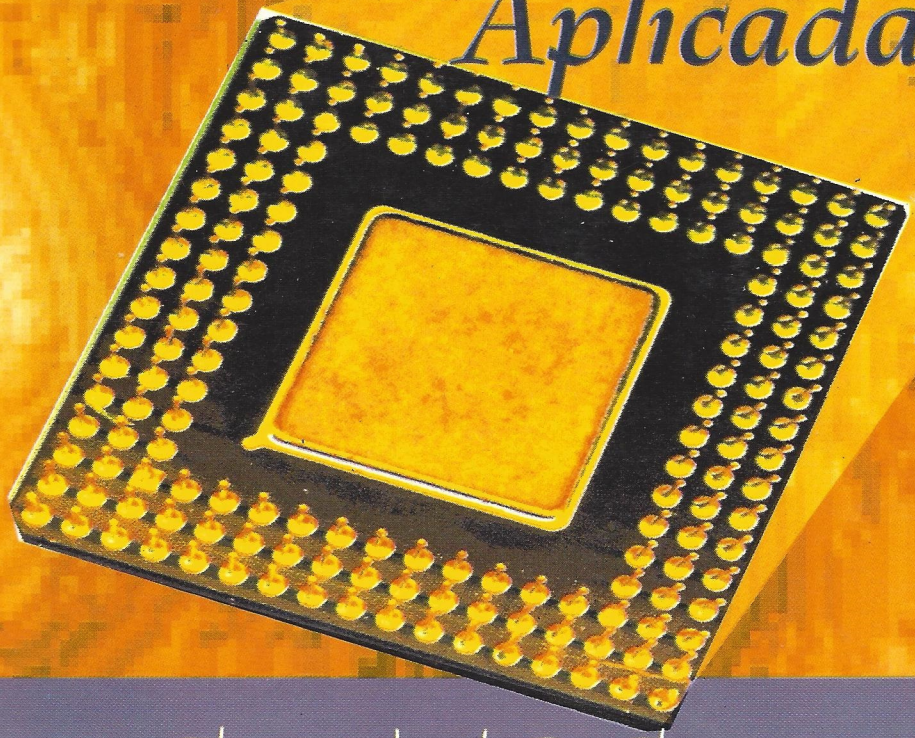
INTRODUCCION GENERAL A LA INFORMATICA 4 UNIDAD

Algebra de Boole

Representación de datos en el interior de una PC

Aplicada a



CIRCUITOS DE COMPUTACIÓN

Biblioteca Tecnica Superior

2ª edicion

M.C.GINZBURG

ALGEBRA DE BOOLE APLICADA

A CIRCUITOS DE COMPUTACION y REPRESENTACION DE NUMEROS EN EL INTERIOR DE UN COMPUTADOR

MARIO CARLOS GINZBURG

INGENIERO ELECTRONICO (UBA)

DIRECTOR DE LA CATEDRA DE "ESTRUCTURA DEL COMPUTADOR" E INVESTIGADOR EN LA FACULTAD DE INGENIERIA DE LA UNIVERSIDAD DE BUENOS AIRES

A Piyi, Jerónimo y Rafael

A mis alumnos

INTRODUCCION GENERAL A LA INFORMATICA

4 Algebra de Boole Aplicada a Circuitos de Computación y Codificación de Números en el Computador

M. C. GINZBURG

Es propiedad – Queda hecho el depósito que marca la ley
Impreso en la Argentina – Printed In Argentina

I.S.B.N. 950-43-0079-5

PRENSA & ACABADO	Concepción Arenal 4866
Abril del 2001	Ciudad de Buenos Aires

FORMALIZACIÓN DEL ALGEBRA DE BOOLE

A.0 Introducción

George Boole (1815-1864), fue el creador de un sistema algebraico para el estudio sistemático de la lógica, que hoy día se utiliza en áreas tales como las técnicas digitales, la programación, el álgebra de conjuntos y la teoría de las probabilidades.[1]

El álgebra de Boole constituye una formalización apropiada para representar información digital, y permite ***expresar algebraicamente las operaciones lógicas que realizan los circuitos digitales y determinar su respuesta, así como simbolizar el conexionado de los dispositivos lógicos que los constituyen,*** independientemente de la tecnología empleada en su construcción (llaves, relés, diodos, transistores, circuitos integrados, etc).

Entre las aplicaciones de este álgebra, pueden contarse:

- Formulación algebraica de los requerimientos que debe cumplir un cierto circuito lógico, estipulados por escrito o mediante tablas de funcionamiento.
- Análisis y síntesis de circuitos digitales combinacionales.
- Comparación de distintas realizaciones circuitales.
- Minimización del número de cables y compuertas de los circuitos.
- Codificación de información digital.

A.1 El álgebra de Boole

El álgebra de Boole consiste en un conjunto de ***axiomas (postulados)***, o sea premisas que se aceptan como base del sistema, y que no se deducen de otras. Dicho sistema de axiomas **indica propiedades** y **relaciones** entre ciertos elementos, **sin importar qué son realmente los mismos.** De esta manera, por medio de los axiomas, se definen implícitamente los términos que deben a su vez ser definidos.

Así, en Geometría, Euclides definió "recta"como una "longitud sin anchura", pero "longitud" y "anchura" a su vez necesitan ser definidos con precisión. Para evitar un proceso sin fin, se definen "punto" y "recta" como elementos que satisfacen una serie de **propiedades**, explicitadas en axiomas. A partir de éstos, se deducen las restantes propiedades del sistema o ***"teoremas"***.

La elección de los axiomas que constituyen un *"sistema axiomático"* es arbitraria dentro de ciertos límites, pero está sujeta a diversas exigencias. Una de las más importantes es la *"consistencia"*: no debe poder deducirse de los axiomas dos teoremas mutuamente contradictorios. Asimismo, es conveniente, por simplicidad, que los axiomas sean *independientes*: ninguno de ellos debe poder ser deducido de los restantes, para evitar axiomas innecesarios.

> Cualquier conjunto de elementos que verifique los axiomas constituye un **"modelo"** del sistema, y para el mismo valdran los teoremas demostrados a partir de los axiomas.

Este método es también útil para revelar y anticipar semejanzas estructurales entre sistemas físicos aparentemente disímiles

A.2 Postulados de Huntington

Existen varias formas de establecer cuáles son los axiomas del álgebra de Boole, resultan distintos los teoremas que resultan en cada sistema. Una de ellas se debe a Huntington, que en 1904 formalizó los siguientes axiomas;

---

[1] **Boole** reformuló con una simbología algebraica la notación de la lógica aristotélica, creando las bases para la lógica moderna. Posteriores desarrollos de esta idea han permitido crear un poderoso instrumento para el análisis y la deducción. De modo semejante, el sistema arábigo ha permitido agilizar y mecanizar cálculos que en el sistema romano de numeración requerían tiempo y especialistas

Se puede constituir un álgebra de Boole con un conjunto de elementos **C** = {a,b,c,d,.....}, *entre los cuales es posible definir dos operaciones binarias*[1] , *que simbolizaremos* Δ *y* $\square$, *si y sólo si se verifica:*

P_1: **Las operaciones** Δ *y* $\square$ **son conmutativas:** o sea que para elementos cualesquiera a y b de **C**, debe ser

$$a \Delta b = b \Delta a$$
$$a \square b = b \square a$$

P_2: **Cada operación es distributiva respecto a la otra**: esto es, para elementos cualesquiera a, b, c de **C** debe ser

$$a \Delta (b \square c) = (a \Delta b) \square (a \Delta b)$$
$$a \square (b \Delta c) = (a \square b) \Delta (a \square b)$$

P_3: **Existen en C dos elementos distintos que ofician de elementos** *"identidad"* **de las operaciones definidas**, que simbolizaremos e_Δ y $e_\square$ tales que:

$$a \Delta e_\Delta = a$$
$$a \square e_\square = a$$

Vale decir que se debe tener **C** = {a,b,c,d, ... e_Δ....., $e_\square$ .....}

P_4: **Para cada elemento** a **de C debe existir otro que oficie de** a' **tal que:**

$$a \Delta a' = e_\square$$
$$a \square a'' = e_\Delta$$

Vale decir que se debe tener **C** = {a,b,c,d,..... e_Δ ....., $e_\square$ ..... a' ....b' ... c' ... }

El símbolo "=" usado presupone que existen relaciones de equivalencia entre los elementos de dicho conjunto.
Puede demostrarse que el elemento **a'** asociado con el elemento **a** es único.

A.3 Conjuntos que verifican los postulados

A.3.1 Conjunto de los enteros divisores de 70

Sea el conjunto de números enteros **C** = {1, 2, 5, 7, 10, 14, 35, 70} a los que se aplica las operaciones binarias "máximo común divisor" (**M**) y "minimo común múltiplo" (**m**) Así: 2 **M** 7 = 1 2 **m** 7 = 14

esto es, el M.C.D entre 2 y 7 es 1, y el m.c.m entre 2 y 7 es 14, siendo también los resultados 1 y 14 elementos del conjunto **C**.

Verificación de **P_1:** 5 **M** 4 = 1 = 4 **M** 5 5 **m** 14 = 70 = 14 **m** 5, etc.

Verificación de **P_2:** por ejemplo, para tres elementos cualesquiera de **C**, como 2, 7 y 10 debe ser :

Comprobación:

2 **M** (7 **m** 10) = (2 **M** 7) **m** (2 **M** 10) 2 **m** (7 **M** 10) = (2 **m** 7) **M** (2 **m** 10)

2 **M** (7 **m** 10) = 2 **M** 70 = 2 (2 **M** 7) **m** (2 **M** 10) = 1 **m** 2 = 2

2 **m** (7 **M** 10) = 2 **m** 1 = 2 (2 **m** 7) **M** (2 **m** 10) = 14 **M** 10 = 2

Verificación de **P_3:** en el conjunto dado resulta

1 **M** 70 = 1 1 **m** 1 = 1
2 **M** 70 = 2 2 **m** 1 = 2
10 **M** 70 = 10 10 **m** 1 = 10 , etc....

Asimilando las operaciones **M** y **m** a las genéricas Δ *y* $\square$ de los axiomas de Huntington, resulta que el conjunto dado presenta los elementos 70 = e_Δ y 1 = $e_\square$:

Verificación de **P_4:** en el conjunto en cuestión encontramos que

2 **M** 35 = 1 7 **M** 10 = 1
2 **m** 35 = 70 luego: 35 = 2' y 35' = 2 7 **m** 10 = 70 luego 10 = 7' y 10' = 7 etc...

[1] O sea que dos operandos **a** y **b** tomados ordenadamente de un cierto conjunto, dan siempre por resultado un mismo elemento **p**, también de dicho conjunto: **a** "operación" **b** = **p**

En consecuencia, el modelo propuesto verifica los cuatro postulados de Huntington, y gozará de las propiedades dadas por los teoremas de la sección A.4

A.3.2 Algebra de Conjuntos

Cuando los elementos en cuestión son conjuntos, sus operaciones dan lugar a la denominada **"algebra de conjuntos"** que además de verificar los postulados de Huntington, puede englobar, a cualquier otro modelo que satisfaga dichos axiomas.

En el álgebra de conjuntos son útiles, para verificar las operaciones entre conjuntos, los *"diagramas de Venn"*.

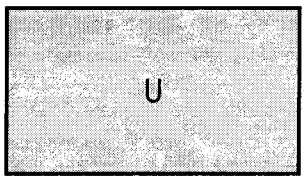


Figura C3.1

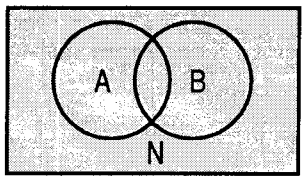


Figura C3.2

Supóngase un censo de todos los jóvenes de una población acerca de las carreras que seguirán. Este conjunto en análisis, se denomina **"conjunto universal"** (**U**), lo mismo que cualquier otro conjunto motivo de algún estudio concreto.

Este conjunto suele simbolizarse mediante los puntos interiores de un rectángulo (figura C3.1), en cuyo interior pueden representarse a su vez los subconjuntos que comprende (figura C3.2)

Los círculos **A** y **B** pueden corresponder respectivamente, al subconjunto de jóvenes anotados en las carreras que designaremos con las mismas letras. El subconjunto **N** constituído por los puntos exteriores a los dos círculos representa a los jóvenes que no siguen ninguna de las carreras **A** o **B**.

Operaciones entre conjuntos:

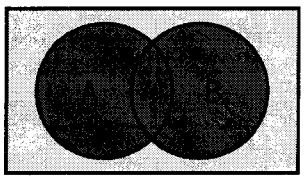


Figura C3.3

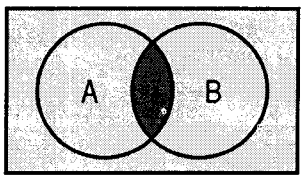


Figura C3.4

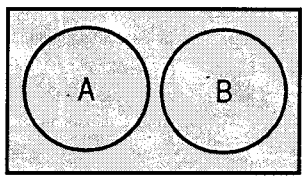


Figura C3.5

Operación unión: dados dos conjuntos **A** y **B**, es posible formar otro subconjunto **C** que contenga a todos los elementos de **A** y a todos los de **B**. Esta operación se simboliza $\mathbf{Z = A \vee B}$

Como indica el rayado de la figura C3.3, los elementos de **Z** pertenecen al conjunto **A**, o al **B** o a ambos simultáneamente. Siguiendo con la aplicación de la figura C3.2 correspoinde a los jóvenes que siguen la carrera **A** o la **B**.

Operación intersección: dados dos conjuntos **A** y **B**, se puede formar otro conjunto **Z** con todos los elementos que pertenecen a la vez a **A** y a **B**. La operación se simboliza: $\mathbf{Z = A \wedge B}$

La figura C3.4 muestra en rayado, para el ejemplo anterior, el conjunto de los jóvenes que siguen simultáneamente la carrera **A** y la carrera **B**. En particular, si los subconjuntos **A** y **B** no tienen ningún elemento en común, su intersección da como resultado un conjunto sin ningún elemento, esto es, un **"conjunto vacio"**, que se simboliza ϕ, como el resultado de la intersección de **A** y **B** en la figura C3.5. Este subconjunto podría darse si no existen jóvenes que sigan ambas carreras simultáneamente.

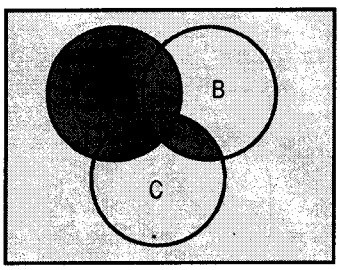
Verificación de los postulados de Huntington:

P_1)

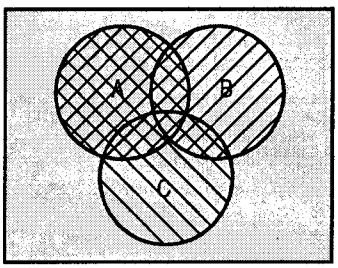
$$\mathbf{A \vee B = B \vee A}$$
$$\mathbf{A \wedge B = B \wedge A}$$

De acuerdo con la definición de cada operación, se verifican las equivalencias escritas, puesto que los conjuntos involucrados contienen los mismos elementos.

P_2)



$A \vee (B \wedge C)$



$(A \vee B) \wedge (A \vee C)$

La igualdad de las áreas sombreadas resultantes verifica una de las propiedades distributivas.

De manera análoga puede verificarse la otra.

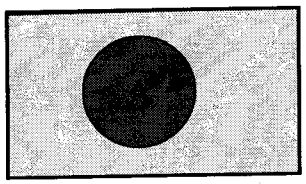


Figura C3.7

P_3)
Dado que ϕ indica cualquier conjunto sin elementos, su unión con cualquier conjunto da por resultado este mismo conjunto; y la intersección del conjunto universal con cualquier conjunto es este último conjunto (figura C3.7). Por consiguiente, es posible asimilar los conjuntos vacío y universal a los elementos e_Δ y $e_\square$: del álgebra de Boole.

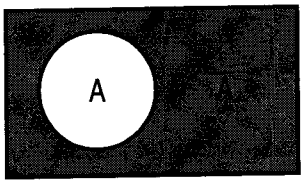


Figura C3.8

P_4)
Conforme se verifica en la figura C3.8, para un subconjunto **A** puede hallarse otro $\overline{A}$ que constituirá su complemento, y que contiene aquellos elementos del conjunto universal que no pertenecen al subconjunto **A:**

$$A \vee \overline{A} = U$$

Asimismo: $$A \wedge \overline{A} = \phi$$

De la verificación efectuada, resulta que el álgebra de Boole es un álgebra de Boole.

A.3.3 Algebra proposicional

En este caso los elementos son las **"proposiciones"**, esto es, enunciados declarativos que afirman o niegan algo en forma categórica, y que tienen la propiedad de ser verdaderos o falsos.

Cualquier proposición puede simbolizarse mediante una *"variable proposicional"* (**p, q, r, ..., z**) que sólo puede tener dos "valores de verdad: *verdadero* (**V**) ó *falso* (**F**).

Sea el enunciado **p** : "la tecla de sumar está apretada" Será **p** verdadero (**p=V**) si se cumple que la tecla está apretada, o será **p** falso (**p=F**), si no se cumple que la tecla está apretada.

Igualmente, el enunciado "llueve", es verdadero si realmente ello ocurre. En cambio frases como "cierre la llave", o "¿qué hora es ?", carecen de valor informativo; no tienen la posibilidad de ser verdaderas o falsas, por lo que no son proposiciones.

Operaciones entre proposiciones:

Dos proposiciones cualesquiera **p** y **q** pueden combinarse o conectarse de varias maneras para formar nuevas proposiciones. Así los enunciados:

p : "el seguro está activado"
q : "el botón está oprimido"

pueden combinarse mediante la conjunción "**y**" para formar la proposición compuesta "el seguro está activado y el botón está oprimido", que será verdadera o falsa según lo sean **p** y **q**.

También es posible combinar **p** y **q** intercalando partículas o palabras tales como "o", "no", "si y solo si", "si... entonces..." formando otras tantas proposiciones compuestas, cuyo valor de verdad depende exclusivamente del valor de verdad de las mencionadas **p** y **q**. Esto último no ocurre si se utilizan otras conectivas del tipo "porque", "a cause de", etc.

Desde el punto de vista lógico sólo interesan las del primer grupo, llamadas *"funciones de verdad"*. Se definen a continuación las tres operaciones básicas: conjunción, disyunción y negación; las restantes pueden definirse a partir de ellas.

Disyunción: la combinación de dos proposiciones **p** y **q** por medio de la palabra "**o**" da por resultado otra proposición denominada "disyunción de **p** y **q**", que será verdadera si **p** o **q** o ambas son verdaderas, y que será falsa únicamente, cuando **p** y **q** sean falsas. Esta operación se simbolizará (**p + q**) siendo (+) el operador "o".

p	q	p+q	
F	F	F	→ F+ F=F
F	V	V	→ F+V=V
V	F	V	→ V+F=V
V	V	V	→ V+V=V

Designando:
p : "ese hombre es el tío de Pedro"
q : "ese hombre es el padrino de Pedro"
la proposición compuesta "ese hombre es el tío o el padrino de Pedro" puede ser V o F según lo sean **p** y **q** conforme la tabla de la izquierda

Conjunción: la combinación de dos proposiciones **p** y **q** por medio de la palabra "**y**" da por resultado otra proposición denominada "conjunción de **p** y **q**" que será cierta si y sólo **p** y **q** son ciertas, y falsa en el caso que una o ambas sean falsas.
La conjunción se simboliza (**p . q**) El operador (.) se denomina "y".

p	q	p.q	
F	F	F	→ F. F=F
F	V	F	→ F.V=F
V	F	F	→ V.F=F
V	V	V	→ V.V=V

Así, el enunciado "ese hombre es el tío de Pedro y el padrino de Pedro" será cierto, sólo en el caso que realmente sea cierto que es el tío y sea cierto que es el padrino, en correspondencia con la tabla de la izquierda, que contempla todas las posibilidades.

Negación: dada una proposición **p**, mediante el agregado de la palabra "**no**" se obtiene otra proposición que es su "negación", simbolizada **p'**, tal que cuando **p** sea cierta, **p'** sea falsa, y viceversa
Designando:

p : "el motor funciona" ; **q** : "el motor no funciona",

si el motor funciona será **p** = V, a la par que **p'** = F, siendo **p** = F y **p'** = V si el motor no funciona.

Tautologías y contradicciones:

Existen aserciones compuestas que por su forma son siempre verdaderas, independientemente de la verdad o falsedad de las expresiones que la componen, como ser: "la lámpara está encendida o no está encendida"

Tales proposiciones denominadas *"tautologías"* pueden reducirse a la forma **p + p'** y no proporcionan información alguna.

p q	p + p'	p . p'
V F	V + F = V	V . F = F
F V	F + V = V	F . V = F

Las proposiciones denominadas *"contradicciones"*, reducibles a la forma **p . p'** son formalmente falsas, y excluyen de antemano toda posibilidad de veracidad.
Son del tipo "llueve o no llueve".

Verificación de los postulados de Huntington

Dos proposiciones son lógicamente equivalentes cuando los mismos hechos que hacen verdadera o falsa a una de ellas, hacen también verdadera o falsa, respectivamente, a la otra, lo que se traduce en que ambas presentan la misma tabla de verdad.

Conforme con esto, por medio de tablas de ver

$$p + q = q + p \qquad p \,.\, q = q \,.\, p \qquad (\mathbf{P_1})$$

$$p + (q \,.\, r) = (p + q).(p + r) \qquad p \,.\, (q + r) = (p \,.\, q) + (p \,.\, r) \qquad (\mathbf{P_2})$$

Verificación de $\mathbf{P_3}$ y $\mathbf{P_4}$

De acuerdo con las tablas de la disyunción y conjunción, resulta:

p	p + F	p . V
F	F + F = F	F . V = F
V	V + F = V	V . V = V

o sea: **p + F = p** y **p . V = p** $(\mathbf{P_3})$

Por lo tanto, la disyunción o la conjunción de una proposición lógica con otra que siempre es, respectivamente, falsa o verdadera, no afecta el valor de verdad de la primera.

Como una proposición siempre falsa es una "contradicción", y otra siempre verdadera es una "tautología", la primera puede definirse como el elemento e_Δ , y la segunda como el elemento $e_\square$, respectivamente, del álgebra de Boole.

Asimismo, conforme con la negación, disyunción y conjunción

p	p'	p + p'	p . p'
F	V	F + V = V	F . V = F
V	F	V + F = V	V . F = F

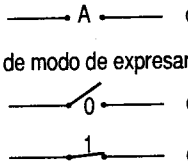
o sea: **p + p' = V** y **p . p' = F** $(\mathbf{P_4})$

En consecuencia, se verifica que para cada proposición, su negación en disyunción o conjunción, da como resultado una tautología (elemento $e_\square$) o una contradicción (elemento e_Δ), respectivamente.

A.3.4 Conjunto de contactos tipo "si-no"

Un contacto, tal como una llave común de luz eléctrica, o un timbre, presenta dos estados o posiciones claramente definiO. das *"cerrado"* o *"abierto"* (figura A.9). Las mismas también pueden identificarse con los símbolos **1** y **0**, respectivamente. Si bien éstos coinciden con sus similares de los sistemas numéricos, su significado es diferente, dado que implican un significado cualitativo: *"todo-nada"* o *"si-no"*, respectivamente.

Por la relación originaria de los símbolos **1** y **0** con los valores *"verdadero"* y *"falso"* de la lógica clásica, también se denominan valores lógicos **1** y **0**.



De forma más abstracta, la existencia de un contacto eléctrico puede representarse por una letra mayúscula entre los puntos que conmuta:

Al escribir **A** = 0, el signo "igual" es usado en el sentido de afirmar que cuando el contacto está abierto, **A** "resulta" cero, o que **A** "vale" cero, o también que **A** tiene valor lógico cero.

Figura A.9

Operación de combinar dos contactos en paralelo

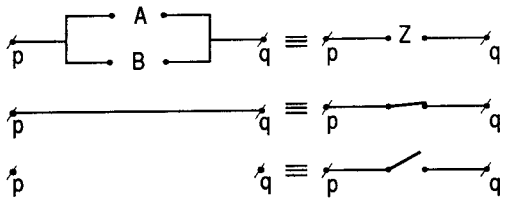


Figura A.10

El estado *"conductivo"* o *"no conductivo"* que producen entre los puntos **p** y **q** de la figura A.10, dos contactos en paralelo, **A** y **B**, se representa por un contacto equivalente **Z**.
Dichos contactos originan cuatro combinaciones de estados **"abierto"-"cerrado"**, que se muestran en la figura A.11, junto con el resultado de cada combinación.
El estado conductivo entre **p** y **q** tendrá lugar, cuando al menos uno de los contactos **A ó B** está cerrado. Resulta así formalmente enunciada una proposición que es la disyunción de otras dos. Asimismo, los valores lógicos relacionados con las combinaciones de contactos en paralelo y sus resultados (figura A.11) se corresponden con los casos de la operación lógica **A + B** que expresa la disyunción. (sección A.3.3). Esta última expresión representa por lo tanto dos contactos en paralelo, así como cualquier enunciado donde aparece la disyunción "o"

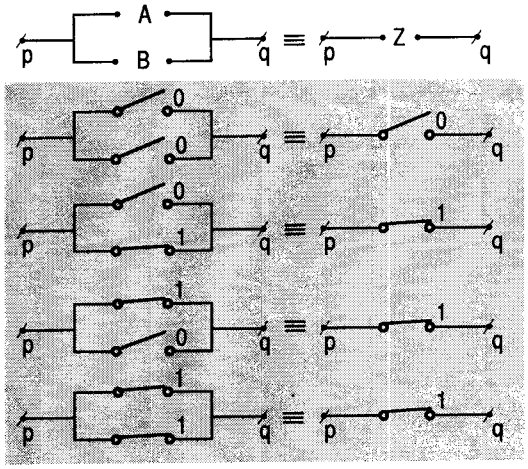


Figura A.11

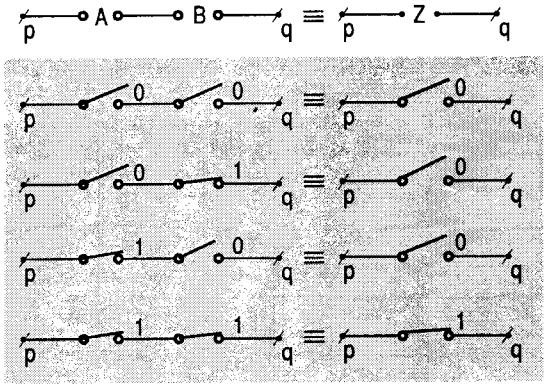


Figura A.12

Operación de combinar dos contactos en serie:

En la figura A.12 aparecen las cuatro combinaciones de estados de dos contactos en serie. y el contacto equivalente que resulta en cada situación.

Para esta disposición, existirá el estado de continuidad entre **p** y **q** sólo si ambos contactos están cerrados. Los valores lógicos que intervienen en las combinaciones dibujadas pueden hacerse corresponder con las cuatro posibilidades de efectuar el producto lógico de dos valores lógicos, definidos en la sección A.3.3 Así se verifica que el producto lógico de dos variables, que se expresa **A.B** representa la operación de dos contactos en serie.

Contactos en oposición o en operación complementaria:

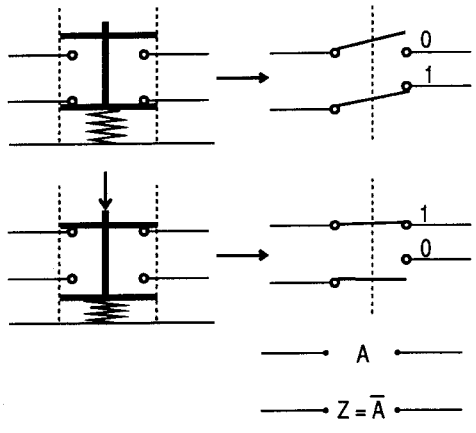


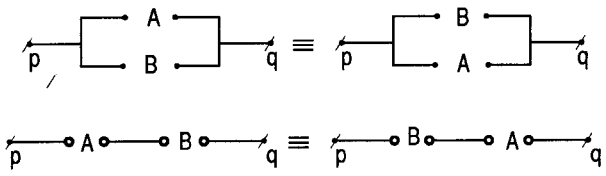
Figura A.13

Es posible construir dos contactos accionados mecánicamente al unísono, de modo que al mismo tiempo que uno está abierto, el otro está cerrado, y viceversa (figura A.13). Mecánicamente, por la posición de los resortes, el contacto superior es del tipo "normalmente abierto" y el inferior es "normalmente cerrado"
Como el contacto **Z** vale **1** cuando **A** no vale **1**, y **Z** vale **0** cuando **A** no vale **0**, puede enunciarse que **Z** es **no A**, o que es la **negación** de **A**, simbolizándose:
$\mathbf{Z} = \overline{\mathbf{A}}$.

De manera recíproca, **A** es la negación de **Z**, o sea $\mathbf{A} = \overline{\mathbf{Z}}$

Verificación de los postulados del álgebra de Boole:

P_1) : Las dos primeras operaciones definidas son conmutativas.



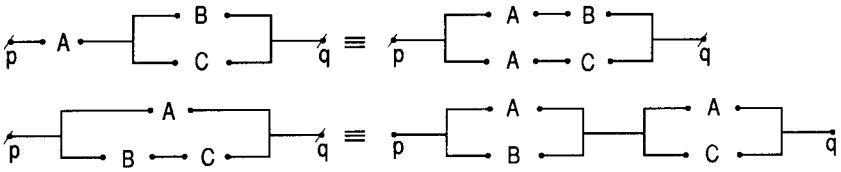
Formalmente deber ser : **A + B = B + A** y **A . B = B. A** lo cual implica que se debe verificar las equivalencias entre los conexionados dibujados. Es evidente que la continuidad o no entre los puntos **p** y **q** en contactos en paralelo no depende de cuál contacto está arriba o abajo; y en contactos en serie no interesa el orden de conexionado, o sea cuál contacto está primero y cuál segundo

En consecuencia se cumplen las expresiones indicadas.

P_2): Cada una de las operaciones es distributiva respecto de la otra:

Formalmente debe ser: **A . (B + C) = A.B + A.C** y **A + (B.C) = (A + B) . (A + C)**

por lo que se debe probar las siguientes equivalencias entre contactos combinados:



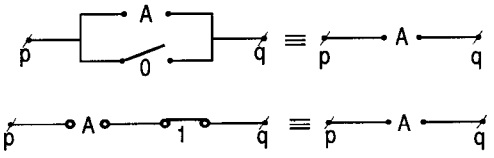
Una forma de verificar esto es dibujar todas las combinaciones posibles de estas dos en que pueden estar los contactos: uno abierto y dos cerrados, dos abiertos y uno cerrado, etc. Para 3 contactos, **A, B, C**, habría que dibujar $2^3 = 8$ situaciones distintas, y comprobar en cada caso que entre **p** y **q** se tenga el mismo estado conductivo en ambas disposiciones de contactos que se están verificando.

No es necesario realizar estos 8 dibujos. Puesto que las operaciones suma y producto lógico expresan contactos en paralelo y los valores **0** y **1** de las variables a estados abierto o cerrados de los contactos, el resultado **0** ó **1** de la expresión que simboliza el conexionado de contactos en cuestión permite calcular cuál será el estado (abierto o cerrado) del contacto equivalente, sin realizar ningún dibujo. Esto es, mediante tablas de verdad, como se hace en la sección A.14 para verificar este postulado para compuertas, puede comprobarse mediante cálculos lógicos las equivalencias dibujadas. Si para cada combinación de valores de variables **A, B, C** (de 000 a 111) el contacto equivalente entre **p** y **q** está en el mismo estado, los conexionados serán equivalentes.

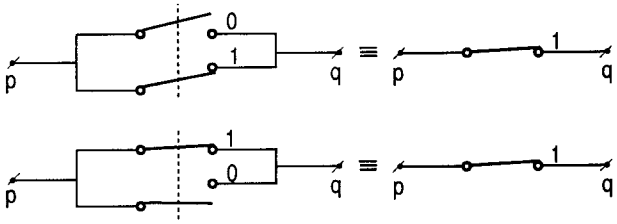
P_3) : Verificaremos que la suma lógica es invariante respecto del **0**, y el producto lógico respecto del **1**.

O sea **A + 0 = A** **A . 1 = A**

Una variable **A** representa un contacto que puede estar abierto (**0**) o cerrado (**1**). Si en una expresión booleana aparece escrito un **0** (ó un **1**) en lugar de una variable, dicho valor constante representará un contacto que por algún motivo está *siempre* abierto (o siempre cerrado). La figura siguiente pone en evidencia que un contacto **A** en paralelo con otro **siempre** abierto (0) no afecta la acción del primero, verificándose que **A + 0 = A**. Lo mismo un contacto en serie con otro **siempre** cerrado (**1**)



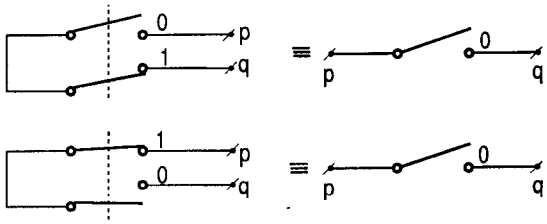
Resulta así que un contacto siempre abierto (**0**) ó cerrado (**1**) puede asimilarse a los elementos e_Δ y e_∇ de los postulados de Huntington.



P_4) :
La suma y el producto lógicos de una variable y su negación, deben resultar siempre **1** y **0**, respectivamente

O sea: $A + \overline{A} = 1$ y $A \cdot \overline{A} = 0$

Verificación mediante contactos: conforme la figura de la derecha



De esta forma hemos verificado que un conjunto constituído por contactos del tipo "si-no", mediante los cuales pueden definirse las operaciones de combinarlos en paralelo, en serie y en oposición, satisface los postulados de Huntington, por lo que podrán ser tratados algebraicamente empleando el álgebra de Boole, así como también se les podrá aplicar las propiedades (teoremas) de este álgebra (sección A.4)

A.3.5 Conjunto de cables con dos valores de tensión eléctrica

Un conjunto de elementos constituído por cada uno de los **cables de un circuito** eléctrico que presentan uno de dos valores o niveles específicados de tensión eléctrica, (asimilables al **1** y **0** lógicos) conectados a dispositivos llamados **compuertas**, que pueden realizar operaciones lógicas con dichos valores de tensión, verifican los 4 postulados de Huntington.

Tales cables y compuertas constituyen un *circuito lógico*.

Conforme se tratará en detalle, las compuertas **OR, AND** e inversor realizan operaciones lógicas.

En la sección A.14 se verifican los 4 postulados, resultando que un cable siempre conectado a masa (**0**) no afecta el resultado de la operación que realiza una **OR**, por lo que puede asimilarse al elemento e_Δ A su vez, un cable siempre conectado a la tensión de alimentación (**1**) no afecta el resultado de una **AND**, pudiéndose asimilar al elemento $e_\square$.

Asimismo, en la salida de una compuerta inversora se puede obtener un cable cuyo estado lógico sea complementario del que está en su entrada, pudiéndose verificar $\mathbf{P_4}$ mediante tablas de verdad.

Por lo tanto a este conjunto particular se le podrán aplicar las propiedades del álgebra de Boole.

A.4 Principales teoremas que se deducen de los postulados

Habiendo tratado varios conjuntos de elementos que verifican los 4 postulados de Huntington, se demostrarán los teoremas más importantes que se deducen de ellos. Estos teoremas o propiedades pueden aplicarse a dichos conjuntos o a cualquier otro conjunto con el que se pueda definir un álgebra de Boole.

1) $a + 1 = 1$ — $a \,.\, 0 = 0$ (dual)

$1 = a + a' = a + a' \,.\, 1 = (a + a') \,.\, (a + 1) = 1 \,.\, (a + 1) = a + 1$
por $\mathbf{P_4}$ por $\mathbf{P_3}$ por $\mathbf{P_4}$ por $\mathbf{P_2}$ por $\mathbf{P_4}$

T_2) $a + a = a$ — $a \,.\, a = a$ (dual)

$a = a + 0 = a + a \,.\, a' = (a + a) \,.\, (a + a') = (a + a) \,.\, 1 = a + a$
por $\mathbf{P_3}$ por $\mathbf{P_4}$ por $\mathbf{P_2}$ por $\mathbf{P_4}$ por $\mathbf{P_3}$

T_3) $a + a \,.\, b = a$ — $a \,.\, (a + b) = a$ (dual)

$a = 1 \,.\, a = (1 + b) \,.\, a = 1 \,.\, a + b \,.\, a = a + a \,.\, b$
por $\mathbf{P_3}$ por $\mathbf{T_1}$ por $\mathbf{P_2}$ por $\mathbf{P_3}$ y $\mathbf{P_1}$

T_4) $a + (b + c) = (a + b) + c$ — $a \,.\, (b \,.\, c) = (a \,.\, b) \,.\, c$ (dual)

T_5) $(a + b)' = a' \,.\, b'$ — $(a \,.\, b)' = a' + b'$ (dual)

$(a + b) + a' \,.\, b' = [(a + b) + a'] \,.\, [(a + b) + b'] = (1 + b) \,.\, (1 + a) = 1$
por $\mathbf{P_2}$ por $\mathbf{T_4}$, $\mathbf{P_4}$ y $\mathbf{T_1}$

$(a + b) \,.\, (a' \,.\, b') = a \,.\, a' \,.\, b' + b \,.\, a' \,.\, b' = 0 \,.\, b' + a' \,.\, 0 = 0$
por $\mathbf{P_2}$ por $\mathbf{P_1}$, $\mathbf{P_4}$ y $\mathbf{T_1}$

Los resultados de las sumas y productos efectuados, de acuerdo con $\mathbf{P_4}$ implican que $a' \,.\, b' = (a + b)$

T_5) $(a')' = a$

por $\mathbf{P_4}$ $a' + (a')' = 1$ y $a' \,.\, (a')' = 0$

por $\mathbf{P_1}$ y $\mathbf{P_4}$ $a' + a = 1$ y $a' \,.\, a = 0$

Puesto que en ambos casos **(a')'** y **a** resultan complementarios con **a'**, y dado que puede demostrarse que cada elemento **a'** asociado con un elemento **a** es único, se demuestra el teorema.

ALGEBRA DE BOOLE EN CIRCUITOS DE COMPUTADORES

A1.1 Cables con dos niveles de tensión

En los circuitos digitales o lógicos, la información digital (figura A1.1) es portada y transmitida por ***los cables,*** elementos conductores de la electricidad. Estos en los circuitos digitales binarios pueden presentar ***sólo dos niveles de tensión*** perfectamente diferenciados: ***alto (High = H)*** o ***bajo (Low = L)***

Por ejemplo, si los circuitos están alimentados con una tensión de 5 volts, un cable cualquiera de ellos, en un instante dado puede estar en un nivel ***L*** de 0 volts, o en un nivel ***H*** de 5 volts, ***sin posibilidad de presentar valores intermedios de tensión*** (salvo cuando cambia de un nivel a otro) en condiciones normales de funcionamiento.

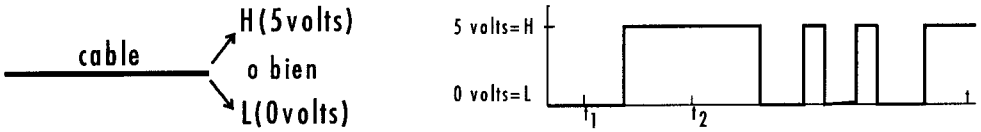
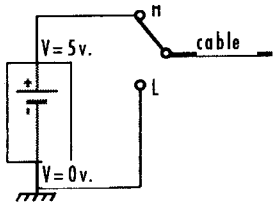


Figura A1.1

A lo largo del tiempo, los niveles de tensión de un cable pueden fluctuar arbitrariamente como ejemplifica la figura A1.1. Asi, en el instante t_1 estaría en el nivel bajo, en t_2 en el nivel alto. Eléctricamente no es posible que en un cable, en un cierto instante, puedan superponerse el nivel alto con el bajo.

Debe consignarse que en lugar de hablarse de dos niveles fijos, por razones de tolerancia en la generación y reconocimiento de los niveles lógicos, en realidad cuando un cable presenta un nivel alto, puede estar en ***una*** tensión cualquiera de la *gama* de tensiones establecida como nivel alto. Lo mismo respecto del nivel bajo.

Conmutación del nivel de un cable mediante una llave



Una forma de cambiar el estado eléctrico de un cable para que pase de un nivel de tensión al otro se indica en la figura A1.2. Suponiendo una batería o fuente de 5 volts, si se considera su terminal negativo (-) como tensión de referencia de 0 volts (o sea masa), el terminal positivo (+) estará a 5 volts respecto del terminal negativo tomado como referencia.Mediante una llave de dos posiciones como la dibujada sería factible seleccionar que el cable se encuentre a 0 volts (***L***) o en 5 volts (***H***)

⟵ Figura A1.2

A1.2 Variables lógicas o "booleanas"

Cuando se tuvo que determinar el efecto de un número grande de contactos combinados de una manera compleja, se planteó la necesidad de tratar el problema mediante símbolos y operaciones algebraicas que representen el conexionado.

Los contactos presentan dos estados: *"cerrado-abierto", "si-no" , "1 ó 0"* , como se los quiera denominar. Las tablas que indican el efecto de conectar contactos en paralelo o en serie según el estado cerrado o abierto de los contactos componentes. tienen la misma estructura formal que las correspondientes a las operaciones disyunción y conjunción de la lógica simbólica, que opera con los valores lógicos *"verdadero-falso" (V ó F).*

Esta analogía formal permitió ***aplicar el simbolismo algebraico usado en el cálculo lógico***, cuyo pionero fue G. Boole, ***para resolver el problema de los contactos***, como hizo Shannon en la década del 40 en los EEUU.

Con el advenimiento de los circuitos digitales electrónicos, cuyos cables sólo presentan un nivel de tensión alto o bajo, y con compuertas cuyas tablas de funcionamiento son formalmente semejantes a las tablas de verdad que definen a las operaciones lógicas, se siguió empleando la denominada "álgebra de Boole" para obtener expresiones simbólicas (denominadas *lógicas o booleanas*) que representen el comportamiento de tales circuitos.

Por tal motivo se habla de que al nivel de tensión alto de un cable se le asigna el "***valor lógico 1***", y al nivel bajo el "***valor lógico 0***", para indicar la correspondencia establecida con los valores *verdadero* y *falso* de la lógica:

Cable en el nivel ***L*** → **0**

Cable en el nivel ***H*** → **1**

Esta convención para asignar valores lógicos se denomina ***"logica positiva"***, y es la que se usa corrientemente.

Aunque los símbolos **0** y **1** coinciden con los similares de los sistemas numéricos, su significado aca es diferente, dado que tienen el sentido de *"falso-verdadero", "no-si"*.

No debe confundirse este significado particular que asumimos para los dos niveles de un cable (que usaremos para expresar algebraicamente el comportamiento lógico de un circuito digital binario y para calcular su respuesta) con las aplicaciones del mismo para representar información binaria. Así, puede convenirse que los valores *lógicos* uno y cero representen los correspondientes valores *numéricos* del sistema numérico binario. Por ejemplo, 8 cables pueden representar dos números de 4 bits cada uno. Si estos 8 bits entran a un circuito binario, puede hacerse que el comportamiento lógico del circuito sea tal que los valores lógicos de cuatro cables de salida sean tales que –interpretados como valores numéricos– constituyan los 4 bits del número que es el resultado de la suma de los dos números citados, como se trata en la sección A1.17

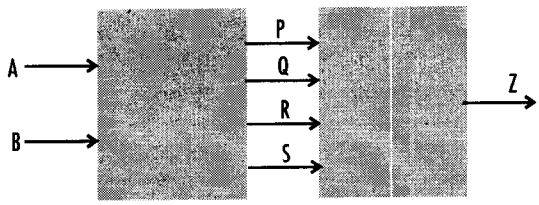


Figura A1.5

En forma más abstracta, cada cable de un circuito digital puede identificarse y representarse mediante letras mayúsculas, de **A** a **Z**. De esta forma los cables del circuito (figura A1.5) se identifican con las letras que los representan, y el nivel de tensión en que se encuentran mediante los símbolos **0** y **1**. Podemos simbolizar:

R = 0 si el cable R está en el nivel bajo;
R = 1 si el cable R está en el nivel alto;

y decir ***"R** vale cero"* y ***"R** vale uno"* respectivamente.

Puesto que el valor lógico que se le puede asignar a la letra **R** puede ser **1** ó **0**, se dice que **R** es una ***"variable lógica"*** cuyo valor puede *variar* de un valor lógico al otro.

En general, el término ***variable lógica, o "booleana"*** designa a cualquier símbolo literal **A, C, D, ..., Z** empleado para representar elementos o magnitudes físicas que exhiben dos estados posibles claramente definidos, correspondientes a los valores lógicos **1** y **0** que ella puede presentar.

Así también con **F**=1 y **F**=0 se puede indicar un foco encendido y apagado, respectivamente.

A1.3 Compuertas lógicas

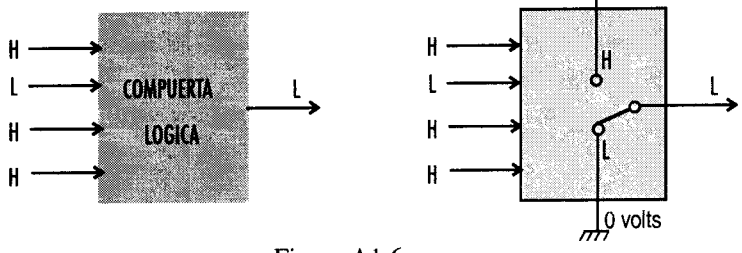


Figura A1.6

Anteriormente (sección A1.1) se vió que se puede conmutar el nivel de tensión de un cable si se conmuta el estado de una llave.

Mediante un dispositivo electrónico denominado ***"compuerta lógica"*** se puede conmutar el nivel de tensión del cable conectado a su salida (figura A1.6) cambiando adecuadamente la combinación de niveles de tensión existente en los cables que llegan a las entradas de dicha compuerta.

Todo sucede como si en lugar de la llave manual de la figura A1.2 se tuviera una llave electrónica ligada al cable de salida, que se conecta a 5 ó 0 volts según el nivel alto o bajo de los cables de entrada (figura A1.6)

La denominación "compuerta" se relaciona con el hecho de que este dispositivo puede usarse para permitir o no que el nivel alto o bajo de un cable que llega a una de sus entradas se repita en el cable que está en su salida. El nivel existente en una entrada aparece en el cable de salida, como si la compuerta "dejara pasar" una señal eléctrica de un cable a otro, como se ejemplifica en la sección A.8 Esto en cierta medida es semejante a lo que ocurre en un dique, en el cual una compuerta deja pasar o no un curso de agua de un lado a otro del mismo.

El aditamento *"lógica"* se refiere a que una compuerta en esencia realiza electrónicamente una operación lógica: para cada combinación de valores lógicos existentes en las entradas, resultará un valor lógico (1 ó 0) en su salida. Se puede representar el comportamiento o respuesta de una compuerta para todas las combinaciones de valores lógicos posibles que pueden darse en sus entradas mediante una *tabla de funcionamiento* o una *"tabla de verdad"*

A continuación definiremos el funcionamiento, el símbolo circuital empleado en los manuales de circuitos integrados y la operación lógica de las compuertas **Or**, **And**, e **Inversor**

A1.4 Compuerta OR.

Una ***compuerta OR*** de dos entradas (simbolizada en la figura A1.7) es un dispositivo electrónico que presenta dos entradas, a las cuales llegan los niveles de tensión de dos cables (**A** y **B**), y una salida. Esta genera en el cable (**Z**) un nivel que depende de los niveles existentes en las entradas de la forma indicada en la figura A1.8

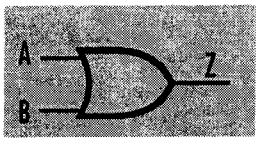


Figura A1.7

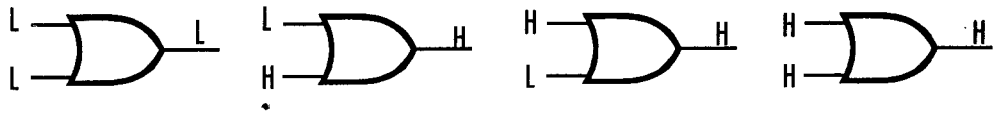


Figura A1.8

A	B	Z
L	L	L
L	H	H
H	L	H
H	H	H

El comportamiento eléctrico se resume en la *tabla de funcionamiento* de la figura A1.9, que representa la respuesta en la salida de la compuerta para todas las combinaciones posibles de niveles que pueden darse en las entradas

Figura A1.9

También podemos enunciar así:

"Un cable **Z** en la salida de una compuerta **OR** estará en el nivel alto si el cable de entrada **A** **o** el cable de entrada **B** está en el nivel alto, **o** si ambos están en el nivel alto." [1] Más brevemente:

> "Un cable **Z** en la salida de una compuerta **OR** de dos entradas estará en el nivel alto si el cable de entrada **A** o **B** está en el nivel alto"

O también: "*basta que una de las entradas* de una compuerta **OR** esté en el nivel alto, para que su salida también lo esté", enunciado que pone de relieve que existen varias alternativas para lograr el mismo resultado

Cualquier enunciado que sea del tipo de los enunciados anteriores, o que pueda llevarse a una forma semejante, responde a una tabla formalmente igual a la tabla de la figura A1.9

þ **EJEMPLO**

Sea el enunciado: "en un semáforo la luz roja ($\mathbf{R_1}$) para una calle "1" se debe encender si en la calle "2" está encendida la luz verde ($\mathbf{V_2}$) **o** la luz amarilla ($\mathbf{A_2}$), **o** si ambas están encendidas en el momento del cambio".

Asignando los valores **1** y **0** a los estados encendido y apagado, respectivamente, de cada luz, el estado que debe tener $\mathbf{R_1}$ en función de los estados de $\mathbf{V_2}$ y $\mathbf{A_2}$ se tabula en la figura A1.10

A_2	V_2	R_1
0	0	0
0	1	1
1	0	1
1	1	1

Figura A1.10

Comparando las tablas de las figuras A1.9 y A1.10 se deduce que si a las entradas de una compuerta **OR** (figura A1.11) llegan dos cables cuyos niveles H y L representan los valores lógicos **1** y **0** correspondientes a los estados "encendido-apagado" de las luces verde y amarilla de la calle "2", la salida de la compuerta generará un nivel de tensión que será el apropiado para comandar el encendido-apagado de la luz roja de la calle "1".

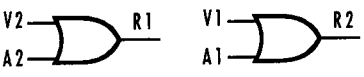


Figura A1.11

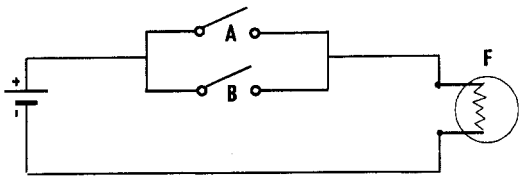


Figura A1.12

Análogamente, otra compuerta **OR** se usará para comandar la luz roja ($\mathbf{R_2}$) de la calle "2", siendo sus entradas $\mathbf{V_1}$ y $\mathbf{A_1}$ de la calle "1".

Del mismo modo, una especificación como "el motor se detendrá si el operario acerca su mano al sensor de zona peligrosa **o** si oprime el botón de parada", puede ser concretada mediante una compuerta **OR** que controlará al motor.

---

1 Implícitamente resulta que **Z** estará en el nivel bajo sólo si ambos cables de entrada están en el nivel bajo.

En los enunciados anteriores se ha puesto de relieve el nexo **o** (**Or** en inglés) que realiza la **disyunción** de enunciados y que da nombre a esta compuerta.

La estructura lógica común que presenta el comportamiento operativo de la compuerta OR y la conexión en paralelo de contactos, se pone de manifiesto en la similitud formal de las tablas [1] que se corresponden en cada una de esas áreas.

Al respecto, en el circuito de dos contactos **A** y **B** en paralelo (figura A1.12), el foco **F** se encenderá si **A o B** está cerrado.

Operación suma lógica

Reemplazando en la tabla de funcionamiento de la figura A.9 los símbolos **H** y **L** por los valores lógicos **1** y **0** , resulta *la tabla de verdad* de la figura A.13 correspondiente a una compuerta **OR** de dos entradas

La compuerta **OR** realiza una *operación,* dado que para cada par de valores lógicos **A** y **B** que combina, genera en el cable **Z** un valor lógico resultante.

A	B	Z	
0	0	0	→ **0 + 0 = 0**
0	1	1	→ **0 + 1 = 1**
1	0	1	→ **1 + 0 = 1**
1	1	1	→ **1 + 1 = 1**

Figura A1.13

Esta operación la simbolizaremos con el operador binario representado por el signo "más" (+) que indica esta operación definida por la tabla de verdad de la figura A1.13, la cual contempla todos los casos posibles

Se denomina ***"suma lógica"*** *u "operación Or"* , siendo que coincide formalmente hasta el tercer renglón de la tabla con la suma aritmética, aunque el significado de los valores lógicos **1** y **0** es distinto que el de los valores aritméticos 1 y 0, como se aclaró en A1.3

A la derecha de la figura A1.13 se ha escrito otra manera equivalente de expresar cada renglón de la tabla, mediante el simbolismo de una suma de valores lógicos.

Puesto que los valores lógicos de cada suma corresponden a las variables **A, B** y **Z**, en forma sintética podemos expresar:

Z = A + B (que se lee *"Z igual A o B")*

Representando cada cable por una variable que puede valer **0** ó **1** (sección A1.2), decimos que la salida de la compuerta **OR** genera el valor **0** ó **1** de la variable **Z**, que es igual al valor del resultado de la suma lógica **A + B** de las variables **A** y **B**, cuyas combinaciones de valores opera, como simboliza la figura A.14

La figura A.15 da cuenta de todas las sumas lógicas que realiza la compuerta **OR**.

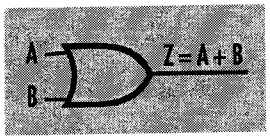


Figura A1.14

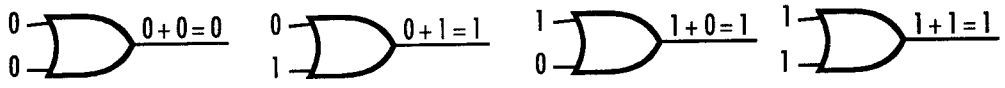


Figura A1.15

La expresión **Z = A + B** representa algebraicamente ya sea a todas las tablas dibujadas, a todos los dibujos de compuertas OR graficados, así como a todos los enunciados antes escritos que contienen la conectiva "**o**".

Acorde con esto, **F = A + B** expresa en la figura A1.12 la relación lógica entre **A** y **B**.

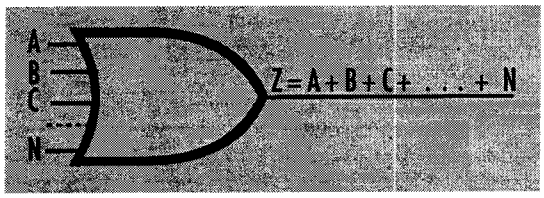
Ahora podemos expresar algebraicamente la operación lógica que realiza una compuerta OR de dos entradas y calcular su resultado sin necesidad de dibujar la compuerta o su tabla de verdad. Inversamente, dada dicha expresión algebraica, sabemos que una compuerta OR realiza la suma lógica simbolizada.

Compuerta OR de más de dos entradas

En una compuerta OR de un número cualquiera **n** de entradas (figura A.16) a las que llegan igual número de cables designados **A, B, C, ..., N**, el cable **Z** conectado en su salida estará en el nivel alto, si **A o B o C o ... o N** está en el nivel alto, o sea si uno o más de dichos cables está en el nivel alto.

El cable **Z** estará en el nivel bajo sólo si todos los cables de entrada están en ese nivel

[1] La compuerta **OR** cumple con la misma tabla formal que las operaciones citadas, sólo que calcula la columna resultado de dicha tabla en forma automática, por ser un dispositivo electrónico.



Esta compuerta realiza la suma lógica:

Z = A + B + C + ... + N de **n** variables.

Figura A.16

A	B	C	Z	
0	0	0	0	→ **0+0+0 = 0**
0	0	1	1	→ **0+0+1 = 1**
0	1	0	1	→ **0+1+0 = 1**
0	1	1	1	→ **0+1+1 = 1**
1	0	0	1	→ **0+1+0 = 1**
1	0	1	1	→ **1+0+1 = 1**
1	1	0	1	→ **1+1+0 = 1**
1	1	1	1	→ **1+1+1 = 1**

Figura A.17

Esta figura A.17 indica la tabla de verdad de una compuerta **OR** de tres entradas designadas **A, B** y **C**; siendo: **Z = A + B + C** la operación que realiza.

Al lado de la tabla se indica la forma equivalente de expresar en forma algebraica los renglones de la misma.

A1.5 Compuerta AND

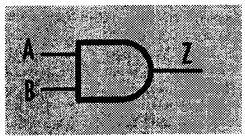


Figura A.18

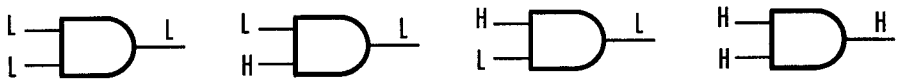


Figura A.19

A	B	Z
L	L	L
L	H	L
H	L	L
H	H	H

Figura A.20

Una ***compuerta AND*** de dos entradas (simbolizada en la figura A.18) es un dispositivo electrónico que presenta dos entradas, a las cuales llegan los niveles de tensión de dos cables (**A** y **B**), y una salida. Esta genera en el cable (**Z**) un nivel que depende de los niveles existentes en las entradas de la forma indicada en la figura A.19. Este comportamiento eléctrico se resume en la *tabla de funcionamiento* de la figura A.20, y puede enunciarse así:

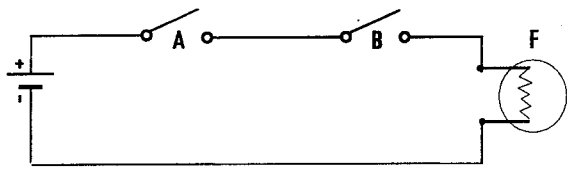
"Un cable **Z** a la salida de una compuerta **AND** estará en el nivel alto solo si el cable de entrada **A** y el cable de entrada **B** están en el nivel alto".

Se exige, pues, que ambas entradas estén *simultáneamente* en el nivel alto.

P	L	M	
0	0	0	
0	1	0	
1	0	0	
1	1	1	

⟵ Figura A.21

Figura A.22 ⟶



Un enunciado como "el motor (**M**) se activará (**M=1**) solo si está cerrada la puerta (**P=1**) y se oprime la llave (**L=1**) de accionamiento", se corresponde con la tabla de la figura A.21

En los enunciados anteriores se puso en negrita el nexo "y" (**And** en inglés), que realiza la *conjunción* de enunciados. A esta proposición debe su denominación esta compuerta.

La operación que realiza una compuerta And tiene semejanzas formales con la conjunción de enunciados, la intersección de conjuntos y la conexión serie de contactos antes tratados

En relación con esto, en el circuito de la figura A.22 con dos contactos en serie, **A** y **B**, el foco (**F**) se encenderá sólo si el contacto **A** y el contacto **B** están cerrados.

Operación producto lógico

Reemplazando en la tabla de la figura A.20 los símbolos **H** y **L** por los valores lógicos **1** y **0** , resulta *la tabla de verdad* de una compuerta **AND** de dos entradas (figura A.21)

A	B	Z	
0	0	0	→ **0.0 = 0**
0	1	0	→ **0.1 = 0**
1	0	0	→ **1.0 = 0**
1	1	1	→ **1.1 = 1**

Figura A.23

La compuerta **AND** la simbolizaremos con el operador binario presentado por un punto (.) que indica dicha operación, definida por la tabla de verdad de la figura A.23.

Se denomina ***"producto lógico"*** por coincidir simbólicamente los resultados de los productos lógicos y numéricos.

A la derecha de la tabla se ha escrito otra manera equivalente de expresar cada renglón de la misma, mediante el simbolismo de un producto de valores lógicos.

Puesto que los valores lógicos de cada producto corresponden a las variables **A**, **B** y **Z**, en forma sintética podemos expresar:

Z = A . B [1] (que se lee *"Z igual A **por** B")*

Representando cada cable por una variable, decimos que la salida de la compuerta **AND** genera el valor **0** o **1** de la variable **Z**, que es igual al valor del resultado del producto lógico de las variables **A** y **B**, cuyas combinaciones de valores opera, como se simboliza la figura A.24. Las figuras que están a su derecha dan cuenta de todas los productos lógicos que realiza la compuerta **AND**

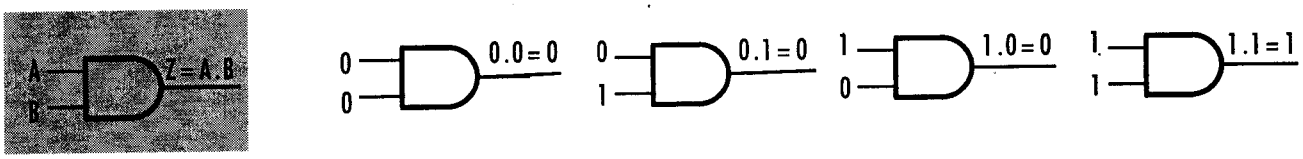


Figura A.24

Compuertas AND con más de dos entradas

En una compuerta **AND** de un número cualquiera **n** de entradas (figura A.25), a las que llegan igual número de cables designados **A, B, C, ..., N**, el cable **Z** conectado en su salida estará en el nivel alto, sólo si A y B y C y ... y N están simultáneamente en el nivel alto. Esta compuerta realiza el producto lógico de **n** variables: **Z = A.B.C. ... .N**

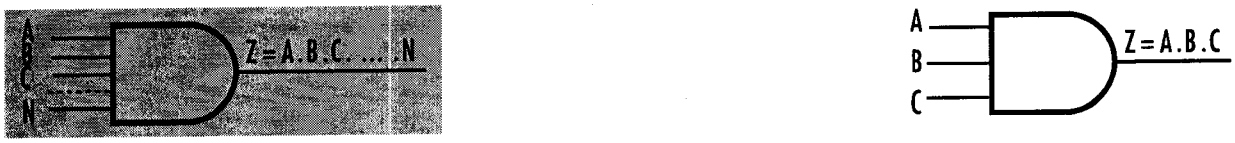


Figura A.25

Una compuerta **AND** de 3 entradas (figura A.25 derecha) realiza el producto lógico **Z = A.B.C**

Su tabla de verdad tendrá $2^3 = 8$ filas, como la tabla de la figura A.17, siendo que la columna **Z** tendrá un solo uno, en correspondencia con la combinación de entradas 111 para la cual será **Z = A.B.C = 1.1.1 = 1**

---

[1] También se asume, como en el álgebra numérica, que dos o más variables o expresiones entre paréntesis, *escritas en forma adyacente* están relacionadas por una operación producto lógico, ***aunque no exista explícitamente un punto entre ellas*** que denote tal operación. Ejemplo:

Z = AB; **Z = ABC** en vez de **Z = A.B.C** ; ó **Z = ( ) ( ) B** en lugar de **Z = ( ) . ( ) . B**

A1.6 Compuerta Inversor

Una compuerta inversora o ***inversor*** (simbolizado en la figura A.26) es un dispositivo electrónico que genera en el cable que está en su salida un nivel de tensión alto si el cable que está en su entrada presenta un nivel bajo, y viceversa. Esto se indica la figura A.27, y resume la tabla de la figura A.28.

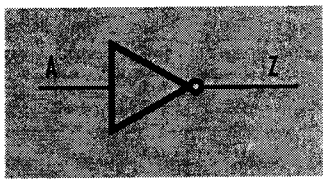


Figura A.26

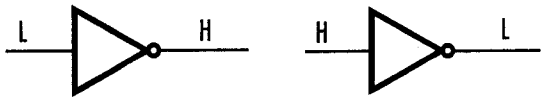


Figura A.27

A	Z
L	H
H	L

Figura A.28

Decimos que los cables **A** y **Z** son ***complementarios***, o que uno es la ***inversa*** del otro, o que están ***en oposición***, o que uno está en el nivel en que *no* está el otro, o que uno es ***la negación*** del otro, como quiera expresarse

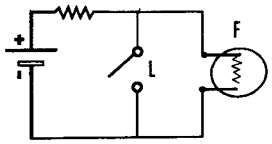


Figura A.29

A	Z	
0	1	→ $\bar{0}=1$
1	0	→ $\bar{1}=0$

Figura A.30

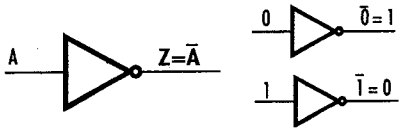


Figura A.31

En la figura A.29 podemos decir que el contacto (**L**) y el foco luminoso (**F**) están en oposición, en el sentido de que cuando el contacto está abierto el foco está encendido, y cuando el contacto está cerrado el foco está apagado, puesto que toda la corriente circula por el contacto, por cortocircuitar éste al foco.

Operación negación

Asignando **1** y **0** a los niveles **H** y **L** de la figura A.28 resulta la tabla de verdad de la compuerta inversora (figura A.30), que define la *operación inversión* que ésta realiza

Puesto que **Z** vale **1** cuando **A** *no* vale **1**, y que **Z** vale **0** cuando A *no* vale **0**, se puede enunciar que **Z** es *no* **A**, o que **Z** es la ***negación*** de **A**, simbolizándose: $\mathbf{Z = \bar{A}}$

Esta es la operación lógica que realiza el inversor, como aparece en la figura A.31.

De manera recíproca **A** es la negación de **Z**, o sea: $\mathbf{A = \bar{Z}}$

En general la negación de una variabla **A** es $\bar{\mathbf{A}}$

El símbolo de barra (¯) sobre la variable indica la *operación de negación* que el inversor realiza sobre los valores de esa variable, definida por la tabla de la figura A.30.

En particular simbolizamos $\bar{\mathbf{0}} = \mathbf{1}$ a la acción del inversor de negar un valor lógico **0** presente en su entrada para que resulte en su salida el valor lógico **1**. Del mismo modo, $\bar{\mathbf{1}} = \mathbf{0}$ simboliza la otra posibilidad de inversión.

Dados dos cables que están en oposición, resulta indiferente cuál se designa **A** y cual $\bar{\mathbf{A}}$. Si el cable **A** está con el valor lógico **0** (**A = 0**), algebraicamente en el otro cable será $\bar{\mathbf{A}} = \bar{\mathbf{0}} = \mathbf{1}$ el valor lógico del cable $\bar{\mathbf{A}}$. Asimismo, si **A = 1** es $\bar{\mathbf{A}} = \bar{\mathbf{1}} = \mathbf{0}$. Para el circuito de la figura A.29 puede escribirse $\mathbf{F = \bar{L}}$

Debe mencionarse que en general un círculo (figura A.32) denota inversión, esté o no acompañado por un triángulo como en la figura A.26

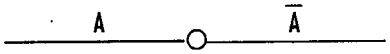


Figura A.32

Doble inversión (negación de negación)

En la figura A.33 se han dispuesto dos inversores en serie, de modo que el cable $\overline{A}$ que está a la salida del primero sea a su vez entrada del segundo inversor.

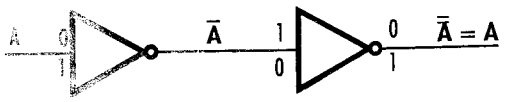


Figura A.33

El cable que está a la salida del segundo inversor está en oposición con el cable $\overline{A}$, por lo cual se simboliza $\overline{\overline{A}}$.

De esta forma, este cable en cualquier instante de tiempo tendrá un nivel con el mismo valor lógico que el cable A, que es entrada del primer inversor, también opuesto al cable $\overline{A}$. Por lo tanto podemos escribir: $\overline{\overline{A}} = A$

A1.7 Compuerta OR EXCLUSIVA (X-OR)

La palabra **o** del castellano tiene un significado ambiguo, al igual que la **or** del inglés. En el sentido débil dado en la sección A1.4 "**A o B**" significa "uno u otro, posiblemente ambos". En realidad es "**A y/o B**", puesto que *incluye* la posibilidad de simultaneidad, como a veces se aclara .

Una ***compuerta OR excluyente***, más abreviadamente ***X-OR***, *("exclusive Or")* de dos entradas (simbolizada en la figura A.34) es un dispositivo electrónico que presenta dos entradas, a las cuales llegan los niveles de tensión de dos cables (**A** y **B**), y una salida, que genera en el cable (**Z**) un nivel que depende de los niveles existentes en las entradas de la forma indicada en la figura A.35

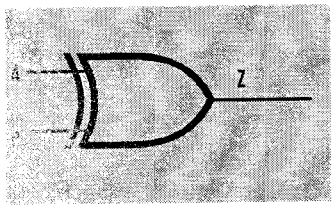


Figura A.34

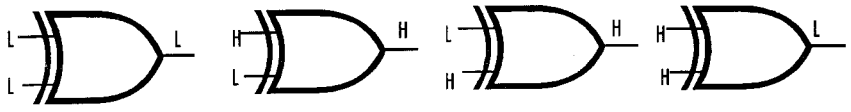


Figura A.35

Interesa ahora el sentido fuerte de la disyunción, cuando significa *"al menos uno y a lo sumo uno"*, por lo que *excluye* la posibilidad de simultaneidad, como en "Carlos está en su casa **o** en el trabajo", lo mismo que en "Un cable de un circuito lógico está en un nivel alto **o** en un nivel bajo"

A	B	Z
L	L	L
L	H	H
H	L	H
H	H	L

El comportamiento eléctrico de la **X-OR** se resume en la tabla de la figura A.36, que difiere de la tabla de la compuerta OR (figura A.9) sólo en el último renglón.

Figura A.36

Esta tabla puede enunciarse así:

> "Un cable **Z** a la salida de una compuerta **X-OR** estará en el nivel alto si ***al menos uno y a lo sumo uno*** de los cables de entrada **A** o **B**, está en el nivel alto.

A	B	Z	
0	0	0	$\rightarrow$ $0 \oplus 0 = 0$
0	1	1	$\rightarrow$ $0 \oplus 1 = 1$
1	0	1	$\rightarrow$ $1 \oplus 0 = 1$
1	1	0	$\rightarrow$ $1 \oplus 1 = 0$

Esta operación la simbolizaremos con el operador binario representado con el símbolo $\oplus$ que indica esta operación, definida por la tabla de la figura A.37, la cual contempla todos casos posibles. Se denomina *"operación Or exclusiva"* .

A la derecha de la tabla se ha escrito otra manera equivalente de expresar cada renglón de misma algebraicamente, mediante la operación Or exclusiva

Figura A.37

Puesto que los valores lógicos de cada producto corresponden a las variables **A, B** y **Z**, podemos expresar:

$$\mathbf{Z = A \oplus B}$$ (que se lee *"Z igual A* ***or excluyente*** *B")*

Representando cada cable por una variable, decimos que la salida de la compuerta **X-OR** genera el valor **0** o **1** de la variable **Z**, que es igual al valor del resultado de la operación or excluyente de las variables **A** y **B**, cuyas combinaciones de valores opera, como se simboliza la figura A.38. La figura A.39 da cuenta de todas las sumas lógicas que realiza la compuerta **X-OR**.

Figura A.38

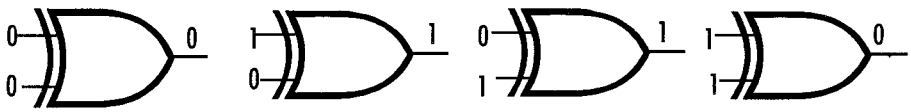


Figura A.39

A1.8 Diagramas temporales de compuertas

Los niveles de tensión en los cables que entran y salen de las compuertas varían en el tiempo, dando lugar a pulsos de tensión de ancho variable. Un diagrama que dé cuenta del comportamiento de un circuito digital a través del tiempo, es muchas veces imprescindible para su análisis.

El diagrama temporal de la figura A.41 representa los cambios acaecidos en los cables de entrada, y la tensión resultante en las salidas de una compuerta **AND**, supuesta ideal, sin retardos.

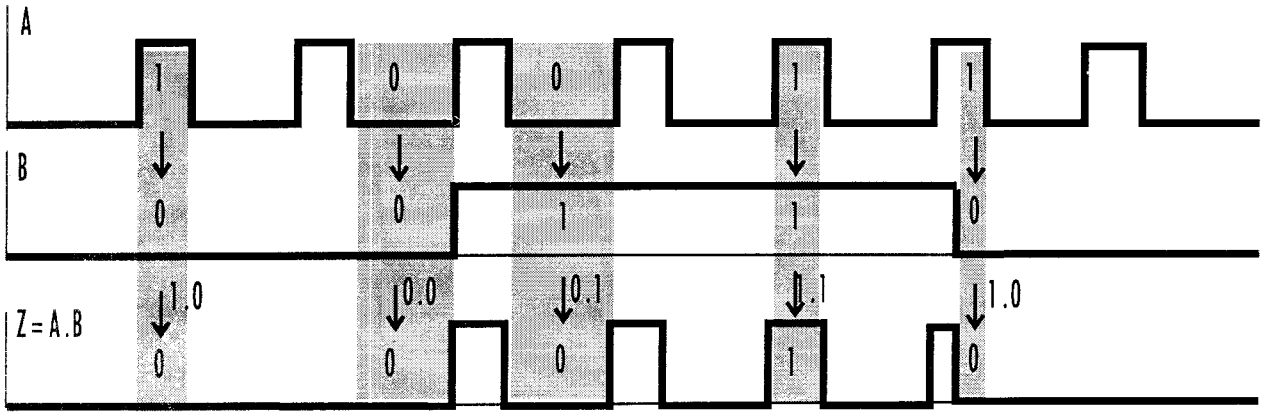


Figura A.41

Se observa en este ejemplo, cómo la señal **B** convenientemente aplicada, permite que la salida **Z** presente o no una señal semejante a la señal **A.** Esto último ocurre mientras **B** está en el nivel alto, (durante tres pulsos de la señal **A** en este caso), y es como si la compuerta dejara pasar la señal del cable **A** hacia el cable **Z**. Esto justifica la denominación "compuerta" (sección A1.3)

Del mismo modo, una compuerta **OR** dejará pasar la señal existente en una entrada, cuando la otra esté en el nivel bajo, como puede apreciarse en el diagrama de la figura A.42

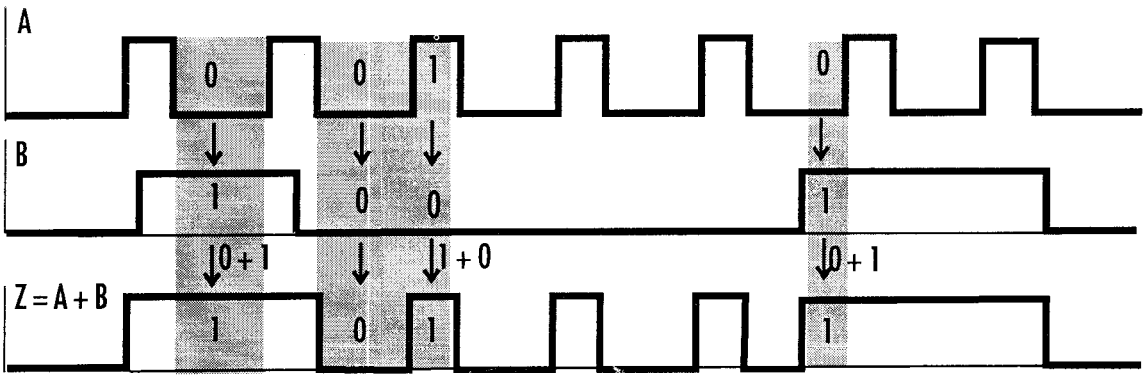


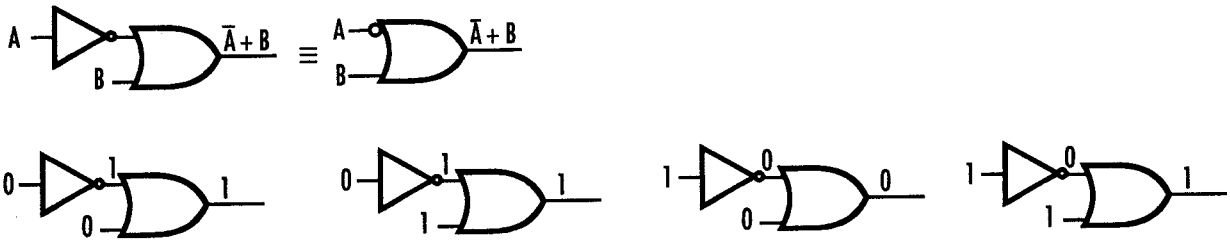
Figura A.42

A1.9 Compuertas combinadas con inversores

A1.9.1 Compuertas con inversores en sus entradas

(OPERACIONES LOGICAS CON VARIABLES NEGADAS)

Una compuerta **OR** con un inversor en su entrada **A** se puede dibujar de las dos formas indicadas en la figura A.43, dado que un círculo denota negación



Figuras A.43 (superior) y A.44 (inferior)

A	B	$\bar{A}+B$	
0	0	1	$(\bar{0}+0=1+0=1)$
0	1	1	$(\bar{0}+1=1+1=1)$
1	0	0	$(\bar{1}+0=0+0=0)$
1	1	1	$(\bar{1}+1=0+1=1)$

Figura A.45

El conjunto realiza la suma lógica $\bar{A}+B$ (que se lee "*no A o B*") cuyos resultados se han determinado en la figura A.44, en conocimiento de la tabla de la com puerta **OR** (figura A.9) y se resumen en la figura A.45 A la derecha se han calculado algebraicamente los mismos resultados.

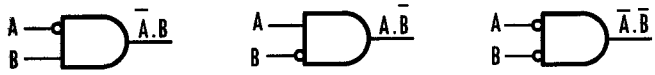
A	B	$A+\bar{B}$
0	0	1
0	1	0
1	0	1
1	1	1

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

De la misma forma pueden deducirse las dos combinaciones de compuertas siguientes que realizan las operaciones "**A ó no B**" y "**no A ó no B**" (figuras A.46 y A.47)

←Figuras A.46 y A.47

Igualmente pueden realizarse combinaciones de compuertas **AND** con una o más entradas negadas, que dan lugaa las operaciones definidas en las tablas de las figuras A.48, A.49 y A.50



A	B	$\bar{A}.B$
0	0	0
0	1	1
1	0	0
1	1	0

A	B	$A.\bar{B}$
0	0	0
0	1	0
1	0	1
1	1	0

A	B	$\bar{A}.\bar{B}$
0	0	1
0	1	0
1	0	0
1	1	0

Figuras A.48, A.49 y A.50

El procedimiento realizado puede generalizarse a compuertas de cualquier número de entradas, por ejemplo, una que realice la operación $\mathbf{Z = A + \bar{B} + C + \bar{D}}$

A1.9.2 Compuertas con un inversor en su salida
(OPERACIONES LOGICAS NEGADAS)

Compuerta NOR

Una compuerta **NOR** (figura A.51) es una compuerta OR con un inversor en su salida que complementa cada resultado que ésta genera, de modo de realizar una suma lógica negada.[1]

Por lo tanto, (figura A.52) el resultado de cada operación que realiza una compuerta **NOR**[2] será la negación del que aparece en la figura A1.8 Como indica la misma figura, en lugar del inversor completo se dibuja el círculo que denota negación.

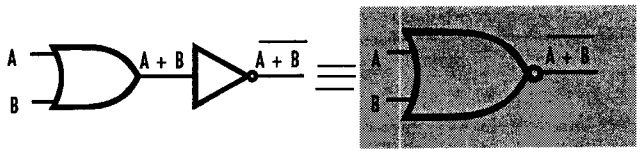


Figura A.51

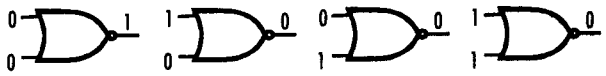


Figura A.52

La tabla de verdad de esta compuerta compuesta (figura A.53) se obtiene simplemente invirtiendo los resultados de la tabla de la **OR** (figura A.13)

A	B	$\overline{A+B}$	
0	0	1	$\rightarrow \overline{0+0} = \bar{0} = 1$
0	1	0	$\rightarrow \overline{0+1} = \bar{1} = 0$
1	0	0	$\rightarrow \overline{1+0} = \bar{1} = 0$
1	1	0	$\rightarrow \overline{1+1} = \bar{1} = 0$

Figura A.53

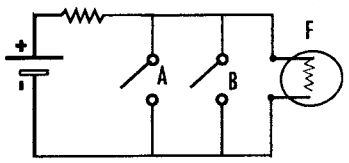


Figura A.54

En la figura A.54 se muestra una analogía con llaves que cumple con la tabla de la figura A.53.

Existe una oposición entre el cierre de uno u otro contacto (**A o B**) y el encendido del foco (**F**), puesto que ello produce un cortocircuito que no permite la circulación de corriente por el foco. Resulta $\mathbf{F = \overline{A + B}}$.

Igualmente, un enunciado como "el automovil **no** avanza con luz roja **o** cuando cruza un peatón" es de la forma $\mathbf{Z = \overline{A + B}}$.

Una compuerta **NOR** de **n** entradas realiza la operación $\mathbf{Z = \overline{A + B + C + \ldots + N}}$

Compuerta NAND

La figura A.56 indica el símbolo de la compuerta **NAND** que surge de invertir la salida de una AND, por lo que los resultados de la figura A.57 serán los complemen-tarios de las correspondientes operaciones de la figura A.23, a la par que la tabla de la figura A.58 negará los resultados de la figura A.23

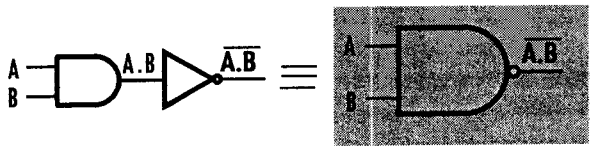


Figura A.56

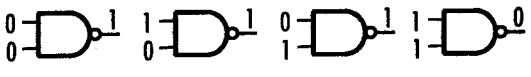


Figura A.57

---

[1] **CONVENCION:** una expresión negada se simboliza mediante una barra de negación que abarca **todas** las variables y operaciones. No debe confundirse una operación con variables negadas con una operación negada:

$\mathbf{\overline{A+B}}$ **no es igual a** $\mathbf{\bar{A}+\bar{B}}$. Esto puede verificarse cotejando las tablas de ambas expresiones (figuras A.53 y A.47), que presentan distintos resultados para iguales valores de **A B**. Igualmente $\mathbf{\overline{A.B}}$ **no es igual a** $\mathbf{\overline{A}.\overline{B}}$

[2] Ahora puede verse que resulta más claro usar **or** en vez de la **o** del castellano, pues resulta más claro decir y escribir **nor** que **no-o**

A	B	$\overline{A.B}$	
0	0	1	$\rightarrow \overline{0.0} = \bar{0} = 1$
0	1	1	$\rightarrow \overline{0.1} = \bar{0} = 1$
1	0	1	$\rightarrow \overline{1.0} = \bar{0} = 1$
1	1	0	$\rightarrow \overline{1.1} = \bar{1} = 0$

Figura A.58

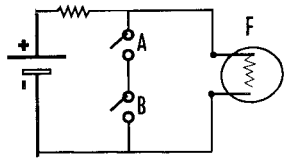


Figura A.59

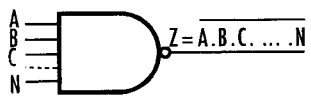


Figura A.60

En la figura A.59 se indica una analogía circuital con llaves, que cumple con la tabla anterior : el foco (**F**) se apaga si las llaves **A** y **B** se cierran: $F = \overline{A.B}$. También ella se corresponderá con un enunciado como "**no** avanzar si el semáforo está con luz verde **y** amarilla".

Una compuerta **NAND** de varias entradas realiza la operación $Z = \overline{A.B.C....N}$, como simboliza la figura A.60

A1.9.3 Compuertas NOR y NAND con inversores en sus entradas

(OPERACIONES LOGICAS NEGADAS CON VARIABLES NEGADAS)

Invirtiendo los resultados de las tablas comprendidas entre las figuras A.45 y A.50 inclusive, resultan las siguientes tablas que representan las operaciones que realizan las combinaciones circuitales dibujadas arriba de las mismas.

AB	$\overline{\bar{A}+B}$
00	0
01	0
10	1
11	0

Figura A.61

AB	$\overline{A+\bar{B}}$
00	0
01	1
10	0
11	0

Figura A.62

AB	$\overline{\bar{A}+\bar{B}}$
00	0
01	0
10	0
11	1

Figura A.63

AB	$\overline{\bar{A}.B}$
00	1
01	0
10	1
11	1

Figura A.64

AB	$\overline{A.\bar{B}}$
00	1
01	1
10	0
11	1

Figura A.65

AB	$\overline{\bar{A}.\bar{B}}$
00	0
01	1
10	1
11	1

Figura A.66

A1.10 Definición de función lógica

Habiendo realizado un conjunto de operaciones básicas con dos variables, y calculado sus resultados asignables a una variable **Z**, podemos establecer la siguiente definición:

> Una **función lógica o booleana** es ***una variable lógica cuyo valor es equivalente al valor de una expresión algebraica,*** constituída por otras variables lógicas, relacionadas entre sí por medio de las operaciones suma lógica, y/o producto lógico, y/o negación, y/o o exclusiva.

El valor **1** ó **0** de dicha expresión depende de los valores lógicos asignados a las variables que la constituyen, y de la realización de las operaciones indicadas.

Así, en la función $Z = \overline{\overline{(A.\bar{B})} + C} \oplus \bar{A}$, para **A = 0, B=1** y **C=0** la variable **Z** vale:

$$Z = \overline{\overline{(0.\bar{1})} + 0} \oplus \bar{0} = \overline{\overline{(0.0} + 0) \oplus 1} = \overline{(\bar{0} + 0) \oplus 1} = \overline{(1+0) \oplus 1} = \overline{1 \oplus 1} = \bar{0} = 1$$

Las operaciones algebraicas con **1** y **0** se han realizado conforme a los que figuran al lado de las tablas respectivas.

Son funciones lógicas: de 2 variables con una operación las expresiones como **Z = A+B**; de una variable: $Z = \bar{C}$; y también lo son todas las expresiones de operaciones antes escritas.

A cada función lógica le corresponde una tabla de verdad, que indica el valor de la función para cada una de las 2^n combinaciones de valores binarios de sus **n** variables.

A1.11 Cuadro de funciones lógicas de dos variables

A fin de extraer algunas conclusiones, compararemos todas las tablas de verdad halladas hasta ahora para dos variables. Ellas *difieren* en la disposición vertical de unos y ceros de la columna de resultados, para el orden 00, 01, 10, 11 de valores de A y B que todas tienen en común, y que se ha respetado a la izquierda del cuadro de la figura A.67. Esta contiene en sus columnas verticales todas las combinaciones de resultados posibles, desde 0000 hasta 1111, como puede apreciarse en sentido vertical.

La columna $\mathbf{f_2}$ con los resultados leídos en forma vertical 0001 corresponde a las tablas –iguales– de las figuras A.21 y A.63 de las funciones **A.B** y $\overline{\overline{A}+\overline{B}}$ respectivamente.

Del mismo modo se han determinado las funciones que corresponden a las columnas $\mathbf{f_3}$, $\mathbf{f_5}$, $\mathbf{f_7}$, $\mathbf{f_8}$, $\mathbf{f_{12}}$, $\mathbf{f_{14}}$ y $\mathbf{f_{15}}$, como puede verificarse cotejando las figuras correspondientes.

La columna $\mathbf{f_{10}}$ es la opuesta de $\mathbf{f_7}$, por lo que la operación será: $\overline{\mathbf{A \oplus B}}$.

La columna $\mathbf{f_4}$ coincide con la columna de la variable **A**. Circuitalmente es igual que recibir directamente el valor del cable A.

Lo mismo en la columna $\mathbf{f_6}$ respecto del cable **B**.

AB	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	0	$\mathbf{A.B}$ $\overline{\overline{A}+\overline{B}}$	$\mathbf{A.\overline{B}}$ $\overline{\overline{A}+B}$	**A**	$\overline{\mathbf{A}}\mathbf{.B}$ $\overline{A+\overline{B}}$	**B**	A⊕B	$\overline{\overline{A}.\overline{B}}$ A+B	$\overline{A}.\overline{B}$ $\overline{A+B}$	$\overline{A \oplus B}$	$\overline{\mathbf{B}}$	$\overline{\overline{A}.B}$ $\overline{A+B}$	$\overline{\mathbf{A}}$	$\overline{\overline{A}.B}$ $\overline{A+B}$	$\overline{A.\overline{B}}$ $\overline{A+\overline{B}}$	1

Figura A.67

Las columnas $\mathbf{f_1}$ y $\mathbf{f_{16}}$ presentan el mismo valor **0** y **1**, respectivamente, para todos los valores de **A** y **B**, por lo que no corresponden a funciones (pues éstas deben poder variar entre dos valores) sino a ***constantes*** de valor **0** y **1** Estos valores constantes corresponderían a cables que estén *siempre* en el nivel bajo y alto, Tal sería el caso de un cable conectado a 0 volts y otro a 5 volts, respectivamente (masa y "vivo" de la fuente de alimentación) como se estipuló en la sección A.1

El presente cuadro para todas las funciones de dos variables tiene 2^2 combinaciones distintas (renglones), y el total de columnas es $16 = \mathbf{2^{2^2}}$.

En general, para funciones de **n** variables, se tendrá un cuadro con $\mathbf{2^n}$ renglones y $\mathbf{2^{2^n}}$ funciones (columnas) diferentes.

Del ejemplo anterior resulta que se puede definir una **función lógica** de **n** variables como una asignación particular de valores **1** y **0** para todas las 2^n combinaciones posibles de los valores de las **n** variables.

A1.12 Equivalencia entre funciones lógicas

Dos expresiones booleanas o funciones son ***equivalentes*** si tienen igual tabla de verdad

Esta definición permite igualar expresiones que en la figura A.67 han resultado con la misma tabla, como ser:

$$\mathbf{A.B = \overline{\overline{A}+\overline{B}}}$$

$$\mathbf{\overline{A.B} = \overline{A}+\overline{B}}$$

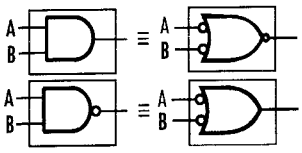


Figura A.68

$$\mathbf{\overline{A}.\overline{B} = \overline{A+B}}$$

$$\mathbf{\overline{\overline{A}.\overline{B}} = A+B}$$

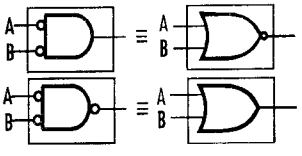


Figura A.69

$$\mathbf{\overline{A}.B = \overline{A+\overline{B}}}$$

$$\mathbf{\overline{\overline{A}.B} = A+\overline{B}}$$

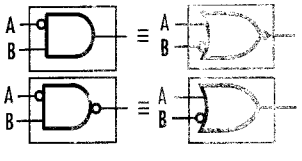


Figura A.70

En la sección A.14.2 ($\mathbf{P_{10}}$) se verá que en las igualdades halladas, una expresión se puede hallar a partir de la otra, aplicando la propiedad de De Morgan.

De las tablas y expresiones anteriores se deduce que:

Una expresión lógica le corresponde una sola tabla de verdad, mientras que una misma tabla de verdad puede formularse algebraicamente mediante diversas expresiones equivalentes.

Asimismo:

Circuitos lógicos que correspondan a expresiones algebraicas equivalentes, o sea que realicen la misma función lógica, tendrán la misma tabla de funcionamiento, por lo que podrán reemplazarse unos por otros

Si encerramos en un par de cajas dos circuitos equivalentes de los dibujados más arriba, y sólo se tiene acceso a sus cables, sin saber que contiene cada caja, no hay forma de identificar por su respuesta cuál es cada uno.

A1.13 Circuitos con varios tipos de compuertas (Operaciones Negación, OR y AND combinadas)

Análisis de un circuito combinacional

Dado un circuito combinacional de varias entradas y una o más salidas, que ya está construído, o cuyo plano circuital lógico se conoce, el ***análisis*** del mismo tiene por objetivo ***determinar su tabla de verdad o una expresión booleana de la misma.***

Esto permitirá conocer su comportamiento, o sea cuál será el nivel alto o bajo de cada salida para cada una de las combinaciones posibles de valores que puedan aplicarse en las entradas.

El análisis puede ser:

1. *Experimental:* con el circuito en cuestión funcionando
2. *Lógico:* si se tiene el plano lógico del mismo, o sea las compuertas que lo conforman y como éstas se interconectan

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

Figura A.73

En el primer caso habrá que aplicar, mediante algún dispositivo eléctrico, las distintas combinaciones de niveles de tensión altas y bajas en los cables de entrada, y medir para cada una los correspondientes niveles de tensión que resultan en las salidas. Así, suponiendo construído el circuito de la figura A.72 (página siguiente) y del cual sólo se tiene acceso a los cuatro cables de entrada y al cable de salida –sin que sea necesa rio conocer qué compuertas lo constituyen ni su intercone xionado– si se lleva a cabo la prueba anterior con los cables identificados como en dicha figura, y se traduce a valores lógicos los niveles **H** y **L**, resultará la tabla de la figura A.73

Si se tiene un plano circuital (fig, A.72) habrá que efectuar su seguimiento lógico, existiendo al respecto dos alternativas para el análisis:

2.a *Asignar cada una de las combinaciones posibles de valores lógicos en las entradas*, y para cada una calcular de izquierda a derecha el valor lógico que resulta en la salida de cada compuerta, hasta llegar a obtener el valor lógico resultante en el cable dede salida **Z.** Esto se ha ejemplificado en la figura A.72 para la combinación (ABCD) = 0000, indicándose los resultados entre paréntesis en cada salida de compuerta. Para 4 variables de entrada serán necesarios 2^4 seguimientos como el indicado para obtener la tabla de la figura A.73

2.b *Asignar una variable booleana (letra) a cada cable de entrada*[1] y realizar un solo seguimiento como el efectuado en 2.a , escribiendo a la salida de cada compuerta en forma algebraica la operación lógica parcial que realiza, como también se indica en

[1] **CONVENCION SIMBOLICA CIRCUITAL:** en el dibujo de un circuito, cada cable y sus remificaciones pueden simbolizarse:

a) con una sola letra que lo identifica, dibujando totalmente todas las ramificaciones, como aparece el cable **A** en la figura A.72, donde ***un punto en el cruce o unión de dos caminos rectilíneos simboliza que se trata del mismo cable*** (implícitamente dos caminos que se cruzan sin el punto pertenecen a cables distintos)

b) con menos caminos indicadores de cables, con la convención de que ***todas las letras del mismo nombre designan un mismo cable***, como se ejemplifica en dicha figura para los cables **B** y **C**

la figura A.72, hasta llegar a la expresión final en el cable de salida **Z**. Luego se deberá calcular algebraicamente el valor de ésta para cada combinación de valores de entrada, a fin de obtener la correspondiente tabla de verdad buscada.

Para la figura A.72 la expresión final que simboliza las operaciones que realizan las compuertas del circuito resultó:

$$Z = (A.\overline{B}.C + A.\overline{C.\overline{D}}).\overline{\overline{A} + (B \oplus C)}.(A + \overline{B} + \overline{D})$$ [1]

Esta expresión para la combinación de valores de las entradas (ABCD) = 0000 resulta

$$Z = (0.\overline{0}.0 + 0.\overline{0.\overline{0}}).\overline{\overline{0} + (0 \oplus 0)}.(0 + \overline{0} + 0) = (0.1.0 + 0.\overline{0.1}).\overline{1 + 0}.(0 + 1 + 0) = 0.\overline{1}.1 = 0$$

con lo cual queda determinado el valor de **Z** del primer renglón de la tabla de la fig. A.73.

Calculando del mismo modo para las combinaciones 0001 a 1111, se obtiene dicha tabla completa, como puede verificarse.

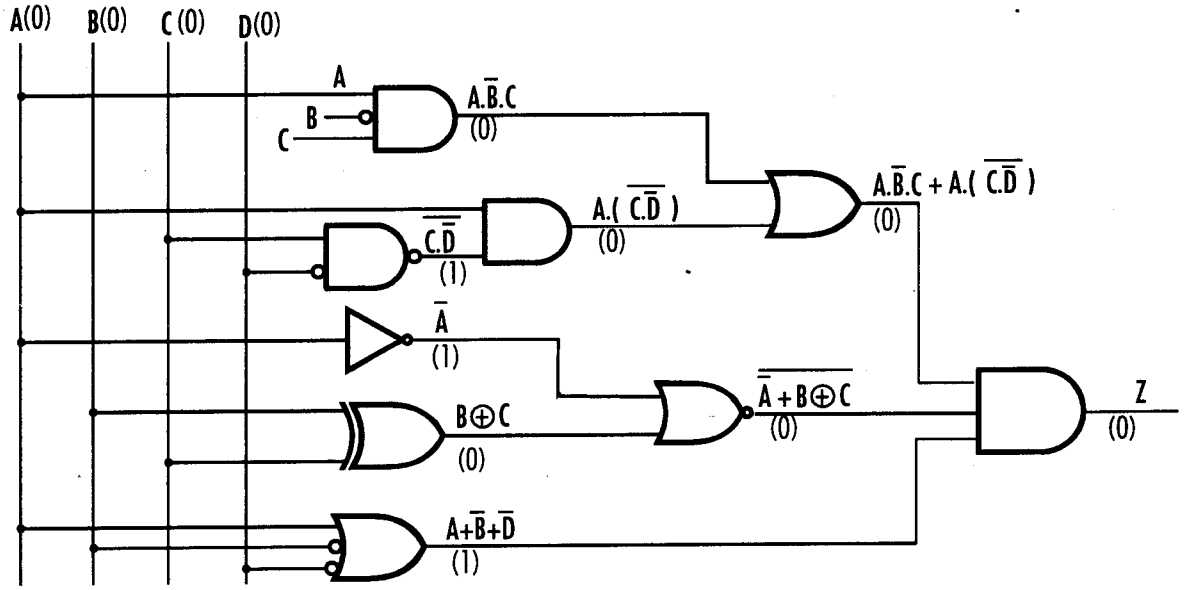


Figura A.72

Debe tenerse presente que puesto que el número de combinaciones para **n** entradas es 2^n, este número crece exponencialmente con **n**, por lo que así crecerá la cantidad de seguimientos a realizar en el circuito con el método 2.a.

A diferencia, la complejidad de una expresión circuital final sólo crece proporcionalmente al número de variables (entradas), pudiéndose agilizar los cálculos indicados en dicha expresión, conforme se requiere en el método 2.b. Asimismo es factible simplificar la expresión final, como se trata en el apéndice "K"

---

[1] **CONVENCIONES PARA LOS PARENTESIS:** como en el álgebra numérica, ciertos paréntesis pueden eliminarse siempre y cuando no resulte ninguna ambigüedad en el orden de realización de las operaciones parciales que afecte el resultado total, teniendo siempre presente que el símbolo "más" (+) de la suma lógica predomina en la separación de operaciones parciales en relación con el símbolo "por" (.) del producto lógico.

Así, en la figura A.72 se escribió $A.\overline{B}.C + A.\overline{C.\overline{D}}$ en vez de $(A.\overline{B}.C) + (A.\overline{C.\overline{D}})$ dado que resulta claro que deben realizarse los productos parciales que componen los dos sumandos para luego sumar los resultados de esas operaciones.

A su vez $A.\overline{C.\overline{D}}$ implica claramente que es $A.(\overline{C.\overline{D}})$, o sea que **A** multiplica el resultado de $\overline{C.\overline{D}}$, sin necesidad de dibujar los paréntesis. En general una barra sobre una expresión implica que en sus extremos existen paréntesis, al igual que en $\overline{\overline{A} + (B \oplus C)}$.

En el resultado final hubo que poner entre paréntesis a $A.\overline{B}.C + A.\overline{C.\overline{D}}$ pues de no colocarlos, no queda claro si se trata de la expresión

$Z = (A.\overline{B}.C + A.\overline{C.\overline{D}}).\overline{\overline{A} + (B \oplus C)}.(A + \overline{B} + \overline{D})$ o bien de la operatoria $A.\overline{B}.C + [A.\overline{C.\overline{D}}.\overline{\overline{A} + (B \oplus C)}.(A + \overline{B} + \overline{D})]$

También hubo que poner entre paréntesis $(B \oplus C)$ para evitar la interpretación $(\overline{A} + B) \oplus C$

Por último en el tema de las convenciones, teniendo presente la existente para no escribir los puntos que denotan producto, indicada en la sección A1.5, que la expresión final se pudo haber expresado $Z = (A\overline{B}C + A\overline{CD})\,\overline{\overline{A} + (B \oplus C)}\,(A + \overline{B} + \overline{D})$

Conocida una expresión de un circuito, ***se puede transformar algebraicamente*** aplicando las propiedades del álgebra de Boole, según se verá, de modo de poder hallar ***otro más sencillo, o comparar varias opciones de realización*** , a fin de construirlo con el tipo de circuitos integrados que mejor convenga., económica y técnicamente.

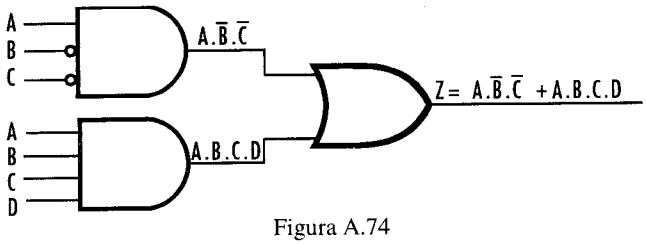


Figura A.74

Al respecto el lector podrá verificar luego de leer el apéndice que la expresión final antes hallada puede reducirse a la equivalente

$$\mathbf{Z = A.\overline{B}.\overline{C} + A.B.C.D}$$

que también le corresponde la tabla de la figura A.73. Esta acotación anticipada tiene por finalidad corroborar lo afirmado anteriormente de que una tabla de verdad puede expresarse algebraicamente de múltiples maneras, en correspondencia con las cuales pueden resultar circuitos más convenientes, según el tipo de escala de integración que se disponga.

A la última expresión le corresponde el circuito de la figura A.74, según resulta de aplicar la metodología de síntesis que se pasa a desarrollar.

Síntesis de un circuito a partir de una expresión algebraica

Al final del tema anterior se planteó la cuestión de hallar el plano lógico o dibujo de un circuito digital conociendo una expresión booleana de la operatoria que realiza.

Este problema forma parte de la **síntesis** de los circuitos digitales, que también puede incluir una etapa anterior de obtención de dicha expresión a partir de su tabla de funcionamiento (tema que se trata en la sección A1.16). Esta a su vez puede provenir de una etapa previa de traducción a variables y operaciones lógicas de las condiciones que debe cumplir dicho circuito, establecidas verbalmente o por escrito

Dada una expresión booleana, las prioridades operacionales que pueden seguirse para sintetizarla son, en orden:

1. Escritura de todas las variables en juego.
2. Implementación de las inversiones que involucran una sola variable.
3. Realización de las operaciones entre paréntesis o las negaciones que abarcan el menor número de operaciones.
4. Idem de los paréntesis o negaciones que abarcan más operaciones, etectuando primero los productos y luego las sumas.
5. Implementación de los productos restantes.
6. Realización de las sumas principales.

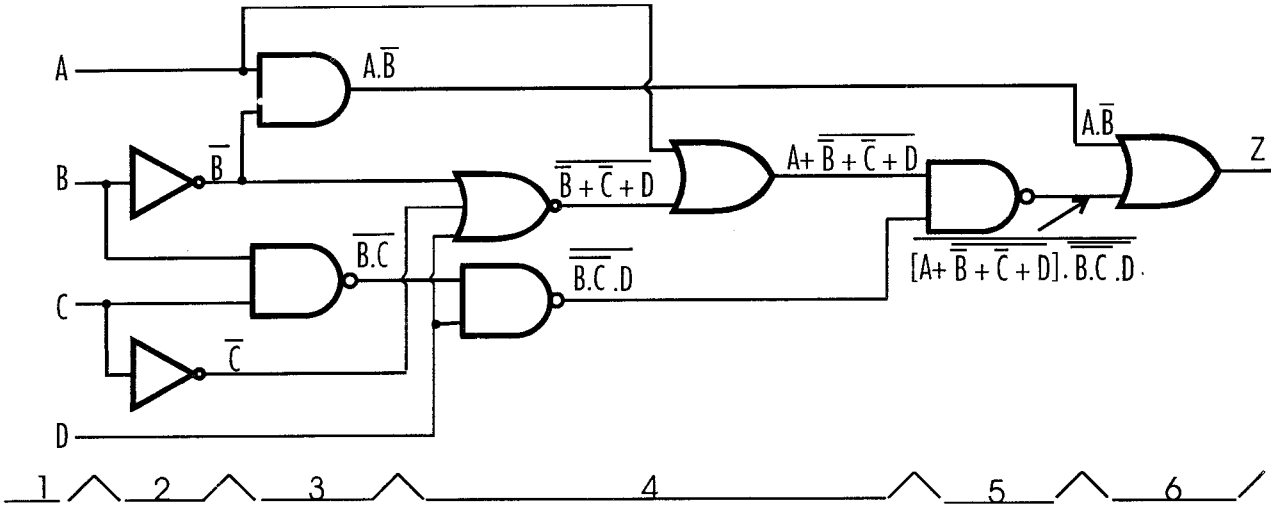


Figura A.75

þ **Ejemplo** (figura A.75): Sintetizar $\mathbf{Z = \overline{A + \overline{(\overline{B} + \overline{C} + D).\overline{B.C.D}}} + A.\overline{B}}$

Obsérvese cómo la expresión algebraica de un circuito permite deducir el conexionado que debe existir entre las compuertas que lo componen, para que éstas realicen las operaciones simbolizadas, de modo de obtener el plano lógico del mismo

A1.14 Verificación y aplicaciones de propiedades booleanas

A fin de tener cierta coherencia en el orden de presentar las propiedades y sus aplicaciones, se ha optado por seguir el planteo que aparece en el apéndice de este capítulo, debido a Huntington, quién estableció el álgebra de Boole como un sistema axiomático basado en cuatro propiedades primeras o postulados que deben cumplir los sistemas a los que puede aplicarse éste álgebra.

De éstos pueden deducirse algebraicamente las restantes propiedades como teoremas, siendo éstas válidas para cualquier sistema material que verifique las cuatro propiedades-postulados, siendo que a cualquier par de elementos del mismo se les pueda aplicar dos operaciones binarias. Estas en el caso de los circuitos digitales son las operaciones **OR** y **AND** antes definidas.

Otras propiedades pueden ser tomadas como postulados en otras formalizaciones axiomáticas de este álgebra. Por tal motivo es de relativa importancia el orden en que se dan estas propiedades, al igual de cuáles son consideradas como primeras. En principio importa conocerlas y poder aplicarlas cuando sea necesario.

A1.14.1 Cuatro propiedades primeras

P_1) ***Propiedad conmutativa de las operaciones suma y producto lógicos***

En una suma lógica o en un producto lógico no importa el orden en que se toman los operandos:[1]

$$A + B = B + A \quad \text{(A1.a)}$$
$$\downarrow \qquad \downarrow$$
$$A \,.\, B = B \,.\, A \quad \text{(A1.b)}$$

Circuitalmente:

Verificación:

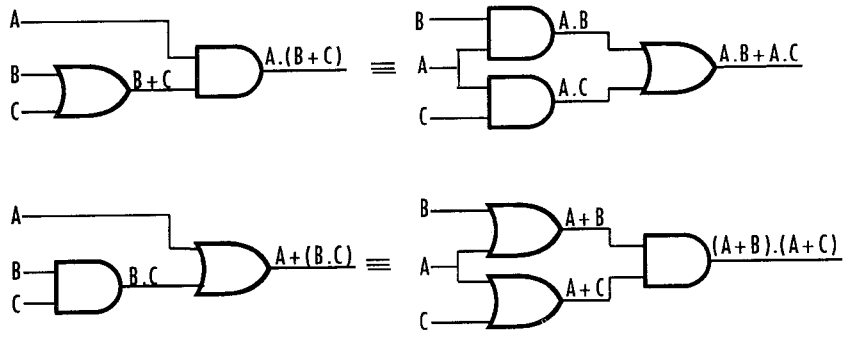
Dado que· **0+0 = 0**; **0+1 = 1+0 = 1**; y **1+1 = 1**; da lo mismo decir que se está haciendo **A + B** que **B + A .** El mismo razonamiento vale para el producto lógico.

P_2) ***Propiedad distributiva de una operación respecto de la otra***

El producto lógico es distributivo respecto de la suma lógica y viceversa:

$$A \,.\, (B + C) = A \,.\, B + A \,.\, C \quad \text{(A2.a)}$$
$$\downarrow \quad \downarrow \qquad \downarrow \quad \downarrow \quad \downarrow$$
$$A + (B \,.\, C) = (A + B) \,.\, (A + C) \quad \text{(A2.b)}$$

Circuitalmente:



Verificación:

Una forma de verificar ambas igualdades es calcular para todas las combinaciones posibles de valores lógicos de las variables **A,B** y **C** el valor del resultado de cada una de las expresiones que constituyen cada igualdad, realizando las operaciones en ellas indicadas. Si las tablas de verdad que resultan luego de efectuar todas las operaciones son iguales, las expresiones en cuestión serán

---

1 Las flechas verticales relacionan las dos propiedades, y sirven para hallar una conociendo la otra, como se plantea en la scción A.15 al tratar ***"el principio de dualidad"***

equivalentes. Mediante la tabla siguiente se verifica la equivalencia (A.2.b), y en ella se indica debajo de cada columna con operaciones el cálculo pertinente.

Como se remarca en negrita, los resultados de las operaciones de ambas expresiones son iguales para cada combinación de valores de **A**,**B** y **C** Por consiguiente se verifica la igualdad (A.2.b), y ambas expresiones tendrán la tabla de verdad de la derecha

A	B	C	A + B.C	(A+B).(A+C)
0	0	0	0+0.0 = 0+0 = **0**	(0+0).(0+0) = 0.0 = **0**
0	0	1	0+0.1 = 0+0 = **0**	(0+0).(0+1) = 0.1 = **0**
0	1	0	0+1.0 = 0+0 = **0**	(0+1).(0+0) = 1.0 = **0**
0	1	1	0+1.1 = 0+1 = **1**	(0+1).(0+1) = 1.1 = **1**
1	0	0	1+0.0 = 1+0 = **1**	(1+0).(1+0) = 1.1 = **1**
1	0	1	1+0.1 = 1+0 = **1**	(1+0).(1+1) = 1.1 = **1**
1	1	0	1+1.0 = 1+0 = **1**	(1+1).(1+0) = 1.1 = **1**
1	1	1	1+1.1 = 1+1 = **1**	(1+1).(1+1) = 1.1 = **1**

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Del mismo modo puede verificarse la igualdad (A.2.a)[1]

Aplicación:

Típicamente la propiedad (A.2.a) aplicada reiteradamente como en el álgebra numérica se usa para transformar productos de sumas en sumas de productos:

$$(\overline{A}+B).(C+D) = C.(\overline{A}+B) + D.(\overline{A}+B) = C.\overline{A} + C.B + D.\overline{A} + D.B$$

↑ Producto de sumas — ↑ Suma de productos

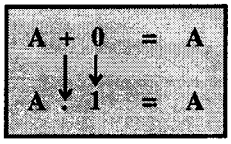
También se emplea en sentido inverso, para factorear una suma de productos, sacando factor común, para convertir una suma de productos en un producto de sumas, como sería en la expresión anterior ir de la expresión de la derecha hasta llegar a la primera de la izquierda.

Igualmente se usa la propiedad (A.2.b) para obtener productos de sumas:

$$A.(\overline{B}+C) + D = D + A.(\overline{B}+C) = (D+A).[D+(\overline{B}+C)] = (D+A).(D+\overline{B}+C)$$[2]

P_3) ***Invariancia de la suma y producto lógicos respecto del sumando 0 y del factor 1***

*El valor lógico de una variable no cambia si se le suma el valor **0** o si se la multiplica por el valor **1**:*



Verificación:

A	B	Z	
0	0	**0**	→**Z=A=0** si **B=0**
0	1	**1**	
1	0	**1**	→**Z=A=1** si **B=0**
1	1	**1**	

Se verifica en la tabla de **Z = A + B** que si **B = 0** resulta: **Z =A+ 0 =A** ;
y en la tabla de **Z = A . B** si **B = 1** resulta **Z = A . 1 = A**

---

[1] Esta tiene la misma forma simbólica de la propiedad distributiva del producto numérico respecto de la suma numérica, por lo cual resulta familiar su utilización. En cambio la igualdad (A.2.b) de la lógica *no existe* para los números, pues en éstos la suma no es distributiva respecto al producto, por lo que su aplicación resulta más difícil

[2] El paréntesis dentro de otro que se ha eliminado en el último paso, pertenece a la inversa de la propiedad asociativa de la suma lógica, aún no definida, pero se ha preferido usarla para mostrar un ejemplo más complejo.

Aplicaciones: ***compuertas NOR y NAND usadas como inversores***

Para un circuito esta propiedad puede enunciarse así *"si uno de los cables que llega a una compuerta **OR** de dos entradas está siempre en el nivel bajo (**0** lógico)[3] , el cable que está a la salida resulta con el mismo nivel que tiene el otro cable"*

*Lo mismo sucede con una compuerta **AND** de dos entradas, si uno de los dos cables está siempre en el nivel alto (**1** lógico)*[4]

Figura A.76.a

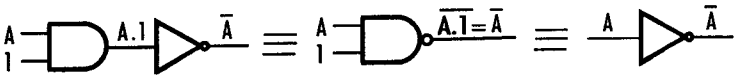


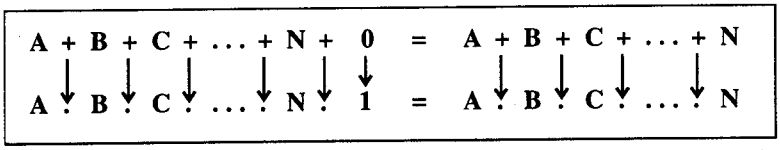
Figura A.76.b

Esto es lo que se ha simbolizado en una y otra compuerta de las figura A.76.a y b. Si a cada salida de las mismas se conecta un inversor, a la salida de éste y de compuertas NOR y NAND equivalentes a cada conjunto se tendrá $\overline{\mathbf{A}}$ También puede verificarse en las tablas de estas compuertas que si B = 0 los valores de A y Z son opuestos.

Por lo tanto una compuerta NOR o una NAND de dos entradas pueden usarse como inversores si una de las entradas se coloca a 0 volts (masa) en la NOR o a la tensión de alimentación en la NAND

Utilización de una compuerta como si tuviera menos entradas:

Muchas veces sucede que se necesitan compuertas con un cierto número de entradas, siendo que se dispone de otras del mismo tipo, pero con más entradas. La generalización de esta propiedad puede aplicarse para resolver este problema, y se enuncia así: *"en una compuerta OR (o AND) de varias entradas, si uno o más cables que llegan a las mismas están siempre con el valor lógico 0 (con el valor logico 1 en la AND) no afectarán el resultado que producen en su salida las combinaciones de valores que pueden darse en los restantes cables de entrada"*. Simbólicamente:



A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

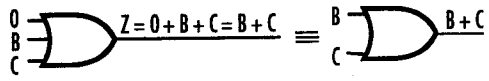
Figura A.77

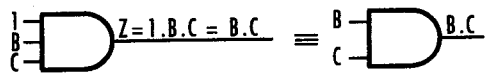
Esto se verifica para tres variables, en la tabla de **Z = A+B+C** de la fig. A.77 en cuyos cuatro primeros renglones es **A = 0**, y los valores de **Z** corresponden a la función **Z = B+C** de dos variables.

O sea **Z = 0+B+C = B+C**

Igualmente puede verificarse que **Z = 1.B.C = B.C**

Como indican las figs. A.78 y A.79, *una compuerta OR o una AND de 3 entradas con una de ellas conectada permanentemente a masa (**0** lógico) o a la tensión de alimentación (**1** lógico), respectivamente, equivalentes a sendas compuertas de 2 entradas.*



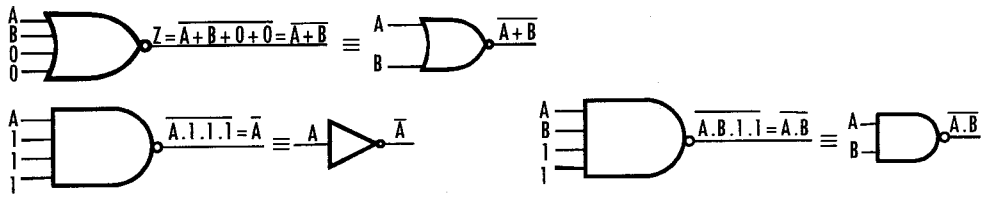


←— Figuras A.78 (superior) y A.79 (inferior)

Del mismo modo, compuertas NOR ó NAND de 4 entradas pueden usarse como compuertas similares de 3, 2 y de una entrada, en cuyo caso resulta un inversor (figuras A.80 y A.81)

3 Tal cable debe estar conectado permanentemente al terminal de la fuente que tiene 0 volts. Se identifica con un 0 en lugar de usarse una letra.

4 Tal cable debe estar conectado permanentemente al terminal "vivo" de la fuente, como ser a 5 volts. Se identifica con un 1 en lugar de usarse una letra.



Figuras A.80 (superior) y A.81 (inferior)

P_4) ***Propiedad que deben cumplir una variable y otra que es su complemento***

Si para una variable cualquiera **A** *se puede obtener otra* $\overline{A}$ *cuyo valor es siempre complementario del valor de la primera, ambas verificarán las siguientes relaciones*

$$A + \overline{A} = 1$$
$$A \cdot \overline{A} = 0$$

Estas expresiones implican que **A** y $\overline{A}$ no pueden valer simultáneamente **0** (pues resultaría $A+\overline{A} = 0$), ni simultáneamente **1** (pues sería $A.\overline{A} = 1$). Se trata pues de otra forma de dar la definición del complemento de una variable.

A1.14.2 Propiedades deducibles de las cuatro primeras

En el apéndice de este capítulo las propiedades siguientes se deducen como teoremas de las cuatro primeras consideradas como postulados. Siguiendo con la metodología utilizada, nos limitaremos a verificarlas.

P_5) ***Anulación de variables***

Si a una variable cualquiera se le suma ***1*** *(o se la multiplica por* ***0****), el resultado será dicha constante:*

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

Verificación:

Procediendo como en P_3, en la segunda y cuarta fila de la tabla de **Z = A + B** para las cuales es **B=1** se verifica que es **Z=1** independientemente que **A** valga **0** o **1**. O sea **Z = A + 1 = 1**

Igualmente en la primer y tercer filas de **Z = A.B**, donde es **B=0**, resulta **Z = A . 0 = 0**

Aplicaciones:

Prácticamente valen las mismas consideraciones mencionadas en **P_4**.

Generalización:

A+B+C+ ... +N + 1 = 1
A. B . C. ... . N . 0 = 0

P_6) ***Propiedad "idempotencia"***

Si a cualquier variable se le suma o multiplica ella misma, resulta la misma variable::

$$A + A = A$$
$$A \cdot A = A$$

Verificación:

Nuevamente procederemos como en **P_3**. Esta vez consideraremos la primera y cuarta filas en ambas tablas, donde los valores de **A, B** y **Z** coinciden. **(A=B=Z)** Para las mismas se puede escribir: **Z = A+B = A+A = A** y **Z = A.B = A.A = A** , que son las expresiones anteriores.

Aplicaciones:

Estas propiedades por un lado se usan algebraicamente, para obtener expresiones con menos variables por operación, y circuitalmente *de manera semejante a las propiedades* $\mathbf{P_3}$. Para poner esto de manifiesto se dan los mismos ejemplos.

Compuertas NAND y NOR usadas como inversores:

Como indican las figuras A.82 y 83 uniendo las dos entradas de una compuerta a un solo cable, se obtiene la función inversión:

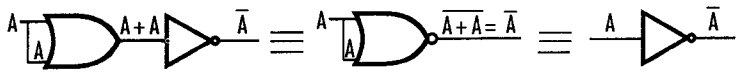


Figura A.82

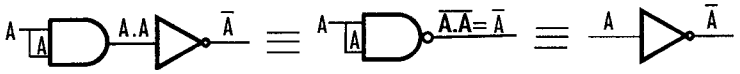


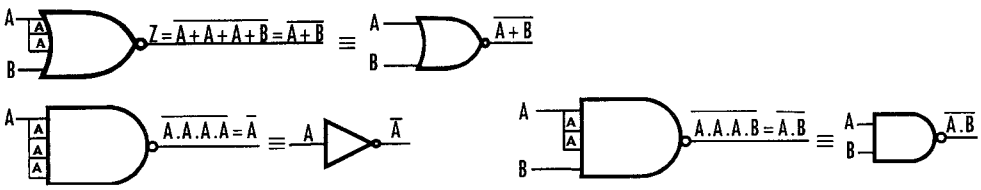
Figura A.83

Utilización de una compuerta como si tuviera menos entradas:

La generalización de las presentes propiedades puede enunciarse así: *"un sumando o un factor que se opere repetido en una suma o producto, respectivamente, no interviene más que una vez en el resultado"*. Simbólicamente:

$$\mathbf{A+A+ \dots +A+ B+C+ \dots +N = A+ B + C+ \dots + N}$$
$$\mathbf{A \,.\, A \,.\, \dots \,.A \,.\, B \,.\, C \,.\, \dots \,.\, N = A \,.\, B \,.\, C \,.\, \dots \,.\, N}$$

Esto autoriza a unir entradas de una compuerta para que realice la misma operación que otra similar de menor número de entradas, como se indica en las figuras A.84, y A.85



Figuras A.84 y A.85

P_7) ***Propiedad denominada "absorción"***

Una variable más (por) ella misma por (más) otra variable resulta ella misma:

$$\mathbf{A + A \,.\, B = A}$$
$$\mathbf{A \,.\, (A+B) = A}$$

Verificación:

Puede realizarse por tablas, como se hizo en $\mathbf{P_2}$

Aplicación:

Estas propiedades suelen usarse en la obtención de expresiones con menos variables

P_8) ***Propiedad asociativa de la suma y el producto***

En una suma (producto) de varias variables, las mismas pueden agruparse libremente en sumas (productos) parciales sin que el resultado final varíe:

$$\mathbf{A+ (B + C) = (A + B) + C}$$
$$\mathbf{A \,.\, (B \,.\, C) = (A \,.\, B) \,.\, C}$$

Verificación:

Las expresiones anteriores pueden verificarse usando tablas de verdad

Aplicaciones:

En las transformaciones algebraicas estas propiedades son de uso corriente.

Una aplicación circuital consiste (figura A.86) en reemplazar varias compuertas OR (o AND) de varias entradas combinadas en una suma (o producto) por una sola de más entradas, o combinarlas acorde a la disponibilidad de entradas por compuerta existente.

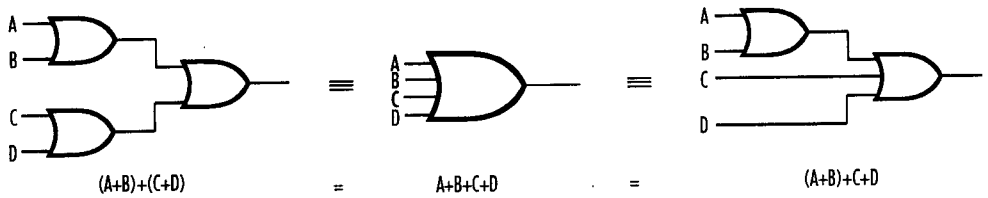


Figura A.86

P_9) ***Propiedad de la negación de la negación o "involución"***

Si una variable se niega dos veces resulta la misma variable sin negar:

$$\overline{\overline{A}} = A$$

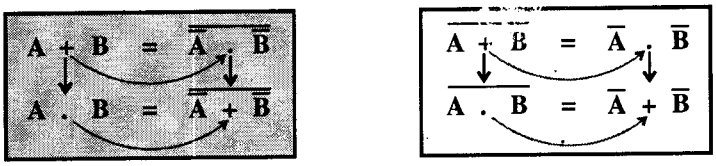
Verificación:

A	$\overline{A}$	$\overline{\overline{A}}$
0	1	0
1	0	1

Esta propiedad ya se trató aplicada a compuertas en la sección A1.6 y se verifica según la tabla de la izquierda

P_{10}) ***Propiedades de De Morgan***

Dadas las operaciones suma y producto lógicos, una cualquiera de ellas sin negar, puede transformarse en la otra negada, negando tambien las variables:



Verificación:

La equivalencia de las expresiones escritas arriba ya ha sido realizada en la sección A1.12 comparando las tablas de verdad correspondientes. Las expresiones que están a la derecha de las recuadradas indican la forma de transformar una operación *negada* en la otra sin negar.

Generalización:

$$A+B+C+...+N = \overline{\overline{A}.\overline{B}.\overline{C}.....\overline{N}}$$

$$A.B.C.....N = \overline{\overline{A}+\overline{B}+\overline{C}+...+\overline{N}}$$

Aplicaciones:

Equivalencia entre configuraciones "AND-OR" y "NAND-NAND"

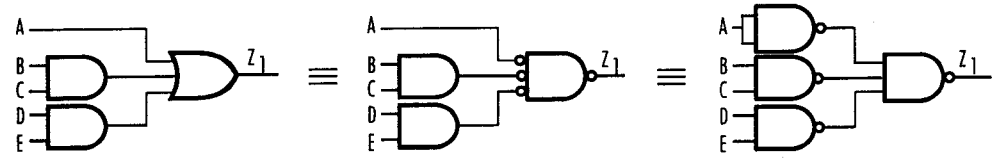


Figura A.87

Método algebraico: se halla la expresión del circuito a transformar (extremo izquierdo) y se aplica De Morgan, para obtener la expresión deseada. Con ella se dibuja el circuito buscado (de la extrema derecha), sin necesidad de dibujar el circuito del centro.

$$Z_1 = A + B.C + D.E = \overline{\overline{A}.\overline{B.C}.\overline{D.E}}$$

A1.15 Principio de Dualidad

Como indican las flechas en todas los pares de expresiones con dos o más variables de secciones anteriores, en el álgebra de Boole si se conoce una cualquiera de ellas se puede conocer la otra, que es su "dual". Esto se conoce como ***"principio de dualidad"***, y puede enunciarse:

> Cualquier propiedad en el álgebra de Boole sigue siendo válida si se intercambian todos los operadores "más" (+) y "por" (.) entre sí, y además se intercambian **1** y **0**

Igualmente puede establecerse una dualidad en la aplicación de la lógica positiva y negativa (sección A1.2) Así, si definimos conforme con esta última los niveles H y L como 0 y 1, respectivamente, la tabla de funcionamiento de la figura A.13 –que habíamos hecho corresponder en la lógica positiva con la tabla de la función OR– con la convención de la lógica negativa resulta la tabla de verdad de una función AND.

A1.16 Función como suma de minitérminos (forma "canónica")

Al tratar el análisis de circuitos (sección A1.13.1) dado el plano de un circuito hemos hallado su tabla de verdad.

Ahora nos proponemos una de las formas de resolver el problema inverso, ligado a la síntesis de circuitos: *dada una tabla de funcionamiento* que expresa ciertos requerimientos que se deben cumplimentar circuitalmente, *construir un circuito que responda a dicha tabla.*

También se vió (sección A1.12) que una tabla de verdad puede expresarse mediante tantas expresiones algebraicas equivalentes como se quiera. A su vez, cada una de estas expresiones da lugar a un circuito lógico cuya tabla de funcionamiento será formalmente igual a dicha tabla. La obtención o síntesis de un circuito a partir de una expresión algebraica ya se trató en la sección A1.13.2. Pero *aún no hemos tratado las formas de hallar circuitos que verifiquen una tabla dada.*

Para resolver esta cuestión en general hace falta determinar al menos una de las *tantas* expresiones algebraicas equivalentes que cumpla con la tabla en cuestión. Hallada esta expresión, se determina el circuito que ella simboliza.

Puesto que existen muchas expresiones equivalentes a partir de una tabla de verdad, *en cada caso se trata de hallar aquella o aquellas que correspondan a circuitos que tecnológica y económicamente interese implementar.*

En el presente capítulo las expresiones booleanas a determinar se corresponden con estructuras circuitales del tipo de las que constituirán los circuitos de una ROM, tratados en detalle en la sección A1.22, que vienen integrados en un solo "chip", parte de cuyo conexionado interno el usuario puede determinar.

Estas expresiones son las formas *"canónicas" o "normales" o "expansiones booleanas"* de una función lógica.

Existen dos formas canónicas: la *"normal disyuntiva" o "suma de minitérminos", y la "normal conjuntiva" o "producto de maxitérminos.* Trataremos la primera, por ser la de aplicación típicamente usada.

Cada una de ellas es *única:* a cada tabla de verdad le corresponde *una sola* expresión suma de minitérminos y *una sola* expresión producto de maxitérminos. Ambas son equivalentes, por expresar la misma tabla de verdad.

A1.16.1 Minitérminos

> *Dado un número n de variables, un* ***minitérmino*** *es un producto lógico, cuyos factores son todas las variables (negadas o no)*

Un producto lógico resulta con el valor **1** para *una sola* combinación de valores de las variables que son sus factores. Esto ya se apreció en las tablas de verdad de las figuras A.23 , A.48, A.49 y A.50

$\bar{A}.\bar{B} \Leftrightarrow$ **00** pues	$\bar{0}.\bar{0} = 1.1 = 1$
$\bar{A}.B \Leftrightarrow$ **01**	$\bar{0}.1 = 1.1 = 1$
$A.\bar{B} \Leftrightarrow$ **10**	$1.\bar{0} = 1.1 = 1$
A.B $\Leftrightarrow$ **11**	$1.1 = 1$

Para 2 variables **A** y **B**, los 2^2 productos minitérminos que pueden formarse aparecen en columna, junto con la combinación de valores de las variables para la cual cada producto vale **1**

Dada otra combinación de valores que no sea la correspondiente a un producto, este resulta **0**

Así, para **01** aplicada en $\bar{A}.\bar{B}$ resulta $\bar{A}.\bar{B} = \bar{0}.\bar{1} = 1.0 = 0$

$\overline{A}.\overline{B}.\overline{C}$	⇔	000	pues $\overline{0}.\overline{0}.\overline{0} = 1.1.1 = 1$
$\overline{A}.\overline{B}.C$	⇔	001	$\overline{0}.\overline{0}.1 = 1.1.1 = 1$
$\overline{A}.B.\overline{C}$	⇔	010	etc.
$\overline{A}.B.C$	⇔	011	
$A.\overline{B}.\overline{C}$	⇔	100	
$A.\overline{B}.C$	⇔	101	
$A.B.\overline{C}$	⇔	110	
A.B.C	⇔	111	

Del mismo modo, para 3 variables **A,B** y **C** se indican los 2^3 minitérminos que pueden formarse, junto con la combinación para la cual cada producto resulta **1**, como se verifica para los dos primeros.

Considerando cuatro variables, **A,B,C** y **D** se pueden formar 2^4 minitérminos distintos:

$\overline{A}.\overline{B}.\overline{C}.\overline{D} - \overline{A}.\overline{B}.\overline{C}.D - \overline{A}.\overline{B}.C.\overline{D} - \overline{A}.\overline{B}.C.D - \overline{A}.B.\overline{C}.\overline{D} - \overline{A}.B.\overline{C}.D - \overline{A}.B.C.\overline{D} - \overline{A}.B.C.D$

$A.\overline{B}.\overline{C}.\overline{D} - A.\overline{B}.\overline{C}.D - A.\overline{B}.C.\overline{D} - A.\overline{B}.C.D - A.B.\overline{C}.\overline{D} - A.B.\overline{C}.D - A.B.C.\overline{D} - A.B.C.D$

Por ejemplo, el producto $\overline{A}.B.\overline{C}.D$ vale **1** para la combinación **0101**, pues solo para ella resulta $\overline{0}.1.\overline{0}.1 = 1.1.1.1 = 1$

Obsérvese que si bien $\overline{A}.B.C$ es un minitérmino si se tiene 3 variables, **no** lo es para cuatro, pues si bien es un productdo, se exige que esté conformado por *todas* las variables. Por ello los minitérminos también se denominan *"productos completos"*.

Igualmente **no** son minitérminos para 4 variables $\overline{A.\overline{B}.C.D}$ ni $\overline{A}.\overline{B.C.D}$, por contener productos negados, dado que la definición exige que sea un producto sin negar, total o parcialmente. Sólo pueden estar negadas variables en forma individual.

Podemos establecer la siguiente *correspondencia biunívoca:*

> Para cada minitérmino hay una sola combinación para la cual el producto resulta **1**, y recíprocamente, dada una combinación de valores de las variables, existe un solo minitérmino que resulta **1** para esa combinación.

Se observa en cada correspondencia **minitérmino⇔combinación**, que los ceros y unos se corresponden individualmente con las variables negadas y sin negar, respectivamente. Así: : $0 \Leftrightarrow \overline{A}$ $1 \Leftrightarrow A$

A1.16.2 Expresión de una función como suma de minitérminos [1]

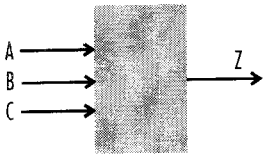


Figura A.88

Supongamos que se necesite un circuito de tres entradas, **A,B,C** y una salida **Z** (figura A.88), tal que ésta valga **1** si en las entradas se aplica la combinación 011, o la 101 o la 111. La tabla de este circuito será, pues, la que aparece en la figura A.89, y puede describirse así:
"**Z** vale **1** si la combinación de valores de **A,B,C** es 011 **o** si ella es 101 **o** si ella es 111".
Dada la correlación existente entre combinaciones y minitérminos, podemos decir:
"**Z** vale **1** si la combinación de valores de **A,B,C** es la correspondiente al minitérmino $\overline{A}.B.C$, **o** si la misma es la correspondiente al minitérmino $A.\overline{B}.C$, **o** si la misma es la correspondiente al minitérmino **A.B.C**"

A	B	C	Z
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	**0**
1	1	1	**1**

← Figura A.89

Puesto que para dichas combinaciones esos minitérminos toman el valor **1**, también podemos enunciar:
" **Z** vale **1** si la combinación de valores de **A,B,C** es aquella para la cual
$\overline{A}.B.C$ vale **1** (y cualquier otro minitérmino **0**), **o** si es aquella para la cual
$A.\overline{B}.C$ vale **1** (y cualquier otro minitérmino **0**), **o** si es aquella para la cual
A.B.C vale **1** (y cualquier otro minitérmino **0**)".
Verificaremos que acorde con este enunciado, una expresión que sigue la tabla a la figura A.88, es la formada por la suma de esos tres miniterminos.

---

1 En la sección problemas (A1.34), ejercicio 16 se indica a modo de verificación, una **forma algebraica de obtención de los minitérminos** a partir de la expresión de una función

$$Z = \overline{A}.B.C + A.\overline{B}.C + A.B.C$$

para 011 $\leftrightarrow \overline{A}.B.C$	la suma será	$Z =$	1	+	0	+	0	= 1
para 101 $\leftrightarrow A.\overline{B}.C$	la suma será	$Z =$	0	+	1	+	0	= 1
para 111 $\leftrightarrow A.B.C$	la suma será	$Z =$	0	+	0	+	1	= 1

La idea básica ha sido formar una suma de tantos sumandos como unos tiene la tabla, en este caso tres. Esta suma vale **1** para cualquiera de las tres combinaciones 011. 101, 111

Forzosamente, dado que los sumandos son productos minitérminos correspondientes a esas combinaciones, para cada una de ella un solo producto resulta **1** , y los otros dos resultan **0**. Como en una suma, basta que un sumando valga **1** para que el resultado también sea **1**, para cada una de las combinaciones será necesariamente **Z = 1.**

Para las restantes 8–3 = 5 combinaciones de valores de la tabla, será **Z=0**, como en la tabla, puesto que en la suma anterior no hay ningún minitérmino que valga **1** para una cualquiera de las mismas.

Por ejemplo, para la combinacíon 001 resulta:

$$Z = \overline{A}.B.C + A.\overline{B}.C + A.B.C = \overline{0}.0.1 + 0.\overline{0}.1 + 0.0.1 = 0 + 0 + 0 = 0$$

y para cualquier otra de esas cinco el resultado será igualmente **0+0+0**, dado que ninguno de los tres minitérminos que conforman la suma puede valer **1** para las mismas.

Por lo tanto se verifica que la suma de minitérminos así formada es una función que para cada combinación de valores de las variables tiene el mismo valor que la función representada por la tabla de verdad, tanto para los unos como para los ceros

En general: *dada una tabla de verdad de una función, si se hace una suma con los minitérminos correspondientes a las combinaciones de valores de las variables para las cuales la función vale **1**, dicha suma de minitérminos responde a dicha tabla.*

En una suma de minitérminos, existen pues tantos minitérminos como unos tiene la columna de la función en la tabla de verdad originaria.

Cada combinación de valores de variables para la cual la función vale **1**, da lugar a su correspondiente minitérmino en la suma, que se forma de la manera vista: *con la variable negada cuando el valor de ésta es **0** , y con la variable sin negar cuando el valor de ella es **1**.*

Dada la forma en que se construye una suma de minitérminos a partir de una tabla de verdad, puede establecerse:

A una tabla de verdad le corresponde una sola expresión suma de minitérminos, y viceversa

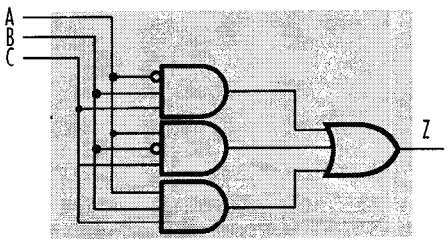
Respecto a la denominación "suma de minitérminos", debe tenerse en cuenta que la misma ***es una SP particular***, en el que todos las productos son minitérminos.

A	B	Z
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Resulta útil tener siempre presente la suma de minitérminos de la función **X-OR**, que puede deducirse de su tabla de verdad de la izquierda: $Z = A \oplus B = \overline{A}.B + A.\overline{B}$

La función suma lógica expresada como suma de minitérminos es: $Z = A + B = \overline{A}B + A.\overline{B} + A.B$ como surge de su tabla de verdad (figura A1.13). Una función puede tener un solo minitérmino. Por ejemplo, de la tabla de verdad de la función AND (figura A.23) se corrobora la expresión: $Z = A.B$

Circuito correspondiente a una suma de minitérminos:



Volviendo al problema planteado al comienzo, el circuito de la figura A.90 que corresponde a la expresión antes hallada:

$Z = \overline{A}.B.C + A.\overline{B}.C + A.B.C$, es uno de los que puede ir dentro de la caja de la figura A.88, por cumplir con la tabla de la figura A.89

⟵ Figura A.90

Suma de minitérminos de la negación de una función:

Dada la columna de la función **Z** de la tabla de la figura A.89, su complemento $\overline{Z}$ tendrá una tabla con su columna con los valores opuestos a los de **Z**, o sea con tres "ceros" y cinco "unos". Estos últimos originarán los 5 minitérminos de $\overline{Z}$, para las combinaciones de valores de **A,B,C** que en la tabla de la figura A.89 hacían **Z = 0**.

Por lo tanto será: $\overline{Z} = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.\overline{C}$

Suma de todos los minitérminos de n variables

Sumando los minitérminos hallados para **Z** y $\overline{Z}$, resulta:

$$(\overline{A}.B.C + A.\overline{B}.C + A.B.C) + (\overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.\overline{C}) = Z + \overline{Z} = 1$$

La sumatoria contiene todos los 2^3 minitérminos posibles para n = 3 variables.

*En general, dados los minitérminos de una función **Z**, de **n** variables, los de $\overline{Z}$ serán los que completan el total de 2^n minitérminos posibles, siendo la suma de todos ellos siempre **1**.*

Una función **Z** que sea la suma de todos los minitérminos correspondería a una tabla de verdad que en la columna de **Z** presente todos "unos". No sería una función, variable, sino la constante **1**, semejante a la columna $\mathbf{f_1}$, para dos variables, en la figura A.67. Asimismo, en la sumatoria de todos los minitérminos, para cada una de las 8 combinaciones posibles de valores de **A.B.C**, hay un minitérmino que resulta **1**, por lo que la sumatoria valdrá siempre **1**.

Circuitalmente, si en lugar de 3 compuertas **AND** como en la figura A.90, se tiene 8 de forma de generar todos los minitérminos posibles, para cada combinación que se aplique en las entradas **A,B,C**, siempre una de las compuertas **AND** tendrá su salida en **1**, y las restantes en **0**, con lo que la salida **Z** será siempre **1**.

A1.17 Sumador aritmético de dos números de n bits

Los circuitos lógicos también se emplean para realizar opereraciones aritméticas, en cuyo caso la función lógica que se genera en una salida es una *"función aritmética"*.

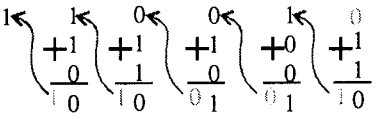
Dado que los niveles de tensión "alto" y "bajo" (1 y 0) pueden representar los dígitos 1 y 0 del sistema binario, es posible operar con números de dicho sistema numérico. En este sentido, para un circuito digital, la combinación 0101 puede significar tanto información acerca de un cierto evento como el número cinco escrito en base dos.

De la misma manera que una operación aritmética entre dos números da por resultado otro número, un circuito combinacional puede transformar dos combinaciones de "unos" y ceros presentes en sus entradas –correspondientes a dos números binarios– en otra combinación, que en sus salidas que sea el resultado de aquella

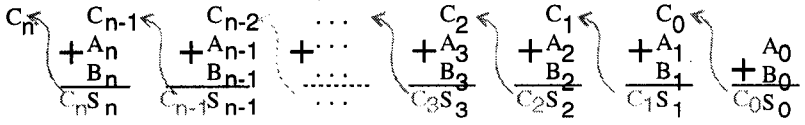
Importante: en lo que sigue para diferenciar el símbolo de la suma aritmética con el de la suma lógica, usaremos **+** para la **aritmética** y **+** para la **lógica** (como venimos haciendo)

Se tratará un circuito para sumar dos números binarios **A** y **B** cualesquiera, siendo su resultado otro números binario **S**.

Con el método manual de sumar "columna por columna", comenzando por la derecha, y teniendo presente el *acarreo ("Carry")* si se arrastra 1 de una posición a otra, la suma ejemplificada implica la operatoria siguiente:



Esta forma de operar será emulada por el circuito sumador a desarrollar, siendo que en general para dos números **A** y **B** cualesquiera deberá realizarse:



Dado que la suma de los bits de cada posición sigue un proceso idéntico, se puede esquematizar la suma de todas las posiciones mediante un conjunto de bloques encadenados como ilustra la figura A.91, en cuyos primeros bloques se han indicado los bits correspondientes a las tres primeras columnas de la suma antes ejemplificada.

Cada bloque puede sumar tres bits de una columna, y se conoce como, ***"sumador completo"*** (SC) o ***"full adder"***

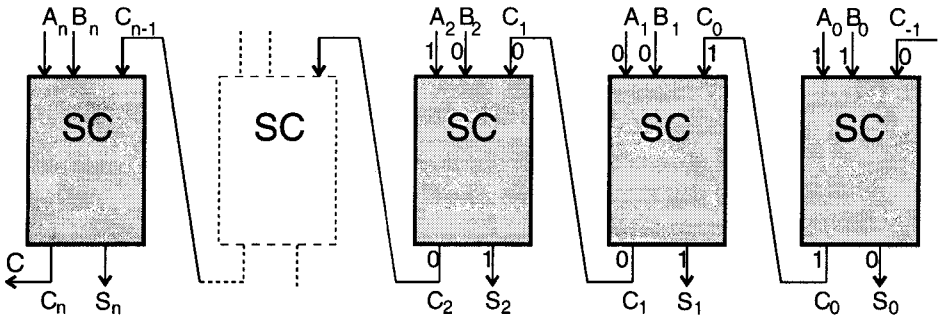


Figura A.91

Al bloque que suma A_0 con B_0 para mayor versatilidad es igual a los demás, designando C_{-1} su entrada de acarreo[1]. Esta puede provenir de un sumador anterior, o servir para sumar un "uno" extra, si se usa el sumador para restar, como se verá. Para la porción de suma ejemplificada debe ser $C_{-1} = 0$, como aparece.

La salida $C_n = C$ es el acarreo final del conjunto. De acá en más reservaremos la palabra inglesa ***"Carry"*** para hacer referencia al mismo.[2]

Enseguida se determinará un circuito que realiza la operación de un bloque SC cualquiera. Conectando repetidamente este circuito como indica la figura 8.1, se obtiene un ***"sumador paralelo-paralelo con acarreo serie"*** *de dos números de* ***n*** *bits.*

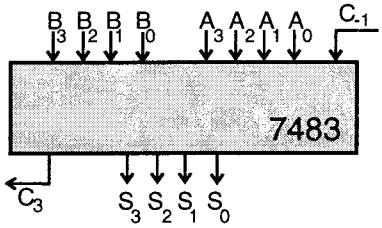
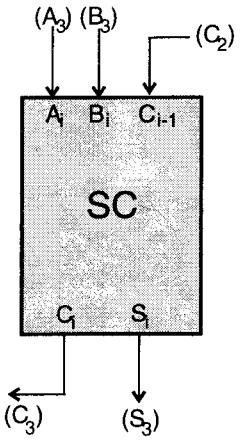


Figura A.92

Su denominación se debe a que al bloque entran simultáneamente (en paralelo) todos los bits de los dos números a sumar ($A_n,...A_2,A_1,A_0$, $B_n,...B_2,B_1,B_0$ y C_{-1}), y el resultado de la suma se tiene en paralelo en todos los cables de salida, luego que en éstos se van generando uno tras otro en el tiempo los bits del resultado, en el orden $S_0,S_1,S_2,...S_n$. Dado que cada SC para generar el resultado definitivo debe esperar que el SC anterior genere el acarreo definitivo, se dice que el acarreo se propaga "en serie".

Circuito de un sumador completo obtenido como suma de minitérminos



Cada sumador de la fig. A.91 considerado aislado (fig A.93) realiza la suma aritmética de 3 números de un bit: A_i y B_i de la posición de subíndice **i**, y el bit de acarreo C_{i-1} provenien te de la suma efectuada por el sumador de la posición anterior, de subíndice **i–1**.
Entre paréntesis se indican los subíndices suponiendo i = 3
En sus salidas aparece el bit de la suma S_i, de la posición correspondiente a los sumandos A_i y B_i, y otro bit de acarreo C_i , que deberá sumar el SC de la posición **i+1**.

La tabla de un sumador completo aparece en la figura A.94, junto con un ejemplo que indi ca cómo se confeccionó una de sus filas (indicada con una flecha)

⟵ Figura A.93

---

[1] El subíndice –1 de **C** para la posición caracterizada con el subíndice "cero", corresponde al hecho de que cada sumador completo opera en una posición identificada por un subíndice decimal, en este caso la "cero", la cual recibe el acarreo de la anterior, que tendría como subíndice dicho número –1

[2] Esta salida se puede conectar a la entrada C_{-1} de otro sumador conformado por los mismos bloques que el sumador tratado, si se quieren sumar más bits que los operados por un solo sumador.

A_i	B_i	C_{i-1}	C_i	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

→

$0 \rightarrow A_i$

$+1 \rightarrow B_i$

$1 \rightarrow C_{i-1}$

C_i ↵ ↓ S_i

Figura A.94

Conforme a la sección A1.16, consideradas en forma en forma independiente, las funciones S_i y C_i de la tabla de la figura A.94 pueden expresarse como sigue:

$$C_i = \overline{A}_i.B_i.C_{i-1} + A_i.\overline{B}_i.C_{i-1} + A_i.B_i.\overline{C}_{i-1} + A_i.B_i.C_{i-1}$$

$$S_i = \overline{A}_i.\overline{B}_i.C_{i-1} + \overline{A}_i.B_i.\overline{C}_{i-1} + A_i.\overline{B}_i.\overline{C}_{i-1} + A_i.B_i.C_{i-1}$$

Estas expresiones dan lugar a los dos circuitos de la figura A.95, que conforman la lógica interna de un bloque SC. Ambos comparten las variables A_i, B_i y C_{i-1}, que van a las entradas de las compuertas.

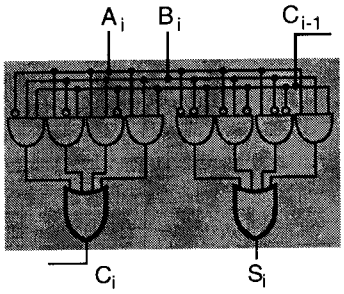


Figura A.95

Observando la tabla de la figura A.94 se corrobora lo expresado en la introducción Por ejemplo para el renglón indicado, cuando se aplica la combinación 011

las salidas generan: $10 = 0 + 1 + 1$.

No es que un sumador completo en su interior mediante sumas lógicas lleve a cabo sumas aritméticas, sino que su circuito lógico obedece a una tabla, que para cada combinación de los 3 bits a sumar genera una combinación de 2 bits, tal que coincide con el resultado de la suma aritmética de esos 3 bits (lo cual implica asimilar los valores lógicos **1** y **0** con los correspondientes valores numéricos).
Se trata pues de un circuito combinacional particular, que transforma cada combinación de 3 bits a sumar, en una combinación que puede interpretarse como la suma aritmetica de los mismos.

A1.18 Sumador-Restador de la UAL

El circuito de la figura A.96 permite ejemplificar como un sumador puede utilizarse como restador, y para comprender cómo se generan los principales indicadores **SZVC** (sección N.3 de esta unidad)

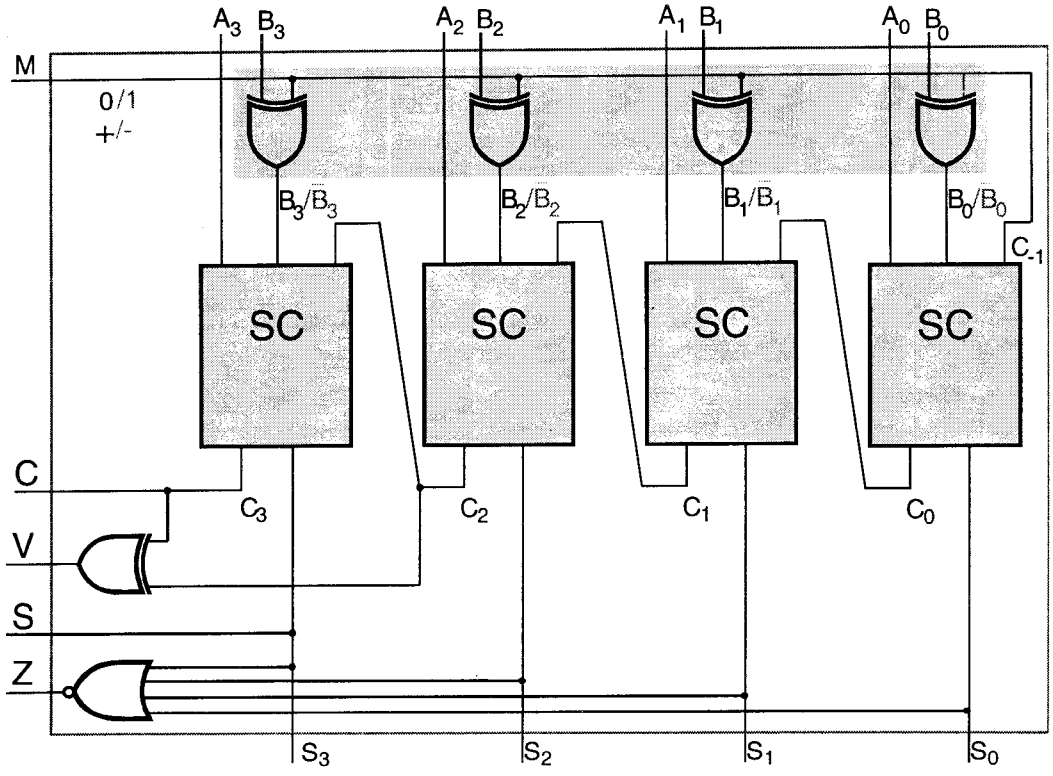


Figura A.96

Este circuito –semejante al

Este circuito –semejante al sumador de la figura A.91– puede realizar la suma o diferencia de dos números **A** y **B** (naturales o enteros), compuestos de 4 bits cada uno, que simbolizaremos $\mathbf{A(A_3,A_2,A_1,A_0)}$ y $\mathbf{B(B_3,B_2,B_1,B_0)}$.

En el apéndice de la Unidad 1 para naturales, y en la sección N.4 para enteros, se realiza **A–B**, haciendo $\mathbf{A+C_B}$, siendo $\mathbf{C_B}$ el complemento a la base de **B**. A su vez $\mathbf{C_B}$ se calcula sumando 1 a $\mathbf{C'_B}$, complemento a la base menos uno de **B**, que se obtiene invirtiendo **B** (sección N1.2)

M	B_i	$Z=M\oplus B_i$
0	0	0 ($Z=B_i$)
0	1	1 ($Z=B_i$)
1	0	1 ($Z=\overline{B}$)
1	1	0 ($Z=\overline{B}$)

Al restar, el valor de $\mathbf{C'_B}$ *se obtiene en las salidas de las compuertas X-OR.* A cada una de ellas llega por un lado una de las líneas de entrada $\mathbf{B_3,B_2,B_1,B_0}$, y por otro, la señal de control **M** (que para restar debe valer **1**).
Conforme a la tabla izquierda de una X-OR si **M**=0 la salida de cada X-OR repetirá el valor **0** ó **1** existente en la entrada $\mathbf{B_i}$ correspondiente; y si **M**=1 *invertirá* el valor de dicha entrada.

Esta inversión gobernada por **M** se ha querido indicar escribiendo $\mathbf{B_i/\overline{B}_i}$ en correspondencia con el valor **0**/1 (+/–) de **M**. Así, con el cable **M** se logra que los valores presentes en las entradas $\mathbf{B_3,B_2,B_1,B_0}$ se inviertan en la resta para obtener $\mathbf{C'_B}$, o que mantengan su valor en la suma.

Este cable también puede ir a la entrada $\mathbf{C_{-1}}$ del primer SC, de modo que en la resta (**M**=1) amén de formarse $\mathbf{C'_B}$ en las salidas de las X-OR, a este complemento $\mathbf{C'_3}$ se le sume al mismo tiempo el valor **1**, presente en el cable **M**, para obtener $\mathbf{C_B = C'_B + 1}$

Entonces, **en la suma** debe ser **M**=0, por lo que las salidas de las X-OR repetirán los valores existentes en $\mathbf{B_3,B_2,B_1,B_0}$, siendo además $\mathbf{M = C_{-1} = 0}$, por lo que el sumador realiza la operación:

$\mathbf{A(A_3,A_2,A_1,A_0) + B(B_3,B_2,B_1,B_0) + 0}$ de la forma vista anteriormente.

Para **restar** debe ser **M**=1, por lo que los bits del sustraendo $\mathbf{B(B_3,B_2,B_1,B_0)}$ se invertirán en las salidas de las **X-OR**, obteniéndose en ellas $\mathbf{\overline{B}(\overline{B}_3,\overline{B}_2,\overline{B}_1,\overline{B}_0) = C'_B}$; siendo además $\mathbf{M = C_{-1} = 1}$, por lo que el sumador efectuará de la forma vista:

$$\mathbf{A(A_3,A_2,A_1,A_0) + \overline{B}(\overline{B}_3,\overline{B}_2,\overline{B}_1,\overline{B}_0) + 1 = A + C'_B + 1}$$

Esto es, en la resta las **X-OR** generan el complemento a uno de **B**, al cual se suma 1 por la entrada $\mathbf{M = C_{-1}}$, obteniéndose así el complemento a la base $\mathbf{C_B}$ que se suma al minuendo **A**

Generación de los indicadores ("flags") SZVC

El indicador de *signo* **S** del conjunto **SZVC** (sección N.3) es el bit extremo izquierdo de la suma, sin considerar el último acarreo, o sea en este caso, con 4 bits, $\mathbf{S = S_3}$, por lo que será el mismo cable de $\mathbf{S_3}$. Del mismo cable de $\mathbf{C_3}$ se obtiene el indicador **C** de **carry,** por ser éste por definición el último acarreo del sumador.

La indicación **Z=1** de *resultado cero* (en este caso $\mathbf{S_3=S_2=S_1=S_0=0}$) se logra detectar si $\mathbf{Z = \overline{S_3+S_2+S_1+S_0}}$, expresión correspondiente a la compuerta NOR dibujada.

El indicador **V** de *overflow* resulta circuitalmente más sencillo de obtener si se hace la operación *X-OR entre los dos últimos acarreos*, en este caso $\mathbf{V = C_3 \oplus C_2}$, generado por la correspondiente compuerta dibujada en el circuito. Esto, conforme a la tabla de esa función, equivale a enunciar que **V=1** (sumandos con igual signo y resultado con signo opuesto) si $\mathbf{C_3}$ y $\mathbf{C_2}$ tienen valores opuestos, como se verifica a continuación para la suma de dos números positivos y negativos de 4 bits:

$\mathbf{C_2 \rightarrow}$ 1 $\quad\quad$ $\mathbf{C_2 \rightarrow}$ 0

$\mathbf{\underline{0}\ A_2\ A_1\ A_0}$ $\quad\quad$ $\mathbf{\underline{1}\ A_2\ A_1\ A_0}$

$\mathbf{+\ \underline{0}\ B_2\ B_1\ B_0}$ $\quad\quad$ $\mathbf{+\ \underline{1}\ B_2\ B_1\ B_0}$

$\mathbf{0\ \underline{1}\ S_2\ S_1\ S_0}$ $\quad\quad$ $\mathbf{1\ \underline{0}\ S_2\ S_1\ S_0}$

$\mathbf{\uparrow}$ $\quad\quad$ $\mathbf{\uparrow}$

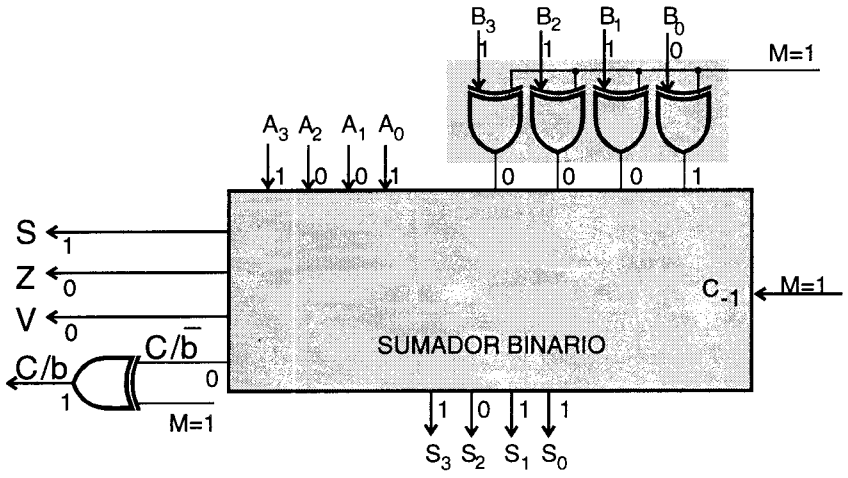
$\mathbf{C_3=C}$ $\quad$ **V=1** $\quad\quad$ $\mathbf{C_3=C}$ $\quad$ **V=1**

Para los casos en que **V=0**, resultan $\mathbf{C_3}$ y $\mathbf{C_2}$ *con igual valor,* como puede verificarse, con lo cual $\mathbf{V = C_3 \oplus C_2 = 0}$

En las dos sumas anteriores no importa los valores particulares de las variables $\mathbf{A_i,B_i}$ y $\mathbf{S_i}$; sólo interesa el valor de $\mathbf{C_2}$ generado por dichos valores.

Todo lo anterior puede generalizarse cualquiera sea el número de bits de los dos números a sumar o restar.

El sumador/restador de la figura A.96 puede esquematizarse conforme la figura A.97



⟵ Figura A.97

Se ha agregado una segunda X-OR, también comandada por el cable **M** = 0/1, para que en la resta se invierta el valor de **C**, a fin de obtener en su salida el **b**orrow (sección N.4). En la suma esta salida repite el valor de **C**[1]

Los valores en los cables de este sumador/restador, ejemplifican la operación **A–B** = (–7) – (–2) = 1001 – 1110.

Como en la resta debe ser $\mathbf{M} = \mathbf{C_{-1}} = 1$, el sumador realiza la operación de la derecha

A =	1001
C'B = +	0001
	1
	1011

Por lo tanto 1001 – 1110 = 1011 con **S**=1, **Z**=0, **V**=0 **C**=0

Verificación: 1010 = –(0100 + 1) = –0101 = –5

En definitiva, si se opera con números enteros a las entradas de un sumador/restador llegan los dos operandos con su bit de signo, y en las salidas se genera el resultado con bit de signo.

A1.19 Circuito Decodificador

Este circuito forma parte de cualquier RAM o ROM, dado que gracias al mismo una u otra son memorias "random", al azar. Cuando en sus **n** entradas se coloca una combinación de **n** bits (dirección), **una** sola de sus 2^n salidas se activa, permitiendo seleccionar la posición o celda de 8 bits direccionada. Como el circuito responde con igual retardo para cualquier combinación existente en sus entradas, todas las posiciones se localizan en igual tiempo, característica que define el llamado acceso "random", al azar.

> Un circuito ***decodificador*** *("decoder")* completo permite reconocer qué combinación binaria está presente en sus n entradas –entre 2^n posibles– según cuál de sus 2^n cables de salida está activado

Esto es, cada vez que en las entradas de un decodificador se aplica una cierta combinación o número binario, en correspondencia sólo ***una*** determinada línea de salida *se activa, permaneciendo las restantes salidas sin activar.*

Cada combinación distinta activa **una** línea de salida específica que la **identifica**.

Si se designa cada cable de salida con un número decimal *igual* al número binario que lo activa, podrá detectarse ("decodificarse") la combinación binaria presente en las entradas. En la figura A.98.b aparece un decodificador de n=3 entradas, y 2^3=8 salidas, y en la figura A.98.a su tabla de verdad.

Suponiendo que en las entradas **A,B,C** exista la combinación 110, se debe activar la salida $\mathbf{Z_6}$ mientras esté presente dicha combinación, y las restantes salidas estarán desactivadas. El subíndice **6** de **Z** permite deducir que está presente la combinación 110

[1] Esta salida que debiera llamarse (**C/b**), en la práctica se llama *indistintamente* **C** para la suma o la resta, debiéndose discriminar si se trata de **b** ó **C**, según la operación ordenada al circuito sumador/restador

Del mismo modo, cualquier combinación aplicada en las entradas, da lugar a la activación del correspondiente cable de salida que permite identificarla.

A	B	C	Z_7	Z_6	Z_5	Z_4	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	0	0	0	0	0	**1**
0	0	1	0	0	0	0	0	0	**1**	0
0	1	0	0	0	0	0	0	**1**	0	0
0	1	1	0	0	0	0	**1**	0	0	0
1	0	0	0	0	0	**1**	0	0	0	0
1	0	1	0	0	**1**	0	0	0	0	0
1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	**1**	0	0	0	0	0	0	0

Figura A.98.a

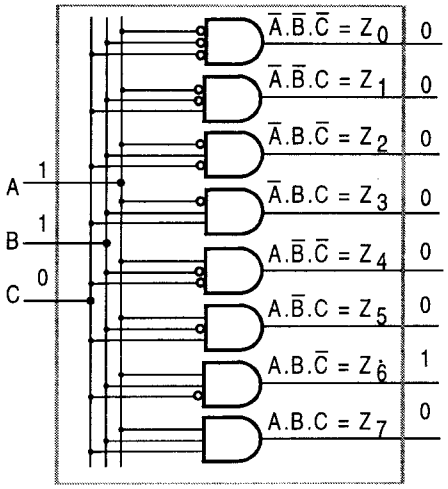


Figura A.98.b

La tabla del decodificador muestra que cada función de salida tiene un solo "uno", por lo que se trata de una AND de 4 variables, y hay un "uno" por renglón. O sea que cada función se expresa mediante un solo minitérmino específico. Conforme con esto, resulta el circuito con 8 compuertas **AND**, dibujado dentro del bloque de la figura A.97, **el cual genera todos los minitérminos de una función** de 3 variables.(sección A1.16)

> *En general, un decodificador completo de **n** entradas consta de 2^n compuertas AND, de **n** entradas. Cada salida de una de ellas es una salida del circuito.*

La combinación 110 supuesta en las entradas, llegará a las entradas de todas las AND, pero sólo $\mathbf{Z_6}$ resultará **1**, dado que $\mathbf{Z_6 = A.B.\bar{C} = 1.1.\bar{0} = 1}$

Para este tema se debe tener presente las relaciones establecidas en la sección A1.16

Se comprende que si el decodificador de una memoria principal tiene 20 entradas, correspondientes a una capacidad de 2^{20} = 1 Mbyte, las direcciones tendrán 20 bits, y para cada dirección una sola de las 2^{20} (≈ 1 millón!) de salidas del decodificador, para seleccionar la posición direccionada.

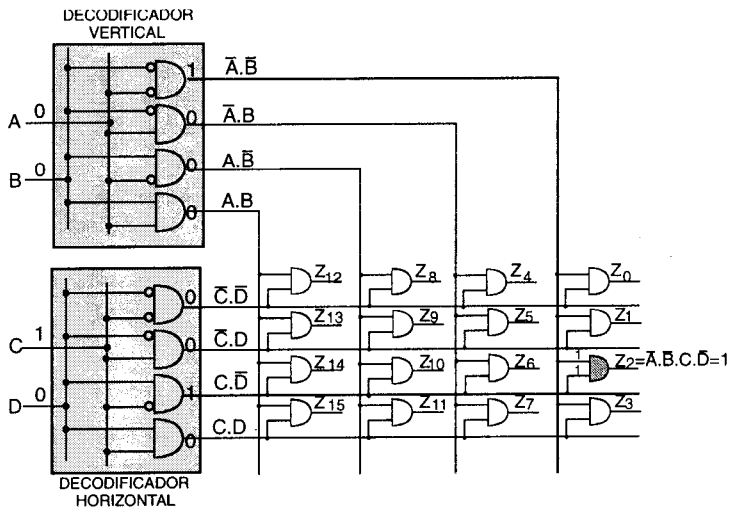


Figura A.99

Debe consignarse al respecto, que por razones tecnológicas, el decodificador de una pastilla de memoria está dividido en dos (un "horizontal" y otro "vertical"), cada uno de los cuales por ejemplo con la mitad de los bits de cada dirección, requiriéndose en el interior de la pastilla una compuerta AND extra por posición, para tener el efecto final de un solo decodificador sin dividir. En la figura siguiente se ejemplifica cómo formar un decodificador de 4 entradas y 16 salidas me-diante dos decodificadores de 2 entradas.

De lo que se trata en todos los casos es generar en cada salida el producto minitermino correspon-diente, formando productos completos a partir de productos parciales.

A1.20 Selector de datos/Multiplexor

Se trata de un circuito ampliamente usado en los sistemas de computación, en particular en teleprocesamiento, o cuando se manejan muchas terminales en un equipo.

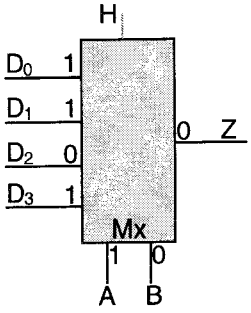


Figura A.100

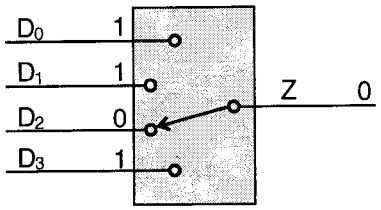


Figura A.101

A	B	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Figura A.102

Un circuito ***selector de datos/multiplexor*** (figura A.100) presenta **k** entradas (D_0,D_1,....) de **datos** (k = 2^n es una potencia de dos), **n** entradas de **selección** (A,B,C,....)[1] y una salida **Z**.

De manera similar a la operación de un dial o llave selectora mecánica (figura A.101):

> Un ***circuito de datos/multiplexor*** [2] permite seleccionar el valor lógico que aparecerá en su salida, entre los valores lógicos presentes en sus **k** entradas de datos

Para tal fin en las **n** entradas de *selección* debe aplicarse un número binario igual al subíndice decimal de la línea de datos que se quiere seleccionar.

Por ejemplo, si en las entradas de selecciónde l circuito de la figura A.100 se aplica la combinación **A,B**=10, la salida **Z** resultará **0**, correspondiente al valor supuesto para la entrada D_2, o sea será $Z = D_2 = 0$.

En general el valor 0/1 de la salida es el mismo que el valor 0/1 que en un instante dado presenta la entrada de datos seleccionada mediante la combinación aplicada en las entradas de control. Resulta así la tabla de verdad reducida de la figura A.102 del selector /multiplexor de la figura A.100, ampliada de la figura A.103. En ésta con "X" se indica un valor que puede ser 0 ó 1, no interesa, pues no aparecerá en la salida.

A	B	D_3	D_2	D_1	D_0	Z
0	0	X	X	X	**0**	$0=D_0$
0	0	X	X	X	**1**	$1=D_0$
0	1	X	X	**0**	X	$0=D_1$
0	1	X	X	**1**	X	$1=D_1$
1	0	X	**0**	X	X	$0=D_2$
1	0	X	**1**	X	X	$1=D_2$
1	1	**0**	X	X	X	$0=D_3$
1	1	**1**	X	X	X	$1=D_3$

Figura A.103

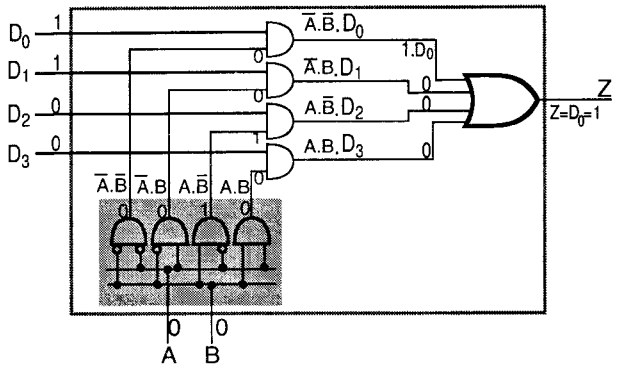


Figura A.104

La figura A.104 muestra un circuito multiplexor, basado en un decodificador cuyas entradas son las de selección **A,B**. Cada AND que va la OR de salida recibe una entrada de datos y una salida del decodificador. Puesto que en un

---

[1] Estas entradas también se llaman de *control* o de *"direccionamiento"*, apareciendo a veces designadas A_0,A_1,A_2, ... Las líneas D_0,D_1,... también se denominan ***"vías"***

[2] Por lo general un multiplexor es un selector de datos particular, en el cual las entradas de datos se seleccionan en orden rotativo-cíclico durante un tiempo igual para todas, por ejemplo siguiendo siempre la secuencia: $D_0 \rightarrow D_1 \rightarrow D_2 \rightarrow D_3 \rightarrow D_0 \rightarrow D_1 \ldots$

decodificador una salida vale 1 por vez, una sola de esas AND repetirá en su salida el valor 1/0 de la entrada **D** de datos que entra a esa compuerta (**1.D=D**). Las restantes AND estarán con su salida en 0. De esta forma, la salida **Z** presentará el valor 0/1 de la entrada de datos habilitada por el decodificador. Con los valores supuestos en las entradas, para la combinación 00 presente en **A,B**, será **1** la salida $\bar{A}.\bar{B}$ del decodificador y las restantes serán **0**.

Por lo tanto en la **AND** de salida $\bar{A}.\bar{B}.D_0$ será : $1.D_0 = 1.1 = 1$, y las otras AND estarán en **0**.

Resulta: $Z = D_0 + 0 + 0 + 0 = D_0 = 1$

La salida **Z** del multiplexor será $Z = \bar{A}.\bar{B}.D_0 + \bar{A}.B.D_1 + A.\bar{B}.D_2 + A.B.D_3$

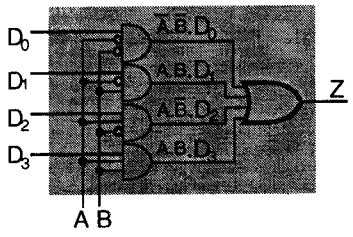
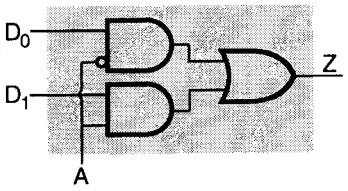


Figura A.105

Cada producto de las variables de selección, **A,B** aparece multiplicado por la variable de subíndice decimal igual a la combinación binaria que hace **1** ese producto. Se verifica que si se aplica a las entradas **A,B** la combinación 00 resulta

$$Z = \bar{0}.\bar{0}.D_0 + \bar{0}.0.D_1 + 0.\bar{0}.D_2 + 0.0.D_3 = 1.D_0 + 0.D_1 + 0.1.D_2 + 0.D_3 = D_0$$

En la figura A.105 aparece la forma más simple y corriente del circuito de un selector/ multiplexor que cumple con la expresión de **Z** antes hallada. Básicamente cada producto se genera con una sola AND que reemplaza a dos de la figura A.104



El selector más sencillo sería uno para dos entradas o vías (figura A.106), con una entrada de selección **A**. Según sea **A**=0 ó **A**=1, se selecciona D_0 ó D_1, respectivamente.

←Figura A.106

A1.21 Demultiplexor

Un ***demultiplexor*** realiza la función inversa de un multiplexor:
el valor lógico de la línea de entrada puede ser seleccionado, mediante **n** entradas de selección, para que aparezca en una de las $k = 2^n$ líneas de salida.

La figura A.107 muestra un demultiplexor de 4 salidas, que cumple una función semejante a la llave mecánica de la figura A.108, que distribuye cada bit que presenta en el transcurso del tiempo la línea **D** entre las líneas Z_0 a Z_3, cambiando secuencialmente de posición en los instantes $t_0, t_1, t_2, t_3 \ldots$

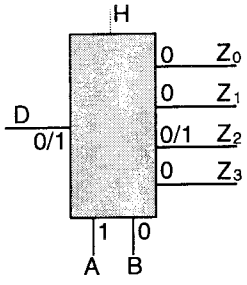


Figura A.107

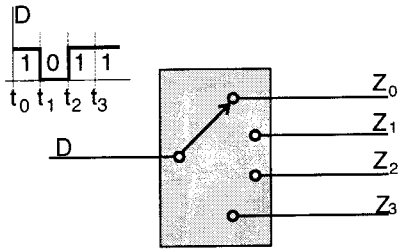
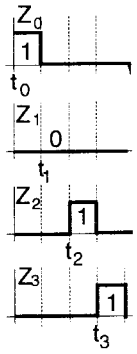


Figura A.108



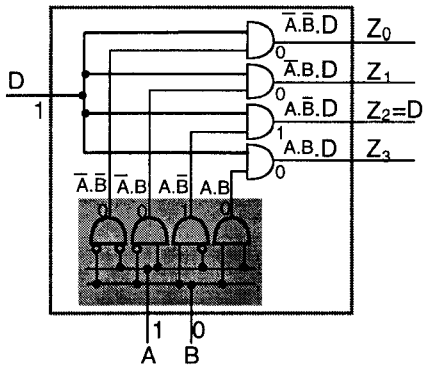
Aplicando en las entradas de selección **A,B** un número binario, se habilita la salida que tiene como subíndice decimal el mismo número. El valor 1/0 de la entrada **D** aparecerá en dicha salida (Z_2 en la figura A.107 para la combinación 10 presente en **A,B**) Las restantes salidas tendrán valor 0.

En la figura A.109 se muestra el circuito de dos entradas de selección, cuya tabla de verdad aparece en la figura A.110, en la cual "**X**" significa que no importa si vale 1 ó 0.

D	A	B	Z_0	Z_1	Z_2	Z_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Figura A.110

Figura A.109 ⟶



A1.21 Generador/Verificador de paridad

Se trata de un circuito lque se emplea en teleprocesamiento, en cintas magnéticas, y en la verificación de operaciones de lectura en la memoria principal, entre otras aplicaciones donde *se supone que por motivos del ruido sólo un bit puede cambiar mientras se transmite un mensaje de un punto a otro.*

Si se quiere enviar por un cierto canal de transmisión la información combinaciones de 000 a 111, por ejemplo la 010, y por algún ruido en la transmisión se recibe con un bit de error 001 (un uno se transformó en cero), como esta combinación es del mismo código, el sistema receptor la aceptará como tal, sin "darse cuenta" que no es la combinación originaria.

En general esta situación se repetirá *siempre que una combinación de un cierto código se convierta en otra del mismo,* por error de inversión de uno o más bits durante su transmisión.

Como se verá a continuación,un ***código detector*** o ***"autoverificador"*** está pensado para que los errores que tengan lugar en una combinación correctamente originada, se detecten por que **convierten** dicha combinación en otra **que no** pertenece al código.

Así, en la figura A.113, a las combinaciones citadas de 000 a 111 se les agregó un bit **Z** tal que cualquier combinación de la tabla tenga un número par de unos[2] (paridad par de unos). El código originario de 3 bits (**A,B,C**) *se ha transformado* en otro de 4 bits: **A,B,C,Z.**

De esta forma, si luego de que el transmisor generó una combinación del código de 4 bits se invierte un bit, el número total de unos aumentará o disminuirá en una unidad, perdiéndose la paridad par de unos originaria. Si ahora 0101 por un ruido en el canal de transmisión se transforma en 0010, al llegar al receptor esta combinación será detectada como **no perteneciente al código** de la figura A.113, **por no tener la paridad par que deben tener todas las combinaciones del código,** y por lo tanto será rechazada, *pidiéndose por lo general al transmisor que retransmita el mensaje*; **será rechazada** por no presentar un número par de unos.

Entonces, las combinaciones que se reciban con paridad impar de unos **no** serán del código, pues se supone que transmisor genera combinaciones del código.

Para reconocer en el receptor las combinaciones del código puede usarse un circuito verificador de paridad, que se trata (fig A.144) junto con el generador del bit de paridad.

Se determinará un circuito (que será el de la figura A.111) denominado ***"detector de paridad"*** que sigue la tabla de la figura A.113. O sea si la combinación binaria presente en sus entradas **A,B,C** presenta un *número impar* de unos, la salida **Z** será **1**; y si el número total de unos en **A,B,C** es par, debe ser **Z** = 0. Esto es, la salida vale **1** si la combinación de 3 bits que existe en las entradas tiene paridad impar de unos.

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Figura A.113

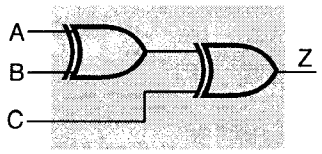


Figura A.111

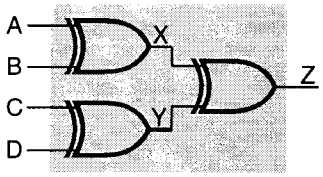


Figura A.112

De la figura A.113 resulta **Z** como la siguiente suma de minitérminos:

[2] También se podía haber elegido que sea **Z**=1 si se detecta número par de unos, o bien que la detección de la paridad impar o par sea respecto de los ceros.

$Z = \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.C = (\overline{A}.\overline{B} + A.B).C + (\overline{A}.B + A.\overline{B}).\overline{C} =$

$= (\overline{A \oplus B}).C + (A \oplus B).\overline{C} = (A \oplus B) \oplus C = A \oplus B \oplus C$ [1] siendo el circuito correspondiente el de la figura A.111

Si se realiza una tabla para 4 variables resultaría:

$Z = (A \oplus B) \oplus (C \oplus D) = A \oplus B \oplus C \oplus D$ siendo el circuito resultante el de la figura A.112

Generalizando para **n** variables, la expresión que indica con **Z**=1 si el número de variables que valen **1** es impar es:

$$Z = A \oplus B \oplus C \oplus D \oplus E \oplus \ldots \oplus N$$

El circuito detector de paridad se designa ***"generador/verificador" de paridad*** *("parity generator/checker")* dado que típicamente tiene usos complementarios.

1. Con el fin de formar combinaciones a transmitir que tengan una paridad establecida. Para ello su salida **genera** el bit que agregado a cada una de las combinaciones de **n** bits presente en sus entradas, permite construir combinaciones de **n+1** bits, todas ellas con un número par (o impar) de unos (o ceros), según se convenga.
2. Para **verificar** que cada combinación recibida –que fue formada y transmitida según se indicó en el paso anterior– presente la paridad establecida. En este caso, la salida del detector es **0** si la paridad es correcta, y con **1** si no lo es.

þ **EJEMPLO:**

Suponiendo un código con combinaciones **A,B,C** de 3 bits como el de la figura A.113, la salida **Z** del circuito de la figura A.111 genera **1** si la combinación **A,B,C** tiene paridad impar.

Si el valor de **Z** se agrega a la combinación existente en las entradas, de modo de formar combinaciones **A,B,C,Z** de 4 bits, éstas tendrán siempre un número par de unos. Así pueden construirse las combinaciones de cada renglón de la tabla, las cuales en la figura A.114 se supone que se transmiten desde un extremo que hemos llamado "generador-transmisor".

Estas combinaciones son recibidas (figura A.114) en el extremo "receptor-verificador" (semejante al de la figura A.111), cuya salida Z_2 será **0** si la combinación **A,B,C,Z** existente en sus entradas presenta un número par de unos, como se ha convenido que debe ser la paridad transmitida y recibida. De ser $Z_2 = 1$ implica que la combinación recibida tiene un bit errado por no tener la paridad par de unos estipulada, en cuyo caso se pide la retransmisión de la misma.

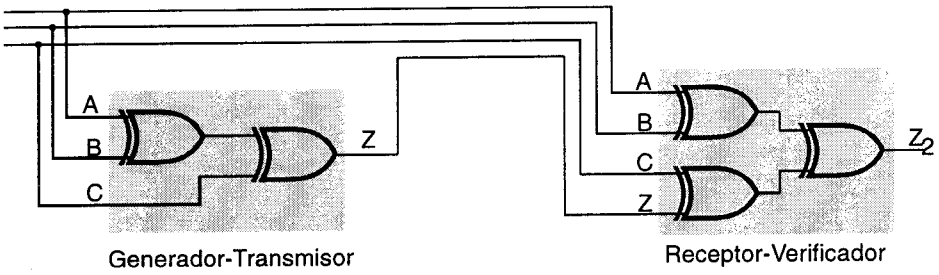


Figura A.114

Si bien el desarrollo se realizó para combinaciones de n = 3 bits, puede generalizarse cualquiera sea el número **n** de bits que tengan las combinaciones de un código.

En el caso de una memoria principal, al escribir cada posición (n = 8) se genera un noveno bit de paridad, o sea que *en cada posición se almacenan 9 bits*. En el momento de leer una posición se verifica la paridad, para determinar por la salida Z_2 si hubo o no error.

1 Ya en una compuerta X-OR la salida es 1 si **una** (número impar) de las entradas vale 1, y vale 0 si las **dos** entradas valen 1, o si ambas están en 0

A1.22 Memorias ROM

El circuito de una "memoria **ROM**" se construye mediante un decodificador (sección A1.19) y compuertas OR, siendo *en esencia un circuito combinacional de varias entradas y salidas que cumple con una determinada tabla de verdad.* El circuito se construye con el método visto de implementar circuitalmente funciones mediante suma de minitérminos, que ya para dos funciones se aplicó para hallar el circuito sumador completo.

A continuación se ejemplifica la construcción de una ROM correspondiente a una tabla con m = 4 funciones **P,V,W,Y** que comparten n=4 variables **A,B,C,D** pero que puede generalizarse para tablas con los valores de **m** y **n** que hagan falta.

A	B	C	D	P	V	W	Y
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	0	1	0
0	0	1	1	1	0	1	0
0	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	1	0	1	1
1	0	0	0	0	1	0	0
1	0	0	1	1	1	0	0
1	0	1	0	0	1	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	1	1	0
1	1	0	1	1	1	1	0
1	1	1	0	1	0	0	0
1	1	1	1	1	1	1	1

Figura A.115

Cada función **P,V,W,Y** se construirá mediante la suma de sus minitérminos, como se hizo entre las figura A.94 a y b. Ahora las compuertas AND del decodificador proveerán los minitérminos que necesitan dichas funciones, por generar las AND todos los minitérminos de 4 variables (como aparecen en la figura A.118). Se requiere una compuerta OR por función para efectuar esa suma, como se indica. Una salida del decodificador a la que se conectan varias entradas de compuertas OR, implica que las funciones correspondientes comparten el minitérmino que ella genera, o sea que presentan el mismo en la sumatoria que expresa dichas funciones.

Cada OR tendrá tantas entradas como minitérminos debe sumar.

Los circuitos ROM son de conexionado interno programable o **"dispositivos lógicos programables"**, en el sentido de que la conexión entre las salidas de los circuitos AND (del decodificador), y las entradas de las OR es configurable *("programable")* por el usuario

Vale decir, **una ROM es un dispositivo lógico programable por el usuario, para que cumpla con una determinada tabla de verdad.**

Las conexiones citadas se indican como puntos de intersección posibles de una matriz de líneas horizontales y verticales conductores (figura A.117). Un punto o una cruz en una intersección implica que las líneas en cuestión están conectadas formando un solo conductor. Puesto que un decodificador de **n** entradas tiene 2^n salidas, y las entradas de cualquier OR se deben poder conectar a cualquiera de dichas salidas, se deduce que **cada OR también debe tener 2^n entradas**. A fin de simplificar el dibujo se usa la representación de la derecha de la figura A.117, en la cual cada línea de una OR representa sus 2^n entradas reales.

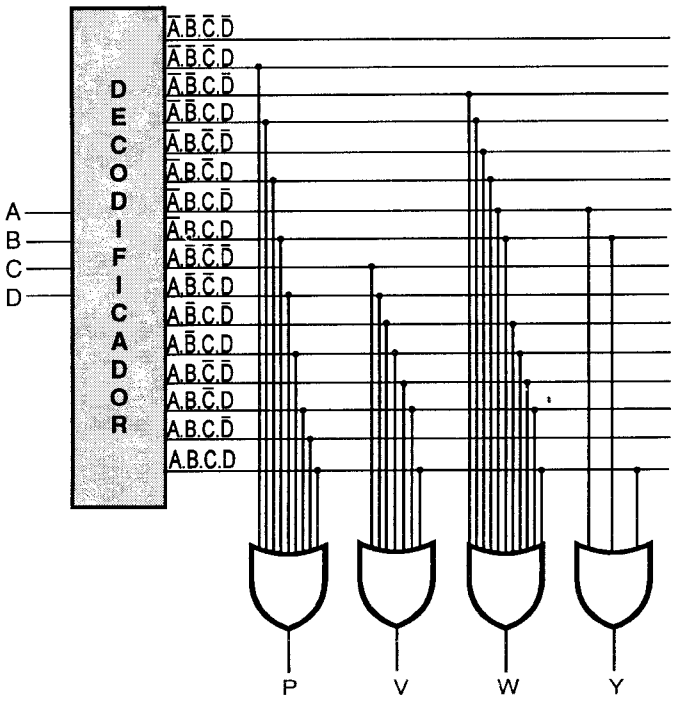


Figura A.116

Con esta convención, el circuito de la figura A.116 puede representarse como en la figura A.118, donde cada "cruz" sobre una misma vertical implica una entrada distinta de la OR en cuestión conectada a la salida correspondiente del decodificador.

O sea que en esta representación, una OR tiene tantas entradas como intersecciones posibles tenga la línea que llega a ella

Así, las 9 cruces de intersección a la línea que va a la OR que genera **P**, simbolizan 9 entradas de esa OR, que están conectadas a 9 salidas del decodificador, como se dibujó en la figura A.118 Las cruces corresponden a los "unos" que cada función presenta (figura A.115)

Si en las entradas de la ROM se aplicara la combinación 0000, **solamente** la salida del decodificador $\mathbf{m_0 = \bar{A}.\bar{B}.\bar{C}.\bar{D}}$ valdrá **1**. Como en las intersecciones de la línea horizontal $\mathbf{m_0}$ con las verticales de las **OR** no hay ninguna cruz, significa que ninguna entrada de las **OR** está conectada a la línea $\mathbf{m_0}$, para recibir el "uno" que ella genera. En consecuencia, para dicha combinación 0000, las cuatro salidas estarán en **0**.

Suponiendo que en **A,B,C,D** se tenga luego 0110, será: $\mathbf{m_6 = A.B.C.\bar{D} = 1}$, por lo que dicho **1** se comunicará a las correspondientes entradas de las OR que generarán **W** e **Y**, dado que existen "cruces" de conexionado en la intersección de

la línea horizontal de $\mathbf{m_6}$ con las verticales que van a esas compuertas. Luego, en las salidas **P,V,W,Y**, se tendrá 0011 para las entradas 0110.

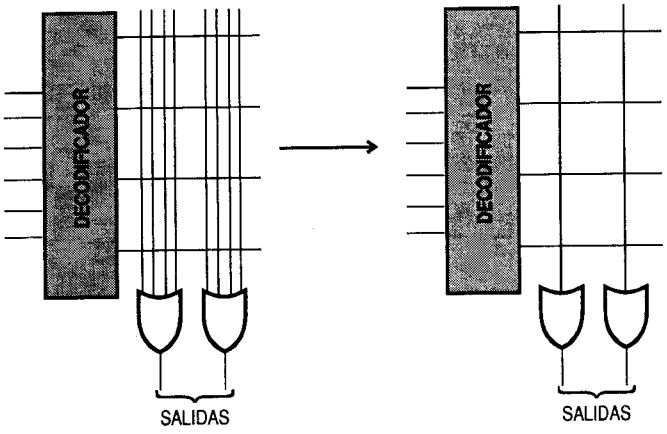


Figura A.117

Programación de una ROM

Las salidas del decodificador con las entradas de cada OR determinan una "matriz programable" El usuario puede decidir en ella, a qué OR irá conectada cada salida.

En este sentido se trata de un dispositivo de lógica programable por el usuario, quién deberá indicar en una tabla de verdad, qué combinación binaria debe darse en las salidas para cada combinación de bits en las entradas. O sea qué conexiones debe tener la matriz para que ello tenga lugar

Un tipo de ROM corriente viene de fábrica con todas las conexiones de la matriz realizadas (figura·A.119), por lo que el proceso de "grabación" consiste en destruir (mediante corrientes eléctricas específicas generadas por un dispositivo especial) aquellas conexiones que no deben existir, a fin de cumplir con la tabla citada. Al respecto obsérvese (fig A.118) que, por ejemplo, la combinación de entrada 0110, que hace m_0 =1, genera un **1** en una salida si existe una conexión (cruz) en la matriz; y genera un **0** si no existe conexión alguna.

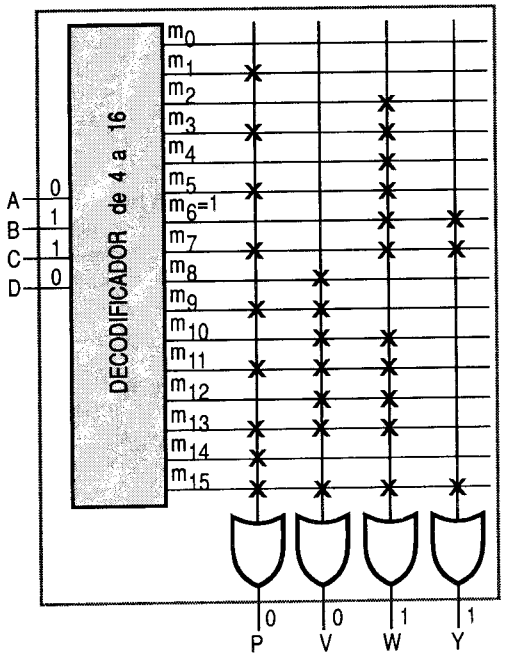


Figura A.118

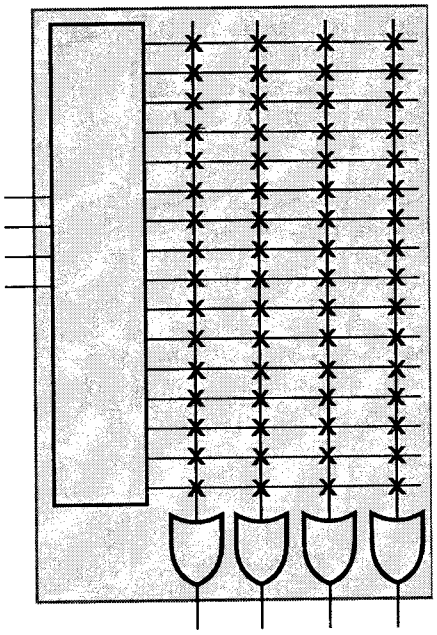


Figura A.119

Tipos de ROM:

Conforme al grado de reusabilidad que admite la matriz de conexionado, las ROM pueden clasificarse como sigue:

1. ***ROM programables una sola vez o inalterables***
   Una vez realizado el conexionado programado por el usuario, no se puede modificar.

   1.a ROM con conexionado a pedido *("custom programmed ROM" o "masked ROM")*: el usuario programa las conexiones que tendrá la matriz, siendo el fabricante de la ROM quién realiza el conexionado programado en un proceso complementario a la fabricación del chip. Su costo sólo se justifica para un consumo masivo de tal chip, tardando cierto tiempo el fabricante en entregarlo

   1.b PROM ("Programable ROM"): el usuario además de especificar la programación de la matriz, lleva a cabo la misma (o lo encarga a quién provee ese servicio), en un proceso posterior a la fabricación del chip.

2. ***ROM reprogramables (RPROM)*** [1]: una vez que el usuario configuró la matriz de conexionado, puede modificarla, de modo de cambiar la tabla que cumple la ROM.

   2a. EPROM *("Erasable-Programmable ROM")*: antes de reconfigurar la matriz, se debe hacer incidir luz U.V. por una "ventana" transparente en la cápsula del chip. Esta acción de borrado dura 15'; luego puede reprogramarse

   2.b EEPROM o E^2PROM *("Electrically Erasable PROM")*[2] : son EPROM cuyo borrado se hace eléctricamente, sin luz UV, hasta 10.000 veces, pudiéndose seleccionar la palabra que se quiere borrar y reescribir, sin borrar toda la ROM. Además esto puede hacerse sin sacar la pastilla ROM del circuito.

   2.c "Flash" ROM es un tipo de EEPROM, cuyo borrado eléctrico es total (como la EPROM), sin selección.

La ROM como memoria

Según se vió, una ROM *circuitalmente* es una *lógica* ***combinacional*** AND-OR, sin ningún lazo de realimentación entre salidas y entradas, con la particularidad de que en el circuito integrado puede configurarse el conexionado entre las compuertas AND y OR. Como tal se ejemplificó la figura A.118, donde las AND están en el decodificador

Una vez configurada la matriz de conexionado, como todo circuito combinacional, para cada combinación presente en sus entradas, responde en sus salidas siempre con una combinación determinada, conforme a una tabla. O sea transforma cada combinación de entrada en otra de salida que le corresponde unívocamente según dicha tabla. Los unos y ceros de las funciones de salida de esta tabla pueden verse como almacenados en la matriz de conexiones, como surge de comparar las figuras A.115 y A.118

Así, puede pensarse que esa información se almacenó cuando se programó la ROM, la cual obviamente se mantendrá *aunque se corte el suministro de energía* a la pastilla.

Un usuario que no conozca la estructura lógica interna de una ROM de **n** entradas y **m** salidas, puede imaginarla como una caja negra donde están memorizadas las 2^n combinaciones binarias de **m** bits cada una, pudiendo aparecer cada una de ellas en las salidas, cada vez que se aplica en las entradas la combinación correspondiente.

Para él, las combinaciones que aplica en las entradas de la ROM son ***"direcciones"***, que permiten localizar directamente –sin demora por búsqueda alguna– la combinación que aparecerá en las salidas.

O sea que todo sucede como si mediante una *"operación de lectura"* de una memoria, consistente en aplicar una *dirección* en las **n** entradas, se puede acceder a una combinación de **m** bits *("palabra")* almacenados en la ROM, que aparecerá en las **m** salidas, correspondientes a esa dirección.

Considerada como una memoria, la operación de configurar una ROM, o de reconfiguración, se denomina ***"escritura"***

Entonces, una ROM de 11 entradas y 8 salidas que cumpla con la tabla de la figura A.119, almacenaría las combinaciones de 8 bits que aparecen en las 8 columnas de valores de las funciones de salida. Al lado de cada grupo de 8 bits, aparece la "dirección" que permite "localizarlo", o sea la combinación que debe aplicarse en las entradas de la ROM, para que 8 bits seleccionados aparezcan en las salidas.

Si por ejemplo, en las 11 entradas se direcciona 00000000100, luego del tiempo de respuesta del circuito, se habrá leído el contenido de memoria 01111101, correspondiente a esa dirección.

Dirección	Contenido	
00000000000	0 0 1 0 1 1 1 0	↑
00000000001	0 0 0 1 1 1 0 1	
00000000010	0 1 1 1 1 0 0 1	2^{11}
00000000011	0 1 0 1 1 0 0 1	posiciones
00000000100	0 1 1 1 1 1 0 1	
.....	. . . .	↓

Entonces, figura A.119, un usuario puede visualizar la tabla de la ROM como un conjunto de 2^n ***"posiciones"***, cada una *conteniendo* una ***"palabra"*** (en este caso de 8 bits), y con una dirección que permite seleccionarla, esto es, localizarla en la lectura y escritura. La acción de seleccionar una posición para leer su contenido o para reconfigurarlo (escritura) se conoce como *"direccionamiento"* de la memoria.

⟵ Figura A.119

Dado que las primeras ROM no eran reprogramables, una vez escrita sólo podían ser leídas ("read only"). Actualmente la denominación **RMM** es más apropiada para las reprogramables.

En una clasificación de memorias, las ROM serían un caso particular de ***"Random Acces Memory"*** **(RAM)**, o sea *"memorias de acceso aleatorio"*, dado que el tiempo que insume la lectura de información contenida en una ROM (tiempo de acceso) es el mismo, cualquiera sea la dirección colocada al azar en las entradas. Por lo tanto, dicho tiempo es independiente de la posición dónde se encuentra la información.

El tiempo de respuesta del circuito de una ROM, que media entre la aplicación de una "dirección" en las entradas y la aparición en las salidas de la correspondiente combinación "contenida" en la memoria, se denomina ***"tiempo de acceso"***.

Caracterización de una ROM como memoria de 2^n (palabras) x m (bits)

Una ROM se caracteriza por una determinada ***"capacidad*** *de almacenamiento"* que en general depende de la cantidad de posiciones o palabras direccionables (o sea de direcciones) y del número de bits que contiene cada posición

---

[1] También se denominan **"RMM"***("Read Mostly Memory")*, esto es memorias mayormente leídas, dado que la escritura (programación) es poco frecuente frente a la operación corriente de lectura de las salidas.

[2] A veces aparecen como **EAROM** *("Electrically Alterable ROM")*

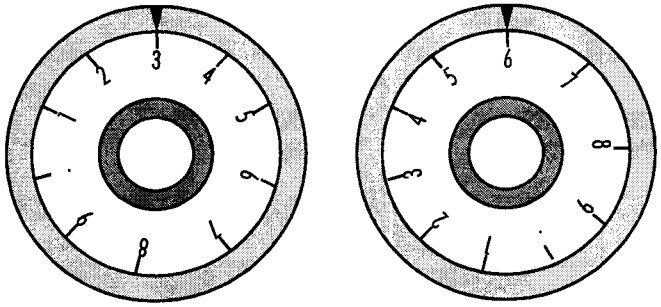
Volviendo a la ROM de la figura A.119, la misma tendrá $2^n = 2^{11} = 2048$ posiciones o palabras de m=8 bits cada una. Esto se expresa **2048x8**, o sea que en total se almacenarán (2048x8) bits = 49152 bits, (y por lo tanto vendrá con 49152 fusibles o transistores en la matriz); y en general:

Una ROM de **n** entradas y **m** salidas tendrá una **capacidad** de 2^n x **m** **[bits]**

Con n=15 entradas, y palabras de 8 bits tendría una capacidad dada por (2^{15} x 8) bits = (16384 x 8) bits = 16384 bytes.
Puesto que la unidad de capacidad es el número $1024 = 2^{10} = 1K$, respecto a esta unidad, la ROM anterior tendrá: 16384/1024 bytes = 16 Kbytes

A1.23 Dispositivos combinacionales y secuenciales

Se compararán dos tipos de cerraduras de seguridad que operan con información digital decimal. La "N° 1" consiste en dos perillas giratorias con números del 0 al 9 (figura siguiente), para formar las combinaciones 00 a 99, y se abre con la combinación 82. Las restantes combinaciones no la abrirán.



Otra cerradura, la "N° 2", presenta también dos perillas, pero para abrirse requiere la aplicación sucesiva de las combinaciones 37 y 82.
Por lo tanto, en un caso cada vez que se establezca la combinación 82 se abrirá la "N°1", mientras que para la "N° 2" ello no basta; se debe aplicar antes la 37.

Decimos que la cerradura "N°1" es ***"puramente combinacional"*** o *"combinatoria"* o *"sin memoria"*. Se caracteriza por responder de igual manera cada vez que se aplica la misma combinación. Esto es, siempre se abrirá para la 82 y permanecerá cerrada para las otras combinaciones.

En cambio la "N°2" es de tipo ***"secuencial"*** o ***"con memoria"*** o *"por pasos"*, pues su respuesta depende del orden o **secuencia** con que se aplican las combinaciones, y no sólo de la combinación existente en un determinado momento. Esto supone que debe poder retener o *memorizar* por lo menos en este caso, la combinación anterior a la presente, y en general el orden o "historia" en que aparecen las combinaciones en el tiempo. Entonces para una combinación presente, como la 82, puede responder de diferente forma: abriéndose si antes existió la 37, o permaneciendo cerrada si la anterior no fue 37.

Del mismo modo es lícito pensar que una comunicación telefónica depende de una secuencia de números discados y memorizados en un cierto orden. También una calculadora de bolsillo es secuencial, o sea posee memoria, dado que cada resultado alcanzado depende de la secuencia de pasos realizados antes de apretar la tecla igual.

El botón de un timbre no tiene memoria, pues toda vez que es accionado la respuesta es la misma. En cambio el botón de una perilla de velador posee memoria pues responde encendiendo la luz si antes de ser accionado estaba apagada, y apagándola si estaba encendida. Depende pues del efecto que él mismo produjo con anterioridad

*Hasta el presente hemos tratado con **circuitos sólo combinacionales**: para cada combinación binaria aplicada en sus entradas, unívocamente **siempre** da por resultado la misma combinación binaria en sus salidas, con absoluta independencia de otras combinaciones binarias recibidas anteriormente.*
*También puede decirse, que estos circuitos **transforman** unívocamente cada combinación binaria recibida en otra en las salidas, que es su respuesta.*

El funcionamiento así definido se manifiesta en el hecho de que no es posible que una misma combinación aplicada en las entradas de un combinacional en distintas oportunidades, produzca resultados diferentes, situación que se da en los circuitos con memoria, como se verá.

*Genéricamente, un componente o porción de un sistema tiene **memoria** si tiene la propiedad de permanecer en una cierta condición o estado luego de desaparecer la acción generatriz.*

En este sentido, son elementos de memoria los dominios magnéticos de una cinta magnética, que se mantiene orientados en una cierta dirección por efecto de una campo magnético, aunque éste deje de actuar. También lo son las marcas o perforaciones de una tarjeta producidos por un punzón o sacabocados. En el caso de los circuitos eléctricos la memoria se debe a que salidas de los mismos son también entradas, acción conocida como "realimentación"

A1.24 Memorias biestables ("Flip-Flops")

Una llave de electricidad de dos posiciones, que memoriza mecánicamente la última orden recibida (quedar abierta o cerrada) aunque deje de presionarse, constituye una memoria mecánica *de dos estados*, o sea ***"biestable"***.

También puede decirse que cuando está en "**si**" mantiene un **1**, y en "**no**" mantiene un **0**.

Del mismo modo, un circuito que presente dos estados claramente definidos en los que puede permanecer todo el tiempo que sea necesario, puede usarse como célula o ***unidad de memoria de un bit***, capaz de mantener, almacenar, un **1** o un **0**, según sea.

El cambio de un estado al otro se logra activando sus entradas mediante uno de los dos niveles de tensión **H** ó **L**.

Estos circuitos con dos estados puede servir como *unidades de memoria*, para construir a partir de ellos *cualquier tipo de bloque con memoria más complejo,* sean contadores, registros, memorias SRAM, etc, como se verá.

Existen principalmente dos tipos de memorias de un bit:

1. Los "**flip-flops**" o celdas de memoria biestables "***estáticas***": permanecen en uno u otro estado mientras no se corte el suministro de energía eléctrica al circuito. En ellos existe por lo menos un lazo de realimentación que proporciona un camino para que la salida de una compuerta se propague hacia una de sus entradas.
2. Las celdas de memoria biestables "***dinámicas***"[1]: se basan en el estado cargado o descargado de un capacitor que forma parte de un circuito con uno o más transistores. Dado que un capacitor no puede permanecer cargado indefinidamente, el mismo debe recargarse periódicamente (por ejemplo cada 4 mseg.), de modo de aportarle cargas eléctricas de *"refresco"*, para compensar las cargas que se pierden.

A1.25 Clasificación de los Flip Flops

Según en qué instantes pueden actuar sus entradas y cambiar de estado (y en correspondencia cambiar el valor de sus salidas), los flip flops pueden ser:

1. ***Asincrónicos:*** presentan dos entradas designadas **S** y **R** (figura A.120) que pueden actuar *en cualquier instante* para cambiar el estado del circuito, y en consonancia cambiar el valor de sus salidas, designadas **Q** y $\overline{\mathbf{Q}}$.[2] Se conocen con las siglas "**R-S**", por la denominación Reset y Set de sus entradas,que permiten hacer **Q=0** y **Q=1**, respectivamente. La salida $\overline{\mathbf{Q}}$ se genera para el caso que se necesite esa señal. Los flip flops asincrónicos se construyen simplemente interconectando dos compuertas NOR o NAND, como se verá.
2. ***Sincrónicos o temporizados:*** presentan una o dos entradas ($\mathbf{E}_1$ y $\mathbf{E}_2$ en la fig A.121)[3] *que pueden actuar sobre el estado del circuito sólo en determinados instantes, definidos en relación a pulsos que llegan por otra entrada de sincronismo* conocida como reloj o "clock" (**Ck**).

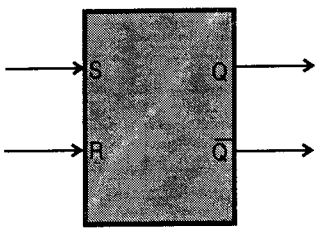


Figura A.120

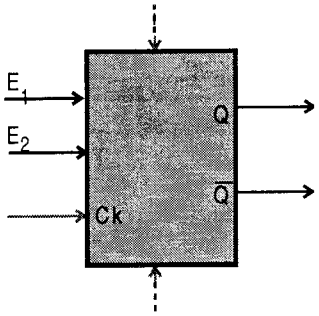


Figura A.121

La entrada **Ck** recibe pulsos de sincronismo que son generados por lo general a una recuencia regular por algún tipo de oscila dor, de donde deviene el nombre "reloj". Pueden existir una o dos entradas asincrónicas, o de "fuerza",indicadas en punteado, para obligar en cualquier momento que el flip flop presente uno de sus dos estados, independientemente de la presencia de pul sos de sincronismo. Esto es útil para fijar, por ejemplo, inicialmente el estado del circuito luego de energizarlo.

---

1 Un tipo de estas celdas de un bit constituyen las memorias dinámicas.

2 La salida $\overline{\mathbf{Q}}$ se genera para el caso de que se necesite esa variable, dado que "no cuesta nada" generarla, pues según se verá es intrínseca al circuito

3 $\mathbf{E}_1$ y $\mathbf{E}_2$ se designan típicamente en los CI como "**J** y **K**" o "**S** y **R**", (o "**D**" o "**T**" en caso de no existir $\mathbf{E}_2$)

A1.26 Funcionamiento de un biestable asincrónico

BIESTABLE "$\overline{\mathbf{R}}$-S" ASINCRONICO

En la figura A.122.a el cable de salida está conectado también a una entrada, por lo cual el valor de la salida se reinyecta, realimenta, a la entrada, y ésta actúa a su vez sobre el valor de la salida, formándose así un lazo cerrado. Un circuito así se conoce como ***"circuito realimentado"***.

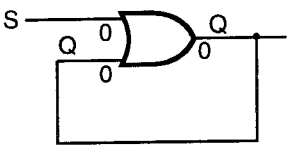


Figura A.122.a

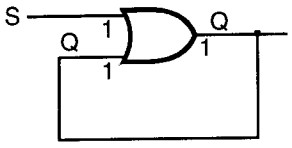


Figura A.122.b

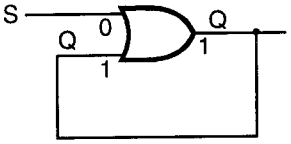


Figura A.122.c

Suponiendo que inicialmente el lazo no existe y que las dos entradas tienen el valor **0**, entonces el valor la salida de la compuerta OR sería **Q=0**. Si en esas condiciones se conecta la salida a una entrada, se forma el lazo dibujado en la fig. A.122.a, y el circuito permanecerá como en la situación inicial supuesta, con los valores lógicos indicados.

Ahora se supondrá que la entrada **S** toma el valor **1** (figura A.122.b), en cuyo caso luego del retardo de respuesta, el cable de salida de la OR valdrá **1**, valor que también se tendrá realimentado en la entrada conectada a dicho cable.

Aunque luego **S** valga otra vez **0** (figura A.122.c), esta situación con **Q=1** se mantendría indefinidamente, dado que en la otra entrada de la OR se está realimentando el valor **1**. La única manera de conseguir que vuelva a ser **Q=0** sería abriendo el lazo y procediendo como se hizo al inicio.

A continuación se verá una forma circuital de cambiar a voluntad el valor de **Q** mediante dos entradas de control.

Agregando una compuerta AND en el camino del lazo (figura A.123.a), si **S=0**, cuando la entrada designada $\overline{\mathbf{R}}$ vale **0**[1] se consigue que sea **Q=0**.

Si luego $\overline{\mathbf{R}}$ cambia a **1** (figura A.123.b), seguirá siendo **Q=0**, y este valor existirá a lo largo del lazo, lo mismo que si más tarde vuelve a ser $\overline{\mathbf{R}}$**=0**, como puede verificarse.

Vale decir, que siendo **S=0**, una vez que con $\overline{\mathbf{R}}$**=0** se logró hacer **Q=0**, esta situación persistirá aunque $\overline{\mathbf{R}}$ vuelva a valer **1**, mientras no cambie **S**.

Decimos que el circuito en esas condiciones se encuentra en un ***estado estable*** con **Q=0**, y que en su lazo quedo "*atrapado*", retenido, un cero, que se conservará mientras persista este estado.

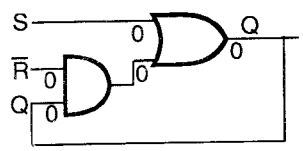


Figura A.123.a

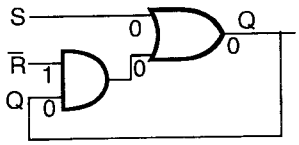


Figura A.123.b

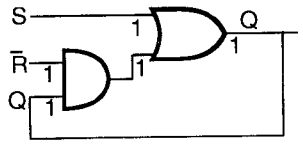


Figura A.123.c

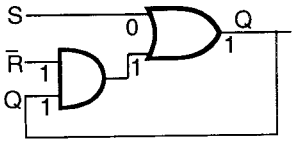


Figura A.123.d

Si encontrándose el circuito en este estado y estando $\overline{\mathbf{R}}$ desactivada ($\overline{\mathbf{R}}$**=1**), se hace **S=1** (figura A.123.c), la salida de la compuerta OR resultará **Q=1**, realimentándose el valor **1** a lo largo del lazo.

Se ha producido un ***cambio de estado***, quedando esta vez "atrapado" en el lazo el valor **1**.

Aunque luego **S** cambie a **0** (figura A.123.d) continuará el circuito en ese estado estable con **Q=1**, mientras $\overline{\mathbf{R}}$ no cambie a **0**. En caso de suceder esto último, el circuito cambiaría de estado nuevamente, para pasar al estado con **Q=0** (figura A.123.a)

La letra **S** (de "*Set*") identifica la entrada de *"puesta a 1"* que al activarse predispone que la salida **Q** se posicione en **1**, mientras que la **R** (de "*Reset*"), señala la entrada de *"puesta a 0"* que restaura o reposiciona **Q** en **0**.

Del circuito analizado, podemos extraer algunas conclusiones importantes:

Se ha verificado la existencia de ***dos estados*** en cada uno de los cuales el circuito permanece estable, en el sentido de que los valores lógicos que lo caracterizan se *autosustentan*, conservándose en el tiempo.

Por esta razón el circuito se denomina ***"biestable"*** pudiendo cambiar de un estado a otro en cualquier momento, si el valor de las entradas **S** y $\overline{\mathbf{R}}$ así lo determinan.

---

[1] Esta entrada se designa $\overline{\mathbf{R}}$ por que se activa con un **0**. La letra **S** (de "*Set*") señala la entrada de *"puesta a 1"* cuya activación da como resultado que la salida **Q** se posicione en **1**, mientras que la **R** (de "*Reset*"), señala la entrada de *"puesta a 0"* que restaura o reposiciona **Q** en **0**. El presente circuito puede identificarse con las siglas "$\overline{\mathbf{R}}$**-S**"

Podemos decir que en un estado se regenera permanentemente un cero, y en el otro un "uno", según sea.

Sensando (leyendo) el valor **0** ó **1** de la salida **Q** se puede determinar en qué estado se encuentra el circuito. También puede expresarse que si **Q=0** el circuito mantiene, memoriza[1], un cero, y si **Q=1** memoriza un "uno", dado que uno u otro valor se conservará mientras el circuito permanezca en uno de los dos estados.

BIESTABLE "R-S" ASINCRONICO

El biestable de la figura A.123 lo hemos designado "$\overline{\mathbf{R}}$ **-S**" por que la entrada **R** se activaba con **0**.

Para que se active con un nivel alto, a la entrada $\overline{\mathbf{R}}$ se le ha antepuesto una inversión (figura A.123.d) , por lo que la entrada se designa **R**. Asimismo en esta figura, a los fines de llegar al circuito más conocido del biestable "**R-S**" (figura A.123.f) se ha agregado en el lazo de realimentación una doble inversión, indicada por dos círculos extras. Esto no afecta la operación lógica del circuito, obteniéndose la variable **Q** negada. En la figura A.123.e se reemplazó la compuerta OR seguida del inversor por una NOR; y la AND de entradas negadas por otra NOR, como se deduce de aplicar De Morgan.

Con fines didácticos, en la figura A.123.d se han mantenido los valores de la figura A.123.c, en particular para verificar que de esta forma las salidas de las compuertas NOR serán siempre opuestas. Teniendo presente esto, en la figura A.123.f se ha redibujado la figura A.123.e, designando **Q** y $\overline{\mathbf{Q}}$ las salidas de las NOR, resultando así la forma simétrica típica de dibujar el biestable "**R-S**" asincrónico.

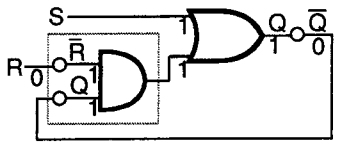


Figura A.123.d

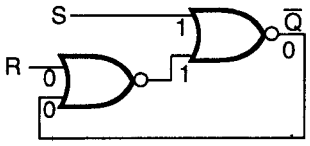


Figura A.123.e

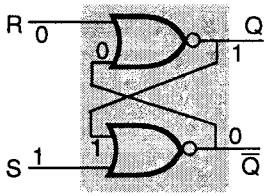


Figura A.123.f

A1.27 Principio de funcionamiento de un Flip Flop sincrónico "D" tipo "cerrojo"[2] ("latch")

El circuito de la figura A.124, puede verse como otra forma de solucionar el problema planteado en la figura A.122.c, en lo concerniente a obtener un circuito que permita cambiar a voluntad el valor de la salida **Q** y luego retener dicho valor, pero en sincronismo con pulsos que recibe por su entrada **Ck**.

La figura A.125 indica el bloque que lo simbolizará.

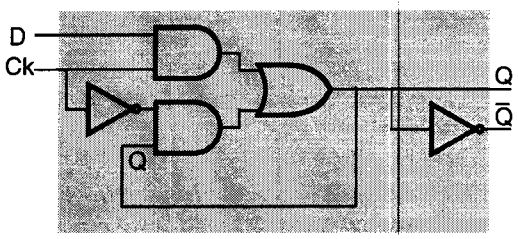


Figura A.124

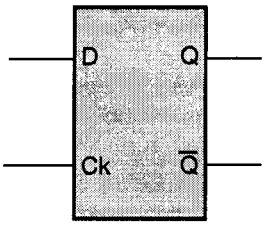


Figura A.125

Se trata de un multiplexor con dos entradasa de datos (figura A.106) cuya entrada de selección es controlada por una señal designada **Ck** (*"clock"*) citada en la sección A.25
A una de las entradas de datos llega la señal **Q** realimentada, y a la otra designada **D**, el valor que se quiere memorizar.

El tiempo establecido para presentar en la entrada **D** el valor a memorizar está dado por la duración del nivel alto de **Ck**; y el tiempo que debe mantenerse ese valor en el lazo de realimentación está determinado por la duración del nivel bajo de **Ck**. Los cambios de nivel de **Ck** marcan los tiempos en que se pasa de una situación a la otra.

1 Otros sinónimos pueden ser "retiene", "almacena"

2 "Cerrojo" ("*latch*") hace referencia al hecho de encerrar, trabar, un bit. También se usa en el sentido de trabar una compuerta para que su salida no cambie.

Así resulta que la operación del circuito está regida temporalmente por la señal **Ck**, lo cual corrobora su denominación de ***señal de sincronismo***.

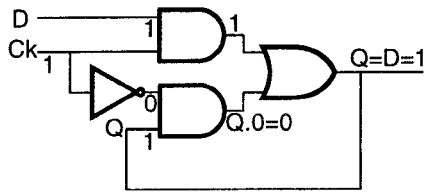


Figura A.126

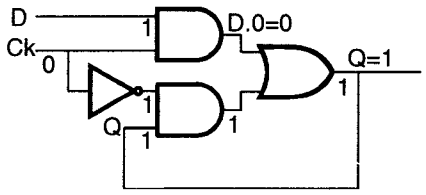


Figura A.127

Suponiendo Ck=1 (figura A.126), se seleccionará en el multiplexor el valor a memorizar, presente en **D**, por lo que su salida será **Q = D**; o sea simplemente **Q** repetirá el valor existente en **D**, *sin memorización,* dado que la porción del circuito seleccionada es puramente combinacional, sin ningún lazo de realimentación.

Por lo tanto, *si mientras* **Ck=1** *el valor que tiene* **D** *varía* ***n*** *veces, lo mismo hará la salida* **Q,** dado que ésta en ese lapso no puede autosostener su valor, pues el lazo no actúa[1]

Esto es típico del funcionamiento del flip flop **D "latch"** en el cual el lazo sólo actúa en el nivel bajo del **Ck**.

Volviendo a la figura A.126, si cuando **Ck=1** se quiere memorizar el valor **1**, se hace **D = 1**, en la salida será: **Q = D =1**.

Obsérvese que la salida de la compuerta AND del lazo en estas condiciones resulta **Q.0 = 0**, valor que como entrada a la **OR** no influirá en el valor de su salida **Q = D** que cambiará si lo hace **D**, pues el lazo no actúa, como se remarcó.

Cuando sea **Ck=0** (figura A.127) se habilita la entrada **Q** del multiplexor, que merced al cable que la une con su salida **Q** tiene el mismo valor que ésta. Entonces *el valor* **Q =1** *que tenía* **Q** *en el último instante en que fue* **Ck=1**, *entrará ahora* que es **Ck=0** *realimentado a la* OR, *autosustentando ese mismo valor que tiene la salida* [2]

De esta forma en todos los puntos del lazo se tendrá el valor **1**, como en las figuras A.122.b y A.122.c, automanteniéndose este valor mientras sea **Ck=0**, dado que la porción del circuito con el lazo permanecerá en esas condiciones en un ***estado estable*** *identificable con el valor de la salida* **Q=1**, en el cual queda "atrapado" el valor **1** en el lazo.

Ahora este valor permanecerá en la salida durante el tiempo que sea **Ck=0,** y **aunque cambie de valor existente en D la salida Q no cambiará**, almacenándose así un **1**.

Esto es así, dado que por ser **Ck=0**, la salida de la **AND** ligada a **D** será: **D.0 = 0**, valor que no influye en la salida de la **OR**, independientemente del valor que tenga **D** .

De manera semejante a la descripta, si cuando **Ck=1** se hace **D = 0** para almacenar un **0**, esto tendrá lugar cuando sea **Ck=0.** Entonces el circuito permanecerá en otro estado estable, esta vez "atrapando" el valor **0** en el lazo, identificable por ser **Q = 0** mientras sea **Ck=0**.

Por lo tanto, el presente circuito presenta dos estados estables distintos, distinguibles sensando la salida **Q**, manteniéndose cada uno durante el tiempo en que es **Ck=0**, debiéndose indicar cuando **Ck=1** –lapso en que no está en ninguno de esos estados– cuál de ellos debe ser el próximo estado en que se mantendrá el circuito

Resumiendo, el flip flop "**D**" **sincrónico "latch"** analizado, considerado como un bloque lógico (figura A.125) presentará dos entradas y dos salidas, tales que:

---

1 Es como si existiera un cable conectado directamente entre la entrada **D** y la salida **Q**, como si el circuito no existiera, o fuese *"transparente"*, de donde resulta su denominación *"transparent latch"*.

En los otros biestables, como el ***"R-S"*** *estudiado y los sincrónicos que se verán,* ***el lazo de realimentación actúa permanentemente****, por lo que se pasa de un estado estable al otro, sin que el circuito se comporte sin memoria. Esto sucederá en el* ***D*** *"latch" mientras* ***Ck*** *esté en el nivel alto.*

2 Cuando se pasa de la configuración circuital de la figura A.126 a la A.127 puede ocurrir que debido a los retardos en los caminos de las señales tenga lugar un ***"azar"("hazard")***. Esto es, que **Q** no se mantenga en el valor 1, sino que durante un breve lapso valga 0, por existir momentáneamente dos ceros en las entradas de la OR. Ello es factible, dado que por el retardo del inversor, la AND ligada a **D** tendrá su salida en 0 antes que la salida de la otra AND cambie a 1. Una forma de eliminar este problema, es agregar otra AND con entradas **D** y **Q**, cuya salida sea la tercer entrada de la OR .

La entrada *de datos*, "**D**", sirve para indicar ("*escribir*") el valor **0** ó **1** que se quiere memorizar. La entrada *de sincronismo*, **Ck**, cuando está en el nivel alto permite a la entrada **D** actuar sobre el circuito; y cuando está baja determina la *memorización* del valor que existía en **D** en el momento en que **Ck** cambió de **1** a **0**

La salida **Q** mientras sea **Ck=1** repite el valor que tiene **D** (**Q = D**), y durante el tiempo en que **Ck=0** permanece con el valor que memorizó el circuito, de modo que se pueda conocer ("*leer*") el mismo sensando el nivel del cable **Q**.

En la figura A.128 se indica el funcionamiento temporal de este flip flop.

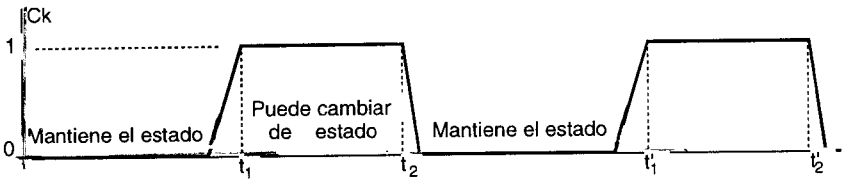


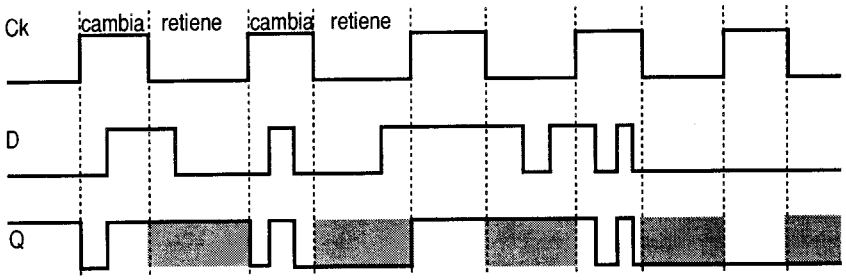
Figura A.128

Desde 0 a t_1, mientras **Ck=0**, el circuito se queda en uno u otro estado. Luego, cuando en t_1 es **Ck=1**, y durante el tiempo que siga así, se podrá cambiar el valor del bit que se quiere almacenar, y también en consonancia cambiará de valor la salida, dado que en ese lapso es **Q = D**.

En otro instante t_2, a partir del cual otra vez se hace **Ck=0**, se mantendrá el valor que tenía la entrada **D** (y tambien **Q**) en t_2, el cual será indicado por la salida **Q** hasta que la señal **Ck** vuelva a valer **1**, en t'_1. Entonces el valor de **Q** otra vez dependerá del valor existente en la entrada **D**

El siguiente diagrama temporal ejemplifica una variación supuesta en las entradas de un flip flop **D**. En grisado se ha puesto de relieve los tiempos de retención

Figura A.129 ⟶



A1.28 Flip flop "maestro-esclavo" ("master-slave")

El flip flop **D** puede cambiar de estado mientras **Ck=1** tantas veces como lo haga la entrada **D**. Esto no es apropiado para muchas aplicaciones prácticas, entre las que se encuentran los registros de desplazamiento, tema a tratar. Por tal motivo, para almacenar un bit se utiliza un conjunto de dos flip flops (figura A.130), denominados *"maestro y esclavo" ("master-slave")* para lograr que la salida del "esclavo" –que es la salida del conjunto– **cambie una sola vez**, aunque la entrada **D** del "maestro" cambie varias veces mientras **Ck**=1.

La entrada del "esclavo" toma la salida del "maestro", por lo que está obligado a seguirlo, y su entrada **Ck** está invertida respecto de la de éste.

De este modo el "maestro" puede cambiar (cuantas veces lo haga la entrada **D**) luego del flanco ascendente de cada pulso de **Ck**, cuando **Ck**=1. Mientras esto último ocurre, el "esclavo" no puede cambiar, hasta que después del flanco descendente de **Ck** sea **Ck**=0, y por consiguiente $\overline{\mathbf{Ck}}$=1. Entonces el "esclavo" tomará de la salida del "maestro" el valor (0 ó 1) que guardaba éste. Por lo tanto el "esclavo"podrá cambiar una sola vez, cuando $\overline{\mathbf{Ck}}$=1, dado que el "maestro" ya no puede cambiar, por ser **Ck** = 0.

En síntesis: el "maestro" recibe la información presente en la entrada **D** cuando **Ck**=1, y la almacena en su circuito cuando **Ck**=0. Es entonces cuando el "esclavo" copia dicho valor almacenado, presentándolo por la salida **Q**.

*O sea que la salida **Q** del flip flop sólo puede cambiar luego que **Ck** pase de 1 a 0.* Mientras sea **Ck**=0 el "maestro" no puede cambiar, y por lo tanto una vez que cambió el "esclavo" éste tampoco podrá cambiar. Cuando nuevamente sea **Ck**=1, el "maestro" recibirá nuevamente la información presente en **D**, pudiendo cambiar su salida, pero la salida del "esclavo" permanecerá con el valor que tenía cuando **Ck**=0, dado que si **Ck**=1 en correspondencia es $\mathbf{Ck_2}=\overline{\mathbf{Ck}}$=0, por lo cual el "esclavo" está en estado de retención.

*Vale decir, que la salida **Q** del "esclavo" una vez que cambió –luego que **Ck** pase de 1 a 0– no puede volver a hacerlo hasta que nuevamente **Ck** pase de 1 a 0.*

En la figura A.131 se suponen los mismos cambios en **Ck** y **D** que en la figura A.128. La salida que en ésta se indica **Q** será $\mathbf{Q_1}$, pasando a ser ahora **Q** la salida del "esclavo"

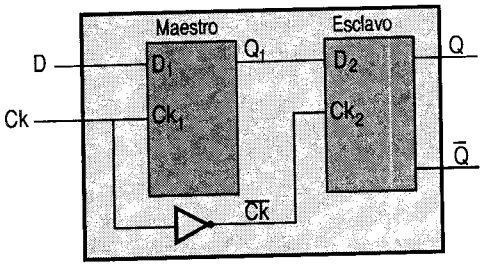


Figura A.130

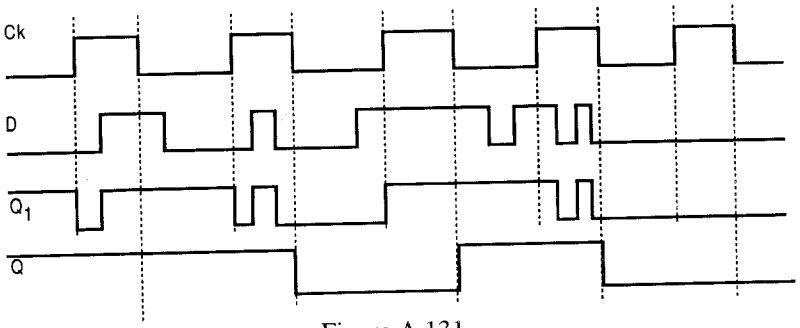


Figura A.131

A1.29 Registros

Un *registro* es un conjunto de **n** flip flops asociados, que permiten almacenar temporalmente una palabra o grupo de **n** bits.

Estudiaremos dos clases de registros típicos constituídos por flip flops sincrónicos: *el registro de entradas y salidas en paralelo, y el registro de desplazamiento,* que son ampliamente usados en el ámbito de las computadoras y de transmisión de datos

El *"registro universal"* combina ambas modalidades de registro, y presenta entradas para seleccionar una u otra. En la UAL existe un registro semejante, para realizar las operaciones **shift** (right ó left) y **rotate** (right ó left)

Registro con entradas y salidas en paralelo

Este tipo de registro es el constituyente de los registros de la UCP y los ports, entre otros

Un registro "**de paralelo a paralelo**" permite transferir simultáneamente bits hacia o desde el mismo.

La figura A.132 ilustra este registro para n=8 bits Consta de 8 flip flops **D** "latch" que tienen unidas sus entradas de **Ck**, de modo de poder cambiar juntos con cada pulso reloj

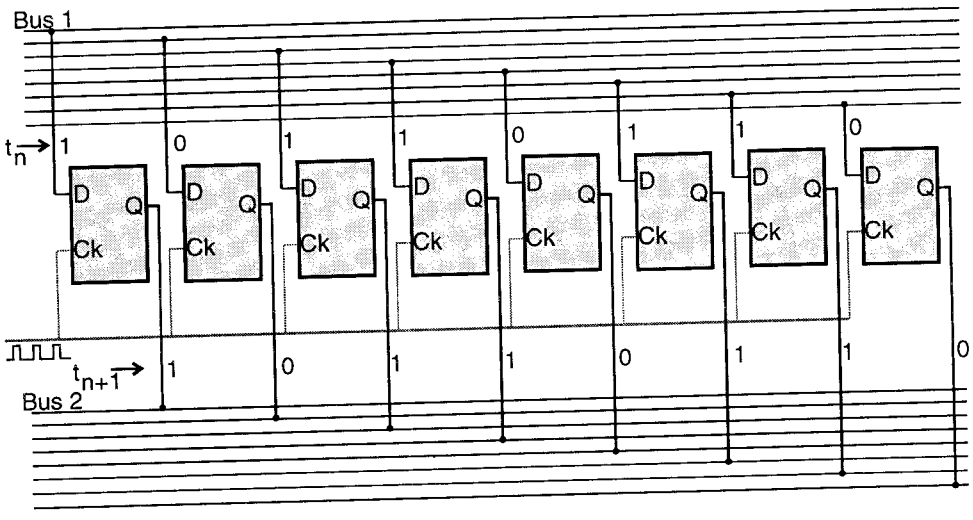


Figura A.132

Suponiendo que se quiere transferir hacia el registro para su memorización la palabra 10110110 que llega por las 8 líneas del bus **1**, los bits que la componen deben estar presentes en las correspondientes entradas **D** de los flip flops cuando **Ck=0**. Cuando **Ck** pasa al nivel alto cada bit que forma la palabra se almacenará en el flip flop respectivo, transfiriéndose esa palabra hacia el registro. Esto se ha indicado en el instante t_1 de la figura A.133 para un flip flop genérico del conjunto que cambia de **0**

La transferencia de nueva información suele denominarse *"carga"* del registro.

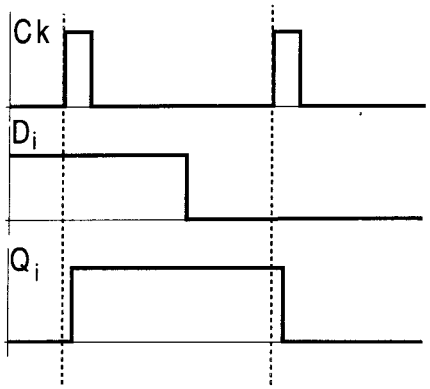


Figura A.133

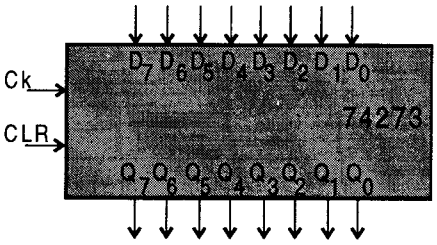


Figura A.134

Una vez que cambiaron todos los flip flops, los bits que componen la palabra 10110110 estarán disponibles en las 8 salidas de los mismos, y por ende en las 8 líneas del bus **2**, durante el tiempo en que **Ck=0**, para ser transferidos hacia donde sea necesario. Vale decir, que las 8 salidas indican la información almacenada en el registro.

Si en ese lapso en el bus **1** cambia la palabra presente en las 8 entradas, esto no afectará el valor que tienen las salidas. Con la llegada del próximo pulso reloj (en t_2), se repite el ciclo anterior.

El bloque representativo de este registro se indica en la figura A.134

Registro de Desplazamiento ("Shift Register")

Un ***registro de desplazamiento*** *a derecha* permite mover en esa dirección los bits que contienen los flip flops que lo constituyen, de forma tal que cuando todos reciben juntos un pulso reloj, cada flip flop pasa el bit que tenía almacenado al siguiente que está a su derecha, para que lo almacene.

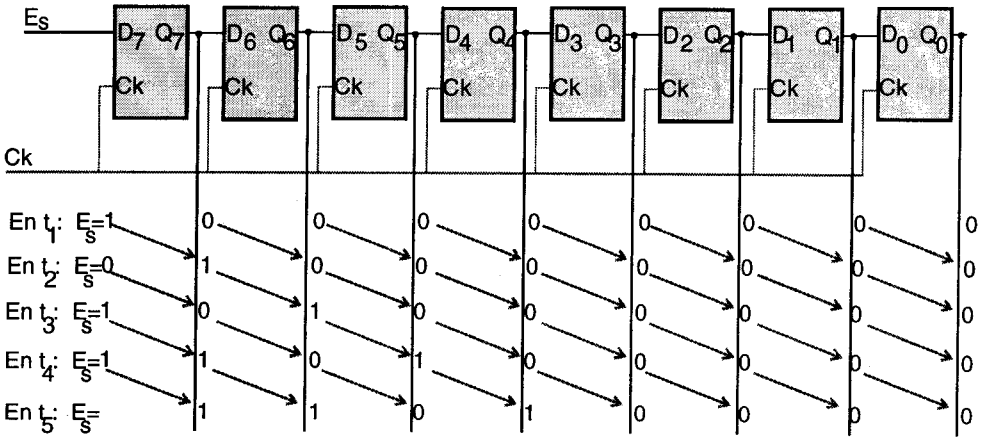
El flip flop de la extrema izquierda almacenará el valor presente en su entrada **D**, y el bit que almacenaba el de la extrema derecha se perderá, dejará de ser memorizado por el registro.

De igual modo se define un registro de desplazamiento a izquierda (*"shift left register"*)

A continuación verificaremos esta operación en un circuito conformado por 8 flip flops "**D**" (fig. A.135), que necesariamente deben ser "maestro-esclavo" (sección A1.28), puesto que así –mientras **Ck**=1– ***el "maestro" de cada flip flop recibe el bit a almacenar del "esclavo" del flip flop que está a su izquierda, al mismo tiempo su "esclavo" transmite el bit que almacena al "maestro" del flip flop que está a su derecha.***

La figura A.132 debe verse en relación con el diagrama temporal de la figura A.136

Después del flanco descendente del **Ck**, cada flip flop guardará el bit que le pasó el que está a su izquierda, *reteniéndolo hasta el mismo flanco del próximo pulso reloj.*



← Figura A.135

Supondremos que en un instante t_1, cuando llega el primer pulso de reloj todos los flip flops están en **0**: cables $Q_7 = Q_6 = ... = Q_1 = Q_0 = 0$ lo cual se logra activando previamente un cable que une las entradas puesta a cero de los flip flops (conexión no dibujada por no ser esencial). Esta situación se indica en el renglón "En t_1" de la figura A.135, y se corresponde con los valores de las variables en t_1 de la figura A.136

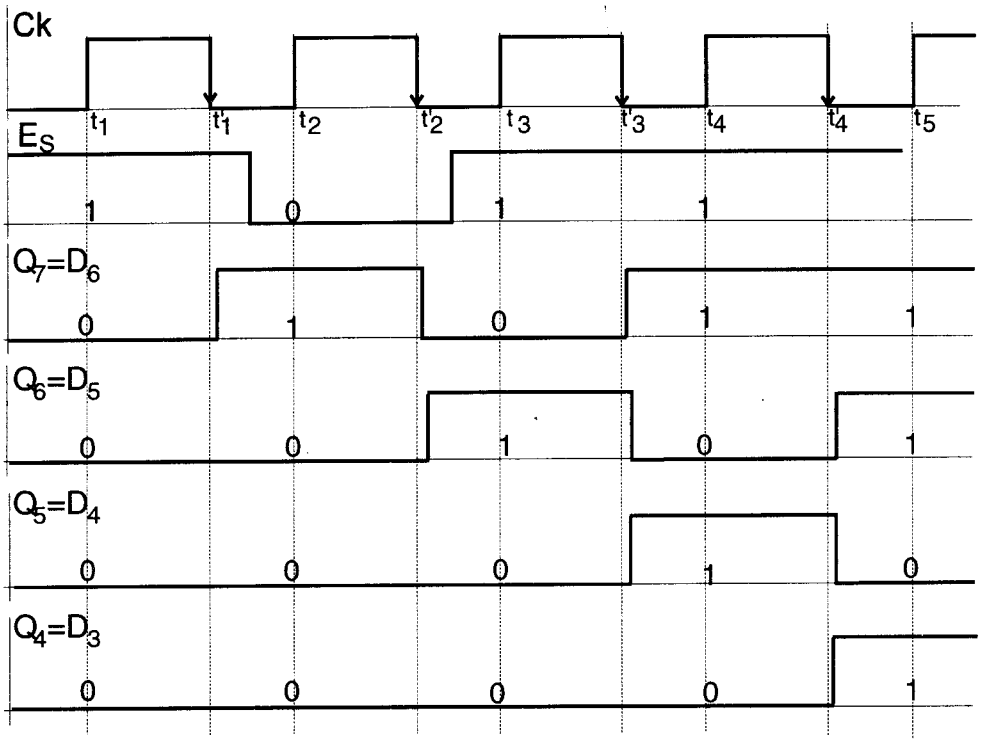


Figura A.136

Se ejemplificará el funcionamiento para cuatro pulsos reloj sucesivos, por lo cual se ha supuesto que antes de cada uno de ellos en la entrada $\mathbf{E_S}$= $\mathbf{D_7}$ sucesivamente se tendrá: 1011, como aparecen verticalmente los valores que toma $\mathbf{E_S}$ en la figura A.135, al igual que en el diagrama temporal de dicha variable en la figura A.136

Conforme al funcionamiento sintetizado más arriba, cuando esté presente el primer pulso (**Ck=1**) el "maestro" de cada flip flop tomará el valor que tiene el "esclavo" del flip flop que está a su izquierda, siendo que el "maestro" de $\mathbf{Q_7}$ recibe el valor $\mathbf{E_S}$ proveniente del exterior. Acorde con esto, como se ha supuesto inicialmente que todos están con su salida **Q** en **0**, cuando esté presente el primer pulso (**Ck=1**), cada "maestro" toma un **0** del "esclavo" anterior, menos el del flip flop $\mathbf{Q_7}$, cuya entrada es $\mathbf{D_7 = E_S = 1}$ dado que ésta se supuso que presenta el valor 1

Después del flanco descendente de **Ck** ($\mathbf{t'_1}$), cada "esclavo" habrá copiado el bit que guarda su "maestro", resultando 10000000 en los cables $\mathbf{Q_7}$ a $\mathbf{Q_0}$ (segundo renglón de valores de la figura A.135) estado que perdurará hasta el flanco descendente del próximo próximo pulso reloj. En el diagrama temporal se indica el cambio en la salida $\mathbf{Q_7}$, siendo la única que varía.

Vale decir, que luego del primer pulso, en el registro se habrá almacenado la palabra 10000000, cuyos bits están disponibles simultáneamente en las 8 salidas del registro, que son las salidas $\mathbf{Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0}$ de los flip flops que lo constituyen.

Respecto a la situación o estado 00000000 determinado en $\mathbf{t_1}$, *todo sucede como si la información se desplazó hacia la derecha*, como indican las flechas en grisado en la figura A.135, perdiéndose el **0** que guardaba el flip flop $\mathbf{Q_0}$, siendo reemplazado en este caso por otro **0**

Cuando en $\mathbf{t_2}$ llega el segundo pulso reloj encuentra las salidas $\mathbf{Q_7}$ a $\mathbf{Q_0}$ en el estado 10000000, y $\mathbf{D_7 = E_S = 0}$. Por lo tanto los "maestros" de los flip flops $\mathbf{Q_5}$ al $\mathbf{Q_0}$ recibirán un **0** del "esclavo" anterior; el del flip flop $\mathbf{Q_6}$ recibirá $\mathbf{1=Q_7}$; y el "maestro" de $\mathbf{Q_7}$ ahora tomará el valor **0** supuesto para $\mathbf{D_7 = E_S}$. Entonces, luego del flanco descendente de este pulso ($\mathbf{t'_2}$), el registro almacenará 01000000 (tercer renglón de valores de la figura A.135) hasta el flanco descendente del tercer pulso reloj.

Esta vez habrán cambiado $\mathbf{Q_7}$ y $\mathbf{Q_6}$ como aparece luego de $\mathbf{t'_2}$ en la figura A.136

En la misma se indican las transiciones hasta que luego de un cuarto pulso se almacena la palabra 11010000. No se han dibujado $\mathbf{Q_3Q_2Q_1Q_0}$, dado que no cambian. Los renglones de valores en las salidas de la figura A.135 acompañan estos cambios.

La instrucción SHIFT RIGHT ordena un solo desplazamiento a la derecha. El bit que estaba en el flip flop extremo derecho no se pierde, sino que pasa a ser el bit **C** del conjunto **SZVC**, guardándose en el flip flop que almacena dicho bit. Así, por ejemplo, se puede conocer cuál es el valor de dicho bit que forma parte de un byte

El registro de desplazamiento como registro "serie-paralelo"

Los 4 bits 1101 que entraron por la entrada **A** *en serie*, uno tras otro con cada pulso reloj, luego de 4 de estos pulsos se habrán almacenado en los 4 primeros flip flops de la izquierda. Al cabo del cuarto pulso pueden ser obtenidos *en paralelo*, simultáneamente, en los cables de las salidas $\mathbf{Q_7Q_6Q_5Q_4}$, que pueden estar conectados a un bus como en la figura A.132 Esos bits luego de 4 pulsos más se habrán desplazado a los 4 flip flops restantes, ocupando su lugar los bits que para cada uno de estos pulsos fueron entrando por $\mathbf{E_S}$.

De esta forma, luego de 8 pulsos reloj la información que con cada pulso fue entrando *en serie* al registro, queda memorizada en el mismo, pudiendo ser obtenida *en paralelo*, de los 8 cables de las salidas $\mathbf{Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0}$

Cuando el registro de desplazamiento se usa para convertir información que llega *en serie* por un solo cable, en información *en paralelo* por **n** cables, se denomina ***registro "serie-paralelo"***

Para esta aplicación se requiere que las salidas de los flip flops lleguen al exterior.

La figura A,137 esquematiza en esencia el bloque representativo de un registro "serie-paralelo", teniendo presente el registro tratado, sin tener en cuenta las entradas **A** y **B**.

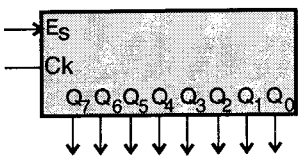


Figura A.137

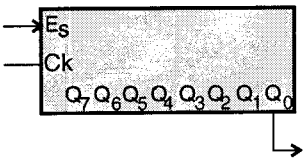


Figura A.138

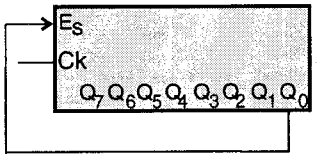


Figura A.139

El registro de desplazamiento como registro "serie-serie"

Si los bits que entran en serie por $\mathbf{E_S}$ se toman luego de 8 pulsos reloj de la salida $\mathbf{Q_0}$ se tiene un ***registro "serie-serie"***, con entrada y salida serie de datos, utilizado cuando se quiere retener información seriada durante el lapso que duran dichos pulsos. Esta aplicación permite simular una ***línea de retardo*** mediante un registro de desplazamiento de **n** flip flops: un bit que entra queda demorado **n** pulsos reloj hasta que sale. La figura A.138 da cuenta del bloque representantivo de este uso del registro.

El registro de desplazamiento como registro para "rotar" información

Si se realimenta la entrada $\mathbf{E_S}$ de la salida $\mathbf{Q_0}$, se tiene un ***registro o memoria "recirculante"*** (figura A.139) en el cual la información ***rota*** sin que se pierda ningún bit. Esto se ve más en detalle en la figura A.141

De esta forma en la UAL se realiza la operación que ordena la instrucción ROTATE RIGHT, desplazándo un solo bit por instrucción de este tipo. Análogamente puede implementarse la ejecución de la instrucción ROTATE LEFT

Registro universal

Un registro "universal" puede presentar 4 modos de operación:

- Carga en paralelo sincrónica.
- Desplazamiento a derecha (*"shift right"*)
- Desplazamiento a izquierda (*"shift left"*)
- Retención (*"hold"*) de la información durante una cantidad de pulsos reloj requerida.

Un registro que puede desplazar a derecha o izquierda se denomina *"bidireccional"*.

Usando un multiplexor de 4 entradas de datos (vías) como el de la figura A.104, es posible elegir uno de esos 4 modos, para lo cual hacen falta 2 entradas de selección que en lugar de **A** y **B** ahora se designan $\mathbf{S_1}$ y $\mathbf{S_0}$.

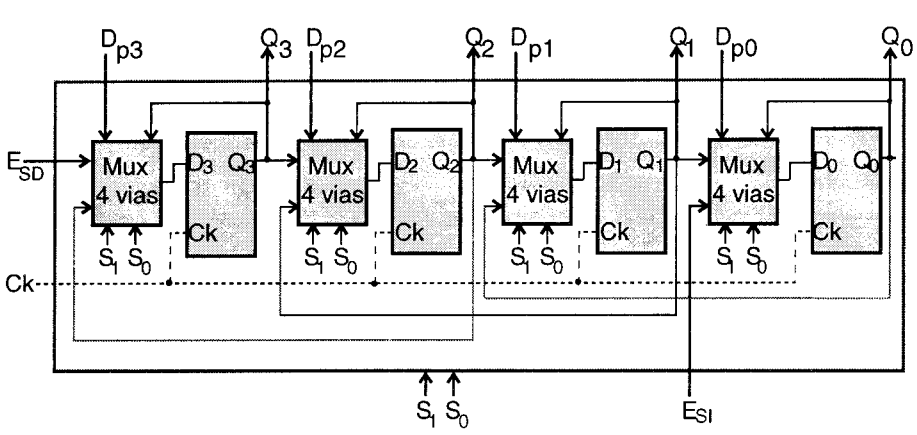


Figura A.140

Cuando $\mathbf{S_1}$**=1** y $\mathbf{S_0}$**=1** se seleccionan las entradas de carga en paralelo, y el registro puede actuar como el descripto en la figura A.132

Si $\mathbf{S_1}$ **= 0** y $\mathbf{S_0}$ = 1el registro opera desplazando a derecha, con nueva información serie tomada a través de la entrada serie a derecha $\mathbf{E_{sd}}$

La tercer vía del multiplexor se habilita con $\mathbf{S_1}$**= 1** y $\mathbf{S_0}$ **= 0**, para que cada entrada **D** reciba el valor de la salida **Q** del flip flop que está a su derecha (conexión en trazo gris).

Así cada flip flop pasa el bit que tenía almacenado al flip flop que está a su izquierda, y el de la extrema izquierda recibe la información serie del exterior a través de la entrada serie a izquierda ($\mathbf{E_{si}}$). Con cada pulso reloj, pues, la información en el registro se desplaza un bit a la izquierda.

Mientras sea S_1=**0** y S_0=**0** se selecciona la cuarta vía de cada multiplexor, de forma que la entrada **D** de cada flip flop reciba su propia salida **Q** (dibujada en la parte superior del Mux, por lo que con cada pulso reloj que llegue se vuelva a escribir en el flip flop el valor que tenía **Q**, manteniéndose *("hold")* así almacenado dicho valor sin cambios

Dado los 4 modos de operación de un registro universal, y por presentar al exterior las salidas de los flip flops que lo componen, este circuito permite:

- La conversión en forma sincrónica con los pulsos del reloj de datos en paralelo a serie y viceversa.
- El almacenamiento temporario de datos: así un registro universal de **n** bits puede guardar **n** bits durante un pulso reloj si opera como registro con entradas y salidas en transmisión serie a serie; o bien mantener sin cambios los datos almacenados durante la cantidad de pulsos que se requiera, en modo "*hold*".
- Multiplicar o dividir por dos.

Ejemplificaremos una aplicación. Si en las entradas D_{p3} D_{p2} D_{p1} D_{p0} está la combinación 1000, y si S_1=**1** y S_0=**1** al próximo pulso que entre por **Ck** ese número se carga en paralelo en los flip flops, resultando Q_3 Q_2 Q_1 Q_0 = 1000.
En estas condiciones supondremos que antes que llega otro pulso se ordena desplazamiento a derecha (S_1=**0** y S_0=**1**) y la entrada E_{SD} se pone en **0**. Entonces luego que entre otro pulso por **Ck** el registro contendrá Q_3 Q_2 Q_1 Q_0 = 0100. De esta forma se ha dividido por 2 en un solo paso, pues de 1000 = 8 se ha pasado al 0100 = 4. Si se mantienen las entradas con los valores indicados, cuando llegue otro pulso será Q_3 Q_2 Q_1 Q_0 = 0010 (nueva división por dos).
Para multiplicar por dos se debe desplazar a izquierda (S_1=**1** y S_0=**0**) y tener E_{SI} valor **0**, como puede verificarse.

A1.30 Circuitos Contadores-Secuenciadores

En el registro de desplazamiento de la figura A.141 –similar al de la figura A.135– se ha unido la salida Q_0 con la entrada D_3. De este modo luego de cada pulso reloj el bit que guarda el flip flop de salida Q_0 pasará al de salida Q_3 .
Supondremos que inicialmente está almacenada una combinación cualquiera, como la **0100**. Entonces, como indica la figura A.142, luego de cuatro pulsos reloj, por desplazamientos sucesivos, se volverá a tener la combinación **0100**. Si luego siguen llegando pulsos reloj por la entrada **Ck**, el circuito permanentemente repetirá la secuencia:

0100 ⟶ 0010 ⟶ 0001 ⟶ 1000 ⟶ **0100** ⟶ 0010 ⟶ ....

Decimos que el presente circuito presenta 4 **"estados"**, o que sigue un **código** compuesto por 4 estados.[1]

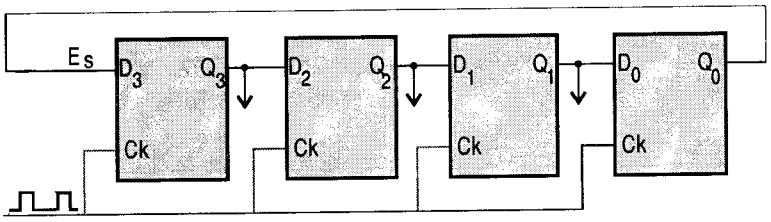


Figura A,141

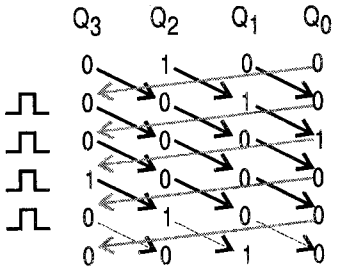


Figura A.142

El circuito dado puede usarse como **contador** de la cantidad de pulsos que entraron por la entrada **Ck**. Si el estado inicial 0100 se usa para indicar que no entró ningún pulso, el estado siguiente, 0010 (determinable en los cables de salida Q_3 Q_2 Q_1 Q_0) implica que entró un pulso por **Ck**. Los estados 0001 y 1000 significan que entraron dos y tres pulsos, respectivamente. Tres sería el número máximo de pulsos que podría contar este circuito, pues el estado siguiente 0100 representa que no entró pulso alguno[2] .

Otra aplicación del mismo circuito como *generador de secuencias de combinaciones binarias* (**secuenciador**). Suponiendo que cada milisegundo se recibe un pulso por **Ck**, a igual ritmo en las salidas Q_3 Q_2 Q_1 Q_0 se sucederá la secuencia 0100 ⟶ 0010 ⟶ 0001 ⟶ 1000

---

1 Un flip flop presenta dos estados: 0 y 1. Con dos flip flops es posible encontrar en sus salidas las combinaciones de estados 00, 01, 10, 11, según el estado particular en que se encuentre cada uno de ellos, o sea son posibles 4 estados. Tres flip flops permiten definir 8 estados distintos, del 000 al 111, etc
Se trata siempre de un número *finito* de estados, pudiendo un circuitos evoluciones automáticamente de un estado a otro con cada pulso reloj, siguiendo una *secuencia de estados* que comprende algunos o todos los 2^n estados posibles que pueden obtenerse con **n** flip flops.

2 Del mismo modo, el cuenta vueltas de un pasacasetes que va de 000 a 999, cuenta hasta 999, pues el estado 000 indica comienzo de nueva cuenta.

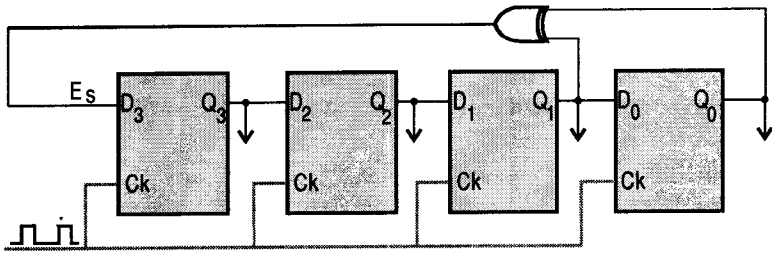


Figura A.143

Si el circuito además tiene la posibilidad de carga en paralelo, como el registro universal de la figura A 140, luego de un primer pulso se podrá conseguir que aparezca otra combinación distinta de las anteriores, por ejemplo, la 1101. Con los pulsos siguientes se comportará como el circuito de la figura A.141, resultando ahora la secuencia:

1101 → 1110 → 0111 → 1011 → 1101 → .

Puede verificarse que el circuito de la figura A.143, también basado en un registro de desplazamiento, sigue la secuencia: **1111** → 0111 → 0011 → 0001 → 1000 → 0100 → 0010 → 1001 → 1100 → 1110 → **1111** → . . .

En general, si se tiene un conjunto de flip flops y compuertas, puede conseguirse que a partir de una primer combinación dada, se generen secuencias fijas compuestas por un número predeterminado de combinaciones binarias.

Un secuenciador de este tipo se usa en ciertos procesadores para generar la secuencia de combinaciones binarias que debe aparecer en las salidas de la UC (Ver Unidad 1) para controlar la ejecución de cada instrucción. A la entrada de **Ck** de este circuito llegan los pulsos del oscilador a cristal que llegan a la UC (por ejemplo de 100 Mhz). Cada código de instrucción desencadena una secuencia distinta de combinaciones.Este circuito cumple las mismas funciones que la ROM de Control, pero puede permitir una mayor velocidad de generación de combinaciones.

También es factible construir circuitos que sigan una secuencia de combinaciones de números binarios sucesivos a partir de un número inicial cualquiera. Por ejemplo: 1011000 → 1011001 → 1011010 → 1011011 → 1011100 → . . .

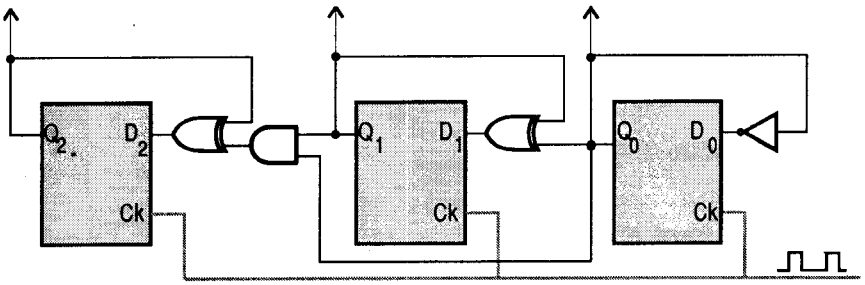


Figura A.144

El denominado **contador de programa** o **registro puntero de instrucción** (IP) opera de este modo, controlado por la UC, para formar la dirección de memoria donde se encuentra la siguiente instrucción a ejecutar. Este contador progresa contínuamente hasta que una instrucción de salto ordena que su valor cambie abruptamente. A partir del nuevo valor indicado por dicha instrucción, el circuito comienza una nueva cuenta progresiva, y así de seguido.

Se usa también un contador progresivo para llevar registro del reloj-calendario de un computador, el cual progresa merced a pulsos que llegan regularmente generados por un oscilador a cristal destinado a tal fin.

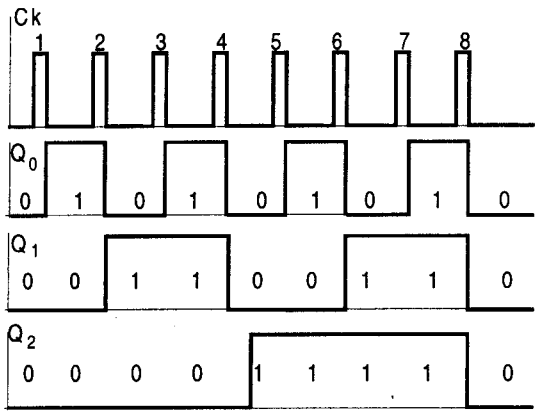


Figura A.145

La figura A.144 ilustra un contador progresivo que genera la secuencia (indicada temporalmente en la fig A.145) en $Q_2Q_1Q_0$

000 → 001 → 010 → 011 → 100 → 101 → 110 → 111 → 000 → . . . (cada flecha indica un pulso reloj)

En el flip flop de salida Q_0, la entrada **D** tiene siempre valor contrario a Q_0, por lo que con cada pulso reloj la salida Q_0 cambia de valor, como debe ser según se observa en la secuencia dada. También de ella surge que la salida Q_1 cambia de valor siempre que en el estado anterior haya sido $Q_0 = 1$. En el flip flop de salida Q_1 el valor de su entrada D_1 depende de la salida de una X-OR.

Esta tendrá valor opuesto al de Q_1 toda vez que $Q_0 = 1$, pues la salida de una X-OR niega el valor de una entrada (Q_1) si la otra (Q_0) vale 1 (como se vió en la sección A.18). De esta forma Q_1 cambiará de valor toda vez que llegue un pulso reloj, si antes era $Q_0 = 1$, como exige la secuencia. Igualmente de ésta se deduce que Q_2 cambia de valor si en el

estado anterior eran $Q_1 = 1$ y $Q_0 = 1$. Esto mismo ocurre con el flip flop de salida Q_2, cuya entrada D_2 será contraria a Q_2 si antes que llegue un pulso reloj son $Q_1 = 1$ y $Q_0 = 1$.

A1.31 Estructura básica de una RAM estática (SRAM)

Una memoria RAM que almacena cada bit mediante un flip flop se denomina ***"estática"*** (*"Static RAM"*-**SRAM**)

La pequeña SRAM de la figura A.146 con 4 celdas de 3 bits (4x3) servirá de modelo generalizable para cualquier otra de **mxn** bits

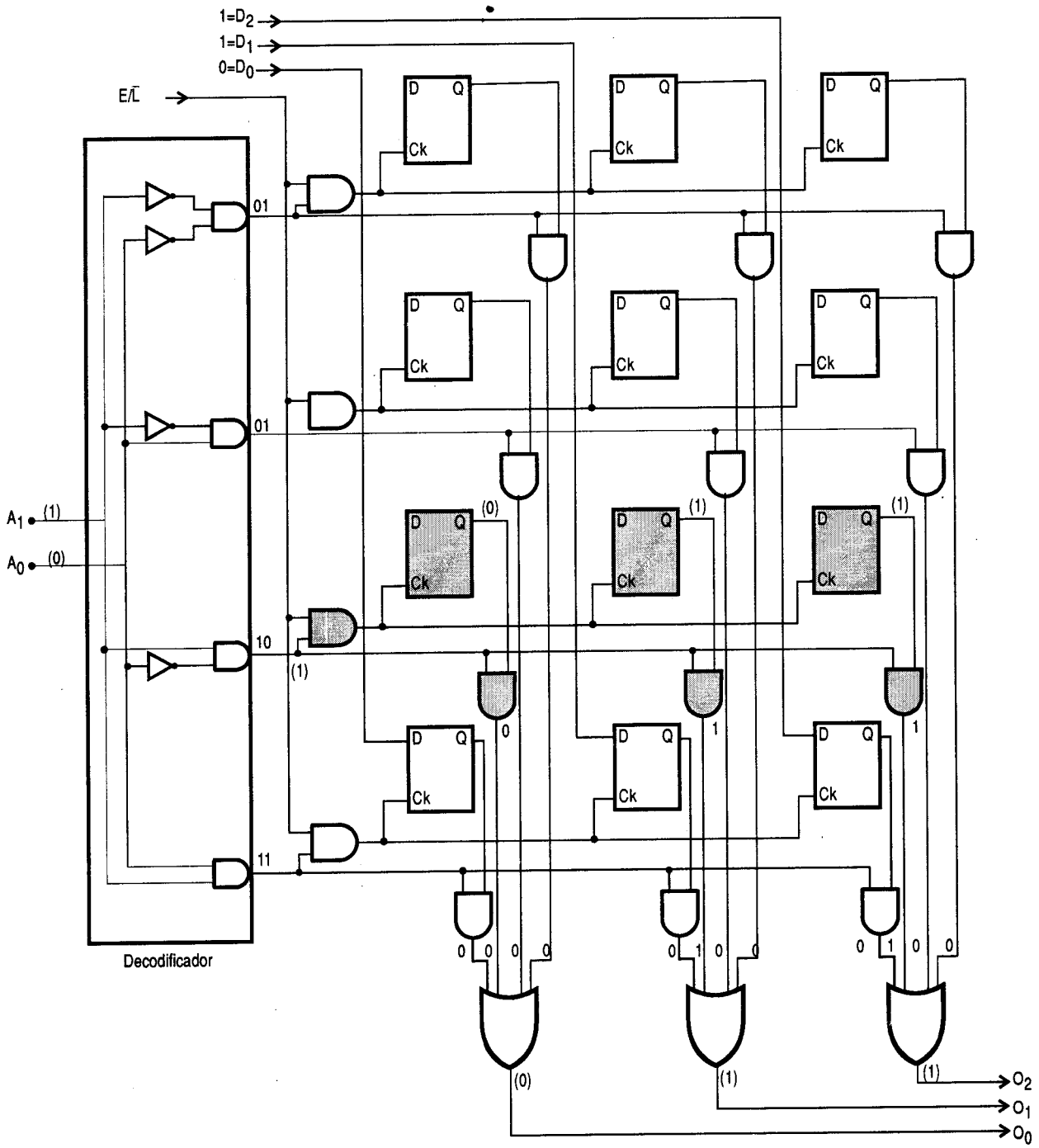


Figura A.146

Cada fila de 3 flip flops constituye una celda que puede ser seleccionada para ser escrita o leída, mediante una de las 4 salidas del decodificador. Suponiendo que se quiere acceder a la celda sombreada, de dirección $\mathbf{A_1A_0}$ = 10, entonces sólo la salida del decodificador designada **10** valdrá **1**, por lo que una de las entradas de cada una de las compuertas **AND** sombreadas también estará en **1**.

Si se quiere memorizar (escribir) los bits 110, en las entradas de datos $\mathbf{D_2D_1D_0}$ deberá estar presente dicha combinación, la cual aparecerá en las correspondientes entradas **D** de los flip flops de cada una de las celdas.

A continuación deberá llevarse a **1** el cable de escritura/lectura (E/$\overline{\mathbf{L}}$=1), con lo cual sólo valdrán **1** las entradass **Ck** de los 3 flip flops de la celda direccionada, dado que la compuerta AND que ataca esas entradas **Ck** es la única que tiene sus dos entradas en **1**. Como consecuencia, los 3 biestables almacenarán la combinación 110 –según indican las salidas **Q**– conforme al valor presente en la entrada **D** de cada uno.

Luego debe hacerse E/$\overline{\mathbf{L}}$ = **0** (orden de lectura), para evitar nuevas escrituras no deseadas en la celda direccionada.

En el supuesto que más tarde se quiera leer el contenido de la celda antes escrita, con la línea E/$\overline{\mathbf{L}}$= 0 (lectura), simplemente se direcciona la misma ($\mathbf{A_1A_0}$ = 10). Las 3 compuertas AND repetirán en su salida el valor existente en las respectivas salidas **Q**, dado que son las únicas con una entrada en **1**. Las restantes salidas de las compuertas AND, ligadas a las salidas de la RAM estarán en **0**, como se indica. Por lo tanto, en las salidas $\mathbf{O_2O_1O_0}$ se tendrá la combinación 110.

A1.32 Estructura básica de una RAM dinámica (DRAM)

En las memorias RAM *dinámicas* (**DRAM**), cada bit no se almacena como en las SRAM en un flip flop construído con 4 ó 6 transistores MOS (figura A.147), sino que la celda de almacenamiento (figura A.148) puede estar constituída por un solo transistor MOS *que oficia de llave,* cuya pequeña capacitancia parásita **C** de uno de sus terminales se usa para almacenar un bit. (Por ejemplo: **0** cuando **C** está descargado y **1** cuando **C** está cargado).

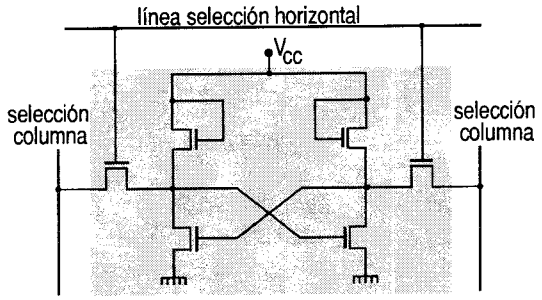


Figura A.147

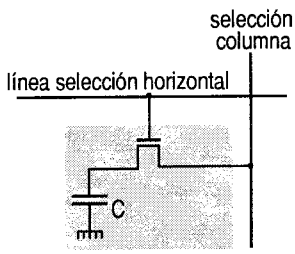


Figura A.148

Esto permite en las DRAM lograr grandes densidades de integración por pastilla (6 ó 4 veces más que las SRAM, según los transistores que ésta tenga por celda), un bajo consumo de corriente cuando la memoria no es accedida, y una economía de costo por bit almacenado. Las DRAM en el presente se construyen en tecnología CMOS con 4 Mbits por pastilla, y con tiempos de acceso de 60 nseg

Por las fugas de corriente siempre presentes, el capacitor de cada celda puede mantener su carga unos 10 - 20 mseg., razón por la que es necesario recargarlo antes que transcurra ese tiempo, operación conocida como ***"refresco"***. Este movimiento permanente de cargasa eléctricas requerido para mantener los datos almacenados que caracteriza a estas memorias, determina su denominación ***"dinámicas"*** para estas RAM.

En contraposición, en las SRAM los datos almacenados se mantienen indefinidamente en flip flops (mientras no se corte la energía o se escriban nuevos datos)

Las DRAM requieren un circuito de refresco y tienen comparativamente tiempos de acceso casi 6 veces mayores que las SRAM (suponiendo igual tecnología de los transistores) Estas últimas más fáciles de operar, y aunque son proporcionalmente más caras, en aplicaciones donde se requiere rapidez, como en las memorias "cache" de un computador, son insustituíbles.

Esquema de funcionamiento de una DRAM de 64Kx1

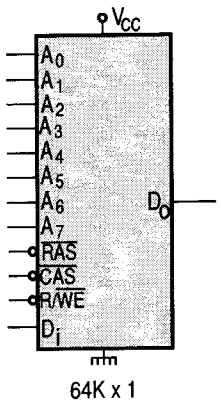
Desarrollando un modelo planteado por H. Taub[1], se describe a continuación la estructura interna básica (figura A.151) y las operaciones de lectura, escritura y refresco de una DRAM de 64K x 1 (65536 palabras de 1 bit, guardadas en igural número de capacitores designados C_0 a C_{65535}) La disposición de patas de la pastilla de esta DRAM se indica en la figura A.149.

Esta descripción puede ser útil para comprender el funcionamiento de otras DRAM de mayor capacidad.

La capacidad de una DRAM se considera como el número total de bits que almacena. Así una de 256K x 4 (figura A.150) es una "DRAM de 1 Mbit". O sea se especifica en bits, no en bytes.

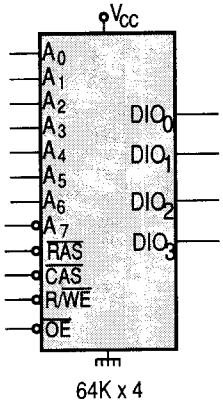
---

1 En su obra "Circuitos Digitales y Microprocesadores" (Mc Graw Hill - 1982)



64K x 1

Figura A.149



64K x 4

Figura A.150

Proceso que tiene lugar en la *lectura* de una celda:

Suponiendo que se quiere leer la posición 1111111100000001 = $FF01_H$ = $=65281_D$, y que la línea **R/$\overline{WE}$**= 1 (lectura), los pasos a seguir hasta obtener en $\mathbf{D_o}$ el valor del bit almacenado en esa posición, son los siguientes:

1a.

Se debe colocar en las líneas de direcciones $\mathbf{A_7}$ ... $\mathbf{A_0}$ de la pastilla (figura A.149) los 8 bits menos significativos (00000001) de la dirección, y luego (figura A.152) activar la señal $\overline{\mathbf{RAS}}$ (Row Address Strobe), señal de sincronismo para direcciones de filas de la matriz de memoria.

Después del flanco descendente de $\overline{\mathbf{RAS}}$ esos 8 bits se almacenan en el registro tipo "latch" (figura A.132) de direcciones de filas, por lo que en sus salidas se tendrá también 00000001^2, al igual que en las entradas del decodificador de filas, que así activará una de sus 2^8 líneas de salida, seleccionándola.

Para 00000001 se activará la línea N° 1 (en trazo más grueso),. con lo cual los 256 transistores MOS conectados a ella pasan al estado de conducción, equivaliendo a *llaves cerradas*, como se indica en grisado, *y los 256 capacitores* ($\mathbf{C_1}$, $\mathbf{C_{257}}$ ... $\mathbf{C_{65281}}$) *de las 256 celdas correspondientes a esa línea horizontal quedan conectados a las 256 lineas verticales de sensado de bits.* Los restantes (2^{16}–256) transistores MOS manejados por las 255 salidas del decodificador desactivadas, se comportan como *llaves abiertas*. En consecuencia los capacitores de las celdas vinculados a los mismos, quedarán desconectados de las 256 líneas verticales citadas.

1.b

Asimismo, luego de la activación de $\overline{\mathbf{RAS}}$ los 256 sensores-amplificadores[3]-biestables (SAB) conectados a las 256 lineas verticales sensarán y almacenarán con valores de tensión apropiados, el estado **1** ó **0** en que se encuentra cada capacitor de las 256 celdas seleccionadas por la salida del decodificador activada. Esto se ha querido simbolizar con los **1** y **0** dibujados en las salidas de los SAB, que repiten los valores que supuestamente almacenan las celdas. De esta forma, *los 256 SAB se comportan en definitiva como un registro de 256 bits luego de haber sensado y amplificado las tensiones existentes en los 256 capacitores de las celdas seleccionadas por la línea del decodificador activada.*

2

A continuación se debe colocar en $\mathbf{A_7}$ ... $\mathbf{A_0}$ de la pastilla la otra mitad superior de la dirección ($\mathbf{A_{15}}$ ... $\mathbf{A_8}$ que es 11111111 en nuestro ejemplo) y activar la señal $\overline{\mathbf{CAS}}$ (Column Address Strobe), señal de sincronismo para direcciones de columnas, conforme a la figura A.152

Ahora, luego del flanco de caída de $\overline{\mathbf{CAS}}$, se habrán almacenado esos 8 bits en el registro "latch" de direcciones de columnas, por lo que en sus salidas se tendrá $11111111 = 255_D$, al igual que en las entradas del decodificador de columnas, que así entre sus 256 salidas activará sólo la N° 255 (en trazo más grueso). En consecuencia, el transistor MOS conectado a dicha salida queda en el estado de conducción, como una *llave cerrada*, resultando sólo conectada a la salida $\mathbf{D_o}$ la linea que se quería leer. Los restantes 255 transistores MOS conectados a las salidas del decodificador desactivadas quedan abiertos.

En definitiva, colocando en las entradas $\mathbf{A_7}$ ... $\mathbf{A_0}$ de la pastilla primero la porción inferior y luego la superior de la dirección a leer, y en correspondencia activando $\overline{\mathbf{RAS}}$ y después $\overline{\mathbf{CAS}}$, *se habrá realizado un multiplexado (o sea primero sobre ésas líneas se seleccionan* $\mathbf{A_7}$ ... $\mathbf{A_0}$ y luego sobre las mismas $\mathbf{A_{15}}$ ... $\mathbf{A_8}$) *de dicha dirección, quedando dichas porciones repartidas en dos registros cuyas salidas van a sendos decodificadores, activándose así una salida en cada uno.*

La salida del decodificador de líneas permite seleccionar 256 celdas del total de 2^{16}=65536, cuyos capacitores quedan conectados a las 256 lineas de sensado, y así pasan los 256 bits que guardan dichas celdas a los 256 biestables que constituyen los SAB.

La salida del decodificador de columnas selecciona cuál de esos 256 SAB transmitirá el valor de su salida al terminal de salida (output) $\mathbf{D_o}$ *realizándose de hecho en la lectura un multiplexado de las columnas de la matriz de almacenamiento.*

Puede considerarse que se ha seleccionado la celda en grisado correspondiente a la dirección 1111111100000001, y que el bit que ella almacena $\mathbf{C_{65281}}$ *está presente para ser leído en la salida* $\mathbf{D_o}$ (conectada al exterior por estar cerrada la llave que simula un dispositivo "tri-state" comandado por la compuerta AND, dado que **R/$\overline{WE}$**= 1 en la lectura y $\overline{\mathbf{CAS}}$ esta activada)[4]

Esto aparece en la figura A.152 como dato de salida válido leído, luego de un tiempo de acceso $\mathbf{t_{rac}}$ (que es el de la DRAM, como ser 70 nseg.) medido desde el flanco de caída de $\overline{\mathbf{RAS}}$

Debe mencionarse que durante el tiempo $\mathbf{t_{rah}}$ (read address hold) los bits de dirección $\mathbf{A_7}$ ... $\mathbf{A_0}$ deben permanecer estables antes de poder entrar los bits $\mathbf{A_{15}}$ ... $\mathbf{A_8}$. Asimimo $\overline{\mathbf{CAS}}$ no puede activarse hasta un lapso $\mathbf{t_{rcd}}$ luego que se activó $\overline{\mathbf{RAS}}$, para dar tiempo a operar a los SAB.

---

2 Si bien 00000001 llega también al registro para direcciones de columnas, por no recibir éste su señal de activación $\overline{\mathbf{CAS}}$, no registrará dicha combinación.

3 "Sense amplifiers" en inglés

4 En una escritura dicha llave está abierta, y está cerrada la otra llave, que permite la entrada del bit a escribir por $\mathbf{D_i}$

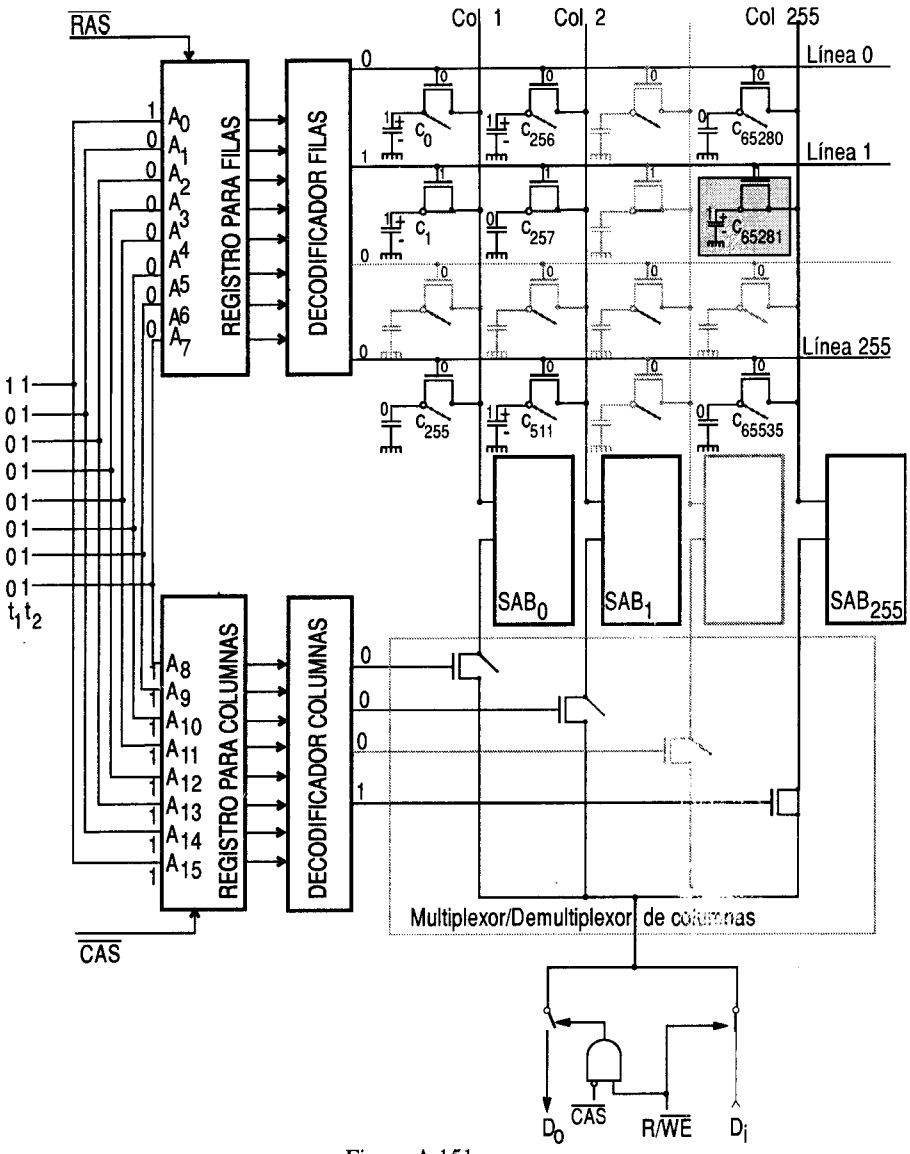


Figura A.151

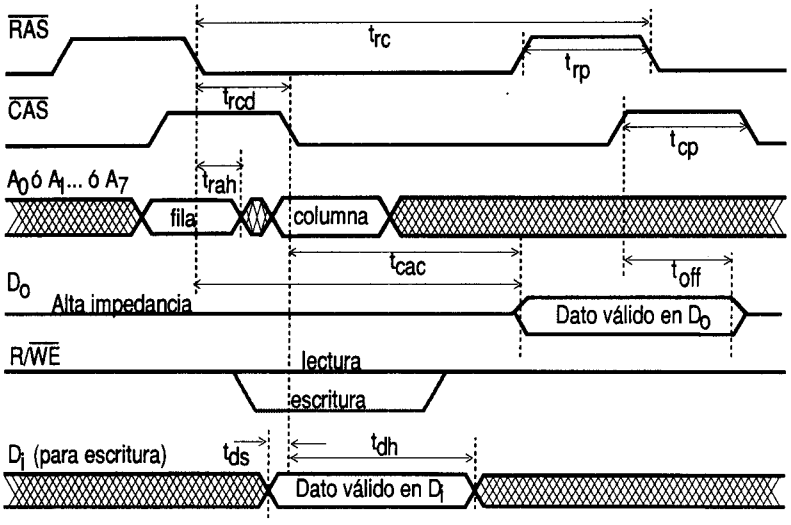


Figura A.152

3

Mientras **RAS** siga activada, cada SAB mantiene la carga del capacitor de celda conectado a él, resultando sobre éste una tensión $\mathbf{V_H}$ o $\mathbf{V_L}$ según sea. Luego del próximo flanco ascedente de **RAS** los cambios que tengan lugar en las entradas $\mathbf{A_7}$ ... $\mathbf{A_0}$ pueden hacer cambiar las salidas del registro latch de filas, y en consonancia el decodificador puede cambiar las 256 celdas seleccionadas Entonces los 256 transistores MOS de las celdas antes seleccionadas (en nuestro ejemplo por la línea N°1) equivaldrán a llaves abiertas, y los capacitores de celdas quedarán aislados de las columnas, pasando a retener un **1** si estaban cargados, ó un **0** si no lo estaban.

Los biestables de los 256 SAB quedarán reteniendo el bit que almacenaban, y mientras siga activa **CAS**, la salida del SAB de la columna antes seleccionada por el transistor MOS cerrado seguirá comunicando su valor a la salida $\mathbf{D_0}$. Cuando **CAS** se desactiva (luego o junto a **RAS**), la salida $\mathbf{D_0}$ pasará al tercer estado de "alta impedancia" (como si estuviera desconectada, aislada), después de un tiempo $\mathbf{t_{off}}$

Refresco de una DRAM:

En cada conjunto horizontal de 256 capacitores de celdas una vez que se desconecta de las 256 columnas, aquellos que guardan un **1** presentan el problema de que la tensión $\mathbf{V_H}$ – que les habían comunicado los SAB correspondientes– va disminuyendo con el tiempo, pues estos capacitores se van descargando por corrientes de pérdidas

Con la tecnología actual se puede sensar correctamente el valor $\mathbf{V_H}$ hasta un tiempo de 10 a 20 mseg luego que cada conjunto se desconectó de las columnas, necesitándose recargar *("refrescar")* los 256 capacitores del mismo luego de ese lapso especificado para cada tipo de DRAM. Obsérvese que tanto en la lectura como en la escritura de una celda, con la activación de **RAS** por el solo hecho de direccionarla, se selecciona uno de dichos conjuntos de 256 capacitores que se conectarán a las 256 columnas.

Según se vió, en el sensado se descargan aquellos capacitores que estaban cargados guardando un **1**, volviéndolos a recargar enseguida los SAB hasta la tensión $\mathbf{V_H}$ Ahora bien, es evidente que las lecturas o escrituras al azar en una DRAM como la dibujada en la figura A.151 no garantizan que cada uno de los 256 conjuntos de 256 celdas sea refrescado sistemáticamente antes de que sus capacitores cargados se descarguen al cabo de 10-20 mseg..

Si con un contador se generan en las entradas $A_7 \ldots A_0$ de la pastilla las 256 combinaciones progresivas 00000000 → 00000001 → .... →11111111 → 00000000 → .... , activando luego de aplicar cada una la entrada $\overline{\mathbf{RAS}}$, y se vuelve por ejemplo a 00000000 cada 10 mseg, se refrescarán las 65536 celdas de la DRAM en grupos de 256. Esta forma de refrescar se conoce como *"refresco con RAS solamente" ("RAS-only refresh")*, dado que debe variarse la dirección de fila y sólo activar $\overline{\mathbf{RAS}}$. Las entradas $\overline{\mathbf{CAS}}$, $\mathbf{R/\overline{WE}}$ y $\mathbf{D_i}$ deben quedar altas. No se puede acceder a la DRAM para lectura o escritura durante cada refresco

Si se debe volver refrescar cada fila de celdas al cabo de 10 mseg, entre dos combinaciones que genera el contador deben mediar 10mseg/256 ≈ 0,04 mseg = 40000 nseg

Asumiendo que una operación de refresco tarda 300 nseg, resulta que ocupa tan solo 300/40000 = 0,75% del tiempo que media entre dicha operación y la siguiente. O sea resta menos del 1% al tiempo total disponible para leer o escribir la DRAM.

La estructura de una DRAM permite refrescar conjuntos de celdas, amén de que el multiplexado de la dirección en dos porciones permite pastillas con menor cantidad de patas para direcciones.

Una **escritura** empieza como una lectura, y puede hacerse semejante a ésta. Por ello se dan en la misma figura A.152 los ciclos de lectura y escritura. La entrada $\mathbf{D_i}$ (con el bit a grabar) queda conectada al SAB seleccionado, que lo pasará a la celda antes seleccionada mediante $\overline{\mathbf{RAS}}$. Para manejar 8 bits hacen falta 8 pastillas como la descripta.

A1.33 Estructura básica de una PAL®

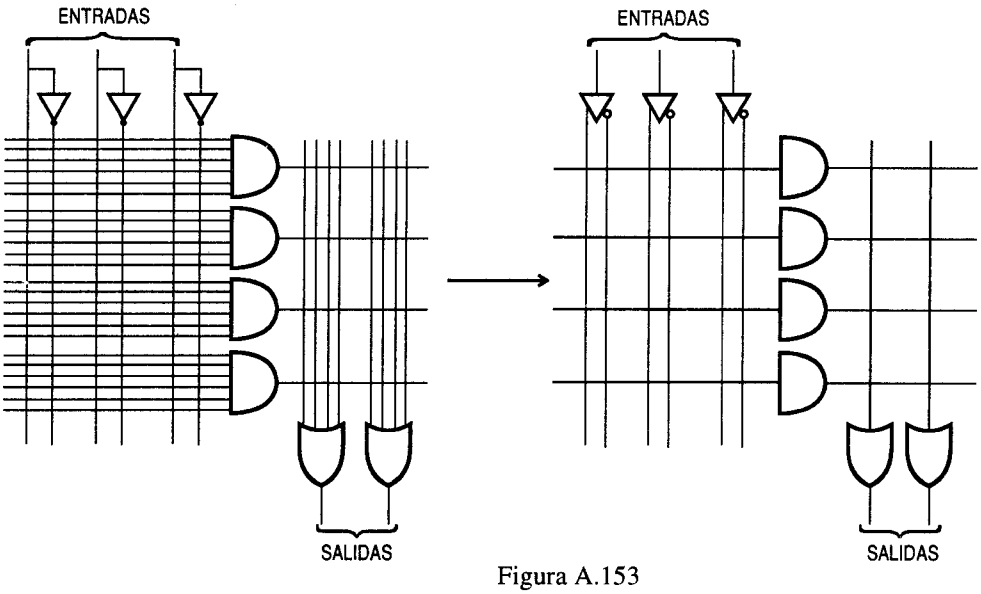


Figura A.153

Los circuitos ROM como los **PAL®**[1] (*"Programmable Array Logic"*) son circuitos integrados en gran escala (*"Large Scale Integration"*-**LSI**) de conexionado interno programable (*"Programmables Logic Devices"*–**PLD**). Son circuitos lógicos cuya función la determina el proyectista después que fue fabricado el chip correspondiente. En ambos casos se trata en esencia de una disposición o arreglo (*"array"*) de compuertas **AND-OR**. (figura A.153) [2] en el cual el usuario puede configurar {"programar") cómo se conectarán entre sí internamente líneas horizontales y verticales a fin de constituir un circuito combinacional que responda a una cierta tabla de verdad. Esto ya se trató en las ROM (figuras A.116 a A.118)

Muchos libros y manuales de fabricantes denominan **PLA** (*"Programable Logic Array"*) a las PAL®. En éstas se configura la matriz de entrada, que determina el producto de variables que genera cada AND (figura A.154), siendo ***fijo*** el conexionado entre las salidas de las AND y las entradas de las OR[3] La figura A.155 es representativa de una PAL programada.

Esta genera funciones lógicas del tipo AND-OR **negadas**, con la posibilidad de realimentar el valor de ciertas salidas (negadas o afirmadas) en cualquier entrada de las AND.

---

1 Siglas registradas por Advanced Micro Devices, Inc

2 El dibujo de la derecha es el mismo que el de la izquierda, habiéndose seguido por un lado la convención usada en las ROM de representar todas las entradas a cada compuerta mediante una sola línea (figura A.117); y por otra parte de un solo triángulo salen una línea de entrada afirmada y otra negada.

3 Esto es totalmente opuesto a las PROM (figura A.118), en las cuales es configurable la matriz de salida, siendo fijo el conexionado del decodificador, pues se deben generar los 2^n minitérminos (productos) posibles para **n** variables de entrada (lo cual no sucede en las PAL) Si bien en la figura A.97 un decodificador se dibujó con los inversores de las entradas como círculos antes de las compuertas, también se puede simbolizar los inversores mediante triángulos como en la figura A.153, siendo fijas las conexiones necesarias entre lineas horizontales y verticales para generar los 2^n minitérminos.

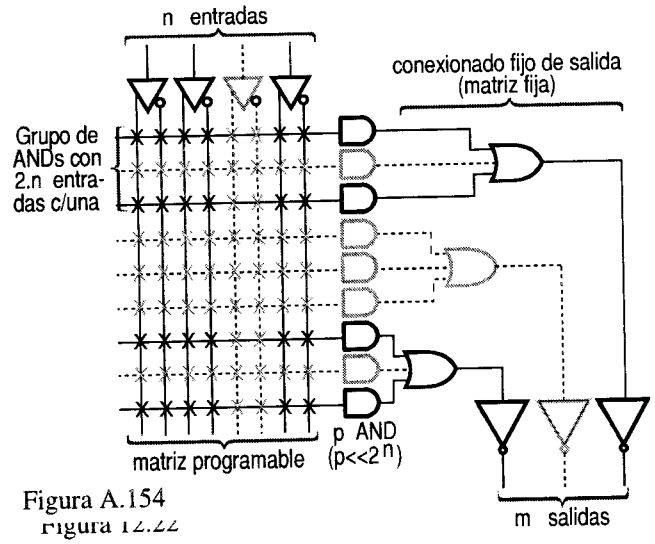


Figura A.154

Figura 12.22

microprocesador.

Esto último confiere gran flexibilidad de conexionado a las PAL. También existen PAL con un registro a la salida.

Otra diferencia con las ROM es que para **n** entradas no hay 2^n compuertas AND para generar todos los minitérminos posibles.

En general, si bien una ROM es un circuito combinacional de **p** salidas preparado para generar **p** funciones cualesquiera de **n** variables correspondientes a sus **n** entradas, frecuentemente resulta que una parte de la misma queda sin aprovechar. Una PAL puede cumplir *las mismas funciones que una ROM en un circuito con menos AND*, por lo cual ocupará menor área de silicio.

Por esta razón, por ejemplo, **se usa una PAL dentro de un microprocesador tipo RISC, siendo que las entradas de la PAL reciben el código de operación de cada instrucción, y sus salidas (que son las de la UC) generan la combinación de bits mediante la cual la UC realiza las funciones de control con cada pulso reloj.** Cabe mencionar que esta PAL, si bien tiene la estructura circuital de las figuras A.154 y A155, ya viene programada, pues se construye en el mismo chip del

A fin de concretar cómo se configuran las PAL, se construirán las siguientes funciones, supuestas minimizadas[1]

$$\overline{X} = \overline{C}.\overline{D} + A.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.\overline{D} + \overline{A}.C.D \qquad \overline{Z} = \overline{B} + \overline{A}.\overline{C}.D$$

Las funciones se expresan negadas, para adaptarlas a la PAL, que tiene un inversor en todas sus salidas.

Se observa que la función $\overline{X}$ es una suma de 5 productos, pero que en la PAL (figura A.155) sólo existen sumas de 3 productos. Por tal motivo se ha creado la función $\overline{Y} = \overline{C}.\overline{D} + A.B.\overline{C} + \overline{A}.B.C$ que es parte de $\overline{X}$, resultando:

$$\overline{X} = \overline{C}.\overline{D} + A.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.\overline{D} + \overline{A}.C.D = \overline{Y} + A.\overline{B}.\overline{D} + \overline{A}.C.D$$

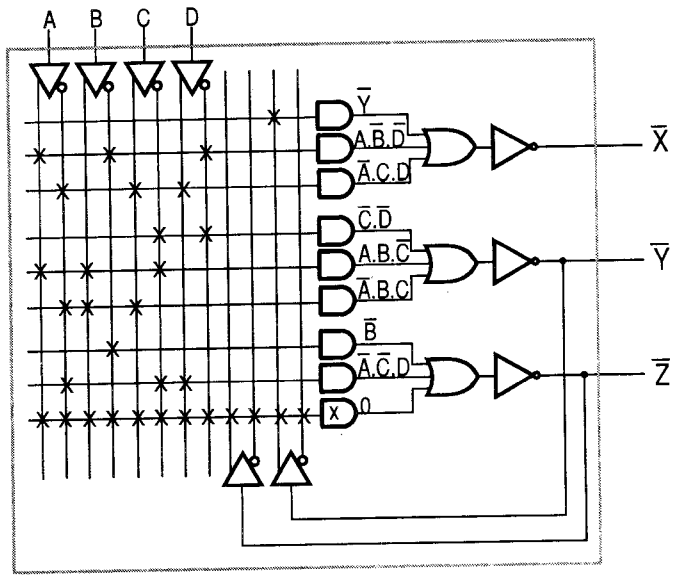


Figura A.155

Entonces, si están unidas las líneas horizontales y verticales que indican las cruces[2] se forman los productos que sumados dan $\overline{Y}$, y esta salida, suma de 3 productos de $\overline{X}$, se conecta como entrada de $\overline{X}$, podrá generarse esta función suma de 5 productos.

Respecto a $\overline{Z}$ nos encontramos que el circuito provee tres AND por función, pero que $\overline{Z}$ consta de dos productos. A fin de anular una AND, se dejan conectadas[3] sus 12 entradas a todas las variables. Así al al hacer el producto de cada variable por su negación (bastaría una sola variable) para que la salida de dicha AND resulte cero y no influya en la suma.

De esta forma se han construído las funciones $\overline{X}$ y $\overline{Z}$ aprovechando al máximo la circuitería existente.

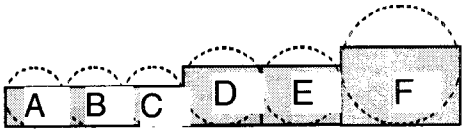
---

1 O sea con un mínimo de variables y de operaciones entre ellas. Existen programas para minimizar funciones lógicas de **n** variables. Los lectores interesados en los principios de la minimización por diagramas de Veitch-Karnaugh pueden consultar el libro "Introducción a las Técnicas Digitales con Circuitos Integrados" (8va edición) de M.C. Ginzburg. En esta obra también se ven con más detalle PAL, ROM y otros temas tratados en esta unidad.

2 Tener presente que cada línea horizontal representa en este caso 12 entradas de cada AND (8 por las entradas **ABCD** y sus negaciones y cuatro masa por las dos salidas y sus negaciones que pueden conectarse a entradas de cualquier AND)

3 Se supone que en este circuito la "programación" consiste en eliminar interconexionados (cruces), o sea que originariamiente cada línea horizontal (12 entradas) está conectada a las 12 verticales. En particular así lo estará la línea con las 12 cruces, por lo que se deja como está.

A1.34 Problemas



Una máquina expendedora automática entrega un producto que vale $ 15 si recibe monedas que se introducen en un "monedero" detector de tamaño y peso de monedas como el dibujado, construido de modo que:
Las ranuras A , B y C; aceptan una moneda de $ 5 cada una
Las ranuras D y E aceptan una moneda de $ 10 cada una.
La ranura F acepta una moneda de $ 25 .

A	B	C	D	E	F	Z	V	W
0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	0	1
0	0	0	1	0	0	0	0	0
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Completar diez filas más de la tabla de verdad izquierda que indica cuándo las monedas insertadas son de valor suficiente para que la máquina expida el producto, y que devuelva una moneda de vuelto –de valor adecuado– cuando se superen los $ 15 necesarios
Se supone por simplicidad, que no se introducirán monedas superfluas , por ejemplo, una de $ 25 con otra u otras menores, o dos de $ 10 y una de $ 5 , etc.
Enunciados utilizados:

"existe una moneda en la ranura A" : **A** "entregar el productor" : **Z**
"entregar $ 5 de vuelto" : **V** "entregar $ 10 de vuelto: **W**

Será **Z** = 1 si el dinero introducido, alcanza o supera los $ 15 establecidos.
Será **A** = 1 si se introdujo una moneda en la ranura A; y **V** = 1 si debe entregarse $ 5 de vuelto, etc.
Se emplean las variables **B, C, D, E** y **F** con el mismo sentido que **A**.

2 Verificar que una **NAND** de lógica positiva es una **NOR** de lógica negativa. Idem que si se quiere un circuito cuya salida sea 1 si todas las entradas están en cero, se requiere una compuerta **NOR**

3 ¿Es cierto o falso que si $A + B = 1$ y $A.B = 0$, debe ser $A = \overline{B}$?

4 Verificar las siguientes identidades usando tablas de verdad o algebraicamente

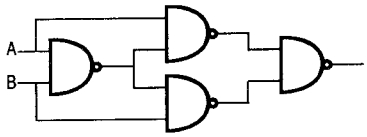
$0 \oplus A = A$ $\quad 1 \oplus A = \overline{A}$ $\quad \overline{A} \oplus A = 1$ $\quad A \oplus A = 0$

$\overline{A} \oplus B = \overline{A \oplus B}$ $\quad A \oplus \overline{B} = \overline{A \oplus B}$ $\quad A \oplus B = \overline{A} \oplus \overline{B}$

5 Conforme a las identidades del problema anterior indicar:
a) La forma de usar una compuerta **X-OR** para que opere como un inversor.
b) Idem para que invierta o no el valor de una entrada según lo ordene la otra.
c) Idem para que indique si **A = B**, o si son distintos.
d) Idem si en las entradas hay un número par o impar de unos

6 Verificar usando tablas de verdad la siguiente identidad, y de ser cierta expresarla circuitalmente: $A \oplus B = \overline{A}.B + A.\overline{B}$

7 Verificar que el siguiente circuito realiza $A \oplus B$



8 Para la utilización de una compuerta X-OR como se pide en el punto b) del problema 2, realizar un diagrama de tiempos indicando la operación requerida.

9. Realizar la operación $1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1$ ¿Qué sucede cada vez que se suma cero o dos unos seguidos?
Verificar que si el número de operandos que valen 1 es par, el resultado es siempre cero, y si dicho número es impar el resultado será 1
Establecer la operatoria a realizar para determinar si una combinación de 8 bits tiene un número par o impar de unos. Idem de ceros. Dibujar un circuito que realice tal operatoria

10 Determinar un circuito que permita que aparezca en su salida una señal periódica de forma cuadrada que recibe por una de sus 3 entradas, sólo si las otras dos tienen valor opuesto

11 Construir el circuito de cada una de las siguientes funciones, y hallar su tabla de verdad:

$Z = \overline{B + C + D}.A + B + D.\overline{B + C} + B + \overline{B}.C + B$

$W = \overline{B + D}.(\overline{A}.C + B.E) + \overline{B + C.D}.(A + B)$

$Y = \overline{A}.\overline{B.C \oplus A.B} + A.C.(\overline{B.D} + A).(\overline{C} + \overline{D})$

12 ¿Una SP puede presentar operaciones negadas? ¿Y variables negadas ?

13 Mediante una **NAND** e inversores conformar un circuito cuya salida detecte si en las 4 entradas existe la combinación 0100 entre las combinaciones posibles 0000 a 1111 que pueden aparecer. Realizar la misma función con una NOR e inversores.

14 ¿Un producto minitérmino es sinónimo de producto minitérmino ?

15 ¿Es correcta la aseveración de que en un minitérmino una variable negada siempre vale cero, o se trata de una regla práctica para formar minitérminos a partir de combinaciones de valores de las variables ?

16 Dada la función $\overline{\overline{\overline{\overline{A}.\overline{B}.C.(A+B.\overline{C})}}}+B.(C+A)$ verificar el desarrollo algebraico siguiente, que permite expresarla como una suma de productos y luego como una suma de minitérminos, indicando en cada caso qué propiedad se aplica

$Z = \overline{\overline{\overline{\overline{A}.\overline{B}.C}}}+A+\overline{\overline{B.\overline{C}}}+B.(C+A)$

$Z = \overline{A}.\overline{B}.C+A+B.\overline{C}+B.(C+A)$

$Z = \overline{A}.\overline{B}.C+\overline{A}.\overline{\overline{B.\overline{C}}}+B.(C+A)$

$Z = \overline{A}.\overline{B}.C+\overline{A}.(\overline{B}+\overline{\overline{C}})+B.(C+A)$

$Z = \overline{A}.\overline{B}.C+\overline{A}.\overline{B}+\overline{A}.C+B.C+B.A$ (Suma de productos)

$Z = \overline{A}.\overline{B}.C+\overline{A}.\overline{B}.1+\overline{A}.C.1+B.C.1+\overline{B}.A.1$

$Z = \overline{A}.\overline{B}.C+\overline{A}.\overline{B}.(C+\overline{C})+\overline{A}.C.(B+\overline{B})+B.C(A+\overline{A})+B.A.(C+\overline{C})$

$Z = \overline{A}.\overline{B}.C+\overline{A}.\overline{B}.C+\overline{A}.\overline{B}.\overline{C}+\overline{A}.B.C+\overline{A}.\overline{B}.C+A.B.C+\overline{A}.B.C+A.B.C+A.B.\overline{C}$

$Z = (\overline{A}.\overline{B}.C+\overline{A}.\overline{B}.C+\overline{A}.\overline{B}.C)+(A.B.C+A.B.C)+(\overline{A}.B.C+\overline{A}.B.C)+\overline{A}.\overline{B}.\overline{C}+A.B.\overline{C}$

$Z = \overline{A}.\overline{B}.C+A.B.C+\overline{A}.B.C+\overline{A}.\overline{B}.\overline{C}+A.B.\overline{C}$ (Suma de minitérminos)

17. Usando minitérminos realizar sólo con compuertas **NAND**:
a) Un circuito que reconozca con **Z** = 1 números binarios primos de 4 bits
b) Idem números binarios que sean potencias de dos.

18. A partir de la expresión de la salida de cada uno de los circuitos del problema anterior, expresar la misma como una suma negada de productos (**AND-NOR**) o sea como **AOI** (And Or Inverter)

19. Dada la función $Z = [(\overline{D}+B).C+\overline{A.B}].(\overline{A}.\overline{C}.\overline{D}+B)$, aplicando De Morgan hallar su complemento y buscar formas equivalentes con menor número de variables negadas. Dibujar los circuitos e indicar de cuántos niveles es cada uno.

20. Sea la función $Z = \overline{(\overline{A}+B+C.\overline{D}).E}$ Expresarla en forma no negada como SP y como SP minitérminos.

21 Dado un decodificador de 3 entradas, indicar qué ocurre cuando en ellas está la combinación 111

22 Dibujar un circuito multiplexor de 4 entradas de datos, e indicar el valor 1 ó 0 de cada cable, para que en la salida aparezca el valor de la entrada D_1, siendo que el valor de ésta es 0. Escribir la tabla de este circuito.

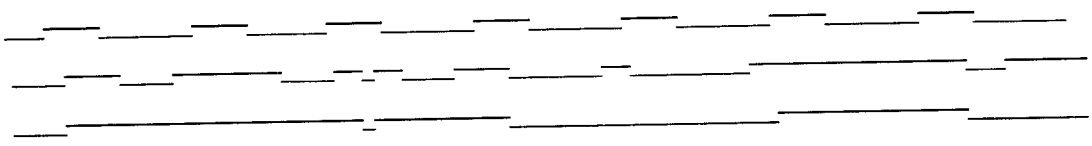
23 Dibujar un circuito demultiplexor de 4 salidas, e indicar el valor 1 ó 0 de cada cable, para que el valor 1 de la entrada aparezca en la salida Z_3. Escribir la tabla de verdad de este circuito.

24 En un sumador-restador del tipo de la figura A.96 efectuar –85 – (–3) y –85 + (–3) indicando el valor 1 ó 0 de cada cable. Verificar que el resultado en cada caso coincida con el esperado en decimal.

25 Mediante un circuito generar el bit de paridad para la combinación 01110101, y mediante otro verificar que la combinación con el bit de paridad incluído presenta la paridad establecida.

26 En un flip flop **D** establecer qué sucede si se conecta permanentemente la salida **Q** negada a la entrada **D**

27 Verificar que el siguiente diagrama temporal es correcto para un flip flop **D** "latch"



28 En un registro universal se quiere entrar en paralelo la información 1100, dividirla por dos, y luego sacar el resultado en serie. Indicar los pasos a realizar, cómo se desplaza la información, y cuántos pulsos reloj son necesarios.

Respuestas de problemas seleccionados de esta sección se encuentran al final de la presente unidad

COMPLEMENTO: SIMPLIFICACION DE EXPRESIONES CON DIAGRAMAS DE KARNAUGH

K.1 Introducción

En la sección A1.16.2 se vió la forma de obtener a partir de una tabla de verdad, una suma de minitérminos que permite construir un circuito con dos niveles de compuertas que cumpla con dicha tabla. Existen otras expresiones con menor número de compuertas y de entradas por compuerta que también pueden expresar la misma tabla de verdad. Las mismas pueden hallarse mediante los denominados **diagramas de Veitch-Karnaugh**

El diagrama de Veitch-Karnaugh[1] *constituye otra forma de representar una función,* **siendo en esencia una tabla de verdad en dos dimensiones.**

Por la disposición de las variables en el diagrama, es posible hallar rápida y mecánicamente una expresión sencilla de la función representada, mediante un adecuado agrupamiento de los "unos" o de los "ceros". Esta simplificación o *"minimización"* permite determinar entre las expresiones del tipo suma de productos (SP) o producto de sumas[2], las más simples, correspon-diéndoles circuitos con dos niveles que presentan el *menor número de compuertas y de entradas por compuerta.*

El diagrama de Karnaugh es además representativo de métodos de simplificación –como el de Quine-Mc Cluskey– para un número muy grande de variables, que hoy día se llevan a cabo usando computadoras

La utilización de los diagramas de Karnaugh se limita a funciones de hasta 5 variables como máximo, debido a que se torna difícil la minimización de funciones más complejas.

En las secciones siguientes se define la estructura del diagrama, y luego se procede a mostrar como se **minimizan** funciones, suponiendo conocida en un principio su tabla de verdad o su expresión como SP minitérminos o una expresión cualquiera de ella. Más adelante se indica como **representar** en el diagrama funciones expresadas como SP

Mediante ejercicios con dificultades progresivas, se van obteniendo en cada ejemplo una SP mínima (o más si hay varias soluciones mínimas), mostrando al principio el paralelismo con el engorroso método algebraico. Los ejemplos sirven para definir los "**implicantes primos**" y enunciar un método general de minimización para el diagrama.

K.2 Estructura del diagrama

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

Figura.K.1

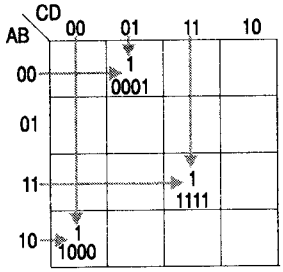


Figura K.2.a

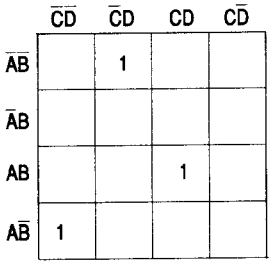


Figura K.2.b

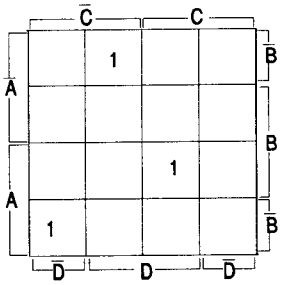


Figura K.2.c

Con el objeto de mostrar la analogía entre una tabla de verdad y el diagrama, se hallará el diagrama correspondiente a la tabla de una función elegida arbitrariamente.

Esta (figura K.1) o cualquier otra puede representarse en dos "coordenadas", constituídas por las filas y columnas de un diagrama (figura K.1.a), en lugar de utilizar una columna por variable como en la figura K.1

De esta manera, de acuerdo con las 16 combinaciones o filas de la tabla, resul-tan igual número de celdas en el diagrama, pudiéndose escribir dentro de cada una el valor correspondiente de la función. Así, (figura K.2.a), la celda intersección de la fila 00 con la columna 01 identifica la combinación 0001, para la cual la función vale **1** según la tabla.

De igual forma se ha determinado la ubica ción de los restantes "unos".

Por claridad no se escriben los ceros de la función en el diagrama.

Por consiguiente, una función expresada por su tabla de verdad puede representarse *directamente* en un diagrama de Veitch o Karnaugh

1 Veitch y Karnaugh son dos nombres equivalentes que se da a este diagrama, siendo que el segundo se popularizó más para los circuitos.

2 Este tipo de expresión no se tratará en esta unidad. El lector interesado puede consular "Introducción a las Técnicas Digitales con CI" de M. Ginzburg

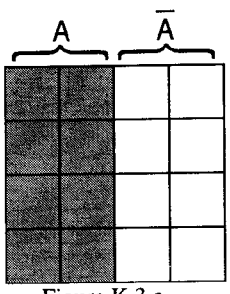


Figura K.3.a

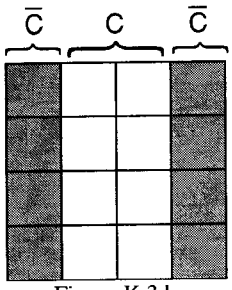
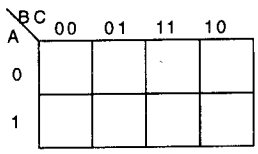


Figura K.3.b

Los diagramas de las figuras K.2.a,.b y.c son equivalentes entre sí, por tener igual disposición de variables, y son a su vez equivalentes a la tabla de la figura K.1, observándose en los mismos igual distribución de "unos".En cualquiera de las tres formas se ha representado una función mediante un diagrama, la cual antes fue expresada por su tabla.
El diagrama de la figura K.2.a, cuyas filas y columnas se identifican con los valores de las variables que aparecen en el ángulo superior izquierdo, se debe a **Karnaugh,** siendo el de la figura K.2.c, con las variables como coordenadas, fue ideado por **Veitch.**



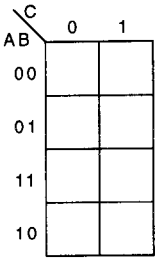


Figura K.4

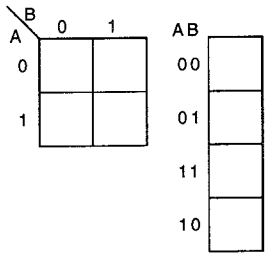


Figura K.5

Las variables horizontales y verticales pueden elegirse arbitrariamente, pero su dis-posición debe respetar las contigüidades: si se reserva para la variable **A** las dos primeras columnas (figura K.3.a), la variable $\overline{A}$ debe ocupar las dos columnas restantes.
Si se quiere ubicar a **C** como la otra variable vertical (figura K.3.b) deben reservarse para **C** las dos columnas centrales, resultando para $\overline{C}$ las columnas extremas. De esta forma, los espacios de **C** o $\overline{C}$ no se superponen totalmente con los de **A** y $\overline{A}$. Diagramas de 3 y 2 variables se dan en las figuras K.4 y K.5, respectivamente.

K.3 Obtención de la suma de minitérminos a partir de un diagrama

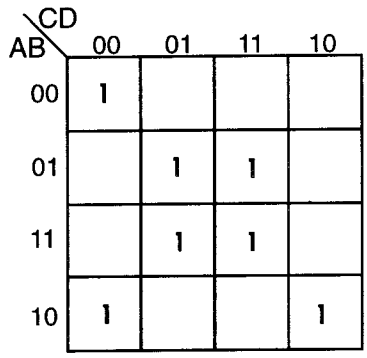


Figura K.6

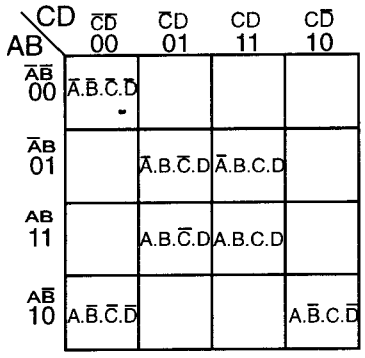


Figura K.7

Puesto que el diagrama es una tabla de verdad, si se desea hallar los minitérminos de la función representada en el mismo, se procede igual como hizo a partir de una tabla (sección A1.16.1)
Esto es, para cada celda con un "uno" de la función se hallan sus coordenadas, a fin de determinar la combinación de variables para la cual la función vale **1**. Obtenida la misma, se forma el minitérmino correspondiente según la regla de la sección A1.16.1: cuando la variabla vale **0** o **1** se escribe dicha variable negada o sin negar, respectivamente.
Entonces, los tres unos de la función de la figura K.2.a. cuyas coordenadas son las combinaciones de valores 0001, 1000 y 1111 de las variables **A,B,C,D** corresponden a la siguiente función expresada como suma de minitérminos: $Z = \overline{A}.\overline{B}.\overline{C}.D + A.\overline{B}.\overline{C}.\overline{D} + A.B.C.D$

Otra función cualquiera, como la representada en el diagrama de la figura K.6, será la suma de los minitérminos correspondientes a cada "uno"de la función (figura K.7)

Propiedad de los miniterminos correspondientes a "unos" adyacentes en el diagrama

Es importante notar que por la disposición de las variables en el diagrama, dos "unos" adyacentes, o sea vecinos horizontal o verticalmente (no en diagonal), tendrán los minitérminos que le corresponden formados por productos *que sólo difieren en una variable*, que está afirmada en un minitérmino y negada en el vecino. *Las restantes variables son comunes.*

Por ejemplo (figura K.7) los miniterminos $\overline{A}.B.\overline{C}.D$ y $\overline{A}.B.C.D$ difieren en la variable **C**
$\overline{A}.B.\overline{C}.D$ y $A.B.\overline{C}.D$ " **A**

Son también adyacentes $A.B.\overline{C}.D$ y $A.\overline{B}.\overline{C}.\overline{D}$, así como $A.\overline{B}.\overline{C}.D$ y $\overline{A}.\overline{B}.\overline{C}.\overline{D}$. En general:

Dos minitérminos correspondientes a dos "unos" adyacentes difieren en una variable

En esta propiedad se basa el método de simplificación que se usa en los diagramas.

K.4 Representación en el diagrama de una suma de minitérminos

Si se conoce la expresión de una función dada por sus minitérminos, resulta inmediata su **representación** en el diagrama, o sea la determinación de las coordenadas de las celdas que contendrán los "unos" de la función.

Se trata de efectuar el proceso inverso al efectuado en la sección anterior. Ahora, dada una función como SP minitérminos, se determina la combinación para la cual cada minitérmino de la misma resulta **1**, y luego se usan los valores de cada combinación como coordenadas para determinar la celda que contendrá un **1**.

Así, partiendo de $\mathbf{Z = \overline{A}.\overline{B}.C.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + A.B.C.D}$, esta función vale **1** para las combinaciones 0010, 1000 y 1111, las cuales serán las coordenadas de los tres "unos" de la figura K.2.a Teniendo una función representada, se puede ver si es minimizable o no.

Por consiguiente una función expresada como una suma de minitérminos puede representarse directamente en el diagrama de Karnaugh

K.5 Minimización de funciones usando el diagrama

K.5.1 Introducción al problema de la minimización

Las sumas de productos (SP), como ser $\mathbf{\overline{A}.\overline{B}.C + \overline{A}.\overline{B} + \overline{A}.C + B.C + B.A}$ se sintetizan en conexionados con dos niveles de compuertas: **AND-OR** o **NAND-NAND** (sección A1.16.2- P_{10}: Propiedades de De Morgan). Poseen las siguientes propiedades:

- Son expresiones simples, *con negaciones que sólo afectan a variables individuales, nunca a operaciones*, siendo así más comprensibles para expresar el funcionamiento lógico de un circuito.
- La síntesis de una SP es directa: consiste en compuertas de un mismo tipo que concurren a otra que también puede ser del mismo tipo, como en el caso **NAND-NAND**

Circuitos equivalentes mínimos del tipo sumas de productos (SP):

Las SP tienen, además de las propiedades citadas, una importancia fundamental en la búsqueda de las denominadas *expresio-nes equivalentes mínimas,* para lograr circuitos más simples, con mínimo número de compuertas y de entradas por compuerta

Su determinación algebraica se conoce como ***"minimización\"*** o simplificación de funciones lógicas.

En general, el objetivo de las equivalencias que se obtienen aplicando De Morgan es transformar un circuito de un tipo de compuertas en otro, pero sin alterar la complejidad del circuito originario como se pretende ahora. Puede apreciarse así las bondades del álgebra de Boole para lograr circuitos diferentes de funcionamiento equivalente.

La búsqueda de circuitos más simples se corresponde algebraicamente con la de expresiones equivalentes *mínimas*, que contengan el menor número de operaciones lógicas y de literales (letras)

Se han desarrollado métodos sistemáticos de minimización *para obtener* ***SP mínimas***[1]***,a partir de*** *funciones expresadas por SP, por suma de minitérminos o por su tabla de verdad* que además correspondan a circuitos con dos niveles de compuertas de retardo, como **AND-OR** (equivalente a **NAND-NAND**).

Si la expresión a simplificar no es del tipo SP, habrá que convertirla primero en alguna de esas formas (como se ejemplifica a continuación), *o hallar su tabla de verdad* (como se realiza en el ejemplo de la sección K.5.2), *dado que no existe un procedimiento general para minimizar cualquier expresión de una función.*

Una SP sera *mínima* si no existe otra SP con menor número de sumandos, ni otra de igual número de sumandos, pero con menor cantidad de literales (letras).

Puede haber dos o más SP igualmente mínimas en cantidad de variables y operaciones AND y OR.

Los métodos de minimización desarrollados garantizan encontrar una o varias SP mínimas entre todas las SP equivalentes de una función, conforme al criterio anterior.

A tal fin el diagrama de Karnaugh (hasta 5 variables), y los métodos computacionales evitan los largos y engorrosos procedimientos de minimización algebraica

Otros caminos algebraicos ***no generalizables*** *en ciertos casos puede proporcionar expresiones equivalentes más económicas que las SP mínimas determinadas con los métodos sistemáticos a tratar, pero no serán SP*

Como se estableció en la sección K.1, *las expresiones mínimas obtenidas mediante los diagramas permiten determinar circuitos combinacionales con el menor número de compuertas y/o de entradas por compuertas, que serán los más sencillos entre todos los circuitos posibles con dos niveles de retardo, pudiendo existir en ciertos casos varias soluciones mínimas.*

---

1 No debe confundirse una SP mínima con una SP miniitérminos. Sólo pueden coincidir en alguna función cuya SP minitérminos (suma de minitérminos) no se puede minimizar, como es el caso de la función ejemplificada en la sección A1.21

K.5.2 Alternativas de representación de una función a minimizar en un diagrama

Para que una función pueda simplificarse usando el diagrama, es necesario *representarla* en el mismo.
La función a representar en general puede estar expresada en alguna de estas formas:

- Por su tabla de verdad
- Algebraicamente, en forma canónica (SP minitérminos)
- Idem, en forma de SP genérica no canónica.
- En una forma algebraica cualquiera que no sea SP

Hasta el presente el lector conoce la forma de representar en el diagrama una función dada por su tabla (sección K.2), y también si se parte de una expresión de la misma como suma de minitérminos (sección K.4)

Si está expresada como SP no canónicas, en las secciones K.5.6 se tratará la manera *directa* de representarla.
En la cuarta de las formas enumeradas, una forma rápida de representarla es *convertirla en una SP* operando algebraicamente como se llega en el cuarto paso del problema 16 resuelto (sección A1.34), y luego emplear el método citado en el párrafo anterior.

Otra alternativa conocida, tratada en la sección A1.13.1, consiste en hallar a partir de esa expresión la tabla de verdad, o mejor *directamente el diagrama*, pues éste es una tabla en dos dimensiones, para lo cual se debe calcular el valor de la función para cada combinación de valores de sus variables.

AB \ CD	00	01	11	10
00		1		
01		1		1
11		1	1	
10		1		

Figura K.8

þ Por ejemplo si se parte de la función

$$Z = \overline{\overline{D} + [(B + C).(\overline{A} + \overline{B} + \overline{C})]} + B.\overline{(C + \overline{D}).\overline{A.C.D}}$$

para la combinación 0000 resulta: $Z = \overline{\overline{0} + [(0 + 0).(\overline{0} + \overline{0} + \overline{0})]} + 0.\overline{(0 + \overline{0}).\overline{0.0.0}} = 0$

para la combinación 0001 resulta $Z = \overline{\overline{1} + [(0 + 0).(\overline{0} + \overline{0} + \overline{0})]} + 0.\overline{(0 + \overline{1}).\overline{0.0.1}} = 1$

y así sucesivamente pueden obtenerse todos los valores de la función.

En el diagrama de la figura K.8 para la combinación correspondiente a cada celda se ha escrito el valor de la función calculado para ella. De esta forma se habrá representado la función. Un método más rápido de representación se trata en la sección K.5.6.

5.5.3 Mecanismo de minimización del diagrama para lograr SP mínimas

þ **EJEMPLO 1**
Supondremos una función representada en el diagrama de la figura K.9.a
*A fin de justificar el procedimiento que se realizará para obtener una SP **mínima**,* en las celdas de dicha figura donde la función vale **1** aparece el minitérmino correspondiente a la combinación de valores de variables determinada por sus coordenadas. A continuación se expresa la función como una suma de esos minitérminos (sección K.3).[1]
El diagrama de la figura K.9.a está representando la función:

1) $Z = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + A.B.C.\overline{D} + \overline{A}.B.\overline{C}.D + A.\overline{B}.C.\overline{D}$

Si se realizara la simplificación por el método algebraico, habría en principio que reordenar la sumatoria, para agrupar los minitérminos que difieren en una variable:

2) $Z = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + A.B.C.\overline{D} + A.\overline{B}.C.\overline{D}$ para poder hacer:
3) $Z = \overline{A}.\overline{B}.\overline{C}.(\overline{D} + D) + \overline{A}.B.\overline{C}.(\overline{D} + D) + A.C.\overline{D}.(B + \overline{B})$ conforme a P_2
Puesto que $(\overline{D} + D) = 1$ y $(B + \overline{B}) = 1$ por P_4, resulta
4) $Z = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.C.\overline{D}$ y nuevamente puede hacerse
5) $Z = \overline{A}.\overline{C}.(\overline{B} + B) + A.C.\overline{D} = \overline{A}.\overline{C} + A.C.\overline{D}$ (SP mínima)

Estos cinco pasos se pueden hacer *sin usar álgebra* en el diagrama de la figura K.9.b que por razones didácticas superpone las convenciones de Veitch y Karnaugh. El ordenamiento de minitérminos realizado en 2) con el fin de eliminar variables, se realiza simplemente en este diagrama *enlazando los "unos" que sean **adyacentes*** (sección K.3) Por ejemplo, los 4 primeros sumandos minitérminos seleccionados en el paso 2), son directamente los correspondientes a los 4 unos adyacentes enlazados en la parte superior del diagrama; y los dos siguientes se corresponden con los 2 "unos" adyacentes enlazados en la parte inferior.

1 Tanto la escritura de los minitérminos en las celdas, como el desarrollo algebraico que sigue, no es necesario realizarlos en la práctica

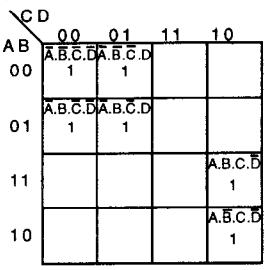


Figura K.9.a

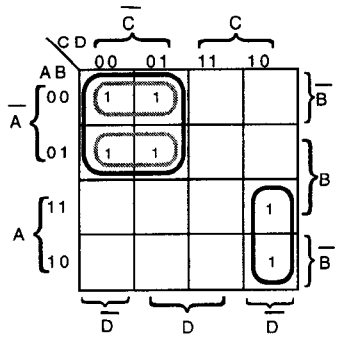


Figura K.9.b

Comenzando con los dos minitérminos correspondientes a este último lazo, en el paso 3) entre ambos se saca factor común $\mathbf{A.C.\overline{D}.(B+\overline{B})}$ para que en el paso 4) desaparezca **B** y quede sólo $\mathbf{A.C.\overline{D}}$. En el diagrama estos pasos se realizan visualmente, ***hallando las variables que son coordenadas del lazo y luego for mando su producto.***

Asi, el lazo con los dos "unos" en la convención de Veitch tiene como coordenadas: horizontal la variable **A**, y vertical a la columna común a las variables **C** y $\overline{\mathbf{D}}$. Formando el producto de estas tres variables se obtiene directamente el resultado parcial $\mathbf{A.C.\overline{D}}$ antes hallado algebraicamente.

En el diagrama de Karnaugh las variables coordenadas de un lazo se determinan –en la obtención de SP mínimas– con la convención de los minitérminos [2] *: **0** $\Leftrightarrow \overline{A}$ y **1** $\Leftrightarrow A$ (lo mismo para cualquier otra variable).*

El lazo en cuestión por un lado ocupa la columna 10, coordenadas binarias verticales comunes a las dos celdas del lazo, que para formar el producto buscado corresponderán a las variables **C** y $\overline{\mathbf{D}}$. Por otro lado, una celda del lazo está en la fila 11 y la otra en la 10, manteniéndose el 1 común a ambas celdas cuando se pasa de una a otra. La coordenada horizontal del lazo, con la convención para los productos, es entonces **A, única variable que mantiene su valor en las dos filas donde se encuentra el lazo**, siendo que **B** (*variable que desaparece para este lazo*) por cambiar de valor de una fila a la otra, no es coordenada común horizontal de ambas celdas del lazo. Entonces, si se forma el producto $\mathbf{A.C.\overline{D}}$ con las coordenadas halladas, llegamos al mismo resultado anterior.

*En general, enlazando **2** celdas adyacentes en un diagrama, se elimina **una** variable en el producto correspondiente a ese lazo*

Respecto del lazo con las 4 celdas adyacentes, cuyos minitérminos correspondientes dan lugar en el paso 5) al sumando minimizado $\mathbf{\overline{A}.\overline{C}}$, éste también se obtiene formando el producto de las variables que son las coordenadas del lazo en el diagrama con la convención de Veitch, como puede verificarse.

Con la convención de Karnaugh en las dos filas y columnas donde está el lazo, las variables **A** y **C** mantienen su valor (**0**), correspondiendo a estos valores las variables $\overline{\mathbf{A}}$ y $\overline{\mathbf{C}}$, respectivamente, conforme la convención establecida, por lo que con las mismas se forma el producto $\mathbf{\overline{A}.\overline{C}}$. Se han eliminado las variables **B** y **D**, que cambiaban de valor en la determinación de las coordenadas de las filas y columnas del lazo.

*En general, enlazando **4** celdas adyacentes en un diagrama, se eliminan **dos** variables en el producto correspondiente a ese lazo*

Dentro del lazo con 4 celdas se han dibujado dos lazos con dos celdas (en grisado).

> ***Si por error** en vez de tomar un lazo de cuatro que los engloba, se hubieran tomado 2 lazos de dos, sus variables coordenadas formarían los productos $\mathbf{A.\overline{B}.C}$ y $\mathbf{A.B.\overline{C}}$, como se puede verificar, correspondientes al paso 4), por lo que la minimización estaría **incompleta***[1]

Conforme con este paso, se hubieran formado dos productos de 3 variables, en lugar de un solo producto de dos variables, como se llega en el paso 5) y como también se obtiene con las coordenadas del lazo de 4 celdas que las engloba.

Recapitulación de la metodología empleada

En síntesis, dejando de lado el procedimiento de minimizacíon algebraico, el diagrama permite llegar al mismo resultado, siguiendo los siguientes pasos *generales*:

---

2 En realidad esta convención es la que establece la correspondencia entre cualquier producto -sea minitérmino o no- y la combinación para la cual el mismo resulta 1. Obsérvese al respecto que en una SP, un minitérmino hace 1 una función para una única combinación de todas las variables, mientras que un producto compuesto por menos variables hace 1 la función para mas de una combinación de todas las variables.

Así, el sumando $\mathbf{A.C.\overline{D}}$, ya sea en el paso 4) ó 5), hace que sea **Z** = 1 siempre que **A** = 1 **C** = 1 y $\overline{\mathbf{D}}$ = 0, independientemente del valorde B. Esto ocurre para las 2 combinaciones 1110 y 1010 de valores de A,B,C,D , coordenadas de las 2 celdas adyacentes enlazadas, pudiéndose simbolizar ambos casos así 1X10, donde X significa que B puede valer 1 ó 0.

En cualquier SP que exprese una función –sea de minitérminos, sea mínima, u otra equivalente cualquiera– los productos de la sumatoria deben ser tales que ésta resulte 1 para todas las combinaciones en que en la tabla de verdad dicha función vale 1.

De ahí la concordancia de las reglas de formación de minitérminos y otros productos.

1 El mismo tipo de error se hubiera cometido si los dos lazos comprendidos dentro del lazo de 4 hubieran sido los dos que unen las celdas adyacentes verticales en vez de las adyacentes horizontales.

1. **Se enlazan las celdas con "unos" adyacentes, *buscando lazos que engloben a otros***
2. **Cada sumando de la expresión minimizada, es el producto que se forma con las variables que son las coordenadas de cada lazo.considerado.**
3. **La expresión SP mínima buscada, *es la suma todos los productos hallados* según el paso anterior**

En el presente ejemplo se ha obtenido $\mathbf{Z = \overline{A}.\overline{C} + A.C.\overline{D}}$ como en la sumatoria del paso 5) algebraico.

þ EJEMPLO 2 ("Unos" compartidos)

Ahora la función a minimizar está representada en el diagrama de la figura K.10.a Como en el ejemplo anterior primero la minimizaremos algebraicamente, obteniendo en el paso 1) la función como una suma de minitérminos correspondientes a los "unos" del diagrama de la misma:

1) $\mathbf{Z = \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.\overline{D} + A.B.\overline{C}.D + A.B.C.D + A.\overline{B}.\overline{C}.D}$
Agrupando miniterminos que difieren en una variable y repitiendo $\mathbf{A.B.\overline{C}.D}$ conforme a $\mathbf{P_6}$:

2) $\mathbf{Z = \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.B.\overline{C}.D + A.B.\overline{C}.D + A.\overline{B}.\overline{C}.D + A.B.\overline{C}.D + A.B.C.D + \overline{A}.B.C.\overline{D}}$
Sacando factor común se obtiene:

3) $\mathbf{Z = \overline{A}.\overline{C}.D.(\overline{B} + B) + A.\overline{C}.D.(B + \overline{B}) + A.B.D.(\overline{C} + C) + \overline{A}.B.C.\overline{D}}$
Por $\mathbf{P_4}$ resulta:

4) $\mathbf{Z = \overline{A}.\overline{C}.D + A.\overline{C}.D + A.B.D + \overline{A}.B.C.\overline{D}}$
Sacando nuevamente factor común:

5) $\mathbf{Z = \overline{C}.D.(\overline{A} + A) + A.B.D + \overline{A}.B.C.\overline{D}}$
Por $\mathbf{P_4}$ resulta:

6) $\mathbf{Z = \overline{C}.D + A.B.D + \overline{A}.B.C.\overline{D}}$ que es la SP mínima buscada[2]

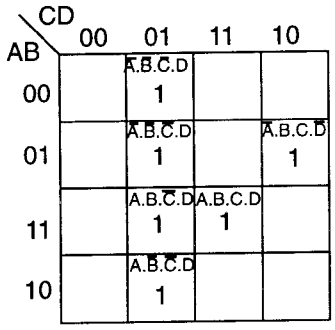


Figura K.10.a

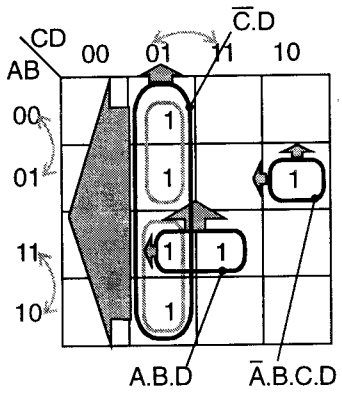


Figura K.10.b

En el diagrama de la figura K.10.b se han formado los lazos en trazo lleno, buscando siempre enlazar el mayor número de "unos" adyacentes vertical u horizalmente.
El lazo que ocupa la columna 01 engloba a los dos lazos en grisado y corresponde a los 4 primeros minitérminos del paso 2). El lazo con los 2 "unos" adyacentes horizontales, comparte el "uno" del minitérmino $\mathbf{A.B.\overline{C}.D}$ con el lazo vertical (que tuvo que repetirse en ese paso). Por lo tanto, si se enlazan bien los "unos", esto no sólo es equivalente a ordenar los minitérminos como en el paso 2), sino que además se está aplicando automáticamente $\mathbf{P_6}$ (idempotencia) cuando un lazo comparte uno o más "unos" con otro.
Las coordenadas verticales del lazo con 4 celdas son las variables $\overline{\mathbf{C}}$ y $\mathbf{D}$, correspondientes a la columna 01 del diagrama, con la convención de minitérminos, dado que ellas bastan para localizarlo. Con las mismas se forma directamente el producto $\overline{\mathbf{C}}.\mathbf{D}$ hallado en el paso 6).

Asimismo, si se recorre verticalmente el lazo a través de todas sus celdas, ninguna variable horizontal es común a todas las filas, dado que tanto la variabla **A** como la **B** cambian de valor en dicho recorrido, por lo que no aparecen en la formación de $\overline{\mathbf{C}}.\mathbf{D}$. Esto resulta acorde con el hecho de que un lazo con 4 celdas permite eliminar dos variables en un producto.

Como en el ejemplo 1, si en lugar del lazo que engloba a los dos lazos de 2 celdas se hubieran considerado estos dos, la minimización hubiese sido parcial, equivalente al paso 4) algebraico.

El "uno" de la celda de coordenadas 0110 no tiene celdas adyacentes que permitan eliminar variables, por lo que da lugar a un lazo que sólo encierra a ella. Sus coordenadas forman el producto minitermino $\mathbf{A.B.C.\overline{D}}$

> *En general, enlazando **1** celda aislada en un diagrama, no se elimina **ninguna** variable en el producto correspondiente a ese lazo, por lo que resulta un minitérmino*

Sumando los tres productos determinados por los tres lazos, se llega a la SP mínima del paso 6)

2 Obsérvese que una SP mínima puede tener uno o más minitérminos

þ **EJEMPLO 3 (Celdas adyacentes por el exterior del diagrama)**

Expresar como una SP mínimos la función dada por los "unos" del diagrama de la figura K.11.a

Habiendo comprendido a través de los ejemplos anteriores el paralelismo entre la minimización algebraica y los pasos que para tal fin se realizan en el diagrama, en lo que sigue dejaremos de lado la primera y operaremos directamente en el diagrama, suponiendo una función a minimizar ya representada en él.

En la figura K.11.b aparece una minimización **mal** realizada por que como se verá, los lazos dibujados pueden ser englobados por otros mayores. Para empezar, los dos lazos de 4 celdas originarían los productos $\overline{A}.\overline{C}$ y $A.\overline{C}$. Si éstos aparecieran en un paso algebraico formando parte de una SP, se podría hacer la siguiente simplificación $\overline{A}.\overline{C}+A.\overline{C}=\overline{C}(\overline{A}+A)=\overline{C}.1=\overline{C}$

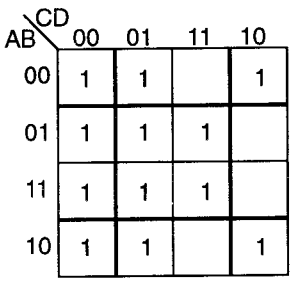


Figura K.11.a

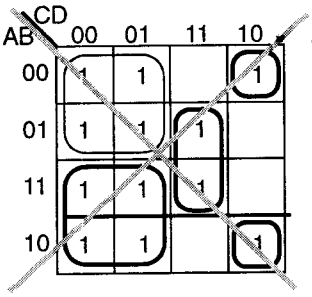


Figura K.11.b

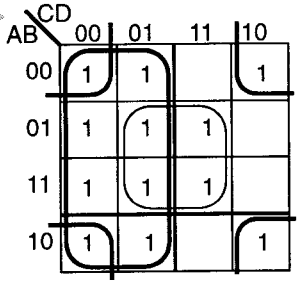


Figura K.11.c

Vale decir, que dos lazos adyacentes de 4 celdas se pueden englobar en un solo lazo de 8 celdas como aparece en la figura K.11.c. Se han eliminado en este caso las variables **A, B** y **D**, pues si se recorre verticalmente el lazo desde una celda extrema hasta la otra, como en el ejemplo anterior, ninguna variable horizontal **A** ó **B** es común a todas las celdas.

Verticalmente, al pasar de la columna 00 a la 01, se mantiene el valor subrayado común a ambas, correspondiente a la variable $\overline{C}$ Esta será la única a considerar para formar el producto, o sea que en lugar de éste se tiene una sola variable para la sumatoria total de los productos de cada lazo.

> En general, enlazando **8** celdas adyacentes en un diagrama, se eliminan **tres** variables en el producto correspondiente a ese lazo.

Siguiendo con la minimización, el lazo con 2 celdas de la figura K.11.b puede ser englobado por un lazo de 4 (figura K.11.c), que agrega dos celdas adyacentes a las primeras, compartidas con el lazo de 8. Las coordenadas de este lazo de 4 son **B** y **D,** con las cuales se forma el producto **B.D**

Las dos celdas de la columna 10, que en la figura K.11.b se habían enlazado aisladamente, en la figura K.11.c aparecen por un lado enlazadas entre sí por un lazo, que así dibujado quiere simbolizar que se cierra por el exterior del diagrama, por que las celdas que lo constituyen son adyacentes por el exterior del mismo; y por otro lado este lazo las agrupa con la otras dos de las esquinas del diagrama de la columna 00, donde existen dos "unos" compartidos con el lazo de 8.

Cuando se pasa de la columna 00 a la 10, ambas tienen en común el valor subrayado, correspondiente a la variable $\overline{D}$; y al pasar de la fila 00 a la fila 10 que también contienen estas celdas, ellas tienen en común el valor correspondiente a la variable $\overline{B}$. Por lo tanto, el producto correspondiente a este lazo de 4 celdas es $\overline{B}.\overline{D}$, habiéndose eliminado **A** y **C**.

En definitiva, la función minimizada como SP será, sumando los productos antes formados: $Z=\overline{C}+B.D+\overline{B}.\overline{D}$

La adyacencia ejemplificada puede visualizarse en el cilindro dibujado al pie de página[1], estas 4 celdas de las esquinas son adyacentes entre sí.

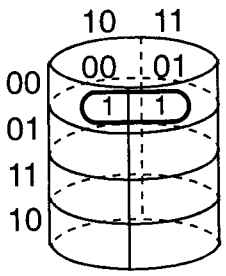
Recapitulación de los objetivos de la minimización usando el diagrama

De los ejemplos resulta que el "juego" consiste en enlazar todos los "unos" de un diagrama según el siguiente principio:

> Utilizar el **menor** número posibles de lazos, siendo que cada lazo abarque la **mayor** cantidad de celdas adyacentes que sea permitido agrupar.

Cuanto más grande es el lazo, más variables desaparecerán del producto a formar

1



þ **.EJEMPLO 4 (Lazo redundante)**:

Minimizar como SP la función dada por los "unos"del diagrama de la figura K.12.a

En dicha figura se han enlazado todos los "unos" adyacentes como corresponde, buscando que cada lazo sea lo mayor posible. Conforme a la metodología establecida, sumando los productos correspondientes a los lazos dibujados, resulta la siguiente SP:

$$Z = \overline{C}.D + \overline{A}.\overline{B}.D + \overline{A}.B.\overline{C} + A.B.D + A.\overline{B}.\overline{C}$$

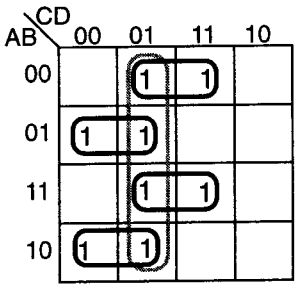


Figura K.12.a

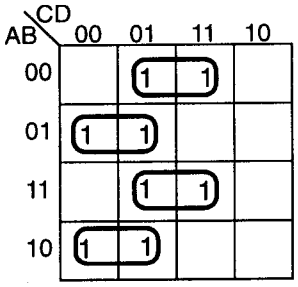


Figura K.12.b

que *no* es mínima, dado que si bien se han tenido en cuenta todos los lazos mayores posibles, *en este caso con un subconjunto de ellos se pueden cubrir todos los "unos"*, como se indica con los lazos en trazo lleno de la figura K.12.b. Así se cumple con el objetivo central de que todos los "unos" queden enlazados con el *menor* número de lazos posibleś (cada lazo implica una compuerta en el circuito), por lo que la SP mínima es:

$$Z = \overline{A}.\overline{B}.D + \overline{A}.B.\overline{C} + A.B.D + A.\overline{B}.\overline{C}$$

El lazo con 4 celdas dibujado en trazo no lleno, correspondiente al producto $\overline{C}.D$ es ***redundante***, pues agrega un sumando innecesario, dado que para cada una de las combinaciones (de valores de **A,B,C,D**) 0001, 0101, 1101 y 1001 para las cuales $\overline{C}.D=1$, en la SP no mínima anterior resulta que también siempre otro de los productos resulta **1**; por lo que en estos 4 casos será **Z=1** porque dos sumandos valen **1**, siendo que en una suma basta que uno solo valga **1** para que el resultado sea **1**. Así:

$$Z = \overline{C}.D + \overline{A}.\overline{B}.D + \overline{A}.B.\overline{C} + A.B.D + A.\overline{B}.\overline{C}$$

para 0001 $Z = \overline{0}.1 + \overline{0}.\overline{0}.1 + \overline{0}.0.\overline{0} + 0.0.1 + 0.\overline{0}.\overline{0} = 1+1+0+0+0 = 1$

para 0101 $Z = \overline{0}.1 + \overline{0}.\overline{1}.1 + \overline{0}.1.\overline{0} + 0.1.1 + 0.\overline{1}.\overline{0} = 1+0+1+0+0 = 1$

para 1101 $Z = \overline{0}.1 + \overline{1}.\overline{1}.1 + \overline{1}.1.\overline{0} + 1.1.1 + 1.\overline{1}.\overline{0} = 1+0+0+1+0 = 1$

para 1001 $Z = \overline{0}.1 + \overline{1}.\overline{0}.1 + \overline{1}.0.\overline{0} + 1.0.1 + 1.\overline{0}.\overline{0} = 1+0+0+0+1 = 1$

þ **EJEMPLO 5 (Dos soluciones mínimas)**

Minimizar la función dada por el diagrama de la figura K.13.a

Las figuras K.13.b y K.13.c que discutiremos en este ejemplo, permiten dos soluciones igualmente mínimas, dado que en ambas se cubren todos los "unos" con 3 lazos de 4 celdas.

Se han mostrado dos soluciones *erróneas* en las figuras K.13.d y K.13.e : en la primera por que si bien resultan 3 lazos, uno de ellos abarca dos celdas, pudiendo enlazar 4, como en la figura K.13.b; y en la otra por que no es factible enlazar 6 celdas

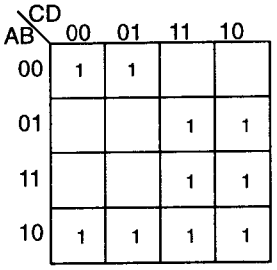


Figura K.13.a

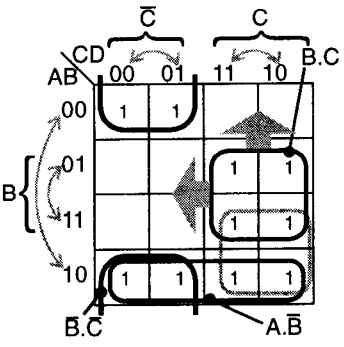


Figura K.13.b

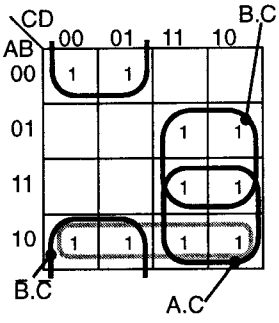


Figura K.13.c

En la figura K.13.b, al lazo que ocupa la fila 10 le corresponde el producto $A.\overline{B}$, y al que ocupa la parte central, el producto **B.C,** como puede verificarse.
El lazo partido, que abarca 4 celdas adyacentes por el exterior por un lado ocupa las columnas 00 y 01 manteniéndose el valor 0 de una variable al pasar de una celda cualquiera a otra.
Le corresponde entonces verticalmente la coordenada $\overline{C}$, desapareciendo la variable **D**, que cambia de valor de una columna a otra. Asimismo, ocupa las filas 00 y 10 , o sea se mantiene al pasar de una a otra el 0 correspondiente a $\overline{B}$, desapareciendo **A**. Entonces, con este lazo se forma el producto $\overline{B}.\overline{C}$.

Con los lazos en trazo lleno se han cubierto todos los "unos" del diagrama, acorde con el principio de usar el menor número de lazos, por lo que si bien el lazo dibujado en grisado encierra 4 celdas adyacentes, el mismo es *redundante*, pues agregaría un sumando no indispensable en la expresión mínima buscada.

La SP mínima que surge de este diagrama es: $Z = A.\overline{B} + B.C + \overline{B}.\overline{C}$

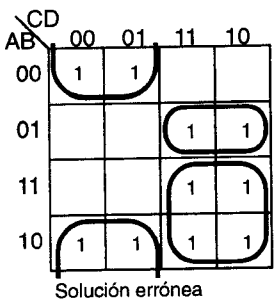


Figura K.13.d

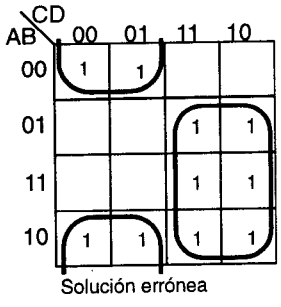


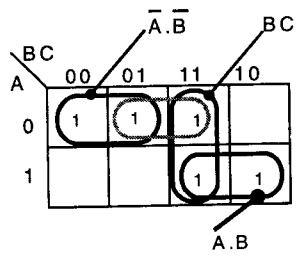
Figura K.13.e

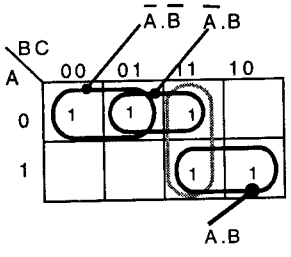
Otra SP mínima resulta para la misma función del diagrama de la figura K.13.c en el cual se ha usado el lazo correspondiente al producto **A.C,** como puede verificarse, que en la figura anterior resultó redundante respecto de los otros. Ahora resulta redundante el lazo generador del producto $\mathbf{A.\overline{B}}$ en el resultado anterior (en grisado en este diagrama)
La segunda SP mínima equivalente a la anterior, teniendo en cuenta los productos correspondientes a los lazos que se han vuelto a usar es:

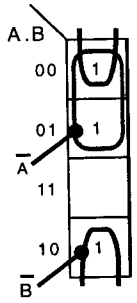
$$\mathbf{Z = A.C + B.C + \overline{B}.\overline{C}}$$

- **EJEMPLO 6 (Diagramas para 3 y 2 variables)** Minimizar la función $\mathbf{Z = \overline{\overline{A.\overline{B}.C}.(A+B.\overline{C})+B.(C+A)}}$

Para representar la función en el diagrama, procederemos como en la sección K.5.2: para cada combinación de valores de las variables **A,B,C** desde 000 a 111 calculamos el valor de **Z**, resultando los "unos" y "ceros" del diagrama de la figura K.14, donde se han indicado, para las dos soluciones mínimas halladas, el producto asociado a cada lazo.
También (figura K.15) sería posible simplificar una función en un diagrama para dos variables







← Figura K.15

OTROS EJEMPLOS: En las figuras siguientes se han minimizado las funciones indicadas, supuestas previamente representadas por los "unos"de cada diagrama. Los lazos redundantes aparecen en grisado.

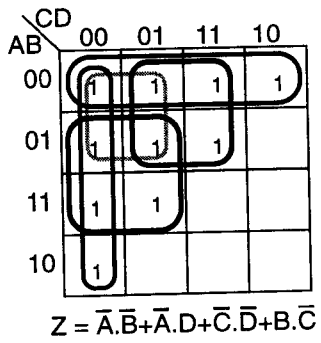


Figura K.15.a

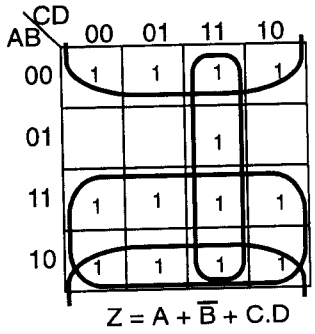


Figura K.15.b

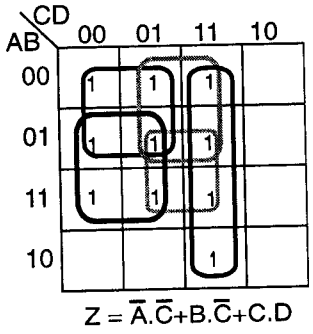


Figura K.15.c

K.5.4 Implicantes primos en la minimización

Antes de indicar algunas reglas generales de minimización es conveniente dar algunas definiciones técnicas que permitirán enunciarlas más sencilla y precisamente.

Así, *todos* los lazos bien usados en los ejemplos anteriores se denominan ***"implicantes primos"*** (IP), cuyo significado en los diagramas se refiere a *agrupaciones de* ***máximo*** *número posible de celdas adyacentes*, tales que el lazo que encierra a cada agrupación no puede ser englobado por otro lazo que permita una agrupación mayor de celdas. Los IP dan lugar a productos con el menor número de variables, objetivo de la minimización.

Por ejemplo, los dos agrupamientos de dos celdas de la figura K.9.b *no* son IP, dado que existe el lazo de 4 celdas que los engloba. Este es un IP, puesto no hay ningún agrupamiento de 8 celdas que puede comprenderlo. Igualmente, en el diagrama (mal simplificado) de la figura K.13.d el lazo de dos celdas que originaría el producto $\mathbf{A.B.C}$ no es un IP, dado que puede formarse un lazo mayor de 4 que lo engloba, como en la figura K.13.b

En consecuencia, el primer paso para minimizar consiste en encontrar **todos** los IP existentes. *Esto es la base para llevar a buen fin una minimización usando el diagrama, siendo la principal causa de los errores que se cometen.*

Una vez determinados todos los IP, luego se podrá decidir cuáles de ellos son necesarios para cubrir todos los "unos" del diagrama, y cuáles son redundantes, amén de poder conocer si existe más de una solución mínima. [1] Asimismo, el concepto de IP apunta al objetivo de determinar y seleccionar aquellos agrupamientos de celdas que permitan obtener simultáneamente una SP con mínimo número de sumandos, constituídos por productos con el menor número de variables, como se pretende.

Determinados en un diagrama *la totalidad* de IP posibles, hay que ver cuáles de ellos, denominados ***IP "no esenciales",*** tienen ***todos*** los "unos" que agrupan enlazados por otros IP, dado que según el caso *pueden aparecer o no* en la SP mínima buscada, los productos correspondientes a ellos.

Así, en las figuras K.13.b y K.13.c (que podían haber sido una sola con todos los lazos posibles) los IP en grisado son "no esenciales". En una PS mínima aparece uno de ellos y el otro no; y *en la otra sucede a la inversa.*

La figura K.16.a muestra todos los IP de una función dada, con tres IP no esenciales en grisado, *de los cuales se necesita sólo uno* en la minimización efectuada en la figura K.16.b Otro caso es el de la figuras K.12.a, K.15.a y K.15.c, donde los lazos en grisado son IP no esenciales que *no* aparecen en el resultado final.

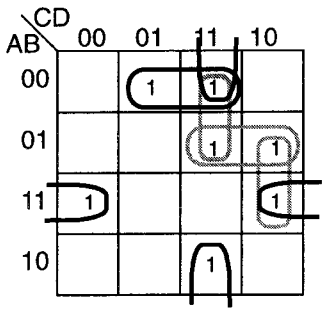


Figura K.16.a

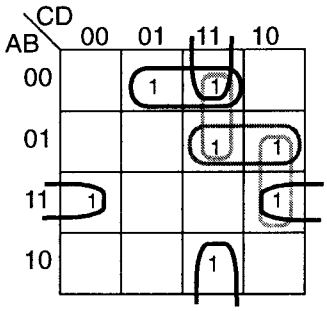


Figura K.16.b

Si un IP no esencial no aparece en una SP mímina es por que resulta ***"redundante"***, dado que los otros lazos cubren todos los unos. De aparecer implicaria obtener una SP equivalente con un sumando innecesario, que no sería mínima como se verificó algebraicamente en el ejemplo 4.

Ahora podemos definir los ***IP "esenciales"***: una vez determinados **todos** los IP de una función, serán aquellos que presentan entre las celdas que agrupan, *una o más no compartidas con otros IP.*

Determinarán productos que necesariamente deben aparecer *en todas* las soluciones mínimas que se obtengan.

En las figuras K.9.b, K.10.b, K.11.c y K.15.b, todos los IP son esenciales. Son esenciales en las figuras K.13.b y.c, los que generan los productos $\mathbf{\overline{B}.\overline{C}}$ y **B.C** ; y en la figura K.16 los que dan lugar a los productos $\mathbf{\overline{A}.\overline{B}.D, A.B.\overline{D}}$ y $\mathbf{\overline{B}.C.D}$

K.5.5 Recapitulación: reglas prácticas para minimizar

Suponiendo que se tiene una función representada por una cierta disposición de "unos" en el diagrama, los pasos a seguir –con vista a obtener soluciones *con el menor número de lazos*, siendo que cada uno de éstos encierra la mayor cantidad posible de celdas adyacentes agrupables– pueden ser:

1. Encontrar *todos* los IP de la función, para lo cual:
   - Procurar primero formar todos los lazos posibles que contengan 8 celdas adyacentes.
   - Con las celdas que no fueron cubiertas en el paso anterior, tratar de formar todos los lazos posibles que contengan 4 celdas adyacentes.
   - Si aún quedaran celdas que se pueden enlazar como 2 adyacentes, formar todos los lazos posibles de este tipo.
   - Luego del paso anterior sólo pueden quedar sin enlazar celdas aisladas, que constituirán lazos con una celda.
2. Preferiblemente indicar en punteado los lazos que tienen todos sus "unos" compartidos con otros lazos, o sea los IP no esenciales.
3. Probar si con los lazos en trazo lleno (IP esenciales) se pueden cubrir todos los "unos" del diagrama, teniendo siempre presente que se busca hacer esto con el menor número posible de lazos
4. Si con los lazos en trazo lleno no se cubren todos los "unos" del diagrama, realizar esto usando el menor número posible de lazos en punteado disponibles.
5. En caso de que los lazos en punteado den lugar a más de una solución mínima, realizar preferentemente un diagrama para cada una.
6. Para cada lazo de cada solución mínima hallar las variables que son sus coordenadas y formar el producto correspondiente, desechando las variables que no intervendrán en el mismo[2]
7. Cada solución mínima, expresarla como suma de todos los productos hallados conforme el paso anterior.

---

[1] Del ejemplo de la figura K.13 resulta la necesidad *primordial* de determinar en un diagrama antes que nada la ***totalidad*** de los lazos que encierran la mayor cantidad de celdas adyacentes agrupables, o sea los IP, para luego decidir cuáles son necesarios al objetivo de cubrir los "unos" con el menor número de ellos, especialmente si hay más de una solución. Si en la figura K.13 no se hubiera dibujado el lazo grisado, el lazo que genera $\mathbf{A.\overline{B}}$ hubiese parecido un IP esencial, y se habría dejado de lado la otra solución mínima. Igualmente podría ocurrir lo mismo en la figura K.13.c de no dibujar el lazo en grisado, el que corresponde a **A.C** parecería esencial, no dando lugar a la otra solución mínima.

[2] Tener presente, que *cualquiera* sea el número de variables de un diagrama, lazos de **8** celdas eliminan **3** variables; lazos de **4** celdas eliminan **2** variables; lazos de **2** celdas eliminan **1** variable, y lazos de **1** celda no eliminan ninguna variable, originando miniterminos.

En general un lazo de $\mathbf{2^n}$ permitirá eliminar **n** variables.

NUMEROS CODIFICADOS EN EL COMPUTADOR: Enteros como binarios con bit de signo Reales como binarios en punto flotante

N1 Representación de números enteros (positivos y negativos) usando dígitos de signo

N1.1 Números decimales con dígito de signo

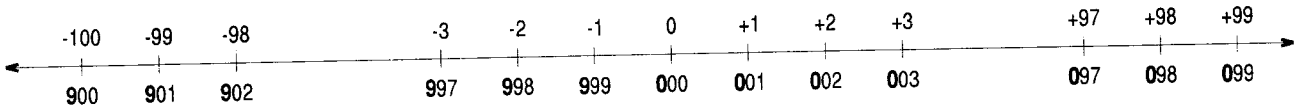
Como se estableció, un número decimal puede tener distintos significados según la convención que se trate.

Nos proponemos representar números decimales negativos y positivos sin emplear los símbolos − y + , usando sólo los símbolos del sistema numérico decimal, y que además la nueva convención adoptada permita sumar los números definidos en ella como si fueran naturales, para aprovechar la circuitería existente para estos números.

Estos requerimientos sirven de base para el planteo de un problema similar en los microprocesadores: dado que en su interior no hay un cable para indicar − ó + ¿cómo se representan por ejemplo los números negativos ?

Volviendo a los decimales, consideraremos el cuenta-vueltas de un pasacasetes con números del 000 al 999 que puede contar progresiva o regresivamente. Se trata, pues, de poder representar números negativos y positivos utilizando las 1000 combinaciones distintas que puede formar el cuenta-vueltas.

Qué ocurre si establecemos que los positivos son los que aparecen a partir del 000 cuando el cuenta-vueltas progresa del **0**01 hasta el **0**99; y que los negativos son las combinaciones que se forman cuando el cuenta-vueltas va hacia atras, de modo de asignar el **9**99 al −1; el **9**98 al −2; el **9**97 al −3, y así sucesivamente hasta retroceder hasta el **9**00 que implicaría 100 pasos hacia atrás, o sea el −100.



El cuenta-vueltas también serviría para efectuar sumas algebraicas. Así −3 + 12 implicaría a partir del 000 ir primero al **9**97 y luego avanzar 12 combinaciones, de modo de llegar al **0**09, que es el resultado de esa suma.

En definitiva hemos reservado el dígito izquierdo como indicador de signo: los positivos tienen todos por lo menos un cero como dígito izquierdo, mientras que los negativos un nueve. Decimos que el signo de los números enteros así definidos está en el dígito extremo izquierdo, en una convención de **números con *"dígito de signo"***.

El costo que hemos tenido que pagar por esta convención es que ahora el mayor número positivo tiene magnitud 99 (en vez de 999 usando los tres dígitos para la magnitud). El **9**00 como se vió corresponde a un número negativo de magnitud 100.

Nos encontramos con una forma distinta de interpretar los números naturales del cuenta-vueltas. Ahora, por ejemplo el 997 como natural representa igual número de unidades, pero interpretado como número con dígito de signo es el −3.

Habiendo verificado en principio que la idea funciona, formalizaremos un poco más la convención establecida, de modo de ir abandonando poco a poco el cuenta-vueltas. Supongamos que queremos representar el número −63 sin tener que ir 63 pasos hacia atrás desde el 000 para ver qué número es entre los que empiezan con 9.

Observando las representaciones de cada positivo y su simétrico negativo respecto del 000 resulta que su suma resulta 1000. Por ejemplo: 001 + 999 = 1000; 003 + 997 = 1000; 099 + 901 = 1000; y así para cada par simétrico.

Por lo tanto, la representación de un número negativo es lo que le falta a su correspondiente numérico positivo (***simétrico*** en la recta de los números) para ser 1000 y viceversa.

Así 997 es lo que le falta a 003 para ser 1000; 997 es lo que le falta a 003 para ser 1000; 901 lo que falta a 099 para ser 1000, etc.

Decimos que 997, que representa al negativo –3, es ***"el complemento a 1000"*** de 003, y viceversa. Entonces, el número que representa a –63 será el complemento a 1000 del positivo 063 que simbolizaremos C_{063}.

O sea debe ser: $063 + C_{063} = 1000$. Por lo que dicho número será: $C_{063} = 1000 - 063 = 937$

De manera inversa, si preguntáramos qué número negativo es el representado por 937, o sea cuál es su *magnitud* [1], la misma sería el complemento a 1000 de 937: $937 + C_{937} = 1000$.

La magnitud será $C_{937} = 1000 - 937 = 063$. Luego 937 representa al número –63.

También podemos escribir **937 ≡ –63** indicando ≡ que 937 "funciona como" –63, siendo la igualdad formal: **937 = –63 + 1000** acorde con que en esta convención la representación de un negativo por el complemento de su positivo, *es dicho negativo* **excedido en el módulo.**

N1.2 Complemento al módulo o "a la base"

Puesto que 1000 es el *número de combinaciones distintas que pueden formarse* en el cuenta-vueltas de 3 dígitos, y técnicamente este número se denomina ***"módulo"*** (**M**), en forma genérica el "complemento a 1000" lo denominamos ***"complemento al módulo"***

Siendo $M = 1000 = 10^3$ o sea la base a la potencia tres, también podría decirse "complemento a la base a la potencia 3", pero ha quedado ***"complemento a la base"*** como **sinónimo** de complemento al módulo.

En este caso que usamos **n** = 3 dígitos, para cualquier número **N** entre 001 y 999 se cumplirá, generalizando expresiones usadas más arriba, que $N + C_N = 1000 = 10^3$

Para combinaciones de **n** dígitos, se tendrá la expresión general:

$$\mathbf{N + C_N = 10^n = 1000\ldots.0 = M}$$

Las máquinas operan con números que sólo pueden tener un número fijo **n** *de dígitos* (bits en binario), o sea con ***"formatos"*** determinados[2] , de donde resulta que *para cada formato de* **n** *dígitos corresponde un valor de módulo igual a la unidad seguida de* **n** *ceros, número que en* ***cualquier*** *base se expresa* $\mathbf{10^n}$

Si operamos en formato de 5 dígitos (módulo $10^5 = 100000$), el número entero –63 se representaría como **99937** = = 100000– 63; mientras que +63 sería **00063**.

COMPLEMENTO AL "MÓDULO MENOS UNO"

Para obtener 937 = 1000 – 063 hay que efectuar una resta "pidiendo prestado".

Esto último puede evitarse si se hace primero 999 – 063 = 936, y al resultado se le suma uno: 936 + 1 = 937.

Siendo 1000 el módulo, es 999 el módulo menos uno; por lo que 936 es el ***complemento al "módulo menos uno"*** del número N=063, que simbolizaremos C'_{063}

De donde resulta que *conviene determinar el complemento al módulo de un número calculando su complemento al módulo menos uno y luego sumarle uno*: $\mathbf{C_N = C'_N + 1}$

---

1 *Magnitud* es sinónimo de *"valor absoluto"* . La magnitud de –80 es el número natural 80, que simboliza el número de unidades que –80 representa.

2 O sea que los números de una máquina sólo pueden constar de una cantidad fija de dígitos, como es de apreciar en una calculadora manual. Se trata de números de "**precisión finita**". Debe tenerse presente que la *precisión* –relacionada con el número de dígitos significativos del resultado de un cálculo, no es lo mismo que la *exactitud* . Así, 3,151569 es mas preciso que 3,14 pero este último es más exacto para expresar Π (pi)

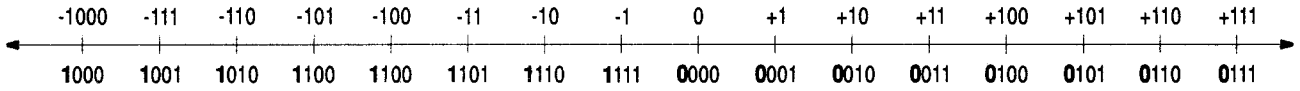
Cuando se opera con precisión finita puede suceder que no se cumplan ciertas propiedades de los números. Así, dado que en el cuenta vueltas ejemplificado no existen números mayores que 999, ni números menores que 000, puede que para ciertos números no se cumpla que sea: (A–B)xC = AxC – BxC o que sea (A–B) + C = A – (B + C), si en el lado derecho de esas igualdades AxC , BxC ó B+C superan 999.

N1.3 Números binarios con bit de signo, correspondientes a "integers" (enteros) en lenguajes de alto nivel

Aplicaremos los conceptos anteriores a los números binarios naturales para representar números binarios enteros.

Podemos imaginar un "cuenta-vueltas" binario, constituído por un tambor que en su superficie tenga escritas, una debajo de otra, las 16 combinaciones binarias de 4 bits[1] que van del 0000 al 1111, el cual puede ir hacia adelante o retroceder. Si está en 0000 y retrocede una combinación se pasa a 1111, del mismo modo que un cuenta-vueltas decimal de 4 dígitos (con 10000 números decimales) pasaría de 0000 a 9999.

A partir del 0000, avanzando desde el 0001 al 0111, estos números naturales con bit extremo izquierdo **0** corresponderán a números positivos; mientras que si el tambor retrocede desde el 0000, se tendrá 1111, 1110, 1101, ... hasta el 1000, números naturales con bit extremo izquierdo **1**, que pasarán a representar números binarios negativos:



Estos números así representados se denominan ***"binarios con bit de signo"*** o ***"binarios signados"***, y se corresponden con los números **binarios enteros** positivos y negativos escritos arriba de la recta de la forma indicada.

Estamos usando los números naturales con su bit extremo izquierdo de valor uno **con dos significados**: así el 1101 como natural simboliza 13_D unidades, y como signado el número binario negativo $-11_B = -3_D$

Dado que estamos usando un bit para el signo, si con 4 bits el número natural mayor que se podía representar es el $1111_B = 15_B$ ahora el mayor positivo es el $0111_B = 7_B$, siendo el más negativo el $1000_B = -1000_B = -8_D$ [2]

Puesto que el número de combinaciones binarias distintas que pueden formarse con 4 bits es $2^4{}_D = 16_D = \mathbf{10000}_B$, dicho número constituirá *el módulo* para el conjunto de combinaciones dadas, representadas antes sobre una recta.

Como en el cuenta-vueltas decimal, también se verifica que la suma de dos combinaciones simétricas cualesquiera respecto del número cero da como resultado el módulo:

$$\begin{array}{r} 1\,1\,1\,1 \\ +\,0\,0\,0\,1 \\ \hline 1\,0\,0\,0\,0 \end{array} \qquad \begin{array}{r} 0\,1\,0\,1 \\ +\,1\,0\,1\,1 \\ \hline 1\,0\,0\,0\,0 \end{array} \qquad \begin{array}{r} 0\,1\,1\,1 \\ +\,1\,0\,0\,1 \\ \hline 1\,0\,0\,0\,0 \end{array} \quad \text{etc...}$$

Por lo tanto la representación de un número negativo es lo que debe sumarse a la representación de su positivo para que el resultado sea el módulo, y viceversa..

Esto es, según esta convención, las representaciones de números negativos y positivos de igual magnitud, son *complementarias* respecto del módulo

Esta relación permite determinar el número con bit de signo correspondiente a un número negativo que se quiere representar, sin necesidad de tener a la vista todas las combinaciones posibles.

þ Por ejemplo, si se quiere conocer para módulo 10000_B como se representa con bit de signo el número -110_B, se parte de su positivo **0**110 y se halla el complemento de este número al módulo:

$C_{0110} = (10000 - 0110)_B = \mathbf{1}010$, combinación que coincide con la representada para -110 en la recta antes dibujada

Obsérvese que en la presente convención para representar un negativo con bit de signo se usa el complemento a la base del correspondiente positivo (bit de signo cero, por lo que **no se trata simplemente de agregar un uno adelante de un número natural.**

Así, -110_B se representa con bit de signo como **1**010 y **no** como **1**110

---

[1] Si bien se ejemplifica con todas las combinaciones que pueden formarse con 4 bits –dado que así pueden tenerse a la vista todos los números que se pueden representar en este formato– el modelo es generalizable para combinaciones de *cualquier* número de bits, del mismo modo que los conceptos/desarrollados para el cuenta-vueltas decimal de 3 dígitos son aplicables a conjuntos de números decimales compuestos por tantos dígitos como se requiera.

[2] Obsérvese al respecto que en esta convención siempre el número negativo de mayor magnitud que puede representarse, supera en uno al positivo de mayor magnitud. Esto se debe a que siendo par el número total de combinaciones, y presentar el número cero el dígito **0** como bit de signo, de hecho es un número positivo más. Esto también se verificó en el cuenta-vueltas decimal, donde los números extremos eran 99 y –100.

Inversamente, si queremos averiguar qué número negativo es el 1001 supuesto con bit de signo, su magnitud la calculamos así: $C_{1001} = (10000 - 1001)_B = 0111$. O sea que es el -111_B

Las dos restas anteriores requieren "pedir prestado". Para evitar esto conviene –como se hizo en base diez en la sección N1.2– calcular primero el complemento al "módulo menos uno" (**C'**), y a éste sumarle uno ($\mathbf{C_N = C'_N + 1}$)

Puesto que operamos con números de 4 bits, el módulo es 10000 (16_D), y el módulo menos uno será 1111 (15_D) [1]
Si queremos calcular C_{1001} hacemos:

$$\begin{array}{r} 1111 \\ -\ 1001 \\ \hline 0110 = C'_{1001} \\ +\quad\ 1 \\ \hline 0111 = C_{1001} \end{array}$$

Observando los números 1001 y $0110 = C'_{1001}$, resulta que esta complementación siempre puede hacerse directamente, **invirtiendo los unos y ceros del número por ceros y unos en su complemento**, sin que sea necesario en ningún caso realizar resta alguna como la efectuada: 1111 – 0110.

De esto resulta la siguiente

Regla práctica para hallar en base dos el complemento al módulo

Para calcular el complemento al módulo[2] de un número binario con bit de signo positivo, se *invierten* los unos y ceros del mismo por ceros y unos; y luego se *suma uno* al número así formado.

þ **EJEMPLOS**

Aplicaremos los conceptos anteriores a la representación de números enteros suponiendo una máquina que opera con números binarios de 8 bits (*"formato 8"*).

Con números de 8 bits el módulo es $2^8_D = 256_D = \mathbf{100000000}_B = 10^{1000}{}_B$; esto es, la unidad seguida de 8 ceros en binario, que implica que pueden formarse 256_D combinaciones binarias.

En este formato, el mayor número positivo que puede representarse es $\mathbf{0}1111111_B = \mathbf{127}_D$, y el negativo de mayor magnitud es $10000000_B = \mathbf{-128}_D$

Problema directo: representar con bit de signo el número $-59_D = -111011$. Para ello seguiremos los siguientes pasos:

1. Representamos 59 como binario natural: $59 = 111011$
2. representamos +59 con bit de signo completando con ceros el formato[3] $+59 = \mathbf{0}0111011$
3. cambiamos los ceros por unos y los unos por ceros (complemento M–1) [4] 11000100
4. sumamos 1 al complemento al módulo menos uno $+\ 1$ → $\mathbf{1}1000101$

La combinación binaria así obtenida representa – con la convención del complemento al módulo o a la base– el número negativo con bit de signo buscado.

No debe perderse de vista, conceptualmente, que *los pasos indicados sirven en esencia para obtener sin "pedir prestado" el resultado de la resta*: $C_{00111011} = 100000000 - 00111011 = 11000101$

Por lo tanto, en formato 8 resultó $-59_D = -111011_B \equiv \mathbf{1}1000101_B$ [5] (ACLARACIÓN CONCEPTUAL)

Es importante realizar en orden los cuatro pasos indicados, so pena de no llegar al resultado correcto.

---

1 *En general, para números de **n** bits, el módulo será la unidad seguida de **n** ceros (10^n), y el módulo menos uno estará constituído por **n** unos*

2 También denominado *"complemento a la base"* o *"complemento a dos"*

3 *El positivo debe tener por lo menos un cero, sino implica que el número en cuestión no se puede representar.*

4 Vale decir se ha efectuado: $\begin{array}{cccccccc} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{array} + 1 = 11000101$
Obsérvese al respecto el cuidado que debe tenerse de partir del número positivo que en el formato dado *tenga correctamente **completados los ceros de la izquierda*** para dicho formato, para no incurrir en errores.

5 Usamos el símbolo ≡ para indicar que el -111011 coincide en representación con el binario natural 11000101, ya que **no se trata de una igualdad**. Puesto que 11000101 = 100000000 – 111011, la igualdad formal es 11000101 = –111011 + 100000000, acorde con lo expresado que el complemento de un número es igual a su negativo excedido en el módulo, siendo que 11000101 ≡ –111011 simboliza una convención, ya señalada al tratar los números decimales con dígito de signo.

Conforme se efectuó, **un número positivo con bit de signo simplemente resulta de agregar a su magnitud** (número natural) **uno o más ceros a la izquierda hasta completar el formato**.

Esto es, **un número positivo debe tener al menos un cero en su extremo izquierdo** (correspondiente a su signo).

Problema inverso: qué número entero es el representado por el 11010110, supuesto con bit de signo[1]

Puesto que el bit de signo es **1**, es un entero negativo, cuya magnitud hay que determinar, hallando su simétrico positivo, que será el complemento al módulo de 11010110. Para hallarlo, **invertimos este número y sumamos uno**[2] :

La magnitud o valor absoluto de 11010110 será su complemento (simétrico positivo):

$C_{11010110} = 00101001 + 1 = 00101010$; por lo tanto $\mathbf{1}1010110 \equiv -00101010 = -42_D$

Se puede llegar al mismo resultado si al bit de signo se le da el valor –128, y a los siguientes 64, 32, 16, 8, 4, 2, 1

Así resulta: $\mathbf{1}1010110 = [1x(-128) + 1x64 + 0x\,32 + 1x16 + 0x8 + 1x4 + 1x2 + 0x1]_D = (-128 + 86)_D = -42_D$

Propagación de signo

Otra importante conclusión puede extraerse del siguiente ejercicio: representar –59 con 16 bits

En formato 16 será 59 = 0000000000111011, y su complemento a módulo será

$1111111111000100 + 1 = \mathbf{1}111111\underline{11000101} = -111011 = -59$

Comparando con la misma representación de –59 realizada más arriba en formato 8, resulta que ahora se tiene el mismo número con ocho unos mas a la izquierda, de donde se deduce que:

> *Cuando se tiene un número con bit de signo **negativo**, si se agregan **unos** a la izquierda del mismo, el número no cambia; de manera análoga que si a un número **positivo** se le escriben ceros a la izquierda.*

Por lo tanto, cuando se debe pasar un número con bit de signo negativo (positivo) a un formato mayor, se le deben agregar unos (ceros) hasta completar el nuevo formato, acción que se conoce como ***"propagación del signo"***

Valores extremos representables con bit de signo en un formato dado

Determinaremos en base diez los valores límites que existen para números de 4 bits con bit de signo, y encontraremos una expresión generalizable para cualquier número de bits.

Para **n** = 4, el más positivo es $+N_{max} = 0111_B = 7_D = (8-1)_D = (2^3-1)_D = (2^{4-1}-1)_D = (\mathbf{2^{n-1}-1})_D$;

y el más negativo es $-N_{max} = 1000_B = -8_D = (-2^{4-1})_D = (\mathbf{-2^{n-1}})_D$

Por lo tanto, para **n** bits el rango de valores representables con bit de signo, en base diez va de $\mathbf{-2^{n-1}}$ a $\mathbf{2^{n-1}-1}$

Para **n** = 8 bits resulta: $+N_{max} = 2^{8-1}-1 = 2^7-1 = \mathbf{127}$ y $-N_{max} = -2^{8-1} = \mathbf{-128}$

*Por lo general cuando en un lenguaje de alto nivel se definen "**integers**", sin más aditamentos, el compilador o ;una subrutina los traduce a binarios con bit de signo en formato de **16** bits.*

Aplicando las expresiones anteriores, resulta un rango entre **–32768** y **+ 32767**, siendo que los números correspondientes son: $\mathbf{1}000000000000000_B$ (15 ceros) y $\mathbf{0}111111111111111_B$ (15 unos)

El rango hallado también corresponde a cualquier procesador con palabra de 16 bits, como el 80286.

Resulta relevante compararlo con el del 80386 ó el 486, de **32 bits:**

$-N_{max} = [10000...(31\ ceros)..0]_B = (-231)_D \approx (-2x10^9)_D$ y $+N_{max} = [10000...(31\ ceros)..0]_B = (2^{31}-1)_D \approx 2x10^9{}_D$

También es útil notar que 31 bits dan lugar a 9 dígitos decimales, o sea 31/9 ≈ **3,5 bits por cada dígito decimal**, relación que también puede demostrarse matemáticamente.

Se debe tener presente que si se usan números binarios como **magnitudes** (números naturales), partiendo de que para n=4 bits el rango representable va de **0000** a **1111** = $2^4-1 = 15$, si se generaliza para cualquier formato **n**, el rango irá de **cero** a $\mathbf{2^n-1}$. *Para igual **n** se duplica la magnitud máxima respecto de los números con bit de signo*

---

1 Como natural sería el 214, según puede verificarse

2 Así evitamos hacer la resta: 100000000 – 11010110 = 00101010 = $C_{11010110}$

Recapitulación de la representación de enteros como binarios con bit de signo:

- Los números positivos son los naturales con *bit de signo* **0** como *bit extremo izquierdo* en un formato dado (pudiendo existir más ceros a la izquierda de la magnitud, hasta completar dicho formato)
- Un negativo con bit de signo es *el complemento al módulo* (C_N) de su positivo N (bit de signo 0), debiendo presentar por lo tanto un *bit de signo* **1** como *bit extremo izquierdo* en su representación en un formato dado. (Puede verse el complemento C_N como el negativo –N excedido en el módulo: $C_N = -N + M$)
- Los dos párrafos anteriores pueden resumirse así: **los números binarios con bit de signo positivo son los naturales con ceros a la izquierda, y los negativos son los que resultan de complementar a los primeros**
- Un número negativo (positivo) *no cambia* si se le agregan unos (ceros) a la izquierda de su bit de signo, pasando a ser el bit extremo izquierdo el bit de signo *("propagación de signo")*
- La *magnitud* de un número *positivo* (bit de signo **0**) es *directamente* la del número natural que está a la derecha de su bit de signo. En cambio la *magnitud* de un *negativo* (bit de signo **1**) se halla haciendo el *complemento al módulo* de su representación, para lo cual se deben invertir todos sus bits (incluído el bit de signo) y sumar uno

N.2 Suma de binarios enteros representados con bit de signo

Al definir la convención usada para representar números binarios con bit de signo, se vió que uno de sus objetivos era poder *sumarlos como binarios naturales*, así los circuitos desarrollados para éstos sirvan también para los primeros

Dado que los números con bit de signo positivo son naturales con uno o más ceros a la izquierda, sumar dos positivos será como sumar naturales. Por ejemplo: **0**0001101 + **0**1010001 = **0**1011110

En la convención de bit de signo definida, un entero negativo –N es el complemento C_N de su positivo N.

Siendo $N + C_N = M$, **puede verse a C_N como el negativo –N excedido en el módulo: $C_N = -N + M$**

Suma de numeros de signo opuesto con resultado positivo

þ Realizar con bit de signo (A) + (–B) = $(43_D) + (-8_D)$ = 101011 + (–1000) en formato 8

43_D = 101011 = **0**0101011 $\quad -8_D = -1000 = -00001000 \equiv 11110111 + 1 = \mathbf{1}1111000$

$43_D \rightarrow$ **0**0101011
$-8_D \rightarrow$ + 11111000
1 **0**0100011 → **0**0100011 = 100011 = 35_D

↑ se descarta este uno fuera de formato correspondiente al exceso de 100000000 en el número negativo [1]

Se verifica en base diez que $(43_D) + (-8_D) = 35_D$, y en binarios enteros que 101011 + (–1000) = 10011

Ahora analizaremos otra *suma de números de signo opuesto, pero con resultado negativo.*

þ Efectuar con números con bit de signo (–A) + (B) = $(-43_D) + (8_D)$ = (–101011) + (1000) en formato 8

$-43_D = -101011 = -00101011 \equiv 11010100 + 1 = \mathbf{1}1010101$; $\quad 8_D = 1000 = \mathbf{0}0001000$

$-43_D \rightarrow$ **1**1010101
$8_D \rightarrow$ + 00001000
11011101 ≡ –(00100010+1) = **–100011** = -35_D

Se verifica en base diez que $(-43_D) + (8_D) = -35_D$ y en binarios enteros que (–101011) + (1000) = –10011

Esta vez el resultado es negativo (bit de signo **1**), por lo que su magnitud se determinó invirtiendo sus bits y sumando uno. [2]

---

1 A continuación se justificará *por qué se descarta el uno que cae fuera de formato en una suma de números con bit de signo.*
$-8 = -1000 \equiv 11111000 = C_{00001000} = -00001000 + 100000000$
Entonces al hacer la suma anterior **0**0101011 + **1**1111000 también se está haciendo: **0**0101011 + (–00001000 + 100000000) =
= (**101011**) + (–**1000**) + 100000000 Esto es, la suma de enteros requerida (en negrita) más un exceso igual al módulo.
A su vez, el resultado de la suma realizada puede descomponerse así: **1**00100011 = 100000000 + **0**0100011
Por lo tanto este resultado es la suma requerida excedida en 100000000. Este exceso se descarta simplemente no considerando el uno fuera de formato, por lo que que los bits que están a la derecha del mismo (**0**0100011) son el resultado buscado.

2 Justificaremos también para este caso, en el cual no se da un uno fuera de formato, que es correcto sumar numeros signados como si fueran naturales.

Suma de dos negativos:

þ Efectuar en formato 8 con bit de signo (–A) + (–B) = (-43_D) + (-8_D) Aprovechando las representaciones anteriores se tiene:

$-43_D \longrightarrow$ 11010101
$8_D \longrightarrow$ +11111000
1 11001101
↑ este uno no forma parte del resultado 11001101 ≡ –(00110010 + 1) = **–110011** = -51_D

Se verifica que (-43_D) + (-8_D) = -51_D y en binario (–101011) + (–1000) = – 110011

Nuevamente, por tener el resultado bit de signo negativo, para determinar el resultado como un número entero negativo (**–110011**), hubo que hallar el complemento al módulo del primero.[2]

Generalizando: **si en una suma de números considerados con bit de signo aparece un uno fuera de formato, el mismo no forma parte del resultado**, por tratarse de un exceso respecto de la suma que se quiere realizar

N.3 Indicadores de estado SZVC ("flags")

En los microprocesadores cada vez que se realiza una operación aritmética o lógica, el sumador que constituye la Unidad Aritmético-Lógica genera, entre otros, **4 bits** indicadores *("flags")* principales, relacionados con el resultado obtenido, que se conocen por sus iniciales inglesas **S,Z,V,C** que forman parte del denominado **Registro de Estado**.

Ellos pueden indicar, por ejemplo, si un resultado alcanzado fue o no cero, si fue positivo o negativo, si entró o no en el formato de operación y otras características a tratar. Como se ejemplificó en la primer unidad, una instrucción de salto estipulada según el valor de uno o más de dichos indicadores, ordena al microprocesador seguir ejecutando una u otra de dos secuencias de instrucciones programadas.

Como se remarcó y se deduce de las sumas realizadas, ***la UAL de un computador sólo opera aritméticamente con números binarios naturales***. **Los números con bit de signo** (al igual que los números en BCD) **son sumados como naturales**,[3] siendo los negativos números naturales interpretados de otra forma.
Son **los programas** los que interpretan los resultados de una u otra forma.

Por ejemplo, en la última suma realizada: 11010101 + 11111000 = 111001101, interpretada como una suma de números con bit de signo, corresponde a la suma de enteros (–43) + (–8) = –51, según se vió. Pero también puede interpretarse que se originó en la suma de naturales 213 + 248 = 461, como puede verificarse.

Un programa desarrollado para enteros considerará como resultado 11001101, y en caso de tener que convertir este número a decimal (para mostrarlo en pantalla o imprimirlo) al detectar que el número empieza con el bit **1**, generará el código ASCII del signo menos. Luego calculará la magnitud binaria del mismo hallando su complemento al módulo – como se ejemplificó en el segundo y tercer caso de la suma de números con bit de signo de la sección N.2– , y determinará qué número decimal es. Conocido éste generará el código ASCII de cada uno de los dígitos que lo componen, de modo que puedan aparecer en pantalla o papel, formando el número –51.

Si el programa es para naturales (magnitudes) tomará como resultado 111001101, o sea que considerará formando parte del mismo el valor del indicador **C**, que en este caso vale 1. De tener que imprimirse o visualizarse por pantalla dicho resultado, determinará que se trata del número 461 expresado en código ASCII, sin analizar el bit de signo.

(cont) Como en el ejemplo anterior, siendo $C_{00101011}$ = **1**1010101 = –00101011+ 100000000 La suma **1**1010101 + **0**0001000 es equivalente a realizar: –00101011+ 100000000 + **0**0001000 = (**–00101011**) + (**00001000**) + 100000000 o sea la suma pedida excedida en el valor del módulo.

El resultado de la suma efectuada, por ser un número con bit de signo negativo es el complemento de su correspondiente positivo, por lo que también puede expresarse así: 11011101 = $C_{00100011}$ = –00**100011** + 100000000 O sea que este exceso se pone de manifiesto cuando determinamos la magnitud del número con bit de signo negativo. Entonces, el exceso no se manifestó como un uno fuera del formato, sino que pasó a formar parte del resultado en la convención del complemento de positivos para representar negativos. (Este párrafo es la continuación del pie de página anterior)

2 Siendo: $C_{00101011}$ = **1**1010101 = –00101011 + 100000000 y $C_{00001000}$ = **1**1111000 = –00001000 + 100000000

en este caso hay un doble exceso del módulo, por sumarse dos números negativos representados por su complemento. Se combinan los dos casos anteriores.

En consecuencia, uno de dichos excesos será el uno que está fuera de formato, y el otro está formando parte del resultado con bit de signo negativo.

3 Lo anterior implica que las instrucciones para sumar y restar números naturales o enteros *son las mismas* Para el caso de definirse datos "reales", existen instrucciones para operar aritméticamente en punto flotante, siendo el coprocesador matemático el encargado de realizarlas, y no la UAL.

También *ambos programas diferirán en las instrucciones de salto involucradas, pues las hay para enteros y para naturales.*

No debe perderse nunca de vista que el tipo de datos definidos en Pascal, C, Basic, etc (integers , magnitudes, reales, etc) determina las instrucciones que el programa compilador deja traducidas en código de máquina.

La UAL en esencia opera con naturales. Pero como los binarios con bit de signo (provenientes de enteros) en la UAL se suman como naturales, y a fin de que las instrucciones de salto de los programas para números enteros puedan emplearse, de los cuatro indicadores citados **SZVC** –que genera la UAL luego de cada operación aritmética– **SVZ** se usan para enteros, y **CZ** para naturales, como se verá. En cada subconjunto los flags pueden usarse solos o combinados

Esto es, la UAL "no sabe" si el programa en ejecución es para enteros o naturales, y tanto la suma o la resta para ambos tipos de números *las realiza de igual forma.* Luego de cada operación que efectúa, la UAL genera **SZVC**, y son las instrucciones de salto de los programas que se ejecutan las que preguntan por el valor (0 ó 1), de **SVZ** y **CZ** según sea para enteros o naturales. Ejemplos de utilización aparecen en las secuencias de instrucciones de la Unidad 3.

Definiremos a continuación estos indicadores para el resultado de una suma realizada en un formato de **n** bits:

Indicador* S *de Signo:

Será **S=1** *(signo negativo)* si el resultado presenta *el bit de la extrema izquierda del formato* con valor **1**. Caso contrario será **S=0** *(signo positivo).*

Sólo interesa para operaciones con números enteros, representados como números con bit de signo, o sea signados[1]
El formato puede ser de 8, 16, 32, 64 bits.

.Indicador* Z *de resultado cero:

Solamente será **Z=1** si el resultado es cero *("zero")*, o sea en el caso particular que sean ceros *todos* los **n** bits del mismo en el formato dado. Si como ocurre en la mayoría de los casos, el resultado **no** es cero, será **Z=0**

Vale decir, **Z=1** significa *"si, es cero"* o es "cierto que es cero"; y **Z=0** *"no es cero"* o "es falso que es cero"

Dado que el número cero se representa igual para naturales o signados, el indicador **Z** puede usarse indistintamente para detectar resultado cero en ambos tipos de números.

Indicador* C *de acarreo:

Será **C=1** si en el resultado de una suma aparece un uno fuera de formato, o sea si existe acarreo ***("Carry")*** hacia la posición **n+1**. De no ser así será **C=0** (**C=0** es *" carry no "* y **C=1** es *" carry si"*)

En una suma de naturales (apéndice unidad 1) si en el resultado hay un uno fuera de un formato **n** (posición n+1), el mismo forma parte del mismo, o sea que si **C**=1 el programa debe agregar un uno a los otros **n** bits del resultado.

Dado que la UAL también suma (el complemento del sustraendo) para realizar una resta, ella **invierte** el valor de **C** obtenido de esa suma. En una resta interesa si "pido prestado" *("borrow")* o no, suponiendo que la resta siga en la posición n+1. Si al sumar para restar es **C**=1, no "pido prestado" y la UAL indicará **C**=0; y si es **C**=0 indicará **C**=1.

Indicador* V *de "overflow" ("desborde"):

Será **V=1** si el resultado de una suma entre números **supuestos con bit de signo** *excede* el mayor valor positivo o negativo que se puede representar en el formato dado. Caso contrario será **V=0** *("no overflow")*

Según se calculó, para un formato de n=8 bits los valores extremos que se pueden representar son $+127_D$ y -128_D.

Para n = 16 los mismos serán $2^{n-1}-1 = 32767_D$ y $-2^{n-1} = -32768_D$.

De emplearse n = 32 bits (como operan el 386 y el 486) esos extremos superan los dos mil millones expresados en decimal[2]

Por consiguiente será **V=1** si en un formato de operación elegido, el resultado positivo o negativo de la suma de dos números (que pueden representarse en ese formato) supera el máximo positivo o negativo representable en el mismo

No es necesario conocer el valor de un resultado y compararlo con los valores extremos para determinar si hay o no overflow. Una forma práctica de determinar esto manualmente consiste en observar simplemente los bits de signo de los sumandos y del resultado.

Será **V=0** si sumando dos positivos el resultado es positivo; o si sumando dos negativos el resultado es negativo
Será **V=1** si sumando dos positivos el resultado es negativo; o si sumando dos negativos el resultado es positivo

---

1 En algunas máquinas se designa **N**, inicial de negativo: **N=1** indica que es cierto que el resultado es negativo

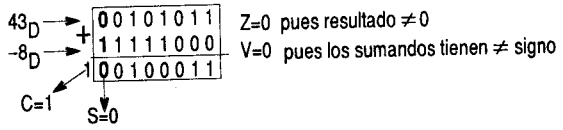
2 Una forma rápida de estimar esto es como sigue: $2^{n-1} = 2^{31} = 2^{10} \times 2^{10} \times 2^{10} \times 2^{1} \approx 1000 \times 1000 \times 1000 \times 2 = 2000.000.000$; siendo $2^{10} = 1024 \approx 1000$

Esto es, lo correcto es que sumando positivos el resultado sea positivo, y que al sumar negativos sea también negativo el resultado. De no ser así, se detecta overflow.

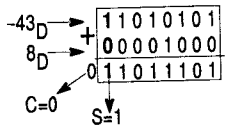
Si se suman dos números de signo contrario jamás puede haber overflow, pues en definitiva se trata de una resta, por lo que su resultado no puede superar en magnitud al mayor de los números restados.

þ **EJEMPLOS SIN OVERFLOW**

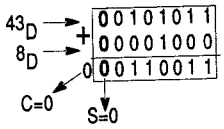
Para las sumas realizadas en la sección anterior, a las que se agrega $(A) + (B) = (43_D) + (8_D)$ se indican los valores de **SZVC**



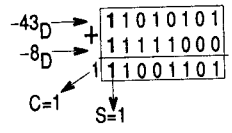
Z=0 pues resultado ≠ 0
V=0 pues los sumandos tienen ≠ signo



Z=0 pues resultado ≠ 0
V=0 pues los sumandos tienen ≠ signo



Z=0 pues resultado ≠ 0
V=0 pues los sumandos tienen = signo que el resultado

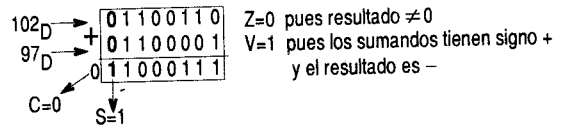


Z=0 pues resultado ≠ 0
V=0 pues los sumandos tienen = signo que el resultado

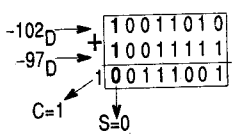
þ **EJEMPLOS CON OVERFLOW:** efectuar $102_D + 97_D$ y $(-102_D) + (-97_D)$ en formato 8

$102_D = 1100110 = \mathbf{0}1100110$ $\qquad -102_D = -01100110 \equiv (10011001 + 1) = \mathbf{1}0011010$

$97_D = 1100001 = \mathbf{0}1100001$ $\qquad -97_D = -01100001 \equiv (10011110 + 1) = \mathbf{1}0011111$



Z=0 pues resultado ≠ 0
V=1 pues los sumandos tienen signo + y el resultado es –



Z=0 pues resultado ≠ 0
V=1 pues los sumandos tienen signo – y el resultado es +

En principio se verifica que ambos resultados interpretados con bit de signo están errados:

11000111 ≡ –(00111000 + 1) = –00111001 = –57 , debiendo ser 102 + 97 = 199 [1]

00111001 = 111001 = 57, debiendo ser –102 + (–97) = –199

A fin de evitar equívocos comunes, debe subrayarse que el overflow se refiere siempre al desborde en la suma o resta de **números con bit de signo**, y que el mismo ***no tiene relación directa con el bit C de "carry"***

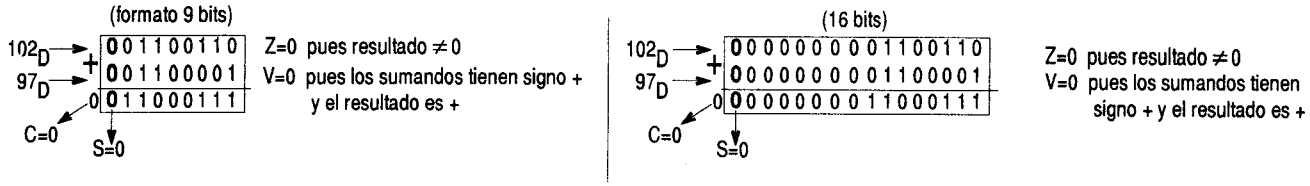
Así, en las sumas anteriores 43 + (–8) y –43 + (–8) es **C=1** pero no hay overflow (**V=0**), mientras que en la suma de enteros 102 + 97 no hay carry (**C=0**), pero hay overflow (**V=1**)

Existe una tendencia natural a asociar al overflow en el sentido de "desborde" cuando en un cierto formato luego de una suma aparece un uno fuera del mismo (**C=1**), tal como ocurrió en las dos primeras sumas citadas en primer término. Si bien es cierto que para los naturales dicho uno fuera de formato puede asimilarse a un desborde, de lo que se trata es del desborde en la suma de números binarios con bit de signo, que representan a números enteros, como se definió más arriba.

Verificaremos que la existencia de overflow en un formato dado, implica *truncamiento* por pérdida del bit más significativo del resultado, y que el problema puede solucionarse operando en un formato que tenga por lo menos un bit más que el formato en el cual se produjo el overflow. Para ello volveremos a realizar 102 + 97, primero en formato 9 y luego en formato 16, duplicando el formato original, como se realizar en la práctica. En ambos casos para pasar a un formato mayor realizaremos una propagación de signo, o sea en este caso con ceros a la izquierda.

---

[1] Suponiendo que los números fueran naturales, el resultado 11000111 = 199 sería correcto, pero como supusimos una suma entre signados, esta disquisición no tiene sentido. Tampoco es válido interpretar a los dos sumandos como signados, y al resultado como natural, ya que en un sistemas de números de la misma clase, una operación entre dos de ellos debe dar como resultado otro de dicha clase.



Se observa que ya en formato 9 deja de existir overflow. Comparando con la misma cuenta efectuada en formato 8, resulta que los 8 bits del resultado en este formato son también los que acompañan al bit de signo **0** en formato 9. O sea que en formato 8 el resultado había quedado *trunco*, por la pérdida de su bit de signo.

Verificación: **0**11000111 = **0**0000000011000111 = +11000111 = +199, como debe ser.

En ambos formatos la ausencia de overflow está de acuerdo con el hecho de que en formato 9 el mayor positivo es 255 > 199 y en formato 16 ese número es 32767 > 199

N.4 Resta de binarios enteros representados con bit de signo

Como los naturales, **los números con bit de signo se restan sumando al minuendo el complemento al módulo del sustraendo:** $\mathbf{A - B \equiv A + C_B}$

Esto es, por ser $B + C_B = M$ resulta que $A - B = A + (-B) = A + C_B - M$; por lo que si para restar A – B sumamos $A + C_B$ al resultado de la suma hay que restarle el módulo M, o sea que *está excedido en M.*

þ **EJEMPLOS**

Primero restaremos, como binarios con bit de signo en formato 8, los enteros en base diez: A – B = (–49) – (– 95)

$A = -49_D = -00110001 \equiv (11001110 + 1) = \mathbf{1}1001111$

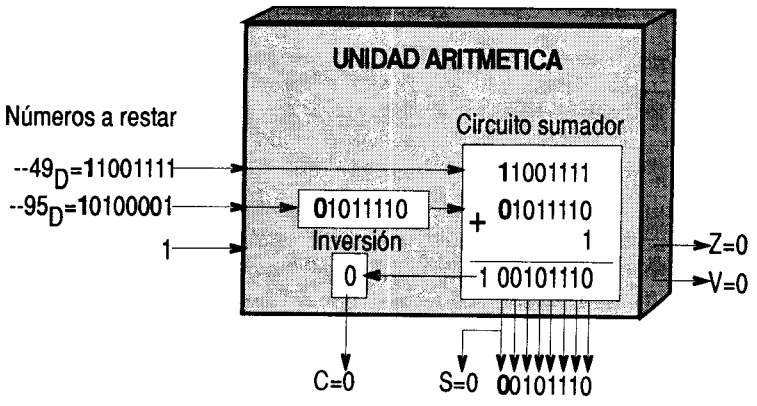
$B = -95_D = -01011111 \equiv (10100000 + 1) = \mathbf{1}0100001$ por lo que será $C_B = 01011110 + 1$

Como indica la figura siguiente, a la Unidad Aritmética de un computador entran los números a restar –representados con bit de signo– junto con el número **uno**. Este uno se sumará a la inversa de **B** para conformar $\mathbf{C_B}$, a fin de realizar $\mathbf{A + C_B \equiv A - B}$

Obsérvese que no interesa si en una etapa anterior tuvo lugar una complementación para representar **B** como negativo con bit de signo. *Tanto en la resta de binarios naturales, como de binarios con bit de signo, la Unidad Aritmética "no sabe que pasó antes"; siempre toma al número **B** para luego invertirlo y sumarle uno. Así suma el C_B al minuendo **A***

*Asimismo, un computador **no** convierte la orden de resta* (–49) – (– 95) en –49 + 95 haciendo "menos por menos igual a mas"

La Unidad Aritmética también generará los valores de los indicadores **SZVC**, siendo que *en una suma para restar **invierte** el valor interno del bit que sale fuera del formato correspondiente al valor que tendrá* **C**, como se anticipó al definir éste, y según se indica en el esquema siguiente.



Verificación de la operación de la izquierda:

$\mathbf{0}0101110 = +46_D = (-49)_D - (-95)_D$

Ex-profeso se han elegido los números enteros -49_D y -95_D, los cuales representados como binarios signados se codifican con *las mismas* combinaciones binarias (11001111 y 10100001) que los números naturales 207_D y 161_D del primer ejemplo dado en la resta de naturales (Apéndice numérico de la Unidad 1)

Esta coincidencia se ha hecho para remarcar que la U Aritmética no está preparada para distingui números con bit de signo, sino que opera toda las combinaciones binarias que recibe *como nme ros naturales*, siendo el resultado obtenido idnti co al del ejemplo citado con naturales.

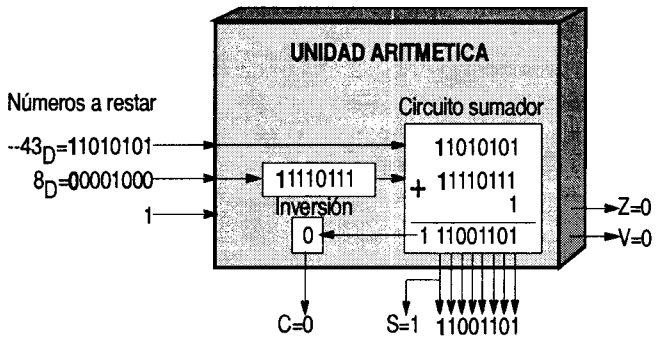
Vale decir, si suponemos que queremos restar $(207–161)_D$ supuestos naturales, las combinaciones binarias que entran y alen de la Unidad Aritmética serían **las mismas**, *incluídos los valores de* **SZVC**, dado que estos indicadores se generan automáti-carente, en prevención de que se opere con naturales o signados.

En esencia, lo único que interesa son las combinaciones binarias que entran, sin importar si se interpretan como correspondientes a números naturales o signados.

Entonces suponiendo que estemos operando con signados, además de los indicadores **SVZ** que interesan para éstos, también se generará el valor de **C**, usado para naturales; y si operamos con éstos, además de **CZ** que interesan para ellos, la U. Aritmética generará **S** y **V**, pues ella *"no sabe"* cómo se están interpretando las combinaciones binarias que recibe, ni las que genera.

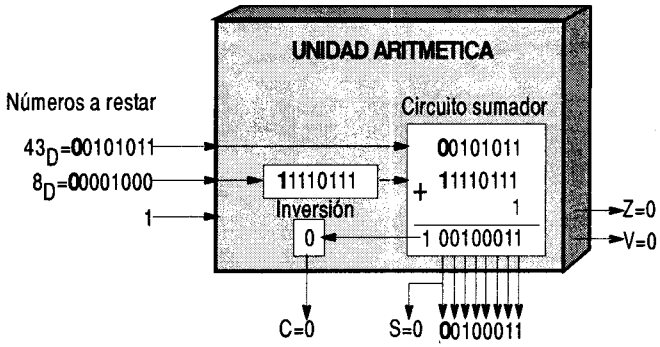
Es importante aclarar que si bien el esquema presentado para la U. Aritmética se ejemplifica para restas, de usarse para sumas no producirá la inversión del segundo número (sustraendo en la resta), como tampoco la del indicador **C**, y el tercer número que entra debe ser **0** en vez de **1** (usado para formar el complemento en la resta).

A continuación, con fines didácticos ejemplificaremos restas con números enteros utilizados en las sumas algebraicas del final de la sección N.3. Primero se efectuará $A - B = (-43) - (8)$; y luego $A - B = (+43) - (+8)$



$11001101 = -(00110010 + 1) = -51_D$

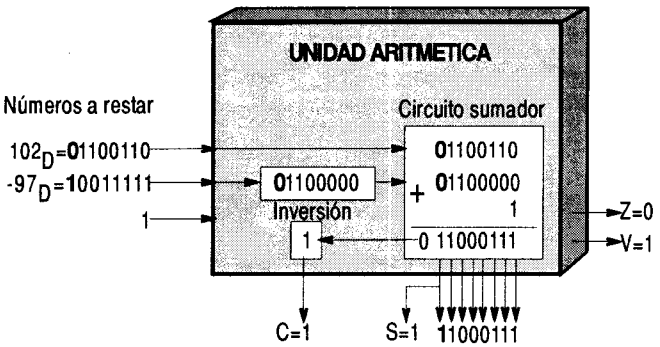
Se verifica, dado que $(-43) - (8) = -51$



$00100011 = 100011 = 35_D$

Se verifica, dado que $(+43) - (+8) = 35$

El último ejemplo trata el ***overflow en restas de binarios signados*** para el caso que se quiere efectuar en formato 8 la resta de enteros $A - B = (102_D) - (-97_D)$ [1]



Por existir overflow el resultado presenta error por truncamiento, sólo rectificable por ampliación del formato

[1] En general las restas del tipo $A - (-B)$ pueden dar overflow, dado que en definitiva son equivalentes a la suma $A + B$ (aunque un computador **no** hace "menos por menos igual a mas"). Lo mismo ocurre con las del tipo $-A - (+B)$, equivalentes a la suma $-A + (-B)$. En ambos casos resultan sumas de dos números del mismo signo.

N.5 Los indicadores SZVC en la comparación de números

La comparación entre dos números se lleva a cabo en un computador mediante una instrucción de comparar, que ordena **restar** el primero menos el segundo en la **UAL**. Los valores de **SZVC** generados en esa resta bastan para determinar si el minuendo es mayor, igual, o menor que el sustraendo.

Como se discutirá, **no hace falta conocer el valor concreto del resultado.**

Así, es inmediato que si dos números naturales o con bit de signo son iguales, su resta dará cero, por lo que será **Z**=1, de acuerdo como este indicador se ha definido. Entonces puede conocerse si dos números en cualquier formato son iguales (**Z**=1) o distintos (**Z**=0) luego de restarlos.

Luego de una instrucción de comparación (resta), **otra** instrucción (de salto) ordenará, en función del valor de **Z**, que se siga ejecutando una secuencia o se pase a otra.

Discutiremos cómo determinar si un número es mayor o menor que otro, primero para los enteros (números con bit de signo) y luego para los naturales (magnitudes sin signo)

Respecto de los **números con bit de signo**, se debe partir de la base que si el resultado de una resta es positivo (**S**=0), el minuendo es mayor que el sustraendo; siendo que un número positivo es siempre mayor que un negativo, sin importar su magnitud relativa. En relación con el signo, se debe tener presente, que si el resultado de una resta arrojó overflow (**V**=1), implica que el signo del resultado está *invertido* respecto del correcto.

Por lo tanto, podemos establecer luego de efectuar una resta A–B entre números supuestos signados:

Si no hay overflow (**V**=0) y es	**S**=0 resulta A ≥ B
	S=1 resulta A < B
Si hay overflow (**V**=1) y es	**S**=1 resulta A ≥ B pues en realidad es **S**=0
	S=0 resulta A < B pues en realidad es **S**=1

Entonces, para comparar dos números signados hay que considerar **S** y **V** [2]

Se ha indicado A ≥ B, pues en el caso que sean iguales, el resultado cero hace que también sea **S** = 0

Las siguientes consideraciones valen para los números no signados (magnitudes).

En la suma efectuada en el apéndice numérico de la Unidad 1 para restar dos binarios naturales A – B resultó un **1** fuera de formato, o sea **C**=1, debiendo ser A ≥ B por tratarse de números naturales

. Se deduce que *si al efectuar una resta de naturales* (para lo cual la UAL hace A + C_B), *el minuendo es mayor o igual que el sustraendo será* ***C=1*** (apareciendo como **C**=0 en el registro de estado, como indicación de borrow).

Esto puede justificarse como sigue:

La resta de naturales a realizar es R = A – B = A + (–B) ≡ A + C_B

Al efectuar en el ejemplo citado (formato 8) la suma A + C_B se está efectuando:

A + (–B) + 100000000 = (A – B) + 100000000 = R + 100000000,

o sea que el resultado de la suma contiene la resta R pedida.

Si A ≥ B será A + C_B = R + 100000000 ≥ 100000000 por ser R ≥ 0 Puesto que un resultado ≥ 100000000 debe tener necesariamente un **1** fuera de formato (correspondiente al bit **C**), se confirma la regla anterior.

En caso de ser A < B sera R < 0, con lo cual R + 100000000 < 100000000, por lo que no aparecerá un **1** fuera de formato por tratarse de un resultado menor que 100000000.

O sea que si al efectuar una resta de naturales mediante una suma, resulta ***C=0*** *(que aparece como* ***C=1*** *en el registro de estado), debe ser* ***A < B***

Este tipo de datos es de **longitud variable**, de forma de poder elegir la longitud de los datos apropiada a la precisión requerida. Por ejemplo puede ser de 1 a 16 bytes.

2 Para el lector familiarzado con el álgebra de Boole (tratada en el apéndice final),la discusión anterior se sintetiza así: A ≥ B si S ⊕ V = 0 y A< B si S ⊕ V = 1 Será A > B sólo si S ⊕ V = 0 y Z = 0, o sea si (S ⊕ V + Z) = 0

N.6 Números binarios fraccionarios

En el sistema decimal, un número fraccionario se puede expresar como un cociente o mediante una coma. Así: 1/4 = 0,25; 4/3 = 1,33..., etc. En base dos o en otra, también podemos representar, con una simbología semejante, un número que sea menor que la unidad, o que presente una parte entera y otra que es una fracción de la unidad.

Volviendo a la balanza decimal del apéndice de la Unidad 1, con 9 pesas de 1, 10, 100, ... gramos, si se necesita apreciar fracciones de un gramo, podemos suponer que existen 9 pesas de una décima de gramo (1/10 = 0,1), 9 pesas de un centésimo de gramos (1/100 = 0,01), etc... Cada 9 pesas que son fracción de gramo son diez veces menores que las 9 existentes de la medida superior siguiente.

Cuando escribimos que un objeto pesa 328,205 la coma indica que el 8 que está a su izquierda son unidades, y que el 2 de su derecha son decimas de gramo.

Esta misma convención se usa en base dos, pero del mismo modo que a partir de la pesa de 1 gramo hacia arriba, cada pesa es el doble que la medida anterior, las pesas menores que 1 gramo van en una progresión en que cada pesa es la *mitad* de la medida mayor anterior, comenzando por la medida de 1/2 gramo, existiendo tanto para las mayores o menores de 1 gramo, **una** pesa de cada tipo. Expresadas en decimal las pesas binarias tendrían los gramajes:

... 32g, 16g, 8g, 4g, 2g, **1**g, 1/2g, 1/4g, 1/8g, 1/16g, ...

pudiendo existir infinitas medidas hacia un lado u otro. Expresadas en binario serían:

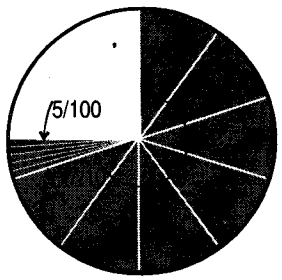
... 100000g, 10000g, 1000g, 100g, 10g, **1**g, 1/10g, 1/100g, 1/1000g, 1/10000g ...

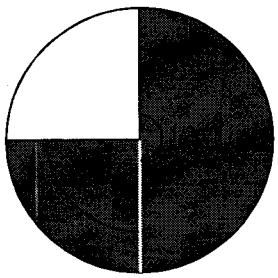
Realizando las divisiones binarias indicadas conforme se vió en el apéndice numérico de la Unidad 1, y usando coma para separar la parte entera de la fraccionaria, también resulta en binario:

1/10 = 0,1 1/100 = 0,01 etc...

Suponiendo un objeto que pese en decimal 0,75 g. en la balanza con pesas binarias se formaría con una de 1/2g y otra de 1/4g

expresado en binario sería: 1/10g + 1/100g = 0,1g + 0,01g = 0,11 g





Para cotejar la estructura fraccionaria en bases dos y diez. en las figuras siguientes se ha dibujado la misma fracción de un círculo. Dicha fracción en base diez se forma con 7 agrupamientos de 1/10 de círculo, más 5 de 1/100, mientras que en base dos con una fracción que es la mitad y otra igual a la cuarta parate del círculo.

Al igual que en base diez, en base dos habrá fracciones que no puede expresarse mediante un número finito de dígitos. También como veremos, puede ocurrir que una fracción representable por un número finito de dígitos en base diez, requiera infinitos en base dos.

Entonces, del mismo modo que en base diez existen la décima, centésima, milésima, ... parte de la unidad, en base dos, se define la *mitad, cuarta, octava, diciseisava,* ... parte de la unidad.

þ De manera inversa, dado 0,11 en binario, para concer que fracción decimal se procede como se indica a continuación:

$$0,\overset{1/2}{\downarrow}1\overset{1/4}{\downarrow}_B = (1/2 + 1/4)_D = (0,5 + 0,25)_D = 0,75_D$$

A continuación generalizaremos el método para pasar a decimal un número binario con parte entera y decimal

$$10110,101_B = \overset{16\ 8\ 4\ 2\ 1\ 1/2\ 1/4\ 1/8}{\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow}\ 10110,101_B = (16+4+2)+(1/2+1/8)_D = (22 + 0,625)_D = 22,625$$

Regla para convertir un número binario con parte fraccionaria a decimal:

1. El bit que está a la iquierda de la coma binaria le corresponde peso 1.
2. A partir del mismo hacia la izquierda los pesos 2, 4, 8, 16, ... se escriben sobre los bits de la parte entera; y hacia la derecha los pesos 1/2, 1/4, 1/8, 1/16, 1/32 ... sobre los bits fraccionarios.
3. Se suman los pesos de las posiciones que tengan un uno, y el número se expresa en decimal.

El pasaje inverso de decimal a binario se realiza según la siguiente:

Regla para convertir un número decimal con partes entera y fraccionaria a binario:

1. Convertir a binario cada parte por separado. La parte entera se convierte por ejemplo conforme al método de las divisiones sucesivas por 2
2. La parte fraccionaria mediante sucesivas multiplicaciones por 2
3. Sumar ambas porciones antes halladas.

þ **Ejemplo**: convertir a binario el número decimal 35,62 = 35 + 0,62

1. $35_D = 100011_B$

2. Para hallar la parte fraccionaria binaria, se procede como sigue, por sucesivas multiplicaciones por 2 de cada nueva parte fraccionaria decimal (Su justificación se da en el Apéndice A2)

2x0,62 = 1,24 = **1** + 0,24
2x0,24 = 0,48 = **0** + 0,48
2x0,48 = 0,96 = **0** + 0,96
2x0,96 = 1,92 = **1** + 0,92
2x0,92 = 1,84 = **1** + 0,84
2x0,84 = 1,68 = **1** + 0,68
2x0,68 = 1,36 = **1** + 0,36
2x0,36 = 0,72 = **0** + 0,72
2x0,72 = 1,44 = **1** + 0,44
2x0,44 = 0,88 = **0** + 0,88
. . . . . .

Las sucesivas partes enteras halladas (en negrita) conforme van apareciendo, constituyen los bits de la parte fraccionaria binaria.
O sea $0,62_D \approx 0,1001111010_B$

3. $35 + 0,62)_D \approx (100011 + 0,1001111010)_B = 100011,1001111010_B$

En este caso se ha interrumpido arbitrariamente el proceso luego de haber obtenido una precisión de 10 bits fraccionarios, sin haber alcanzado una parte fraccionaria decimal igual a 0,00 como aparece por ejemplo si se pasa 0,75 a binario:

2x0,75 = 1,50 = **1** + 0,50
2x0,50 = 1,00 = **1** + 0,00 pudiéndose escribir la igualdad $0,75_D = 0,11_B$

Puede darse el caso que la determinación anterior no termine nunca, o sea, que un número decimal con parte fraccionaria pura, en base dos tenga su parte fraccionaria periódica.

Por ejemplo, cualquier número decimal con parte fraccionaria $0,1_D$ al convertirse en binario será periódico. Puede verificarse que $0,1_D = 0,\mathbf{0011}_B$ (periodo en negrita)

Este método puede aplicarse para pasar de decimal a **cualquier** base **R**, con tal de realizar en decimal el método de las divisiones o multiplicaciones sucesivas por **R**, para la parte entera y fraccionaria, respectivamente.

N.7 Potenciaciónen cualquier base

En cualquier base, siempre que se tengan **p** factores iguales de un número **n**, se podrá escribir: $n \times n \times n \times n \times \ldots n \times n = n^p$ según sea la base, variará la representación de **n** y **p**.

Resulta una buena ejercitación para sistemas numéricos, verificar las siguientes equivalencias:

$(1100 \times 1100 \times 1100)_B = 1100^{11}{}_B = 12^3{}_D = C^3{}_H$

$(10 \times 10 \times 10 \times 10)_B = 10000_B = 10^{100}{}_B$

$10^{100}{}_B = 2^4{}_D = (2 \times 2 \times 2 \times 2)_D = 16_D = 10_H = 2^4{}_H$

$1000_D = 10^3{}_D = 1010^{11}{}_B = (1010 \times 1010 \times 1010)_B = 1111101000_B$

$10^3{}_D = A^3{}_H = (A \times A \times A)_H = 3E8_H$

$10^3{}_H = (10 \times 10 \times 10)_H = 1000_H = 16^3{}_D = 4096_D$

$10^3{}_H = 10000^{11}{}_B = (10000 \times 10000 \times 10000)_B = 1000000000000_B$

Debe tenerse presente que **en cualquier base, la unidad seguida de p ceros puede expresar como la base a la potencia p,** ***simbolizándose 10 la base en todos los sistemas numéricos***

Así: $100000_B = (10 \times 10 \times 10 \times 10 \times 10)_B = 10^{101}$ siendo $10_B = 2_D$

$100_H = (10 \times 10)_H = 10^2{}_H$ siendo $10_H = 16_D$

N.8 Representación en punto flotante de números reales declarados como tales en alto nivel

Así como se definió –mediante el complemento a dos– una forma de representar números positivos y negativos mediante el bit extremo izquierdo, cuando se opera con números que presenten coma se requiere alguna convención representarlos mediante combinacione binarias.

Las representaciones de números reales en ***"punto flotante"*** [1] *("floating point"* = **FP**) –en castellano coma flotante– sirven para tal cometido, permitiendo además operaciones con magnitudes y resultados dentro de un amplio rango de valores, para aplicaciones desde las comerciales hasta los cálculos astronómicos. Además, obviamente, se puede operar con números enteros.

En esencia se trata de una representación de tipo exponencial: $\mathbf{N = \pm m \times 10^{\pm p}}$

semejante a la notación científica decimal, que permite en unos pocos bytes expresar números muy grandes o muy pequeños..

Su denominación se debe a que la posición del punto (coma), en la expresión anterior, puede desplazarse en el número **m** de la expresión anterior, si a la par se ajusta el exponente **p**, para adecuar la representación de operandos o resultados, si así se requiere.

Esto se vincula con el hecho de que los circuitos para operaciones en punto flotante (**coprocesador**) determinan en forma automática el lugar dónde va la coma en cada resultado, desentendiéndose de ello el programador. Cuando un computador no posee tales circuitos, se debe recurrir al software que opera con la **UAL** para emularlos, a costa de velocidad de procesamiento.

En notación científica estandar, los números se expresan de la forma: $N = \pm n\,E \pm p = \pm n \times 10^{\pm p}$

donde **n** es un número comprendido entre 1 y 10; **p** es un número entero.

Por ejemplo:

$0{,}00003 = 3.0\,E{-5} = 3 \times 10^{-5}$

$-246{,}36 = -2.4636\,E{+2} = -2{,}4636 \times 10^2$

$82000000000 = 8.2\,E{+10} = 8{,}2 \times 10^{10}$

En el papel **n** ó **p** pueden tomar cualquier valor, mientras que en el visor de una calculadora tienen valores máximos predeterminados. Supongamos que en un cierto formato **n** y **p** pueden tener hasta 5 y 2 dígitos, respectivamente; y que no escribimos la letra **E** o la base 10.

Los números anteriores podrían representarse como sigue: (+3,0000; –05) ; (–2,4636; +02) ; (+8,2000; +10)

También diremos que constan de una mantisa con 5 dígitos significativos o de precisión, y que cada factor de escala (10^{-5} 10^2 y 10^{10} según sea) determina la verdadera posición de la coma.

Normalización:

En notación exponencial siempre es posible correr ***k*** *lugares la coma a la izquierda (o derecha), si simultáneamente se incrementa (o decrementa) el exponente en un valor* ***k****, sin que cambie el valor del número representado.*

Así, sumando uno a los exponentes de los números anteriores, resulta:

$0{,}3 \times 10^{-4}$; $-0{,}24636 \times 10^3$; $0{,}82 \times 10^{11}$

---

[1] En contraposición los números de ***"punto fijo"*** o *"coma fija"* en castellano, suponen que la coma debe estar en una posición fija en un determinado formato de bits, según la conveniencia del cálculo a realizar. En particular si se supone que está luego del bit extremo derecho, se trata de un número entero. En un formato de 8 bits, estos enteros tienen un rango de 0 a 255 con resolución de una unidad entre una combinación y la siguiente. Si por ejemplo la coma se ubica a la izquierda del bit más significativo, todos los números de dicho formato serán fraccionarios, de la forma 0,..... . Entonces la resolución pasa a ser 1/256, pero el rango estará sólo entre 0 y 1. Vale decir que al aumentar el rango se pierde resolución. *A diferencia, punto flotante permite alta resolución y rango extenso.*

Decimos que se trata de una notación exponencial **"normalizada"**, de la forma: $\mathbf{m} \times 10^{\mathbf{p}}$

donde **m** es la *mantisa*, siendo de la forma $0{,}X_1X_2 \ldots X_n$ con $X_1 \neq 0$; o sea que debe ser distinto de cero el primer dígito que sigue a la coma. En general, será pues $0{,}1 \leq \mathbf{m} < 1$

Un número puede expresarse en forma normalizada en cualquier base.

Representación estándar para punto flotante del IEEE [1]

Existen varias convenciones para expresar números reales en punto flotante.

A los efectos de presentar la representación estándar para punto flotante del **IEEE**, ("Floating Point Standard" - **FPS**) consideraremos para los números binarios que $1_B \leq \mathbf{m} < 10_B$

o sea que **m** será de la forma: $\mathbf{m = 1{,}f}$ (**f** parte fraccionaria de **m**)

þ Por ejemplo, los siguientes números reales en base diez, convertidos a base dos , y en forma exponenencial normalizada serán:

$5_D = 101_B = (1{,}01 \times 100)_B = (1{,}01 \times 10^{10})_B$ [3] (se desplazó la coma $2 = 10_B$ lugares a la izquierda para normalizar)

$20_D = 10100_B = (1{,}01 \times 10000)_B .= (1{,}01 \times 10^{100})_B$ (se desplazó la coma $4 = 100_B$ lugares a la izquierda para normalizar)

$(-4101{,}25)_B = (-1000000000101{,}01)_B = (-1{,}00000000010101 \times 1000000000000)_B = (-1{,}00000000010101 \times 10^{1100})_B$

(en este caso se desplazó la coma $12 = 1100_B$ lugares a la izquierda)

$0{,}015625_D = (0{,}000001)_B$ [4] $= (1{,}0 \times \frac{1}{1000000})_B = (1{,}0 \times 10^{-110})_B$ (se corrió la coma $6_D = 110_B$ lugares a la derecha)

$1_D = 1_B = 1{,}0 \times 10^0{}_B$ (la coma no se corrió ningún lugar hacia derecha o izquierda)

$-0{,}5_D = 0{,}1_B = 1{,}0 \times 1/10_B = 1{,}0 \times 10^{-1}{}_B$ (se corrió la coma un lugar a la derecha)

$12{,}1_D = (12 + 0{,}1)_B = (1100 + 0{,}\underline{00011})_B$ [5] $= 1100{,}\underline{00011}_B = 1{,}10000\underline{011} \times 10^{11}{}_B$

Reglas básicas de representación en FPS:

- La representación es de la forma $\mathbf{N = \pm m \times 10^{\pm p} = \pm 1{,}f \times 10^{\pm p}}$ (siendo $10_B = 2$)
- En *simple precisión* cualquier número requiere 32 bits = 4 bytes.
- Sólo se representa la **parte fraccionaria f** de la mantisa **m**, reservándose para la misma los últimos **23 bits**, sobreentendiéndose que la parte entera es siempre *implícitamente* **1** y que existe una coma antes de **f** [6]
- El signo de la mantisa será un **bit de signo** (**S**) que vale **0** si es positiva, y **1** si es negativa, estando dicho bit el extremo izquierdo de la representación (separado de la mantisa)[7]
- Al exponente ± **p** se le debe sumar $\mathbf{127}_D$ ("exceso o desplazamiento 127"), resultando un número $\mathbf{c = \pm p + 127}$ para el cual se reservan **8 bits** a continuación del bit de signo citado.
- El número **cero** puede representarse con los 32 bits iguales a cero (+0), ó con el bit de signo de valor **1**, y los 31 restantes con valor cero (–0)
- Existe una convención para representar +∞ y –∞

Se aplicarán estas reglas a la representación en FPS de los números antes normalizados

---

1 Instituto de Ingenieros Electrónicos y Electricistas

3 En cualquier base , correr la coma **n** lugares a la derecha (izquierda) es multiplicar (dividir) por un uno seguido de **n** ceros. Asimismo este último número se puede expresar como $10^{\mathbf{n}}$ (que sería $10^{-\mathbf{n}}$ si la coma se corrió **n** lugares a la izquierda)

4 Este número se halla a partir del 0,015625 por sucesivas multiplicaciones por dos (sección N.6)

5 Valor determinado anteriormente

6 Aunque **f** no esté normalizado, se considera que **m = 1,f** sí lo está, pues tiene dicho uno en la posición más significativa. Este bit no forma parte del formato de 32 bits. La unidad de punto flotante (coprocesador) lo incorpora cuando debe operar.

7 Se trata pues de una representación de números del tipo signo y magnitud.

$20 = 1{,}01 \times 10^{100} =$	+	4+127=131	f	$10000011_B = 131_D$ se halla sumando 127_D al exponente $p = 100_B = 4_D$
	0	10000011	01000000000000000000000	
$-4101{,}25 = -1{,}00000000010101 \times 10^{1100} =$	–	12+127	f	$10001011_B = 139_D$ se halla sumando 127_D al exponente $p = 1100_B = 12_D$
	1	10001011	00000000010101000000000	
$0{,}015625 = 1{,}0 \times 10^{-110} =$	+	–6+127	f	$01111001_B = 121_D$ se halla sumando 127_D al exponente $p = -0110_B = -6$
	0	01111001	00000000000000000000000	
$-0{,}5_D = 0{,}1_B = 1{,}0 \times 10^{-1} =$	–	–1+127=126	f	
	1	01111110	00000000000000000000000	
$1 = 1{,}0 \times 10^{0} =$	+	0+127=127	f	
	0	01111111	00000000000000000000000	
$12{,}1 = 1{,}10000\underline{\mathbf{0011}} \times 10^{11} =$	+	3+127=130	f	
	0	10000010	10000011001100110011001	

Obsérvese que *no se emplea la convención de complemento a dos, ya sea para el exponente o la mantisa.* Esta se representa con bit de signo y magnitud normalizada[1] .

La coma está implícita, a la izquierda del bit más significativo de la parte fraccionaria de la mantisa. **Tampoco se representa la base $10_B = 2_D$**

Un exponente positivo más el exceso 127 puede formar un número entre:
01111111 = 127 = 0+127 y 11111110 = 254 = 127 + 127 (exponentes entre 0 y +128);
mientras que exponentes negativos excedidos en 127 forman números entre:
01111110 = 126 = −1 + 127 y 00000001 = 1 = −126 + 127 (exponentes entre −1 y −126).
Los valores máximo 11111111 y mínimo 00000000 para p+127 se reservan para representaciones especiales:

Con 11111111 y **f** = 0 se representa el infinito (+ ó − según el bit de signo)

Con 11111111 y **f** ≠ 0 se usa para indicar operaciones no válidas, como 0 por ∝
Con 00000000 y **f** = 0 se trata del número cero, como se anticipó.
Con 00000000 y **f** ≠ 0 el número está *desnormalizado:* tiene una magnitud menor que el valor mínimo que se representa en el formato normalizado.

En definitiva el intervalo de exponentes va de −126 a 127.

Siguiendo el camino inverso, puede hallarse cuál es el número real correspondiente a su representación en FP

Þ Por ejemplo, si en la memoria se lee el número en FP como

BFC00000 = **1**<u>01111110</u>11000000000000000000000 puede razonarse así:

a. El primer uno implica que es negativo;

b. Los 8 bits siguientes $01111110_B = 126_D = p + 127$ son el exponente excedido en 127, o sea $p = 126 - 127 = -1$

c. Los 23 bits restantes 11000000000000000000000 constituyen la parte fraccionaria (**f**), delante de los cuales debemos agregar un uno y la coma, que se quitaron cuando se pasó de decimal a FPL.

Así se forma la mantisa **m = 1,f** = 1, 11000000000000000000000

d. Puesto que $N = \pm \mathbf{m} \times 10^{\mathbf{p}}$ ($10_B = 2_D$); en este caso $N = -1{,}11 \times 10^{-1}{}_B = -0{,}111_B = -(1/2 + 1/4 + 1/8)_D = -0{,}875_D$

Existe también el formato FPS de ***doble precisión***, con 64 bits: 11 para el exponente excedido en $\mathbf{1023}_D$, 52 bits para la precisión de la parte fraccionaria **f** de la mantisa, y uno para el signo.

Rangos de representación:
Determinaremos primero el mayor número positivo representable ($+N_{MAX}$) que en punto flotante tendrá máximo exponente (11111110 subrayado) y mantisa (23 unos): <u>01111111</u>11111111111111111111111

1 Esta convención prevee operaciones con mantisas no normalizadas.

Siguiendo los pasos a,b,c,d anteriores, resulta:

$11111110_B = 254 = p + 127$; luego $p = 254 - 127 = 127$

$\mathbf{m} = 1{,}11111111111111111111111_B \approx 1{,}99_D \approx 2_D$

$+N_{MAX} = m \times 2^{p}{}_D \approx 2 \times 2^{127}{}_D$

Si se quiere expresar $2^{127}{}_D$ como una potencia de diez, debe ser $2^{127}{}_D = 10^{Y}{}_D$

Aplicando logaritmos: $127 . \log_{10} = Y . \log_{10} 10$ de donde $127 . 0{,}3 = Y . 1$

Resultando $Y = 38{,}1 \approx 38$ y por lo tanto $+N_{MAX} = 2x2^{127}{}_D \approx 2x10^{38}{}_D$

10^{38} es un uno seguido de 38 ceros, por lo que pueden representarse números con hasta 38 dígitos

$-N_{MAX}$ tendrá la misma magnitud que $+N_{MAX}$ dado que su representación sólo diferirá en que empieza con **1**:

$-N_{MAX} = 1\underline{11111110}11111111111111111111111$ Entonces resulta: $-N_{MAX} \approx -2x10^{38}{}_D$

Un número fraccionario positivo más pequeño ($+N_{FMIN}$) en FP se define para $\mathbf{p}+127 = 00000001$, y con $\mathbf{f} = 0$

$+N_{FMIN} = 0\underline{00000001}00000000000000000000000$

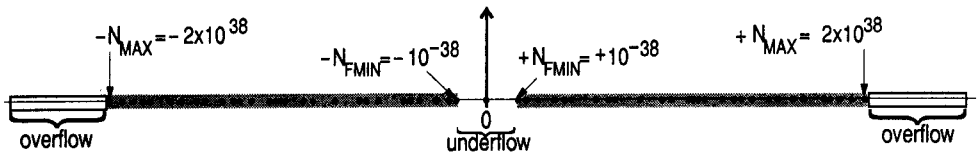
de donde $00000001 = 1 = p + 127$ por lo que $p = -126$; $m = 1,f = 1{,}00000000000000000000000 = 1$

o sea que en decimal se tiene $+N_{FMIN} = 1x2^{-126} \approx 10^{-38}{}_D = 0{,}0\ldots.01_D$ (cero - coma - 38 ceros - uno)

(38 surge de aplicar logaritmos siendo ahora $Y = 126 \times 0{,}3 = 37{,}8 \approx 38$)

El fraccionario negativo mínimo representable correspondiente al anterior tendría igual magnitud, diferiendo en el bit de signo. Sería en decimal: $-N_{FMIN} = -1x2^{-127}{}_D \approx -10^{-38}{}_D = -0{,}0\ldots.01_D$ (cero - coma - 38 ceros - uno)

De acuerdo a los resultados obtenidos, se tiene el siguiente rango de representación:



Si algún dato o resultado positivo o negativo presenta parte entera con magnitud que supere N_{MAX}, se tendrá *"overflow"*, pues su exponente superaría el valor máximo 128. Y si el mismo es un fraccionario positivo o negativo de magnitud menor que 10^{-38}, se tendrá *"underflow"*.

Debe notarse que los números reales representados en PF constituyen sólo un conjunto finito de números reales racionales. Asimismo, sus valores no están uniformemente distribuídos en la recta que los representa, como denotan los puntos irregularmente dibujados, y puede deducirse de la expresión $N = mx10^{p+127}$

En el punto c del problema 2 de la sección N.11, se indica cómo sumar números en punto flotante.

Manejo del underflow*: *desnormalización

Otra de las ventajas de la convención del IEEE, es que el coprocesador puede manejar resultados que arrojen underflow (Obsérvese de paso que esta convención determina el hardware de los actuales coprocesadores, insertos hoy día en el microprocesador). Para tal fin, se definen en FP números *"desnormalizados"*.

Como se vio, el número normalizado más pequeño es $+N_{FMIN} = 0\underline{00000001}00000000000000000000000 = 1x2^{-126}$

El mayor número desnormalizado es $0\underline{00000000}11111111111111111111111 = 0.999999 \times 2^{-127}$, de magnitud muy cercana a $+N_{FMIN}$. Asimismo, estos números son del tipo 0, . . . en lugar de tener el bit 1 implícito a la izquierda, y además exp +127 = 00000000, siendo que los normalizados no admiten el 00000000.

El número desnomalizado más pequeño es $0\underline{00000000}00000000000000000000001$. Resulta, que el exponente es −127, y la parte fraccionaria 2^{-23}; por lo que el valor de dicho número es tan pequeño como $2^{-23} \cdot 2^{-127} = 2^{-150}$

Por lo tanto, cuando un resultado da underflow, el coprocesador opera con números desnomalizados, con los primeros bits de la parte fraccionaria transformados en ceros (reduciéndose el número de bits significativos), y con exp+ 127 = 0

Simple precisión en punto flotante de 32 bits:

Puesto que los números extremos representables en FP son $\pm 10^{38}$, ello implica que, como ser, podemos tipear en el teclado un número en base diez de 37 cifras, que será aceptado dentro del rango de representación. Pero, como se verificará, devolverá con precisión las 7 primeras de sus cifras, dado que en la representación tratada sólo se disponen de 23 bits para guardar cualquier número representable en FP. Esta limitación de la cantidad de dígitos que se pueden representar, propia de cualquier representación de números en un computador (sean reales, integers u otros) determina que siempre se trate de "**números con precisión finita**".

Ejemplo: sea el número $4.294.967.294_D = 11111111111111111111111111111110_B$ que normalizado y representado en

FP resulta $1,\mathbf{11111111111111111111111111111110} \times 10^{11111\ (31)}{}_B = 0\underline{10110100}\mathbf{11111111111111111111111\ 11111110}$

31+127=158 ←====== 23 bits ======→

Como sólo pueden almacenarse 23 bits de la parte fraccionaria, resulta que el número queda truncado, con 8 bits fuera del formato (11111110 = 254). Suponiendo que no exista redondeo para el truncamiento dichos 8 bits se considerarían ceros, por lo que el número almacenado sería:

1111111111111111111111100000000 = 4.294.967.294 – 254 = 4.294.967.040

Se verifica, como se anticipó, que sólo se han conservado intactos 7 dígitos de precisión. Esto está de acuerdo con el hecho de que **a cada dígito decimal le corresponden 3,5 bits**, por lo que 23 bits permiten representar 23/3,5 dígitos, cociente cercano a 7.

Punto flotante doble precisión (64 bits):

Un Pentium está preparado para procesar 64 bits en FP, según la convención para este formato del IEEE, el primer bit es el del signo; luego siguen 11 bits para el exponente más 1023 de exceso, quedando los 52 bits restantes para representar la parte fraccionaria del número normalizado. Esto permite una precisión de 52/3,5 dígitos, cociente cercano a 15.

N.9 Codificación y suma en BCD natural

Según se vió, para pasar del sistema decimal al binario es necesario realizar una serie de operaciones aritméticas, como en el método manual de las sucesivas divisiones por dos.

Los "***códigos BCD***" (*Binary-Coded-Decimal:* decimal codificado en binario) se emplean para convertir *directamente*, sin cálculo alguno, números decimales en combinaciones binarias, según determinadas convenciones, que tienen en común el hecho de que ***a cada digito decimal le corresponden 4 bits***.

La convención ***"BCD natural"*** o ***"BCD 8-4-2-1"*** atribuye a los símbolos decimales la correspondencia binaria dada por la tabla de la figura 2.2 que volvemos a repetir:

	8-4-2-1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Dado un número cualquiera en base diez, para convertirlo a binario BCD basta reemplazar cada dígito del mismo por los 4 bits correspondientes

Esta metodología de conversión es semejante a la utilizada para pasar de hexadecimal a binario para los símbolos del 0 al 9:

$$8125_D = \overbrace{1000}^{8\text{-}4\text{-}2\text{-}1}\ \overbrace{0001}^{8\text{-}4\text{-}2\text{-}1}\ \overbrace{0010}^{8\text{-}4\text{-}2\text{-}1}\ \overbrace{0101}^{8\text{-}4\text{-}2\text{-}1}{}_{BCD}$$

$$\qquad\qquad 8 \quad 1 \quad 2 \quad 5$$

mientras que en el sistema binario se representa

$$8125_D = \overbrace{1}^{4096}\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1_B \qquad (128\ 64\ 32\ 16\ 8\ 4\ 2\ 1)$$

Asimismo, $10_D \equiv 00010000_{BCD}$; siendo en cambio $10_D = 1010_B$

De lo anterior surge claramente, que si bien las conversiones de decimal a BCD, y de hexadecimal a binario natural, se realizan de igual forma, el significado de la combinación binaria es bien diferente. Se usó el símbolo ≡ para indicar equivalencia de representación entre BCD y decimal, y no el símbolo = , pues se trata de una convención.

El pasaje inverso de BCD a decimal consiste en separar en el número binario grupos de 4 bits, y determinar en base diez qué dígito decimal representa cada cuarteto:

Así: $0\,0\,1\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0_{BCD} \equiv 2900_D$

$\lfloor\ \mathbf{2}\ \rfloor\lfloor\ \mathbf{9}\ \rfloor\lfloor\ \mathbf{0}\ \rfloor\lfloor\ \mathbf{0}\ \rfloor$

En definitiva, podemos sistematizar estó así, para indicar que el pasaje es directo sólo en sentido horizontal:

Hexadecimal ⇐ 1 simbolo por cuarteto de bits ⇒ **Binario natural**

Decimal ⇐ 1 simbolo por cuarteto de bits ⇒ **BCD**

Para pasar de decimal a hexa, o en sentido inverso, hay que hacer las operaciones indicadas en la unidad 1.
Así: $0010100100000000_{BCD} \equiv 2900_D = B54_H = 101101010100_B$; siendo también: $0010100100000000_B = 2900_H$

La desventaja del BCD respecto a los binarios naturales, es que en BCD en un byte se pueden representar combinaciones desde 00 a 99, mientras que con los segundos se puede codificar de 0 a 255, en un byte. Asimismo, sólo conviene realizar sobre datos en BCD la suma y resta, existiendo instrucciones para corregir (ajustar) los resultados que se obtienen de sumar cuartetos BCD como si fueran números binarios. En esencia el formato BCD es un buen compromiso entre el tiempo de convertir números de ASCII a binarios –que en BCD es directo– y el tiempo que insumen sumas y restas, que en BCD es bastante corto, según podrá deducirse

Suma de números en BCD natural

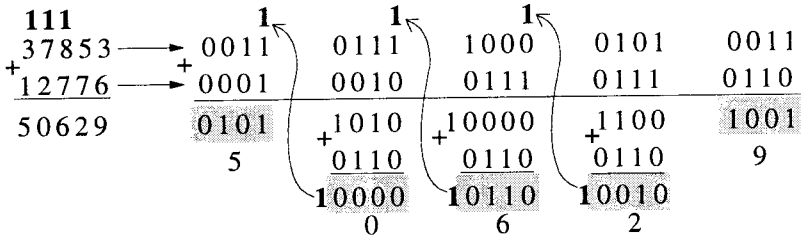
*Un **método para sumar** dos números decimales codificados en BCD "natural" consiste en sumar los cuartetos que los constituyen como si fueran números naturales, y luego sumar* $6 = 0110_B$ *si la suma parcial de dos cuartetos supera* $9 = 1001_B$

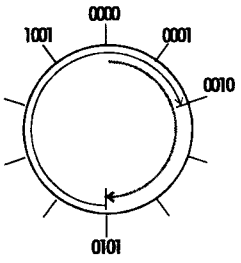
Esto último puede ocurrir de dos formas:

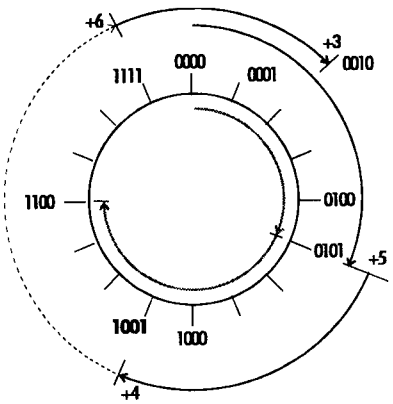
1. Si dicha suma resulta con valores 1010 a 1111 (10_D a 15_D): combinaciones que **no** son BCD
2. Si ella es **1**0000, **1**0001, **1**0010, **1**0011 (16_D a 19_D [2]): combinación BCD con **1** de acarreo

þ Se ejemplifica una suma efectuada con este método, aplicable a cualquier número de dígitos.

Las combinaciones a corregir por no ser BCD, o por que el uno fuera del cuarteto indica resultado ≥ 16, se les ha sumado 0110. En negrita aparece el uno de acarreo en decimal y en BCD.







Para justificar por qué se suma 6, apelaremos a un cuenta-vueltas binario con números del 0000 al 1111 (módulo 16), razonando como sigue. Si (dibujo de la derecha) a partir del 0000 avanzamos 5 posiciones se llega al 0101, y si luego progresamos 7 más se alcanza la 1100, habiendo realizado la suma 5 + 7 = 0101 + 0111 = 1100 con números binarios naturales.

La misma suma en el cuenta-vueltas izquierdo para decimales y BCD (con números de 0000 a 1001): si desde el 0101 se avanza 7 posiciones más se alcanza el 0010 luego de haber dado una vuelta, en correspondencia con el número 12 = $0001\ 0010_{BCD}$

Si queremos usar el cuenta-vueltas binario (con 6 pasos más que el otro) para sumar en BCD, cada vez que al sumar alcanzamos el valor 1001 debemos sumarle 6 = 0110, de modo de pasar al 1111, que está a una unidad del 0000, del mismo modo que lo está el 1001 respecto del 0000 en el cuenta-vueltas BCD.

Podemos pensar esto último como que en el cuenta vueltas binario la suma 5+7 la realizamos como 5+*4*+**6**+*3*. Esto es, descomponemos 7= *4*+*3*, para mostrar que al realizar la suma parcial 5+*4* habremos llegado al 1001 en este cuenta-vueltas, por lo que le sumamos 6=0110 para pasar al 1111, y luego sumamos las 3 unidades que faltaban para completar las 7 de la cuenta. De esta forma, como se ha dibujado, se alcanza el 0010, como debe ser en la suma BCD.

Puesto que en una suma no importa el orden de los sumandos, en vez de efectuar 5+4+6+3 se hace directamente 5+7+6 = 0101+0111+0110 como se realizó para uno de los cuartetos de la suma BCD arriba ejemplificada.

2 La suma de dos dígitos decimales a lo sumo es 19, cuando se suma 9 + 9 + 1 (1 de acarreo de la posición anterior)

Decimal (BCD) empaquetado (packed)

El término *"empaquetado"* se refiere a una representación numérica con uno o más bytes, siendo que en los dos cuartetos de cada byte están codificados en BCD dos dígitos decimales.

En contraposición en BCD *"desempaquetado"* cada dígito decimal ocupa un byte, como ser 3 = 00000011

Existe una representación "clásica" basada en los 4 bits por dígito del 0 al 9 del código BCD, con caracteres adicionales para indicar signo mediante otros 4 bits extras de las combinaciones restantes (1010 a 1111, de A a F) que irán en el extremo derecho. Los números positivos pueden terminar en *cualquiera* de las combinaciones: 1100 (C), 1010 (A), 1111 (F) ó 1110 (E). Para los negativos se pueden usar indistintamente 1011 (B) ó 1101 (D)

Los procesadores de Intel™ para codificar signo agregan un byte extremo izquierdo, cuyo primer bit (0 ó 1) indica el signo. Los siete bits siguientes pueden ser de valor cualquiera (elegidos ceros en los ejemplos siguientes)

þ **Ejemplos:**

	CONVENCION "CLASICA"		CONVENCIÓN DE INTEL™
310289 ≡	0000 0011 | 0001 0000 | 0010 1000 | 1001 1111	≡	0000 0000 | 0011 0001 | 0000 0010 | 1000 1001
	0 3 1 0 2 8 9 F		+ 3 1 0 2 8 9
+26998 ≡	0010 0110 | 1001 1001 | 1000 1100	≡	0000 0000 | 0000 0010 | 0110 1001 | 1001 1000
	2 6 9 9 8 C		+ 0 2 6 9 9 8
−26998 ≡	0010 0110 | 1001 1001 | 1000 1101	≡	1000 0000 | 0000 0010 | 0110 1001 | 1001 1000
	2 6 9 9 8 D		− 0 2 6 9 9 8
1 ≡	0001 1111	≡	0000 0000 | 0000 0001
	1 F		+ 0 1

En ambas convenciones un número debe ocupar un número entero de bytes. En la "clasica" al 310289 hubo que agregarle un cero a la izquierda, siendo que F indica que el número es positivo, no siendo necesario hacerlo en los tres ejemplos siguientes. El + 26998 difiere del −26998 en el cuarteto final indicador de signo, habiéndose elegido C y D para uno y otra caso. Los tres últimos ejemplos requirieron un cero a la izquierda en la otra convención.

Cabe mencionar que existe otra convención tipo "clasica" para números previamente representados en ASCII. Para los mismos se sigue usando la D para negativos, pero los positivos terminan en 0011 = 3.

Así, el +26998 = 0010 0010 1001 1001 1000 0011 = 269983, siendo que el −26998 se representa igual.

Decimal desempaquetado con signo

Cada dígito se codifica en 8 bits ASCII (o EBCD). Los 4 bits superiores de cada byte (0011 en ASCII, 1111 en EBCD), se conocen como bits *"de zona"*. El signo se representa mediante los 4 bits de zona del byte menos significativos.
En ASCII estos bits son 0011 = 3 y 1101 = D para positivos y negativos, respectivamente. En EBCD estos bits son D para los negativos, y para los positivos 1111 = F ó 1100 = C
Ejemplo para ASCII:

+26998 = 00110010 00110110 00111001 00111001 **0011**1000
−26998 = 00110010 00110110 00111001 00111001 **1101**1000

en negrita se indican los bits de zona dedicados al signo.

N.10 Recapitulación de las posibles codificaciones de datos numéricos procesables por la UAL o el coprocesador

El esquema de la página siguiente se intenta mostrar las codificaciones básicas por las que pasa un dato numérico desde que es entrado desde el teclado hasta que debe ser procesado; y por las que pasa un resultado hasta que se hace visible en un monitor o en papel impreso con los símbolos del sistema decimal. Se ha supuesto que se han tipeado las tres cifras que forman un número (250), el cual puede tener distintas alternativas de codificación según el tipo de datos (naturales, enteros, reales, etc) que se hayan definido en un supuesto programa en alto nivel (Pascal, C, Basic, Cobol, etc), siendo que luego en todas esas alternativas se supone que en algún punto de un programa (desarrollado para procesar el tipo de dato en cuestión) se ordena sumarle uno a dicho dato (obteniéndose 251)

Cuando este programa codificado en código de máquina se ejecute (etapa indicada como "procesamiento de los datos tipeados"), también se asume que en algún momento se obtendrá 251, codificado conforme al tipo de dato que se está usando en cada alternativa. Luego se plantea supone que dicho resultado debe aparecer en pantalla (o en papel impreso) como un número de tres cifras decimales. Para tal fin, el monitor (o la impresora) en todos los casos debe recibir dichas tres cifras que componen el 251 en código ASCII.

Por ejemplo, suponiendo (*nivel 1*) que se tipee 250 como dato a procesar de un programa desarrollado para números enteros. Entonces, al tipear el 2, la circuitería del teclado primero codificará el código de dicha tecla (*"scan code"*), que es 00000011, y la enviará a memoria principal (**MP**). Luego, la ejecución de una subrutina del BIOS permitirá determinar el código ASCII del 2, que es 00110010, el cual también quedará en memoria principal. Puesto que la tecla en cuestión además del 2 tiene arriba el símbolo @, dicha subrutina deberá determinar que no se pulsó antes la tecla Shift (para lo cual ella también debe determinar si el código de esta tecla no estaba antes en memoria). Del mismo modo, luego de tipear el 5, este quedará en memoria en ASCII como 00110101; y el 0 como 00110000

En el *nivel 2* hay que pensar que se llamó o debe actuar una *subrutina de traducción*, cuya ejecución permite reconocer que el 2, el 5 y el 0 codificados separadamente en ASCII componen el número 250; y que luego convierte éste a binario en formato 16, lo invierte y le suma uno, de modo de representarlo como **0**000000011111010.

De esta forma, éste y otros datos antes tipeados quedan codificados con bit de signo –y por lo tanto operables en la UAL– cuando se ejecute el programa citado (codificado en código de máquina). En tal oportunidad (*nivel 3*) se supone que una instrucción ordena sumarle uno al **0**000000011111010, para así obtener **0**000000011111011, como se indica, el cual podrá guardarse en memoria principal ocupando dos bytes.

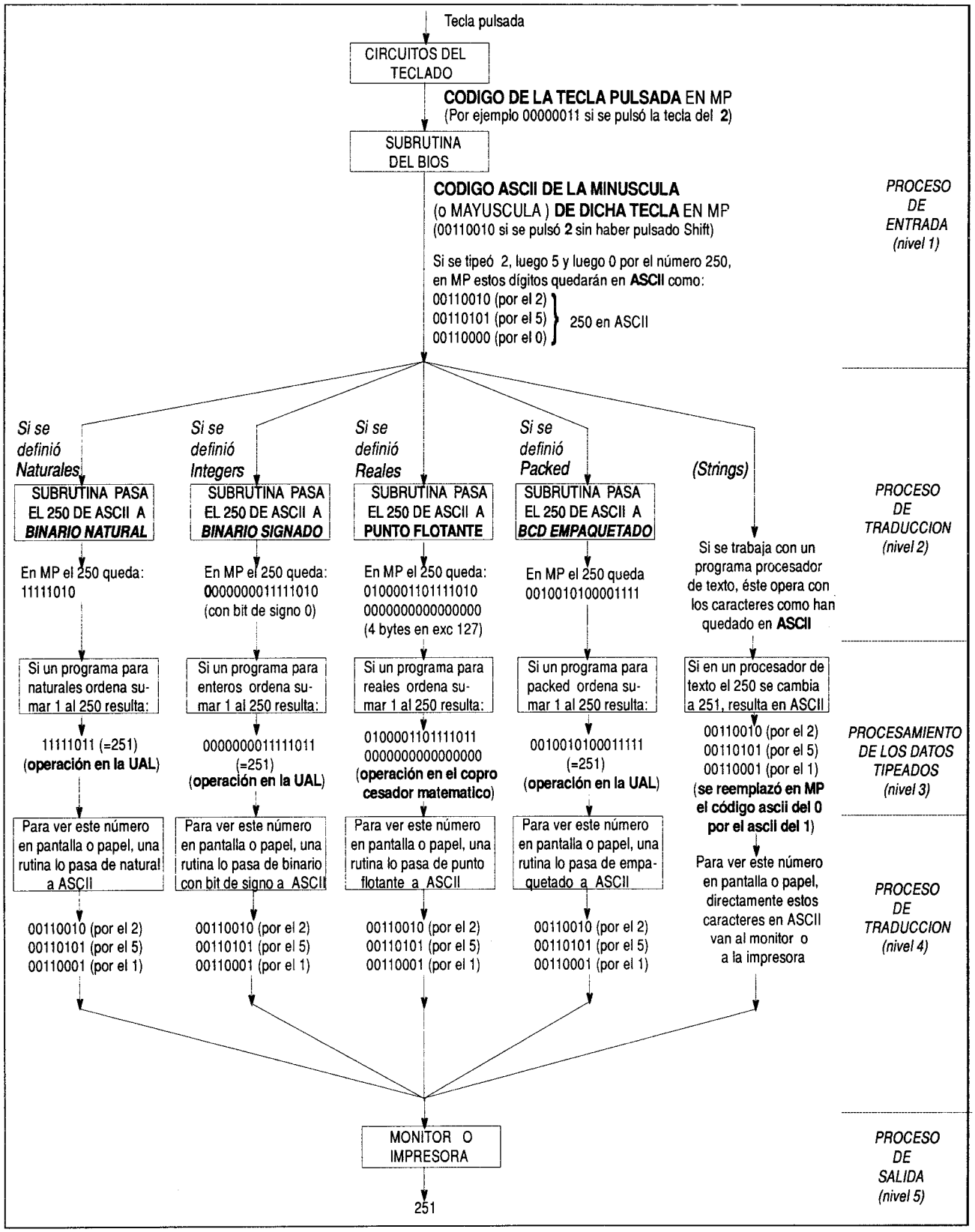
A posteriori (*nivel 4*), si se ordena una salida por impresora (o por monitor) para que este número se vea como el 251, otra subrutina traductora al ser ejecutada pasará el **0**000000011111011 con bit de signo a un número decimal de tres dígitos: 2, 5 y 1, que en ASCII quedarán en memoria como 0011010, 00110101 y 00110001, respectivamente. Estos así codificados serán transferidos al monitor o a la impresora merced a la ejecución de otra subrutina del BIOS.

Uno u otro de estos periféricos convertirán estos dígitos en el número decimal 251 visualizable (*nivel 5*).

De igual modo pueden seguirse las distintas codificaciones que ocurren para otros tipos de datos.

Si se está trabajando con un procesador de texto, los caracteres tipeados una vez que están en memoria codificados en ASCII son directamente datos para el programa procesador. Utilizando éste será factible seleccionar en pantalla el número 0 del 250 y reemplazarlo por el 1, lo cual implica cambiar el código ASCII 00110000 por el 00110001.

Se visualiza en pantalla el 251 merced a que se transfirieron al monitor los correspondientes caracteres ASCII.


Tecla pulsada
CIRCUITOS DEL TECLADO
CODIGO DE LA TECLA PULSADA EN MP
(Por ejemplo 00000011 si se pulsó la tecla del 2)
SUBRUTINA DEL BIOS
CODIGO ASCII DE LA MINUSCULA
(o MAYUSCULA) DE DICHA TECLA EN MP
(00110010 si se pulsó 2 sin haber pulsado Shift)
Si se tipeó 2, luego 5 y luego 0 por el número 250, en MP estos dígitos quedarán en ASCII como:
00110010 (por el 2)
00110101 (por el 5)
00110000 (por el 0)
250 en ASCII
PROCESO DE ENTRADA (nivel 1)
Si se definió Naturales
Si se definió Integers
Si se definió Reales
Si se definió Packed
(Strings)
SUBRUTINA PASA EL 250 DE ASCII A BINARIO NATURAL
SUBRUTINA PASA EL 250 DE ASCII A BINARIO SIGNADO
SUBRUTINA PASA EL 250 DE ASCII A PUNTO FLOTANTE
SUBRUTINA PASA EL 250 DE ASCII A BCD EMPAQUETADO
Si se trabaja con un programa procesador de texto, éste opera con los caracteres como han quedado en ASCII
PROCESO DE TRADUCCION (nivel 2)
En MP el 250 queda: 11111010
En MP el 250 queda: 0000000011111010 (con bit de signo 0)
En MP el 250 queda: 0100001101111010 0000000000000000 (4 bytes en exc 127)
En MP el 250 queda 0010010100001111
Si un programa para naturales ordena sumar 1 al 250 resulta:
Si un programa para enteros ordena sumar 1 al 250 resulta:
Si un programa para reales ordena sumar 1 al 250 resulta:
Si un programa para packed ordena sumar 1 al 250 resulta:
Si en un procesador de texto el 250 se cambia a 251, resulta en ASCII
11111011 (=251) (operación en la UAL)
0000000011111011 (=251) (operación en la UAL)
0100001101111011 0000000000000000 (operación en el copro cesador matematico)
0010010100011111 (=251) (operación en la UAL)
00110010 (por el 2) 00110101 (por el 5) 00110001 (por el 1) (se reemplazó en MP el código ascii del 0 por el ascii del 1)
PROCESAMIENTO DE LOS DATOS TIPEADOS (nivel 3)
Para ver este número en pantalla o papel, una rutina lo pasa de natural a ASCII
Para ver este número en pantalla o papel, una rutina lo pasa de binario con bit de signo a ASCII
Para ver este número en pantalla o papel, una rutina lo pasa de punto flotante a ASCII
Para ver este número en pantalla o papel, una rutina lo pasa de empaquetado a ASCII
Para ver este número en pantalla o papel, directamente estos caracteres en ASCII van al monitor o a la impresora
00110010 (por el 2) 00110101 (por el 5) 00110001 (por el 1)
PROCESO DE TRADUCCION (nivel 4)
MONITOR O IMPRESORA
PROCESO DE SALIDA (nivel 5)
251

N.11 Ejemplos integradores de conocimientos

Los siguientes ejemplos sirven para sistematizar conocimientos, y deben verse a la luz de los cinco niveles del esquema de la página anterior.

1. Desde el teclado de una terminal se edita un cierto programa en Basic, habiéndose definido al comienzo del mismo que las variables serán números *enteros*. Una de las sentencias tipeada es:

20 **W = P – Q + R** habiéndose asignado P = –81 Q = 16 R = 82

a. Conociendo que cada tecla tipeada se guarda primero en memoria principal según su código ASCII, ocupando un byte, indicar como quedarían en supuestas posiciones consecutivas de memoria el tipeado de P = –81

b. Siendo que se han definido variables enteras, el programa traductor tendrá por ejemplo como "materia prima" los caracteres codificados en binario ASCII hallados para la pregunta anterior y la ejecución de dicho traductor permitirá que los números tipeados pasen de ASCII a binarios con bit de signo en formato 16. Indicar cómo deja este programa codificadas en supuestas posiciones consecutivas de memoria los valores asignados a P, Q y R con bit de signo en dicho formato.

c. Suponiendo que la sentencia **W = P – Q + R** fue traducida a código de máquina mediante una secuencia de instrucciones entre las cuales se encuentra una instrucción de resta, seguida de otra de suma, indicar cómo sería la operación en la UAL cuando se ejecutan estas instrucciones, en función de los datos hallados en el paso anterior.

d. Verificar que el resultado obtenido en la UAL pasado a base diez, es el mismo que resulta de hacer –81 – 16 + 82

e. Escribir como queda en memoria principal dicho resultado

f. Suponiendo que otra sentencia del programa en cuestión sea: **21 PRINT "W"=W** , o sea que se dió la orden de imprimir el valor del resultado asignado a **W**, y en conocimiento de que la impresora debe recibir los caracteres a imprimir en ASCII, ¿cómo estarían codificados éstos en posiciones sucesivas de memoria principal ?

g. Si a la sentencia citada en la pregunta anterior le sigue **22 IF P < Q THEN GO TO 30** ¿de qué manera la máquina determina que P < Q y que la misma se refiere a números signados?

h. En el nivel 1, de acuerdo con la tabla del código ASCII:

P es 50_H = 01010000; = es $3D_H$ = 00111101; – es $2D_H$ = 00101101;

8 es 38_H = 00111000; 1 es es 31_H = 00110001; que en posiciones de memoria principal quedarían como

01010000
00111101
00101101
00111000
00110001

b. La subrutina de traducción (nivel 2) en formato 16 convertiría

P = 00101101 00111000 00110001_{ASCII} en 1111111110101111
– 8 1

Q = 00110001 00110110_{ASCII} en 0000000000010000
1 6

R = 00111000 00110010_{ASCII} en 0000000001010010
8 2

Estos números, supuestos datos consecutivos en memoria principal quedarán (escritos horizontalmente) así:

11111111 10101111 00000000 00010000 00000000 01010010

c. En el nivel 3, para hacer P – Q , la **UAL** sumará a P el complemento a la base de Q, que lo obtendrá invirtiendo Q,

(resultando 1111111111101111) y sumándole uno (sección N.4), de modo que en definitiva realizará:

1111111110101111 + 1111111111110000 = 1111111110011111 y será **SZVC** = 1000 (**C** cambia de valor en la resta)

A este resultado le sumará R, de modo de realizar durante la instrucción de suma:

1111111110011111 + 0000000001010010 = 1111111111110001 con **SZVC**=1000

d. 1111111111110001 ≡ – (0000000000001110 + 1) = –1111 = –15 (se verifica)

e. 11111111
11110001

f. En el nivel 4 resultará: 01010111 W (57)
00111101 = (3D)
00101101 – (2D)
00110001 1 (31)
00110101 5 (35)

g. En el nivel 3: primero habría que efectuar una comparación mediante la resta P –Q , para determinar **SZVC**, operación que ya se realizó en el punto **c.** resultando los valores **SZVC** = 1000.
A continuación, una instrucción para números signados, de saltar si el minuendo es menor que el sustraendo, determinará esta condición por el valor de **S** y **V** (Sección N.5). Como en este caso se cumple dicha condición de salto, se pasará a ejecutar las instrucciones de máquina correspondientes a la sentencia 30, en lugar de continuar con la sentencia siguiente.
Que la operación que hizo la UAL se interprete para números signados o no, depende de cuáles señalizadores están involucra dos en la instrucción que ordena el salto citado. Como se vió al tratar **SZVC**, los indicadores **SVZ** son para signados.
Se verifica que para la UAL todos los números que opera son naturales. Gracias a la representación por complemento a la base, los números signados se suman o restan como naturales, desechando el uno fuera de formato, para efectuar la corrección.
La UAL genera **SZVC** por si van a ser usados por instrucciones de salto para signados o no signados (naturales). Vale decir, que las instrucciones de salto permiten interpretar los números como signados.

2. Suponiendo que en el programa anterior se han definido variables **"reales"** en vez de "integers", repetir los puntos **a** hasta **g**

a. Idem problema anterior

b. Siendo que se han definido "reales", es programa traductor transformará (nivel 2) los datos de ASCII a punto flotante, conforme a lo tratado en la sección N.8, resultando para la convención dada:

$P = -1{,}010001 \times 10^{110} = 11000010101000100000000000000000$

$Q = 1{,}0 \times 10^{100} = 01000001100000000000000000000000$

$R = 1{,}010010 \times 10^{110} = 01000010101001000000000000000000$ que en memoria quedarían (escritos horizontalmente):

11000010 10100010 00000000 00000000 01000001 10000000 00000000 00000000 01000010 10100100 00000000 00000000

c. Las operaciones de suma seguida de resta serían ordenadas por instrucciones para punto flotante –correspondientes a variables definidas como "reales"– en vez de instrucciones de suma y resta para enteros. Suponiendo que el computador tenga una unidad de punto flotante (FPU), o sea un coprocesador matemático, esas operaciones activarían éste, cuyo hardware (nivel 3) haría primero P – Q y al resultado parcial le sumaría R, ambas mediante instrucciones para números en punto flotante..

Para hacer P – Q dentro del coprocesador los números deben ser operados como números en forma exponencial:

$P = -1{,}010001 \times 10^{110}$; $Q = 1{,}0 \times 10^{100} = 0{,}01 \times 10^{110}$ *(se han igualado exponentes para poderlos operar)*

$P - Q = -1{,}010001 \times 10^{110} - 0{,}01 \times 10^{110} = -1{,}100001 \times 10^{110}$;

y $W = P - Q + R$ sería: $-1{,}100001 \times 10^{110} + 1{,}010010 \times 10^{110} = -0{,}001111 \times 10^{110} = -1{,}111 \times 10^{11} =$

$=$ 11000001111100000000000000000000 que en memoria principal ocuparía cuatro posiciones consecutivas

f. Idem que – 15 del problema anterior

g. Semejante al problema anterior, pero con números en punto flotante.

3 En una UAL para números de 8 bits se realiza la suma 10010001 + 11110100 a fin de efectuar una suma ordenada por una instrucción de suma.

a. Indicar el resultado de dicha suma, así como los valores de **SZVC** que genera la UAL, y si de la suma efectuada se puede deducir que se sumaron números enteros o naturales.
b. Si los números que originaron la operación fueron codificados en el nivel 2 como *naturales* indicar cuáles fueron ellos, y qué números decimales se tipearon en correspondencia en el nivel 1. Indicar además cómo quedaría el resultado almacenado en memoria, y si el mismo coincide con el que resultaría si la cuenta se efectuaría en el sistema decimal.
c. Idem anterior si en el nivel 2 fueron codificados como números *con bit de signo*

a. 10010001 + 11110100 = 110000101 siendo **S**=1, **Z**=0, **V**=0, **C**=1

Esta suma pudo haber sido hecha para naturales o para enteros

b. Los números naturales que originaron la operación fueron 10010001 y 11110100, los cuales se originaron al tipear 145 y 244
En memoria el resultado incluye al noveno bit de carry, por lo que en memoria quedará en dos posiciones sucesivas como:

00000001
10000101

Este resultado luego de ser procesado en los niveles 4 y 5 aparecerá como 389, que coincide con el de la suma en decimal

c. Los números con bit de signo que originaron la operación fueron 10010001 y 11110100, los cuales provienen de tipear los números enteros decimales −111 y −12, como puede verificarse. El resultado en memoria quedará como 10000101, el cual luego de ser procesado en los niveles 4 y 5 será visualizable como −123, acorde con.la operación (−111) + (−12)

4. Idem problema anterior, pero suponiendo que la suma fue realizada para llevar a cabo una instrucción de resta.

a. 10010001 + 11110100 = 110000101 siendo **S**=1, **Z**=0, **V**=0, **C**=0 (por tratarse de una resta, el carry que se indica es el borrow, como se vió en lasección N.4), por lo que la indicación de **C** será opuesta al que resulta de realizar la suma) Esta suma pudo haber sido hecha para restar números naturales o para enteros

b. Dado que para restar en la **UAL** al minuendo se le suma el complemento del sustraendo, los números naturales a restar fueron 10010001 y 00001100. El segundo (el sustraendo) se obtuvo de invertir el 11110100 y sumarle uno. Dichos números a restar se originaron al tipear en el primer nivel 145 = 10010001 y 12 = 00001100

En memoria el resultado quedará en una posición, como 10000101 (en la resta de naturales el bit **C** no forma parte del resultado). Este, luego de ser procesado en los niveles 4 y 5 aparecerá como 133, que coincide con el de la resta en decimal.

c. Razonando como en b), los números con bit de signo que originaron la operación fueron 10010001 y 00001100, los cuales provienen de tipear los números enteros decimales −111 ≡ 10010001 y 12 = 00001100, como puede verificarse. El resultado en memoria quedará como 10000101, el cual luego de ser procesado en los niveles 4 y 5 será visualizable como −123, acorde con.la operación (−111) − (12)

N.12 Formato, rango y precisión de tipos de datos que puede operar la UAL o el coprocesador del 486

Un procesador 486 puede operar en la UAL o en el coprocesador matemático (FPU), según sea, los tipos de datos que se indican en el cuadro de la hoja siguiente[1]. En el mismo también aparece el rango de valores representables y la precisión (cantidad de dígitos binarios o decimales que se pueden apreciar).

Los tipos de datos pueden ser[2] :

- *No signados* (naturales) de 8, 16 y 32 bits en la UAL.
- *Signados* (con bit de signo) de 8, 16, 32 bits en la UAL y 16, 32 y 64 bits en el coprocesador[3]
- *BCD empaquetados* (8 bits = 2 dígitos en la UAL u 80 bits = 18 dígitos en el coprocesador) o *desempaquetado* (1 dígito)
- Reales *en punto flotante* de 32 bits (simple precisión),. 64 bits (doble precisión) y 80 bits (precisión extendida)
- *Strings*, o sea una cadena de bits, bytes, words o Dwords contiguos. Una cadena de bits puede tener desde 8 bits hasta 4 Gigabits de longitud.

En los formatos que aparecen a continuación se ha querido representar –mediante un pequeño rectángulo en el extremo izquierdo– el bit reservado para el signo.

Para los formatos de punto flotante se ha indicado en grisado los bits reservados para el exponente más el exceso. Dentro de cada campo de un formato aparece el número de bits de su longitud.

El formato mayor de operación que admite el coprocesador es de 80 bits (bit 0 al 79) habiéndose dividido en diez bytes. Esta división en bytes también es empleada para las restantes representaciones.

Las convenciones de cada representación ya fueron tratadas en secciones anteriores.

Quizás puede ser más conveniente expresar la precisión de las representaciones en base diez, teniendo presente que 3,5 bits corresponden a un dígito decimal (sección N.1), por lo cual, por ejemplo en punto flotante, precisión simple, o sea con 23 bits de precisión más un uno que siempre existe pero no se escribe, hacen 24 bits, que divididos por 3,5 resultan casi 7 dígitos decimales de precisión. Vale decir, que aunque se tipee un número de por ejemplo 12 dígitos decimales ($10^{38}{}_{D}$ es el máximo representable, en correspondencia con hasta 38 dígitos tipeados) sólo podrán apreciarse los primeros 7 dígitos más significativos de dicho número.

---

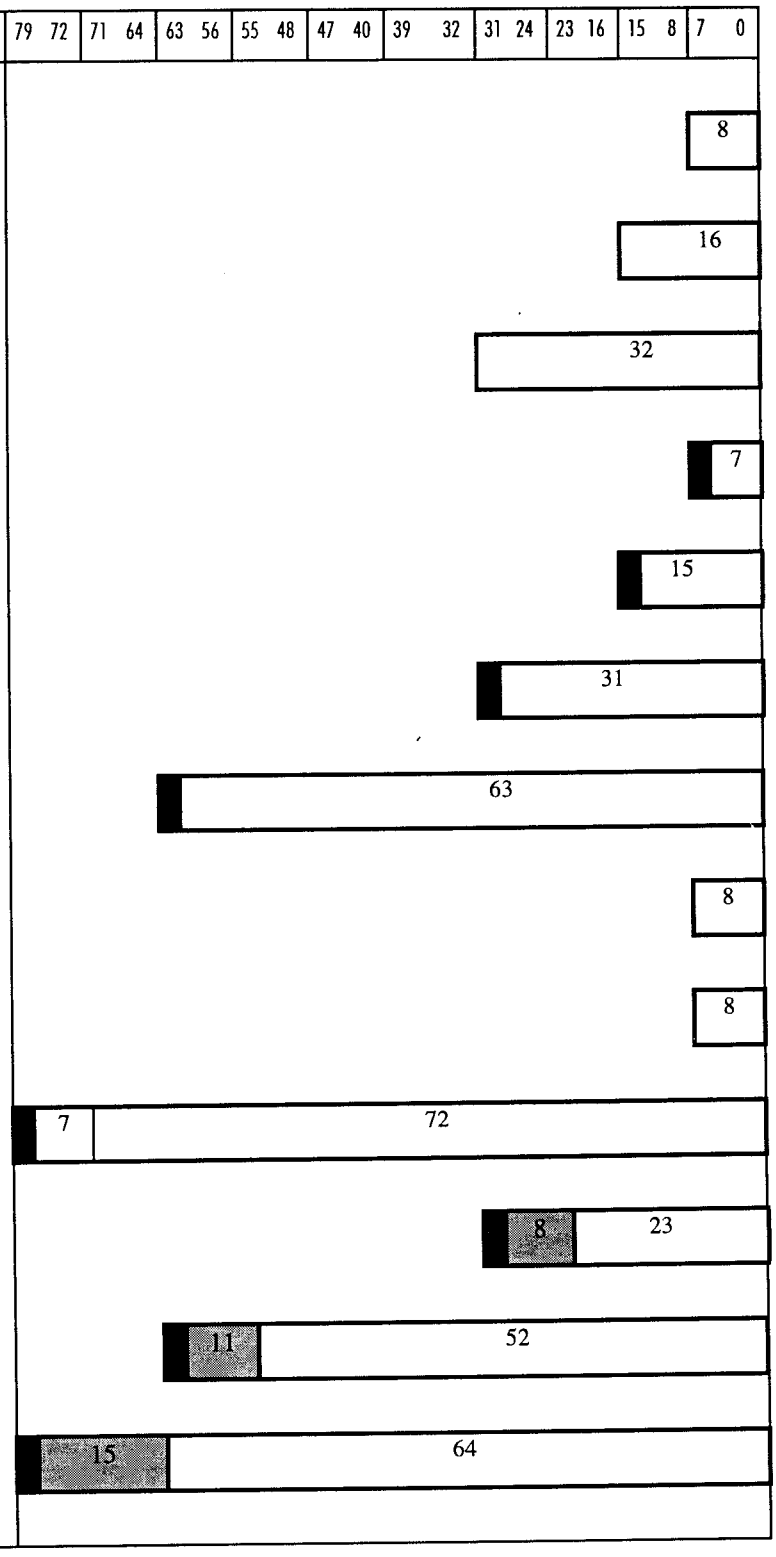
1 Esta información fue tomada del Manual Microprocessors, Vol I, 1993, de Intel™

2 Se debe tener presente que Intel designa los 16 bits como "word", y los 32 bits como "doble word (Dword)", siendo que conforme a la defínción dada en la Unidad 1, los 32 bits con que puede operar la UAL del 386 o del 486 implican un word de 32 bits, que Intel llama "Dword"

3 En un 586 (Pentium ™) puede hacerse en la UAL, pues ésta puede operar con 64 bits.

CUADRO DE FORMATOS NUMERICOS EN QUE OPERA EL 486 Y EL Pentium

Formato del dato	Rango	Precisión
No signado de 8 bits	0 - 255	8 bits
No signado de 16 bits	0 - 65535	16 bits
No signado de 32 bits	0 - $2x10^9$	32 bits
Signado de 8 bits	±127	7 bits
Signado de 16 bits	±32767	15 bits
Signado de 32 bits	$\pm 10^9$	31 bits
Signado de 64 bits	$\pm 10^{19}$	63 bits
Empaquetado 8 bits cada 2 dígitos	00 - 99	2dígitos
No empaquetado 8 bits por dígito	0 - 9	1 dígito
Empaquetado de 80 bits	$\pm 10^{18}$	18 dig.
Punto Flotante simple precisión	$\pm 10^{\pm 38}$	23 bits
Punto Flotante doble precisión	$\pm 10^{\pm 308}$	52 bits
Punto Flotante precisión extendida	$\pm 10^{\pm 4832}$	64 bits



EJERCICIO INTEGRADOR DE CONOCIMIENTOS (INTEGERS Y REALES)

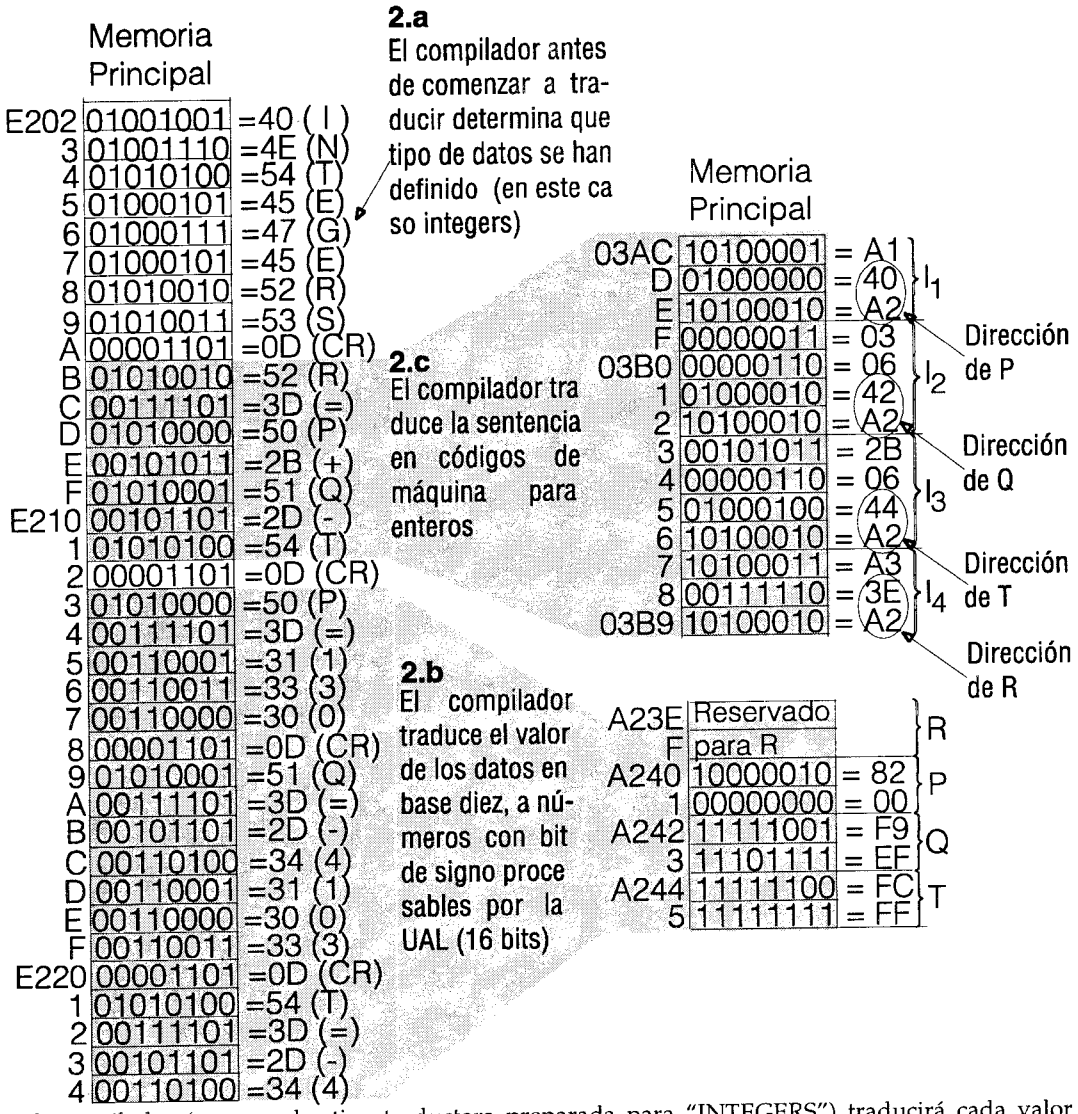
Un programador ha desarrollado para variables que son *integers*, un programa en un cierto lenguaje de alto nivel "X", cuyas sentencias tipeó desde el teclado de un computador. Para una porción del programa ha tipeado la sentencia que más abajo se indica. Se supone que luego de ella se puede escribir el valor de las variables que se ordena operar.

Variable:
INTEGERS ↵ (este símbolo ↵ aparece para indicar que se pulsó "Enter")
R = P + Q – T ↵
R = 130 Q = –4103 T = –4 ↵
↑ ↑ (las flechas señalan que se hizo un espacio mediante la barra espaciadora)

1. Representar cómo quedan en memoria los caracteres tipeados, a partir de la dirección E200.
2. Si posteriormente el programador llama al programa compilador del lenguaje "X", indicar cómo el compilador deja traducida en memoria la sentencia R = P + Q – T en código de máquina, para un procesador de Intel. Para tal fin usar los códigos usados en el ejercicio integrador para magnitudes en la unidad 1 (**puesto que los enteros se suman y restan como si fueran naturales),** y escribirlos en memoria a partir de la dirección 03AC. Asignar a las variables R, P, Q y T las direcciones A23E, A240, A242 y A244 respectivamente.
3. Una vez que la sentencia R = P + Q – T fue traducida a código de máquina, se ordena su ejecución. Indicar, en el supuesto que se ejecute la secuencia traducida en el punto anterior, cómo la UAL lleva a cabos las sumas y restas que las instrucciones correspondientes ordenan, los valores de los flags SZVC generados, y cómo queda el registro AX luego que se ejecuta cada una de las instrucciones. También pasar a base diez cada resultado binario que está en AX, verificando que sea el valor esperado.
4. Representar luego de ejecutar la instrucción I_4, cómo queda en memoria el resultado asignado a la variable R,conforme ordena dicha instrucción.
5. Suponiendo que luego de la sentencia R = P + Q – T siga la sentencia **PRINT "R=" R** (o sea imprimir R = valor hallado de R en base diez), el compilador la traducirá a varias instrucciones en código de máquina, *una de las cuales llamará a una subrutina de impresión*. Si ésta ordena que la impresora opere en modo texto, indicar cómo debe quedar codificada en memoria desde la dirección 7100, la información a enviar a la imporesora, para que imprima en decimal el valor de R (La subrutina deberá pasar el valor de R de binario con bit de signo a base diez (cada cifra codificada en binario ASCII)..

1) Como se estableció en A1.5 (Unidad 1), al tratar el código ASCII, en una operación de entrada desde el teclado, los sucesivos caracteres tipeados del programa en alto nivel quedan almacenados en binario, en posiciones sucesivas de memoria (ver figura siguiente), en este caso desde la dirección E200, codificados en ASCII. Para mayor claridad, al lado de cada de cada casillero aparece el carácter codificado, y su código ASCII en hexa, según la tabla ASCII.

2) Cualquier programa en alto nivel se encuentra codificado en ASCII (se guarda en archivos tipo .TEXT), y al igual que cualquier texto es una sucesión de caracteres aislados, codificados individualmente en combinaciones de 8 bits que los representan.
Si bien el código ASCII es un código binario, la UCP no puede ejecutar ningún programa en lenguaje de alto nivel, cualquiera sea este lenguaje, del mismo modo que la UCP no puede ejecutar una carta o una nota que hayamos escrito, y que haya quedado en memoria codificada en ASCII. **El único que entiende ASCII es el programa compilador.**
Un programa en alto nivel *es un texto* codificado en ASCII, compuesto por órdenes (sentencias) y datos,que debe ser escrito siguiendo estrictamente una diagramación específicada. Esto debe ser a los efectos de que la misma pueda ser reconocida como un programa con sus datos por el programa traductor (compilador). El se encarga de *traducir cada sentencia en una secuencia de instrucciones de máquina* apropiadas al microprocesador que lo va a ejecutar.
Cuando se llama al compilador del lenguaje "X", éste pasa a memoria, desde el archivo del disco rígido donde se encuentra. Al ordenar compilar, para el programa compilador los datos a procesar son los caracteres codificados en binario ASCII, que representan el programa en alto nivel. Luego de ser ejecutado (por la UCP), el progrma compilador dejará en memoria traducido a código de máquina (de Intel en este caso), el programa que originariamente estaba en alto nivel. Esto es, el compilador es un programa cuyo "input"(datos) es un programa en alto nivel codificados en ASCII, y cuyo "output" (resultado) es el programa que fue su "input", pero en código de máquina, para ser ejecutado.
El compilador para "X" es un programa "inteligente" que recorre repetidamente la zona de memoria donde quedó codificado en ASCII el programa en lenguaje "X". Así va identificando cada sentencia y los datos (variables) que ella ordena operar, a fin de traducirlos en instrucciones (códigos) de máquina que el microprocesador pueda ejecutar, y códigos de datos que la UAL pueda operar
Con estos fines, **primero (2a** en la figura) debe determinar el tipo de variables (en este caso integers) que las sentencias ordenan procesar, para llevar a cabo las traducciones a instrucciones para procesar enteros, y códigos de datos correspondientes a éstas. Entonces, el compilador antes de traducir leerá en memoria que en ASCII está escrito "INTEGERS".
Segundo (2b) el compilador reservará dos bytes de memoria para las variables R, P, Q y T, (si fueran "long integers" reservaría 4 bytes) en direcciones de memoria que van de A23E hasta A245. Esto es, para el compilador la variable R será la dirección A23E y la siguiente; la variable P la A240 y la siguiente, etc., bastando la primer dirección de una variable para identificarla.



El compilador (o una subrutina traductora preparada para "INTEGERS") traducirá cada valor de variable decimal escrita en ASCII en el número binario natural de 16 bits equivalente.

Así, el valor decimal 130 de la variable P que en memoria está en ASCII como 00110001 00110011 00110000 ocupando las posiciones E215, E216 y E217 (ver figura) quedará traducida como el valor binario con bit de signo 0000000100000010_B = 130_D que ocupará las posiciones A240 y A241 asignadas a la variable P. El valor decimal –4103 de Q que el compilador encuentra en binario ASCII en las posiciones E21B a E21E (4 bytes). las traducirá como el binario con bit de signo EFF9 = 1110111111111001_B = -4103_D, que ocupará las posiciones A242 y A243. Del mismo modo, el valor –4 de T que al compilador llega como 00101101 00110100 en ASCII, ocupando un byte, pasará a ser 1111111111111100_B = -4_D que el compilador escribirá en las direcciones A244 y A245. Para la variable R, cuyo valor recién se conocerá cuando se ejecute el programa que se está traduciendo, *siendo que el programa que en este paso se ejecuta es el compilador* (en lugar del cual nos ponemos), éste sólo le reserva la posición A23E y la siguiente (como supone el enunciado). De esta forma cada variable, cuyo valor al ser tipeado ocupa tantos bytes de memoria como dígitos decimales tenga, al ser codificada como magnitud binaria ocupa 2 bytes, siendo que el máximo número que se puede representar en 16 bits es el 32767_D = 0111111111111111_B

Una vez asignada a cada variable una dirección de localización y la siguiente, el compilador traducirá la sentencia R = P + Q – T en instrucciones cuyos códigos de máquina son los mismos que el ejercicio integrador de la Unidad 1 para magnitudes. En decimal sería R = 130 + (–4103) – (–4) **sin poder realizar** más por menos igual a menos, ni menos por menos igual a más.

Recordar que con una calculadora de bolsillo, y trabajando con dígito de signo (sección N1) haríamos lo siguiente. Primero entramos 00130 al visor. Segundo le sumamos 95807 = –4103 + 100000, obteniendo 96027. Tercero: a este valor le restamos –4 + 100000 = 99996. Para hacer esta resta sumamos el complemento de 99996, que es 4, resultando R = 96027. Este es un número negativo, por empezar con 9. Su magnitud se calcula haciendo 100000 – 96027. Resulta que el número es –3973, igual al resultado de hacer la cuenta indicada. Cuarto: se supone que 96027 se guarda en la memoria de la calculadora, mediante la tecla correspondiente. Como se describió oportunamente, estos 4 pasos en un microprocesador

se realizan mediante las instrucciones I_1 a I_4, cuyos códigos de máquina son los mismos del ejercicio integrador para magnitudes de la Unidad 1.
Ex profeso se han mantenido las direcciones de las variables como en dicho ejercicio, para recalcar que magnitudes ("unsigned") y enteros se suman y restan con los mismos códigos de instrucción.
En definitiva, el compilador escribe una zona de datos a procesar codificados como binarios con bit de signo (procesables por la UAL), y otra zona con el programa codificado en bajo nivel (instrucciones en código de máquina), que puede ser ejecutado por la UC, **que es la unica que entiende los códigos binarios de máquina**.
Obsérvese que los códigos de máquina para sumar y restar son distintos que los códigos ASCII del + y del –

3) Cuando se ordene ejecutar el programa que el compilador dejó en código de máquina en memoria, y la UC ejecute las instrucciones I_1 a I_4, luego de cada una de ellas, el registro AX quedará como se indica a continuación. La UAL (**que es la única que entiene binario natural**) sólo operará en las instrucciones I_2 e I_3, que ordenan sumar y restar, respectivamente, indicándose para las mismas la operación de la UAL.
Después de ejecutar I_1:
AX = 0000000010000010 = 0082 que es el valor de P (130 en decimal)

Después de ejecutar I_2:
AX = 1111000001111011 = F07B que es P + Q (–3973 en decimal).
Este valor es el resultado de sumar en la UAL ⟶ 0000000010000010 (P)
Los indicadores que genera la UAL son S=1 Z=0 V=0 C=0 +1110111111111001 (Q)
1111000001111011 = –3973 = (P + Q)

Después de ejecutar I_3 que ordena restar:
AX = 1111000001111111 = F07F que es P + Q – T = R (–3969 en decimal).
Este valor es el resultado de sumar en la UAL ⟶ 1111000001111011 (P + Q)
Los indicadores que genera la UAL son S=1 Z=0 V=0 C=1(resta) +0000000000000100 (Compl de T)
1111000001111111 (P + Q) – T = R

0000000000000100 es el complemento de 1111111111111100 = –4
se halla invirtiendo este último número y sumando uno: 0000000000000011 + 1 = 0000000000000100

Obsérvese que al representar –4 en el paso 2. el compilador (software) complementó el +4 binario, y en el paso 3 la UAL (hardware), para restar siempre complementa el sustraendo, por lo que el –4 antes representado en memoria, se vuelve a complementar otra vez, quedando nuevamente +4. En definitiva aritméticamente resulto que –(–4) = +4, aunque, como ordenó la sentencia, la variable T debe ser restada, sin importar que su valor sea negativo o positivo, y eso fue lo que se hizo en el proceso tratado.

Después de ejecutar I_4 el valor de AX no cambiará, pues I_4 ordena que una copia de AX pase a memoria
AX = 1111000001111111 = F07F

4) La ejecución de I_4 ordena que en la dirección A23E y en la siguiente (asignadas a R) copiar el valor de AX, por lo que luego de ejecutarse I_4 en memoria se tendrá lo indicado en la figura de abajo a la izquierda

5) En modo texto, la subrutina de impresión dejará en posiciones sucesivas de memoria los caracteres a imprimir, cada uno codificado en ASCII. Previamente dicha subrutina interpretará la orden de impresión (en este caso R = valor de R), siendo que el valor de R que está en A23E y A23F deberá pasarlo a dígitos decimales codificados en ASCII. Esto es, determinará que el valor de R, que es 1111000001111111 es el –3969, el cual en ASCII resultará 00111101 00110011 00111001 00110110 00111001, como aparece en memoria.
Antes de estos códigos aparecen los códigos de R y de =, como se indica en la figura de abajo a la derecha

6) La ejecución del programa mediante el Debug, para verificar que los resultados y flags obtenidos en el punto 3) coinciden con la ejecución por un procesador de cada una de las 4 instrucciones, se realiza siguiendo los mismos pasos descriptos en el capítulo 1.6 de la unidad 1.

A23E	7F
A23F	10

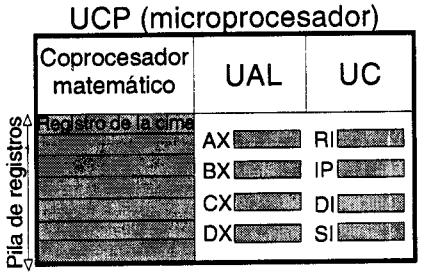
2000	52	(R)
1	3D	(=)
2	2D	(-)
3	33	(3)
4	39	(9)
5	36	(6)
6	39	(9)

Un programador ha desarrollado para variables que son ***reales***, un programa en un cierto lenguaje de alto nivel "X", cuyas sentenciasha tipeado desde el teclado de un computador. Para una porción del programa ha tipeado la sentencia que más abajo se indica. Se supone que luego de ella, se puede escribir el valor de las variables que dicha sentencia ordena operar.

```
Variable:
REALES ↵
R = P + Q – T ↵
P = –130,5;  Q = 4103,75;  T = 4 ↵
      ↑  ↑        (las flechas indican que se hizo un espacio mediante la barra espaciadora)
```

1. Indicar cómo quedan en memoria los caracteres tipeados, a partir de una dirección dada
2. Si posteriormente el programador llama al programa compilador del lenguaje "X", indicar cómo el compilador deja traducida la sentencia R=P+Q–T en código de máquina, para un procesador de Intel, trabajando en punto flotante, y escribirlos en memoria a partir de la dirección 20D1. Asignar a las variables R, **P**, Q y T las direcciones D015, 2003, A200, y BC08, respectivamente
3. Si una vez que todo el programa en el lenguaje "X" fue traducido a código de máquina (y luego "link editado" para agregarle subrutinas), se ordena ejecutarlo ("run"), indicar cuando se ejecute la secuencia traducida en el punto anterior, cómo el coprocesador lleva a cabo las sumas y restas que las instrucciones correspondientes ordenan, y cómo queda la cima de la pila que el coprocesador opera, luego de que se ejecuta cada una de las instrucciones de dicha secuencia. También pasar a base diez cada resultado binario que está en la cima de la pila, verificando que sea el valor esperado
4. Indicar luego de ejecutar la instrucción $\mathbf{I_4}$ cómo queda en memoria el resultado asignado a la variable **R**, conforme ordena dicha instrucción.
5. Suponiendo que luego de la sentencia R = P + Q – T siga la sentencia **PRINT "R=" R** (o sea imprimir R = valor hallado de R en decimal), el compilador la traducirá a varias instrucciones en código de máquina, una de las cuales llamará a una subrutina de impresión. Si ésta ordena que la impresora opere en modo texto, indicar cómo debe quedar codificada en memoria la información a enviar a la impresora, para que imprima en decimal el valor de R.
6. Ejecutar mediante el Debug la secuencia de instrucciones determinadas en el punto 2, verificando que los resultados esperados coincidan con los calculados en el punto 3.

1) No hay grandes diferencias con la concreción de este punto en el ejercicio anterior para enteros.
2) El compilador para el lenguaje "X" recorre la zona de memoria donde quedó codificado en ASCII el programa en lenguaje "X". Así va identificando cada sentencia y los datos que ella ordena operar, a fin de traducirlos en instrucciones (códigos) de máquina que el microprocesador pueda ejecutar, y códigos de datos que el coprocesador pueda operar.
Con estos fines, **primero** (**2a** en la figura de la hoja siguiente) debe determinar el tipo de variables (en este caso reales) que las sentencias ordenan procesar, para llevar a cabo las traducciones a instrucciones para procesar punto flotante, y códigos de datos correspondientes a éstas. Entonces, el compilador antes de traducir leerá en memoria que en ASCII está escrito "REALES".
Segundo (**2b**) el compilador reservará 4 bytes de memoria para las variables R, P, Q y T, (si fueran en doble precisión" reservaría 8 bytes) en las direcciones de memoria indicadas. El compilador (o una subrutina traductora preparada para "REALES") traducirá cada valor de variable decimal escrita en ASCII en el número en punto flotante de 32 bits equivalente.
Así, el valor decimal –1305 de la variable P que en memoria está en ASCII quedará traducida como el valor en punto flotante $11000011000000101000000000000000 = C3028000 = -130{,}5_D$ que ocupará las 4 posiciones asignadas a la variable P. El valor decimal –4103,75 de Q que el compilador encuentra en binario ASCII en las las traducirá como el número en punto flotante $11000101000000000111111000000000 = C5803E00 = -4103{,}75_D$, que ocupará las cuatro posiciones asignadas a la variable Q. Del mismo modo, el valor 4 de T que al compilador llega como 00110100 en ASCII, ocupando un byte, pasará a ser $01000000100000000000000000000000_B = 40800000 = 4_D$ que el compilador escribirá en las cuatro direcciones asignadas a T. Para la variable R, cuyo valor recién se conocerá cuando se ejecute el programa que se está traduciendo, el compilador le reserva las 4 posiciones indicadas.
Una vez asignada a cada variable una dirección de localización y las tres siguientes, el compilador traducirá la sentencia R = P + Q – T en instrucciones cuyos códigos de máquina son los indicados.(**2c** en la figura de la hoja siguiente)
Tales códigos suponen, que el coprocesador matemático hoy integrado al microprocesador (ver dibujo siguiente), está ligado a una pila de registros sin denominación individual.

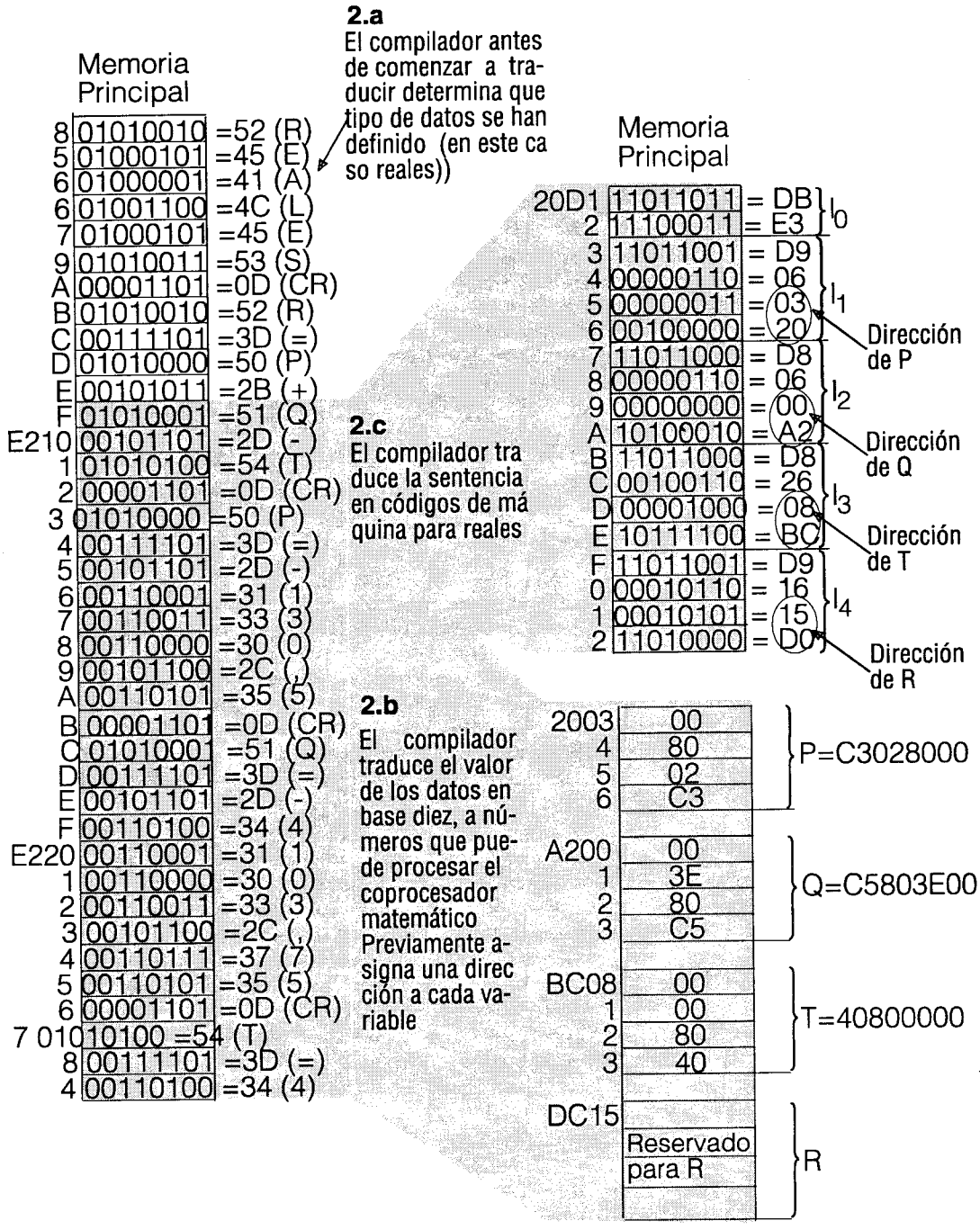


Operaremos, de la misma forma que lo hicimos con AX en el ejercicio anterior, con el registro que está en la cima de la pila. La instrucción $\mathbf{I_1}$ (código D906) ordenará llevar el dato P que está en 2003 a la cima de la pila. $\mathbf{I_2}$ (código D806) ordena sumar contra la cima el dato Q que está en A200, y el resultado guardarlo en la cima. $\mathbf{I_3}$ (código D826) ordena restar contra la cima el dato T que está en BC08. $\mathbf{I_4}$ (código D916) ordena enviar una copia de la cima hacia D015, donde está R en memoria
La instrucción $\mathbf{I_0}$ (código DBE3) habilita el funcionamiento del coprocesador, **único que entiende binario que codifica FP**.

3) Cuando se ordene ejecutar el programa que el compilador dejó en código de máquina en memoria, y la UC ejecute las instrucciones $\mathbf{I_1}$ a $\mathbf{I_4}$, luego de cada una de ellas, la cima de la pila quedará como se indica a continuación. El coprocesador operará las instrucciones $\mathbf{I_2}$ e $\mathbf{I_3}$, que ordenan sumar y restar, respectivamente, de la forma que se indica más abajo.

Después de ejecutar $\mathbf{I_1}$:
Cima = 11000011000000101000000000000000 = C3028000 = $-130{,}5_D$

Después de ejecutar $\mathbf{I_2}$:
Cima = 11000101100001000101001000000000 = C5845200, que es P + Q = $-4234{,}25_D$



Este valor es el resultado de sumar en el coprocesador como sigue, conforme se vió en la sección N11, para sumar números en FP (punto c)

P = 11000011000000101000000000000000 = $1{,}00000101 \times 10^{111}$ = $0{,}0000100000101 \times 10^{1100}$

Q = 11000101100000000001111000000000 = $+\ 1{,}00000000011111 \times 10^{1100}$

$1{,}000010001010001 \times 10^{1100}$

$1{,}000010001010001 \times 10^{1100}$ = 11000101100001000101001000000000 = C5845200

Después de ejecutar $\mathbf{I_3}$
Cima = 11000101100001000111001000000000 = C5847200, que es P + Q −T = $-4238{,}25_D$

El coprocesador procede para la resta, de manera semejante a la suma, primero igualando los exponentes y luego restando
Después de ejecutar $\mathbf{I_4}$ el valor de la cima no cambiará, pues $\mathbf{I_4}$ ordena que una copia de la cima pase a memoria
Cima = 11000101100001000111001000000000 = C5847200

4) Luego de la ejecución de $\mathbf{I_4}$ ordena que en la dirección D015 y en las tres siguiente (asignadas a R) copiar el valor de la cima por lo que luego de ejecutarse $\mathbf{I_4}$ en memoria se tendrá como indica la figura de más abajo a la izquierda

5) En modo texto, la subrutina de impresión dejará en posiciones sucesivas de memoria los caracteres a imprimir, codificados en ASCII. Previamente dicha subrutina interpretará la orden de impresión (en este caso R = valor de R), siendo que el valor de R que está en D015 y siguientes deberá pasarlo a dígitos decimales codificados en ASCII. Esto es, determinará que el valor de R, que es 11000101100001000111001000000000 es el –4238,25, el cual en ASCII resultará como se indica en la figura de abajo a la derecha
Antes de estos códigos aparecen los códigos de R y de = , como se indica en la figura siguiente a la derecha.

DC15	00
6	72
7	84
8	C5

2000	52	(R)
1	3D	(=)
2	2D	(-)
3	33	(3)
4	39	(9)
5	36	(6)
6	39	(9)

6) Como en las unidades 1 y 3 indicaremos en letra itálica los símbolos que se tipean, y con ↵ la acción de pulsar la tecla Enter. En negrita aquello que interesa ver en la UCP a los fines del ejercicio, y en menor tamaño de número o letra, el valor que puede ser disitnto de una PC a otra.

C:\> *DEBUG* ↵
–

– *E 2003* ↵
309D:2003 1F.*00* 26.*80* 7B.*02* 1F.*C3* ↵ Así se ha escrito P = C3028000 en las direcciones 2003 a 2006
En 2003 se supuso que existía 1F, 26 en 2004, etc.El valor 309D implica una dirección de referencia, que en general será XXXX

– *E A200* ↵
309D:A200 3F.*00* E6.*3E* AB.*80* FF.*C5* ↵ Así se ha escrito Q = C5803E00 en las direcciones A200 a A203

– *E BC08* ↵
309D:BC08 33.*00* EF.*00* AA.*80* 22.*40* ↵ Así se ha escrito T = C5803E00 en las direcciones BC08 a BC0B

Escritura en memoria de las 5 instrucciones a ejecutar

– *E 20D1* ↵
309D:20D1 35.*DB* 00.*E3* 00.*D9* AF.*06* 35.*03* 00.*20* 00.*D8*
309D:20D8 FF.*06* DE.*00* 68.*A2* AA.*D8* 95.*26* EE.*08* 38.*BC* 40.*D9*
309D:20E0 67.*16* 35.*15* E7.*D0* ↵

–*R IP*↵
IP 0100 En general en IP puede haber en general, XXXX
: *20D1* ↵ Así se ha escrito 20D1 en el registro IP para que cambie el valor 0100 que tenía por 20D1, que es donde se ha escrito la instrucción DBE3 en el paso anterior, como se indica en la unidad 1.

– *R* ↵
AX=0000 BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=309D ES=309D SS=309D CS=309D **IP=20D1** NV UP EI PL NZ NA PO NC
309D:**20D1 DBE3** **FINIT**

Este paso tiene como fin verificar que todo está en orden antes de ejecutar la instrucción. Como ser, que la próxima instrucción a ejecutar es DBE3

– *T* ↵ (Orden de ejecución de la instrucción DBE3)
AX=0000 BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=309D ES=309D SS=309D CS=309D **IP=20D3** NV UP EI PL NZ NA PO NC
309D:**20D3 D9060320** **FLD DWORD PTR [2003]**

– *T* ↵ (Orden de ejecución de la instrucción D9060320)
AX=0000 BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=309D ES=309D SS=309D CS=309D **IP=20D7** NV UP EI PL NZ NA PO NC
309D:**20D7 D80600A2** **FADD DWORD PTR [A200]**

– *T* ↵ (Orden de ejecución de la instrucción D800600A2)
AX=0000 BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=309D ES=309D SS=309D CS=309D **IP=20DB** NV UP EI PL NZ NA PO NC
309D:**20DB D82608BC** **FSUB DWORD PTR [BCO8]**

– *T* ↵ (Orden de ejecución de la instrucción D82608BC)
AX=0000 BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=309D ES=309D SS=309D CS=309D **IP=20DF** NV UP EI PL NZ NA PO NC
309D:**20DF D91615D0** **FST DWORD PTR [DO15]**

– *T* ↵ (Orden de ejecución de la instrucción D91615D0)
AX=0000 BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=309D ES=309D SS=309D CS=309D IP=20E3 NV UP EI PL NZ NA PO NC
Este tercer rengón no interesa

Siendo que el Debug sólo muestra los registros vinculados a la UAL (que para estas instrucciones no se usan) y a la UC, y que los registros ligados al coprocesador son "invisibles", para todas las instrucciones ejecutadas el único registro visible que ha cambiado es el IP, permaneciendo sin varias los restantes, así como los flags.

– *E D015* ↵ (Examinar las posiciones D015 a D018 correspondientes a la variable R)
309D:D015 *.72 .84 .40*
309D:D018 *.C5*

C5847200 = 1100010110000100011100100000000 = $-1{,}000010001110010000000000 \times 10^{1100}$ = –1000010001110,01 = –4238,25 Se verifica que el valor de R que está en FP es el esperado.

N.13 Problemas

1. Explicar por qué no pueden representarse en memoria números enteros binarios, o sea por qué se usa bit de signo
2. Dado el número con bit de signo 10111001 ≡ –N obtenido a partir invertir el número N y sumarle 1, indicar qué operación aritmética se evita realizar operando de esta forma. Explicar por qué no se usa restar 1 y luego invertir los bits para hallar N
3. Representar el número binario con bit de signo 10110010 en formato 16. ¿Qué conclusión resulta ?
4. ¿Cuántos bytes hacen falta para representar en binario números decimales entre –32768 y 32767 ?
5. Para un formato de 4 bytes: ¿cuál es la máxima magnitud que puede representarse, y cuál el mayor número positivo entero? Idem el número negativo de mayor magnitud.
6. Realizar en módulo 100000 las siguientes operaciones en base diez con números enteros usando dígito de signo con la convención del complemento al módulo. Verificar que los resultados con dígito de signo coincidan con los que se obtendrían con los efectuados en forma corriente.
   a) 96 + 180; b) 96 + (–180); c) –96 + 180; d) –96 + (–180)
7. Para el punto b) anterior justificar por qué al hacer una suma de naturales también se puede efectuar una suma de enteros. Tener presente los ejemplos dados en pie de páginas de la sección N.2
8. Dadas las siguientes sumas algebraicas en base diez, hacerlas en base dos para números con bit de signo, en el menor formato posible, indicando SZVC. Verificar si el resultado binario pasado a base diez concuerda con el esperado en base diez
   a) 18 + 102 b) 18 + (–102) c) –18 + 102 d) –18 + (–102).
9. Idem anterior para las sumas 30 + 102 y –30 + (–102)
   Si no resultara la verificación, explicar por qué no concuerda, y determinar en qué formato mínimo es posible realizar correctamente cada operación. Verificar que es posible realizarla en formato 16
10. Realizar en el menor formato posible sin que dé overflow la operación –534 – (–256) indicando SZVC
11. En función del problema anterior explicar por qué se afirma que el overflow es un error por truncamiento.
12. Efectuar las siguientes restas algebraicas dadas en base diez en base dos, en el menor formato posible, indicando SZVC. *No es factible hacer "menos por menos = mas", etc.* Verificar si el resultado binario convertido a decimal concuerda con el esperado haciendo las operaciones en base diez.
    a) 18 – 102, b) 18 – (–102) c) –18 .– 102 d) –18 – (–102)

13. Efectuar en formato 8 la resta entre números decimales 150 – 37 representado los mismos como binarios **naturales** (magnitudes), indicando SZVC. Verificar si el resultado binario convertido a decimal concuerda con el esperado haciendo las operaciones en base diez. ¿Se podría haber hecho esta misma operación en formato 8 suponiendo que se opere con números signados ?

14. Si con los mismos números se intentara realizar 37 – 150 (operación prohibida entre naturales), indicar una forma de conocer la magnitud de la diferencia. ¿ Se podría haber hecho las operaciones en formato 8 si los números hubiesen sido signados?

15. Para el problema 8, justificar para el caso d) por qué los binarios con bit de signo se pueden sumar como si fueran naturales (magnitudes), conforme aparece en la sección N.2

16. Explicar la diferencia entre los indicadores **V** y **C**. Ejemplificar numéricamente para bases diez y dos. Indicar para qué tipos de números se usa cada uno.

17. Se tiene un apellido que empieza con D y otro que empieza con R, en ese orden. Indicar como haría un computador para determinar en función de SZVC, si están en orden o no.

18. Dado el número 10000000000 en base dos, expresarlo como 10^n en esa base, determinando **n**.
El mismo número en base dieciseis, expresarlo como 10^n en esa base, determinando **n**.
Idem en base diez, expresarlo como 10^n en esa base, determinando **n**.

19. Convertir a binario el número en base diez 2362 usando el procedimiento indicado en el apéndice A2, sección A2.3

20. El número binario 10000000000, expresarlo como 2^k en base diez, determinando **k**

21. Indicar qué número decimal es el binario 11000110,101101

22. Convertir a hexadecimal el número binario 110111001,001. conforme a las reglas de la sección A2.8 del apéndice A2

23. Convertir a base dos el número $83{,}1_D$. El número binario obtenido volverlo a base diez, apreciando seis bits fraccionarios. ¿Qué conclusiones resultan ?

24. Convertir a punto flotante, convención IEEE, los números decimales 0,00375 ; 375 y –1925

25. Convertir a decimal el número 00111110000000000000000000000000 que está en punto flotante, convención IEEE

26. Convertir al código BCD natural el número 11000010010001_B y luego expresar el número también en código ASCII

27. Qué número decimal es la combinación 10010010 considerada como:
a) Un número con bit de signo b) Un número binario natural c) Un número en BCD

28. Indicar si las siguientes igualdades son correctas. Corregir cómo debe ser en caso contrario:
$100101110001_B = 971_H = 971_D = 100101110001_{BCD}$ $000100000000_B = 001001010110_{BCD}$

29. Dado el número 001000000011 en BCD, convertirlo a hexadecimal. y binario

30. Suponiendo un lenguaje de programación que admita definir variables en decimal empaquetado, indicar cómo serían los pasos del ejercicio integrador 1 de la sección N.11

31. En función de los ejercicios anteriores y de los temas tratados en esta unidad, verificar el cuadro siguiente:

CUADRO DE CORRESPONDENCIAS ENTRE VARIABLES DEFINIDAS EN ALTO NIVEL Y SU REPRESENTACION EN MEMORIA PRINCIPAL

TIPO DE VARIABLE DEFINIDA EN ALTO NIVEL	***Enteros ("integers")***	***Reales***	***Empaquetado***	***Caracteres ("strings")***	***Booleanas***
Por ejemplo si el valor de la variable tipeada es:	–25 ↓	–25 ↓	–25 ↓	Sol ↓	V F F F V V F V
Luego de ser tipeada en memoria quedará almacenada en ASCII como:	00101101 (–) 00110010 (2) 00110101 (5) ↓	00101101 (–) 00110010 (2) 00110101 (5) ↓	00101101 (–) 00110010 (2) 00110101 (5) ↓	01010011 (S) 01101111 (o) 01101100 (l)	
Un programa traductor los pasará de ASCII a la representación indicada, quedando en memoria:	11111111 11100111 (formato 16)	11000001 11001000 00000000 00000000	00000010 01011011	01010011 01101111 01101100	10001101
	Números con bit de signo	***Números en punto flotante***	***BCD empaquetado***	***Caracteres en ASCII***	***V=1 F=0***

32. Realizar los ejercicios integradores de conocimientos para integers y reales con valores a elección.

GENERALIZACIONES-JUSTIFICACIONES DE PROCEDIMIENTOS UTILIZADOS

A2.1 Expresión de un número en potencias de su base

Base diez:

Los dígitos de un número son, matemáticamente, los coeficientes que multiplican a las correspondientes potencias de diez en la expresión que es la suma de dichos productos. Habíamos escrito:

$N = 107 = (\mathbf{1}x10^2 + \mathbf{0}x10^1 + \mathbf{7})_D$

Un número de tres dígitos decimales, designados $\mathbf{d_2,d_1,d_0}$[1] (cada uno puede valer 0 a 9) tendrá la forma:

$N = \mathbf{d_2d_1d_0} = \mathbf{d_2}x10^2 + \mathbf{d_1}x10^1 + \mathbf{d_0}$

siendo para 107: $d_2 = 1$ $d_1 = 0$ $d_0 = 7$

Generalizando, cualquier número decimal natural tendrá por expresión

$$\therefore \quad \mathbf{N = d_kd_{k-1} \ldots d_2d_1d_0 = d_k}x10^{\mathbf{k}} + \mathbf{d_{k-1}}x10^{\mathbf{k-1}} + \ldots + \mathbf{d_2}x10^{\mathbf{2}} + \mathbf{d_1}x10^{\mathbf{1}} + \mathbf{d_0} \qquad \textbf{(A2.1)}$$

y en forma más compacta $\mathbf{N} = \sum_{i=0}^{i=k} \mathbf{d_i}x10^{\mathbf{i}}{}_D$ con $d_i = \{0,1,2, \ldots ,9\}$

Base dos

El mismo número 107_D en binario es $N = 1101011_B$, que representa la sumatoria:

$N = (\mathbf{1}x1000000 + \mathbf{1}x100000 + \mathbf{0}x10000 + \mathbf{1}x1000 + \mathbf{0}x100 + \mathbf{1}x10 + \mathbf{1})_B$

Conforme a la sección N.7, puede escribirse

$N = (\mathbf{1}x10^{110} + \mathbf{1}x10^{101} + \mathbf{0}x10^{100} + \mathbf{1}x10^{11} + \mathbf{0}x10^{10} + \mathbf{1}x10^{1} + \mathbf{1})_B$ siendo $10_B = 2_D$

Cualquier número de siete dígitos binarios (cada bit de valor **1** ó **0**) será de la forma:

$N = \mathbf{b_{110}b_{101}b_{100}b_{11}b_{10}b_1b_0} = \mathbf{b_{110}}x10^{110} + \mathbf{b_{101}}x10^{101} + \mathbf{b_{100}}x10^{100} + \mathbf{b_{11}}x10^{11} + \mathbf{b_{10}}x10^{10} + \mathbf{b_1}x10^{1} + \mathbf{b_0}$

En el 101011_B es $b_0 = 1$ $b_1 = 1$ $b_2 = 0$ etc.

Generalizando, cualquier número binario natural puede expresarse como:

$$\mathbf{N = b_nb_{n-1} \ldots b_{11}b_{10}b_1b_0 = b_n}x10^{\mathbf{n}} + \mathbf{b_{n-1}}x10^{\mathbf{n-1}} + \ldots + \mathbf{b_{11}}x10^{\mathbf{11}} + \mathbf{b_{10}}x10^{\mathbf{10}} + \mathbf{b_1}x10^{\mathbf{1}} + \mathbf{b_0} \qquad \textbf{(A2.2)}$$

y en forma más compacta $\mathbf{N} = \sum_{i=0}^{i=n} \mathbf{b_i}x10^{\mathbf{i}}{}_B$ con $b_i = \{0,1\}$, siendo $10_B = 2_D$

Expresión general para cualquier base R

Realizando una generalización mayor, un número natural en cualquier base puede expresarse

$$\mathbf{N} = \sum_{i=0}^{i=p} \mathbf{r_i}x10^{\mathbf{i}}{}_R$$

[1] En lo que sigue, los coeficientes $\mathbf{d_i}$ identificarán a números decimales

cada coeficiente (dígito) $\mathbf{r_i}$ puede tomar tantos valores como símbolos presente la base en cuestión, a partir del valor cero y en sucesión.

De esta forma pueden expresarse los números mediante sumatorias de productos, cuyos factores y potencias están representados en la base de que se trate.

A2.2 Expresión en base diez de un número a partir de su expresión en potencias de otra base

Es común escribir, por comodidad, un número dado en una base **R** como una sumatoria de productos cuyos factores y potencias están en base diez . Esto ya se hizo de hecho en el pasaje de un número que está en una base cualquiera a base diez, cuando se cuantificaba en esta base el valor de los pesos (subgrupos) en que se dividía un conjunto. Así:

$1101011_B = (64 + 32 + 8 + 2 + 1)_D = 107_D$

$1101011_B = (1x64 + 1x32 + 0x16 + 1x8 + 0x4 + 1x2 + 1)_D = 107_D$

$= (1x2^6 + 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1)_D$

Esta equivalencia también podría haberse obtenido reemplazando en la expresión **(A2.2)**, que está en base dos, cada factor y potencia por su equivalente decimal.

En general, la equivalencia en base diez de un número binario podrá expresarse:

$$\mathbf{N = (b_n b_{n-1} \ldots b_{11} b_{10} b_1 b_0)_B = (b_n x 2^n + b_{n-1} x 2^{n-1} + \ldots + b_3 x 2^3 + b_2 x 2^2 + b_1 x 2^1 + b_0)_D = \sum_{i=0}^{i=n} b_i x 2^i \; _D} \qquad \textbf{(A2.4)}$$

Los correspondientes coeficientes $\mathbf{b_i}$ tienen igual valor (0 ó 1) en ambas expresiones, siendo que los símbolos 0 y 1 tienen igual significado en base dos y diez.

Generalizando, un número natural en cualquier base **R**, puede expresar en base diez como: $\mathbf{N = \sum_{i=0}^{i=p} r_i x R^i \; _D}$ **(A2.6)**

siendo el coeficiente $\mathbf{r_i}$ el equivalente decimal del correspondiente dígito del número en base **R**, y el símbolo **R** es la base de que se trata expresada en base diez.

Esta expresión permite hallar el número decimal equivalente de otro dado en cualquier base **R**, efectuando los cálculos en base diez (a los que estamos acostumbrados), de la forma vista.

A2.3 Pasaje de base diez a otra operando en la segunda

Si bien los pasaje inversos de base diez a otra cualquiera **R** se realizan manualmente –por comodidad y rapidez– mediante sucesivas divisiones por **R** (Unidad 1), también es factible efectuarlos realizando operaciones en la otra base. Para tal fin se debe:

1. Expresar el número decimal como suma de potencias de diez.
2. Reemplazar cada factor o potencia decimal por su equivalente en la otra base.
3. Realizar en esta última las operaciones necesarias.

þ **EJEMPLO**: Convertir a binario el número decimal 168

$168_D = (1x10^2 + 6x10^1 + 8)_D = (1x1010^{10} + 110x1010^1 + 1000)_B =$

$= (1100100 + 111100 + 1000)_B = 10101000_B$

A2.4 Justificación del método de las divisiones para pasar de base diez a otra

Conforme a las expresiones **(A2.1)** y **(A2.4)** un mismo número **N** expresado en decimal y binario, deberá ser

:

$N = d_k d_{k-1} \ldots d_2 d_1 d_0)_D = b_n b_{n-1} \ldots b_1 b_0)_B = b_n x 2^n + b_{n-1} x 2^{n-1} + \ldots + b_3 x 2^3 + b_2 x 2^2 + b_1 x 2^1 + b_0)_D$

Se trata de determinar el valor 0 ó 1 de las incógnitas $\mathbf{b_n b_{n-1} \ldots b_1 b_0}$ y cuántas componen el número binario buscado (valor de **n**) en función del número decimal conocido. Por

$N = 12_D = (\mathbf{b_n} x 2^n + \ldots + \mathbf{b_5} x 2^5 + \mathbf{b_4} x 2^4 + \mathbf{b_3} x 2^3$ Sacando factor común 2:

$N = 12_D = [(b_n x 2^{n-1} + ... + b_5 x 2^4 + b_4 x 2^3 + b_3 x 2^2 + b_2 x 2^1 + b_1)x2 + b_0]_D = Q_1 x 2 + b_0$

entonces, la incógnita b_0 es el esta de dividir 12/2, siendo Q_1 el cociente:

$N = 12 \mid 2$
$b_0 = 0 \quad 6_D = Q_1 = (b_n x 2^{n-1} + ... + b_5 x 2^4 + b_4 x 2^3 + b_3 x 2^2 + b_2 x 2^1 + b_1)_D$ Luego $b_0=0$

Sacando nuevamente factor común 2 en la expresión de Q_1:

$6 = Q_1 = [(b_n x 2^{n-2} + ... + b_5 x 2^3 + b_4 x 2^2 + b_3 x 2^1 + b_2)x2 + b_1]_D = Q_2 x 2 + b_1$

$Q_1 = 6 \mid 2$
$b_1 = 0 \quad 3_D = Q_2 = (b_n x 2^{n-2} + ... + b_5 x 2^3 + b_4 x 2^2 + b_3 x 2^1 + b_2)_D$ Luego $b_1=0$

Sacando nuevamente factor común 2 en la expresión de Q_2:

$3 = Q_2 = [(b_n x 2^{n-3} + ... + b_5 x 2^2 + b_4 x 2^1 + b_3)x2 + b_2]_D = Q_3 x 2 + b_2$

$Q_2 = 3 \mid 2$
$b_2 = 1 \quad 1_D = Q_3 = (b_n x 2^{n-3} + ... + b_5 x 2^2 + b_4 x 2^1 + b_3)_D$ Luego $b_2=1$

Sacando nuevamente factor común 2 en la expresión de Q_3:

$1 = Q_3 = [(b_n x 2^{n-4} + ... + b_5 x 2^1 + b_4)x2 + b_3]_D = Q_4 x 2 + b_3$

$Q_3 = 1 \mid 2$
$b_3 = 1 \quad 0_D = Q_4 = (b_n x 2^{n-4} + ... + b_5 x 2^1 + b_4)_D$ Luego $b_3=1$

Puesto que $Q_4 = 0$ debe ser $b_n = ... = b_5 = b_4 = 0$

Los dígitos hallados forman el número $N = (b_3 b_2 b_1 b_0)_B = 1100_B = 12_D$

Si bien se ha justificado el método de las sucesivas divisiones por la base para el caso en que ésta es dos, expresando un número en base diez como sumatoria de potencias de una base **R** también en base diez, y sacando factor común **R** de la forma vista en este ejemplo, se generaliza la justificación para una base cualquiera.

A2.5 Expresión de un número fraccionario en potencias de su base

Números fraccionarios en base diez

En base diez un número fracionario sin parte entera es, por ejemplo

$N_F = 0{,}75_D = (7/10 + 5/100)_D = (7x10^{-1} + 5x10^{-2})_D$

Un número con dos dígitos fraccionarios, designados $d_{-1}d_{-2}$ (cada uno pudiendo valer 0 a 9) tendrá la forma:

$N_F = 0{,}d_{-1}d_{-2} = d_{-1}x10^{-1} + d_{-2}x10^{-2}$ siendo para 0,62 $\quad d_{-1}= 6 \quad d_{-2} = 2$

Generalizando, cualquier número fraccionario sin parte entera, en base diez tendrá la forma:

$$N_F = 0, d_{-1}d_{-2}d_{-3} ... d_{-q} = d_{-1}x10^{-1} + d_{-2}x10^{-2} + d_{-3}x10^{-3} + ... + d_{-q}x10^{-q} \qquad \text{(A2.7)}$$

y en forma más compacta $N_F = \sum_{i=1}^{i=q} d_{-i}\, x10^{-i}{}_D$ con $d_{-i} = \{0,1,2, ... ,9\}$

Números fraccionarios en base dos:

En cualquier base pueden definirse números fraccionarios, usando potencias negativas de la misma multiplicadasa por coeficientes que son los dígitos que están a la derecha de la coma identificatoria de la parte fraccionaria. Los mismos pueden presentar tantos valores como símbolos definen la base. Formalmente será, para una base genérica **R**:

$$N_F = (0,r_{-1}r_{-2}r_{-3} \dots r_{-t})_R = \sum_{i=1}^{i=t} r_{-i}x10^{-i}{}_R \quad \text{con } r_{-i} = \{0,1, \dots\} \qquad \textbf{(A2.8)}$$

Un número fraccionario en base dos podrá expresarse como:

$$\mathbf{N_F = 0, b_{-1}b_{-10}b_{-11} \dots b_{-t} = b_{-1} x\ 10^{-1} + b_{-10} x\ 10^{-10} + b_{-11} x\ 10^{-11} + \dots + b_{-t} x\ 10^{-t}} \qquad \textbf{(A2.9)}$$

siendo $b_{-i} = \{0,1\}$ y $10_B = 2_D$

Por ejemplo: $N_F = \mathbf{0,11}_B = 0, b_{-1}b_{-10\,B} = (1x10^{-1} + 1x10^{-10})_B$ siendo $b_{-1} = 1$ y $b_{-10} = 1$

Convirtiendo a base diez los números del paréntesis resulta:

$$N_F = \mathbf{0,11}_B = (1x2^{-1} + 1x2^{-2})_D = (1x1/2 + 1x1/4)_D = (0,5 + 0,25)_D = 0,75_D$$

Si en forma general en la expresión **(A2.9)** se reeemplazan los números binarios –incluídos los subíndices– por sus equivalentes decimales, se obtiene:

$$N_F = (0, b_{-1}b_{-2}b_{-3} \dots b_{-t})_B = (b_{-1} x\ 2^{-1} + b_{-2} x\ 2^{-2} + b_{-3} x\ 2^{-3} + \dots + b_{-t} x\ 2^{-t})_D =$$

$$= (b_{-1} x\ \mathbf{1/2} + b_{-2} x\ \mathbf{1/4} + b_{-3} x\ \mathbf{1/8} + \dots + b_{-t} x\ \mathbf{2^{-t}})_D \qquad \textbf{(A2.10)}$$

Resultan así los pesos 1/2, 1/4, 1/8 ... para los sucesivos dígitos binarios

A2.6 Expresión de un número real en su base y en base diez

Un número real **N**, expresado en cualquier base **R**, compuesto por una parte entera $\mathbf{N_E}$ más otra fraccionaria $\mathbf{N_F}$, tendrá una expresión general que será la suma de las expresiones antes formalizadas para ambas porciones:

$$N_R = (r_n r_{n-1} \dots r_0 , r_{-1}r_{-2}r_{-3} \dots r_{-t})_R = (r_n r_{n-1} \dots r_0 + \mathbf{0}, r_{-1}r_{-2}r_{-3} \dots r_{-t})_R = (\mathbf{N_E + N_F})_R = \sum_{i=-t}^{i=n} \mathbf{r_i x 10^i}{}_R \quad r_i = \{0,1, \dots R-1\}$$

(A2.12)

Expresando los $\mathbf{r_i}$ en decimal como $\mathbf{r_{id}}$ y con **R** simbolizando la base en decimal será en base diez:

$$N_D = (\mathbf{N_E + N_F})_D = \sum_{i=-t}^{i=n} \mathbf{r_{id} x R^{id}}{}_D \qquad \textbf{(A2.13)}$$

Ejemplo:

$$(A3,0F)_H = (A3 + 0,0F)_H = (Ax10^1 + 3 + Fx10^{-2})_H = (10x16^1 + 3 + 15x16^{-2})_D = (163 + 0,058593)_D = 163,058593_D$$

A2.7 Pasaje de un número real decimal a binario

El caso más general resulta cuando se tiene un número **N** con parte entera ($\mathbf{N_E}$) y parte fraccionaria ($\mathbf{N_F}$), como se ejemplifica.

$N = 23,62_D = (23 + 0,62)_D = N_E + N_F$

La parte entera se convierte de la forma conocida: $23_D = 10111_B = N_E$

En la siguiente igualdad, basada en la expresión (A2.10), los dígitos $b_{-1}b_{-2}b_{-3}$ ... de la parte fraccionaria serán las incógnitas, cuyo número y valor (0 ó 1) se desconoce:

$$N_F = 0,62_D = (b_{-1}x2^{-1} + b_{-2}x2^{-2} + b_{-3}x2^{-3} + \dots + b_{-n}x2^{-n})_D$$

Multiplicando por 2 ambos miembros[1] :

[1] A los efectos de determinar dichas incógnitas, en lo que sigue se debe tener presente que un coeficiente que **no** esté multiplicado por una potencia negativa de 2 **no** es un dígito de la parte fraccionaria, aunque su subíndice sea negativo. Así: $b_{-1} = b_{-1}x20$ siendo su valor (0 ó 1) el que corresponde a la parte entera del número en cuestión

$2 \times 0{,}62 = b_{-1} + b_{-2}.2^{-1} + b_{-3}.2^{-2} + \ldots + b_{-n}.2^{-n+1} = b_{-1} + N_{F2} = 1{,}24 = \mathbf{1} + 0{,}24$ Luego $b_{-1} = 1$

Parte entera ↑ ; N_{F2} ↔

Multiplicando la nueva parte fraccionaria obtenida N_{F2} por 2 resulta:

$2 \times N_{F2} = 2 \times 0{,}24 = 0{,}48 = \mathbf{0} + 0{,}48 = b_{-2} + b_{-3}.2^{-1} + b_{-4}.2^{-2} + \ldots + b_{-n}.2^{-n+2} = b_{-2} + N_{F3}$ $\quad b_{-2} = \mathbf{0}$

Parte entera ↑ ; N_{F3} ↔

Similarmente:

$2 \times N_{F3} = 2 \times 0{,}48 = 0{,}96 = \mathbf{0} + 0{,}96 = b_{-3} + N_{F4}$	Luego $b_{-3} = \mathbf{0}$
$2 \times N_{F4} = 2 \times 0{,}96 = 1{,}92 = \mathbf{1} + 0{,}92 = b_{-4} + N_{F5}$	Luego $b_{-4} = \mathbf{1}$
$2 \times N_{F5} = 2 \times 0{,}92 = 1{,}84 = \mathbf{1} + 0{,}84 = b_{-5} + N_{F6}$	Luego $b_{-5} = \mathbf{1}$
$2 \times N_{F6} = 2 \times 0{,}84 = 1{,}68 = \mathbf{1} + 0{,}68 = b_{-6} + N_{F7}$	Luego $b_{-6} = \mathbf{1}$
$2 \times N_{F7} = 2 \times 0{,}68 = 1{,}36 = \mathbf{1} + 0{,}36 = b_{-7} + N_{F8}$	Luego $b_{-7} = \mathbf{1}$
$2 \times N_{F8} = 2 \times 0{,}36 = 0{,}72 = \mathbf{0} + 0{,}72 = b_{-8} + N_{F9}$	Luego $b_{-8} = \mathbf{0}$
$2 \times N_{F9} = 2 \times 0{,}72 = 1{,}44 = \mathbf{1} + 0{,}44 = b_{-9} + N_{F10}$	Luego $b_{-9} = \mathbf{1}$
$2 \times N_{F10} = 2 \times 0{,}44 = 0{,}88 = \mathbf{0} + 0{,}88 = b_{-10} + N_{F11}$	Luego $b_{-10} = \mathbf{0}$
.....	.....

El procedimiento finaliza cuando se obtiene un número de bits estipulado, acorde a la precisión binaria deseada, o si antes alguna parte fraccionaria fue $0 = N_F$

En nuestro ejemplo, suponiendo que se necesiten diez bits fraccionarios:

$0{,}62_D = (b_{-1}b_{-2}\, b_{-3} \ldots b_{-n})_B = 0{,}1001111010_B = N_F$; y sumando N_E antes hallado:

$N = N_E + N_F = (23 + 0{,}62)_D = (10111 + 0{,}1001111010)_B = 10111{,}1001111010_B = 23{,}62_D$

A2.8 Pasaje de un número real binario a hexadecimal

En la unidad 1 se dieron las reglas prácticas para convertir un número entero binario a hexadecimal y viceversa. A través de un ejemplo se mostrará la consistencia de estas reglas, que pueden extenderse al pasaje de números reales entre dichas bases.

Regla para convertir un número hexadecimal con partes entera y fraccionaria a binario:

A partir de la coma, hacia la derecha e izquierda, se reemplaza cada dígito hexadecimal por el número binario de 4 bits equivalente. De existir, se eliminan los ceros superfluos.

Ejemplo: convertir a binario $3B{,}74_H$

$3B{,}74_H = 00111011{,}01110100_B = 111011{,}011101_B$

De hecho también da validez al pasaje inverso de hexa a binario, por el método de reemplazar cada dígito hexa por el número binario equivalente de 4 bits: si se parte del número binario hallado, se sigue el camino inverso expresándolo como suma de potencias positivas y negativas de 2, para luego sacar factor común 2^4, 2^{-4} y 2^{-8}, y por último hallar los valores decimales de cada paréntesis de 4 sumandos, al ser éstos convertidos a hexa se llega al número $3B{,}74_H$ Por lo tanto, es válida la siguiente regla:

Regla para convertir un número binario con partes entera y fraccionaria a hexadecimal:

1. *A partir de la coma, formar grupos de 4 bits hacia la izquierda y derecha de la misma. De ser necesario, se agregan ceros a uno u otro lado.*
2. *Se reemplaza cada grupo de 4 bits por su equivalente hexadecimal.*

þ **Ejemplo:** convertir a hexadecimal el número binario 101011111,0000101

$101011111{,}0000101_B = 000101011111{,}00001010_B = 15F{,}0A_H$

TABLA DE CONTENIDOS

A2.10 Ejercicios

1. Expresar el número 2036_D
   a) En una serie de potencias de diez en el sistema decimal
   b) Idem de dos en el sistema decimal;
   c) Idem de dieciseis en el sistema decimal
   d) Idem de dos en el sistema binario;
   e) Idem de dieciseis en el sistema binario.
2. Realizar el problema anterior para el número $202,34_D$
3. Pasar a binario y hexadecimal el número 1522_D y luego justificar conforme a la sección A2.8 el método algorítmico seguido
4. Dado el número 384_D, pasarlo a base dos, realizando operaciones en esta base, a partir de la expresión:
   $384_D = (3x10^2 + 8x10^1 + 4)_D$ Luego pasarlo a hexadecimal, operando en esta base.
5. Convertir a hexadecimal el número binario 110111001,001. Luego justificar el método utilizado, conforme a la sección A1.8

Apéndice: Respuestas de problemas seleccionados

Problemas de la sección A1.34

2. Si
5 a) Una de las entradas debe estar siempre en **1** (5 volts)
   b) Una de las entradas debe estar en **1** cuando se quier invertir la otra. Caso contrario debe estar en **0**
   c) Una entrada debe ser **A** y la otra **B**. Si **Z** = 0 es **A** = **B**; si **Z** = 1 será **A** ≠ **B**
   d) Si el número de unos es impar (01 ó 10) en **A** y **B**, será **Z** = 1; y si es par (00 ó 11) será **Z** = 0
10 La expresión del circuito es $\mathbf{Z = \bar{A}.B.C + A.\bar{B}.C}$
12 No y si, respectivamente
14 Si
15 No, se trata de una regla práctica
21 La salida $\mathbf{Z_7}$ vale **1** y las restantes **0**
26 Con cada pulso reloj se invierte el valor de la salida **Q**

Problemas de la sección N.13

1. 100000000 – 01000111
2. 1111111110110010
3. 2 bytes
4. La máxima magnitud es 4.294.967.295. El mayor entero positivo es +2.147.483.647 y el más negativo es –2.147.483.648
5. a) 00096 + 00180 = 00276 = +276 b) 00096 + 99820 = 99916 ≡ –84 c) 999094 + 00180 = 00084 = +84
   d) 99904 + 99280 = 99724 ≡ –276
8 a) 00010010 + 01100110 = 01111000 = 120 SZVC=0000; b) 00010010+10011010 = 10101100 ≡ –84 SZVC=1000
   c) 11101110 + 01100110 ≡ 01010100 = 84 SZVC=0001 d) 11101110+10011010 ≡ 10001000 ≡ –120 SZVC=1001
9, La verificación no concuerda en formato 8 por que hay overflow. En cambio concuerda en formato 16
10. a) 1010110 ≡ –84 b) 01111000 = 120 c) 10001000 ≡ –120 d) 01010100 = 84
13. 10010110 + 11011011 = 01110001 = 113 (se descarta el 1 de carry en la resta de naturales como en la de signados)
    Estas operación no se puede hacer en formato 8 para signados, pues para representar 150 se necesitan por lo menos 9 bits.
17. Al código ASCII de la D se le resta el ASCII de la R, y en función del indicador C se conoce el orden.
18. $10^{1010}{}_B$; $10^{A}{}_H$ $10^{10}{}_D$
20. 2^{10}
22 $139,2_H$
25 $1,0x10^{-11}{}_B = 0,001_B = 1/8_D$
26 $11000010010001_B = 3091_H = 12433_D = 0001001001000011001 1_{BCD} = 00110001\ 00110010\ 00110100\ 00110011\ 00110011_{ASCII}$
27 a) -110_D b) 140_D c) 92_D

Algebra de Boole

INFORMACION PARA EL LECTOR

- [ ] Principiante
- [ ] Medio
- [ ] Avanzado
- [x] Todos los niveles

Esta unidad ha sido pensada y estructurada de manera que los temas progresen desde conceptos generales hasta su desarrollo en detalle, prevaleciendo el criterio de empezar cualquier tema a partir de conocimientos ya adquiridos por el lector en su vida cotidiana, y que le son familiarires. A partir de ellos se construyen nuevos saberes, acorde con modernas metodologías.

Con referencia a la representación de datos, el lector encontrará sistematizadas las distintas codificaciones que puede presentar un número dentro de un computador, en lugar de simples reglas de representación, aisladas de la cadena de procesos que se dan en un computador. En particular se tratan los formatos en un procesador 486/Pentium. Asimismo, se definen y se muestran aplicaciones de los "flags" SZVC.

A los fines de la ejercitación, se presenta un conjunto de problemas, cuya respuesta se indica en la mayoría de los casos. Entre los ejercicios resueltos se incluye un detalle de la codificación y suma de enteros con bit de signo, usando registros y memoiras de una PC. Para concretarlos se emplea el programa Debug del DOS

Diseño de tapa: Rafael Ginzburg

La presente Unidad forma parte de la obra "INTRODUCCION GENERAL A LA INFORMATICA", que consta de las siguientes 4 unidades que pueden leerse en forma independiente:

1. *"La PC por dentro, Arquitectura y Funcionamiento de Computadoras"*
2. *"Periféricos y Redes"(en prensa)*
3. *"Assembler desde cero"*
4. *"Algebra de Boole Aplicada a Circuitos de Computación" (editado)*

Del mismo autor "INTRODUCCION A LAS TECNICAS DIGITALES CON CIRCUITOS INTEGRADOS" (7ma edicion)