ELECTRÓNICA DIGITAL

CARRERA: ANÁLISIS DE SISTEMAS

SEMESTRE: TERCERO

Introducción a la asignatura

La asignatura Electrónica Digital es una asignatura teórico-práctica perteneciente al pensum de la carrera técnica Análisis de Sistemas, la cual proporciona al estudiante los conocimientos básicos y aspectos fundamentales de esta rama de la electrónica, la cual se encarga del estudio, diseño e implementación de los circuitos electrónicos que procesan la información en formato digital. Esta característica la hace especialmente adecuada para el procesamiento de información de forma precisa, confiable y a mayor velocidad.

Los Circuitos Digitales tienen una importancia fundamental en los sistemas informáticos, ya que permiten almacenar, recuperar, modificar y transmitir la información de manera codificada en dos únicos estados (código binario). La Electrónica Digital es, por tanto, la base de los Sistemas Computacionales, ya que se basa en el estudio de sus aspectos básicos lógico-matemáticos como el Álgebra Booleana, y circuitales como las Compuertas Lógicas, así como de los Circuitos Secuenciales que forman parte esencial de los computadores, y el Microprocesador como unidad de procesamiento principal de un ordenador.

OMAR RODRIGUES

- Ingeniero Electricista - Profesor Instructor -

Una publicación de

Instituto Universitario de Tecnología para la Informática

Programa

UNIDAD I

NOCIONES DE ÁLGEBRA BOOLEANA

Operaciones Básicas y Auxiliares del Álgebra Booleana

Tabla de la verdad

Representación de Funciones Booleanas

Álgebra Booleana

Tabla de la Verdad

Mapa de Karnaugh

Compuertas Lógicas

UNIDAD II

CIRCUITOS LÓGICOS

Circuitos Secuenciales

Concepto de Biestable

Flip-flop’s

Tipos de flip-flop’s RS, JK

Registros

Tipos de Memorias Semiconductoras

UNIDAD III

ARQUITECTURA DEL COMPUTADOR

Microprocesadores

- Registros
- Unidad de Control
- Unidad Lógico-Aritmética
- Buses
- Memorias
- Unidad de Entrada/Salida

Representación Interna de las Instrucciones y los Datos

- Caracteres
- Números Enteros
- Números Reales

INDICE

UNIDAD I

NOCIONES DE ÁLGEBRA BOOLEANA

La rama de la matemática que utiliza símbolos (letras y signos) y números para representar cantidades y operaciones en forma general, se conoce como *álgebra*. El *álgebra de Boole* o *álgebra booleana* fue creada por George Boole (1815-1864) como un sistema de cálculo lógico en el que las proposiciones lógicas (expresiones lingüísticas del razonamiento) se reducen a símbolos sobre los que puede operarse matemáticamente. En informática y electrónica digital se utiliza el álgebra booleana para esquematizar y analizar proposiciones lógicas a través de operaciones lógicas.

El presente capítulo abarca los aspectos del álgebra booleana que permiten analizar matemáticamente diferentes tipos de problemas y requerimientos en el área de la electrónica digital, planteados a través del razonamiento lógico. Con este propósito, se estudian los operadores básicos y auxiliares del álgebra booleana, las operaciones que se derivan de la combinación de dichos operadores con variables booleanas, las funciones booleanas que resultan de estas operaciones booleanas, y las formas de representar las funciones booleanas para facilitar su estudio y análisis, con la finalidad de obtener soluciones óptimas a los diferentes problemas y requerimientos.

Operaciones Básicas y Auxiliares del Álgebra Booleana

El *álgebra booleana* es una estructura algebraica axiomática basada en la teoría de conjuntos para manejar ecuaciones lógicas matemáticas. Podemos decir entonces que es una herramienta para representar

proposiciones lógicas en forma algebraica que presenta las siguientes características:

- Está definida a partir de unos elementos o variables binarias que se relacionan mediante operaciones lógicas reglamentadas por una serie de postulados y teoremas.
- Permite crear funciones lógicas que relacionan una variable binaria de salida con una o más variables binarias de entrada
- Cualquier expresión o variable solo puede interpretarse con uno de los dos valores de significado posibles representados por el número “1”, equivalente al valor semántico “verdadero”, y el número “0” equivalente al valor semántico “falso”.
- Las relación entre las variables binarias se basa en tres operaciones básicas denominadas *suma* (**+**) o *disyunción* (**∨**), *producto* (**·**) o *conjunción* (**∧**) y *complemento* (¯) o *negación* (**¬**). En teoría de conjuntos, estas operaciones corresponden respectivamente a *unión*, *intersección* y *complemento*.
- Las variables binarias se conectan o agrupan mediante símbolos lógicos denominados *operadores booleanos* para formar una función lógica.
- Las funciones booleanas cumplen con las propiedades *conmutativa*, *distributiva*, *asociativa*, *identidad*, *complemento* y *elemento neutro*.

Operadores Booleanos

Un *operador booleano* en un símbolo que indica una operación lógica entre dos o más números booleanos o variables booleanas. Los operadores booleanos básicos, *suma*, *producto* y *complemento*, y su notación y simbología según el campo de la lógica utilizado, se muestran en el cuadro 1.

Cuadro 1
Operadores booleanos básicos

Operador Booleano Básico	Símbolo en lógica			Función
	Booleana	Proposicional	Computacional	
Suma Lógica	+	∨ (Disyunción)	OR	Unión
Producto Lógico	·	∧ (Conjunción)	AND	Intersección
Complemento Lógico	$\bar{}$	¬ (Negación)	NOT	Complemento

La combinación de los operadores booleanos básicos da lugar a nuevos operadores booleanos denominados *operadores booleanos auxiliares*, los cuales se muestran en el cuadro 2 con su simbología según el campo de la lógica respectiva.

Cuadro 2
Operadores booleanos auxiliares

Operador Lógico Auxiliar	Símbolo en lógica		
	Booleana	Proposicional	Computacional
Suma Lógica Negada	$+, \bar{}$	↓	NOR
Producto Lógico Negado	$\cdot, \bar{}$	↑	NAND
Suma Lógica Exclusiva	⊕	⊕	XOR
Suma Lógica Exclusiva Negada	$\oplus, \bar{}$	$\oplus, \bar{}$	XNOR

Operaciones Booleanas

Las variables binarias se conectan a través de los operadores booleanos básicos y auxiliares para formar las expresiones o funciones booleanas denominadas *operaciones booleanas* básicas y auxiliares. En el cuadro 3 se muestran las operaciones booleanas básicas y auxiliares correspondientes a cada operador booleano, así como su simbología.

Cuadro 3
Operaciones booleanas básicas y auxiliares

Variables Booleanas	Operador Booleano	Operación Booleana	Símbolo
Variables de entrada: **A** y **B** Variable de salida: **F**	OR	$F = A + B$	A *OR* B
	AND	$F = A \cdot B$	A *AND* B
	NOT	$F = \overline{A}$, $F = \overline{B}$	*NOT* A , *NOT* B
	NOR	$F = \overline{A + B}$	A *NOR* B
	NAND	$F = \overline{A \cdot B}$	A *NAND* B
	XOR	$F = A \oplus B$	A *XOR* B
	XNOR	$F = \overline{A \oplus B}$	A *XNOR* B

Operaciones Booleanas con Números Binarios

Cada una de las operaciones booleanas vistas anteriormente, aplicadas a números binarios, da lugar a una serie de reglas básicas. Como puede verse en la tabla 3, cada operación booleana es una expresión o función booleana formada por una variable ***F*** (variable de salida), una o dos variables ***A*** y ***B*** (variables de entrada) y uno o dos operadores booleanos (suma, producto y complemento). Las variables de entrada se relacionan entre sí mediante los operadores booleanos, y pueden tomar solo uno de los dos valores binarios, cero (***0***) o uno (***1***). La variable de salida toma el valor binario cero (***0***) o uno (***1***) resultante de la operación booleana definida por el o los operadores booleanos presentes en la función.

Las reglas básicas de cada operación booleana con números binarios se desprenden de aplicar dicha operación a cada una de las combinaciones de valores binarios que resultan de acuerdo al número de variables de entrada. En las tablas 1 a 7 se presentan dichas reglas para cada operador booleano básico y auxiliar.

Tabla 1

Operador OR: F=A+B

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 2

Operador AND: F=A·B

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 3

Operador NOT: $F=\overline{A}$

A	F
0	1
1	0

Tabla 4

Operador NOR: $F=\overline{A+B}$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Tabla 5

Operador XOR: $F=A \oplus B$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 6

Operador XNOR: $F=\overline{A \oplus B}$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Tabla 7

Operador NAND: $F=\overline{A \cdot B}$

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

Tabla de la Verdad

Una *tabla de la verdad* es una estrategia lógica gráfica que permite establecer la validez de una proposición lógica compuesta, determinando las condiciones necesarias para que sea verdadero o falso un enunciado, mediante una tabla que muestra el valor de dicha proposición lógica compuesta, para cada combinación de valores que se pueda asignar a las proposiciones simples que la componen.

Fundamentalmente, una *tabla de la verdad* es un dispositivo gráfico para demostrar ciertas propiedades lógicas y semánticas de enunciados del lenguaje natural o de fórmulas del lenguaje del cálculo proposicional.

Estas tablas pueden construirse haciendo una interpretación de los signos lógicos como: ***no***, ***o***, ***y***, ***si…entonces***, ***sí y sólo si***, etc. La

interpretación corresponde al sentido que estas operaciones tienen dentro del razonamiento.

Puede establecerse una correspondencia entre los resultados de estas tablas y la deducción lógico-matemática. En consecuencia, las tablas de la verdad constituyen un método de decisión para chequear si una proposición es o no un teorema. Para la construcción de la tabla se asignará el valor ***1*** (uno) a una proposición cierta y ***0*** (cero) a una proposición falsa.

- En el siguiente video se muestra como construir las combinaciones de la tabla de la verdad: https://youtu.be/TTpCRffVyJc

La definición de *tabla de la verdad* nos lleva a otro concepto importante, como es el concepto de *proposición lógica.*

Proposición Lógica

Una *proposición lógica* puede entenderse como un producto lógico del pensamiento, expresado mediante un lenguaje natural que puede ser interpretado. Para el filósofo griego Aristóteles (384 a.C. a 322 a.C.), uno de los padres de la filosofía occidental, la *proposición* es un discurso enunciativo perfecto que se expresa en un juicio que significa lo verdadero y lo falso. Entonces, una proposición lógica expresa un contenido semántico al que es posible asignarle un valor, "verdadero" o "falso“, pero no ambos a la vez.

La lógica proposicional estudia las proposiciones simples, las relaciones entre ellas y los métodos de vincularlas mediante conectores lógicos, formando así proposiciones compuestas.

Proposición Lógica Simple: La *proposición lógica simple* constituye la forma elemental de la lógica. Es todo enunciado que tiene un valor de verdad: verdadero (***V***) o falso (***F***), pero no ambos a la vez. Es una afirmación con sentido, que da información de un acontecimiento que puede tener solo un valor de veracidad: verdadero o falso.

Ejemplo 1:

Las siguientes afirmaciones son proposiciones simples:

- El universo es infinito.
- Hoy va a llover
- 9 no es un número par.

Proposición Lógica Compuesta: Las *proposiciones compuestas*, también conocidas como *proposiciones complejas*, están formadas por más de una proposición simple, las cuales están unidas a través de conectivos lógicos, como: *no*, *y*, *e*, *si*, *o*, *entonces*, etc.

Ejemplo 2:

Las siguientes afirmaciones son proposiciones compuestas:

- Si el piano *no* es un instrumento de cuerda *o* viento, *entonces* es de percusión.
- La tierra es plana *si* no es redonda.
- El ave nacional, el turpial, es amarillo *y* negro.

Tabla De La Verdad de Una Proposición Lógica

La *tabla de la verdad* de una *proposición lógica compuesta*. se obtiene mediante el razonamiento lógico del enunciado, siguiendo el siguiente procedimiento:

1. Identificar las proposiciones simples contenidas en el enunciado o que forman la proposición compuesta.
2. Asignarle una variable a cada proposición simple, las cuales serán verdaderas o falsas de acuerdo a las condiciones que puedan presentarse.
3. Dibujar la tabla de la verdad con todas las combinaciones posibles de valores verdaderos (***1***) y falsos (***0***) de las variables.
4. Obtener el valor verdadero (***1***) o falso (***0***) de la proposición compuesta (variable de salida) para cada combinación de valores verdadero (***1***) o falso (***0***) de las proposiciones simples (variables de entrada), de acuerdo a las condiciones dadas en la propuesta.

***Ejemplo 3*:**

Obtener la tabla de la verdad de la siguiente proposición lógica:

*"Para aprobar una asignatura, un estudiante debe presentar tres exámenes, y debe obtener una calificación mayor o igual a **15** puntos en dos de ellos"*

1. Se definen las proposiciones simples:
   - Aprobación de cada examen (tres exámenes) con 15 o más puntos
2. Se asigna una variable a cada proposición simple (variables de entrada):
   - Variable ***A***: aprobación del primer examen con ***15*** o más puntos
   - Variable ***B***: aprobación del segundo examen con ***15*** o más puntos
   - Variable ***C***: aprobación del tercer examen con ***15*** o más puntos

3. Se dibuja una tabla con todas las combinaciones posibles de valores binarios, de acuerdo a la cantidad de variables de entrada asignadas a las proposiciones simples.
4. La condición de la propuesta indica que si dos cualesquiera de las tres proposiciones simples son verdaderas (***1***), sin importar el valor de la tercera (puede ser ***0*** o ***1***), entonces la proposición compuesta (variable de salida) es verdadera (***1***), de lo contrario es falsa (***0***). Por tanto, la tabla de la verdad de la proposición lógica queda como se muestra a continuación:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Representación de Funciones Booleanas

A continuación estudiaremos lo que es una *función booleana* y sus diferentes formas de representación, tanto analítica como gráficamente. En su forma analítica, una función booleana se puede representar mediante *álgebra booleana*, y en su forma gráfica, mediante *tabla de verdad*, *mapa de Karnaugh* y *compuertas lógicas*.

Función Booleana

Una *función booleana* es una sucesión de símbolos correspondientes a variables booleanas, relacionadas mediante operadores booleanos, que da

lugar a uno de dos posibles valores (***0*** ó ***1***) conocidos como valores booleanos o constantes booleanas. Esta forma de representación mediante símbolos correspondientes a variables booleanas y operadores booleanos es una forma de *representación algebraica de la función*.

En análisis matemático, el concepto general de función se refiere a una regla que asigna a cada elemento de un primer conjunto un único elemento de un segundo conjunto, es decir, las funciones son relaciones entre los elementos de dos conjuntos.

Para ***n*** variables booleanas (x_1, x_2, x_3,…, x_n), podemos definir una función booleana ***f*** de ***n*** variables como: $f_{(x)}: B^n \rightarrow B$, donde:

B^n**:** Es el conjunto de partida (dominio de la función) de todas las posibles combinaciones de valores binarios (***0*** y ***1***) que pueden tomar las variables.

B: Es el conjunto de llegada (codominio de la función) de los dos únicos valores binarios (***0*** ó ***1***) que puede tomar la función.

***Ejemplo 4*:**

A continuación veamos un ejemplo de función booleana ***F:***

F = A + B

Número de variables booleanas: ***n = 2***

Variables booleanas: (***A, B***)

Dominio de la función: $B^n = \{(0, 0); (0, 1); (1, 0); (1, 1)\}$

Codominio de la función: ***B = {0, 1}***

Una función booleana presenta las siguientes características:

1. Está formada por variables booleanas (***A***, ***B***, ***C***,***…***), (***W***, ***X***, ***Y***,***…***), etc.

2. Las variables se relacionan mediante los operadores booleanos ***OR*** (**+**), ***AND*** (·), ***NOT*** (¯) y sus operadores auxiliares ***NOR***, ***NAND***, ***XOR*** y ***XNOR***..
3. Obedece a las prpiedades del álgebra booleana, como son: conmutativa, asociativa, distributiva, identidad, complemento y elemento neutro..
4. Puede evaluarse directamente para diferentes combinaciones de valores binarios de las variables.

Ejemplo 5:

Las siguientes expresiones algebraicas corresponden a funciones booleanas:

$F_1 = \overline{A+B} + A \cdot B$

$F_2 = \bar{X} \cdot Y + X \cdot \bar{Z} + \overline{Y \cdot Z}$

Tabla de la Verdad de una Función Booleana

La tabla de la verdad de una función booleana es una representación gráfica de dicha función, o si se quiere, es la representación aritmética de la función, ya que presenta la lista completa de valores numéricos (binarios) que puede tomar la función (variable de salida) al ser evaluada para todos los posibles valores binarios que pueden tomar las variables de entrada.

El valor de una función booleana (variable de salida) se puede calcular a partir de los valores de las expresiones más simples (variables de entrada) que la componen y de los operadores booleanos que relacionan las variables de entrada, mediante un algoritmo llamado *tabla de la verdad*.

La *tabla de la verdad de una función booleana* se obtiene evaluando dicha función para todas las diferentes combinaciones de valores binarios de las variables de entrada y presenta las siguientes características:

1. Es una representación gráfica de todos los posibles casos que se pueden presentar en una función lógica y sus resultados.
2. Despliega una lista de todas las combinaciones de valores posibles de niveles lógicos cero (***0***) y uno (***1***) que puedan tomar las entradas, y sus correspondientes niveles lógicos resultantes para la salida.
3. El número de combinaciones de valores de la entrada es igual a $\mathbf{2^n}$ para una función de ***n*** variables de entrada.

❖ Pasos para construir la tabla de la verdad de una función booleana: https://youtu.be/CjX8q8lXWdg

Ejemplo 6:

Calcular la tabla de la verdad para la función booleana F_1 *del ejemplo 5*

$F_1 = \overline{A+B} + A \cdot B$

Variables de entrada: ***A, B***

Variable de salida: F_1

Número de variables de entrada: $n = 2$

Número de combinaciones: $2^n = 2^2 = 4$

Tabla de la verdad:

A	B	F_2	
0	0	1	$\longleftarrow F_2 = \overline{0+0} + 0 \cdot 0 = \overline{0} + 0 = 1 + 0 = 1$
0	1	0	$\longleftarrow F_2 = \overline{0+1} + 0 \cdot 1 = \overline{1} + 0 = 0 + 0 = 0$
1	0	0	$\longleftarrow F_2 = \overline{1+0} + 1 \cdot 0 = \overline{1} + 0 = 0 + 0 = 0$
1	1	1	$\longleftarrow F_2 = \overline{1+1} + 1 \cdot 1 = \overline{1} + 1 = 0 + 1 = 1$

Note que en el ejemplo 6 se calculó cada valor de salida sustituyendo los niveles lógicos correspondientes a cada fila en la función booleana y realizando todas las operaciones presentes en la función. Este método, aunque válido, pudiera generar errores de cálculo, sobre todo en funciones muy complicadas, además de tener que escribir la secuencia de cálculos.

Para evitar escribir y realizar todas las operaciones, se pueden agregar columnas a la tabla, una por cada variable negada y una o más por cada término de la función (un solo término pudiera generar varias columnas de acuerdo a su complejidad), luego se calcula cada columna según las variables y el, o los, operadores presentes y, finalmente, se calcula la función realizando las operaciones booleanas entre los diferentes términos (columnas) de la función.

Ejemplo 7:

Calcular la tabla de la verdad de las funciones booleanas F_1 *y* F_2 *del ejemplo 5 agregando columnas a la tabla*

$F_1 = \overline{A+B} + A \cdot B$

Variables de entrada: ***A, B***

Variable de salida: F_1

Número de variables de entrada: $n = 2$

Número de combinaciones: $2^n = 2^2 = 4$

Tabla de la verdad:

A	B	A+B	$\overline{A+B}$	AB	F_1
0	0	0	1	0	1
0	1	1	0	0	0
1	0	1	0	0	0
1	1	1	0	1	1

$F_2 = \bar{X} \cdot Y + X \cdot \bar{Z} + \overline{Y \cdot Z}$

Variables de entrada: ***X, Y, Z***

Variale de salida: F_2

Número de variables de entrada: $n = 3$

Número de combinaciones: $2^n = 2^3 = 8$

Tabla de la verdad:

X	Y	Z	$\bar{X}$	$\bar{Z}$	$\bar{X}Y$	$X\bar{Z}$	$\overline{YZ}$	F_2
0	0	0	1	1	0	0	1	1
0	0	1	1	0	0	0	1	1
0	1	0	1	1	1	0	1	1
0	1	1	1	0	1	0	0	1
1	0	0	0	1	0	1	1	1
1	0	1	0	0	0	0	1	1
1	1	0	0	1	0	1	1	1
1	1	1	0	0	0	0	0	0

Forma Canónica de una Función Booleana

La *forma canónica de una función booleana* es la función genérica o estándar de dicha función que permite su estudio y análisis y facilita alguna operación particular. Está formada por una serie de términos que presentan características particulares propias del álgebra booleana.

Los términos de la función canónica están relacionados entre sí por los operadores booleanos ***OR*** o ***AND***, es decir, la función puede consistir de una suma de términos o de un producto de términos. Por otra parte, cada uno de estos términos están formados por variables booleanas relacionadas entre sí por los operadores ***OR*** y ***NOT***, o por los operadores ***AND*** y ***NOT***, lo que

significa que cada término puede consistir de una suma de variables directas y/o negadas, o de un producto de variables directas y/o negadas.

La forma canónica de una función booleana es la expresión algebraica booleana que se obtiene a partir de la tabla de la verdad mediante procedimientos analíticos o gráficos determinados. Más específicamente, se puede definir como:

Forma canónica de una función booleana: Es una función booleana formada por una suma de productos (suma de mintérminos) o por un producto de sumas (producto de maxtérminos), con la característica particular de que en cada mintérmino (si es una suma de productos) o en cada maxtérmino (si es un producto de sumas) aparecen todas las variables de entrada en su forma directa o negada.

Mintérmino: Es un producto de variables booleanas en el que cada variable aparece solo una vez en forma directa o negada, es decir, es un término de una función booleana en el que las variables se relacionan por medio de los operadores ***AND*** y ***NOT***. Por ejemplo: $\boldsymbol{A \cdot \bar{B} \cdot C}$.

Maxtérmino: Es una suma de variables booleanas en la que cada variable aparece solo una vez en forma directa o negada, es decir, es un término de una función booleana en el que las variables se relacionan por medio de los operadores ***OR*** y ***NOT***.. Por ejemplo: $\boldsymbol{\bar{A} + B + \bar{C}}$.

Toda función booleana se puede expresar en forma canónica como una *suma de mintérminos*, llamada *forma canónica disyuntiva* o *SOP* (suma de productos), o como un *producto de maxtérminos*, llamada *forma canónica conjuntiva* o *POS* (producto de sumas).

Forma Canónica Disyuntiva o SOP (Suma de Productos)

Dada la tabla de la verdad de una función booleana, la forma canónica disyuntiva se obtiene según las siguientes condiciones:

- Los mintérminos (***m_i***) corresponden a cada fila en la cual la salida toma el valor lógico uno (***1***).
- Cada mintérmino (***m_i***) es el producto de las variables de entrada, tomando cada variable en forma directa si la misma tiene el valor lógico uno (***1***), o en forma negada si la misma tiene el valor lógico cero (***0***), en la fila ***i***.
- La forma canónica disyuntiva de la función booleana es la suma de todos los mintérminos: $F = m_1 + m_2 + m_3 + \ldots + m_i$

Ejemplo 8:

Para la función booleana ***F_3***, *obtener la función canónica disyuntiva (SOP):*

Función Booleana *Tabla de la Verdad* *Mintérminos*

$F_3 = \bar{x}y + \bar{x}z + y\bar{z}$

x	y	z	F	
0	0	0	0	
0	0	1	1	$\longrightarrow m_1 = \bar{x}\bar{y}z$
0	1	0	0	
0	1	1	1	$\longrightarrow m_3 = \bar{x}yz$
1	0	0	1	$\longrightarrow m_4 = x\bar{y}\bar{z}$
1	0	1	0	
1	1	0	1	$\longrightarrow m_6 = xy\bar{z}$
1	1	1	1	$\longrightarrow m_7 = xyz$

Función canónica disyuntiva (SOP): $F_3 = \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}\bar{z} + xy\bar{z} + xyz$

Forma Canónica Conjuntiva o POS (Producto de Sumas)

Dada la tabla de la verdad de una función booleana, la forma canónica conjuntiva se obtiene según las siguientes condiciones:

- Los maxtérminos (M_i) corresponden a cada fila en la cual la salida toma el valor lógico cero (***0***).
- Cada maxtérmino (M_i) es la suma de las variables de entrada, tomando cada variable en forma directa si la misma toma el valor lógico cero (***0***), o en forma negada si la misma toma el valor lógico uno (***1***), en la fila ***i***.
- La forma canónica conjuntiva de la función booleana es el producto de todos los maxtérminos: $F = M_1 \cdot M_2 \cdot M_3 \cdot \ldots \cdot M_i$

Ejemplo 9:

Para la función booleana F_3 dada, obtener la tabla de la verdad y la función canónica disyuntiva o SOP (suma de productos):

Función Booñeana

$F_3 = \bar{x}y + \bar{x}z + y\bar{z}$

Tabla de la Verdad

x	y	z	F	*Maxtérminos*
0	0	0	0	$\longrightarrow M_0 = x+y+z$
0	0	1	1	
0	1	0	0	$\longrightarrow M_2 = x+\bar{y}+z$
0	1	1	1	
1	0	0	1	
1	0	1	0	$\longrightarrow M_5 = \bar{x}+y+\bar{z}$
1	1	0	1	
1	1	1	1	

Función canónica conjuntiva (POS): $F_3 = (x+y+z)(x+\bar{y}+z)(\bar{x}+y+\bar{z})$

- En el siguiente video se puede ver un ejemplo de cálculo de las funciones canónicas de una función booleana: https://youtu.be/kDmHdid0OYA

Mapa de Karnaugh

La forma canónica de una función lógica puede resultar extensa, con una gran cantidad de términos, lo cual complica el análisis y procesamiento de dicha expresión. Una herramienta gráfica que se usa para representar funciones booleanas de forma simplificada se conoce como *mapa de Karnaugh*, el cual es una representación bidimensional de la tabla de la verdad de la función a simplificar, ya que posee un cuadro por cada fila de la tabla de la verdad. El mapa de Karnaugh es un procedimiento alternativo a la simplificación de funciones booleanas mediante axiomas y teoremas del álgebra booleana, que aprovecha la capacidad del cerebro para trabajar mejor con patrones gráficos que con ecuaciones y expresiones analíticas.

Mapa de Karnaugh: Es un diagrama utilizado para la simplificación de funciones booleanas en forma canónica, permitiendo obtener la forma canónica mínima de una función booleana, es decir, la función canónica con el mínimo número de términos. Se puede utilizar para funciones de hasta seis variables, pero se usa fundamentalmente para simplificar funciones de hasta cuatro variables. Se conocido también como *Mapa-K*, Mapa-KV, *Tabla de Karnaugh* o *Diagrama de Veitch*.

Construcción del Mapa de Karnaugh de una Función Booleana

A continuación se presenta un procedimiento para la construcción del *mapa de Karnaugh de una función booleana*. Para facilitar la comprensión del

mismo se desarrollará un ejemplo (*Ejemplo 11*) con dos tablas de la verdad a medida que se vayan definiendo los pasos a seguir.

Dada la tabla de la verdad de una función booleana de ***n*** variables, los pasos para construir el mapa de Karnaugh son los siguientes:

1. Se dividen las ***n*** variables de entrada en dos grupos, en el mismo orden de la tabla de la verdad. El primer grupo será de n_1 variables y el segundo grupo será de n_2 variables.

Si ***n*** es par, n_1 será igual a n_2, es decir, $n_1 = n_2 = n/2$

Si ***n*** es impar, n_1 será menor que n_2 por uno, es decir, n_1 tendrá una variable menos que n_2, o lo que es lo mismo $n_1 = n_2 - 1$.

Veamos dos ejemplos para ilustrar la forma de crear los dos grupos de variables de una tabla de la verdad cualquiera:

Ejemplo 10:

Tabla (a)

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Variables de entrada: ***A***, ***B***, ***C***

n = 3 (impar)

$n_1 = 1$

$n_2 = 2$

Grupo 1: ***A***

Grupo 2: ***B***, ***C***

Tabla (b)

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Variables de entrada: ***A***, ***B***, ***C***, ***D***

n = 4 (par)

$n_1 = 2$

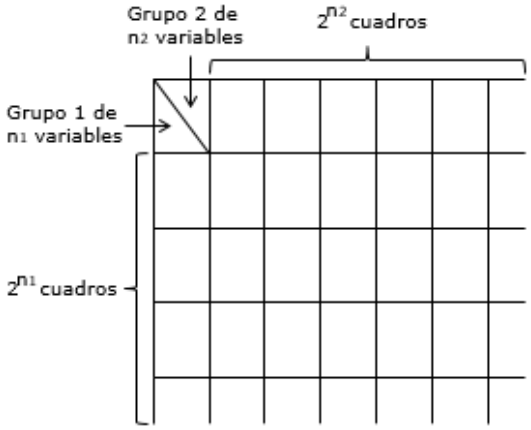
$n_2 = 2$

Grupo 1: ***A***, ***B***

Grupo 2: ***C***, ***D***

2. Se dibuja una tabla como la mostrada en la figura 1, y se escriben los grupos de variables, definidos en el paso anterior, en el cuadro de la esquina superior izquierda como se muestra a la derecha de la figura.

Figura 1
Forma general del gráfico del mapa de Karnaugh



Colocación de las variables en el cuadro de la esquina superior izquierda para las tablas (a) y (b) del punto anterior:

a) Tabla (a) variables: ***A***, ***B***, ***C***

b) Tabla (b) variables: ***A***, ***B***, ***C, D***

3. Se escriben en la primera columna (izquierda) todas las combinaciones de valores del primer grupo de variables, de arriba hacia abajo, y en la primera fila (superior) todas las combinaciones de valores del segundo grupo de variables, de izquierda a derecha, considerando en ambos casos que de un cuadro al siguiente solo puede cambiar de valor una sola variable. Para las tabla (a) y (b):

a) Tabla (a), variables: ***A***, ***B***, ***C***

BC / A	00	01	11	10
0				
1				

b) Tabla (b), variables: ***A***, ***B***, ***C***, ***D***

CD / AB	00	01	11	10
00				
01				
11				
10				

4. Se llenan los cuadros restantes con los valores de la variable de salida (***F***) correspondientes a cada combinación de valores de las variables de entrada en cada fila de la tabla de la verdad.

A continuación se muestra la ubicación de un valor en el gráfico, donde se indica la correspondencia del valor de ***F*** y su combinación para las tablas (a) y (b):

b) Variables de entrada: ***A***, ***B***, ***C***

BC / A	00	01	11	10
0				
1			0	

Valor de la variable de salida correspondiente a la combinación ***111*** de la tabla de la verdad

a) Variables de entrada: ***A***, ***B***, ***C***, ***D***

CD / AB	00	01	11	10
00				
01		1		
11				
10				

Valor de la variable de salida correspondiente a la combinación ***0101*** de la tabla de la verdad

Una vez rellenados todos los cuadros del gráfico para las tablas (a) y (b), el resultado es el siguiente:

b) Variables de entrada: ***A***, ***B***, ***C***

BC / A	00	01	11	10
0	1	0	0	1
1	1	1	0	0

a) Variables de entrada: ***A***, ***B***, ***C***, ***D***

CD / AB	00	01	11	10
00	0	1	0	0
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

5. Se agrupan los unos (***1's***) correspondientes a los valores de la variable de salida (no se incluyen los unos ubicados en la primera fila superior y en la primera columna de la izquierda). Los grupos de unos (1's) deben cumplir las siguientes condiciones:

- Los grupos deben ser lo más grandes posibles.
- Debe haber la menor cantidad de grupos posible.
- La cantidad de unos (***1's***) en un grupo debe ser igual a ***2***, con ***n = 0***, ***1***, ***2***, ***3***,***…***, es decir, los grupos pueden tener ***1***, ***2***, ***4***, ***8***, ***16***,***…*** unos (***1's***), y no puede haber grupos con ***3***, ***5***, ***6***, ***7***, ***9***,***…*** unos (***1's***).
- Se deben agrupar los unos (***1's***) adyacentes.
- Los unos (***1's***) de los extremos izquierdo y derecho de la misma fila son adyacentes, y los unos (***1's***) de los extremos superior e inferior de la misma columna son adyacentes.
- No debe quedar ningún uno (***1***) fuera de un grupo.
- Puede haber solapamiento de grupos.
- Los grupos pueden formarse en horizontal y/o vertical, nunca en diagonal.

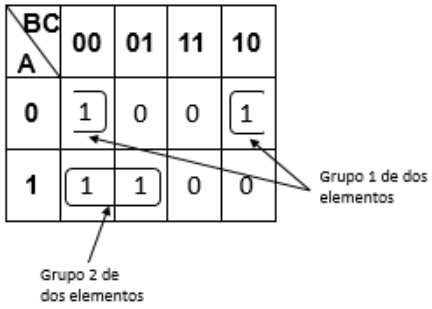
6. Para la agrupación de los unos (***1's***) no existe un procedimiento normalizado, lo importante es que se cumplan las condiciones del punto anterior para asegurar una agrupación óptima y por tanto una expresión mínima de la función booleana, es decir, una expresión con el mínimo número de términos. Sin embargo, se sugiere el siguiente procedimiento:

- Localizar los unos (***1's***) aislados, es decir, que no sean adyacentes a otros unos (***1's***), y encerrarlos en un círculo. Cada uno (***1***) aislado forma un grupo.
- Localizar los unos (***1's***) que sean adyacentes solamente a otro uno (***1***) y agruparlos. Cada par de unos (***1's***) adyacentes forman un grupo.

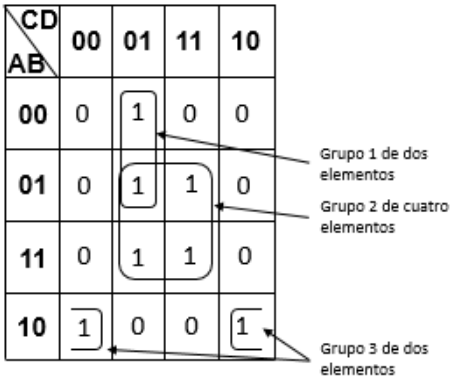
- Agrupar cualquier octeto de unos (***1's***) adyacentes, incluso si contiene unos que ya han sido agrupados.
- Agrupar cualquier cuarteto de unos adyacentes que contengan uno o más términos que no hayan sido agrupados.
- Agrupar cualquier par de unos (***1's***) para incluir los que hasta ahora no hayan sido agrupados.

Una vez agrupados los unos (***1's***), se completa el mapa de Karnaugh de la función booleana. A continuación se presentan los mapas de Karnaugh de las tablas (a) y (b):

a) Tabla (a), variables: ***A***, ***B***, ***C***



b) Tabla (b), variables: ***A***, ***B***, ***C***, ***D***



El mapa de Karnaugh también puede construirse a partir de una función canónica, es decir, si se tiene la forma canónica de una función booleana se puede obtener el mapa de Karnaugh directamente de esta, sin necesidad de obtener la tabla de la verdad.

❖ En el siguiente video se muestra cómo obtener el mapa de Karnaugh a partir de la función canónica: https://youtu.be/ZiJRBgym_es

Forma Canónica Simplificada o Mínima de una Función Booleana

Como sabemos, el mapa de Karnaugh es una herramienta gráfica ideada para representar funciones booleanas en forma simplificada, a fin de facilitar el análisis y procesamiento de funciones canónicas que poseen un gran número de términos. El mapa de Karnaugh permite reducir la cantidad de términos, así como también, simplificar dichos términos.

La función booleana obtenida a partir del mapa de Karnaugh es la *expresión o función booleana simplificada o mínima*, la cual se puede definir operativamente como "la suma lógica (unión) de los productos lógicos (intersecciones) de las variables de entrada que mantienen su valor en cero (***0***) o en uno (***1***) en todos los elementos de cada grupo del mapa de Karnaugh. En cada producto lógico se coloca la variable en forma directa si esta mantiene su valor en uno (***1***), o negada si esta mantiene su valor en cero (***0***)".

La forma simplificada de una función se puede obtener directamente analizando el mapa de Karnaugh y aplicando la definición operativa dada anteriormente, sin embargo. Sin embargo, una forma gráfica que permite obtener la función booleana simplificada de forma sistematizada a través de una tabla sencilla y ordenada, minimizando los posibles errores, se presenta a continuación. Para mayor comprensión, se utilizarán las tablas (a) y (b) del ejemplo 11 que se ha venido desarrollando del punto anterior para ilustrar el procedimiento:

1. Se escriben en forma horizontal las variables de entrada en el mismo orden de la tabla de la verdad y se traza una línea horizontal debajo de ellas:

a) Tabla (a)

A	B	C

b) Tabla (b)

A	B	C	D

2. Se escriben, en forma vertical, la combinaciones de ceros y unos correspondientes a cada elemento (***1***) de un grupo, colocando debajo de cada variable el valor que le corresponde en el mapa de Karnaugh a ese elemento del grupo, y se coloca una línea horizontal de separación debajo:

a) Tabla (a)

A	B	C
0	0	0
0	1	0

b) Tabla (b)

A	B	C	D
0	0	0	1
0	1	0	1

3. Se repite el paso 2 por cada grupo. Se puede iniciar con cualquier grupo, sin un orden específico, y colocar las combinaciones de los elementos del grupo, también sin ningún orden específico:

a) Tabla (a)

A	B	C	
0	0	0	Grupo 1
0	1	0	
1	0	0	Grupo 2
1	0	1	

b) Tabla (b)

A	B	C	D	
0	0	0	1	Grupo 1
0	1	0	1	
0	1	0	1	Grupo 2
0	1	1	1	
1	1	0	1	
1	1	1	1	
1	0	0	0	Grupo 3
1	0	1	0	

4. Por cada grupo de combinaciones se observa cada columna del grupo, y se verifica si se mantiene o no el valor en uno (***1***) o en cero (***0***) a lo largo de la columna. Se tachan las columnas en las cuales la variable cambia su valor de cero (0) a uno (1) o viceversa:

a) Tabla (a)

A	B	C
0	~~0~~	0
0	~~1~~	0
1	0	~~0~~
1	0	~~1~~

b) Tabla (b)

A	B	C	D
0	~~0~~	0	1
0	~~1~~	0	1
~~0~~	1	~~0~~	1
~~0~~	1	~~1~~	1
~~1~~	1	~~0~~	1
~~1~~	1	~~1~~	1
1	0	~~0~~	0
1	0	~~1~~	0

5. Para cada grupo, se multiplican las variables cuya columna no fue tachada, colocando la variable en forma directa si su columna está formada por unos (***1's***), o en forma negada si su columna está formada por ceros (***0's***):

a) Tabla (a)

A	B	C	
0	~~0~~	0	$\bar{A}\bar{B}$
0	~~1~~	0	
1	0	~~0~~	$A\bar{B}$
1	0	~~1~~	

b) Tabla (b)

A	B	C	D	
0	~~0~~	0	1	$\bar{A}\bar{C}D$
0	~~1~~	0	1	
~~0~~	1	~~0~~	1	
~~0~~	1	~~1~~	1	BD
~~1~~	1	~~0~~	1	
~~1~~	1	~~1~~	1	
1	0	~~0~~	0	$A\bar{B}\bar{D}$
1	0	~~1~~	0	

6. La suma lógica de los productos lógicos obtenidos, forma la expresión booleana mínima o simplificada de la función booleana:

a) Tabla (a)

$$F = \bar{A}\bar{B} + A\bar{B}$$

b) Tabla (b)

$$F = \bar{A}\bar{C}D + BD + A\bar{B}\bar{D}$$

- En el siguiente sitio Web se muestran dos ejemplos de simplificación de funciones booleanas mediante mapa de Karnaugh: https://unicrom.com/mapas-de-karnaugh-simplificacion-de-funciones/

Operadores Lógicos Gráficos (Compuertas Lógicas)

Los operadores lógicos booleanos se representan gráficamente mediante componentes lógicos conocidos como compuertas o puertas lógicas.

Compuerta Lógica: Es una representación gráfica de los operadores lógicos booleanos básicos y auxiliares, la cual produce un nivel lógico de salida relacionando una o más variables (niveles lógicos) de entrada mediante las operaciones booleanas. Las compuertas lógicas a su vez se representan a través de símbolos específicos para cada tipo de compuerta. Físicamente, las compuertas lógicas son circuitos electrónicos de uno más terminales de entrada y uno o más terminales de salida, conformados internamente por transistores, y que tienen como principal característica que generan una señal por el terminal de salida de acuerdo a una operación booleana entre las señales que son aplicadas por los terminales de entradas de la compuerta.

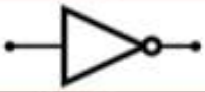
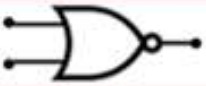
En el cuadro 1 se presentan las compuertas lógicas (nombre, símbolo y operación lógica correspondiente).

Representación de Funciones Booleanas Mediante Compuertas Lógicas

La funciones booleanas pueden ser representadas gráficamente a través de *compuertas lógicas*, sustituyendo los operadores booleanos por sus equivalentes gráficos (compuertas lógicas), es decir, se dibujan las compuertas lógicas correspondientes a los operadores booleanos presentes

en la función booleana y se interconectan siguiendo la secuencia lógica de la función, formando así un circuito denominado *circuito lógico*. Las entradas del circuito son las variables de entrada de la función booleana, y la salida del circuito es la variable de salida de la función booleana.

Cuadro 1
Compuertas lógicas

NOMBRE	SÍMBOLO	OPERACIÓN BOOLEANA
OR		$F = A + B$
AND		$F = A \cdot B$
NOT		$F = \overline{A}$
NOR		$F = \overline{A + B}$
NAND		$F = \overline{A \cdot B}$
XOR		$F = A \oplus B$
XNOR		$F = \overline{A \oplus B}$

Circuito lógico

Un *circuito lógico* es cualquier circuito que se comporte de acuerdo a un conjunto de reglas lógicas. Se le conoce también como *diagrama lógico* o *logigrama*. En el ámbito de la electrónica digital, un *circuito lógico* es aquel que ejecuta funciones de procesamiento y control de información en código binario mediante operaciones lógicas.

No existen reglas para representar una función booleana mediante compuertas lógicas, sin embargo, se pueden seguir los siguientes pasos con la finalidad de construir un circuito lógico:

1. Se escribe las variables de entrada a la izquierda en forma vertical (ver ejemplo 11-a). También pueden escribirse en forma horizontal y se trazan líneas verticales hacia abajo desde cada variable (ver ejemplo 11-b).

2. Se relacionan las variables de entrada de cada término de la función booleana mediante compuertas lógicas de la siguiente manera:

2.1. Si el término está formado por dos variables (relacionadas por un operador), se dibuja la compuerta lógica que corresponda a dicho operador, y se trazan líneas horizontales desde cada variable hasta la compuerta lógica, intercalando una compuerta NOT si alguna variable aparece negada (ver ejemplos 11-a y 11-b)

2.2. Si el término está formado por tres variables (relacionadas por dos operadores) se realiza el paso anterior para las dos primeras variables, luego se dibuja la compuerta correspondiente al segundo operador y finalmente se trazan líneas horizontales desde la primera compuerta y la tercera variable hasta dicha compuerta (ver ejemplo 11-c).

2.3. Si el término está formado por más de tres variables, se repite el paso anterior tantas veces como sea necesario (ver ejemplo 11-d).

3. Una vez dibujadas todas las compuertas y conexiones correspondientes a cada término, se dibujan a la derecha las compuertas correspondientes a los operadores que relacionan a dichos términos (de dos en dos) y se

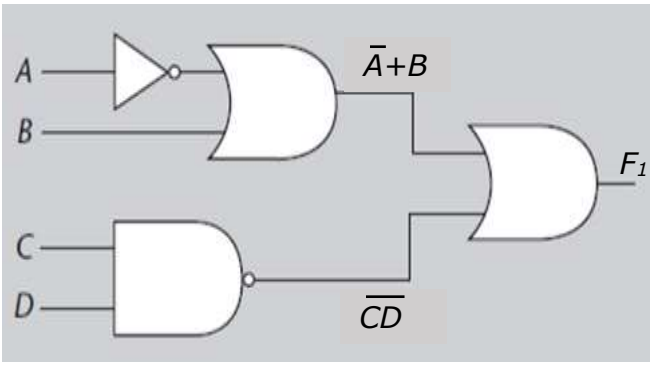
trazan las conexiones desde las salidas de las compuertas de cada término hasta las entradas de las compuertas que los relacionan (ver ejemplo 11).

4. La salida de la última compuerta será la salida del circuito (variable de salida).

Ejemplo 11:

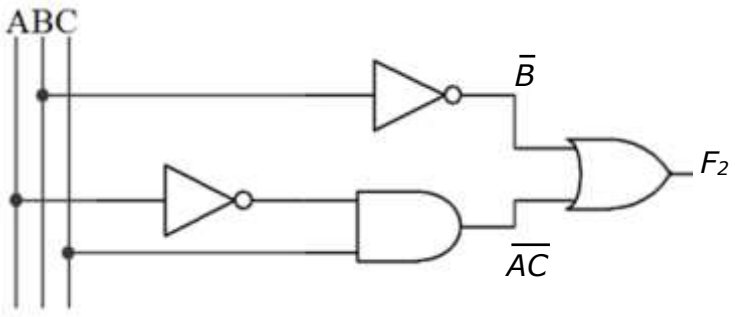
a. Representar por medio de compuertas lógicas la siguiente función booleana:

$F_1 = (\bar{A}+B) + \overline{CD}$



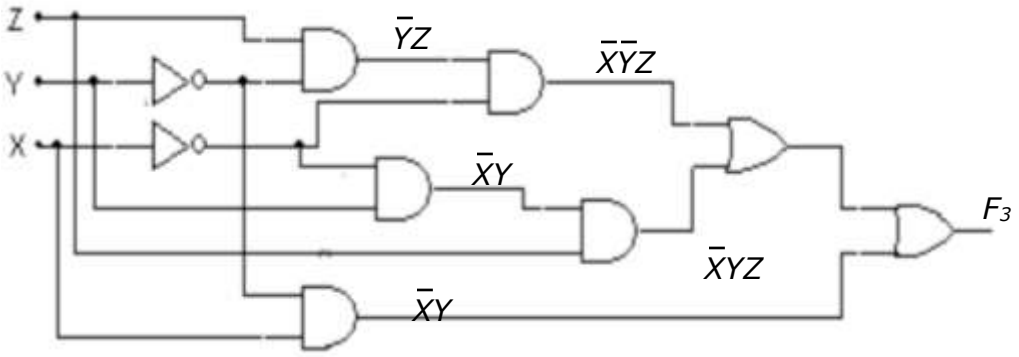
b. Representar por medio de compuertas lógicas la siguiente función booleana:

$F_2 = \bar{B} + \bar{A}C$



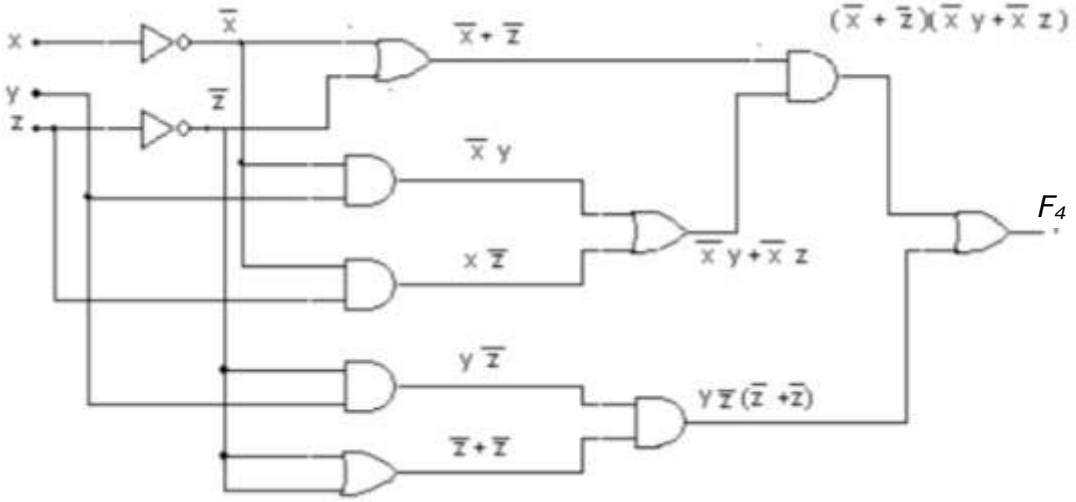
c. Realice el logigrama de la siguiente función booleana:

$F_3 = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}$



d. Construya el circuito lógico de la siguiente función booleana:

$F_3 = (\bar{X} + \bar{Z})(XY + XZ) + YZ(\bar{Z} + \bar{Z})$



En el siguiente video se explican varios ejemplos de circuitos lógicos a partir de la función booleana: https://youtu.be/Tca0yIOjXqs

AUTOEVALUACIÓN UNIDAD I

1. Obtener la tabla de la verdad de las siguiente función booleana:

   $F = (\overline{B+A})(\overline{A}+B)+(\overline{A}+\overline{B})(B+A)+(\overline{B}+A)$

2. Obtener las funciones canónicas disyuntiva (SOP) y conjuntiva (POS) de la siguiente función booleana:

   $F = [(\overline{A} \oplus \overline{B})+C][C+(\overline{C \oplus B})]+AB+CA$

3. Obtener el mapa de Karnaugh y la forma simplificada de la siguiente función booleana:

   $F = \overline{A}\overline{B}\overline{C}D+\overline{A}\overline{B}C+CD+A\overline{B}CD+A\overline{B}C\overline{D}$

4. Representar mediante compuertas lógicas la siguiente función booleana:
   $F = \overline{A}BC(\overline{C+D})+(A+\overline{B})\overline{BCD}$

5. Un circuito digital consta de cuatro entradas y una salida. La salida se activa (toma el valor lógico ***1***) solo cuando existen mayoría de entradas en nivel lógico alto (***1***):

   a) Construya la tabla de la verdad
   b) Obtenga las funciones canónicas disyuntiva (SOP) y conjuntiva (POS)
   c) Simplifique la función canónica mediante el mapa de Karnaugh
   d) Represente la función simplificada mediante compuertas lógicas (circuito lógico).

UNIDAD II

CIRCUITOS LÓGICOS

Todo circuito que funcione de acuerdo a las leyes de la lógica, manejando información en forma de señales eléctricas de dos niveles de tensión (alto y bajo), representados por los números binarios cero (***0***) y uno (***1***), es un *circuito lógico*. Los circuitos lógicos se clasifican en *circuitos combinacionales* y *secuenciales*, los cuales se diferencian en que los circuitos secuenciales utilizan elementos de memoria, mientras que los combinacionales no.

Los circuitos estudiados en el capítulo anterior son circuitos combinacionales, ya que no utilizan elementos de memoria, y la información de sus salidas en un instante depende únicamente de la combinación de las entradas en dicho instante. En este capítulo se estudiarán los circuitos conocidos como *biestables* que constituyen la base para la construcción de los circuitos secuenciales.

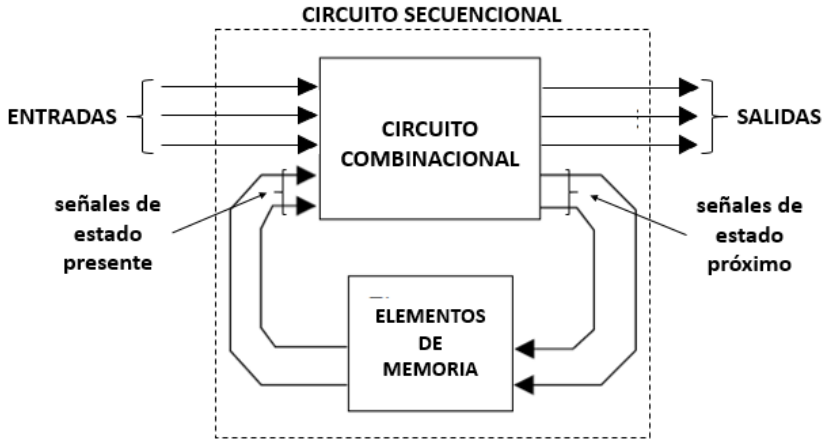
Circuitos Secuenciales

Un circuito secuencial es aquel en el cual las salidas, en cualquier instante, dependen de la combinación de las entradas en dicho instante y de los estados previos (historia pasada) de las entradas, es decir, de la secuencia de las entradas, por lo que necesitan elementos de memoria que recojan la información de la historia pasada del sistema.

Un circuito secuencial, como puede verse en la figura 2, está formado por un circuito de lógica combinacional y unos elementos de memoria que almacenan el valor de la entrada en un instante determinado por una señal de

control externa, y lo mantienen hasta que esta señal de control ordene el almacenamiento de un nuevo valor.

Figura 2
Diagrama de bloques de un circuito secuencial



Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. Fundamentos de los Circuitos Secuenciales

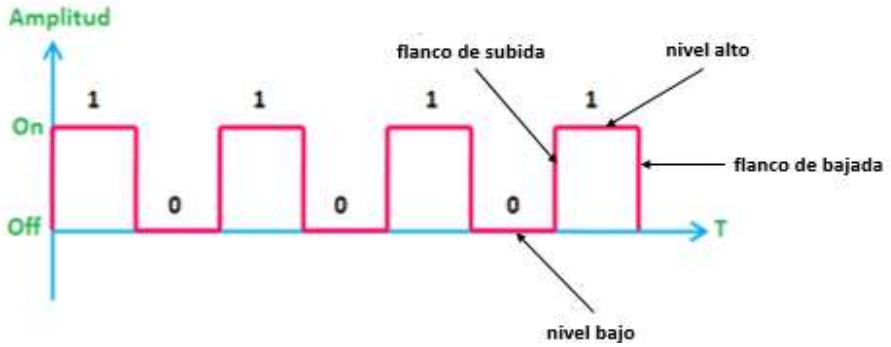
Los circuitos secuenciales se dividen en dos categorías que son los *circuitos secuenciales síncronos y asíncronos*.

Circuitos Secuenciales Asíncronos: Son aquellos en los cuales el tiempo en que los estado previos de las salidas son aplicados a las entradas del circuito combinacional, es decir, los cambios en los estados de las salidas de estado próximo a estado presente, depende exclusivamente de los retardos propios del circuito combinacional.

Circuitos Secuenciales Síncronos: Son aquellos en los cuales el tiempo en que los estado previos de las salidas son aplicados a las entradas del circuito combinacional, es decir, los cambios de estado próximo a estado presente de

las salidas, depende exclusivamente de una señal externa al sistema conocida como señal de reloj. Esta señal de reloj controlará el comportamiento de los elementos de memoria. La forma de onda de la señal de reloj se puede observar en la figura 3.

Figura 3
Forma de onda de la señal de reloj



Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. Fundamentos de los Circuitos Secuenciales

Concepto de Biestable

Los *circuitos biestables* son circuitos lógicos elementales capaces de permanecer indefinidamente en uno de dos estados posibles (alto o bajo) mientras no haya orden de cambio de estado, lo que se conoce como *orden de disparo*. En otras palabras, tienen la capacidad de retener el estado (almacenar un uno o un cero), lo que los convierte en la célula elemental de memoria, y por tanto, constituyen la base para la construcción de circuitos secuenciales.

Los circuitos biestables asíncronos y los síncronos activados por nivel se denominan *latches*, mientras que los circuitos biestables síncronos activados por flanco se denominan *flip-flop's*, ambos tipos de biestables pueden utilizarse para almacenar un bit de memoria, con la diferencia principal

entre ambos tipos de dispositivos en el método empleado para cambiar de estado. En este capítulo se estudiaran los flip-flop´s como elementos básicos de memoria.

Flip-Flop's

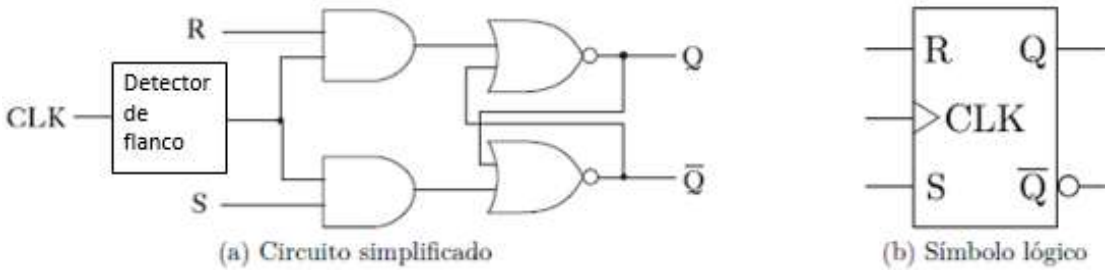
Un *flip-flop*.es un circuito lógico biestable síncrono, que constituye una celda básica de memoria capaz de guardar un bit de información. Como indica el término *biestable*, un flip-flop posee dos estados estables: uno (***1***) o activación y cero (***0***) o desactivación, y conmuta entre estos dos estados, en cada uno de los cuales se puede mantener indefinidamente. El término *síncrono* significa que la salida cambia de estado únicamente en un instante específico determinado por una entrada de disparo denominada ***CLK*** (reloj), la cual recibe el nombre de entrada de control (***C***). El cambio de estado se produce durante el flanco positivo (de subida) o negativo (de bajada) del impulso de reloj y es sensible a sus entradas solo durante esta transición de reloj.

Existen cuatro tipos de flip-flop's: SR, JK, T y D. En esta sección se estudiarán los flip-flop's tipo SR y JK.

Flip-flop SR

El *flip-flop SR* es un flip-flop que tiene dos entradas síncronas ***S*** (SET) y ***R*** (RESET), una entrada de reloj (***CLK***) y dos salidas complementarias ***Q*** y $\overline{\boldsymbol{Q}}$. Las entradas ***S*** y ***R*** se denominan entradas síncronas, dado que los datos de estas entradas se transfieren a las salidas solo con el flanco de disparo del impulso de reloj. En la figura 4 se muestra el circuito con compuertas ***NOR*** y el símbolo lógico de un flip-flop SR.

Figura 4
Circuito simplificado y símbolo lógico del flip-flop SR



https://www.cartagena99.com/recursos/alumnos/temarios/tema_3_-_sistemas_secuenciales.pdf

Funcionamiento del flip-flop SR: El nombre de la entrada ***S*** viene de la palabra SET, o grabar en memoria. El de la entrada ***R*** viene de RESET, o borrar la memoria. Cuando la entrada ***S*** tiene nivel lógico alto (***1***) y la entrada ***R*** un nivel lógico bajo (***0***), significa que el circuito tiene orden de grabar un bit. Si la entrada ***R*** tiene nivel lógico alto (***1***) y la entrada ***J*** un nivel lógico bajo (***0***), significa que el circuito tiene orden de borrar el bit. Por tanto, no pueden estar ambas entradas ***S*** y ***R*** en nivel lógico alto simultáneamente. La salida $\overline{\boldsymbol{Q}}$ se caracteriza por presentar siempre el estado opuesto al de la salida ***Q***, salvo que se haga un uso indebido aplicando simultáneamente orden de grabar y borrar. Esta situación debe ser evitada y se le llama *estado prohibido*.

En la Tabla 8, la cual se conoce como la *tabla de la verdad completa del flip-flop RS*, $\boldsymbol{Q_n}$ y $\boldsymbol{Q_{n+1}}$ representan siempre a la misma salida ***Q*** del circuito (a) de la Figura 4. Representan respectivamente el estado actual (***n***) y el estado futuro o estado final (***n+1***) que dicha salida adopta.

Tabla 8
Tabla de la verdad completa del flip-flop SR

S	R	Qn	Qn+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. Fundamentos de los Circuitos Secuenciales

Como se puede observar en la tabla 8, cuando ***R*** y ***S*** son ***0***, el estado futuro de la salida ***Q*** permanece igual a su estado actual ($Q_{n+1} = Q_n$), es decir, no graba ni borra. Cuando ***S*** es ***0*** y ***R*** es ***1***, el estado futuro de la salida ***Q*** es ***0*** independientemente de que el estado actual de ***Q*** sea ***0*** o ***1***, es decir, borra el bit. Cuando ***S*** es ***1*** y ***R*** es ***0***, el estado futuro de la salida ***Q*** es ***1*** independientemente de que el estado actual de ***Q*** sea ***0*** o ***1***, es decir, graba un bit. Finalmente, si ambas entradas son ***1***, el circuito no funciona correctamente ya que tiene orden de grabar y borrar simultáneamente, por lo que se señala con una ***x*** en la tabla de la verdad.

La tabla de la verdad completa del flip-flop SR (tabla 8) se puede simplificar y obtener la tabla de la verdad resumida del flip-flop SR, la cual se presenta en la tabla 9.

Tabla 9
Tabla de la verdad resumida del flip-flop SR

S	R	Qn+1	
0	0	Qn	Mantiene el valor (no graba, no borra)
0	1	0	Reset (borra)
1	0	1	Set (graba)
1	1	X	Estado prohibido

Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. Fundamentos de los Circuitos Secuenciales

Flip-flop JK

El *flip-flop JK* se comporta como el SR en las condiciones de operación SET, RESET y de permanencia de estado (no borra, no graba), pero resuelve el problema de tener una salida indeterminada cuando las entradas se encuentran activas a la vez. La entrada ***J*** es el equivalente a la entrada ***S*** y la entrada ***K***, el equivalente a la entrada ***R***. Cuando las dos entradas se colocan a nivel alto la salida cambia al estado opuesto al que se encontraba.

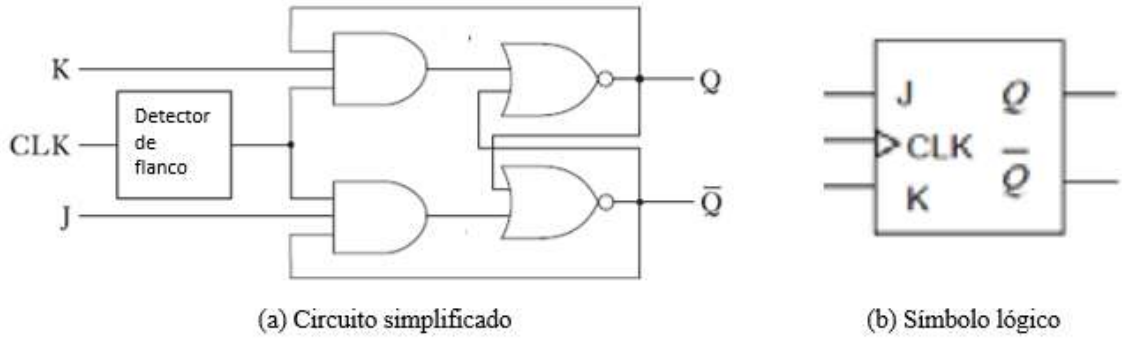
Agregando dos compuertas ***AND*** en las entradas de un flip-flop SR sincrónico por flancos, o agregando lazos de realimentación desde las salidas hacia las puertas ***AND*** de entrada, se evita el estado prohibido que se produce al querer dar las señales de grabar y de borrar simultáneamente. Cuando las entradas ***J*** y ***K*** aparecen simultáneamente activas, la salida que en ese momento se encuentre a ***1*** hace que la salida de la puerta ***AND*** correspondiente se ponga a ***1*** (la otra permanecerá en ***0***), lo que produce el cambio de estado del biestable en cualquier caso.

Funcionamiento del flip-flop JK: Cuando la entrada ***J*** tiene un nivel lógico alto (***1***) y la entrada ***K*** un nivel lógico bajo (***0***), significa que el circuito tiene

orden de grabar un bit. Si la entrada ***K*** tiene un nivel lógico alto (***1***) y la entrada ***J*** un nivel lógico bajo (***0***), significa que el circuito tiene orden de borrar el bit. Por tanto, no deben estar ambas entradas ***J*** y ***K*** en nivel lógico alto simultáneamente. La salida $\overline{Q}$ se caracteriza por presentar siempre el estado opuesto al de la salida ***Q***, incluso si se aplica simultáneamente orden de grabar y borrar, a este modo de funcionamiento se le traduce a veces como *modo de basculación*.

En la figura 5 se muestra el circuito simplificado y el símbolo lógico de un flip-flop JK.

Figura 5
Circuito simplificado y símbolo lógico del flip-flop JK



Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. Fundamentos de los Circuitos Secuenciales

El comportamiento del flip-flop JK se puede resumir en la tabla 10, conocida como tabla de la verdad completa del flip-flop JK, y en la tabla 11, la cual es la tabla de la verdad resumida del flip-flop JK.

Tabla 10
Tabla de la verdad completa del flip-flop JK

J	K	Qn	Qn+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. Fundamentos de los Circuitos Secuenciales

Tabla 11
Tabla de la verdad resumida del flip-flop JK

S	R	Qn+1	
0	0	Qn	Mantiene el valor (no graba, no borra)
0	1	0	Reset (borra)
1	0	1	Set (graba)
1	1	$\overline{Qn}$	Estado de basculación

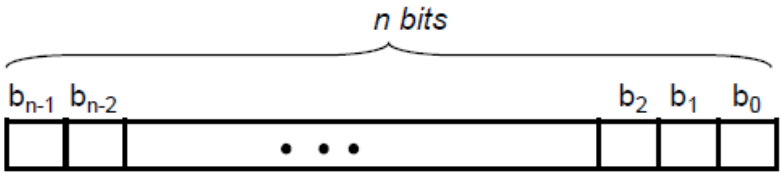
Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. Fundamentos de los Circuitos Secuenciales

Registros

Un *registro* es un circuito lógico secuencial que se utiliza para almacenar y desplazar datos digitales (***1's*** y ***0's***) sin una secuencia característica interna de estos. Son capaces de memorizar un conjunto de bits y su capacidad de almacenamiento lo convierte en un importante dispositivo

de memoria. Un registro consta, básicamente, de un conjunto de celdas de almacenamiento binarias, las cuales son generalmente flip-flop's, más un conjunto de puertas lógicas encargadas de realizar su conexión. Un esquema básico simplificado de un registro e muestra en la figura 6.

Figura 6
Esquema básico simplificado de un registro

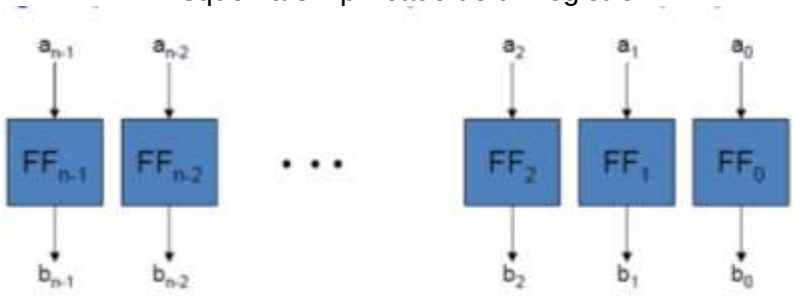


Sistemas Digitales I. Sistemas Secuenciales. Universidad Pública de Navarra

Los registros son muy importantes en las aplicaciones que precisan almacenar y transferir datos dentro de un sistema digital ya que, tanto los datos que se procesan en dichos sistemas, como la información sobre las operaciones a realizar, son almacenados en registros antes de ser procesados. Se diferencian de las memorias en que el tiempo de almacenamiento es menor. La capacidad de desplazamiento de un registro es la que permite el movimiento de los datos de una etapa o celda de almacenamiento a otra, dentro del registro.

La diferencia entre un registro y un flip-flop o biestable es que éste solo puede almacenar un bit, mientras que un registro es capaz de almacenar ***n*** bits. Un registro que almacena ***n*** bits está compuesto de ***n*** flip-flop's (uno por cada bit), cada uno con una entrada $\boldsymbol{a_i}$ y una salida $\boldsymbol{b_i}$, como se puede observar en la figura 7.

Figura 7
Esquema simplificado de un registro



Sistemas Digitales I. Sistemas Secuenciales. Universidad Pública de Navarra

Clasificación de los Registros

Según el modo en que los datos ingresan en las celdas de almacenamiento de un registro y la forma en que los datos se extraen de dichas celdas, se tienen siguientes configuraciones básicas:

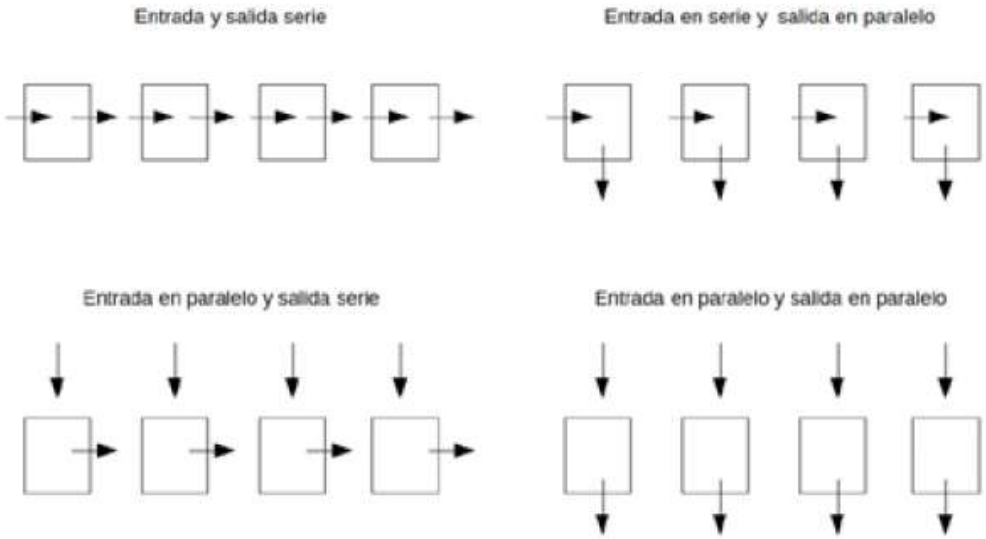
- Registros de entrada serie – salida serie
- Registros de entrada serie – salida paralelo
- Registros de entrada paralelo – salida serie
- Registros de entrada paralelo – salida paralelo

Un esquema básico simplificado de cada una de estas configuraciones se puede observar en la Figura 8.

La clasificación anterior no es exhaustiva ya que también hay registros que combinan en un mismo circuito dos o más configuraciones de los tipos presentados, por lo que existen otras variantes de registros, pero basados en las configuraciones anteriormente presentadas.

A excepción del registro con entrada y salida en paralelo de la clasificación anterior, a todos los demás se les llama *registros de desplazamiento*. Esto se debe a que los datos se tienen que desplazar pasando cada bit de una etapa a la siguiente, es decir, de flip-flop a flip-flop.

Figura 8
Esquema simplificado de las diferentes configuraciones básicas de los registros



Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. Fundamentos de los Circuitos Secuenciales

Registros de Desplazamiento

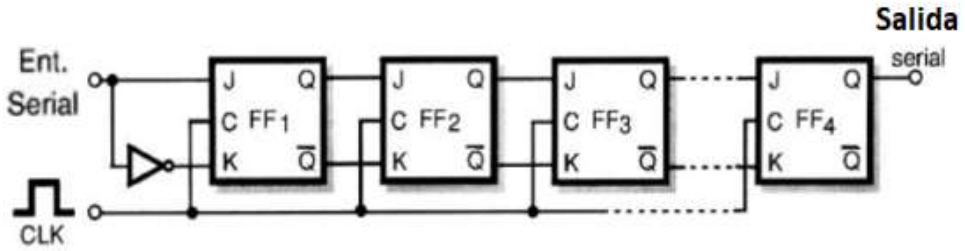
Los *registros de desplazamiento* permiten desplazar la información (los bits) a lo largo del registro (de un biestable a otro). Los dos tipos de registros de desplazamiento son *serie* y *paralelo*. En un *registro de desplazamiento paralelo* los bits se almacenan simultáneamente a partir de líneas paralelas, mientras que en un *registro de desplazamiento serie*, los bits se almacenan de uno a uno mediante una línea serie.

Como se mencionó anteriormente, tenemos tres tipos de registros de desplazamiento:

- Registros de desplazamiento de entrada serie – salida serie
- Registros de desplazamiento de entrada serie – salida paralelo
- Registros de desplazamiento de entrada paralelo – salida serie

En la figura 9 se muestra un registro de desplazamiento de entrada serie – salida serie implementado con flip-flop´s JK.

Figura 9
Registro de desplazamiento de entrada serie – salida serie con flip-flop JK



https://www.incb.com.mx/index.php/curso-de-electronica/96-curso-de-electronica-electronica-digital/3419-curso-de-electronica-electronica-digital-parte-11-como-funcionan-los-registros-de-desplazamiento-shift-registers-cur6004s

Memorias Semiconductoras

Las *memorias* son dispositivos de almacenamiento de datos binarios de largo o corto plazo. En el tema anterior se vieron los registros de desplazamiento, que son dispositivos de almacenamiento, esencialmente un registro es una memoria a pequeña escala. En este capítulo se estudian las memorias para almacenamiento a largo plazo y de cantidades grandes de información.

Unidades de Medida de Datos Binarios

La unidad básica de medida de datos binarios es el ***bit***. Como regla general las memorias almacenan datos en unidades, generalmente, de ***8 bits*** (***8 bits = 1 byte***). El ***byte*** se puede dividir en dos unidades de ***4 bits***, que reciben el nombre de ***nibbles***. Una unidad completa de información se denomina ***palabra*** y está formada por uno o varios ***bytes***. Algunas memorias almacenan datos en grupos de ***9 bits***; los cuales constan de ***un byte*** más *un bit de paridad* (bit adicional que se agrega a un código binario para garantizar la precisión de la transmisión o el almacenamiento de datos).

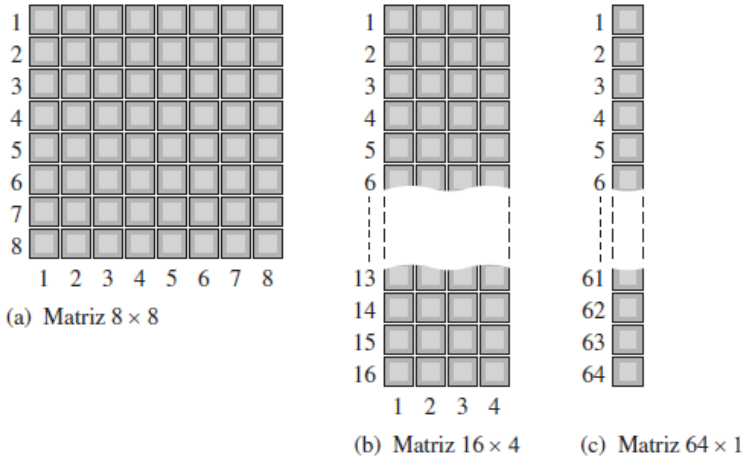
Matriz de Celdas de Memoria

Cada elemento de memoria puede almacenar un '***1***' o un '***0***' y se denomina *celda*. Las memorias están formadas por *matrices de celdas*. En la figura 10 se ilustra un ejemplo de una matriz de ***64*** celdas. Una matriz de celdas se puede organizar de muchas maneras en función de las unidades de datos.

La figura 10(a) muestra una matriz de ***64 celdas*** organizada como una memoria de ***8 palabras*** de ***8 bits*** cada una (***8 × 8***), que se puede entender como una memoria de ***64 bits*** o como una memoria de ***8 bytes***. La figura 10(b) nos muestra la misma matriz organizada como una memoria de ***16 palabras*** de ***4 bits*** cada una (***16 × 4***), que es una memoria de ***16 nibbles***. Finalmente, la figura 10(c) presenta dicha matriz organizada como una memoria de ***64 palabras*** de ***1 bit*** cada una (***16 × 1***), que es una memoria de ***64 bits***. La matriz de la figura 10 es una matriz bidimensional ya que todas sus celdas están en un mismo plano.

Figura 10

Matriz de almacenamiento de 64 celdas organizada de tres formas distintas



Floyd, T. Fundamentos de Sistemas Digitales

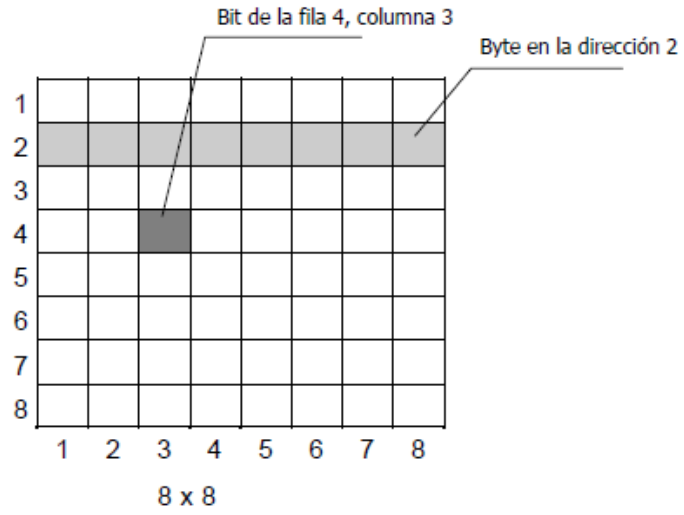
Direccionamiento en una Memoria

Una *unidad de dato*s puede ser una *celda* (almacena un dato de ***un bit***) o una *palabr*a (almacena un dato de ***un byte*** o de ***8 bits***). La posición de una *unidad de datos* en una matriz de memoria se denomina *dirección*.

La dirección de cada celda de almacenamiento (dirección de un bit) se especifica por una fila (numeradas de arriba hacia abajo) y una columna (numeradas de izquierda a derecha), y la dirección de una palabra (dirección de un byte) se especifica únicamente mediante la fila.

En la figura 11 se ilustra la dirección de una celda y la dirección de una palabra en una matriz bidimensional de ***64 celdas*** organizada como una memoria de ***8 bytes***. En este caso el dato más pequeño que se puede direccionar es un dato de ***un bit***.

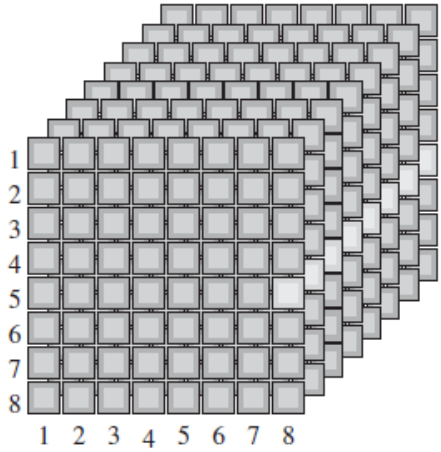
Figura 11
Dirección de una celda (bit) y de una palabra (byte) en una memoria de 64 celdas organizada como una memoria 8 bytes.



Las matrices de memoria pueden ser tridimensionales, es decir, que tienen grupos de celdas en planos distintos. En una matriz tridimensional de

512 celdas organizada como una memoria de ***64 bytes***, la dirección de ***un byte*** se especifica mediante la fila y la columna correspondientes, como se observa en la figura 12. En este caso, el grupo más pequeño de bits al que se puede acceder es ***un byte*** u ***ocho bits***.

Figura 12
Dirección de una palabra (byte) en una memoria de 512 celdas organizada como una memoria 64 bytes.



Floyd, T. Fundamentos de Sistemas Digitales

Como se puede ver en las figuras 10 y 11, la dirección en una matriz de celdas depende de cómo se organice la memoria en unidades de datos.

Capacidad de Almacenamiento de una Memoria

La *capacidad de almacenamiento* de una memoria puede expresarse en unidades de bits, bytes o palabras, siendo el byte la unidad utilizada con mayor frecuencia. Para expresar la capacidad en bits, se multiplica el número de palabras que puede almacenar por el tamaño de la palabra.

Una memoria se identifica por su *capacidad de almacenamiento*, esta capacidad es el número total de unidades de datos (bits, bytes o palabras) que

puede almacenar. La manera habitual es indicar ***nnK x mm***, en el que ***nn*** es el número de direcciones que tiene la memoria con capacidad para poder almacenar ***mm*** bits en cada una.

En la matriz de memoria organizada en bytes de la figura 10(a), la capacidad total es de ***8 bytes*** equivalente a ***64 bits***.

En la matriz de memoria organizada en nibbles de la figura 10(b), la capacidad total es de ***16 nibbles***, que es lo mismo que ***64 bits***.

En la matriz de memoria organizada en bits de la figura 10(c), la capacidad total es de ***64 bits***. En matriz de memoria organizada en bytes de la figura 11, la capacidad es de ***64 bytes***, que es lo mismo que ***512 bits***.

Las medidas más empleadas para la capacidad de memoria son las indicadas en la tabla 11.

Tabla 11
Capacidades de memoria más usadas

Denominación	Abreviatura	Equivalencia	Valor aproximado
Kilobyte	KB	$2^{10} = 1{,}024 \times 10^{3}$	1000 bytes
Megabyte	MB	$2^{20} = 1{,}049 \times 10^{6}$	1 millón de bytes
Gigabyte	GB	$2^{30} = 1{,}074 \times 10^{9}$	1000 millones de bytes
Terabyte	TB	$2^{40} = 1{,}10 \times 10^{12}$	1 billón de bytes
Petabyte	PB	$2^{50} = 1{,}13 \times 10^{15}$	1000 billones de bytes

Ejemplo 12:

El número de palabras de ***4 bits*** cada una de una memoria es ***16K***

a) ¿Cuántas palabras de ***4 bits*** es capaz de almacenar la memoria?
b) ¿A cuántas posiciones se puede acceder o cuántas posiciones son direccionables en la memoria?

c) ¿Cuántas unidades de datos de un bit posee la memoria?

Respuesta a:

16K = 16 x 1024 = 16.384 palabras de ***4 bits***

Respuesta b:

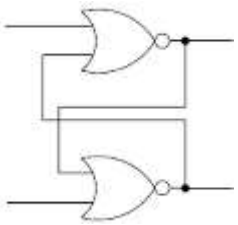
El número de direcciones o posiciones direccionables de una memoria es igual al número de palabras que la memoria es capaz de almacenar (***16.384***)

Respuesta c:

16K x 4 = 65.536 bits

AUTOEVALUACIÓN UNIDAD II

1. Considere y analice el circuito lógico de la figura. Construya su tabla de la verdad y explique su funcionamiento:



2. ¿Cuál es la diferencia entre un flip-flop SR y un flip-flop JK?
3. ¿Cuál es el propósito de la entrada de reloj en un flip-flop?
4. Diga qué es un *registro* y en qué se diferencia de un *flip-flop*.
5. Diga qué es un *registro de desplazamiento* y como se clasifica.
6. ¿En cuántos impulsos de reloj se desplaza un dato en un registro?
7. Una palabra de 128 bits está formada por:

   a) 8 bytes b) 20 nibbles c) 16 bytes d) 10 bytes y 8 nibbles

8. El número de direcciones de una memoria es ***1K***, y puede almacenar ***8 bits*** en cada dirección:

   a) ¿Cuántas palabras de ***4 bits*** es capaz de almacenar la memoria?
   b) ¿A cuántas posiciones se puede acceder o cuántas posiciones son direccionables en la memoria?
   c) ¿Cuántas unidades de datos de un bit posee la memoria?

9. ¿Cuántas líneas de dirección tiene una memoria con 256 direcciones?

   a) 32 b) 256 c) 8 d)16 e) 128 f) 6

UNIDAD III

ARQUITECTURA DEL COMPUTADOR

En sentido general, la arquitectura de un sistema es un modelo que representa los componentes necesarios para desarrollar dicho sistema dentro de un contexto específico y desde una perspectiva particular.

La arquitectura del computador es el enfoque, desde la perspectiva de un programador, del comportamiento funcional de un sistema de computación, que incluye aspectos tales como los tipos de instrucciones y datos que es capaz de procesar, la velocidad de procesamiento, el tamaño de los diferentes tipos de datos y los tipos de operaciones que puede realizar. El término *arquitectura del computador* generalmente se refiere en forma amplia tanto a la arquitectura como a la organización del computador.

En esta unidad estudiaremos los diferentes subsistemas y componentes que integran la arquitectura del computador, así como la interconexión de dichos subsistemas, los flujos mutuos de datos entre ellos (comunicación) y el control de las operaciones (sincronización) que permiten su funcionamiento.

Microprocesadores

El *microprocesador* es un circuito integrado que procesa y ejecuta las operaciones lógicas y aritméticas necesarias para el funcionamiento de las computadoras. Estas operaciones lógicas y aritméticas son los cálculos que permiten la ejecución de los programas, tanto los propios del sistema operativo, como las aplicaciones ejecutadas por el usuario, por tanto, el

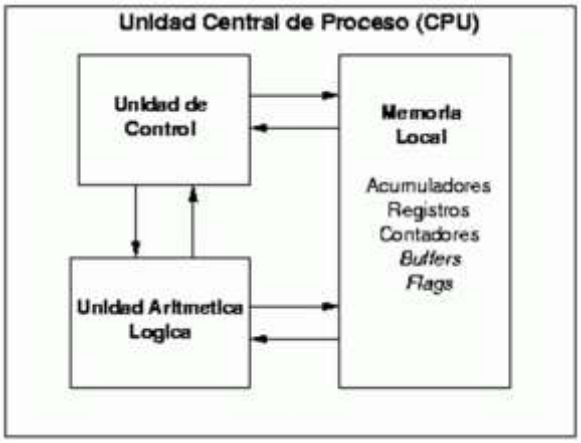
microprocesador es el cerebro de la computadora, diseñado para ejecutar los programas.

El microprocesador (µP) solo ejecuta instrucciones programadas en lenguaje de bajo nivel (lenguaje de máquina), por lo que requiere de un sistema operativo que provea las funcionalidades adecuadas a los programas, además de gestionar los recursos del hardware.

Con frecuencia se hace referencia al microprocesador como la *unidad central de procesamiento* (CPU), aunque estrictamente la CPU es la parte del microprocesador cuya función consiste en reconocer y ejecutar las instrucciones de un programa y procesar los datos.

A continuación se muestra, en la figura 13, la arquitectura general de un microprocesador, donde se observa que el microprocesador consta de la unidad aritmético-lógica (ALU), la unidad de control (UC) y los registros.

Figura 13
Arquitectura general de un microprocesador (µP)



Unidad Lógico-Aritmética (ALU)

Es el órgano operativo del microprocesador. Se encarga de realizar las operaciones aritméticas (suma, resta, multiplicación y división), lógicas (OR,

AND, NOT,…) con números enteros y números reales, operaciones de desplazamiento de datos, y en general de la gestión de los datos.

Unidad de Control (UC)

Genera señales de temporización necesarias para leer instrucciones de programa en memoria, determinando la temporización y secuencia de las operaciones, ya que cada operación se caracteriza por la cantidad de ciclos de reloj que necesita para ser ejecutada. Gobierna y sincroniza la actuación conjunta de las unidades internas y externas del microprocesador.

Memoria Local

Es el área de las células de memoria, es decir, la zona del microprocesador donde se almacenan los datos. También se le llama zona de registros.

Registros

Se emplean como almacenamiento temporal de los datos internos que el microprocesador utiliza mientras ejecuta las instrucciones, dado que solo es posible leer una dirección de memoria a la vez. Es un elemento de memoria de acceso rápido dentro del microprocesador. Cada microprocesador dispone de un conjunto de registros que lo caracterizan y cumplen una determinada función. Los tipos de registro que componen un microprocesador son:

- *Acumuladores*
- *Registros de trabajo*
- *Contador de programa*
- *Registros de direcciones*
- *Registros de instrucciones*
- *Registros de estado*
- *Buffer de datos*

- *Puntero de pila*

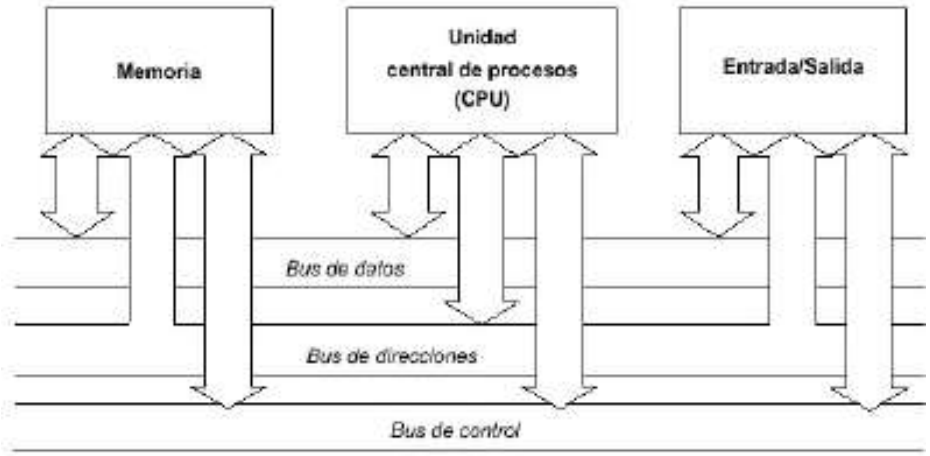
Buses

Son canales a través de los cuales se transfieren los datos entre los diferentes subsistemas que componen un computador y su función es permitir la interconexión lógica entre estos subsistemas. Los buses son sistemas digitales formados por cables o pistas de un circuito impreso, resistores, condensadores y circuitos integrados que manejan un protocolo que les permite transmitir datos útiles.

En la arquitectura interna del microprocesador encontramos tres tipos de buses de acuerdo a su función y tipo de información que manejan, como se observa en la figura 14.

Figura 14

Diagrama de una memoria mostrando los buses de comunicación con el procesador: bus de direcciones, bus de datos y bus de control



http://bnm.me.gov.ar/giga1/documentos/EL007282.pdf

Bus de Direcciones

En una operación de escritura o de lectura de una unidad de datos, se selecciona una dirección introduciendo un *código de dirección* binario (un conjunto de bits), que representa la dirección deseada, en un conjunto de líneas denominado *bus de direcciones*. El código de dirección se decodifica internamente y de esa forma se selecciona la dirección adecuada (ver figura 14). La capacidad de memoria viene dada por el *bus de direcciones* que establece el máximo número de posiciones direccionables por el computador, es decir, el número de líneas del bus de direcciones depende de la capacidad de la memoria. Si se tienen $\boldsymbol{2^n}$ posiciones direccionables, entonces el bus de direcciones será de ***n*** bits (***n*** líneas).

Ejemplo 13:

¿Cuántas líneas tiene el bus de direcciones de una memoria que es capaz de almacenar ***16K*** palabras de ***4 bits*** cada una?

$\boldsymbol{16.384 = 2^{14} = 2^n}$, ***n = 14*** (***n***: número de bits o líneas del bus de direcciones)

Bus de Datos

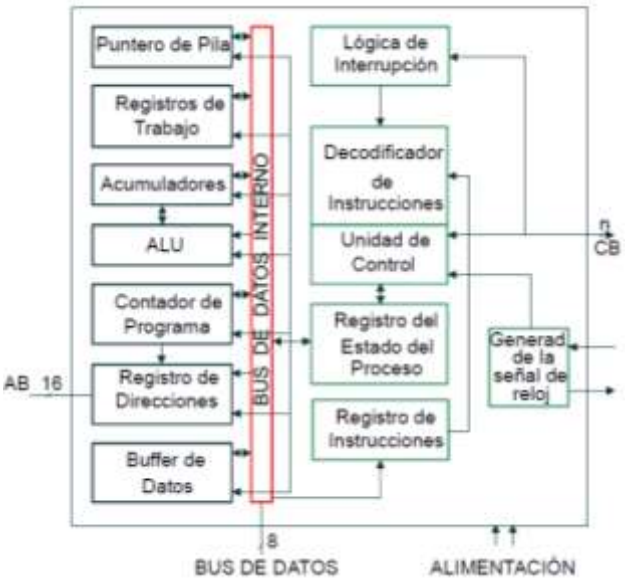
Las unidades de datos se introducen en la memoria durante la operación de escritura y se extraen de la memoria durante la operación de lectura a través de un conjunto de líneas que se denominan *bus de datos*. Como se indica en la figura 14, el bus de datos es bidireccional, lo que significa que los datos pueden ir en cualquiera de las dos direcciones (hacia la memoria o desde la memoria).

Bus de Control

Es un tipo de bus de sistema en un ordenador que se encarga de gestionar la comunicación entre los principales componentes del sistema, como la unidad central de procesamiento (CPU), el controlador de acceso directo a memoria (DMA), el controlador de interrupciones y los módulos de memoria. Un bus de control se emplea principalmente para controlar el flujo de datos dentro del sistema gestionando el acceso a las líneas de datos y a las direcciones. Las señales de este bus permiten transmitir tanto órdenes como información y su función principal es facilitar que el sistema funcione, sin que se produzca colisión de información.

La arquitectura básica interna de un microprocesador, donde se observa la zona de registros y la zona de control, con los bloques más importantes, los buses de datos y el generador de señal de reloj se muestra a continuación en la figura 15.

Figura 15
Arquitectura básica interna de un microprocesador (µP)



http://www.isa.uniovi.es/~alonsog/Microcontrolador/T2%20El%20Microprocesador.pdf

Tipos de Memorias en Microprocesadores

Una *memoria* es un dispositivo electrónico capaz de guardar y retener información y datos por un periodo de tiempo. Posee casillas o localidades (celdas), cada una con capacidad de almacenar un dato de un tamaño específico medido en bytes (***1 byte = 8 bits***). Tiene un bus de direcciones para identificar cada una de las localidades y un bus de datos por donde entran y salen los datos hacia y desde cada una de las casillas o localidades de la memoria.

Las *memorias de semiconductores* son memorias implementadas mediante dispositivos semiconductores que incorporan un mecanismo de direccionamiento cableado y, por lo tanto, son estáticas y de acceso aleatorio. Distinguiremos dos categorías dentro de este tipo de memorias:

a. **Memorias de Solo Lectura**, utilizadas habitualmente para almacenar los programas de arranque de los ordenadores. Son memorias no volátiles y de lectura no destructiva. Existen distintas variantes de estas memorias que proporcionan características adicionales:

1. ***Memoria de solo lectura (ROM: Read Only Memory)***: Es una memoria no volátil, es decir, conserva la información aun cuando se desconecta la alimentación eléctrica, por tanto, es de almacenamiento permanente. Su funcionamiento se basa en el principio fusible. Se denomina de solo lectura porque permite la lectura de la información y no su escritura, es decir, los datos almacenados no se pueden modificar. Los datos contenidos en ellas son implementados en fábrica.

2. ***Memoria de solo lectura programable (PROM: Programmable ROM)***: Es una memoria no volátil de solo lectura programable por el usuario final a través de pulsos de corriente mediante un dispositivo especial. Al igual que en las ROM's básicas, la programación solo se puede realizar una vez, quedando la información permanentemente almacenada en la memoria sin posibilidad de modificación. Se conoce también como *OTP: One Time Programmable* (programable una vez).
3. ***Memoria de solo lectura programable y borrable (EPROM: Erasable Programmable Read Only Memory)***: Es una memoria no volátil de solo lectura que puede borrarse y grabarse muchas veces. La información es almacenada, al igual que en las PROM, por el usuario mediante un grabador especial y el borrado se realiza mediante luz ultravioleta, para lo cual disponen de una ventana de cristal en su superficie por la que se somete a estos rayos. Estas memorias permiten múltiples reprogramaciones. El borrado se realiza en todas las posiciones de memoria a la vez.
4. ***Memoria de solo lectura programable y borrable eléctricamente (EEPROM: Electrically Erasable Programmable Read Only Memory)***: Es una memoria no volátil de solo lectura que puede borrarse y grabarse muchas veces. Se diferencian de las anteriores en que tanto el grabado como el borrado se realiza eléctricamente. Permite la lectura y escritura de múltiples posiciones de memoria en la misma operación, por lo que pueden ser selectivos a nivel de palabra, es decir, pueden reprogramarse posiciones específicas de la memoria. El grabado y el borrado se realizan mediante impulsos eléctricos por medio de los voltajes de operación normal de los circuitos electrónicos, por lo que no disponen de ventana de cristal en su superficie.

5. ***ROM con mascara (MROM: Masked ROM)***: Es una memoria no volátil de solo lectura cuyo contenido se graba durante la fabricación del chip, es decir, se fabrica con los datos almacenados de forma permanente, por lo que su contenido no puede ser modificado de ninguna forma.
6. ***Memoria Flash***: Es una memoria no volátil programable eléctricamente. Permite la lectura y escritura de múltiples posiciones de memoria en la misma operación. También permiten el borrado selectivo aunque a diferencia de las EEPROM este se realiza a nivel de bloque y no de palabra. El grabado y el borrado se realizan mediante impulsos eléctricos por medio de los voltajes de operación de aproximadamente ***5 V***, menores que en las memorias EEPROM que son de aproximadamente ***13 V***. Posee una velocidad de operación mayor que la EEPROM y tiene mayor densidad que esta última, además tolera más ciclos de escritura/borrado.

b. **Memorias de Lectura-Escritura**, utilizadas como memoria principal en los ordenadores. Son memorias volátiles de lectura no destructiva. Están presentes en cualquier ordenador y es la memoria donde se almacenan de forma temporal los datos de los programas que se están utilizando. Se conocen como memorias de acceso aleatorio (***RAM: Random Acces Memory***). Dependiendo del mecanismo de almacenamiento utilizado para cada posición de la memoria pueden clasificarse en dos categorías: estáticas (***SRAM***), en las que se utilizan flip-flop's y dinámicas (***DRAM***), que utilizan condensadores. Existen dos tipos de memorias RAM:

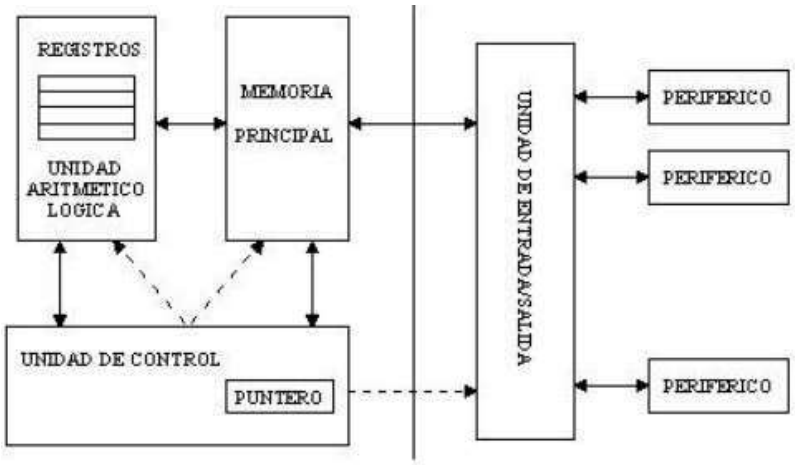
1. ***Memoria RAM de tipo DDR (Double Data Rate)***: Se caracterizan por ser capaces de llevar a cabo dos operaciones de lectura o escritura en cada ciclo de reloj.

2. ***Memoria RAM de tipo SDR (Single Data Rate)***: Solo ejecutan una operación de lectura o escritura en cada siclo de reloj.

Unidad de Entrada/Salida (E/S)

La *unidad de entrada/salida (E/S)* permiten la comunicación del sistema microprocesador con el mundo exterior. Los dispositivos de E/S se denominan habitualmente *periféricos* (por ejemplo: teclado, pantalla, impresora, unidades de disco, etc.). Cualquier periférico necesita un módulo adicional que permite realizar su conexión con los buses del sistema microprocesador; este módulo se denomina *interface*. En la figura 16 se ilustra la arquitectura básica del microprocesador incluyendo la unidad de entrada/salida y los dispositivos periféricos.

Figura 16
Arquitectura básica interna de un microprocesador incluyendo la unidad de entrada/salida (E/S) y los periféricos



https://asir.sudo.es/fhw/docs_exposiciones_y_cuestionarios/67-Microprocesador3.pdf

Representación de las Instrucciones y de los Datos

En informática, un *dato* es un número, una palabra, un valor o un concepto expresados mediante uno o varios símbolos. Asimismo, la *información* es un conjunto de datos ordenados que aportan un conocimiento, el cual tiene un propósito, es decir, la información permite generar, modificar, regular o detener acciones que controlan las operaciones de un programa informático, que a su vez puede emitir órdenes a un proceso físico.

Para generar las acciones que permiten realizar las operaciones en un programa informático, la información debe ser procesada en un ordenador, el cual, partiendo de unos datos de entrada, los procesa y genera una información de salida. Este procesamiento de información se realiza según un conjunto de *instrucciones*.

En conclusión, la ejecución de un programa informático implica el procesamiento de información (datos) en un ordenador, según un conjunto de instrucciones. Estos datos e instrucciones deben ser interpretados por el computador, para lo cual se utilizan diferentes formas de representación que le permita al computador reconocer, almacenar, modificar y generar distintos tipos de información.

Composición y Estructura de la Información

Los datos (información) pueden ser magnitudes numéricas, palabras, o un conjunto de valores cualitativos que se representan mediante símbolos, por lo tanto, la información se representa mediante una secuencia de símbolos que son tomados de los sistemas alfabéticos y numéricos y son codificados para poder ser reconocidos, interpretados, almacenados y procesados por el computador.

Sistema alfabético: Un sistema alfabético o alfabeto, es un sistema de escritura formado por símbolos llamados letras que representan sonidos (fonemas) en una lengua determinada.

Sistema numérico: Un sistema numérico o de numeración, es un conjunto de símbolos llamados números y reglas que permiten expresar todas las cantidades válidas que se pueden construir con dichos números.

Codificación: La codificación de datos es el proceso de convertir información en un formato legible por un ordenador para su almacenamiento y procesamiento.

Representación de las Instrucciones

Una *instrucción* es un conjunto de símbolos que la computadora es capaz de interpretar con la finalidad de realizar las operaciones de las que consta un programa.

El funcionamiento del procesador está determinado por las instrucciones que ejecuta. Estas instrucciones se denominan *instrucciones máquina* o *instrucciones del computador*. Al conjunto de instrucciones distintas que puede ejecutar el procesador se denomina *repertorio de instrucciones*.

Elementos de una Instrucción Máquina

Cada instrucción debe contener la información que necesita el procesador para su ejecución. Dichos elementos son:

Código de operación: especifica la operación a realizar (suma, resta, etc.). La operación se indica mediante un código binario denominado *código de operación*, o de manera abreviada *CodOp*.

Referencia a los operandos origen (Modo Origen): la operación puede implicar a uno o más operandos origen, es decir, operandos que son entradas para la operación.

Referencia al resultado (Modo Destino): la operación puede producir un resultado y entonces puede ser necesario indicar donde se almacenará.

Referencia a la siguiente instrucción: Indica al procesador de donde captar la siguiente instrucción tras completarse la ejecución de la instrucción actual.

Formatos de Instrucciones

Los formatos de instrucciones definen la organización de los bits dentro de una instrucción en términos de las partes que la componen. La instrucción debe incluir información acerca de la operación a ejecutar y sobre qué datos. Entonces, una instrucción debe incluir mínimamente el código de la operación y un mecanismo para llegar hasta el/los operando/s, mediante lo que llamamos modos de direccionamiento.

Campos de una Instrucción

Normalmente una instrucción se divide en dos campos: *código de operación*, designa la operación que va a ser realizada, y *datos de la operación* u *operandos*, dependiendo del tipo de instrucción, este campo puede estar dividido en otros o ser único, incluso no existir.

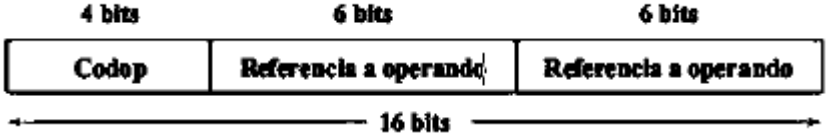
Código de operación: este campo está presente en todas las instrucciones y es el campo que indica la operación a realizar. No hay dos operaciones diferentes con el mismo código de operación, por lo que este campo diferencia una instrucción de otra. El código de operación suele ocupar los bits más significativos de una instrucción (los bits ubicados más a la izquierda). Si una instrucción ocupa más de una palabra, el código de operación estará en la primera palabra que lea la CPU.

Operandos: este campo no está presente en todas las instrucciones ya que algunas instrucciones no emplean datos, y en otras instrucciones, los datos están localizados implícitamente en el propio código de operación. La longitud de este campo es normalmente variable dependiendo del número de operandos que utilice la instrucción y de la forma que se indique a la CPU el acceso a los mismos.

Representación de las Instrucciones

Dentro del computador, cada instrucción se representa por una secuencia de bits. La figura 17 muestra un ejemplo sencillo de formato de instrucción. En la mayoría de los repertorios de instrucciones se emplea más de un formato.

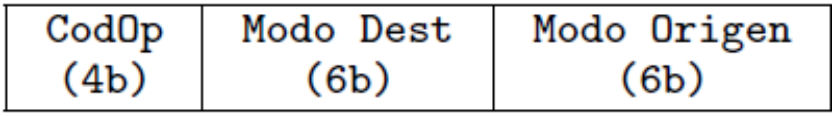
Figura 17
Ejemplo de un formato sencillo para la representación de instrucciones



Stallings W. Organización y Arquitectura de Computadores.

Como se dijo anteriormente, una instrucción debe incluir mínimamente el código de la operación y un mecanismo para llegar hasta el/los operando/s, mediante lo que llamamos *modos de direccionamiento*, por tanto, los campos *Referencia a Operando* de la figura 17 deben contener los direccionamientos a los datos de origen (Modo Origen) y los de destino (Modo Destino), como se observa en la figura 18.

Figura 18
Ejemplo de un formato sencillo para la representación de instrucciones que contiene los tres campos mínimos de una instrucción



Representación de instrucciones. Organización de Computadoras.
Universidad Nacional de Quilmes.

Entonces, la instrucciones de la figura 18 se ensamblan como se muestra en la figura 19 (Notar que el código maquina viene de un código fuente, se expresa en binario y luego se compacta en hexadecimal a los fines de una mejor legibilidad).

Figura 19
Un formato sencillo de instrucciones expresadas en código fuente y en código máquina binario y hexadecimal

Código fuente	Código máquina	Hexa
MOV R0, R1	0001 100000 100001	1881

Representación de instrucciones. Organización de Computadoras.
Universidad Nacional de Quilmes.

Representación Interna de los Datos

Existen varios tipos de representación de datos, entre los cuales distinguimos cinco:

- **Representación de textos**

La información es suministrada al ordenador en forma escrita mediante un conjunto de símbolos denominados caracteres, tomados del alfabeto que habitualmente utiliza el usuario.

- **Representación de valores numéricos**

  La información se introduce en el ordenador en forma escrita mediante uno o un conjunto de caracteres, los cuales son símbolos tomados del sistema de numeración decimal y forman los dígitos de la cantidad introducida.

- **Representación de audios**

  La información se suministra al computador en distintos formatos, como MP3, WAV, etc. Cada formato utiliza diferentes algoritmos de compresión y descompresión.

- **Representación de imágenes**

  La información se suministra al computador en distintos formatos, tales como JPG, BMP, etc. Cada formato utiliza diferentes algoritmos de compresión y descompresión.

- **Representación de Videos**

  La información se introduce en el computador en distintos formatos, como MP4, AVI, etc. Cada formato utiliza diferentes algoritmos de compresión y descompresión.

La codificación facilita y optimiza las tareas de representación, almacenamiento, procesamiento, transmisión y generación de la información, obedeciendo a las siguientes condiciones o limitaciones:

1. La memoria del ordenador es finita, por lo que las representaciones deben se acotadas.
2. Las unidades funcionales del computador están diseñadas para trabajar con cadenas de bits de tamaños específicos.

3. Los elementos de comunicación (buses) entre las unidades funcionales permiten transmitir simultáneamente un número de bits determinado.

El proceso de codificación permite transformar la información de una forma de representación comprensible por los usuarios a una forma de escritura legible por el ordenador y viceversa. Los tipos de codificación de información principales son las siguientes:

- **Codificación de caracteres**

  La codificación de la información de tipo texto escrito (alfanumérica) utiliza un conjunto de 8, 16 y 32 bits según el sistema de codificación utilizado. Los sistemas de codificación de caracteres más utilizados son los siguientes:

  - ✓ **ASCII (American Standard Code for Information Interchange)**: Es el más utilizado hoy en día. Dispone de 8 bits para representar 256 caracteres, aunque solo utiliza los 7 primeros para el uso de letras, números y caracteres especiales. Los primeros 32 caracteres (del 00 al 32) y el carácter 127 son de control, los caracteres del 33 al 126 representan los números, letras mayúsculas, letras minúsculas y caracteres especiales, y los caracteres del 128 al 255 representan letras latinas, símbolos y signos gráficos.
    - ❖ Para mayor información sobre Código ASCII, puede visitar el siguiente sitio web: https://elcodigoascii.com.ar/

  - ✓ **UNICODE**: Utiliza una codificación de longitud variable (8, 16, 32 bits) para representar más de 143.000 caracteres. Esto permite incluir una amplia gama de idiomas y símbolos, incluyendo números, símbolos,

signos diacríticos (ortográficos), scripts (conjunto de instrucciones de programación) no latinos y emojis. Esto convierte al código Unicode en una herramienta imprescindible en aplicaciones internacionales y multilingües.

❖ Para mayor información sobre Código UNICODE puede visitar: https://www.godaddy.com/resources/es/crearweb/unicode-que-es-como-funciona-aplicacion

- **Codificación numérica**

  Cualquier conjunto de números, como los números naturales, enteros, etc., es infinito, mientras que la capacidad de representación de las computadoras es finita, por lo que no es posible representar todos los números.

En informática se asigna una cantidad fija ***n*** de bits para representar un número. Teniendo en cuenta que ***n*** bits permiten representar 2^n números distintos, existirá un rango de representación en un intervalo comprendido entre dos valores, el menor y el mayor valor representable.

La diferencia que existe entre un número representable y el inmediato siguiente se conoce como *resolución de representación*.

Para la representación de cantidades numéricas en informática se deben tener en cuenta los siguientes factores:

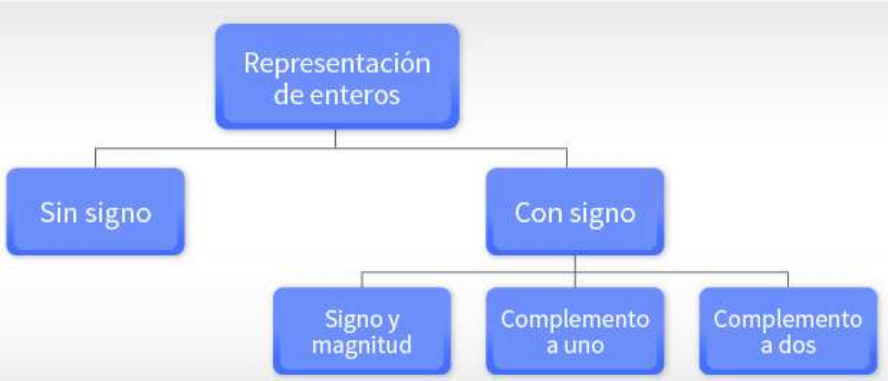
1. Tipo de números a representar (reales, enteros, etc.)
2. Rango de números representables
3. Posición del dato numérico
4. Precisión de la representación

Representación de Números Enteros

Los enteros pueden ser positivos o negativos, por lo que el intervalo definido para los números negativos sería del cero hasta infinito negativo, y el intervalo definido para números positivos sería del cero hasta infinito positivo. Para almacenar todos los enteros de estos intervalos se requeriría una computadora con capacidad de almacenamiento infinita, lo cual resulta prácticamente imposible.

Para usar la memoria de una computadora de manera más eficiente, se han desarrollado dos categorías de representación de enteros: enteros sin signo y enteros con signo. Los enteros con signo se pueden representarse de tres maneras distintas. En la figura 20 se indica mediante un diagrama de flujo las formas de representación de números enteros.

Figura 20
Diagrama de flujo par la representación de números enteros



https://www.uacj.mx/CGTI/CDTE/JPM/Documents/IIT/Representacion/representacion-de-numeros.html

Representación de Enteros Sin Signo

Formato de Enteros Sin Signo

Un entero sin signo es un entero que no tiene intervalo, su rango está entre 0 y el infinito positivo. No obstante, como no hay manera de que una computadora represente todos los enteros de este intervalo, la mayoría de las computadoras definen una constante llamada el *entero máximo sin signo*.

Un entero sin signo varía entre 0 y esta constante. El entero máximo sin signo depende del número de bits que la computadora asigna para almacenar un entero sin signo.

Intervalo de los Enteros Sin Signo

El intervalo de los enteros sin signo que pueden ser almacenados en un ordenador se define mediante la siguiente expresión: ***Intervalo = 0,…, (2^n – 1)***, donde ***n*** es el número de bits asignado para representar un entero sin signo.

Ejemplo 14:

Si tenemos disponibles 8 bits para el almacenamiento de un entero sin signo, ¿cuál es el intervalo de números que se pueden almacenar?

Intervalo = 0,…, (2^8 – 1) = 0,…, (256 – 1) = 0,…, 255

Esto quiere decir que podemos almacenar los números enteros sin signo del ***0*** al ***255***.

Almacenamiento de Enteros Sin Signo

Para almacenar un número entero sin signo debemos seguir los siguientes pasos:

1. Calcular el intervalo de números que se pueden almacenar dependiendo de los bits asignados ***n***.
2. Verificar si el número entra dentro del intervalo de números que se pueden almacenar.
3. Convertir el número del sistema de numeración decimal al sistema de numeración binario.
4. Si el número de bits del número binario obtenido es menor que ***n***, se añaden ***0's*** a la izquierda del número binario de manera que haya un total de ***n*** bits.

Ejemplo 15:

*Almacenar el número **114** en una localidad de memoria de **8 bits***.

Intervalo = 0,…, (2^8 – 1) = 0,…, (256 – 1) = 0,…, 255

Esto quiere decir que podemos almacenar los números enteros sin signo del ***0*** al ***255***, por lo tanto, el número ***114*** está dentro del intervalo.

El equivalente en el sistema binario del número decimal ***114*** resulta ***1110010***
El número de bits que ocupa ***114*** en binario son ***7 bits***, por lo que debemos agregar un cero a la izquierda para completar los ***8 bits*** asignados.

El número listo para almacenar es ***01110010***.

Representación de Enteros Con Signo

Formato de Enteros Con Signo (Signo y Magnitud)

El almacenamiento de un entero en el formato de signo y magnitud requiere ***1 bit*** para representar el signo (***0*** para positivo, ***1*** para negativo). Es decir, por ejemplo, en una asignación de ***8 bits***, se utilizan ***7 bits*** para representar el valor absoluto (número sin signo) del número y ***1 bit*** para representar el signo. Intervalo de los Enteros Con Signo

El intervalo de los enteros con signo y magnitud que pueden ser almacenados en un ordenador se define mediante la siguiente expresión: ***Intervalo =*** – ***$(2^{n-1} – 1)$,…, $(2^{n-1} - 1)$***, donde ***n*** es el número de bits asignado para representar un entero con signo y magnitud.

Ejemplo 16:

Si tenemos disponibles 8 bits para el almacenamiento de un entero con signo y magnitud, ¿cuál es el intervalo de números que se pueden almacenar?

Intervalo = – $(2^{8-1} – 1)$,…, $(2^{8-1} – 1)$ = – $(2^{7} - 1)$,…, $(2^{7} – 1)$ = –127,…, 127

Esto quiere decir que podemos almacenar los números enteros con signo y magnitud del ***–127 al 127***.

Almacenamiento de Enteros Con Signo y Magnitud

Para almacenar un número entero con signo y magnitud debemos seguir los siguientes pasos:

1. Calcular el intervalo de números que se pueden almacenar dependiendo de los bits asignados ***n***.
2. Verificar si el número entra dentro del intervalo de números que se pueden almacenar.
3. Convertir el número del sistema de numeración decimal al sistema de numeración binario ignorando el signo.
4. Si el número de bits del número binario obtenido es menor que (***n – 1)***, se añaden ***0's*** a la izquierda del número binario de manera que haya un total de (***n – 1) bits***.
5. Si el número decimal a almacenar es positivo, se añade un ***0*** a la izquierda, y si el número es negativo, se añade un ***1*** a la izquierda, con lo cual se convierte en un número de ***n bits***.

Ejemplo 17:

*Almacenar el número **12** en una localidad de memoria de **8 bits*** utilizando la representación signo y magnitud.

Intervalo = – $(2^{8-1} – 1)$,…, $(2^{8-1} – 1)$ = – $(2^{7} - 1)$,…, $(2^{7} – 1)$ = –127,…, 127

Esto quiere decir que podemos almacenar los números enteros con signo y magnitud del ***–127 al 127***, por lo tanto, el número ***12*** está dentro del intervalo.

El equivalente en el sistema binario del número decimal ***12*** resulta ***1100***

El número de bits que ocupa ***12*** en binario son ***4 bits***, por lo que debemos agregar tres ceros a la izquierda para completar los ***7 bits*** asignados, quedando el número como ***0001100***

Ya que el número ***12*** es positivo debemos agregar un cero a la izquierda

El número listo para almacenar es ***00001100***

Ejemplo 18:

Almacenar el número –250 en una localidad de memoria de ***16 bits*** utilizando la representación signo y magnitud.

Intervalo = – (2^{16-1} – 1),…, (2^{16-1} – 1) = – (2^{15} - 1),…, (2^{15} – 1)

Intervalo = –32767,…, 32767

Esto quiere decir que podemos almacenar los números enteros con signo y magnitud del ***–32767 al 32767***, por lo tanto, el número ***–250*** está dentro del intervalo.

El equivalente en el sistema binario del número decimal ***250*** resulta ***11111010***

El número de bits que ocupa ***250*** en binario son ***8 bits***, por lo que debemos agregar siete ceros a la izquierda para completar los ***15 bits*** asignados, quedando el número como ***000000011111010***

Ya que el número – 250 es negativo debemos agregar un uno a la izquierda

El número listo para almacenar es ***1000000011111010***

- Para estudiar otros procedimientos de representación de números enteros con signo puede visualizar los siguientes videos:
  Complemento a 1: https://youtu.be/FigY6wvI6Ng
  Complemento a 2: https://youtu.be/xyGXtdJpiDg

- Puede visualizar el siguiente video para reforzar los conceptos y procedimientos dados sobre representación digital de números enteros: https://youtu.be/pSPuGiQ0vdE

Representación de Números Reales

Para representar números reales en un ordenador se puede utilizar el *método de punto fijo o coma fija*, o el *método de punto flotante o coma flotante*. Nos enfocaremos en el método de punto flotante.

Antes de estudiar el método de representación de punto flotante es preciso repasar la forma de expresar un número en *notación científica normalizada*.

Notación Científica Normalizada

En el sistema decimal, cualquier número real puede expresarse mediante la denominada *notación científica normalizada*. Para expresar un número en notación científica normalizada multiplicamos o dividimos por 10 tantas veces como sea necesario para que todos los dígitos aparezcan a la derecha del punto decimal y de modo que el primer dígito después del punto no sea cero. Por ejemplo:

$$\begin{aligned} 732.5051 &= 0.7325051 \times 10^{3} \\ -0.005612 &= -0.5612 \times 10^{-2} \end{aligned}$$

En general, un número real ***x*** distinto de cero, se representa en notación científica normalizada en la forma:

$x = \pm r \times 10^n$, donde ***r*** es un número tal que:

$\frac{1}{10} \leq r < 1$, y ***n*** es un entero (positivo, negativo o cero).

Exactamente del mismo modo podemos utilizar la notación científica en el sistema *binario*. En este caso, tenemos que:

$x = \pm q \times 2^m$, donde ***m*** es un entero.

El número ***q*** se denomina *mantisa* y el entero ***m*** *exponente*. En un ordenador binario tanto ***q*** como ***m*** estarán representados como números en base **2**. Puesto que la mantisa ***q*** está normalizada, en la representación binaria empleada se cumplirá que:

$$\frac{1}{2} \leq |q| < 1$$

Representación de Números en Punto Flotante

Para representar números reales se utiliza el método de *número de punto flotante*. Un número de punto flotante es un número que tiene una parte entera y una parte fraccionaria.

En un ordenador típico los números en punto flotante se representan en notación científica normalizada, pero con ciertas restricciones sobre el número de dígitos de ***q*** y ***m*** impuestas por la longitud de palabra disponible (es decir, el número de bits que se van a emplear para almacenar un número).

Para ilustrar este punto, consideraremos un ordenador que dispone de una longitud de palabra de **32 bits** (muy similar a la de muchos ordenadores

actuales). Para representar un número en punto flotante, los bits se acomodan del siguiente modo:

Signo del número real ***x***: ***1 bit***
Signo del exponente ***m***: ***1 bit***
Exponente (entero ***|m|***): ***7 bits***
Mantisa (número real ***|q|***): ***23 bits***

En la mayoría de los cálculos en punto flotante las mantisas se normalizan, es decir, se toman de forma que el bit más significativo (el primer bit) sea siempre '1'. Por lo tanto, la mantisa ***q*** cumple siempre la ecuación ***½ ≤ |q|<1***.

Dado que la mantisa siempre se representa normalizada, el primer bit en ***q*** es siempre **1**, por lo que no es necesario almacenarlo, proporcionando un bit significativo adicional. Esta forma de almacenar un número en punto flotante se conoce con el nombre de *técnica del 'bit fantasma'*.

Se dice que un número real expresado como aparece en notación científica normalizada y que satisface la ecuación ***½ ≤ |q|<1*** tiene la forma de *punto flotante normalizado*. Si además puede representarse exactamente con ***|m|*** ocupando ***7 bits*** y ***|q|*** ocupando ***24 bits***, entonces es un *número de máquina* en el ordenador de ***32 bits***.

La restricción de que ***|m|*** no requiera más de ***7 bits*** significa que:

$$|m| \leq (1111111)_2 = 2^7 - 1 = 127$$

Ejemplo 19:

Representar el número ***26,32*** *en una localidad de memoria de* ***32 bits*** *repartidos del siguiente modo: 1 bit para el signo, 15 bits para la parte entera y 16 bits para la parte fraccionaria,* utilizando la representación de punto flotante.

El número ***26,32*** en binario se escribe del siguiente modo: ***11010.01010001111010111000***

La representación en punto flotante se escribe del siguiente modo:

01000101 10100101000111101011100

- Para profundizar en la representación de números reales por los métodos de punto fijo y punto flotante, puede visualizar el siguiente video: https://youtu.be/EWWKFanXei0

- Para reforzar los conceptos y procedimientos dados sobre representación digital de números reales puede visualizar el video: https://youtu.be/L2YUAIWXlco

AUTOEVALUACIÓN UNIDAD III

1. Diga qué es un Microprocesador y dibuje un diagrama de bloques básico de la estructura del Microprocesador.
2. ¿Cuáles son las tres partes principales de la Unidad Central de Proceso (***CPU***) de un Microprocesador y cuál es la función de cada una de ellas?
3. Diga qué son los Registros del Microprocesador y cuál es su función
4. Diga cuales son y defina los tipos de registros del microprocesador.
5. ¿Cuáles son los tres buses principales de un microprocesador y cuál es la función de cada uno de ellos?
6. Diga cuál es la función de la Unidad Lógico-Aritmética (***ALU***)
7. Explique cómo funciona una memoria ***RAM*** y cuál es la diferencia entre una RAM dinámica y una RAM estática.
8. Explique cómo funciona una memoria ***ROM*** y cuáles son las variantes de este tipo de memoria.
9. Si se dispone de ***16 bits*** para el almacenamiento de un entero sin signo, ¿cuál es el intervalo de números que se pueden almacenar?
10. Representar el número ***180*** para ser almacenado en una localidad de memoria de ***8 bits*** en formato sin signo.
11. Si se dispone de ***32 bits*** para el almacenamiento de un entero con signo y magnitud, ¿cuál es el intervalo de números que se pueden almacenar?
12. Representar el número ***25*** en una localidad de memoria de *8 bits* utilizando la representación signo y magnitud.
13. Representar el número ***–525*** en una localidad de memoria de ***16*** bits utilizando la representación signo y magnitud.

REFERENCIAS BIBLIOGRÁFICAS

Jarne, C., Araujo G., Bramajo, R., Canteros-Castelli, R., Felipe, C., Ortaneche, F., y Sánchez-Jovic, M. (2022), Fundamentos de los Circuitos Secuenciales, Instituto Nacional de Educación Tecnológica, Argentina.

Floyd, T. (2006), Fundamentos de Sistemas Digitales. 9ª Edición. Madrid, España. Pearson/Prentice Hall.

Stallings W. (2005), Organización y Arquitectura de Computadores. 7ª Edición. Madrid, España. Pearson/Prentice Hall.

Parra, L. (2012), Microprocesadores, Red Tercer Milenio S.C. México.

Noriega, S. (2008), Álgebra de Boole. Introducción a los Sistemas Lógicos y Digitales.

Huertas, M., Lógica y Álgebra de Boole, Universitat Oberta de Catalunya.

https://www.ejemplos.co/40-ejemplos-de-proposiciones-simples-y-compuestas/

https://www.incb.com.mx/index.php/curso-de-electronica/96-curso-de-electronica-electronica-digital/3419-curso-de-electronica-electronica-digital-parte-11-como-funcionan-los-registros-de-desplazamiento-shift-registers-cur6004s

https://www.oposinet.com/temario-de-informatica/temario-3-informatica/tema-10-representacin-interna-de-los-datos-2/

https://www.esic.edu/rethink/marketing-y-comunicacion/que-es-codificacion-datos-tipos-ejemplos-c

https://www.uacj.mx/CGTI/CDTE/JPM/Documents/IIT/Representacion/representacion-de-numeros.html

https://www.uv.es/diaz/mn/node11.html