, observemos las tres columnas de números de cuatro dígitos binarios de la Tabla Interpretación del signo. La columna 1 se forma sumando 1 a partir de 0000. Observemos que cuando se suma 1 al número 1111 se pasa a 0000. Recuerda que los números son de 4 dígitos, por eso se ignora el quinto del bit del resultado. Por lo tanto sin considerar el último bit como signo se tienen 16 números binarios desde $(0000)_2$ a $(1111)_2$.

1111	**1 111**	**0111**
1110	1 110	**0110**
1101	1 101	**0101**
1100	1 100	**0100**
1011	1 011	**0011**
1010	1 010	**0010**
1001	1 001	**0001**
1000	1 000	**0000**
0111	0 111	**1111**
0110	0 110	**1110**
0101	0 101	**1101**
0100	0 100	**1100**
0011	0 011	**1011**
0010	0 010	**1010**
0001	0 001	**1001**
0000	0 000	**1000**
Columna 1	**Columna 2**	**Columna 3**

Tabla Interpretación del signo

En columna 2 de la Tabla Interpretación del signo se forman dos grupos de 8, la diferencia es el bit más significativo. Si utilizamos al bit más significativo para el bit de signo, la parte superior de la columna 2 forma los números negativos y la parte inferior forma los números positivos.

Por otro lado hemos observado que si al número $(1111)_2$ (de la columna 2) se le suma 1 se obtiene $(0000)_2$ ó $(0)_{10}$ por lo que reconocemos al número binario 1111 con el número decimal -1, al número binario 1110 como decimal -2, etcétera y al 1000 como -8.

Con este resultado, si al grupo de los números negativos lo colocamos debajo del número 000 se obtiene la columna 3 de la tabla 2.2. Abajo del $(0000)_2$ ó $(0)_{10}$ tendremos ahora el número $(1111)_2$ ó $(-1)_{10}$ y arriba del $(0000)_2$ el número $(0001)_2$ ó $(+1)_{10}$.

Si consideramos al cero como número positivo, tendremos una cantidad de números negativos igual a la cantidad de números positivos más uno, es decir, con cuatro dígitos binarios tendremos del 1_{10} al 7_{10} (8 dígitos positivos) y del -1_{10} al -8_{10} (8 dígitos negativos).

Este método se puede extender a cualquier cantidad de dígitos binarios. La tabla Representación de números en diferentes bases con signo está formada por números binarios de 8 bits con signo.

Binario	Octal	Hexadecimal	Decimal
0111 1111	177	7F	+ 127
0111 1110	176	7E	+ 126
0111 1101	175	7D	+ 125
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
0000 0010	002	02	+ 2
0000 0001	001	01	+ 1
0000 0000	000	00	0
1111 1111	377	FF	- 1
1111 1110	376	FE	- 2
1111 1101	375	FD	- 3
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
1000 0010	202	82	- 126
1000 0001	201	81	- 127
1000 0000	200	80	- 128

Tabla Representación de números en diferentes bases con signo

A partir de la tabla Representación de números en diferentes bases con signo se observa que los números de 377 a 200 en el sistema octal y de FF a 80 en el sistema hexadecimal son negativos. El tercer dígito de los números en el sistema decimal es únicamente de dos bits, ya que estamos trabajando con ocho bits.

Operaciones aritméticas con números asignados

En esta sección presentamos las operaciones aritméticas números principalmente con signo negativo, en base 2 y base 16, ya que dichas bases son las más utilizadas.

Operaciones aritméticas en base 2

Suma binaria

Caso a) Sumando menor que el sustraendo

Ejemplo. Realice la suma siguiente 17 + -29

Solución. Para resolver esta suma lo primero que tenemos que realizar es calcular el complemento a dos del número -29 y luego aplicar las reglas de la suma y en caso de que exista un bit de acarreo se debe ignorar.

(29)10= (1 1 1 0 1)

Complemento a 1

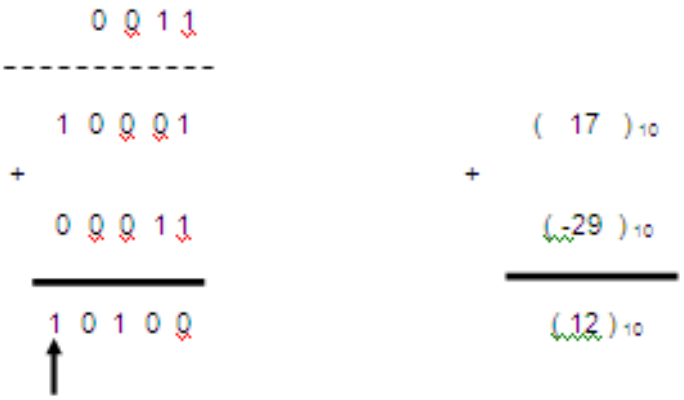
0 0 0 1 0

\+ 1

0 0 0 1 1

Finalmente realizamos la operación de suma

Comprobación



Bit de Acarreo

Caso b) Sumando mayor que el sustraendo

Ejemplo. Realice la operación $(15)_{10}$ + $(-11)_{10}$

Solución.

Comprobación

$$\begin{array}{r} 0000 \quad 1111 \\ + \\ 1111 \quad 0101 \\ \hline 10000 \quad 0100 \end{array} \qquad \begin{array}{r} (+15)_{10} \\ + \\ (-11)_{10} \\ \hline (+04)_{10} \end{array}$$

Para el caso de números fraccionarios

Ejemplo. Realice la operación $(5.5)_{10} + (3.25)_{10}$ indicada

Solución

$$\begin{array}{r} 0101.1000 \\ + \\ 0011.0100 \\ \hline 1000.1100 \end{array} \qquad \begin{array}{r} (5.5)_{10} \\ + \\ (3.25)_{10} \\ \hline (8.75)_{10} \end{array}$$

Ejemplo. Realice la operación $(5.75)_{10} – (3.5)_{10}$ indicada

Solución

$$\begin{array}{r} 0011.0100 \\ \hline 1000.1100 \end{array} \qquad \begin{array}{r} (3.25)_{10} \\ \hline (8.75)_{10} \end{array}$$

Ejemplo. Realice la operación $(5.75)_{10} – (3.5)_{10}$ indicada

Solución.

$(3.50)_{10}$ = 0011.1000

1100.0111 ---------- Complemento a 1

\+ 1 ---------- Complemento a 2

$(-3.50)_{10}$ = 1100.1000

Luego realizamos la suma

Decimal	Binario
$(5.75)_{10}$	0101.1100
\+ $(-3.50)_{10}$	\+ 1100.0100
---	---
$(2.25)_{10}$	1 0010.0100

Este bit de acarreo se ignora

Nota: Observa cómo se obtiene el complemento a dos para números fraccionarios.

Multiplicación binaria

Ejemplo. Realizar la operación 10 por -8

Solución.

Decimal	Binario	Octal	Hexadecimal
10	0000 1010	012	0A
x (-8)	x 0000 1000	x 008	x 0C
-80	0000 0000	120	78
	0 0000 000		
	00 0000 00		
	000 0101 0		
	000 0101 0000		
	1010 1111	257	AF
	+ 1	+ 1	+ 1
Complemento a 2	(1011 0000)	260	B0

División Binaria

Ejemplo. Realizar la operación de 36/3 en el sistema binario.

Solución.

12	00001100	14	C
3 / 36	11/ 001000100	3 / 44	3/ 24
-3	1101	-3	-24
06	00011	14	0
- 6	1101	-14	
0	00000	0	

Apoyo

$(3)_{10} = (00000011)_2$ $(-3)_{10} = (11111101)_2$

Nota: En el sistema octal y hexadecimal los valores negativos se obtienen con los complementos a 8 y a 16 respectivamente de los valores positivos. El complemento a 8 de un número octal se obtiene restando cada uno de los dígitos del número 7 sumándole 1 al resultado. El complemento a 16 de una cantidad hexadecimal se obtiene restando cada uno de los dígitos de $(F)_{16}$ y sumándole 1 al resultado.

Nota: A partir de los ejemplos anteriores observamos que del complemento de un número positivo se obtiene su negativo y del complemento de un número negativo se obtiene su valor positivo.

Operaciones aritméticas en base 16

Suma en base 16

Ejemplo. Realizar la suma de $(–13)_{10} + (-11)_{10}$ en base hexadecimal

Solución.

$$\begin{array}{r} -13_{10} \\ -11_{10} \\ \hline -24_{10} \end{array}$$

$(+13)_{10} \Rightarrow (0D)_{16}$ $(+11)_{10} \Rightarrow (0B)_{16}$

Sacamos el complemento a 15

```
  F F
-
  0 D
-------
  F 2
+   1
-------
  F 3
```

Sacamos el complemento a 15

```
  F F
-
  0 B
-------
  F 4
+   1
-------
  F 5
```

Finalmente, realizamos la suma

```
    F 5
+
    F 3
-------
  1 E 8
```

Bit de signo

Comprobación

La comprobación se realiza obteniendo el valor absoluto del resultado de la suma de la manera siguiente

```
    F 3
-
    E 8
-------
  1 7
+
    1
-------
  1 8
```

Con lo que $18_{16} = 24_{10}$

Ejemplo. Realizar la suma de $(-19957)_{10} + (10999)_{10}$ en base hexadecimal

Solución.

$$(-19957)_{10}$$
$$+$$
$$(-10999)_{10}$$
$$(-30956)_{10}$$

$(+19957)_{10} \Rightarrow (4DF5)_{16}$ $(+10999)_{10} \Rightarrow (2AF7)_{16}$

Complemento a 15 del número (-19957) 10

$$\begin{array}{r} F\ F\ F\ F \\ - \\ 4\ D\ F\ 5 \\ \hline B\ 2\ 0\ A \\ + \\ 1 \\ \hline B\ 2\ 0\ B \end{array}$$

Complemento a 15 del número (-10999) 10

$$\begin{array}{r} F\ F\ F\ F \\ - \\ 2\ A\ F\ 7 \\ \hline D\ 5\ 0\ 8 \\ + \\ 1 \\ \hline D\ 5\ 0\ 9 \end{array}$$

Realizamos la suma de los complementos

$$\begin{array}{r} B\ 2\ 0\ B \\ + \\ D\ 5\ 0\ 9 \\ \hline 1\ 8\ 7\ 1\ 4 \end{array}$$

Bit de signo

Comprobación

Para comprobar el resultado obtenemos el valor absoluto del resultado de la operación de la manera siguiente:

```
  F F F F
-
  8 7 1 4
-------------
  7 8 E B
+
        1
-------------
  7 8 E C
```

Ejemplo. Realizar la suma

Solución.

$$-19957_{10}$$
$$-10999_{10}$$
$$-30956_{10}$$

$(+19957)_{10} \Rightarrow (4DF5)_{16}$ $(+10999)_{10} \Rightarrow (2AF7)_{16}$

complemento a 15 del número 4DF5

```
  F F F F
-
  4 D F 5
-------------
  B 2 0 A
+
        1
-------------
  B 2 0 B
```

complemento a 15 del número 2AF7

```
  F F F F
-
  2 A F 7
-------------
  D 5 0 8
+
        1
-------------
  D 5 0 9
```

Realizamos la suma de los complementos

```
  B 2 0 B
+
  D 5 0 9
---------
1 8 7 1 4
```

Bit de signo

Finalmente la suma de

$$-19957_{10} + -10999_{10} = -30956_{10}$$

$$B\,2\,0\,B_{16} + D\,5\,0\,9_{16} = 8\,7\,1\,4_{16}$$

Comprobación

Para comprobar el resultado, obtenemos el valor absoluto del resultado de la suma de la manera siguiente

```
  F F F F
-
  8 7 1 4
---------
  7 8 E B
+
        1
---------
  7 8 E C
```

RESUMEN

La unidad está dividida en dos temas. La estructura algebraica para la construcción de cualquier número en un sistema base n que se puede aplicar a cualquier sistema numérico. Tenemos habilidad para reconocer de inmediato cualquier cifra en el sistema decimal sin necesidad de estar elaborando la notación extendida, esto se debe a que desde niños hemos estado en contacto con su construcción empleando el número 10 como base. Sin embargo, como se vio en la unidad anterior, la forma de representar cantidades y manejarlas en las computadoras es mediante los números binarios.

Las computadoras utilizan dispositivos electrónicos que pueden mantener dos estados de voltaje, alto y bajo, en consecuencia el sistema binario, aunque ya se había desarrollado desde el siglo XVII, se empezó a aplicar a las computadoras hasta los años cuarentas del siglo pasado. Es importante en nuestro curso entender por lo tanto el funcionamiento de este sistema.

En el primer tema se describió la forma de conversión del sistema decimal al binario y por extensión, del sistema decimal a cualquier sistema de base diferente. El manejo de números binarios no nos es familiar, sin embargo el sistema hexadecimal, derivado fácilmente del binario, nos es más informativo. Le conversión entre estos dos sistemas y el octal es muy sencilla por lo que también se abordaron en este tema. Finalmente se explicó la conversión inversa, cómo pasar una cifra en base n a base diez y específicamente de base 2, 8 y 16 a base 10.

En el segundo tema se trató el fundamento de las operaciones aritméticas en base 10 y desde ahí se explicaron las diferentes operaciones básicas en diversos sistemas, para ello se desarrollaron varios ejemplos en diferentes bases. Las operaciones explicadas fueron suma, resta, multiplicación y división.

Adicionalmente se presentaron los conceptos de números signados, su relación y representación a partir del concepto ‘complemento’. Estos dos conceptos son importantes debido a que la operación de resta en el sistema binario y por lo tanto en las computadoras se realizan empleando el concepto de complemento a la base n y a la base disminuida, n menos uno.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo	Páginas
Quiroga (2010)	1	1-14 11-16 180-192
Mano (1986)	1	4-10 18 26
Stallings (2006)	Apéndice A	725-731

Mano, Morris. (1986). *Lógica Digital y diseño de computadores*. México. Prentice Hall Hispanoamericana.

Quiroga, Patricia. (2010). *Arquitectura de computadoras*. México: Alfaomega.

Stallings, Williams. (2006). *Organización y arquitectura de computadores*. Madrid: Prentice Hall.

Unidad 3.

Códigos

OBJETIVO PARTICULAR

Al finalizar la unidad, el alumno podrá realizar representaciones de cantidades en diferentes códigos y secuencias y generar códigos de detección de errores.

TEMARIO DETALLADO
(8 horas)

3. Códigos

- 3.1. Códigos numéricos
  - 3.1.1. Binario
  - 3.1.2. BCD
  - 3.1.3. Exceso3
  - 3.1.4. Gray
- 3.2. Códigos alfanuméricos
  - 3.2.1. ASCII
  - 3.2.2. BCDIC
  - 3.2.3. EBCDIC
- 3.3. Códigos por detección de error
  - 3.3.1. Paridad par
  - 3.3.2. Paridad impar

INTRODUCCIÓN

Las computadoras digitales emplean el sistema binario para representar y manipular cualquier información. Lo anterior implica que las señales que se manejan en el mundo real, señales analógicas, tienen que ser representadas en los sistemas informáticos empleando solamente los símbolos uno y cero. Existen diferentes formas de codificar estas señales, sin embargo podemos especificar que sólo tenemos dos tipos de caracteres en las computadoras, numéricos y alfanuméricos. En el primer caso ya se ha visto en la unidad dos, la forma de manipulación de estos. Sin embargo también los caracteres numéricos se pueden representar (codificar) en diferentes formas pero manteniendo la estructura y formas de manipulación de un sistema decimal.

En la primera parte de esta unidad se describen los códigos binarios, BCD, X3 y gray. En la segunda se describen las formas de codificación de los caracteres alfanuméricos: ASCII, BCDIC y EBCDIC; finalmente en la tercera se explica el funcionamiento de los códigos de detección de error de paridad.

3.1. Códigos numéricos

Una computadora digital trabaja internamente con números discretos, generalmente las unidades de Entrada/Salida (a través de sus periféricos) reciben o envían información en forma decimal. Dado que la mayor parte de los circuitos lógicos solo aceptan señales discretas, los números decimales se pueden codificar en términos de señales binarias mediante diversos códigos como son: Código Binario, BCD, Exceso-3, etc.

3.1.1. Binario

El código binario es quizá uno de los códigos más utilizados en una computadora. La razón de esto obedece a la facilidad que representa construir cualquier “sistema” basado solamente en 2 dígitos: 0 y 1, los cuales, dentro de un “sistema”, son interpretados de diversas maneras, tales como: SÍ o NO, VERDADERO o FALSO, niveles Alto y Bajo de voltaje, OFF y ON, etc. Como se ve, el hecho de manejar tan sólo 2 dígitos hace posible la versatilidad que este código ha cobrado hoy en día. La tabla Código Binario especifica la codificación de caracteres numéricos utilizando el código binario.

Decimal	Código Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110

7	0111
8	1000
9	1001

Código Binario

3.1.2. BCD

El código BCD se utiliza en las computadoras para representar los números decimales 0 a 9 empleando el sistema de numeración binario. Los números representados en código BCD se escriben utilizando ceros y unos. La tabla Código BCD especifica la codificación de caracteres numéricos.

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

Código BCD

A partir de la tabla Código BCD, se observa que este código requiere el empleo de un carácter binario de cuatro posiciones (cuatro bits) para especificar el carácter de un dígito decimal. Evidentemente, este código es mucho menos eficiente que el sistema decimal, pero presenta la ventaja de especificar los caracteres mediante las cifras 0 y 1, que constituyen el lenguaje del computador, por lo que el código BCD puede ser utilizado en una computadora. Algunos ejemplos de representación de números decimales en este código son:

Decimal	BCD			
22			0001	0010
35			0011	0101
671		0110	0111	0001
2579	0010	0101	0111	1001

Puede verse que cada cifra decimal requiere un equivalente de cuatro bits codificado o “nibble” (palabra de 4 bits) en binario. Para especificar un número, el código BCD requiere más posiciones que el sistema decimal. Pero, por estar en notación binaria, resulta extremadamente útil. Otro punto que debe tenerse presente es que la posición de cada bit, dentro de los cuatro bits de cada cifra, es muy importante (como sucede en todo sistema de numeración posicional). Puede especificarse la ponderación de cada una de las posiciones y algunas veces se emplea para indicar la forma de codificación. El peso de la primera posición (situada a la derecha es 2^0=1, el de la segunda, 2^1=2; el de la tercera, 2^2= 4 y el de la cuarta, 2^3=8. Leyendo el número de la izquierda a derecha, la ponderación es 8-4-2-1, por lo que este código se denomina también un código 8421.

Cabe aclarar, que este código (8421) no es el mismo que los números binarios, consideremos los casos siguientes:

Diez en Binario es 1010	Diez en BCD es 0001 0000.
Dieciséis en Binario es 10000	Dieciséis en BCD es 0001 0110.

La confusión entre los códigos BCD y Binario se origina debido a que son exactamente iguales las nueve primeras cifras en BCD y en Binario. Después, los números son completamente diferentes.

La característica principal de la codificación BCD es análoga a la de de los números en el sistema octal; puede ser reconocida y leída fácilmente. Por ejemplo, compárense las representaciones Binaria y BCD leyendo los números en cada una de sus formas.

Decimal	Binario	BCD
141	10001101	0001 0100 0001
2179	100010000011	0010 0001 0111 1001

Sin embargo, cuando se utiliza esta forma de codificación en operaciones aritméticas se presentan dificultades adicionales. Veamos lo que sucede cuando se suman 8 y 7 en ambas formas (Binario y BCD).

```
   8          1000          1000
+  7       + 0111        + 0111
-----      --------      --------
  15          1111          1111 → No es un carácter aceptable
                                   en BCD (15 es 0001 001)
```

Para realizar operaciones aritméticas con el código BCD se necesitan sumadores especiales. Cuando se desea la propiedad de fácil reconocimiento y la manipulación aritmética, puede utilizarse un código modificado.

Los conceptos anteriores también son aplicables a números decimales con fracciones.

Por ejemplo exprese el número decimal 7324.269 en BCD.

7 - 0111
3 - 0011
2 - 0010
4 - 0100
2 - 0010
6 - 0110
9 - 1001

Por tanto

0111 0011 0010 0100 . 0010 0110 1001

7 3 2 4 . 2 6 9

Finalmente, las razonas del empleo de este código son:

a) Ahorra espacio al representar un número decimal,

b) Permite trabajar en forma binaria con un mínimo de espacio.

3.1.3. Exceso-3

Este código se deriva del BCD y se obtiene sumando 3 al mencionado código. Este código es particularmente útil en la ejecución de operaciones aritméticas usando complementos. Al igual que el código BCD ponderado, este código sirve para representar números decimales a binarios, por grupos de 4 bits por cada dígito decimal.

La tabla Código de Exceso en tres muestra las cifras decimales 0-9, el código BCD y el código de exceso en tres, que es una forma modificada del código BCD.

Decimal	BCD	Exceso en Tres
0	**0000**	0011
1	**0001**	0100
2	**0010**	0101
3	**0011**	0110
4	**0100**	0111
5	**0101**	1000
6	**0110**	1001
7	**0111**	1010
8	**1000**	1011
9	1001	1100

Código de Exceso en tres

Como su nombre lo indica, cada carácter codificado en exceso en tres es tres unidades mayor que en BCD. Así, seis ó 0110 se escriben 1001, que es nueve en BCD. Ahora bien, 1001 solamente es nueve en BCD, en el código de exceso en tres, 1001 es seis.

Ejemplo

Decimal	Exceso en tres
2	0101
25	0101 1000
629	1001 0101 1100
3271	0110 0101 1010 0100

El código de exceso en tres facilita la operación aritmética, es decir,

$$\begin{array}{r} 3 \\ +\ 9 \\ \hline 12 \end{array} \qquad \begin{array}{r} 0110 \\ +\ 1100 \\ \hline 1\ \ 0010 \\ + \\ 0011\ \ 0011 \\ \hline 0100\ \ 0101 \end{array}$$

Se lee 0100 0101 ó 12 (exceso en tres).

Existen algunas reglas especiales aplicables a la suma (como la adición de 3 a cada uno de los números del ejemplo anterior), pero estos pasos se realizan fácilmente, y de modo automático, en la computadora, haciendo del código de exceso en tres muy conveniente para las operaciones aritméticas. En el código de exceso en tres, el reconocimiento de la representación de las cifras no es directo, ya que al leer cada dígito debe restarle mentalmente en tres, si bien ello resulta más fácil que la conversión de números grandes representados en el sistema binario puro.

Ya hemos indicado que el BCD es un código ponderado; el de Exceso en Tres no lo es. Un bit de la segunda posición (2) de BCD representa un 2. En el código de exceso en tres, un bit situado en una cierta posición no indica la adición de un valor numérico al número. Por ejemplo, en BCD, 0100 es 4 y al sumarle el bit 2 se añade un 2,

resultando el número 0110, o sea, dos unidades mayor. En el código de Exceso en tres, 0111 representa la cifra 4 y la cifra 6 es 1001, no existiendo un cambio numérico sistemático.

3.1.4. Gray

El código Gray es uno de los códigos cíclicos más comunes y esto es debido a las siguientes características:

- Cambia solamente uno de sus bits al pasar a la siguiente posición, es decir, el cambio entre dos números progresivos es de un bit.

Por esta característica este código es empleado con frecuencia en la detección de errores, como veremos más abajo en la parte 3.

- Facilita la conversión a la forma binaria.

Además, este código suele emplearse en codificadores de desplazamiento angular con el eje óptico o mecánico, es decir, emplea un tipo de rueda codificadora que presenta posiciones sucesivas, cubriendo la superficie de un disco, cada una de las cuales está representada por una nueva palabra; el código de Gray admite ambigüedad en una posición.

En la tabla Código Gray se muestra la equivalencia para los números decimales 0 a 15, del código Gray, el sistema decimal y del binario puro. A todo número binario le corresponde una representación en el código Gray, por lo que la lista de equivalencias indicadas sólo tiene carácter ilustrativo.

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

Código Gray

A partir de la tabla Código Gray se puede observar que entre cada dos palabras cualesquiera sucesivas del código Gray, solamente cambia un bit. Esto no ocurre en el sistema binario; al pasar del decimal 7 al 8, cambian los cuatro bits del código binario, mientras que solamente cambia un bit en el código Gray. Al pasar de decimal 9 al 10, y el 2 de 0 a 1, es decir, se producen dos cambios, mientras que en el código Gray se pasa de 1101 a 1111 con un sólo cambio, el del bit 2^1 de 0 a 1.

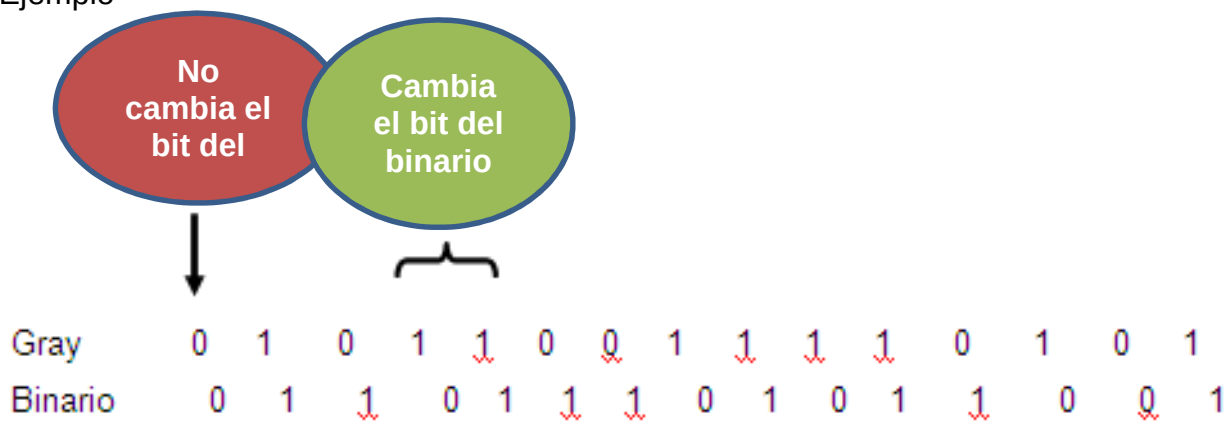
Para convertir la palabra representada en código Gray a su forma binaria, debe empezarse primeramente por la conversión del Bit Más Significativo (BMS). En binario, el bit menos significativo es el 2^0 y el bit más significativo es el más alto en la posición ponderada (para cuatro bits, es el 2^3). Un ejemplo de un número binario y de su forma Gray equivalente es estando el BMS a la izquierda.

Gray	1	011010111001
Binario	1	101100101110

Para hacer la conversión, se repite en la forma binaria el mismo bit que aparece en la forma Gray hasta alcanzar el primer 1, que se repite también. En nuestro ejemplo, la forma Gray empieza por 1, el cual se repite como primer bit (BMS) de la forma binaria. Se sigue repitiendo este bit, en el código binario, esperando que los siguientes bits, en la forma Gray, sean 0 (una posición en el ejemplo).

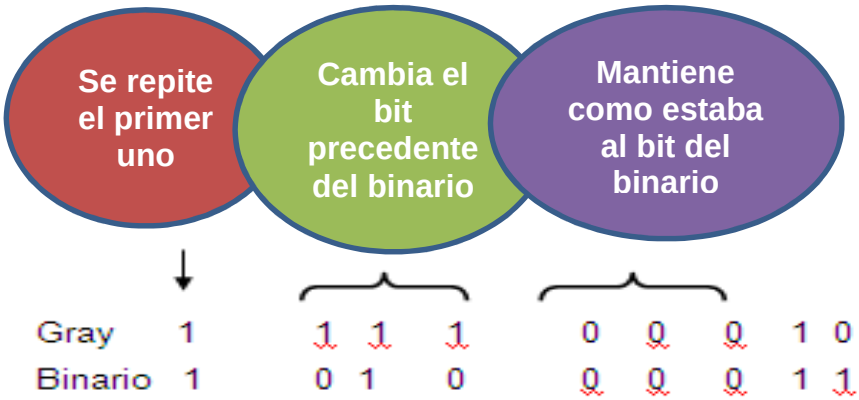
Para cada 1 que aparezca a continuación (después del primero) en la forma de Gray, se cambia el bit correspondiente, de la palabra codificada en binario, respecto del que le precede en la forma binaria. El segundo 1 de la forma Gray indica cambio de bit en la forma binaria; como anteriormente era 1 pasa a ser 0. De acuerdo con esta regla, el siguiente 1 de la forma Gray indica el cambio del bit anterior (0) a 1. El siguiente 0 de la forma Gray significa que se mantiene el bit precedente, de la forma binaria, repitiéndose de nuevo el 1. Este procedimiento se reitera para el resto de la palabra.

Ejemplo



Primer 1

Ejemplo:



Cuando aparece un 1 en la forma Gray se produce un cambio del bit precedente de la forma binaria (cualquier que fuese); un 0 mantiene el bit de la forma binaria como estaba.

El procedimiento para convertir una palabra binaria en su forma Gray es sencillo. La regla cosiste en comparar cada par de bits sucesivos (empezando con el Bit Más Significativo). Si son iguales se escribe un 0 en la forma Gray y si son diferentes se escribe un 1. Al empezar la comparación el primer bit se compara con cero.

Ejemplo

Binario

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

Gray

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

Ejemplo

Binario 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1

Gray 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0

Nota

El resultado se obtiene al sumarse los dos bits, en módulo dos (sin arrastre), y el resultado se coloca debajo, como bit de la forma Gray.

3.2. Códigos alfanuméricos

La información que se va a procesar por una computadora está formada por letras del alfabeto, números decimales, caracteres especiales u órdenes, las cuales han de codificarse en binario. Para representar los dígitos decimales se necesita un código de cuatro bits (como sucede en el código BCD). Pero para representar estos dígitos (0 - 9), más las 26 letras del alfabeto, más algunos caracteres especiales, se necesita un código de por lo menos, seis bits ($2^6 = 64$ combinaciones). Para tal representación se utilizan los códigos ASCII, BCDIC y EBCDIC.

3.2.1. ASCII

El código ASCII (*American Standard Code for Information Interchange*) es un código normalizado que está siendo muy aceptado por los fabricantes de computadoras. Este código ocupa ocho bits con los cuales se permite la representación de los números decimales (0-9), caracteres alfabéticos (letras minúsculas y mayúsculas), signos especiales (por ejemplo, *, +, =, etc.) y más de treinta órdenes o instrucciones de control (por ejemplo, comienzo de mensaje, final de mensaje, retorno de carro, cambio de línea, etc.).

La tabla Códigos ANSCII y EBCDIC muestra la representación de los números decimales, caracteres alfabéticos y algunos caracteres especiales en código ASCII. La numeración convenida para el código ASCII establece una secuencia de izquierda a derecha, de tal modo que la posición del bit 7 es la posición del bit de orden más elevado. Esta misma codificación puede emplearse para impresoras de alta velocidad, ciertos teletipos, etc.

Carácter	Código ASCII	Código EBCDIC	Carácter	Código ASCII	Código EBCDIC
Blanco	**P010 0000**	**0100 0000**	**A**	**P100 0001**	1100 0001
.	**P010 1110**	**0100 1011**	**B**	**P100 0010**	1100 0010
(	**P010 1000**	**0100 1101**	**C**	**P100 0011**	1100 0011
+	**P010 1011**	**0100 1110**	**D**	**P100 0100**	1100 0100
S	**P010 0100**	**0100 1011**	**E**	**P100 0101**	1100 0101
*	**P010 1010**	**0100 1101**	**F**	**P100 0110**	1100 0110
)	**P010 1001**	**0110 0000**	**G**	**P100 0111**	1100 0111
	P010 1101	**0110 0001**	**H**	**P100 1000**	1100 1000
/	**P010 1111**	**0110 1011**	**I**	**P100 1001**	1100 1001
‘	**P010 1100**	**0111 1101**	**J**	**P100 1010**	1101 0001
	P010 0111	**0111 0001**	**K**	**P100 1011**	1101 0010
=	**P010 1101**	**0111 0001**	**L**	**P100 1100**	1101 0011
0	**P010 0000**	**1111 0000**	**M**	**P100 1101**	1101 0100

1	**P010 0001**	**1111 0001**	**N**	**P100 1110**	1101 0101
2	**P010 0010**	**1111 0010**	**O**	**P100 1111**	1101 0110
3	**P010 0011**	**1111 0011**	**P**	**P101 0000**	1101 0111
4	**P010 0100**	**1111 0100**	**Q**	**P100 0001**	1101 1000
5	**P010 0101**	**1111 0101**	**R**	**P100 0010**	1101 1001
6	**P010 0110**	**1111 0110**	**S**	**P100 0010**	1110 0001
7	**P010 0111**	**1111 0111**	**T**	**P100 0100**	1110 0010
8	**P010 1000**	**1111 1000**	**U**	**P100 0101**	1110 0011
9	**P010 1001**	**1111 1001**	**V**	**P100 0110**	1110 0101
			W	**P100 0111**	1110 0110
			X	**P100 1000**	1110 0111
			Y	**P100 1001**	1110 1000
			Z	P100 1010	1110 1001

Códigos ANSCII y EBCDIC

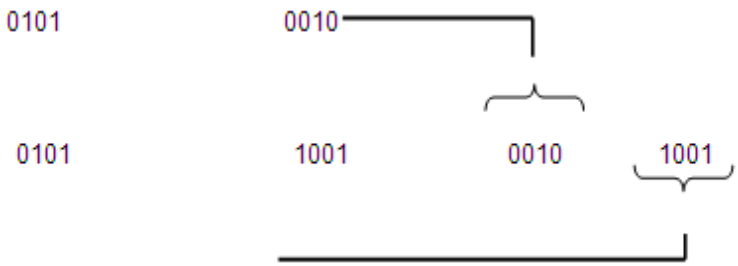
Donde:

P es un bit de paridad.

Nota: Cuando se transmiten o almacenan datos binarios, frecuentemente se añade un bit adicional (denominado bit de paridad) el cual se utiliza en la detección de errores, ver sección (3).

En los equipos de entrada/salida se utiliza el código de ocho niveles para representar los caracteres. Una vez introducidos en la computadora, los caracteres pueden ser tratados del modo más conveniente para las distintas operaciones. Por ejemplo, las cifras decimales no necesitan mantenerse en la computadora como palabras de ocho bits. Para las cifras 0-9 se han elegido, intencionadamente, los 4321 en coincidencia con la forma codificada BCD. Al eliminar los bits 765X, la computadora solo necesita conservar los cuatro bits de la forma BCD para representar las cifras decimales. Si la longitud de la palabra es de ocho bits, como hemos indicado, la computadora puede procesar internamente las cifras decimales agrupando en una palabra los dos dígitos BCD de cuatro bits.

Por ejemplo, si deseamos agrupar en una palabra de ocho bits dos dígitos codificados en BCD. El número 29, que en las unidades de entrada/salida se representa como 01010010 01011001, puede reagruparse en la computadora en la forma



Que forma una palabra de ocho bits con dos cifras decimales

Los diseñadores de computadoras han utilizado el término descriptivo "byte" a un grupo de 8 bits que pueden representar una palabra o algunos caracteres. La información se procesa por bytes en lugar de por bits, en el interior de una computadora. El byte de ocho bits (octeto) podría utilizarse para representar un carácter ASCII, al introducirlo en la computadora o para representar dos cifras decimales en las operaciones aritméticas. De este modo las palabras pueden representarse mediante un número prefijado de bytes.

Nota: Un *bit*, por definición, es un dígito binario y el cual puede tomar el valor de “0” o “1”.

Por estar íntimamente ligado al byte u octeto (y por consiguiente a los enteros que van del 0 al 127), el problema que presenta es que no puede codificar más que 128 símbolos diferentes (128 es el número total de diferentes configuraciones que se pueden conseguir con 7 dígitos binarios (0000000, 0000001,..., 1111111), usando el octavo dígito de cada octeto (como bit de paridad) para detectar algún error de transmisión). Un cupo de 128 es suficiente para incluir mayúsculas y minúsculas del abecedario inglés, además de cifras, puntuación, y algunos "caracteres de control" (por ejemplo, uno que permite a una impresora que pase a la hoja siguiente), pero el ASCII no incluye ni los caracteres acentuados ni el comienzo de interrogación que se usa en castellano, ni tantos otros símbolos (matemáticos, letras griegas...) que son necesarios

en muchos contextos y por esto que se propuso el Código ASCII Extendido. Debido a las limitaciones del ASCII se definieron varios códigos de caracteres de 6 u 8 bits entre ellos el código BCDIC (6 Bits) y el código ASCII Extendido. El código ASCII Extendido es un código de 8 bits que complementa la representación de caracteres faltantes del código ASCII estándar.

Sin embargo, el problema de estos códigos de 8 bits es que cada uno de ellos se define para un conjunto de lenguas con escrituras semejantes y por tanto no dan una solución unificada a la codificación de todas las lenguas del mundo. Es decir, no son suficientes 8 bits para codificar todos los alfabetos y escrituras del mundo, por lo tanto hay que buscar otros códigos de codificación más eficientes. Una posible solución al problema de la codificación de todas las lenguas del mundo es utilizar el código Unicode.

Código Unicode

Como solución a estos problemas, desde 1991 se ha acordado internacionalmente utilizar la norma Unicode, que es una gran tabla, que en la actualidad asigna un código a cada uno de los más de cincuenta mil símbolos, los cuales abarcan todos los alfabetos europeos, ideogramas chinos, japoneses, coreanos, muchas otras formas de escritura, y más de un millar de símbolos especiales.[2]

3.2.2. BCDIC

El código BCDIC (del idioma inglés, *Standard Binary Coded Decimal Interchange Code*) usualmente utiliza 6 bits de codificación (2^6 = hasta 64 caracteres) y en ocasiones se adhiere un bit adicional para verificar posibles errores de transmisión o grabación.

[2] Wikipedia: “Codificación de caracteres”, actualizado el 28/01/09, disponible en: http://es.wikipedia.org/wiki/Codificaci%C3%B3n_de_caracteres, recuperado el 23/02/09.

Un inconveniente de este código es que con 6 bits son insuficientes para representar (codificar) los caracteres alfabéticos, numéricos, símbolos especiales, etc. Debido a este inconveniente se propuso el BCDIC extendido, el cual explicaremos a continuación.

3.2.3. EBCDIC

El código de Intercambio BCD Extendido -EBCDIC- (del idioma inglés, *Extended BCD Interchange Code*) es un código de 8 bits y se utiliza para representar hasta 256 caracteres (símbolos) distintos. En este código, el bit menos significativo es el b7 y el más significativo es el b0. Por consiguiente, en el código EBCDIC se transmite primero el bit de mayor orden, b7, y al último se transmite el bit de menor orden, b0. Este código no facilita el uso de un bit de paridad.

La tabla Códigos ANSCII y EBCDIC muestra la representación (codificación) de los caracteres con el código EBCDIC y su comparación con el código ASCII. Con respecto a la columna del código EBCDIC, las letras mayúsculas de la A a la Z, se dividen en tres grupos (A-I), (J-R), (S-Z) y en las primeras cuatro posiciones se identifica el grupo al cual pertenece la letra y en las restantes cuatro posiciones el dígito correspondiente a la posición de la letra en el grupo. Los dígitos del 0 al 9 se identifican con un uno en las primeras cuatro posiciones y en las restantes cuatro posiciones el dígito en binario.

3.3. Códigos por detección de error

La detección y/o corrección de errores es un campo de estudio de mucho interés y aplicación creciente en la transmisión, codificación, compresión y almacenamiento de datos digitales debido a las limitaciones del canal de transmisión.

Si en el envío de una información, por ejemplo 1000, por efecto del canal de transmisión, se recibe como 1001, ambos números pertenecen al mismo código y no será posible saber si ha habido algún error en la información que se recibe. Una medida que se toma para detectar si hubo algún error, es la de agregar a cada símbolo o carácter alfanumérico un bit a la izquierda del mismo, dicho bit recibe el nombre de "Bit de paridad". Estos bits pueden ser de paridad, par o impar según el método de verificación de paridad que se tenga y los explicaremos a continuación.

3.3.1. Paridad par

La paridad par consiste en verificar que la suma de todos los 1's existentes en un carácter alfanumérico o un símbolo es par, incluyendo en dicha suma el bit de paridad. De lo anterior se desprende, que un bit de paridad par será aquel que asumirá el estado de 1 si la suma de los 1's restantes es impar y será 0 si la suma de los 1's en el carácter es par.

La verificación de paridad par deberá cumplir con la ecuación siguiente:

$$X^n + X^{n-1} + X^{n-2} + \ldots + X^1 + X^0 + P = 0$$

Donde
P es el bit de paridad y las X^n son los bits que forman el carácter.

3.3.2. Paridad impar

La paridad impar se obtiene verificando que la suma de todos los 1's existentes en un carácter alfanumérico o un símbolo, incluyendo al bit de paridad, sea un número impar, y por tanto un bit de paridad impar será aquel que sumando a los 1's restantes deberá producir un número impar por tanto este bit será 1 si la suma de los 1's restantes es impar.

La verificación de paridad impar deberá cumplir con la ecuación siguiente:

$$X^n + X^{n-1} + X^{n-2} + ...+ X^1 + X^0 + P = 1$$

Donde
P es el bit de paridad y las X^n son los bits que forman el carácter. Puesto que *P* puede tomar los valores de 0 ó 1, se puede demostrar que:

$$P(PAR) = X^n + X^{n-1} + X^{n-2} + ...+ X^1 + X^0 + 1$$

$$P(IMPAR) = X^n + X^{n-1} + X^{n-2} + ...+ X^1 + X^0$$

Por otro lado, el bit de paridad también se puede aplicar a otros códigos como se muestra a continuación.

Por ejemplo, si modificamos el código BCD mediante un bit de paridad. Este bit se añade a la derecha de la posición 2^0. En un código con paridad par, el bit de paridad añadido hace par el número total de 1's y en un código de paridad impar, se selecciona

el bit de paridad de modo que haga impar el número total de 1's. Cuando se recibe una palabra codificada, se compara su paridad (par o impar, según haya sido elegida previamente) y se acepta como correcta si pasa la prueba. La tabla Paridad en el código BCD muestra los códigos BCD, BCD con paridad impar y BCD con paridad par.

Decimal	BCD	BCD con paridad impar	BCD con paridad par
0	**0000**	**0000 1 o sea 00001**	0000 0 o sea 00000
1	**0001**	**0001 0 00010**	0001 1 00011
2	**0010**	**0010 0 00100**	0010 1 00101
3	**0011**	**0011 1 00111**	0011 0 00110
4	**0100**	**0100 0 01000**	0100 1 01001
5	**0101**	**0101 1 01011**	0101 0 01010
6	**0110**	**0110 1 01101**	0110 0 01100
7	**0111**	**0111 0 01110**	0111 1 01111
8	**1000**	**1000 0 10000**	1000 1 10001
9	1001	1001 1 10011	1001 0 10010

Paridad en el código BCD

Ejemplo. Verifique la existencia de errores en las siguientes palabras, codificadas en BCD con paridad par.

Solución.

	Palabra	**Bit de paridad**	**Tipo de paridad**
a)	1001	0	paridad impar
b)	1000	0	paridad impar
c)	0001	0	paridad par
d)	0110	1	paridad impar

Los ejemplos (a) y (c) son incorrectos y los (b) y (d), correctos.

La paridad también se puede utilizar en otros códigos distintos del código BCD. Cuando se envía un conjunto de palabras con paridad añadida, el bit de paridad se elige de manera similar.

Ejemplo. Verifique la existencia de errores en las siguientes palabras.
Solución.

	Palabra	Bit de paridad	Tipo de paridad
a)	0110111101	1	paridad par
b)	1101110100	0	paridad impar
c)	1110111011	0	paridad impar
d)	1011011100	0	paridad par
e)	1010111010	1	paridad impar

Los ejemplos (b) y (c) son incorrectos: (a), (d) y (e) son correctos,

Los códigos BCD, BCD con paridad impar y BCD con paridad par no son los únicos códigos para la detección de errores. Existen otros códigos para la detección de errores de un solo bit. Entre los cuales se encuentra el código Biquinario. El código biquinario es un código ponderado de 7 bits cuya distancia mínima es dos y permite la detección de errores de un bit, como se explicará a continuación.

Código Biquinario

Para facilitar la comprobación de posibles errores cuando se transmiten datos binarios, se puede utilizar el código biquinario o la adición de un bit de paridad a cada carácter codificado. Hasta ahora se han empleado otros códigos, dependiendo de la elección del grado de fidelidad requerido, de la cantidad de información que puede enviarse y de la cuantía del equipo transmisor y receptor necesario para realizar las operaciones

de comprobación, pero no un código para la detección de errores, el cual explicaremos a continuación.

El código biquinario es un código ponderado y consta de 7 bits, de los cuales, los 2 de la izquierda y los 5 de la derecha se consideran partes separadas del conjunto. La tabla Código Biquinario muestra las formas codificadas del 0 a 9, así como la ponderación de cada una de las posiciones de sus bits.

Decimal		Biquinario						
		5	0	4	3	2	1	**0**
0		0	1	0	0	0	0	**1**
1		0	1	0	0	0	1	**0**
2		0	1	0	0	1	0	**0**
3		0	1	0	1	0	0	**0**
4		0	1	1	0	0	0	**0**
5		1	0	0	0	0	0	**1**
6		1	0	0	0	0	1	**0**
7		1	0	0	0	1	0	**0**
8		1	0	0	1	0	0	**0**
9		**1**	**0**	**1**	**0**	**0**	**0**	**0**

Código Biquinario

En esta tabla se puede observar que se necesitan siete bits para especificar una cifra decimal (mientras que en BCD o Exceso en tres se requieren cuatro bits). El código biquinario presenta, como ventaja importante, la propiedad intrínseca de indicar cuándo existe error en la palabra codificada. En general, cuando se transmite información de un lugar a otro, como sucede en la computadora, resulta muy conveniente el empleo de un código que permita determinar si se ha producido un error en la transmisión.

Analizando el código biquinario de la tabla Código Biquinario observamos lo siguiente: cada palabra solamente tiene dos 1's. Por consiguiente, si apareciera cualquier otro 1 extra en la respuesta significaría que se había producido un error y la palabra no debería ser aceptada. Si solamente se hubiese recibido un 1, de nuevo sería evidente la existencia de error. Además, el reconocimiento y aceptación de una palabra, como correcta, exige que haya un solo bit entre los dos primeros de la izquierda y que haya también un solo bit entre los cinco restantes de la derecha. La comprobación se establece fácilmente, debido a que es fácil realizar un circuito que compruebe la existencia de un 1 entre dos bits y de otro circuito que detecte la presencia de un 1 entre cinco bits.

Por ejemplo: determine la existencia de errores en el siguiente grupo de palabras codificadas en biquinario.

	Biquinario	Decimal
a)	01 10001	4
b)	01 10010	5
c)	10 10101	6
d)	11 00010	6
e)	01 01000 01 00010	31
f)	10 10000 10 00010	31

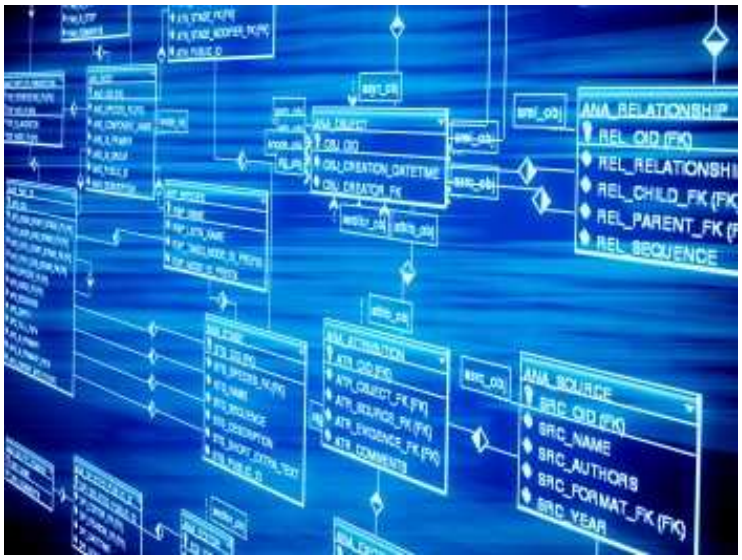
Los ejemplos (a) y (d) son incorrectos, mientras que los (e) y (f) son correctos.

RESUMEN

La forma de representación de datos en las computadoras se realiza mediante los símbolos cero y uno, que corresponden a los valores que manejan los dispositivos electrónicos con los que están construidas las computadoras digitales. Las reglas de asociación de estos dos valores permiten la generación de estándares de representación denominados códigos. Así, un código es un conjunto de símbolos y reglas de relación para representar información de manera sistemática y estandarizada.

En la teoría de la información, código es la forma que toma la información que se intercambia entre la fuente (el emisor) y el destino (el receptor) de un ciclo de comunicación. Un código implica la comprensión o decodificación del paquete de información que se transfiere, pues además de definir los símbolos por utilizar para la representación de la información, define también las reglas de utilización de dichos símbolos.

Los principales códigos utilizados para definir datos numéricos son el código BCD, X3 y Gray. Este último tiene la ventaja de facilitar la identificación de errores, por ser autocomplementado. Es importante señalar que codificar y convertir una cantidad numérica no tienen el mismo significado. Convertir un número decimal a una base diferente implica que las



operaciones aritméticas serán realizadas en el sistema base al que se realiza la conversión, mientras que la codificación representa cada dígito de una manera diferente (en este caso secuencias binarias para cada dígito) pero manteniendo las estructuras y operaciones de un sistema decimal.

Para representar información alfanumérica en forma binaria, la computadora emplea los códigos ASCII, BCDIC y EBCDIC, los cuales se indicaron en las tablas correspondientes en el desarrollo de la unidad.

Finalmente, debido a que la transferencia de información en un sistema informático es susceptible de errores, se hace necesaria la adición de bits de codificación de error que indiquen la existencia de diferencias entre la información emitida y recibida por las diferentes unidades de una computadora o entre redes de computadoras. Existen diversos códigos de detección y corrección de error, el más común es el de paridad, el cual agrega simplemente un bit a la cadena de bits de la secuencia de información de acuerdo a la regla par o impar.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo	Páginas
Mano (1986)	1	16-20
Quiroga (2010)	2	56-72

Mano, Morris. (1986). *Lógica Digital y diseño de computadores*. México: Prentice Hall Hispanoamericana.

Quiroga, Patricia. (2010). *Arquitectura de computadoras*. México: Alfaomega.

Unidad 4.

Álgebra de boole

OBJETIVO PARTICULAR

Al termina la unidad, el alumno explicará los principios fundamentales del álgebra y funciones booleanas y los procesos algebraicos.

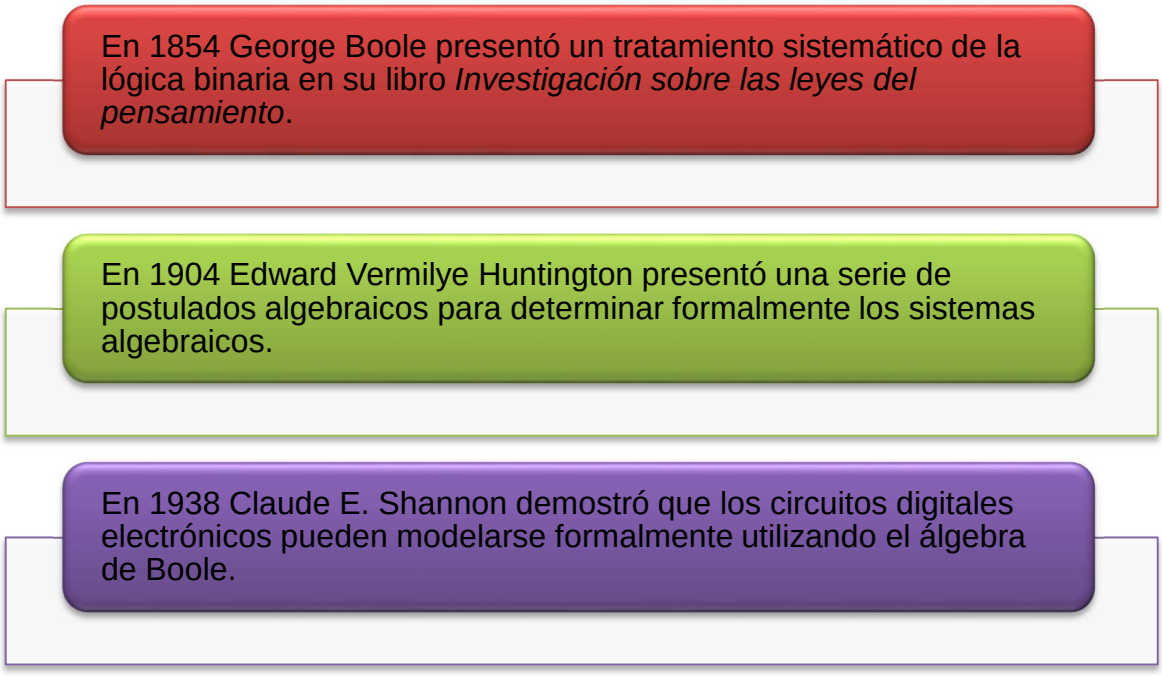
TEMARIO DETALLADO
(8 horas)

4. Álgebra de Boole

- 4.1. Principios de electrónica básica
  - 4.1.1. Lógica binaria
- 4.2. Propiedades fundamentales del álgebra de Boole
  - 4.2.1. Leyes de Morgan
  - 4.2.2. Compuertas lógicas
  - 4.2.3. Función booleana
- 4.3. Técnicas de minimización de funciones
  - 4.3.1. Proceso algebraico
  - 4.3.2. Mapas de Karnaugh

INTRODUCCIÓN

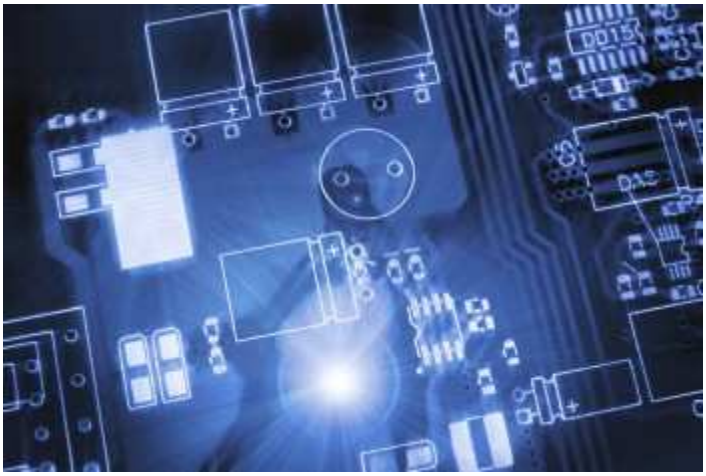
La tecnología nos permite construir compuertas digitales a través de transistores y mediante las compuertas diseñamos los circuitos digitales empleados en las computadoras. Sin embargo el empleo de esta tecnología no determina por sí solo la aparición de las computadoras como procesadores de información, es necesaria la aplicación de principios lógicos y algebraicos que nos permitan manipular, con rigor matemático, variables mediante dispositivos electrónicos. La forma como las computadoras realizan operaciones lógicas es mediante el álgebra de Boole aplicada a los circuitos electrónicos. El álgebra booleana es importante pues permite la sistematización y representación matemática del funcionamiento de los circuitos electrónicos digitales. La sistematización del estudio de los circuitos electrónicos digitales ha tenido tres momentos importantes:



Para entender el funcionamiento de las computadoras, es necesario entender los principios, axiomas, teoremas y postulados del álgebra que nos interesa. El presente capítulo está dividido en tres temas: principios de electrónica binaria y álgebra booleana; propiedades fundamentales y tercero, técnicas de minimización de funciones. En el primero se establecen los elementos de funcionamiento de circuitos digitales; en el segundo se establecen formalmente los axiomas y postulados que le dan forma y estructura matemática al álgebra de Boole. En el último tema, se presentan las dos principales formas de minimizar funciones booleanas que son manipulación algebraica y mapas de Karnaugh.

Es importante aclarar que múltiples problemas y procesos del funcionamiento de circuitos digitales se pueden modelar mediante estas funciones y que para su diseño eficiente, estas deben representarse en muchas ocasiones en su forma mínima, por lo el proceso de minimización adquiere relevancia. En el último tema abordaremos estas formas de minimización y otras formas de representación de funciones booleanas.

4.1. Principios de electrónica básica

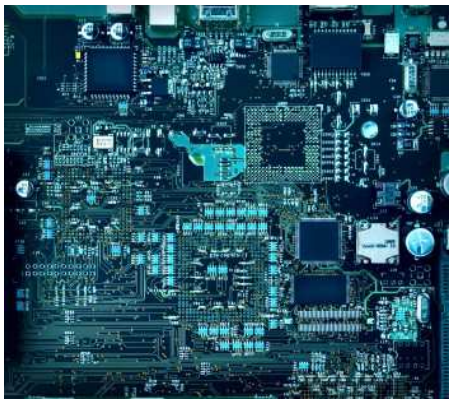


La electrónica se dedica al análisis y síntesis de circuitos electrónicos. La electrónica se puede dividir en tres áreas: Analógica, Digital e Industrial. La Electrónica Digital es aquella que trabaja con señales eléctricas discretas, esta señal únicamente tiene dos valores: cero (“0”) lógico y uno (“1”) lógico. La electrónica digital es la herramienta principal para el diseño y construcción de algunas unidades que constituyen una computadora digital, por ejemplo, el decodificador, el multiplexor, la unidad aritmética-lógica, etc., (ver unidad 5) y en el diseño de circuitos secuenciales basados en flip-flops, (ver unidad 6).

4.1.1. Lógica binaria

Lógica binaria	La electrónica digital utiliza dos estados: cero “0” lógico o uno “1” lógico. A la vez que dicha electrónica trabaja de dos formas:
Lógica Positiva	La Lógica positiva define al “0” lógico como falso y al “1” como verdadero.
Lógica Negativa	La Lógica negativa define al “0” lógico como verdadero y al”1” como falso.

4.2. Propiedades fundamentales del álgebra de Boole



El álgebra de Boole es la técnica matemática empleada en el estudio de problemas de naturaleza lógica. Con el desarrollo de las computadoras, el empleo del álgebra de Boole se ha incrementado en el campo de la electrónica digital hasta alcanzar la posición que actualmente ocupa, siendo utilizada por los ingenieros como ayuda para el diseño y construcción de circuitos lógicos combinacionales y/o secuenciales. En el campo de las computadoras, el álgebra de Boole se emplea para describir circuitos cuyo estado puede caracterizarse por 0 ó 1. Los signos lógicos 1 ó 0 pueden ser los números base del sistema de numeración binario. También pueden identificarse con las condiciones de “abierto” o ”cerrado” o con las condiciones de ”verdadero” o ”falso”, que son de naturaleza binaria.

Puesto que las variables booleanas pueden adoptar dos valores y, por tanto cualquier incógnita puede ser especificada con 0 ó 1, el álgebra de Boole resultará sencilla en comparación en donde las variables son continuas.

4.2.1. Leyes de De Morgan

El álgebra de Boole se apoya en un conjunto de teoremas y leyes que permiten diseñar y construir circuitos combinacionales y secuenciales más sencillos. Dicho conjunto de teoremas y leyes se resumen en la tabla Teoremas de Álgebra de Boole

Relación	Dual	Propiedad
$AB = BA$	$A + B = B + A$	Conmutativa
$A(B+C) = AB + AC$	$A + BC = (A+B)(A+C)$	Distributiva
$1A = A$	$0 + A = A$	Identidad
$A\,\overline{A} = 0$	$A + \overline{A} = 1$	Complemento
$0A = 0$	$1 + A = 1$	Teoremas del cero y el uno
$AA = A$	$A + A = A$	Idempotencia
$A(BC) = (AB)C$	$A+(B+C) = (A+B) + C$	Asociativa
$\overline{\overline{A}} = A$		Involución
$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$	Teorema de DeMorgan
$AB + \overline{A}C + BC = AB + \overline{A}C$	$(A+B)(\overline{A}+C)(B+C) = (A+B)(\overline{A}+C)$	Teorema del concenso
$A(A+B) = A$	$A + AB = A$	Teorema de absorción

Teoremas de Álgebra de Boole

En el álgebra de Boole, una variable binaria puede adoptar el valor de cero (“0”) lógico o uno (“1”) lógico. Estos valores se relacionan con los valores de 0 y 5 Volts (lógica positiva). La asignación puede invertirse en términos de las tensiones asignadas al 0 y al 1, es decir, asigna al cero (“0”) lógico el valor de 5 Volts y al uno “1” lógico el valor de

0 Volts (lógica negativa). A fin de comprender el correcto funcionamiento de los circuitos digitales, únicamente utilizaremos los valores lógicos (“0” lógico y “1” lógico) en lugar de los valores físicos (0 Volts y 5 Volts).

Las leyes de De Morgan y los teoremas del álgebra de Boole se utilizan para reducir una función booleana, como se explicará en la sección 3. (Técnicas de minimización de funciones), a continuación daremos una breve explicación del uso de las leyes De Morgan.

Las leyes de De Morgan son:

1.) $\overline{A + B + C + \ldots} = \overline{A}\,\overline{B}\,\overline{C}\ldots$

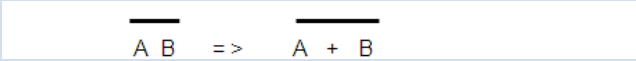
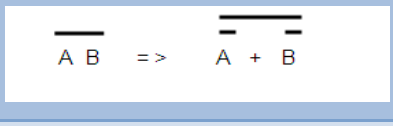
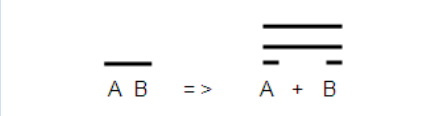
2.) $\overline{A\,B\,C\ldots} = \overline{A} + \overline{B} + \overline{C} + \ldots$

Para poder utilizar de manera correcta las leyes de De Morgan se debe aplicar los siguientes pasos:

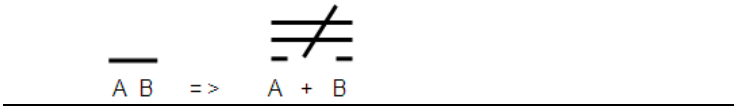
Se intercambia el operador OR (+) por el operador AND () o si es el caso intercambiar el operador AND () por el operador OR (+).
Se niegan cada una de las variables
Se niega todo el término.

Para comprender la aplicación de los pasos mencionados anteriormente, demostraremos la segunda ley de De Morgan, únicamente para el caso de 2 variables.

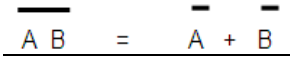
$$\overline{A\,B} = \overline{A} + \overline{B}$$

Intercambio del operador AND () por el operador (+)  La negación del término en este paso se sigue conservando debido a que únicamente se intercambio el operador.
Se niega cada una de las variables 
Se niega todo el término 

Aplicando la propiedad de involución (ver tabla 4.1) al resultado anterior



con lo cual se obtiene el resultado



y de esta manera queda demostrado la segunda ley de de Morgan.

Finalmente, las leyes de De Morgan se pueden utilizar para cualquier número de variables, siempre y cuando se tomen dos variables a la vez.

4.2.2. Compuertas lógicas

Una compuerta lógica es un dispositivo físico que implementa una función básica del álgebra de Boole. La electrónica digital utiliza tres compuertas básicas como son: la compuerta OR, AND y NOT y a partir de estas compuertas se crean compuertas complementarias como son: NAND, NOR, OR-exclusiva y NOR-exclusiva, las cuales explicaremos a continuación:

Compuerta OR

La figura 1a muestra el símbolo lógico, la tabla de verdad y la ecuación característica de la compuerta OR. La compuerta OR presenta en su salida un nivel alto, si cualquiera de sus entradas A o B están en nivel alto. La salida tiene un nivel bajo si todas las entradas tienen un nivel bajo o “0”.

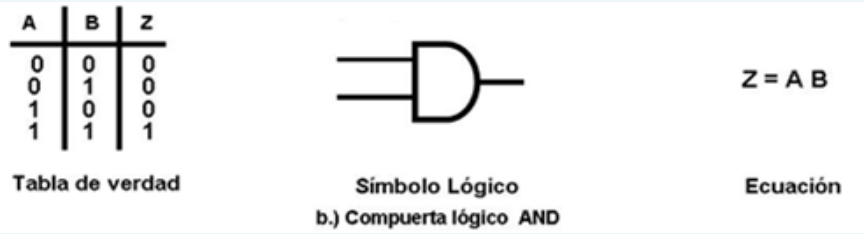


Figura 1a

En la tabla de verdad de la figura Compuertas básicas, el dígito binario 1 representa un nivel alto de voltaje, y el dígito binario 0 un nivel bajo de voltaje.

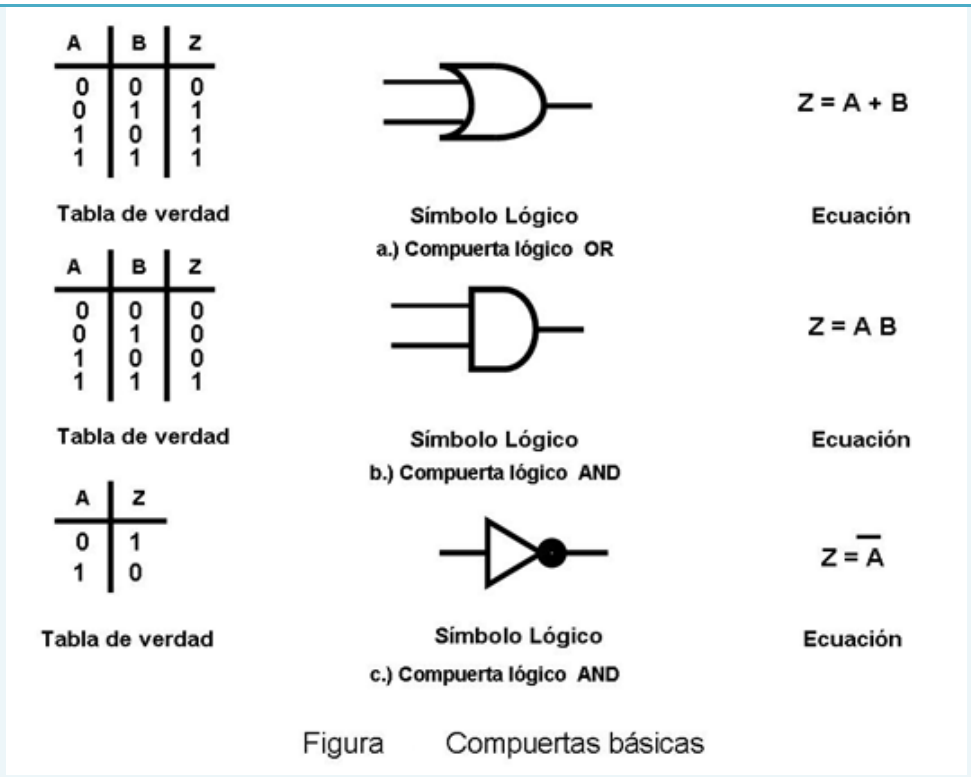


Figura Compuertas básicas

Compuerta AND

Podemos decir que la compuerta AND es un circuito en el cual la salida será un nivel alto solamente cuando todas las entradas se encuentren en el nivel alto. La salida es un nivel bajo si cualquiera de las entradas (A o B) está en nivel bajo. La **figura 1b** muestra el símbolo lógico, su tabla de verdad y su ecuación característica de dicha compuerta.

Figura 1b

Compuerta NOT

La compuerta más sencilla es la compuerta inversora o NOT. La compuerta inversora es aquella en la cual su salida tiene un nivel bajo (“0”) cuando en su entrada presenta un nivel alto (“1”) y viceversa, es decir, su salida tiene un nivel alto (“1”) cuando en su entrada tiene un nivel bajo (“0”). La figura 1c) presenta el símbolo lógico, la tabla de verdad y la ecuación característica de la compuerta NOT.

Figura 1c.

La correcta combinación de la compuerta NOT con las compuertas AND y OR produce una serie de compuertas complementarias como lo son: las compuertas NAND, NOR,

OR-exclusiva y NOR-exclusiva, ver figura Compuertas complementarias. Las razones de la popularidad de las puertas inversoras (NOR y NAND) son:

a) Baratas

b) Rápidas, y

c) Disipan menos potencia

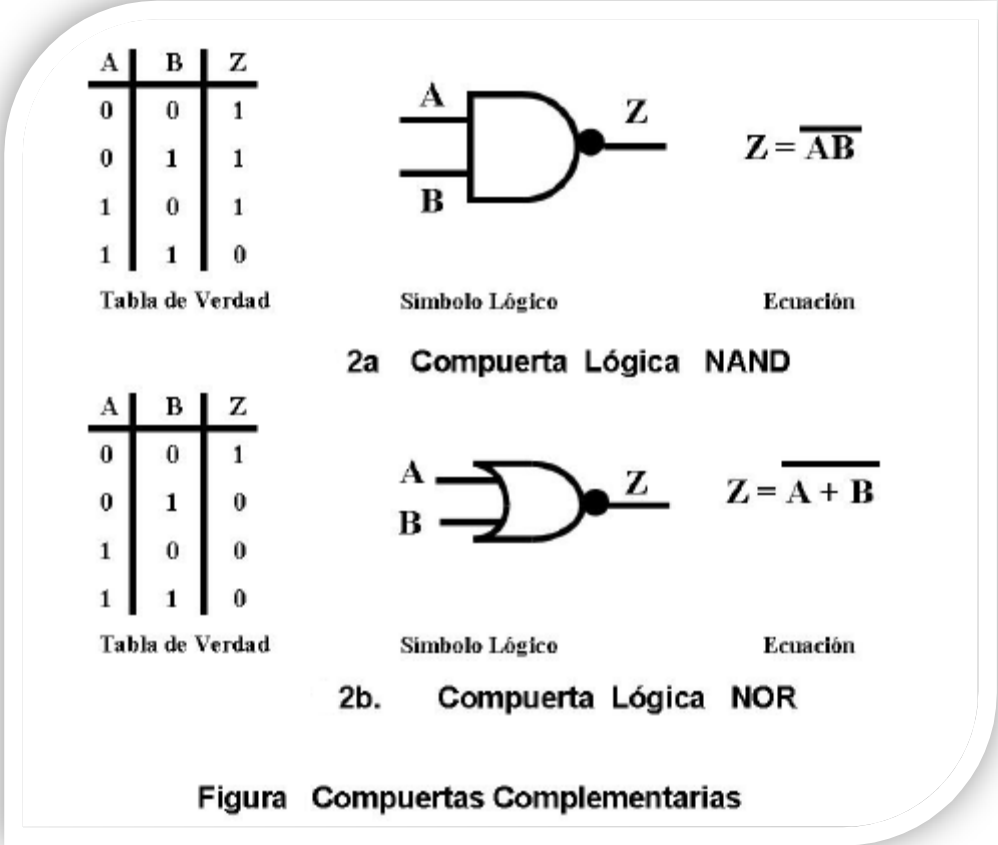
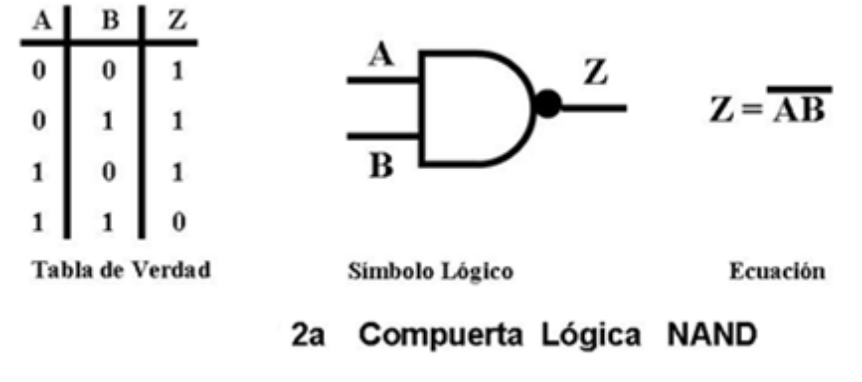


Figura Compuertas Complementarias

Compuerta NAND

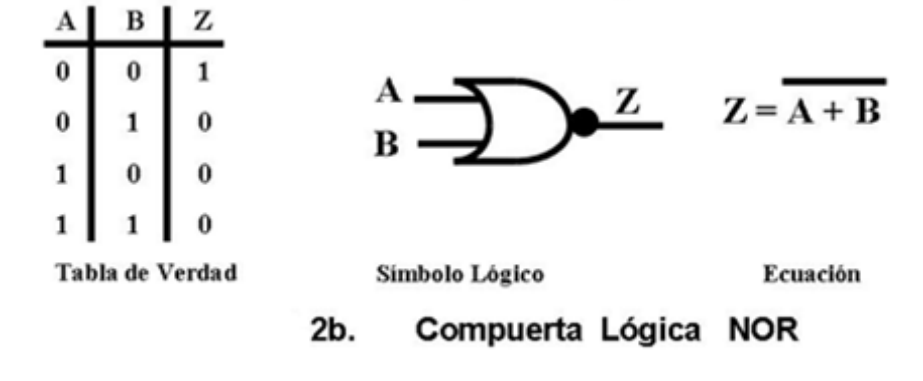
La compuerta NAND es equivalente a una compuerta AND seguida de una compuerta NOT, tal como se muestra en la figura 2a). El funcionamiento de esta compuerta es el siguiente: La salida presenta un nivel bajo solamente si todas las entradas están en nivel alto (“1”). La salida tiene un nivel alto si cualquiera de las entradas está en nivel bajo “0”).



2a Compuerta Lógica NAND

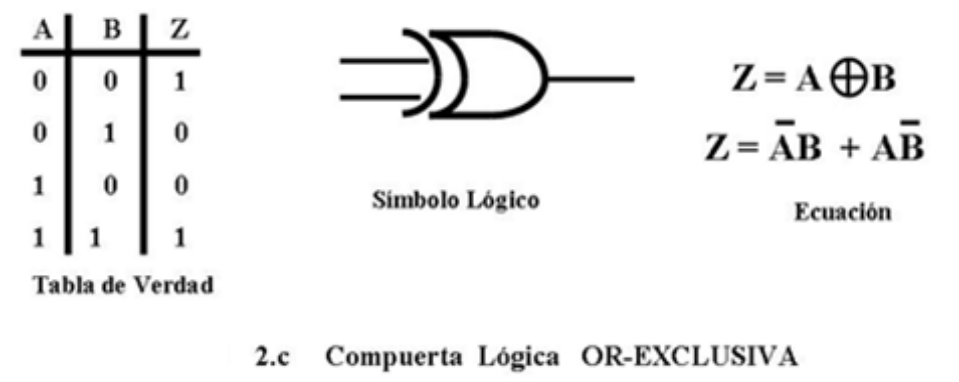
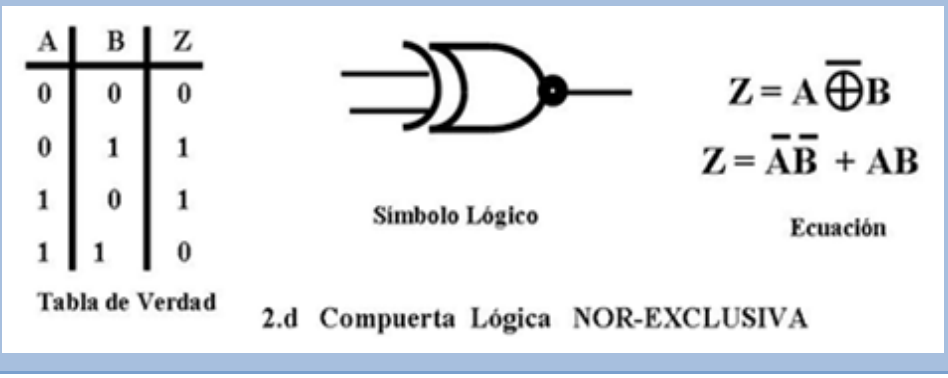
Compuerta NOR

La compuerta NOR es equivalente a una compuerta OR seguida de una compuerta NOT, tal como se muestra en la figura 2b.



2b. Compuerta Lógica NOR

La compuerta NOR es aquella en la cual la salida presenta un nivel bajo o “0” si sus dos entradas está en un nivel alto o “1”y su salida presente un nivel alto ó “1” cuando al menos una de sus entradas tiene un nivel bajo o “0”. La figura 2b. se presenta el símbolo lógico, tabla de verdad y la ecuación característica de la compuerta NOR.

Compuerta OR-Exclusiva	La compuerta OR-exclusiva es aquella en la cual la salida es un nivel bajo si sus entradas son iguales (son 0 ó 1) y presentan un nivel alto cuando sus entradas son diferentes. La figura 2c. muestra el símbolo lógico, tabla de verdad y la ecuación característica.  Tabla de Verdad Símbolo Lógico $Z = A \oplus B$ $Z = \bar{A}B + A\bar{B}$ Ecuación 2.c Compuerta Lógica OR-EXCLUSIVA
Compuerta NOR-Exclusiva	La compuerta NOR-exclusiva es aquella en la cual la salida es un nivel bajo (“0”) si las entradas son diferentes (son “0” ó “1”) y presentan un nivel alto (“1”) cuando sus entradas son iguales. La figura 2d muestra el símbolo lógico, tabla de verdad y ecuación característica.  Tabla de Verdad Símbolo Lógico $Z = A \overline{\oplus} B$ $Z = \bar{A}\bar{B} + AB$ Ecuación 2.d Compuerta Lógica NOR-EXCLUSIVA

4.2.3. Función booleana

Una función booleana representa el análisis y síntesis de un problema determinado. Una función booleana depende de n-variables de entrada y representa a una sola salida.

Definición

Una función booleana es la combinación de variables (de entrada) y operadores lógicos que representan el análisis y/o síntesis de un problema determinado. Una función booleana en algunos casos se puede obtener a partir de una tabla de verdad.

Tabla de verdad

Una contribución fundamental del álgebra de Boole es el desarrollo del concepto de tabla de verdad. Una tabla de verdad captura e identifica las relaciones lógicas entre las n-variables de entrada y las m-funciones lógicas de salida en forma tabular.

4.3. Técnicas de minimización de funciones

La expresión algebraica de una función booleana no siempre es fácil de reducir y generalmente exige cierta intuición e ingenio. Se han desarrollado muchas técnicas para ayudar a la reducción de una función booleana entre las cuales se encuentran el proceso algebraico y los mapas de Karnaugh.

4.3.1. Proceso algebraico

El proceso algebraico: Es una técnica para reducir de manera sistemática una función lógica utilizando las propiedades (teoremas y leyes) fundamentales del álgebra de Boole. Para entender en qué consiste la técnica mostramos una serie de ejemplos a continuación.

Ejemplo: Reduzca la siguiente función utilizando el Algebre de Boole

Solución:

$$f(A,B,C) = \overline{AB + C} + \overline{\bar{A}\bar{C}B} + AC(B + BA)$$

Primero marcamos cada uno de los minitérminos

$$f(A,B,C) = \underset{I}{\overline{AB + C}} + \underset{II}{\overline{\bar{A}C\bar{B}}} + \underset{III}{AC(B + BA)}$$

Factorizando el término III y utilizando el teorema de complemento

$$f(A,B,C) = \underset{I}{\overline{AB + C}} + \underset{II}{\overline{\bar{A}C\bar{B}}} + \underset{III}{AC(B(1 + \cancel{A}))}$$

y ordenando (propiedad conmutativa) la ecuación

$$f(A,B,C) = \overline{AB + C} + \overline{\bar{A}\bar{B}C} + ABC$$

Aplicando la ley de De Morgan a los términos I y II de la expresión anterior

$$f(A,B,C) = \overline{\overline{AB} + \bar{C}} + \overline{\bar{A}\bar{B}} + C + ABC$$

Con lo cual obtenemos la expresión

$$f(A,B,C) = (\overline{AB} + \bar{C}) + (\overline{\bar{A}\bar{B}} + \bar{C}) + ABC$$

A la expresión anterior aplicamos la ley de De Morgan a los términos I y II

$$f(A,B,C) = \overline{(\bar{A} + \bar{B})}\,\bar{C} + \overline{(A + B + \bar{C})} + ABC$$

Se obtiene la siguiente expresión

$$f(A,B,C) = (\bar{A} + \bar{B})\,\bar{C} + (A + B + \bar{C}) + ABC$$

Utilizando la propiedad distributiva

$$f(A,B,C) = (\bar{A}\bar{C} + \bar{B}\bar{C}) + (A + B + \bar{C}) + ABC$$

Factorizando y utilizando el propiedad de complemento, se obtiene

$$f(A,B,C) = \bar{A}\bar{C} + C(\cancel{1}^{1} + B) + A + B + ABC$$

$$f(A,B,C) = \bar{C}(\cancel{A}^{1} + 1) + A(\cancel{BC}^{1} + 1) + B$$

$$f(A,B,C) = A + B + \bar{C}$$

4.3.2. Mapas de Karnaugh

El método de los mapas de Karnaugh es un técnica gráfica que puede utilizarse para obtener los términos mínimos de una función lógica utilizando las variables que les son comunes. Las variables comunes a más de un término mínimo son candidatas a su eliminación. Aunque la técnica puede emplearse para cualquier número de variables, raramente se utiliza para más de seis. El mapa está formado por cajas (o celdas), cada una de las cuales representa una combinación única de las variables. Para una variable, solamente se necesitan dos cajas. Dos variables requieren cuatro combinaciones, ver figura Mapa de Karnaugh para 2 variables. Para tres variables se requieren $2^3 = 8$ cajas, ver figura 4.4 y para cuatro variables $2^4 = 16$ cajas, ver figura 4.5, etc.

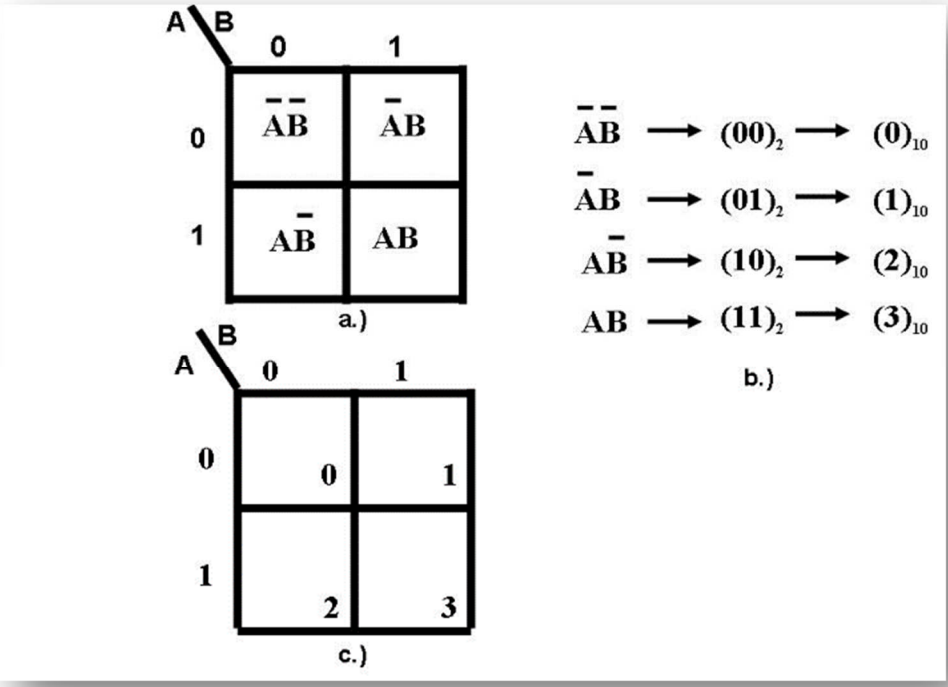


Figura Mapa de Karnaugh para 2 variables

La figura Mapa de Karnaugh para 2 variables muestra las cajas o celdas adyacentes del mapa de Karnaugh para dos variables. En dicha figura se muestra las cuatro únicas combinaciones del mapa, figura Mapa de Karnaugh para 2 variables a) La figura Mapa de

Karnaugh para 2 variables b muestra en forma binaria el valor lógico de cada una de las combinaciones en función de las dos variables, las cuales se pueden pasar al sistema decimal, con lo cual cada una de las cuatro combinaciones anteriores tiene una posición (en el sistema decimal) en cada una de las cajas o celdas en el mapa.

Para tres (ver figura Mapas de Karnaugh para 3 variables), cuatro (ver figura Mapas de Karnaugh para 4 variables) o más variables los mapas se construyen de forma que se solapen cada una de las variables a fin de producir todas las combinaciones requeridas.

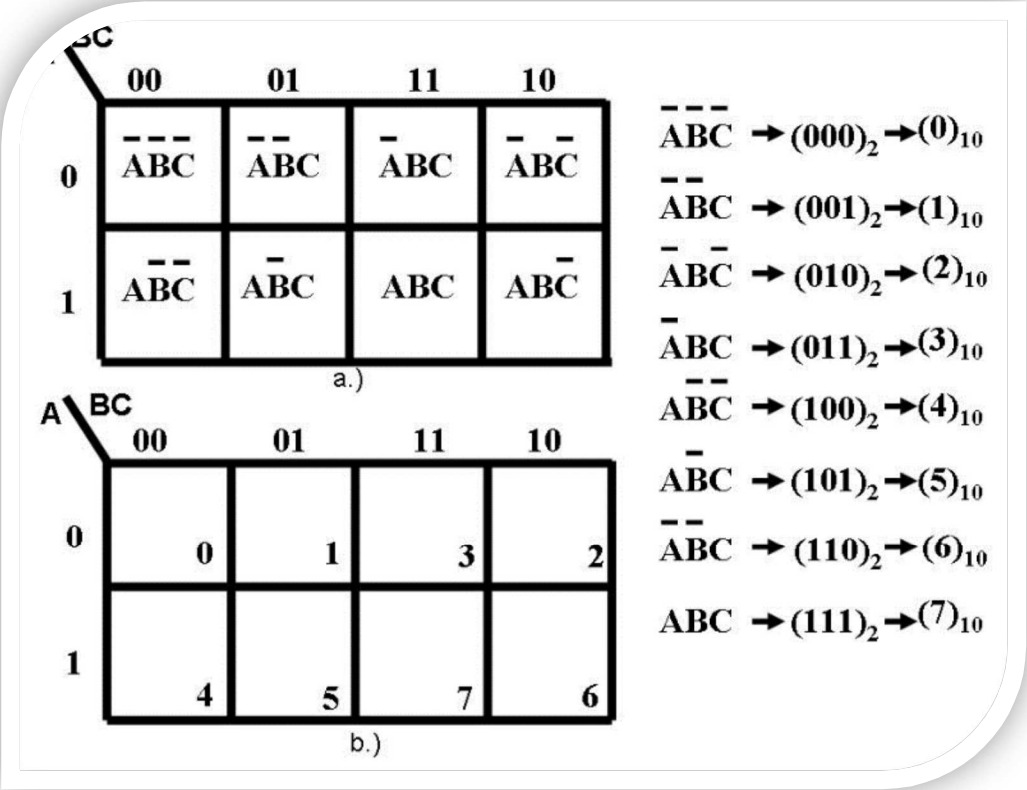


Figura Mapas de Karnaugh para 3 variables

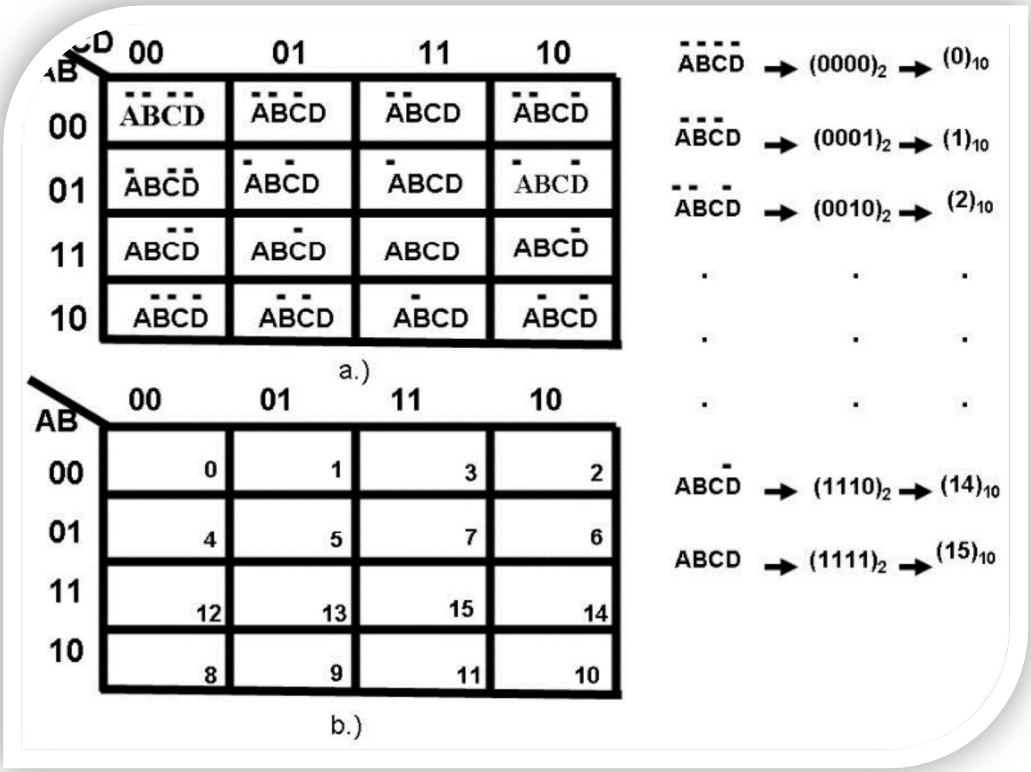


Figura Mapas de Karnaugh para 4 variables

Procedimiento de reducción utilizando mapas de Karnaugh

El proceso de reducción de una expresión booleana utilizando mapas de Karnaugh consiste de la aplicación de los pasos siguientes:

Paso 1

Definir el tamaño del Mapa de Karnaugh

El tamaño del mapa de Karnaugh se define en función del número de las variables de entrada (n) que forman la expresión booleana, por ejemplo si se tienen 3 (n=3) variables, el tamaño del mapa de Karnaugh es de 8 (2n) celdas contiguas, si tuviera cuatro variables de entrada (n = 4) se forma o construye un Mapa de Karnaugh de 16 celdas (24 = 16), etc.

Paso 2

Depositar en cada una de las celdas el valor de “1” donde la función es verdadera y el valor de 0 en las celdas donde la función es falsa. Por claridad únicamente se depositan los “1”s.

Paso 3

Realizar encierros de cajas o celdas (cuyo contenido sea “1”) adyacentes y contiguos de tamaño 2^n, 2^{n-1},2^{n-2},…,2^0., cuyos contenidos tengan el valor de uno. Los encierros de celdas se deben realizar a partir de la potencia de 2 más alta y posteriormente se realizan encierros de una potencia de 2 menor que la anterior y así sucesivamente hasta 2^0.

Los encierros o agrupaciones de cajas o celdas adyacentes se realizan en cantidades de términos mínimos que deben ser potencias de dos, tales como 1, 2, 4 y 8. Estos grupos se conocen con el nombre de implicantes primos.

Las variables booleanas se van eliminando a medida que se logra el aumento de tamaño de estos grupos. Con el objeto de mantener la propiedad de adyacencia, la forma del grupo debe ser siempre rectangular, y cada grupo debe contener un número de celdas que corresponda a una potencia entera de dos.

Los unos adyacentes de un mapa Karnaugh de la figura Mapas de Karnaugh para 3 variables satisfacen las condiciones requeridas para aplicar la propiedad de complemento del álgebra de Boole. Dado que en el mapa Karnaugh de la figura Mapas de Karnaugh para 3 variables existen unos adyacentes, puede obtenerse una simplificación sencilla.

Paso 4

Se obtiene la función booleana reducida a partir de cada uno de los grupos (encierros) formados en el punto 3.

Paso 5

Realizar el diagrama lógico de la función reducida.

Para mostrar el procedimiento exponemos una serie de ejemplos a continuación.

Ejemplo: Utilizando los mapas de Karnaugh reduce la siguiente función booleana

Solución:

$$f(A,B,C) = \sum(3,5,6,7)$$

Paso 1

Esta función booleana depende de 3 variables (A, B y C) por lo tanto tenemos un mapa de Karnaugh de 8 celdas como se muestra en la figura Mapas de Karnaugh para 3 variables.

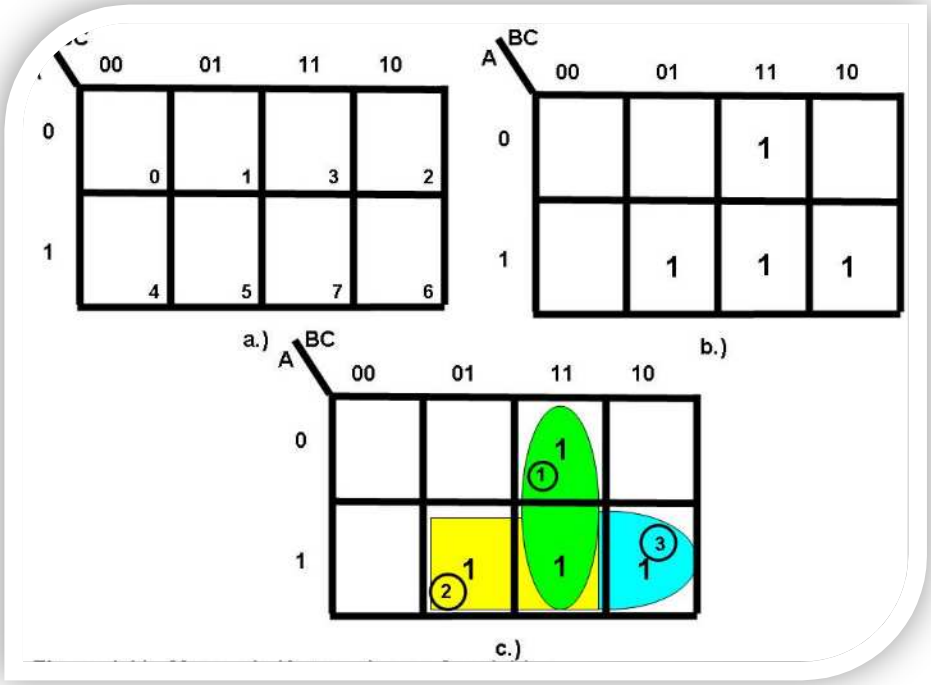


Figura Mapas de Karnaugh para 3 variables a-c

Nota: Cada una de las celdas que forman el mapa de Karnaugh se puede enumerar con la facilidad de vaciar el valor de “1“en cada una de las celdas, ver figura Mapas de Karnaugh para 3 variables a.

Paso 2

En cada una de las celdas que forman el mapa de Karnaugh se coloca el valor de 1 cuyos términos en la función sean verdaderos. A partir de la función observamos los términos que son verdaderos (3, 5, 6 y 7) y los términos que no son verdaderos (0, 1, 2 y 4), por claridad no se colocan los ceros, ver figura Mapas de Karnaugh para 3 variables b.

Paso 3

Agrupar las celdas en grupos de tamaño 2^n

Para agrupar (o realizar los encierros) las celdas cuyo contenido es uno, se agrupan las mismas en potencia de 2, a partir de la potencia mayor hacia una potencia menor o viceversa. En nuestro ejemplo utilizamos la primera forma, es decir, de mayor a menor. Empezamos preguntándonos si se pueden formar grupos de 8 celdas cuyo contenido es uno. No. Si la respuesta es No, preguntamos nuevamente. ¿Se pueden formar grupos de 4 celdas cuyo contenido es uno? No. Si la respuesta es No, preguntamos nuevamente. ¿Se pueden formar grupos de 2 celdas cuyo contenido es uno? Sí. Si la respuesta es Sí. Enumeramos todos los encierros de dos celdas formados en nuestro caso tenemos tres encierros de 2 celdas cada uno, ver figura Mapas de Karnaugh para 3 variables c. Y preguntamos nuevamente. ¿Se pueden formar grupos de una 1 celda cuyo contenido es uno? No. Si la respuesta es No, empezamos a obtener cada término de la función reducida a partir de todos los encierros encontrados.

Paso 4

Se obtiene la función booleana reducida a partir de cada uno de los grupos (encierros) formados en el punto 3. En este ejemplo, se formaron tres grupos de dos celdas cada uno, como se muestra en la figura Mapas de Karnaugh para 3 variables c. Cada celda con

un uno tiene al menos una celda vecina con un 1, por lo que no quedaron grupos de una celda. Al analizar los grupos formados por dos celdas, se observa que todos los elementos unitarios se encuentran cubiertos por grupos de dos elementos. Una de las celdas se incluye en los tres “encierros”, lo que es permitido, en el proceso de reducción.

Para obtener la función lógica reducida procedemos de la manera siguiente. El primer grupo (encierro 1) nos proporciona el término: AC, el segundo grupo (encierro 2) nos proporciona el término: BC y finalmente el tercer grupo (encierro 3): AB, que finalmente agrupando los tres términos tenemos la función booleana reducida siguiente:

$$f(A,B,C) = AC + BC + AB$$

Paso 5

Finalmente a partir de la ecuación reducida construimos el circuito lógico correspondiente, el cual se muestra en la figura Circuito lógico.

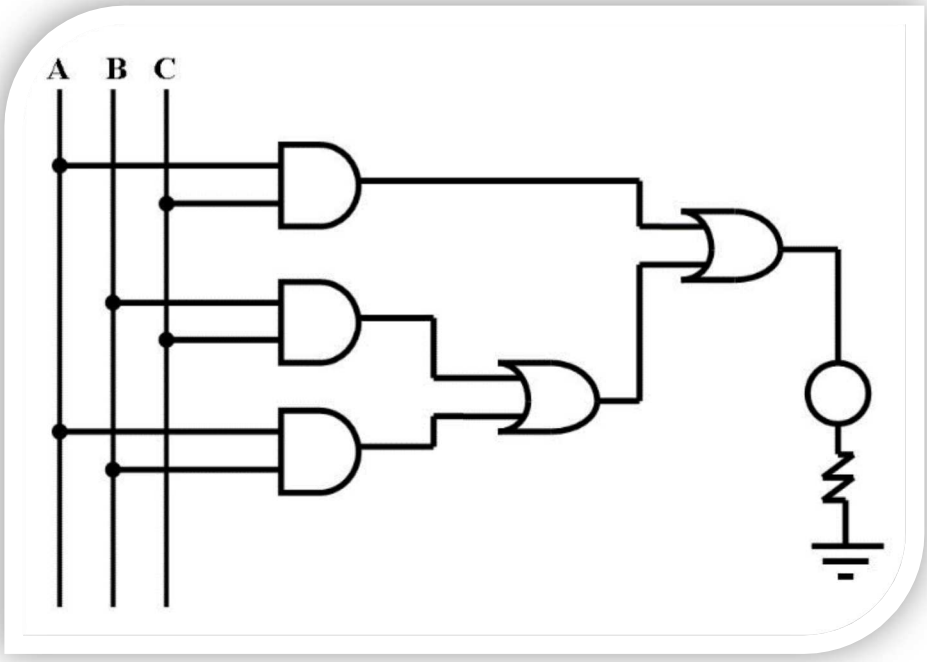


Figura Circuito lógico

Ejemplo: Utilizando los mapas de Karnaugh reduce la siguiente función

Solución:

$$f(A,B,C) = \bar{A}\,\bar{B}\,\bar{C} + A\,\bar{B}\,\bar{C} + \bar{A}\,B\,\bar{C} + A\,B\,\bar{C} + \bar{A}\,B\,C + A\,\bar{B}\,C + A\,B\,C$$

a partir de la función tenemos los términos y su equivalencia en binario y decimal.

$\bar{A}\,\bar{B}\,\bar{C} = (000)2 = (0)10$ $A\,\bar{B}\,\bar{C} = (001)2 = (1)10$ $\bar{A}\,B\,\bar{C} = (010)2 = (2)10$

$\bar{A}\,\bar{B}\,C = (100)2 = (4)10$ $A\,\bar{B}\,C = (101)2 = (5)10$ $\bar{A}\,B\,C = (110)2 = (6)10$

con lo cual la función se puede escribir de la manera siguiente:

$$f(A,B,C) = \sum(0,1,2,4,5,6)$$

En conclusión tenemos dos formas de colocar los 1 en cada una de las celdas del mapa de Karnaugh y son utilizando los términos de la expresión o utilizando la forma canónica de la función por reducir.

Nota: La representación de una función lógica a base de “1” se llama *forma canoníca* (lógica positiva).

Paso 1 Definir el tamaño del Mapa de Karnaugh

El tamaño del mapa de Karnaugh se define en función del número de las variables de entrada. Para este ejemplo se tienen 3 variables, el tamaño del mapa de Karnaugh es de 8 celdas, ver figura Mapas de Karnaugh para 3 variables

Paso 2 Vaciar los términos verdaderos en el mapa

Depositar en cada una de las celdas el valor de 1 donde la función es verdadera y el valor de 0 en las celdas donde la función es falsa. Por comodidad únicamente se depositan los 1’s. El mapa de Karnaugh con los “1”s en sus celdas es el siguiente

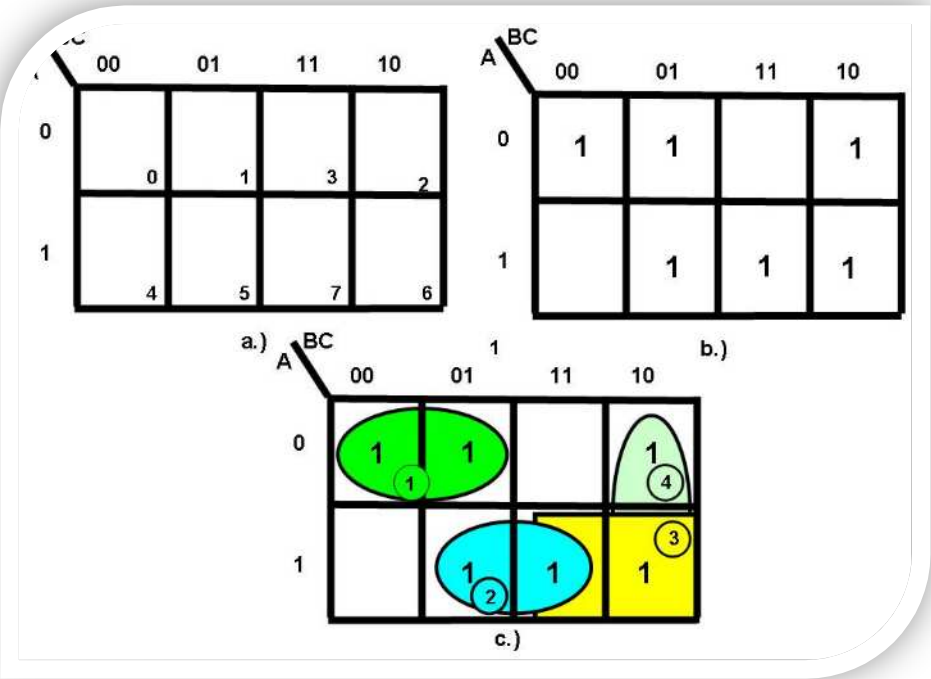


Figura Mapas de Karnaugh para 3 variables

Paso 3 Agrupar las celdas en grupos de tamaño 2^n

Para agrupar las celdas cuyo contenido es uno, se agrupan a partir de la potencia mayor hacia una potencia menor. Empezamos preguntándonos ¿se pueden formar grupos de 8 celdas cuyo contenido es uno? -No. Si la respuesta es No, preguntamos nuevamente. ¿Se pueden formar grupos de 4 celdas cuyo contenido es uno? -No. Si la respuesta es No, preguntamos nuevamente. ¿Se pueden formar grupos de 2 celdas cuyo contenido es uno? -Sí. Si la respuesta es Sí, numeramos todos los encierros de dos celdas formados; en nuestro caso tenemos cuatro encierros de 2 celdas cada uno, ver figura Mapas de Karnaugh para 3 variables c. Y preguntamos nuevamente. ¿Se pueden formar grupos de una 1 celda cuyo contenido es uno? No. Si la respuesta es No, empezamos a obtener cada término de la función reducida a partir de todos los encierros encontrados.

Paso 4

Se obtiene la función booleana reducida a partir de cada uno de los encierros formados en el punto 3. En este ejemplo, se formaron cuatro encierros de dos celdas cada uno, como se muestra en la figura Mapas de Karnaugh para 3 variables c. Cada celda con un uno tiene al menos una celda vecina con un 1, por lo que no quedaron grupos de una celda. Al analizar los grupos formados por dos celdas, se observa que todos los elementos unitarios se encuentran cubiertos por grupos de dos elementos. Dos celdas (celda 6 y 7) se incluyen en dos “encierros”, lo que es permitido, en el proceso de reducción.

Para obtener la función lógica reducida procedemos de la manera siguiente:

Encierro 1 nos proporciona el término: $\overline{A}\,\overline{B}$

Encierro 2 nos proporciona el término: AC

Encierro 3 nos proporciona el término: AB

Encierro 4 nos proporciona el término: $B\overline{C}$

$$f(A,B,C) = \overline{A}\,\overline{B} + A\,C + A\,B + B\overline{C}$$

Paso 5 Realizar el diagrama lógico de la función reducida

Finalmente a partir de la ecuación reducida construimos el circuito lógico correspondiente, el cual se muestra en la figura **Circuito lógico**.

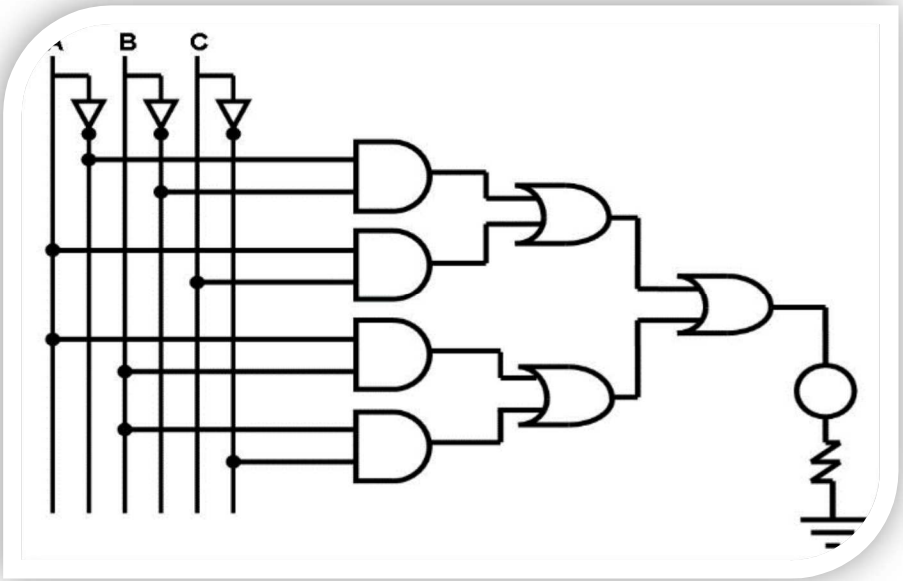


Figura Circuito lógico

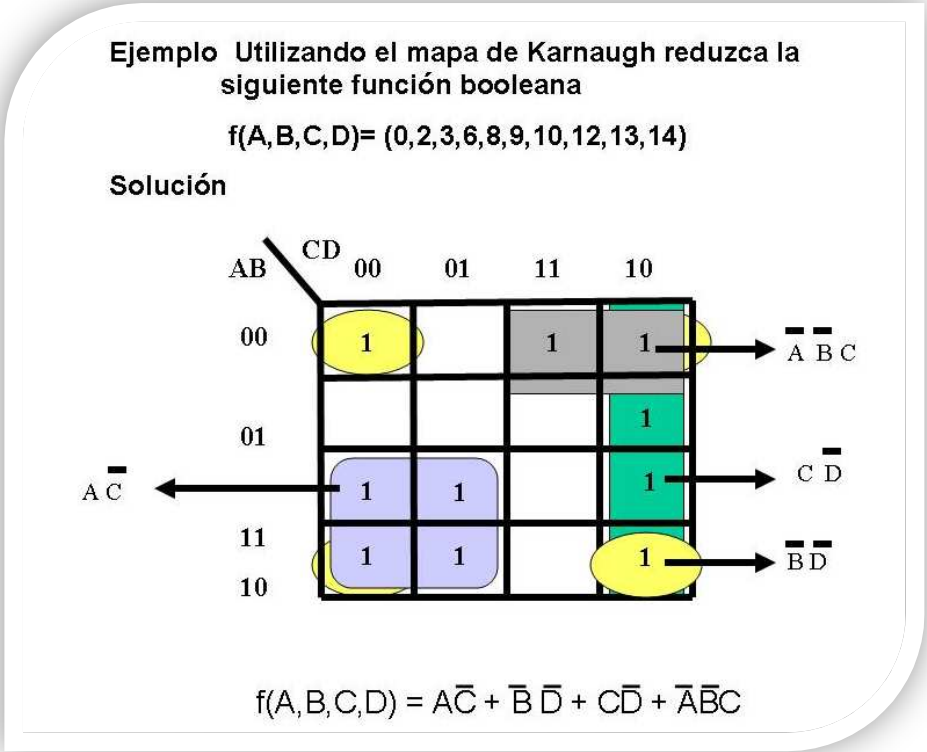


Figura Mapa de Karnaugh para 4 variables f(A, B, C, D)

Ejemplo

Utilizando mapas de Karnaugh reduzca la siguiente función booleana:

F(A,B,C,D,E)= (2,5,6,7,8,9,10,12,13,14,18,21,22,23,24,25,26,18,19,30)

Solución

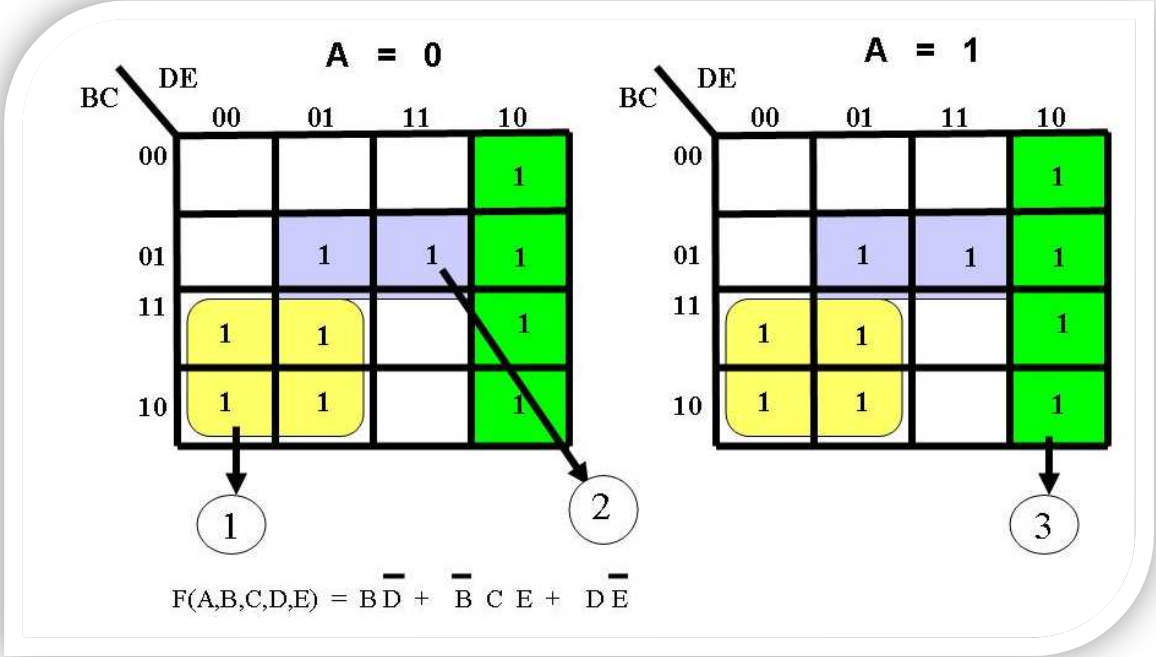


Figura Mapa de Karnaugh para 5 variables

RESUMEN

Inicialmente se presentaron los elementos y axiomas del álgebra de Boole:

Para el álgebra booleana, el conjunto de valores es el conjunto que contiene los elementos cero y uno. Las operaciones definidas son AND, OR y NOT.

OR	AND	NOT
• El operador OR (O) designado también como + es la representa la operación de “suma binaria” no es una suma en el sentido aritmético, sino lógico. C=A + B significa que la variable C será válida cuando alguna de las dos variables A o B sean válidas.	• El operador AND (Y) designado también como es la representación de la multiplicación “binaria” lógica. C es válida cuando las dos variables A y B (ambas) sean válidas.	• El operador lógico NOT (negación) significa que B es igual a A negada. O bien que cuando A es falsa, B es cierta.

Los axiomas del álgebra booleana son:

- Cerradura. Para los operadores binarios AND y OR
- La ley asociativa la cual no se establece en los postulados de Huntington, sin embargo si se cumple en el álgebra booleana.
- Ley conmutativa.
- La ley distributiva de + sobre. no se cumple en el álgebra ordinaria y sí en la booleana.
- El álgebra de Boole no posee elementos inversos aditivos o multiplicativos, por lo que no existe la operación de resta o multiplicación.

- Existencia de elementos identidad e inverso, este último define los elementos llamados complementos, los cuales no existen en el álgebra ordinaria.
- Los elementos del álgebra ordinaria están dentro del conjunto de los números reales, mientras que los elementos del álgebra booleana sólo son el uno y el cero.

Los principales teoremas del álgebra booleana son:

- Idempotencia, de absorción, leyes de De Morgan, teorema de adyacencia y teorema de dualidad. Estos teoremas nos permiten la manipulación de funciones, por ejemplo para encontrar funciones complementos.
- Las funciones booleanas se pueden representar de varias formas: tablas de verdad, canónica, normalizada, mínima, como suma de productos y como producto de sumas. La manipulación algebraica nos permite transformar una presentación en otra de acuerdo a las condiciones del problema. Sin embargo el manejo algebraico siempre representa un proceso laborioso y a veces complicado. Para minimizar funciones se pueden emplear los mapas de Karnaugh que son una aplicación sistemática del teorema de adyacencia a partir de una representación gráfica de funciones basada en los diagramas de Venn. De esta manera la minimización de funciones es una tarea más sencilla. Es importante entender a los mapas de Karnaugh como una forma más de representar funciones boolenas.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo	Páginas
Quiroga (2010)	5	93-102
Mano (1986)	1	26-32
Stallings (2006)	Apéndice B	733-737

Mano, Morris. (1986). *Lógica Digital y diseño de computadores*. México: Prentice Hall Hispanoamericana.

Quiroga, Patricia. (2010). *Arquitectura de computadoras*. México: Alfaomega.

Stallings, Williams. (2006). *Organización y arquitectura de computadores*. Madrid: Prentice Hall.

Unidad 5.

Circuitos combinatorios o combinacionales

OBJETIVO PARTICULAR

Al finalizar la unidad, el alumno podrá reconocer el funcionamiento y la construcción de sumadores, decodificadores y multiplexores a partir de compuertas básicas; diseñar circuitos combinacionales mediante compuertas digitales y deducirá la expresión algebraica a partir de un circuito digital.

TEMARIO DETALLADO
(10 horas)

5. Circuitos combinatorios o combinacionales

5.1. Multiplexores
5.2. Demultiplexores
5.3. Codificadores
5.4. Decodificadores
5.5. Medio Sumador
5.6. Sumador completo
5.7. Restadores
5.8. Comparadores.

INTRODUCCIÓN

Los circuitos combinatorios o circuitos combinacionales transforman un conjunto de entradas en un conjunto de salidas de acuerdo con una o más funciones lógicas. Las salidas de un circuito combinacional son rigurosamente función de las entradas y se actualizan después de cualquier cambio en las entradas. La figura Diagrama en bloques de una unidad lógica combinacional, ilustra un modelo de unidad lógica combinacional.

Esta unidad combinacional recibe un conjunto de entradas i0,…,in y produce un conjunto de salidas f0,…,fm, las que dependerán de las funciones lógicas correspondientes. En este tipo de circuito combinacional no existe retroalimentación de las salidas sobre las entradas como en el caso de los circuitos secuenciales (ver Unidad 6).

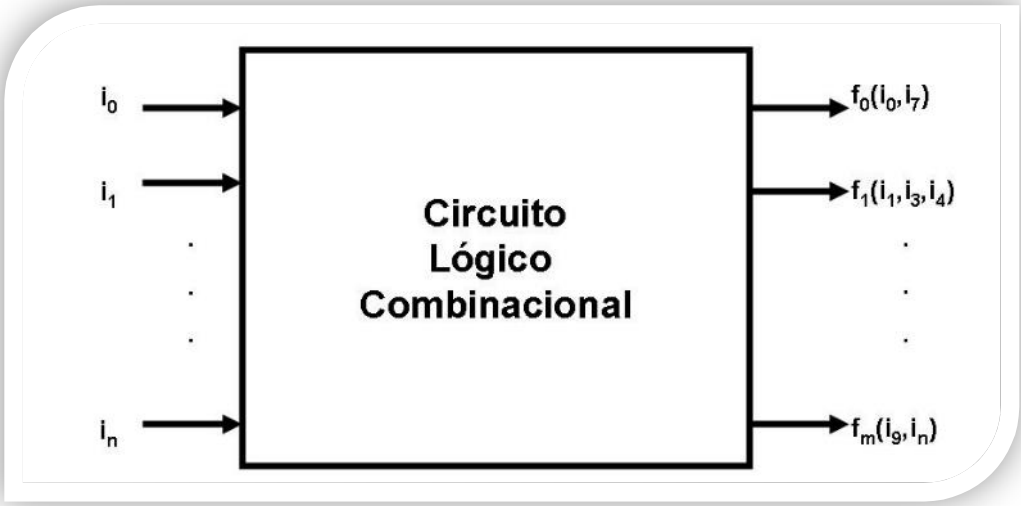


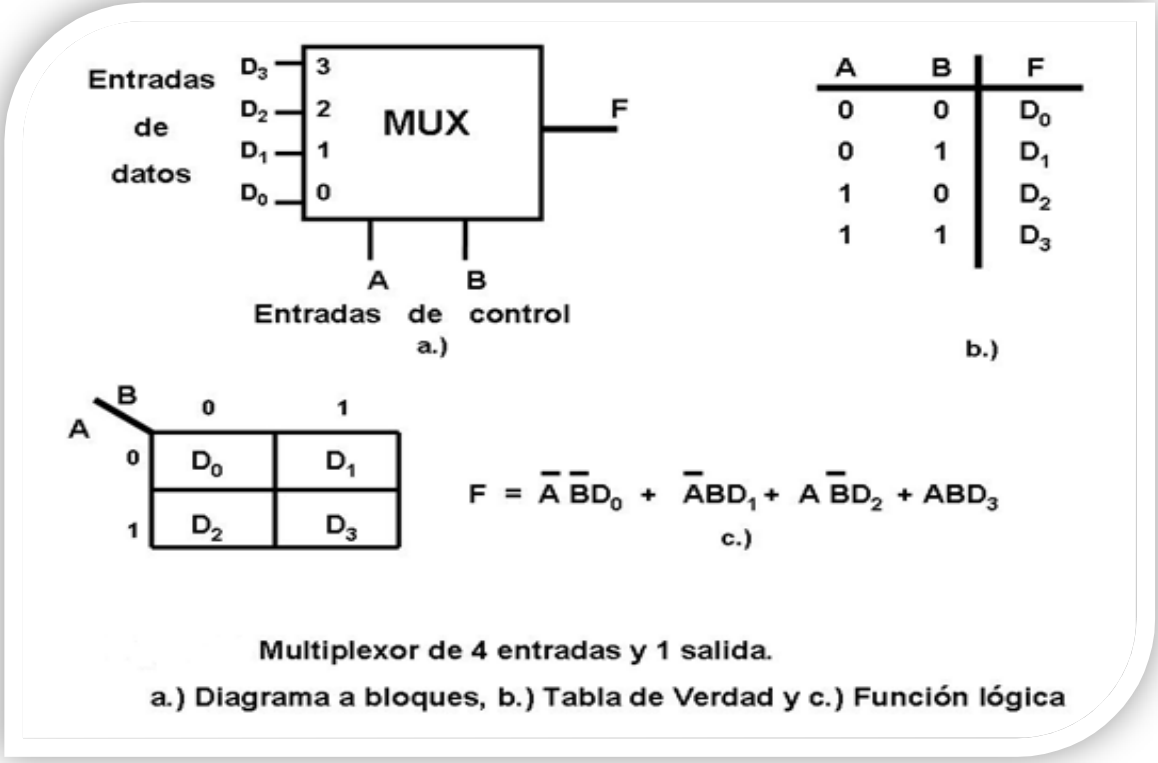
Diagrama en bloques de una unidad lógica combinacional

Un circuito combinacional recibe entradas y genera salidas en las cuales es habitual considerar como valor bajo el “0” lógico ó 0 Volts, en tanto que se adopta como valor

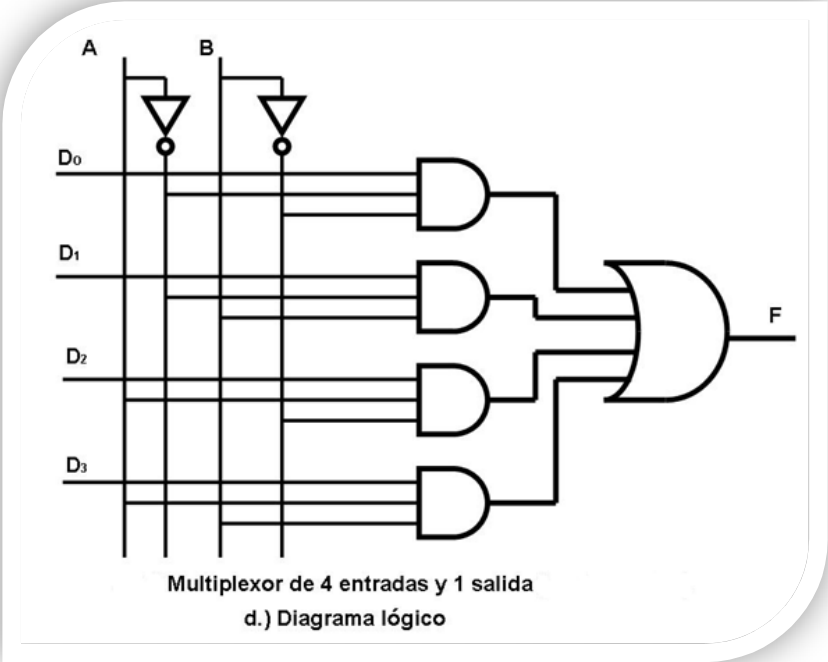
alto el “1” lógico ó 5 Volts. Esta convención no es de uso universal. En los circuitos de alta velocidad se tiende a usar menores valores de tensión. Algunos circuitos de computadora funcionan en el dominio analógico, en el que se admite una variación continua de las señales, y en el caso de los circuitos digitales ópticos se puede utilizar variaciones de fase o polarizaciones, por lo que no es necesario plantear los conceptos de alto y bajo en este momento.

5.1. Multiplexores

Un circuito multiplexor (MUX) es un elemento que conecta una cantidad dada de entradas a una única salida. La figura Multiplexor 4 entradas 1 salida muestra el diagrama en bloques y la tabla de verdad de un multiplexor de 4 entradas y 1 salida. La salida *F* adopta el valor correspondiente a la entrada de datos seleccionada por las líneas de control *A* y *B*. Por ejemplo, si A = 0 y B = 1, el valor que aparece en la salida es el que corresponde a la entrada D_1, ver figura Multiplexor 4 entradas 1 salida. b.) Tabla de Verdad. En la figura Multiplexor 4 entradas 1 salida. c.) Función lógica se muestra la obtención de la función lógica del multiplexor a partir de su tabla de verdad y en la figura Multiplexor 4 entradas 1 salida. d.) Diagrama lógico se presenta el diagrama lógico del multiplexor.

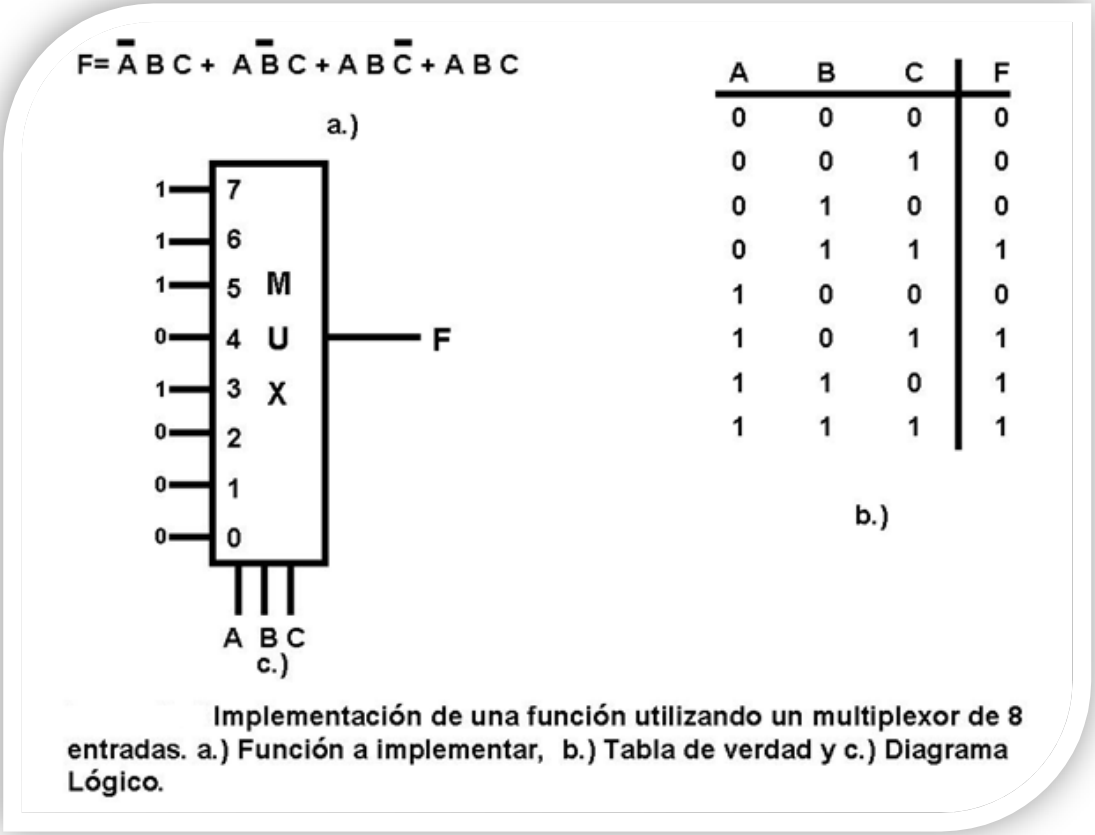


Multiplexor de 4 entradas y 1 salida.
a.) Diagrama a bloques, b.) Tabla de Verdad y c.) Función lógica



Multiplexor de 4 entradas y 1 salida
d.) Diagrama lógico

Una aplicación de los multiplexores es la implementación de funciones lógicas como se muestra en la figura Implementación de una función utilizando un multiplexor de 8 entradas. En dicha figura se desea implementar una función lógica usando un multiplexor de 8 entradas y 1 salida. Las entradas de datos se toman directamente de la tabla de verdad de la función por implementar y se asignan las variables A, B y C como entradas de control. El multiplexor transfiere a la salida los unos correspondientes a cada término mínimo de la función. Las entradas cuyos valores son 0 corresponden a los elementos del multiplexor que no se requieren para la implementación de la función, y como resultado hay compuertas lógicas que no se utilizan. Si bien en la implementación de funciones booleanas siempre hay porciones del multiplexor que no se utilizan, el uso de multiplexores es amplio debido a que su generalidad simplifica el proceso de diseño y su modularidad simplifica la implementación.

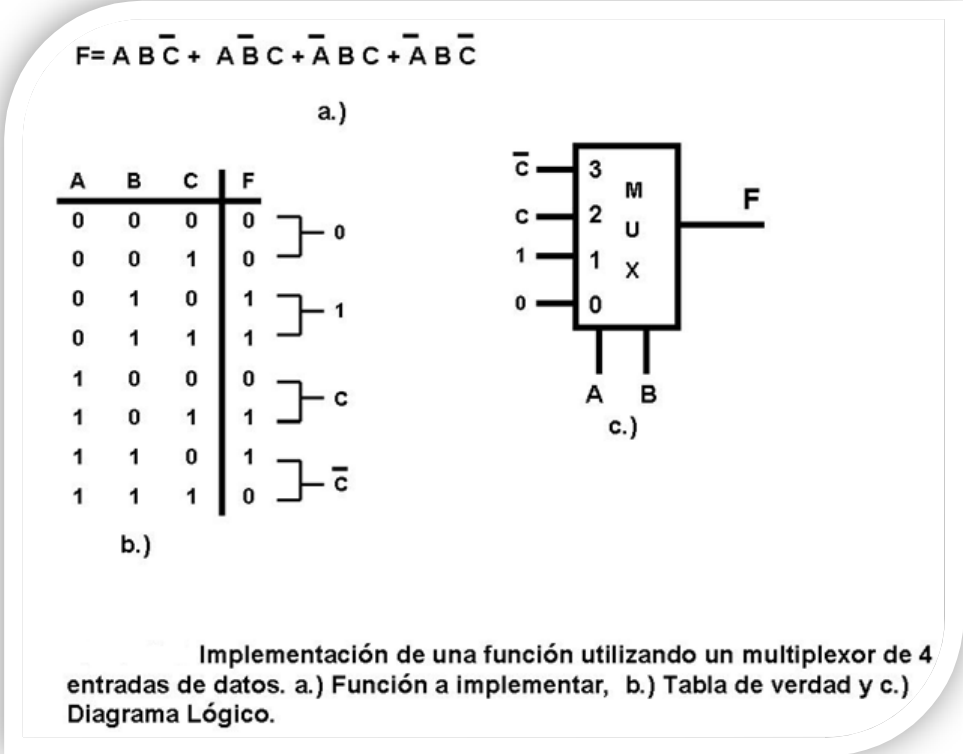


Implementación de una función utilizando un multiplexor de 8 entradas. a.) Función a implementar, b.) Tabla de verdad y c.) Diagrama Lógico.

Otro ejemplo del uso de los multiplexores en la implementación de funciones lógicas es similar al que se muestra en la figura Implementación de una función utilizando un multiplexor de 4 entradas de datos. La figura Implementación de una función utilizando un multiplexor de 4 entradas de datos b) Tabla de verdad ilustra la tabla de verdad de tres variables de la función lógica por implementar (ver, figura Implementación de una función utilizando un multiplexor de 4 entradas de datos a) Función por implementar) y el multiplexor de 4 entradas utilizado en la implementación de la función lógica. Las entradas de datos se toman del conjunto {0, 1, C, C} y la agrupación se obtiene de acuerdo con lo que se muestra en la tabla de verdad. Cuando A = 0, B = 0, la función F = 0 independientemente del valor de C, y por lo tanto, la entrada de datos 00 del multiplexor tendrá un valor fijo de 0, cuando A = 0, B = 1, F = 1, independientemente del valor de la variable C, por lo que la entrada de datos 01 adopta un valor de 1.

Cuando A = 1,

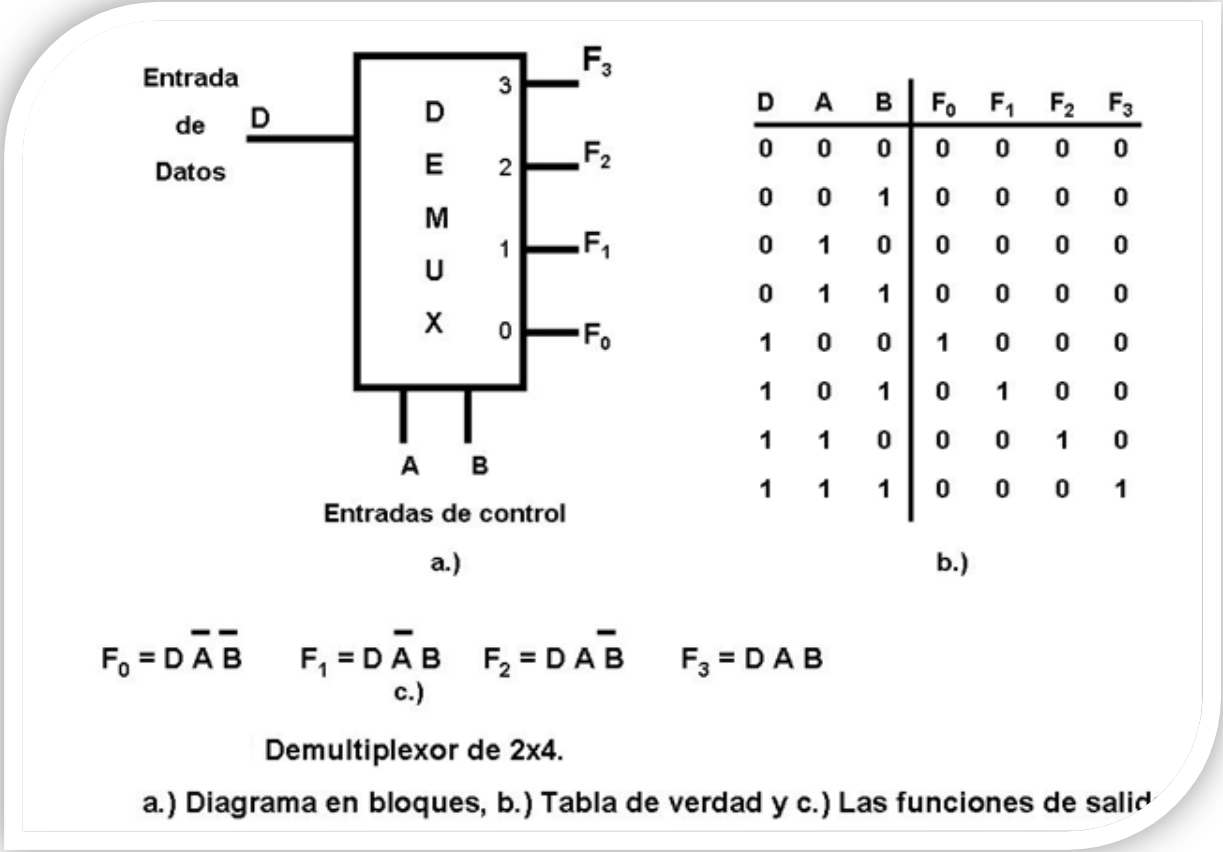
B = 0, la función F = C dado que su valor es 0 cuando C es 0 y es 1 cuando C es 1. Finalmente, cuando A = 1, B = 1, la función F = C, por lo tanto, la entrada de datos 11 adopta el valor de C. De esta manera, se puede implementar una función de tres variables usando un multiplexor con cuatro entradas de datos y dos entradas de control.



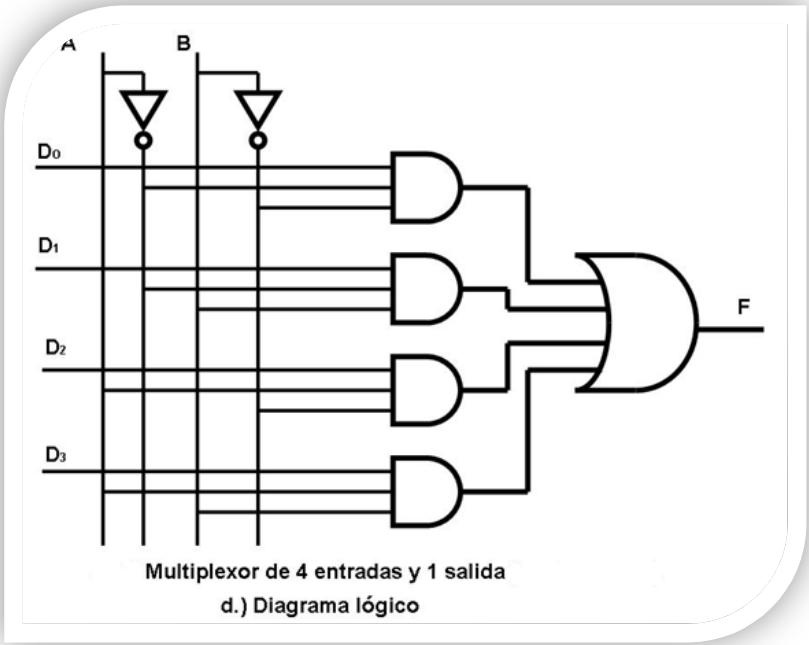
Implementación de una función utilizando un multiplexor de 4 entradas de datos. a.) Función a implementar, b.) Tabla de verdad y c.) Diagrama Lógico.

5.2. Demultiplexores

Un demultiplexor (DEMUX) es un circuito que cumple la función inversa a la de un multiplexor. La figura Demultiplexor de 2x4 ilustra el diagrama en bloques correspondientes a un demultiplexor de cuatro salidas, cuyas entradas de control son A y B, su correspondiente tabla de verdad, su función lógica y su diagrama lógico. Un demultiplexor envía su única entrada de datos D a una de sus Fi salidas de acuerdo con los valores que adopten sus entradas de control. La figura Demultiplexor de 2x4 muestra el circuito de un demultiplexor de cuatro salidas.



D	A	B	F_0	F_1	F_2	F_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	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	0	1



Multiplexor de 4 entradas y 1 salida
d.) Diagrama lógico

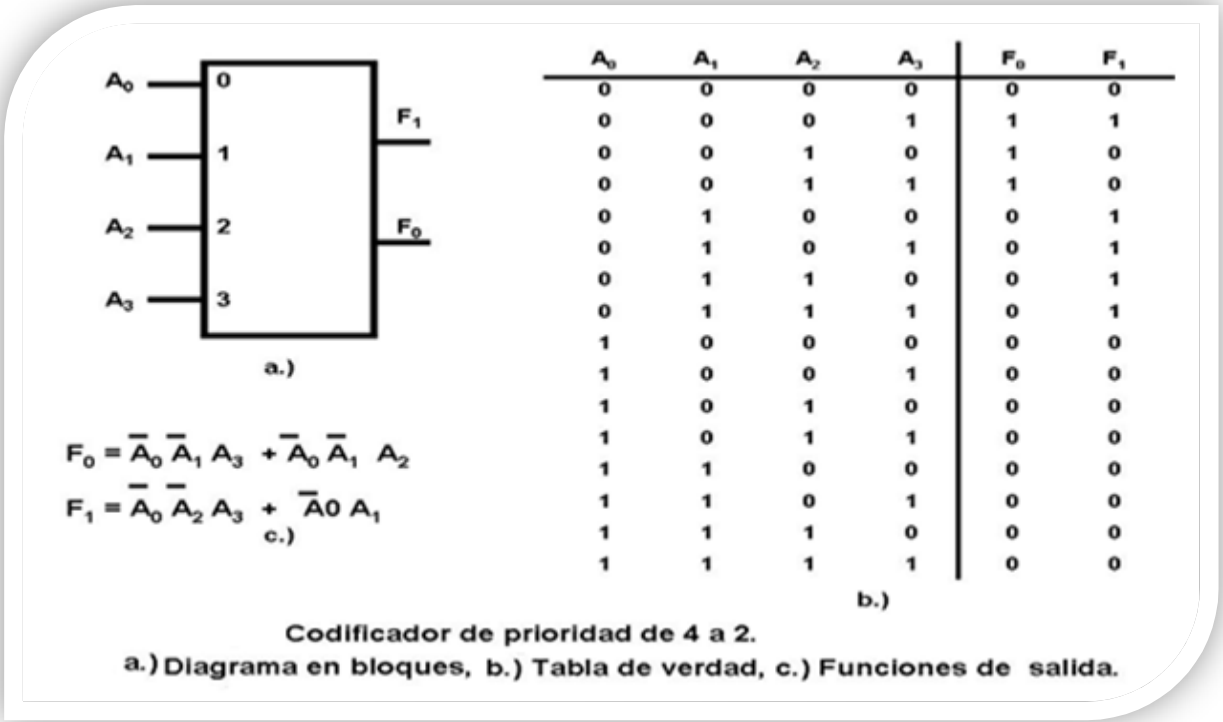
5.3. Codificadores

Un codificador tiene 2^n (o menos) líneas de entrada y n líneas de salida. Las líneas de salidas generan el código binario para las 2^n variables de entrada. Un ejemplo de un circuito codificador es el codificador de prioridad.

Un codificador de prioridad es un codificador en el que se establece un ordenamiento de las entradas. El diagrama en bloques y la tabla de verdad de un codificador de prioridad de 4 entradas a 2 salidas se muestra en la figura Codificador de prioridad de 4 a 2. El esquema de prioridades impuesto sobre las entradas hace que A_i tenga una prioridad mayor que A_{i+1}. La salida de dos bits adopta los valores 0_{10}, 1_{10},2_{10} u 3_{10}, dependiendo de las entradas activas y de sus prioridades relativas. Cuando no hay

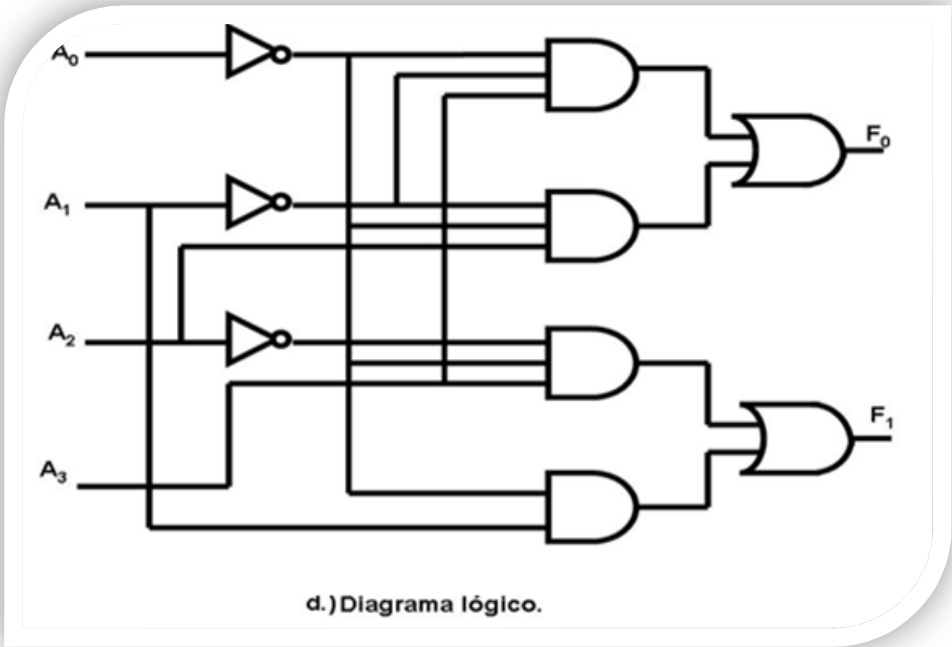
entradas activas, las salidas llevan, por defecto, a asignarle prioridad a la entrada A_0 ($F_0 = 0$ y $F_1 = 0$).

Los codificadores de prioridad se utilizan para arbitrar entre una cantidad de dispositivos que compiten por un mismo recurso, como cuando se produce el intento de acceso simultáneo de una cantidad de usuarios a un sistema de computación. La figura Codificador de prioridad de 4 a 2. c) Función de verdad ilustra el diagrama lógico para un codificador de prioridad de 4 entradas y 2 salidas.



A_0	A_1	A_2	A_3	F_0	F_1
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

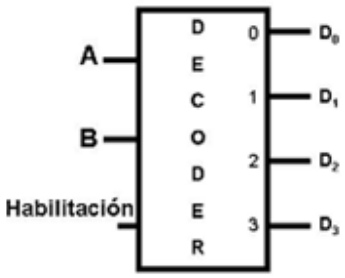
Codificador de prioridad de 4 a 2.
a.) Diagrama en bloques, b.) Tabla de verdad, c.) Funciones de salida.



d.) Diagrama lógico.

5.4. Decodificadores

Un decodificador traduce una codificación lógica binaria hacia una ubicación espacial. En cada momento, solo una de las salidas del decodificador está en el estado activo (“1” lógico), según lo que determinen las entradas de control. La figura Decodificador 2 a 4 muestra el diagrama en bloques, la tabla de verdad de un decodificador de 2 entradas a 4 salidas, cuyas entradas de control son *A* y *B*. El diagrama lógico correspondiente a la implementación del decodificador se muestra en la figura Decodificador 2 a 4 c) Funciones de salida. Un circuito decodificador puede usarse para controlar otros circuitos, aunque a veces resulta inadecuado habilitar cualquiera de esos otros circuitos. Por esta razón, se incorpora en el circuito decodificador una línea de habilitación, la que fuerza todas las salidas a nivel “0” (inactivo) cuando se le aplica un “0” en la entrada.



a.)

$D_0 = \overline{A}\,\overline{B}$ $D_1 = \overline{A}B$

$D_2 = A\overline{B}$ $D_3 = A\,B$

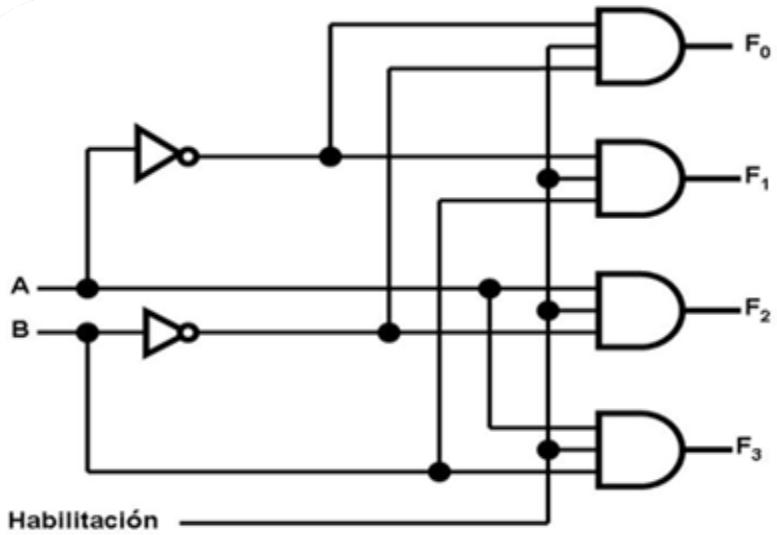
c.)

Habilitación = 1					
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Habilitación = 0					
A	B	D_0	D_1	D_2	D_3
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0

b.)

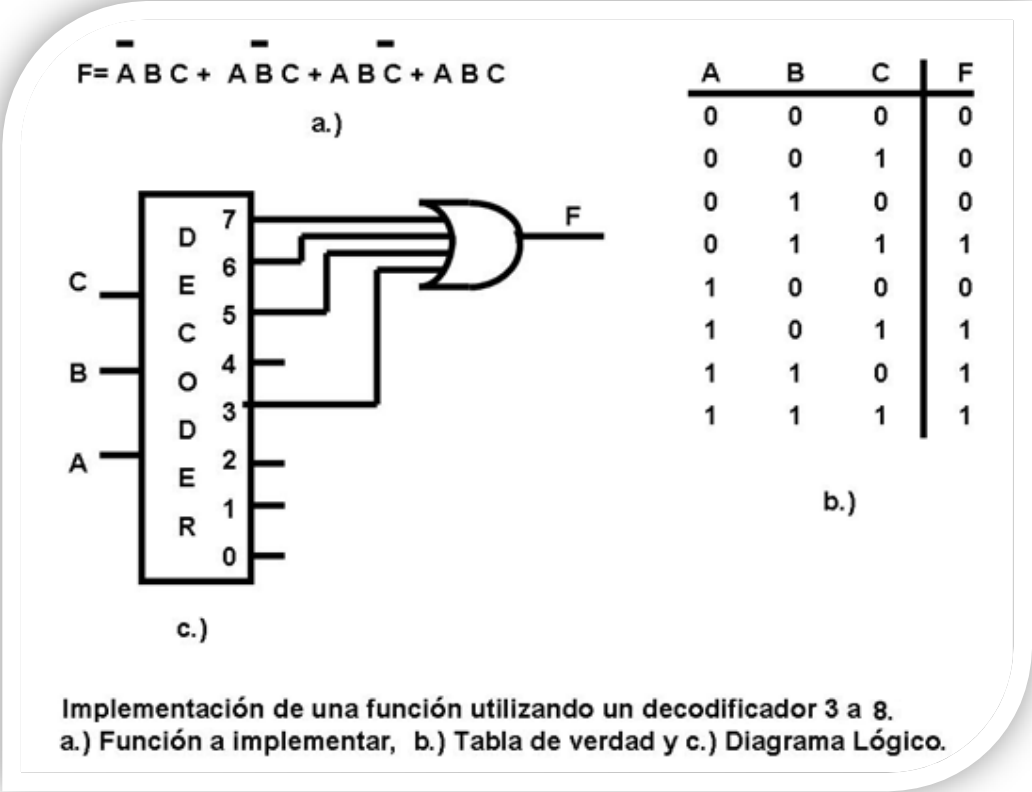
Decodificador 2 a 4. a.)Diagrama a bloques, b) Tabla de verdad y c.) funciones de salida.



decodificador 2 a 4

d.) Implementación de un decodificador 2 a 4.

Una aplicación para un circuito decodificador puede ser la traducción de direcciones de memoria a sus correspondientes ubicaciones físicas o para la implementación de funciones lógicas. Para el caso de implementación de funciones, dado que cada línea de salida corresponde a un término mínimo distinto, puede implementarse una función por medio de la suma lógica de las salidas correspondientes a los términos que son ciertos en la función. Por ejemplo en la figura Implementación de una función utilizando un decodificador 3 a 8 se puede ver la implementación de la función con un decodificador de 3 a 8. Las salidas no utilizadas se dejan desconectadas.



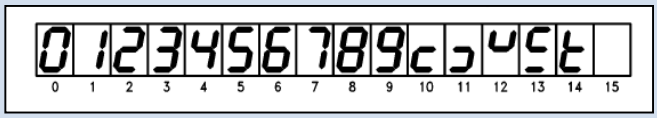
Implementación de una función utilizando un decodificador 3 a 8.
a.) Función a implementar, b.) Tabla de verdad y c.) Diagrama Lógico.

Diseño de un Decodificador BCD

Un decodificador también puede utilizarse en la visualización de información de un "formato" a otro "formato" como lo es desplegar información en un "Display" de 7 Segmentos. Este circuito decodifica la información cuya entrada está en BCD a un código de siete segmentos adecuado para que se muestre en un visualizador de siete segmentos. El diseño de dicho decodificador se presenta a continuación:

Se enuncia el problema

Diseñe un decodificador BDC a siete segmentos utilizando compuertas básicas



Se determina el número requerido de variables de entrada (*n*) y el número de funciones de salida (*N*).

$$n=4, N=2^n=2^4=16$$

Para representar 16 combinaciones (una por cada símbolo) necesitamos cuatro entradas y siete salidas.

Se le asigna letras a las variables de entrada y a las funciones de salida.

Entradas => A, B, C, y D

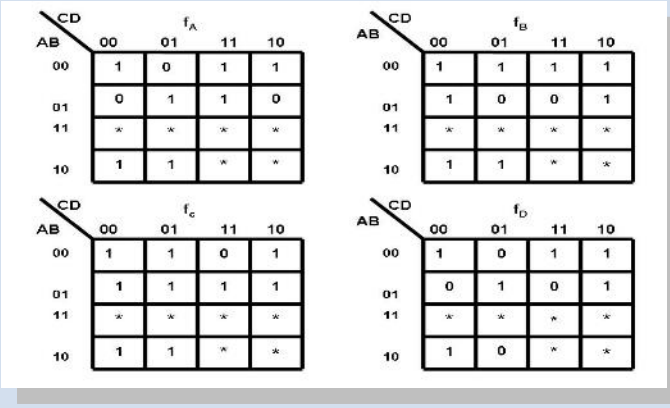
Salidas => f_a, f_b, f_c, f_d, f_e, f_f, y f_g.

Se deduce la tabla de verdad que define las relaciones entre las entradas y las salidas.

Entradas					Salidas						
A	B	C	D		a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0	0	0	0
0	0	1	0	2	1	1	0	1	1	0	1
0	0	1	1	3	1	1	1	1	0	0	1
0	1	0	0	4	0	1	1	0	0	1	1
0	1	0	1	5	1	0	1	1	0	1	1
0	1	1	0	6	0	0	1	1	1	1	1
0	1	1	1	7	1	1	1	0	0	0	0
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	0	0	1	1
1	0	1	0	10	0	0	0	1	1	0	1
1	0	1	1	11	0	0	1	1	0	0	1
1	1	0	0	12	0	1	0	0	0	1	1
1	1	0	1	13	1	0	0	1	0	1	1
1	1	1	0	14	0	0	0	1	1	1	1
1	1	1	1	15	0	0	0	0	0	0	0

TABLA DE VERDAD BCD 7 DE SEGMENTOS

Se obtiene la función de Boole simplificada, en este caso utilizamos el método de Karnaugh a cada una de las salidas.

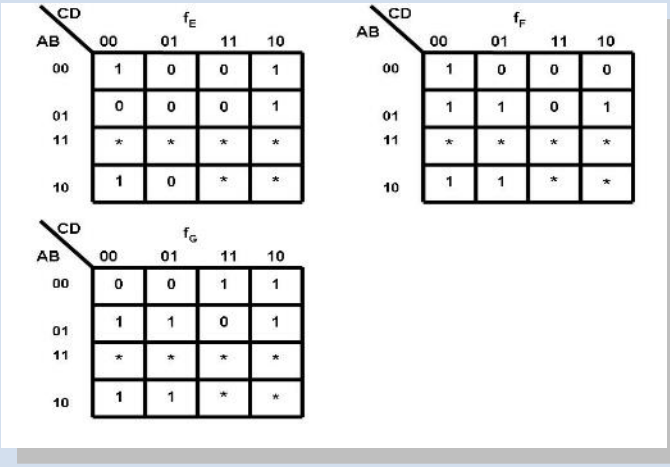


f_A	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	*	*	*	*
10	1	1	*	*

f_B	00	01	11	10
00	1	1	1	1
01	1	0	0	1
11	*	*	*	*
10	1	1	*	*

f_C	00	01	11	10
00	1	1	0	1
01	1	1	1	1
11	*	*	*	*
10	1	1	*	*

f_D	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	*	*	*	*
10	1	0	*	*

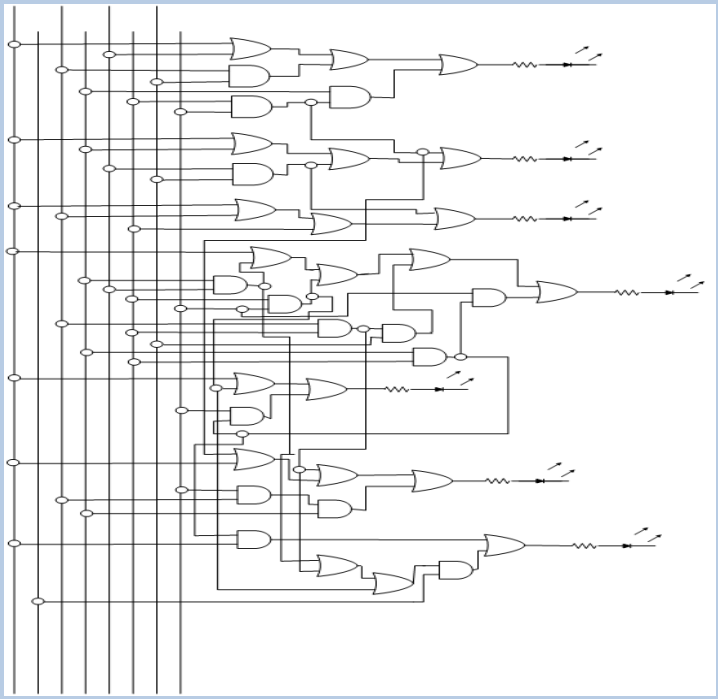


f_E	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	*	*	*	*
10	1	0	*	*

f_F	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	*	*	*	*
10	1	1	*	*

f_G	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	*	*	*	*
10	1	1	*	*

Se dibuja el diagrama lógico del decodificador 7 segmentos



5.5. Medio Sumador

El sumador binario es un circuito combinacional básico en una computadora digital. Este circuito combinacional tiene una característica importante, y es que trabaja en "cascada", es decir, puede realizar la suma de n-bits a la vez. Este sumador inicia con un circuito combinacional llamado *medio sumador* y le siguen n-1 *sumadores completos*. Para diseñar un sumador binario de n–bits, empezamos por definir qué es un medio sumador y un sumador completo para posteriormente diseñar un medio sumador y un sumador completo.

Definiciones:

Un **medio sumador** es un circuito combinacional que suma dos bits.

Un **sumador completo** es un circuito combinacional que suma tres bits.

Diseño de un medio sumador

Para diseñar el circuito combinacional denominado *medio sumador* partimos de que deseamos un sumador de dos números de 1 bit cada uno de ellos, y de esta manera tenemos las siguientes combinaciones:

Con 2 variables, se tienen 22 = 4 combinaciones

A_0	0	0	1	1
+	+	+	+	+
B_0	0	1	0	1
C_0 S_0	0 0	0 1	0 1	1 0
	C S	C S	C S	C S

Donde:

S es el bit del resultado de sumar dos bits, y

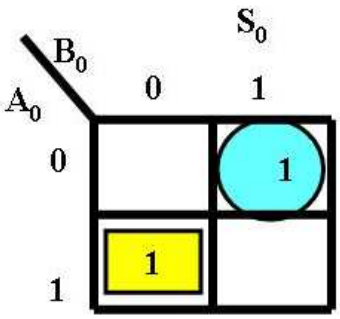
C es el bit de acarreo al momento de sumar dos bits

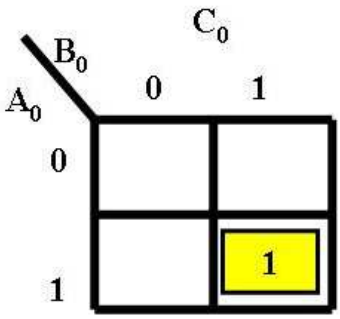
A partir de estos resultados obtenemos la tabla de verdad del medio sumador, la cual presentamos a continuación.

Tabla de verdad: Medio Sumador

A_0	B_0	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

A partir de la tabla de verdad, podemos encontrar la ecuación de salida para el resultado S_0 de la suma de dos bits, así como la ecuación de salida del bit de acarreo C_0 utilizando Mapas de Karnaugh, como se muestra en la figura Obtención de la ecuación de S_0 y C_0 utilizando mapas de Karnaugh.





$$S_0 = A_0\bar{B}_0 + \bar{A}_0B_0$$

$$S_0 = A_0 \oplus B_0$$

$$S_0 = A_0B_0$$

Obtención de la ecuación de S_0 y C_0 utilizando mapas de Karnaugh

La implementación (diagrama lógico) de la ecuación del medio sumador para S_0 y C_0 nos quedaría de la siguiente forma:

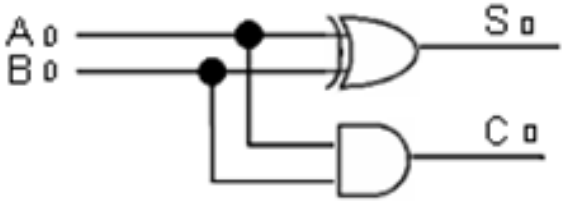


Diagrama lógico: Medio sumador

5.6. Sumador completo

Para diseñar el circuito combinacional llamado *sumador completo* partimos de que deseamos un sumador de tres números de 1 bit cada uno de ellos, y de esta manera tenemos las siguientes combinaciones:

3 variables – (2^3) = 8 Combinaciones

C_i	0	0	0	0
+ A_{i+1}	+ 0	+ 0	+ 1	+ 1
B_{i+1}	0	1	0	1
C_{i+1} S_i	0 0	0 1	0 1	1 0
	C S	C S	C S	C S
C_i	1	1	1	1
+ A_{i+1}	+ 0	+ 0	+ 1	+ 1
B_{i+1}	0	1	0	1
C_{i+1} S_{i+1}	0 1	1 0	1 0	1 1
	C S	C S	C S	C S

Donde

S_{i+1} es el bit del resultado de sumar tres bits, y

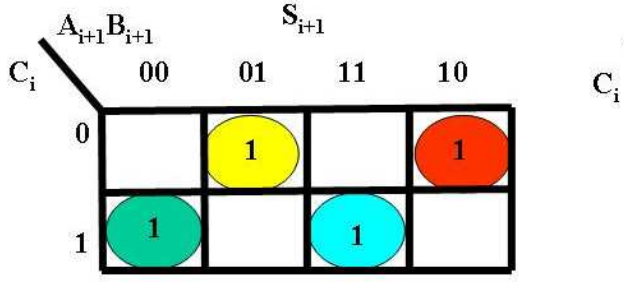
C_{i+1} es el bit de acarreo al momento de sumar tres bits.

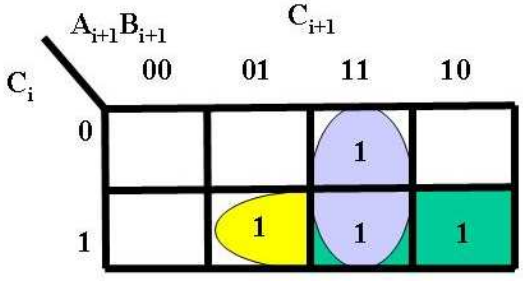
A partir de estos resultados obtenemos la tabla de verdad del sumador completo, la cual presentamos a continuación.

C_i	A_{i+1}	B_{i+1}	C_{i+1}	S_{i+1}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabla de verdad **Sumador completo**

A partir de la tabla de verdad, podemos encontrar la ecuación de salida para el resultado S_{i+1} de la suma de tres bits, así como la ecuación de salida del bit de acarreo C_{i+1} utilizando Mapas de Karnaugh, como se muestra en la figura Obtención de las ecuaciones de S_{i+1} y C_{i+1} empleando mapas de Karnaugh.





$$S_{i+1} = \overline{C}_{i+1}\overline{A}_{i+1}\overline{B}_{i+1} + \overline{C}_{i+1}A_{i+1}B_{i+1} + C_{i+1}A_{i+1}B_{i+1} + C_{i+1}\overline{A}_{i+1}\overline{B}_{i+1}$$

$$S_{i+1} = \overline{C}_{i+1}(\overline{A}_{i+1}\overline{B}_{i+1} + A_{i+1}B_{i+1}) + C_{i+1}(A_{i+1}\overline{B}_{i+1} + \overline{A}_{i+1}B_{i+1})$$

$$S_{i+1} = \overline{C}_{i+1}(A_{i+1} \overline{\oplus} B_{i+1}) + C_{i+1}(A_{i+1} \oplus B_{i+1})$$

$$S_{i+1} = C_{i+1} \oplus (A_{i+1} \oplus B_{i+1})$$

$$C_{i+1} = C_i B_{i+1} + A_{i+1}B_{i+1} + C_{i+1}A_{i+1}$$

$$C_{i+1} = C_i (A_{i+1} + B_{i+1}) + A_{i+1} B_{i+1}$$

La implementación (diagrama lógico) de la ecuación del sumador completo para S_{i+1} y C_{i+1} nos quedaría de la siguiente forma:

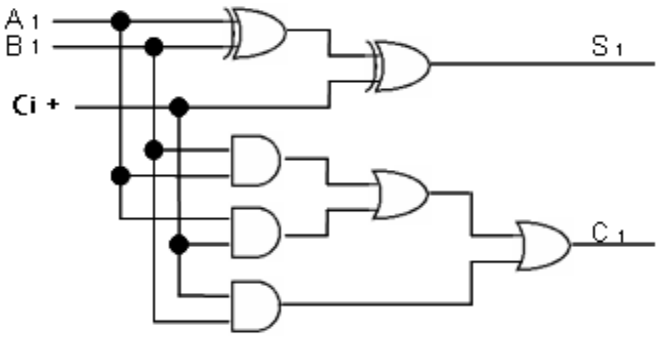
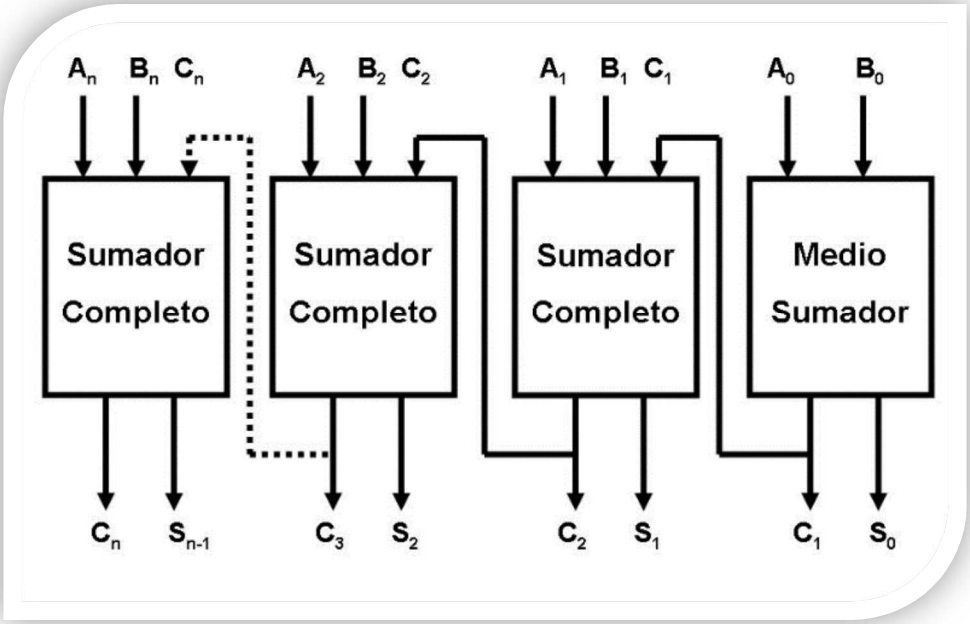


Diagrama lógico: Sumador completo

Sumador completo de n-bits

En algunos casos se desea sumar dos números de n-bits, lo que se hace es poner un medio sumador y n-1 sumadores completo en cascada y de esta manera tenemos un sumador de n bits, como se muestra en la figura Sumador de n-bits implementados con n-1 sumadores completos.



Sumador de n-bits implementados con n-1 sumadores completos

A partir del *diagrama a bloques del sumador de 4 bits* (ver figura Sumador de n-bits implementados con n-1 sumadores completos) se construye el diagrama lógico el cual se presenta en la figura Diagrama lógico de un Sumador de 4 bits en cascada y su respectivo diagrama eléctrico en la figura Diagrama eléctrico de un sumador de 4 bits en cascada.

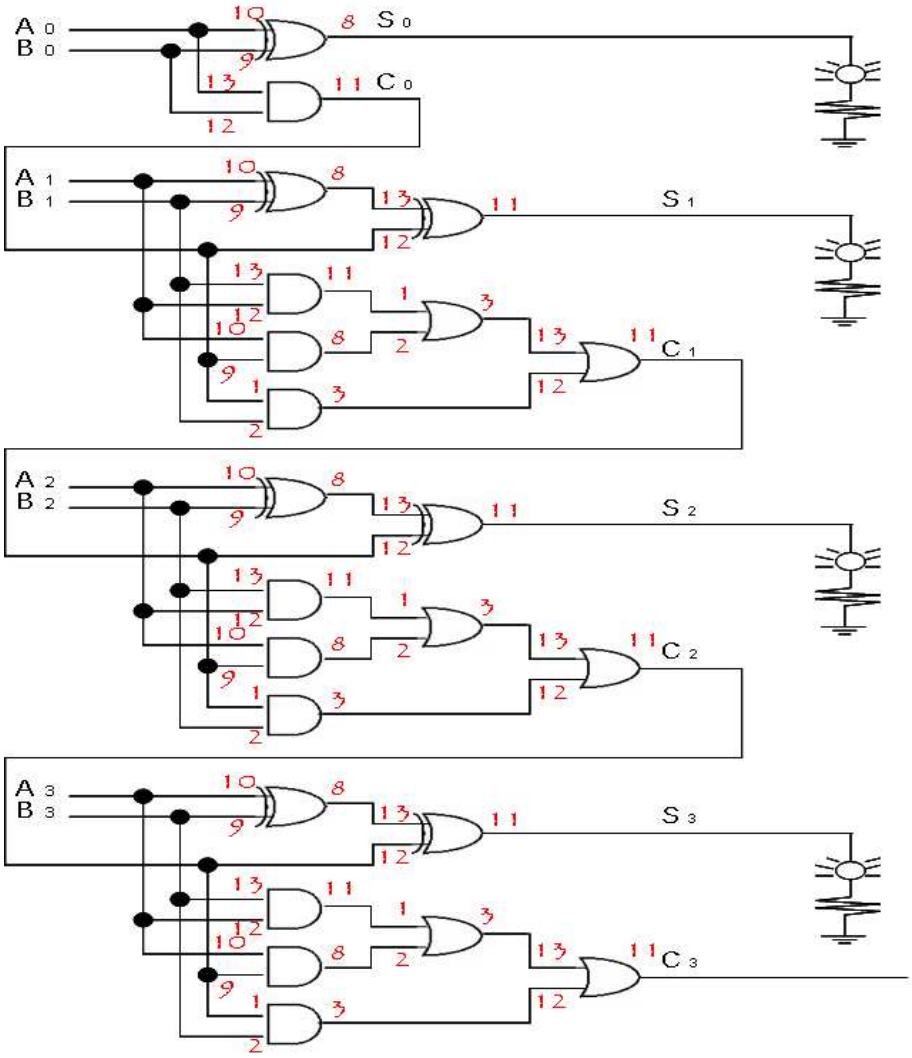


Diagrama eléctrico: Sumador de 4 bits

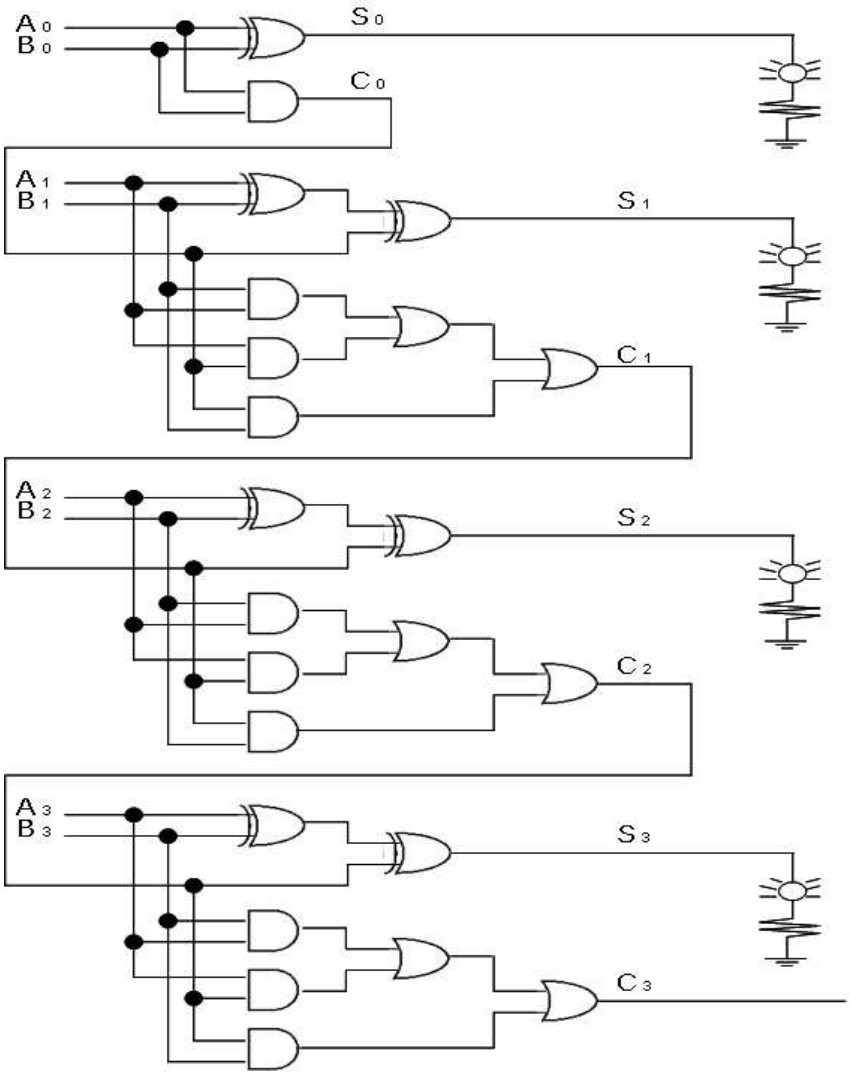


Diagrama lógico: Sumador de 4 bits

5.7. Restadores

El caso de los restadores en cuanto a su diseño mediante dispositivos digitales (compuertas binarias) es semejante al diseño de sumadores. De la misma manera como se crea un sumador de dos bits con acarreo, se puede generar un restador de dos bits con un bit llamado borrow. Para el caso de un restador binario de un digito o medio restador tenemos la siguiente tabla:

0 – 0 = 0
1 – 1 = 0
1 – 0 = 1
0 – 1 = 1

(En este caso se considera un “acarreo” o como lo conocíamos de nuestra aritmética básica “se toma prestado uno”) El concepto de *borrow* es semejante al de acarreo de la suma. Este caso de un restador bit a bit se llama semi restador y análogamente al caso de la suma solo se considera el acarreo como un dígito que indica que el signo del resultado es negativo o positivo.

Tenemos por lo tanto la siguiente tabla para la resta binaria de un dígito:

Minuendo	Sustraendo	Resultado	*Borrow*
0	**0**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**

ERROR: ioerror
OFFENDING COMMAND: image

STACK: