

**ARQUITECTURA DE  
COMPUTADORAS**

**Universidad Tecnológica Nacional – Facultad Regional Mendoza**

**Cátedra Arquitectura de Computadoras**

**Carrera Ingeniería en Sistemas de Información**

# **ARQUITECTURA DE COMPUTADORAS**

*Daniel M. Argüello, Santiago C. Pérez e Higinio A. Facchini*

Mendoza, Argentina, 2018

## **Arquitectura de Computadoras**

*Daniel M. Argüello, Santiago C. Pérez e Higinio A. Facchini*

*Diseño de Tapa: Renzo Guido Facchini*

*Diseño de Interior: Renzo Guido Facchini*

*Corrección de Estilo: Renzo Guido Facchini*

E-book – 4° Edición

ISBN: 978-950-42-0158-8

Rodriguez 273

(5500) Mendoza, República Argentina

*“No hay viento favorable  
para el barco que no sabe adónde va”  
-- Lucio Anneo Séneca (4 a. C. - 65 d. C.)*

*"La imaginación es más importante que el conocimiento.  
El conocimiento es limitado, mientras que la imaginación no"  
-- Albert Einstein (1879 – 1955)*



Capítulo 1: Representación Numérica

Capítulo 2: Códigos Numéricos y Alfanuméricos

Capítulo 3: Algebra de Boole

Capítulo 4: Sistemas Combinacionales

Capítulo 5: Sistemas Secuenciales

Capítulo 6: Memorias Electrónicas

Capítulo 7: Arquitectura Básica de una Computadora

Capítulo 8: Arquitectura Convencional

Capítulo 9: Arquitectura Avanzada

Capítulo 10: Entradas y Salidas (E/S)



## **PRÓLOGO**

El libro tiene como objetivo reunir en una sola obra la experiencia docente y profesional de los autores en las temáticas de Técnicas Digitales y Arquitecturas de Computadoras, y generar un material que integra y adecua tópicos que están dispersos, con diversa profundidad y perspectiva, en bibliografías de grado universitarios. El libro presenta en los primeros Capítulos los contenidos de Sistemas de Numeración, Códigos y Álgebra de Boole, típicos para dar la base de conocimiento. Luego, se discuten los temas más específicos referidos a los Sistemas Electrónicos Digitales, del tipo Combinacional y Secuencial, y específicamente a las Memorias Electrónicas. A continuación, se plantea la Arquitectura Básica de una Computadora, usando una máquina elemental didáctica para facilitar la apropiación de los conceptos. Finalmente, se tratan tópicos más avanzados sobre Arquitecturas Convencionales y Avanzadas, y las Entradas/Salidas en una Computadora.

El libro ha sido pensando especialmente para los alumnos de los primeros años de grado de las carreras de Ingeniería en Sistemas, Ingeniería en Electrónica y Tecnicaturas en TICs de la UTN, y de las carreras de grado y técnicas TICs, en general, de cualquier institución universitaria, con el objeto de otorgar una visión integrada, de acuerdo al perfil y orientación de los Planes de Estudio nacionales y latinoamericanos.

Agradecemos la colaboración de quienes generosamente han aportado su tiempo y conocimientos para la revisión y elaboración de las tablas, gráficas y figuras. Además, el apoyo de las autoridades, presididas por el Ing. José Balacco, Decano de la Facultad Regional Mendoza.

Daniel Argüello - Santiago Pérez - Higinio Facchini  
Mendoza, Argentina, octubre de 2018



---

# CAPÍTULO 1

---

## Representación Numérica

- 1 Sistemas de Numeración**
    - 1.1 Introducción**
    - 1.2 Confiabilidad**
    - 1.3 Costo**
  - 2 Sistema de Numeración Binario**
    - 2.1 Introducción**
    - 2.2 Conversión entre números de distintas bases**
    - 2.3 Complementos binarios**
    - 2.4 Representación de números negativos en binario**
  - 3 Punto Fijo y Punto Flotante**
    - 3.1 Introducción**
    - 3.2 Operaciones aritméticas**
    - 3.3 Norma IEEE 754**
  - 4 Ejercitación**
-

## Capítulo 1

# Representación Numérica

---

## 1 Sistemas de Numeración

### 1.1 Introducción

Los números naturales aparecen al contar los objetos de un conjunto. En la primera infancia se empieza por aprender a contar, luego más tarde se coordinan conjuntos prescindiendo del orden. Lo mencionado parece justificar la introducción del número natural mediante la formulación axiomática que revele en qué consiste la operación de contar.

Con posterioridad a la invención del número natural surge la necesidad de ampliar el concepto de número. Aparece entonces el número entero, el racional, el irracional y el número real.

Podríamos avanzar rápidamente e introducir el concepto de Sistema de Numeración como el conjunto de reglas y convenios que permiten la representación de todos los números mediante varios signos, o varias palabras.

Existen Sistemas de Numeración muy conocidos como el Romano, que descompone el número en suma o diferencia de otros varios, cada uno de los cuales está representado por un símbolo especial:

I, V, X, L, C, D, M

Otro sistema es el Decimal, que en vez de introducir nuevos símbolos para estos diversos sumandos, utiliza el principio del valor relativo (posicional), es decir, una misma cifra representa valores distintos según el lugar que ocupa. Estos sistemas son los que representan interés matemático. El Sistema Decimal, que fue inventado en la India en el siglo IV antes de Cristo y llevado a Europa por los árabes en la Edad Media, está fundado en el número fijo que llamamos diez (habiendo elegido este y no otro quizás porque tenemos diez dedos en las manos).

Toda combinación de operaciones fundamentales efectuadas con números cualesquiera que da origen a un nuevo número, se llama algoritmo de numeración. Para los sistemas de numeración basados en el valor relativo (posicionales), el algoritmo de numeración consiste en un polinomio:

$$N = a_n b_n + a_{n-1} b_{n-1} + a_{n-2} b_{n-2} + \dots + a_1 b_1 + a_0 b_0 + \\ + a_{-1} b_{-1} + a_{-2} b_{-2} + \dots + a_{-k} b_{-k}$$

dónde:

N: Número

$a_i$ : número natural menor que b (símbolo)

b: base del sistema (cantidad de símbolos diferentes del sistema, es un número natural mayor que 1)

$n + 1$ : cantidad de cifras enteras.

k: cantidad de cifras fraccionarias.

Obsérvese que la parte entera del número se corresponde a los términos del polinomio que tienen la base b elevada a exponentes cero o positivos, mientras que la parte fraccionaria se corresponde con los exponentes negativos.

También, puede apreciarse el principio de valor relativo; por ejemplo, el número 24,2 en el conocido sistema de numeración decimal tiene el siguiente polinomio:

$$24,2 = 2 \times 10^1 + 4 \times 10^0 + 2 \times 10^{-1}$$

Obsérvese que 2 tiene un valor que depende de la posición que ocupa en el número.

Las propiedades de los números, que se estudian en Matemáticas, son válidas cualquiera sea la base utilizada, siempre que se utilice el mismo algoritmo de numeración. Es decir, que si elegimos otra base podríamos realizar operaciones algebraicas (suma, resta, multiplicación, división, etc.) sin ningún problema, ya que la axiomática y teoremas parten del algoritmo de numeración. Como se dijo anteriormente se eligió 10 (diez) como base, pero podemos hacernos esta pregunta:

¿Es el Sistema de Numeración Decimal el más adecuado para ser usado en sistemas físicos de representación y tratamiento de la información? La respuesta resulta de tener en cuenta y evaluar aspectos como el costo y confiabilidad del Sistema Físico a construir.

## 1.2 Confiabilidad

Podemos decir que un sistema físico es más confiable cuando su correcto funcionamiento sea lo más independiente posible de la temperatura en la que trabaja, del envejecimiento y de la dispersión (tolerancia) de los componentes que lo forman.

Los componentes que se utilizan en la construcción de los sistemas físicos de procesamiento de información, son eléctricos y electrónicos (transistores, diodos, resistores, capacitores, inductores, etc.); todos ellos cambian sus parámetros con el envejecimiento y la temperatura. Además, es imposible construir dos componentes idénticos; a esto se lo llama dispersión. Los fabricantes especifican el porcentaje de dispersión máximo de los componentes que comercializan.

A fin de aclarar lo mencionado comparemos dos sistemas físicos (Figura 1.1) que representan números en distintas bases: base 10 y base 2. La Fig. 1.1a indica un circuito eléctrico capaz de representar un número decimal (base 10), y la Fig. 1.1b indica un circuito eléctrico capaz de representar un número binario (base 2). Ambos consisten en un foco que se enciende con distintas intensidades luminosas al mover la llave cierta cantidad de posiciones (diez en el caso a y dos en el caso b), y suponen que un observador tiene que ser capaz de determinar el número en cuestión apreciando la intensidad luminosa.

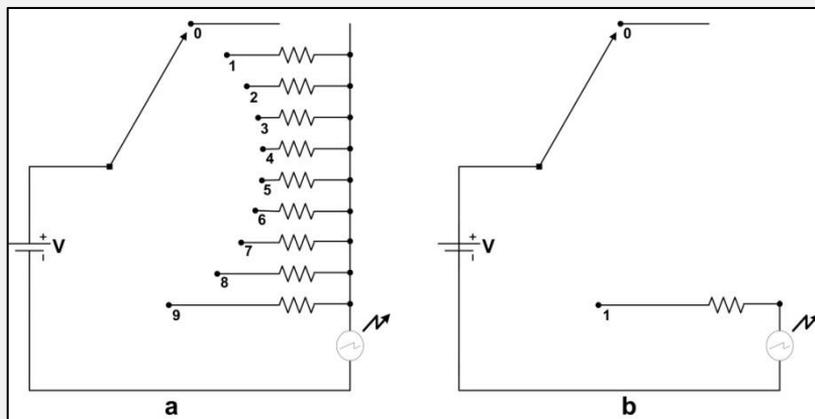


Fig. 1.1 Sistemas físicos que representan números.

Como primera conclusión resulta evidente que es más fácil determinar el estado del foco en el caso b (prendido o apagado) que en el caso a (diez intensidades posibles).

Hay una segunda apreciación. Supongamos que aumenta la temperatura ambiente o que el circuito ha envejecido. Esto modifica el valor de las resistencias y de la intensidad emitida por el foco. La temperatura y el envejecimiento afectan mucho más al caso a que al b.

Finalmente, supongamos que tenemos que reponer el foco porque se quemó. El nuevo foco, debido a la dispersión, tendrá seguramente parámetros diferentes al anterior. (Mismas consideraciones si debiésemos remplazar una resistencia). La dispersión afecta más al caso a que al b.

Como conclusión final puede decirse que el circuito que trabaja en binario es más confiable que el decimal. Podemos decir que es más confiable que cualquier circuito que trabaje en cualquier base.

Si bien los sistemas digitales no se construyen con focos y llaves, lo considerado es aplicable a los componentes que se utilizan en la realidad.

### 1.3 Costo

Nuevamente observando la Figura 1.1 desde el punto de vista del costo, podemos afirmar que es más costoso construir el circuito en decimal que en binario. El Costo es proporcional a la base del sistema de numeración utilizado. Sin embargo, necesitamos más circuitos binarios para representar una misma cantidad, (por ejemplo para representar el número 25 necesitaríamos dos circuitos decimales como el de la Fig.1.1a y cinco circuitos binarios como el de la Fig.1.1b). Entonces, el Costo es proporcional a la cantidad de cifras. Por ello, podemos escribir:

$$\text{Costo} = k \cdot b \cdot (n+1)$$

Entre b y (n + 1) existe una relación:

$$b^{(n+1)} = M$$

Donde M: Módulo del Sistema (máximo número representable con n+1 cifras)

$$(n+1) \cdot \text{Ln } b = \text{Ln } M$$

$$n + 1 = (\text{Ln } M / \text{Ln } b)$$

Reemplazando:

$$\text{Costo} = k \text{Ln } M (b / \text{Ln } b)$$

Haciendo la derivada respecto a la base ( $d\text{Costo}/db = 0$ ) e igualando a cero obtenemos que:

$$b_{\text{óptima}} = e$$

Concluimos que la base óptima, desde el punto de vista del costo, está entre 2 y 3. En realidad, la expresión del costo planteada es aproximada. Teniendo en cuenta la confiabilidad, el Sistema Binario es el que se utiliza en la construcción de los Sistemas Digitales.

## 2 Sistema de Numeración Binario

### 2.1 Introducción

En el Sistema Binario, los símbolos utilizados son el 0 y el 1, llamados dígitos binarios (bits o bitios). Es posible, aplicando el algoritmo de numeración, obtener los números binarios correspondientes a las primeras  $16_{10}$  (el subíndice 10 indica que el número está en Sistema Decimal) cantidades. Igualmente, podríamos obtener los números en el Sistema Octal y en el Sistema Hexadecimal. La Tabla 1.1 indica las correspondencias.

Obsérvese que en el Sistema Hexadecimal se han agregado las letras A, B, C, D, E y F para obtener los 16 símbolos distintos necesarios.

La razón de tener en cuenta este Sistema es que  $2^4 = 16$ , y por lo tanto, cada 4 cifras binarias se corresponderá 1 cifra hexadecimal. También en el Sistema Octal se da algo parecido ya que  $2^3 = 8$ , y cada tres cifras binarias se corresponde una Octal. Esto último nos permite encontrar rápidamente las equivalencias entre binario, octal y hexadecimal. Por ejemplo:

$$\begin{aligned} 0010\ 1001\ 1110\ 0101_2 &= 29E5_{16} \\ 110\ 111\ 111\ 001\ 000\ 110_8 &= 677106_8 \end{aligned}$$

Un grupo de cuatro bits recibe el nombre de nibble y un grupo de ocho bits recibe el nombre de octeto o byte.

<b>Decimal</b> <b>b = 10<sub>10</sub></b>	<b>Binario</b> <b>b = 2<sub>10</sub></b>	<b>Octal</b> <b>b = 8<sub>10</sub></b>	<b>Hexadecimal</b> <b>b = 16<sub>10</sub></b>
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Tabla 1.1. Correspondencia entre sistemas de numeración.

Es usual en estos temas utilizar los prefijos kilo, mega, giga, tera, etc. de una manera similar que en el Sistema Métrico Decimal, pero con algunas diferencias, a saber:

$$\begin{aligned}
 1 \text{ Kilo} &= 1\text{K} = 1024 = 2^{10} \\
 1 \text{ Mega} &= 1\text{M} = 1024\text{K} = 2^{20} \\
 1 \text{ Giga} &= 1\text{G} = 1024\text{M} = 2^{30} \\
 1 \text{ Tera} &= 1\text{T} = 1024\text{G} = 2^{40}
 \end{aligned}$$

Así, cuando decimos por ejemplo 1Mbit, estamos indicando que se trata de 1.048.576 bits ( $2^{20}$ ), o si decimos 1Kbyte estamos indicando 1024 bytes ( $2^{10}$ ).

Todas las operaciones aritméticas estudiadas en el álgebra, aplicadas a los números del Sistema Decimal, son aplicables a los números binarios, por cuanto ambos Sistemas responden al mismo algoritmo de numeración.

## 2.2 Conversión entre números de distintas bases

Para convertir las expresiones de números entre distintas bases puede usarse el siguiente procedimiento general:

- a) Parte entera del número (términos del polinomio con la base elevada a exponentes cero o positivos)  
 Supongamos que el primer miembro de la siguiente igualdad está expresado en un Sistema de Numeración en base  $b = b_1$ , y el segundo miembro en base  $b = b_2$

$$N_{b_1} = a_n b_1^n + a_{n-1} b_1^{n-1} + \dots + a_1 b_1^1 + a_0 b_1^0$$

Si dividimos miembro a miembro por  $b_2$  vemos que el último término del segundo miembro será  $a_0$ , que es el resto de la división y la cifra menos significativa del número en base  $b_2$ .

Repetiendo la división entre el cociente anterior y  $b_2$ , obtendremos como resto  $a_1$ , y así sucesivamente hasta llegar a  $a_n$ . El ejemplo presentado en la Tabla 1.2 aclarará lo expuesto:

Ejemplo:

$$b_1 = 10_{10}$$

$$N_{b_1} = 25_{10}$$

$$b_2 = 2_{10}$$

<b>25</b>	<b>2</b>				
<b>Resto = 1</b>	<b>12</b>	<b>2</b>			
	<b>Resto = 0</b>	<b>6</b>	<b>2</b>		
		<b>Resto = 0</b>	<b>3</b>	<b>2</b>	
			<b>Resto = 1</b>	<b>Resto = 1</b>	<b>2</b>

Tabla 1.2. Secuencia de conversión de decimal a binario.

De acuerdo a lo mencionado, tenemos:  
 $25_{10} = 11001_2$

b) Parte fraccionaria del número (términos del polinomio con la base elevada a exponentes negativos)

Supongamos que el primer miembro de la siguiente igualdad esta expresado en un Sistema de Numeración en base  $b = b_1$ , y el segundo miembro en base  $b = b_2$

$$N_{b_1} = a_{-1} b_1^{-1} + a_{-2} b_1^{-2} + \dots + a_{k+1} b_1^{k+1} + a_{-k} b_1^{-k}$$

Si multiplicamos miembro a miembro por  $b_2$  observamos que el primer término del segundo miembro es  $a_{-1}$ , es decir, la primera cifra más significativa de la parte fraccionaria. Reiterando la multiplicación sólo sobre la parte fraccionaria del resultado, vamos obteniendo las sucesivas cifras menos significativas.

Un ejemplo aclarará lo expuesto:

Ejemplo:

$$b_1 = 10_{10}$$

$$N_{b_1} = 0,21_{10}$$

$$b_2 = 2_{10}$$

$$0,21 \times 2 = 0,42$$

$$0,42 \times 2 = 0,84$$

$$0,84 \times 2 = 1,68$$

$$0,68 \times 2 = 1,36$$

$$0,36 \times 2 = 0,72$$

$$0,72 \times 2 = 1,44$$

$$0,44 \times 2 = 0,88$$

.....

.....

De acuerdo a lo mencionado tenemos:

$$0,21_{10} = 0,0011010\dots\dots_2$$

Si tenemos números con parte entera y fraccionaria, aplicamos la parte a) y la parte b).

## 2.3 Complementos binarios

### a) Complemento a la base:

El complemento a la base de un número  $N$  que posee  $m$  cifras enteras, se define como:

$$C_b(N) = b^m - N$$

**b) Complemento a la base disminuida:**

El complemento a la base disminuida de un número N que posee m cifras enteras, se define como:

$$C_{b-1}(N) = b^m - N - 2^{-k}$$

dónde k es la cantidad de cifras fraccionarias.

Si bien los complementos se aplican a Sistemas de Numeración de cualquier base, los utilizaremos especialmente en el Sistema de Numeración Binario. En este caso, los llamaremos Complemento a Dos y Complemento a Uno, respectivamente.

Veamos algunos ejemplos:

Ejemplo de Complemento a 2:

$$C_2(10011_2) = 10_2^{101_2} - 10011_2 = 100000_2 - 10011_2 = 01101_2$$

Ejemplo de Complemento a 1:

$$C_1(10011_2) = 10_2^{101_2} - 10011_2 - 1_2 = 100000_2 - 10011_2 - 1_2 = 01100_2$$

De ahora en adelante se omitirán los subíndices 2 para indicar números binarios, a menos que no se pueda interpretar correctamente.

Observando los ejemplos, vemos que el Complemento a 1 puede obtenerse cambiando ceros por unos y viceversa. Esto es consecuencia de restar a m 1s un número de m bits. Como consecuencia de lo anterior, el Complemento a 2 de un número binario puede obtenerse cambiando unos por ceros y viceversa, y sumando uno, de acuerdo a la definición de Complemento a 2.

Para el caso de números binarios con parte fraccionaria (por ejemplo 1100011,110) la definición y reglas mencionadas siguen siendo válidas.

## 2.4 Representación de números negativos en binario

Los números negativos se pueden representar de distintas formas, a saber:

### a) Valor Absoluto y Signo:

Supongamos un número binario de  $n$  bits. El bit más significativo se reserva para el signo (bit de signo  $B_s$ ) y los restantes para el valor absoluto (Mantisa  $M$ ). Esta es la forma en que representamos los números negativos en decimal. Bit de Signo 0 (cero): + (positivo), y 1 (uno): - (negativo).

### b) Mediante el Convenio de Complemento a 2:

El bit más significativo es el bit de signo. Los restantes (Mantisa  $M$ ), si el número es negativo, son el complemento a 2. Bit de Signo 0: + (positivo), 1: - (negativo).

### c) Mediante el Convenio de Complemento a 1:

El bit más significativo es el bit de signo, los restantes (Mantisa  $M$ ), si el número es negativo, son el complemento a 1. Bit de Signo 0: + (positivo), 1: - (negativo).

### d) Mediante el Convenio Exceso $2^{n-1}$ :

El bit más significativo es el bit de signo, los restantes (Mantisa  $M$ ), si el número es negativo, son el complemento a 2. Bit de Signo 0: - (negativo), 1: + (positivo).

Veamos un ejemplo de los distintos Convenios para un número de 3 bits ( $n = 3$ ).

Obsérvese en la Tabla 1.3 que:

- Usando Complemento a 2 se tiene un solo cero, lo que puede resultar una ventaja como después se verá.
- Usando Exceso 4, a menores combinaciones binarias absolutas se corresponden menores equivalentes decimales. Esto es útil para la representación de los exponentes en Punto Flotante, como veremos luego.

- En Unidades posteriores se verá la conveniencia del uso de los Convenios que usan Complementos, ya que de esta forma la operación resta se puede realizar mediante una suma. Esto simplifica los circuitos lógicos.

A excepción de la representación con Valor Absoluto y Signo, la utilización de estas representaciones para realizar operaciones aritméticas requiere definir reglas apropiadas que se expondrán posteriormente. No obstante adelantaremos algunos conceptos.

Combinación binaria (n = 3)		Valor absoluto y Signo	Usando Complemento a 2	Usando Complemento a 1	Exceso 4
BS	Mantisa				
0	00	+0	0	+0	-4
0	01	+1	+1	+1	-3
0	10	+2	+2	+2	-2
0	11	+3	+3	+3	-1
1	00	-0	-4	-3	0
1	01	-1	-3	-2	+1
1	10	-2	-2	-1	+2
1	11	-3	-1	-0	+3

Tabla 1.3. Representación de números enteros usando 3 bits.

En la tabla anterior puede observarse que para el caso de los Convenios que usan Complemento a 2 y Complemento a 1, se puede incluir el bit de signo como un bit más del número. Así, por ejemplo, para encontrar la combinación binaria correspondiente a -3 en el Convenio que usa Complemento a 2, basta con partir de la combinación correspondiente a +3 (011), y encontrar su Complemento a 2 incluyendo el Bit de Signo, es decir,  $C_2(011) = 1000 - 011 = 101$ . De esta forma el Bit de Signo se utiliza como un bit más.

Veamos el siguiente ejemplo:  $+3 + (-3) = 0$  usando el Convenio de Complemento a 2:

$$\begin{array}{r}
 011 \\
 + \\
 111 \\
 \hline
 \underline{1000}
 \end{array}$$

dónde el 1 de la izquierda se rechaza por tratarse de números de 3 bits. Si usamos el Convenio de Complemento a 1 será:

$$+1 + (-3) = -2$$

$$\begin{array}{r}
 001 \\
 + \\
 100 \\
 \hline
 101
 \end{array}$$

Como se observa, se ha usado el bit de signo como si se tratara de un bit más.

## 3 Punto Fijo y Punto Flotante

### 3.1 Introducción

Lamentablemente, no todos los números son enteros. Así, surgen representaciones binarias con uno, dos, tres, o más dígitos (bits) fraccionarios, según la necesidad. La llamada Notación en Punto Fijo resuelve ese problema. Por ejemplo, un formato podría ser:

1101001,101

En este caso disponemos de 10 bits para representar el número, de los cuales 7 son enteros y 3 fraccionarios.

El problema es que al construir un Sistema Digital debemos fijar el mismo formato para todos los números. Además, en las ciencias es usual la necesidad de representar números muy pequeños, y a la vez, números muy grandes. Con una cantidad limitada de dígitos resulta imposible abarcar un amplio rango de cantidades sin usar la llamada Notación en Punto Flotante. Esta Notación, muy similar a la Notación Científica, divide al número en tres campos, como se muestra en la Figura 1.2, a saber:

- Signo de la Mantisa
- Exponente: Signo del Exponente - Mantisa del Exponente
- Mantisa



Fig. 1.2. Representación de números reales en punto flotante.

#### a) Signo de la Mantisa (S)

Es un campo de 1 bit.

0 positivo ( + )

1 negativo ( - )

### **b) Exponente (E)**

Es un campo de varios bits con la Convención Exceso  $2n-1$ . Por ejemplo, el Exceso 64 usa 7 bits y el Exceso 128 requiere 8 bits.

### **c) Mantisa(M)**

Es un campo de varios bits fraccionarios. Para que el número esté normalizado, el bit más significativo debe ser 1, es decir, la mantisa debe ser mayor o igual a  $1/2$  y menor que 1.

## **3.2 Operaciones aritméticas**

Las operaciones aritméticas con números en punto flotante tienen ciertas particularidades. Sean  $S_1E_1M_1$  y  $S_2E_2M_2$  dos números en punto flotante normalizados y en Exceso 64.

### **a) Multiplicación**

El producto de los dos números será  $S_pE_pM_p$ , donde:

Signo de la mantisa

$$S_p = 0 \text{ si } S_1 = S_2$$

$$S_p = 1 \text{ si } S_1 \neq S_2$$

Exponente

$$E_p = E_1 + E_2 - 64 - K,$$

siendo K el valor que satisface  $1/2 \leq M_1 \times M_2 \times 2^K < 1$

Mantisa

$$M_p = M_1 \times M_2 \times 2^K$$

### **b) División**

La división de los dos números será  $S_dE_dM_d$ , donde:

Signo de la mantisa

$$S_d = 0 \text{ si } S_1 = S_2$$

$$S_d = 1 \text{ si } S_1 \neq S_2$$

Exponente

$E_d = E_1 - E_2 + 64 - K$ ,  
siendo K el valor que satisface  $1/2 \leq (M_1 / M_2) 2^K < 1$

Mantisa

$$M_d = (M_1 / M_2) 2^K$$

### c) Suma y resta

Para sumar o restar los dos números debemos igualar sus exponentes. Debido a que los exponentes están en Exceso, y en esta convención menores números se corresponden con menores combinaciones binarias absolutas, la comparación es muy sencilla. Supongamos que:

$$E_1 > E_2$$

Entonces encontramos j, tal que  $j = E_1 - E_2$

Modificamos  $S_2E_2M_2$  a fin de igualar los exponentes:

$$S_2(E_2 + j)M_2/2^j$$

Luego, procedemos a sumar o restar las mantisas según sea el caso. El resultado debe ser luego normalizado como se expuso en la división y el producto.

## Base de Exponenciación

Con el objeto de aumentar el rango de representación, la base de exponenciación en vez de 2 puede ser 4, 8 o 16. Por ejemplo, si en una representación en punto flotante la base de exponenciación es 8, la mantisa debe interpretarse en Octal (grupos de 3 bits partiendo desde la coma hacia la derecha). La normalización en estos casos se obtiene cuando el dígito más significativo (grupo de tres bits) de la mantisa no es cero.

## 3.3 Norma IEEE 754

A fin de estandarizar la representación en punto flotante, el IEEE dictó la Norma 754 (1985), la cual es respetada actualmente por la mayoría de los fabricantes de procesadores.

- Esta norma establece tres tipos de representaciones, como se observa en la Figura 1.3:

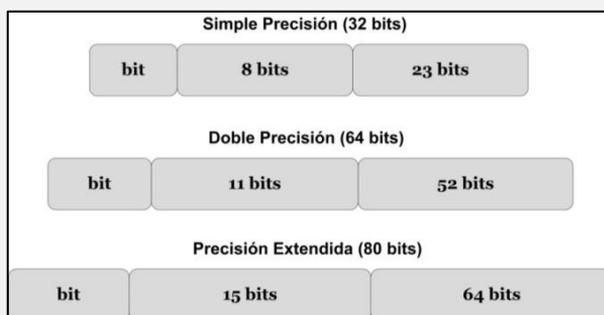


Fig. 1.3. Representaciones de punto flotante de la Norma 7.

- Los exponentes mínimo y máximo se utilizan para casos especiales.

Ya que las mantisas normalizadas siempre empiezan con 1, no es necesario almacenarlo y la coma implícita se coloca a la derecha del 1. Entonces la mantisa, ahora llamada significando, es mayor o igual a 1 y menor que 2.

- Se consideran 5 clases de números, como se indica en la Figura 1.4

<b>1.Normalizados</b>		
+/-	$0 < E < \text{Máximo}$	Cualquier patrón de bits
<b>2.DesNormalizados (el bit 1 a la izquierda de la coma del significando, es 0)</b>		
+/-	0	Cualquier patrón de bits excepto 0
<b>3.Cero (el bit 1 a la izquierda de la coma del significando, es 0)</b>		
+/-	0	0
<b>4.Infinito</b>		
+/-	1111....1111111	0
<b>5.No número (Indica el cociente Infinito/Infinito)</b>		
+/-	1111....1111111	Cualquier patrón de bits excepto 0

Fig. 1.4. Casos especiales de números en punto flotante.

## 4 Ejercitación

### Ejercicio 1:

Convertir a decimal los siguientes números.

- a)  $1101010011_2$       b)  $1B_{16}$       c)  $111101,10101_2$   
d)  $232_8$       e)  $575,54_8$       f)  $2CD,5_{16}$

Resolución del punto a)

Siempre que se quiera convertir de cualquier sistema de numeración a decimal el método más apropiado es el del polinomio de numeración.

Se representa el número a convertir mediante el polinomio y se lo opera en decimal obteniendo el resultado en decimal.

$$1101010011_2 = 1x2^9 + 1x2^8 + 0x2^7 + 1x2^6 + 0x2^5 + 1x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 1x2^0$$
$$1101010011_2 = 512 + 256 + 64 + 16 + 2 + 1 = 851_{10}$$

### Ejercicio 2:

Convertir a octal los siguientes números.

- a)  $1382_{10}$       b)  $7523,236_{10}$       c)  $11101001_2$   
d)  $1011011,1011_2$       e)  $45BA_{16}$       f)  $DCBA,2F_{16}$

### Ejercicio 3:

Convertir a base dos el número 78,3 en base diez, y volverlo a base diez apreciando seis bits fraccionarios. ¿Qué conclusiones resultan?

### Ejercicio 4:

¿Cuál es el mayor número decimal que puede ser representado por 3 y 6 dígitos hexadecimales?

### Ejercicio 5:

Ordenar los siguientes números de mayor a menor.

- $BA3_{16}$        $11001011_2$        $8342_{10}$   
 $E4A_{16}$        $6895_{10}$        $11101101111_2$

### Ejercicio 6:

¿Cuántos símbolos tendrá un sistema de numeración con base 14? Indicar los posibles individuos del mismo.

### Ejercicio 7:

Efectúe las siguientes operaciones utilizando palabras de 8 bits,

- 1) Representando los números negativos mediante el convenio de complemento a dos,

2) Representando los números negativos mediante el convenio de complemento a uno.

- a)  $35 - 21$                       b)  $-48 + 21$   
c)  $-25 - (-39)$                   d)  $-63 - 18$

**Ejercicio 8:**

Indique cuál es el máximo número decimal positivo y negativo representado en el convenio de complemento a dos, si se utilizan palabras de 16 bits y 32 bits.

**Ejercicio 9:**

Disponiendo de una palabra de 36 bits, de los cuáles 1 es destinado para el bit de signo del número, 8 para el exponente y 27 para la mantisa, represente en punto flotante y en base 2, 8 y 16 los siguientes números decimales.

- a) 29                      b) 52,73                      c) 0,125                      d) 0,011

**Ejercicio 10:**

Efectúe las siguientes operaciones de acuerdo a la representación en punto flotante indicada en el ejercicio anterior, y en base 2.

- a)  $110011 + 11100,110110 + 101,11$   
b)  $1110100,0111 + 10110,0011$   
c)  $1010010,00101 + 0,000010011001$

**Ejercicio 11:**

Obtenga los números decimales de la siguiente representación en punto flotante de base 2.

- a) 0 10001101 1110010010100001...  
b) 1 01010001 1111010001100001...

**Ejercicio 12:**

Indique cuáles son los máximos y mínimos números positivos decimales que pueden representarse en el formato de punto flotante indicado en simple y doble precisión, y base 2.

**Ejercicio 13:**

Efectúe las siguientes operaciones de acuerdo a la representación en punto flotante indicada en el ejercicio anterior, y en base 2.

- a)  $0,00000010111101 + 10011110,001$   
b)  $0,11111010 + 0,000000110$   
c)  $1101001010,0001 + 0,000001100001011$

---

# CAPÍTULO 2

---

## Códigos Numéricos y Alfanuméricos

- 1 Códigos**
    - 1.1 Introducción**
    - 1.2 Códigos binarios**
    - 1.3 Códigos BCD**
    - 1.4 Códigos alfanuméricos**
  - 2 Códigos Detectores y Correctores de Error**
    - 2.1 Introducción**
    - 2.2 Distancia mínima**
    - 2.3 Códigos detectores de error**
    - 2.4 Códigos correctores de error**
  - 3 Encriptación o Cifrado**
  - 4 Otros Códigos**
    - 4.1 Códigos de barras**
    - 4.2 Códigos QR**
  - 5 Ejercitación**
-

## Capítulo 2

# Códigos Numéricos y Alfanuméricos

## 1 Códigos

### 1.1 Introducción

Definición: un código es una ley de correspondencia biunívoca entre los elementos de dos conjuntos.

Un código es una representación biunívoca de algún conjunto de elementos de tal forma que, a cada uno de estos, se le asigna una combinación de símbolos determinados y viceversa, como se observa en la Figura 2.1.

Para “comunicarse” hay que conocer el código utilizado.

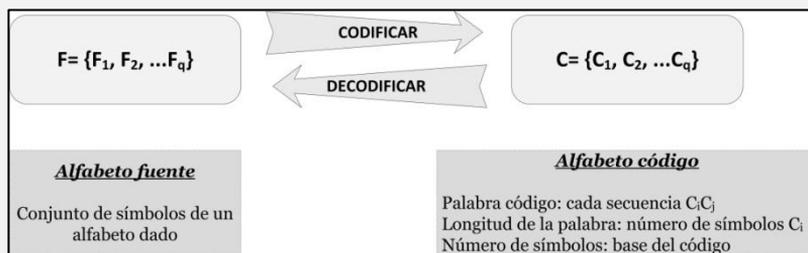


Fig. 2.1. Relación biunívoca de un código.

Códigos numéricos: cuando los elementos de los dos conjuntos son números.

### 1.2 Códigos binarios

Los sistemas de numeración (binario, octal, hexadecimal, etc.) estudiados constituyen códigos de representación de las cantidades.

En los CÓDIGOS BINARIOS, la base del código es 2. Estos códigos son los utilizados por los sistemas digitales.

El sistema de numeración binario, presentado en la Tabla 2.1, recibe el nombre de código binario natural:

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

Tabla 2.1. Código binario natural.

Con  $n$  cifras binarias (bits), se pueden obtener  $2^n$  combinaciones diferentes. Cada combinación se puede asignar a un elemento (o cantidad) distinto.

Es decir, que:

- Un grupo de  $n$  bits binarios es un código de  $n$  bits que puede asumir  $2^n$  combinaciones distintas de 1's y 0's. Para que el código sea útil no se puede asignar ninguna combinación a más de un elemento, y varían de 0 a  $(2^n - 1)$
- Aunque el mínimo número de bits requeridos para escribir un código con  $2^n$  elementos distintos es  $n$ , el máximo número de bits no está especificado.
- Con estas combinaciones podríamos tener  $2^n!$  (las permutaciones de las  $2^n$  combinaciones) códigos distintos; aunque solo se verán algunos con características particulares.

Efectivamente, en el estudio de los códigos binarios, existen algunos muy conocidos por su aplicación computacional, como son los:

**Códigos binarios continuos (o progresivos):** son aquellos donde las combinaciones correspondientes a números decimales consecutivos son adyacentes. Se llaman combinaciones binarias adyacentes las que difieren solamente en un bit.

**Códigos binarios cíclicos:** son aquellos códigos continuos donde la última combinación es adyacente a la primera.

El Cuadro 2.1 muestra estos casos.

Decimal	Continuo	Cíclico
0	0000	0000
1	0001	0001
2	0011	0011
3	0111	0111
4	0110	1111
5	1110	1110
6	1111	1100
7	1110	1000

Cuadro 2.1. Códigos continuos y cíclicos.

El ejemplo más típico de un código continuo y cíclico es el llamado de GRAY, el que puede tener 2, 3, 4 o más bits de acuerdo a la necesidad. También es llamado “reflejado”, porque para construir el código de n bits, se “reflejan” las combinaciones del código Gray de n-1.

Se utiliza principalmente en convertidores muy rápidos y en codificadores. Como de una combinación binaria a la contigua sólo cambia un bit, se aumenta la velocidad y se elimina el riesgo de transiciones intermedias. Para ampliar palabras codificadas en este código, en un bit más, basta con repetir simétricamente las combinaciones (como si fuera un espejo) y agregar un bit a la izquierda, siendo cero para la primera mitad y uno para la segunda mitad de las combinaciones posibles.

En la Tabla 2.2, se representa el código de Gray para 2, 3 y 4 bits.

Decimal	Gray		
	2 bits	3 bits	4 bits
0	00	000	0000
1	01	001	0001
2	11	011	0011
3	10	010	0010
4		110	0110
5		111	0111
6		101	0101
7		100	0100
8			1100
9			1101
10			1111
11			1110
12			1010
13			1011
14			1001
15			1000

Tabla 2.2. Representación del código de Gray en 2, 3 y 4 bits.

Otro ejemplo de código binario continuo y cíclico es el de Johnson, que se observa en la Tabla 2.3 con 5 bits:

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

Tabla 2.3. Representación del código de Johnson de 5 bits.

### 1.3 Códigos BCD

Los números puros se representan con la notación octal o con la hexadecimal, por su facilidad de conversión. Sin embargo, la conversión

binaria a decimal es bastante difícil, y en las calculadoras, los juegos electrónicos, y los instrumentos digitales, en los que generalmente es común la entrada y salida de números en notación decimal, se emplea un código especial para representar esta notación. A este código se le llama CODIGO BCD: decimal codificado en binario (binary - coded - decimal).

Con esta técnica es posible convertir cada número decimal a su equivalente en binario, en lugar de convertir el valor decimal entero a su forma binaria pura. Para poder representar los diez dígitos decimales (del 0 al 9) son necesarios 4 bits. De las 16 combinaciones posibles ( $2^4$ ) que se obtienen con los 4 bits, sólo se utilizan diez.

Los códigos BCD se pueden clasificar en:

**Ponderados:** poseen una regla de formación que adjudica un cierto “peso” a los “unos” binarios según la posición que ocupan en el conjunto de los 4 bits, debiéndose verificar que la suma algebraica de los pesos de cada combinación sea igual al número decimal representado. Los ejemplos más típicos son el BCD natural (que tiene los pesos 8 4 2 1) y el BCD AIKEN (que tiene los pesos 2 4 2 1).

**Libres o no ponderados:** en estos la correspondencia decimal-binaria es arbitraria. No existen “pesos” ni sumas algebraicas, aunque en general las combinaciones se forman según ciertas reglas específicas para cada código. El ejemplo típico es el BCD Exceso 3 que se forma partiendo del BCD natural y sumándole 3 a cada combinación.

La Tabla 2.4 presenta estos tipos de códigos BCD.

Decimal	BCD Natural	BCD AIKEN	BCD Exceso 3
	8 4 2 1	2 4 2 1	n + 3
0	0000	0000	0011
1	0001	0001	0100
2	0010	0010	0101
3	0011	0011	0110
4	0100	0100	0111
5	0101	1011	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100
	Ponderados		Libre

Tabla 2.4. Tipos de códigos BCD.

La conversión de un número decimal a un código BCD se realiza simplemente expresando cada dígito mediante la combinación binaria que le corresponde de acuerdo al código especificado. Por ejemplo, en la Tabla 2.5 se observa el número decimal 926 representado en distintos códigos.

Número Decimal	9	2	6
BCD Natural	1001	0010	0110
BCD AIKEN	1111	0010	1100
BCD Exceso 3	1100	0101	1001

Tabla 2.5. Representación del número 926 en distintos códigos BCD.

## 1.4 Códigos alfanuméricos

Muchas aplicaciones de computadoras digitales, requieren manejar datos que consisten no solamente de números sino también de letras. Por ejemplo, una compañía de seguros con millones de clientes puede usar un computador digital para procesar sus historias. Para representar el nombre del dueño de una póliza en forma binaria, es necesario tener un código binario para el alfabeto. Además, el mismo código binario puede representar números decimales y algunos otros caracteres especiales.

Un código alfanumérico es un código binario de un grupo de elementos consistente de los diez números decimales, los 26 caracteres del alfabeto y de cierto número de símbolos especiales tales como el \$.

Uno de estos códigos es conocido como ASCII (American Standard Code for Information Interchange = Código normalizado americano para el intercambio de información).

A continuación, en la Tabla 2.6, se muestra una lista del código ASCII básico o estándar de 7 bits para los números, letras mayúsculas y minúsculas, y signos de puntuación.

ASCII incluye 256 códigos divididos en dos conjuntos, estándar y extendido, de 128 cada uno. Estos conjuntos representan todas las combinaciones posibles de 7 u 8 bits. El conjunto ASCII básico o estándar, utiliza 7 bits para cada código, lo que da como resultado 128 códigos de caracteres desde 0 hasta 127 (00<sub>H</sub> hasta 7F<sub>H</sub> hexadecimal). El conjunto ASCII extendido utiliza 8 bits para cada código, dando como resultado 128 códigos adicionales, numerados desde el 128 hasta el 255 (80<sub>H</sub> hasta FF<sub>H</sub> extendido).

Oct	Hex	Dec	Carácter	Oct	Hex	Dec	Carácter	Oct	Hex	Dec	Carácter	Oct	Hex	Dec	Carácter
0	00	0	NUL NULI	40	20	32		100	40	64	@	140	60	96	`
1	01	1	SOH Start Of Heading	41	21	33	!	101	41	65	A	141	61	97	a
2	02	2	STX Start of TeXt	42	22	34	"	102	42	66	B	142	62	98	b
3	03	3	ETX End of TeXt	43	23	35	#	103	43	67	C	143	63	99	c
4	04	4	EOT End of Transmission	44	24	36	\$	104	44	68	D	144	64	100	d
5	05	5	ENQ ENquiry	45	25	37	%	105	45	69	E	145	65	101	e
6	06	6	ACK ACKnowledge	46	26	38	&	106	46	70	F	146	66	102	f
7	07	7	BEL BELI	47	27	39	'	107	47	71	G	147	67	103	g
10	08	8	BS BackSpace	50	28	40	(	110	48	72	H	150	68	104	h
11	09	9	TAB horizontal TAB	51	29	41	)	111	49	73	I	151	69	105	i
12	0A	10	LF new Line Feed	52	2A	42	*	112	4A	74	J	152	6A	106	j
13	0B	11	VT Vertical Tab	53	2B	43	+	113	4B	75	K	153	6B	107	k
14	0C	12	FF new page From Feed	54	2C	44	,	114	4C	76	L	154	6C	108	l
15	0D	13	CR Carriage Return	55	2D	45	-	115	4D	77	M	155	6D	109	m
16	0E	14	SO Shift Out	56	2E	46	.	116	4E	78	N	156	6E	110	n
17	0F	15	SI Shift In	57	2F	47	/	117	4F	79	O	157	6F	111	o
20	10	16	DLE Data Link Escape	60	30	48	0	120	50	80	P	160	70	112	p
21	11	17	DC1 Device Control 1	61	31	49	1	121	51	81	Q	161	71	113	q
22	12	18	DC2 Device Control 2	62	32	50	2	122	52	82	R	162	72	114	r
23	13	19	DC3 Device Control 3	63	33	51	3	123	53	83	S	163	73	115	s
24	14	20	DC4 Device Control 4	64	34	52	4	124	54	84	T	164	74	116	t
25	15	21	NAK negative acknowledge	65	35	53	5	125	55	85	U	165	75	117	u
26	16	22	SYN SYNchronous idle	66	36	54	6	126	56	86	V	166	76	118	v
27	17	23	ETB End of Transmission. Block	67	37	55	7	127	57	87	W	167	77	119	w
30	18	24	CAN CAncel	70	38	56	8	130	58	88	X	170	78	120	x
31	19	25	EM End of Medium	71	39	57	9	131	59	89	Y	171	79	121	y
32	1A	26	SUB SUBstitute	72	3A	58	:	132	5A	90	Z	172	7A	122	z
33	1B	27	ESC ESCape	73	3B	59	;	133	5B	91	[	173	7B	123	{
34	1C	28	FS File Separator	74	3C	60	<	134	5C	92	\	174	7C	124	
35	1D	29	GS Group Separator	75	3D	61	=	135	5D	93	]	175	7D	125	}
36	1E	30	RS Record Separator	76	3E	62	>	136	5E	94	^	176	7E	126	~
37	1F	31	US Unit Separator	77	3F	63	?	137	5F	95	_	177	7F	127	DELETE

Tabla 2.6. Código ASCII estándar de 7 bits.

En el conjunto de caracteres ASCII básico, los primeros 32 valores están asignados a los códigos de control de comunicaciones y de periféricos —caracteres no imprimibles, como retroceso, retorno de carro y tabulación— empleados para controlar la forma en que la información es transferida desde una computadora a otra o desde una computadora a una impresora. Los 96 códigos restantes se asignan a los signos de puntuación corrientes, a los dígitos del 0 al 9, y a las letras mayúsculas y minúsculas del alfabeto latino.

Los códigos de ASCII extendido, del 128 al 255, se asignan a conjuntos de caracteres que varían según los fabricantes de computadoras y programadores de software. Estos códigos no son intercambiables entre los diferentes programas y computadoras como los caracteres ASCII estándar. Por ejemplo, IBM utiliza un grupo de caracteres ASCII extendido que suele denominarse conjunto de caracteres IBM extendido para sus computadoras personales. Apple Computer utiliza un grupo similar, aunque diferente, de caracteres ASCII extendido para su línea de computadoras Macintosh. Por ello, mientras que el conjunto de caracteres ASCII estándar es universal en el hardware y el

software de los microordenadores, los caracteres ASCII extendido pueden interpretarse correctamente sólo si un programa, computadora o impresora han sido diseñados para ello.

## 2 Códigos Detectores y Correctores de Error

### 2.1 Introducción

Una vez definida la conexión física para poder transferir información entre los dispositivos o sistemas debe existir un formato para los datos y una estrategia de sincronización de cómo se envían y reciben los mensajes, incluyendo la detección y corrección de los errores.

En la Figura 2.2 se esquematiza un enlace de comunicaciones de datos, con bloques que cumplen diferentes funciones.



Fig. 2.2. Esquema de un enlace de datos.

DTE: Equipo Terminal de Datos

DCE: Equipo de Comunicación de Datos

La transferencia ordenada de información en un enlace de comunicación se logra estableciendo una conexión entre los DTE, identificando el emisor y el receptor, asegurando que todos los mensajes se transfieran correctamente sin errores, controlando toda la transferencia de información.

Los equipos (teléfono, transmisor, modem, etc.), las conexiones, los cables, repetidoras, etc., constituyen el soporte físico que permiten el enlace de datos.

Un elemento básico a considerar es la estructura del mensaje, constituyendo una unidad de información que puede llamarse Trama, Segmento, o Datagrama, según el protocolo de comunicaciones asociado. En general, el mensaje tiene la estructura de la Figura 2.3.



Fig. 2.3. Estructura general de mensaje de un protocolo.

Teniendo en cuenta que los equipos y el canal están expuestos a ruidos internos y externos que pueden alterar la información que se transmite, resulta de fundamental importancia detectar y corregir errores. Hay varias formas de verificar los errores que se producen en el mensaje durante una transmisión.

Especialmente, cuando la información es numérica, no se aceptan errores, por lo que es necesario poder detectar la presencia de los mismos si se han producido. Si se detecta la presencia de un error es necesario retransmitir la información, mientras que si se puede detectar y corregir el error producido, no se produce la retransmisión.

Cuando en un código binario se utilizan todas las combinaciones posibles, no se pueden detectar errores, debido a que si cambia algunos de sus bits por error de una combinación válida, daría por resultado otra combinación de bits válida. Por lo tanto, podemos decir que “para detectar errores es necesario no utilizar todas las combinaciones posibles”. Esto último es una condición necesaria pero no suficiente.

Para poder efectivamente detectar y/o corregir errores se da el concepto de distancia entre combinaciones de un código:

**distancia** de un código entre dos combinaciones binarias cualesquiera es el número de bits que deben cambiarse para pasar de una combinación a la otra.

## 2.2 Distancia mínima

La Distancia Mínima ( $D_m$ ) de un código es la menor de las distancias entre dos combinaciones cualesquiera pertenecientes al código. La distancia mínima de los códigos vistos hasta ahora es la unidad, y por lo tanto no se pueden detectar errores.

Por lo anterior, se deduce que, para poder detectar y/o corregir errores, es imprescindible que:

$$Distancia\ mínima\ (D_m) > 1$$

El razonamiento, esquematizado en la Figura 2.4, nos da una idea de la capacidad de detección y corrección de errores de un código:

$D_m = 2$	$D_m = 3$	$D_m = 5$
Combinación $C_i$	Combinación $C_i$	Combinación $C_i$
Difiere en 1 bit de $C_i$ y $C_j$	Difiere en 1 bit de $C_i$ y 2 de $C_j$	Difiere en 1 bit de $C_i$ y 4 de $C_j$
Combinación $C_j$	Difiere en 2 bits de $C_i$ y 1 de $C_j$	Difiere en 2 bits de $C_i$ y 3 de $C_j$
	Combinación $C_j$	Difiere en 3 bits de $C_i$ y 2 de $C_j$
		Difiere en 4 bits de $C_i$ y 1 de $C_j$
		Combinación $C_j$

Fig. 2.4. Esquema sobre la capacidad de detección y corrección de error de un código.

$C_i$  y  $C_j$  son combinaciones válidas del código considerado, el resto son combinaciones que difieren en 1 o más bits de las válidas (combinaciones no válidas).

Para el caso de  $D_m = 2$ , se concluye que sólo será posible detectar el error en 1 bit, ya que una combinación no válida siempre diferirá en 1 bit de dos combinaciones válidas, lo que imposibilitaría determinar de cual combinación válida proviene, es decir, no se puede corregir. Para el caso de  $D_m = 3$ , será posible detectar el error en 2 bits, pero sólo corregir 1 bit, dado que una combinación no válida siempre diferirá en 1 bit de una combinación válida y dos bits de otra. Esto permitiría determinar de cual combinación válida proviene, bajo el supuesto que lo más probable es que se produzca error en 1 bit y no en 2, es decir, se puede corregir 1 bit. Para el caso de  $D_m = 5$  se concluye que será posible detectar error de hasta cuatro bits y corregir hasta dos bits.

Lo mencionado nos permite deducir que:

$$\begin{aligned} \text{CANTIDAD de ERRORES DETECTABLES} + 1 &= D_m \\ (\text{CANTIDAD de ERRORES CORREGIBLES} * 2) + 1 &= D_m \end{aligned}$$

Comentario: Una aproximación válida para un Sistema de Comunicaciones es que si  $P_1$  es la probabilidad de error en un bit, la

probabilidad de error en dos bits es  $P_2 = P_1^2$  (por ejemplo, si  $P_1 = 10^{-6}$  entonces  $P_2 = 10^{-12}$ ), es decir, las probabilidades de errores de más bits, disminuyen abruptamente.

## 2.3 Códigos detectores de error

Para poder detectar errores debemos lograr que la  $D_m > 1$ . La manera más simple de lograrlo es contar el número de 1's (unos binarios) contenidos en el mensaje y agregarle un dígito binario (un bit) de forma tal que el mensaje tenga un número par de 1's (o un número impar de 1's). A esto se le llama código con paridad, y al uno que se adiciona bit de paridad. Si el número de 1's es par se trabaja con paridad par y si no es impar. Con este procedimiento, logramos un código resultante de  $D_m = 2$  (obviamente a costa de agregar un bit). Antes de enviar una información es necesario ponerse de acuerdo con cuál de las dos convenciones se trabaja. En el Cuadro 2.2 se presenta un ejemplo usando el código BCD natural.

Esta característica se puede realizar con cualquier código con el cual se trabaje.

La detección de errores en estos códigos consiste en comprobar si el número de unos de cada combinación cumple con la convención adaptada. Es decir, si se trabaja con paridad par, se debe chequear que el dato recibido cumpla con esta condición; si no es así existe un error, y se debe solicitar una nueva transmisión del dato.

Decimal	BCD Natural	BCD Natural c/paridad par	BCD Natural c/paridad impar
0	0000	0000 0	0000 1
1	0001	0001 1	0001 0
2	0010	0010 1	0010 0
3	0011	0011 0	0011 1
4	0100	0100 1	0100 0
5	0101	0101 0	0101 1
6	0110	0110 0	0110 1
7	0111	0111 1	0111 0
8	1000	1000 1	1000 0
9	1001	1001 0	1001 1

Bit de Paridad

La distancia entre dos combinaciones cualesquiera es mayor que 1

Cuadro 2.2. Ejemplo de códigos con bit de paridad usando BCD natural.

Otros códigos detectores de error son los de cantidad constante de 1's o peso constante. Entre ellos encontramos el código 2 entre 5 y el código biquinario, que se observan en la Tabla 2.6.

Decimal	2 entre 5	Biquinario
	<i>7421P</i>	<i>5043210</i>
0	11000	0100001
1	00011	0100010
2	00101	0100100
3	00110	0101000
4	01001	0110000
5	01010	1000001
6	01100	1000010
7	10001	1000100
8	10010	1001000
9	10100	1010000

Tabla 2.7. Ejemplos de códigos de peso constante.

Estos códigos tienen dos unos en cada combinación (o sea paridad par) y se forman de acuerdo a los pesos asignados.

## 2.4 Códigos correctores de error

Un código corrector detecta si la información codificada presenta o no errores, y en caso afirmativo determina la posición del bit o bits erróneos, de manera de poder corregirlos por inversión (recordar que estamos usando el sistema binario, por lo que si un bit tiene error, se cambia por su complemento).

Como ya vimos, los códigos de distancia mínima dos ( $D_m=2$ ) no permiten la corrección de errores, porque al producirse un error la combinación obtenida puede poseer dos adyacentes pertenecientes al código, y no es posible saber cuál es la correcta. Por ejemplo, si del código 2 entre 5 se detecta la combinación errónea 01000 (que no es válida), no es posible conocer si el error se produjo en el primer bit (01001) o en el segundo bit (01010), ya que ambas combinaciones son válidas. Por lo tanto, se deduce que para corregir un bit de error es necesario una distancia mínima mayor a dos.

Los códigos con  $D_m > 2$  se llaman códigos correctores de error. Entre ellos, se destacan el Código Hamming, los códigos Bidimensionales y los Códigos de Redundancia Cíclica (CRC).

## 2.4.1 Código de Hamming

Su formación parte de un código cualquiera de  $n$  bits al que se le adicionan  $p$  bits formando un nuevo código de  $n + p$  bits. En este código se realizan  $p$  detecciones de paridad, obteniéndose un bit de paridad uno o cero (dependiendo si el número de bits es par o impar). El conjunto de los  $p$  bits de paridad forma un número en el sistema binario natural cuyo equivalente decimal nos indica la posición del bit erróneo. Si no hay error, el número obtenido debe ser cero.

Dado que con  $p$  bits se obtienen  $2^p$  combinaciones, se debe cumplir la siguiente relación:

$$2^p \geq n + p + 1$$

Como ejemplo tomaremos como código base el código AIKEN, al cual queremos poder detectar y corregir errores de un bit. Como  $n = 4$  hay que agregarle  $p = 3$  bits para cumplir con la condición anterior. El código resultante tendrá 7 bits.

Para detectar los siete posibles errores que se pueden producir en una combinación (7 errores considerando 7 posiciones en cada combinación) y la ausencia de error, son necesarias ocho combinaciones binarias correctoras de error. Dichas combinaciones correctoras de error se obtienen mediante 3 bits ( $c_1$ ,  $c_2$  y  $c_3$ ), y el número decimal formado por ellos indica la posición del bit erróneo.

Las combinaciones correctoras de estos bits se observan en la Tabla 2.8.

Condición de error	$c_3$	$c_2$	$c_1$
Sin errores	0	0	0
Error en el b1	0	0	1
Error en el b2	0	1	0
Error en el b3	0	1	1
Error en el b4	1	0	0
Error en el b5	1	0	1
Error en el b6	1	1	0
Error en el b7	1	1	1

Tabla 2.8. Tabla correctora del código de Hamming.

El bit  $c_1$  toma el valor 1 si se produce un error en los bits  $b_1$ ,  $b_3$ ,  $b_5$  y  $b_7$  de la combinación del código formado. Si el número de unos existentes en esas cuatro posiciones es siempre par, un error en uno cualquiera de esos cuatro bits lo convierte en impar. Por lo tanto,  $c_1$  vale uno si el número de unos en las posiciones dadas es impar, y cero en caso contrario. Es posible representar esta conclusión a través de una función lógica:

$$C_1 = b_1 \oplus b_3 \oplus b_5 \oplus b_7$$

Dónde  $\oplus$  es el símbolo de la función OR-Exclusiva (que se estudiará posteriormente), y que da como resultado 0 si el número de unos es par, y uno si el número de unos es impar. De igual manera obtenemos  $c_2$  y  $c_3$ :

$$C_2 = b_2 \oplus b_3 \oplus b_6 \oplus b_7$$

$$C_3 = b_4 \oplus b_5 \oplus b_6 \oplus b_7$$

Para generar los 3 bits ( $p$  bits) a agregarle al código general consideremos que los bits  $b_1$ ,  $b_2$  y  $b_4$  aparecen una sola vez en las fórmulas anteriores, con lo cual los generamos a partir de los otros, ya que por ejemplo  $b_1$  debe valer uno si el número de unos de  $b_3$ ,  $b_5$  y  $b_7$  es impar; por lo tanto:

$$b_1 = b_3 \oplus b_5 \oplus b_7$$

$$b_2 = b_3 \oplus b_6 \oplus b_7$$

$$b_4 = b_5 \oplus b_6 \oplus b_7$$

En el Cuadro 2.3 se presenta el Código Hamming deducido a partir del Código AIKEN y la Tabla de corrección.

Nº Decimal	b7 b6 b5	b4	b3	b2 b1
0	0 0 0	0	0	0 0
1	0 0 0	0	1	1 1
2	0 0 1	1	0	0 1
3	0 0 1	1	1	1 0
4	0 1 0	1	0	1 0
5	1 0 1	0	1	0 1
6	1 1 0	0	0	0 1
7	1 1 0	0	1	1 0
8	1 1 1	1	0	0 0
9	1 1 1	1	1	1 1

b3, b5, b6 y b7 son los bits del código original Aiken, mientras que los bits restantes se forman de acuerdo a las condiciones de paridad (en este caso paridad par)

Cuadro 2.3. Ejemplo de código Hamming.

## 2.4.2 Verificación de redundancia LRC y CRC

Resulta de interés la utilización de códigos cíclicos, que son aquellos que al realizar una rotación cíclica de una palabra se produce otra palabra que pertenece al mismo código. La ventaja de emplear códigos cíclicos es que su generación se puede implementar fácilmente a partir del empleo de registros de desplazamientos con realimentación.

### Verificación de Redundancia Longitudinal (LRC)

Es un código detector de error de un bit. Utilizado en la adquisición de datos y control en la industria, para la comunicación entre dispositivos y con un PC.

En la Figura 2.5 se observa un ejemplo de una trama con información en ASCII:

3A	30	30	30	30	36	36	33	0D
	31	38	30	30	31	32	34	0A
	← Datos →						← LRC →	

Fig. 2.5. Ejemplo de una trama con información ASCII.

Dónde:

3A: identifica el comienzo de la trama.

0D es CR y 0A es LF: identifican el final de la trama.

El resto de la trama tiene un binario equivalente:

El cálculo del LRC consiste en realizar la suma hexadecimal de los datos y calcular el complemento a 2.

$$\begin{array}{r}
 01 + 08 + 00 + 00 + 61 + 62 = CC = \quad 0011 \quad 0011 \\
 \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad + 1 \\
 \hline
 \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad 0011 \quad 0100
 \end{array}$$

en hexadecimal: 33 34. Luego, LRC = 33 34

### **Verificación de Redundancia Cíclica (CRC)**

Estos códigos consisten en agregar k bits a un mensaje a transmitir de m bits. La condición que se cumple es que  $m \gg k$ . De esta forma se obtiene una transmisión eficiente ya que la cantidad de bits agregados es relativamente pequeña. La utilización de estos códigos permite la detección y corrección de errores.

La forma de obtener los k bits a agregar es mediante un procedimiento de redundancia cíclica implementado con Registros de Desplazamiento y compuertas XOR (Capítulos 4 y 5), fundamentado en una aritmética binaria sin acarreo.

En la Aritmética Binaria sin acarreo (también llamada aritmética de Módulo 2) se realizan las operaciones algebraicas (suma, resta, multiplicación, etc.) sin considerar los acarreo, por ejemplo:

$$\begin{array}{r}
 \quad 1011 \\
 + \quad 1001 \\
 \hline
 \quad 0010
 \end{array}
 \qquad
 \begin{array}{r}
 \quad 1011 \\
 - \quad 1001 \\
 \hline
 \quad 0010
 \end{array}$$

Se observa que la suma y la resta en Módulo 2 dan el mismo resultado. Este puede obtenerse realizando la función XOR bit a bit, simplificando la implementación de sumadores/restadores.

Si se suman o se restan dos números iguales el resultado da 0 (cero). Esta particularidad se usa en el cálculo de los k bits a agregar. A los k bits se los llama FCS (Frame Check Secuency). Se calculan de la siguiente manera:

- M: mensaje a transmitir de m bits
- K: cantidad de bits a agregar (FCS)

El transmisor, antes de transmitir, procesa (usando Módulo 2) el mensaje a transmitir M:

$$\frac{M2^k}{G(x)} = C + \frac{FCS}{G(x)}$$

- Observe que el FCS es el resto de la división y tendrá k bits ya que G(x) tiene k+1 bits.
- G(x): es un patrón de k + 1 bits, obtenido a partir de un polinomio de numeración de k+1 términos enteros. Además, cumple con que el bit más significativo y el menos significativo (MSB y LSB) son uno (1). Estos G(x) están normalizados y dependiendo cuál se usa, resultan CRC que permiten corregir más o menos bits.
- El transmisor envía  $M2^k + FCS$

El Receptor, recibe  $M2^k + FCS$ , y procesa (usando Módulo 2) lo recibido:

$$\begin{aligned} \frac{M2^k + FCS}{G(x)} &= \frac{M2^k}{G(x)} + \frac{FCS}{G(x)} = \\ &= C + \frac{FCS}{G(x)} + \frac{FCS}{G(x)} = \\ &= C + \frac{FCS + FCS}{G(x)} \end{aligned}$$

- Observe que si no hubo error  $FCS + FCS$  debería ser 0 (cero)

Si el resto de la división que realizó el receptor no es cero, ha habido error en la transmisión. Para corregir el error, el receptor debe realizar un procesamiento de lo recibido. Puede ser por software o por hardware (más rápido).

Los polinomios generadores G(X) están normalizados internacionalmente, por ejemplo:

- CRC-8 =  $X^8 + X^5 + X^4 + 1 = 100110001$   
(Redundancia de 8 bits)
- CRC-12 =  $X^{12} + X^{11} + X^3 + X^2 + X^1 + 11$   
(Redundancia de 16 bits)

- $CRC-16 = X^{16} + X^{15} + X^2 + 1$   
(Redundancia de 16 bits)

### 2.4.3 Códigos bidimensionales

La utilización de estos códigos implica la transferencia de un bloque de información organizada en dos dimensiones. El transmisor organiza esta estructura bidimensional y le agrega bits adicionales. El receptor recibe la información más los bits adicionales, los procesa, detecta y corrige los errores.

Esta organización bidimensional se observa en la Figura 2.6.

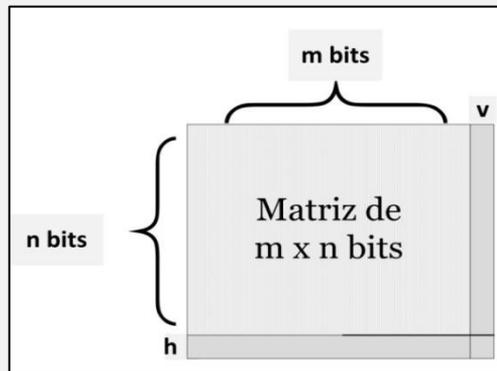


Fig. 2.6. Organización de códigos bidimensionales.

Las filas de la matriz se interpretan como un código binario cuya  $D_m = 1$ . Asimismo, las columnas de la matriz también se interpretan como otro código binario de  $D_m = 1$ . El transmisor agrega bits a las filas (bits  $v$ ) y a las columnas (bits  $h$ ) de forma tal que las distancias mínimas sean mayores a 1. Por ejemplo, en la Figura 2.7 se presenta el Código Bidimensional cuando al código correspondiente a las filas se agregan bits Hamming y al código de las columnas se le agrega un bit de paridad.

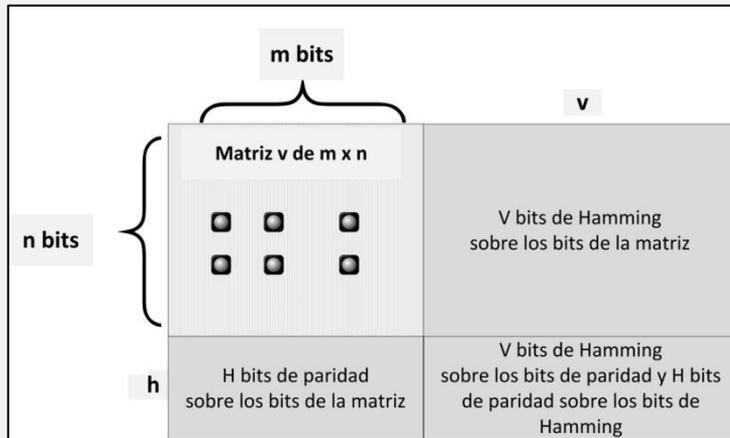


Fig. 2.7. Organización de códigos bidimensionales usando Hamming y bits de paridad.

El código de las filas tendrá una  $D_m = 3$  ya que es un Hamming, el código de las columnas tendrá una  $D_m = 2$  ya que es un código de paridad.

A fin de determinar la  $D_m$  del código Bidimensional podríamos encontrar un patrón de bits de error que no sea detectado por el receptor. Esto se basa en interpretar el concepto de  $D_m$  como la cantidad de bits que deben cambiar en una combinación válida de un código, para encontrar otra combinación válida. Por ejemplo, si la  $D_m$  de un código es 4, partiendo desde una combinación del código, deberíamos cambiar 4 bits de la misma para encontrar otra combinación válida.

En el caso del ejemplo de la figura, el patrón de bits de error no detectado por el receptor es el dibujado. En el caso del código horizontal habrá que cambiar 3 bits, en el caso del vertical sólo 2 bits. Por lo tanto, la  $D_m = 6$ . Se puede generalizar y concluir que:

$$D_{m\text{bidimensional}} = D_{m\text{filas}} D_{m\text{columnas}}$$

Para el cálculo de los bits del rectángulo inferior derecho donde aparecen bits de Hamming y de paridad simultáneamente, se puede afirmar el determinismo, ya que ambos cálculos parten del mismo juego de bits (matriz  $m \times n$ ). Por lo tanto, las paridades deben coincidir.

Se pueden extender los conceptos para códigos tridimensionales, y más aún, para códigos multidimensionales, que consistirán en realizar el ordenamiento de los bits a transmitir en tantas direcciones como se quiera. Este es un procedimiento utilizado para aumentar la distancia

mínima, pero tiene el inconveniente de bajar la eficiencia representativa. Además, el procesamiento en el receptor aumenta críticamente. Por ejemplo, en el caso de la figura ( $D_m = 6$  para corregir 2 bits) aproximadamente el 15% de lo transmitido consiste en bits agregados.

### 3 Encriptación o Cifrado

Con la introducción de las computadoras, especialmente en las empresas, fue evidente la necesidad de herramientas automáticas para proteger los archivos y otras informaciones almacenadas en su memoria. El nombre genérico del tema que trata las herramientas diseñadas para proteger los datos y frustrar a los usuarios no autorizados informáticos es la Seguridad en Computadoras. Esta unidad temática ha necesitado desarrollarse debido a la introducción de redes y facilidades de comunicación para transportar datos. La tecnología esencial en todas las redes automáticas y en las aplicaciones de seguridad en computadoras es la Encriptación o Cifrado.

La medida más efectiva en contra de la amenaza de usuarios no autorizados es el encriptado o cifrado de datos, que debe interpretarse como el almacenamiento o transmisión de datos sensibles en forma cifrada. Dentro de la terminología utilizada en este campo, se destaca el nombre de texto plano asignado a los datos originales. El texto plano es cifrado sometiéndolo a un algoritmo de cifrado, cuyas entradas son el texto plano y la clave de cifrado, y a la salida de este algoritmo (la forma cifrada del texto plano) se le llama texto cifrado.

Aunque los detalles del algoritmo normalmente son de dominio público, la clave de cifrado se mantiene en secreto. El texto cifrado, que debe ser ininteligible para cualquiera que posea la clave de cifrado, es lo que se guarda en las computadoras principales de la organización o se transmite por las líneas de comunicaciones de las redes. El esquema empleado debería ser tal que el trabajo involucrado en romperlo sobrepase cualquier ventaja potencial que pudiera obtenerse al hacerlo.

Existen dos técnicas diferenciadas que pueden utilizar los algoritmos: la sustitución y la permutación. La sustitución es uno de los enfoques básicos del cifrado, como se practica tradicionalmente, donde se usa una clave de cifrado para determinar, para cada carácter del texto plano, un carácter de texto cifrado que va a sustituir a ese carácter. Con la técnica de permutación, los caracteres de texto plano son simplemente reorganizados en una secuencia diferente, bajo la influencia de la clave de cifrado.

Por ejemplo, podríamos usar una técnica de sustitución, en un algoritmo elemental, para cifrar el siguiente texto plano y clave de cifrado:

Texto plano: ESTE ES UN TEXTO DEMO

Clave: PRUEBA

Suponemos, por simplicidad, que los únicos caracteres de datos que tenemos que manejar son las letras mayúsculas y los espacios en blanco. Y que el algoritmo de cifrado por sustitución sea el siguiente:

1. Dividimos el texto plano en bloques de longitud igual a la clave de cifrado.

E S T E + E S + U N + T E X T O + D E M O  
(los espacios en blanco son mostrados explícitamente como “+”).

2. Reemplazamos cada caracter del texto plano por un entero que esté en el rango de 00 a 26, usando espacio en blanco = 00, A = 01, ....., Z = 26.

E S T E + E S + U N + T E X T O + D E M O  
05192005 00 0519 00 2114 00 2005242115 00 04051315

3. Repetimos el paso 2 para la clave de cifrado.

P R U E B A  
16 18 21 05 02 01

4. Para cada bloque de texto plano reemplazamos cada caracter por la suma módulo 27 de su codificación de enteros más la codificación de enteros del carácter correspondiente de la clave de cifrado:

05192005 00 0519 00 2114 00 2005242115 00 04051315  
16182105 02 0116 18 2105 02 0116182105 02 01161821  
=====

21001410 02 0608 18 1519 02 2121151520 02 05210409

5. Reemplazamos cada codificación de enteros del resultado del paso 4 por su equivalente en caracteres:

U J N J B F H R O S B U U O O T B E U D I

El procedimiento de descifrado para este ejemplo es directo, siempre y cuando se tenga la clave. En este caso, pareciera que no sería tan difícil para un posible infiltrado determinar la clave sin ningún

conocimiento previo, teniendo el texto plano y el texto cifrado. Aunque, es obvio que existen esquemas mucho más sofisticados.

Ninguna de las técnicas de sustitución y permutación es particularmente segura en sí misma, pero los algoritmos que combinan a las dos pueden proporcionar un alto grado de seguridad. Uno de estos algoritmos es el DES (Estándar de cifrado de datos) que usa clave de 64 bits. A través de los años muchas personas han sugerido que probablemente el DES no sea tan seguro para ciertas aplicaciones. Alternativamente, han aparecido otros algoritmos que han ampliado el tamaño de la clave, y/o usan dos claves (una para encriptar y otra para desencriptar) como el RSA.

## 4 Otros códigos

### 4.1 Códigos de Barras

El Código de Barras es un arreglo en paralelo de barras y espacios que contiene información codificada en las barras y espacios del símbolo. Esta información puede ser leída por dispositivos ópticos (lectores de código de barras), los cuales envían la información leída hacia una computadora como si la información fuera una entrada de teclado.

También el código de barras puede definirse como un conjunto de símbolos hechos de patrones de barras, y espacios blancos y negros.

Dentro de los códigos de barras se codifican bits de información. Los datos son leídos por scanners especiales de códigos de barras y se usan muy a menudo en conjunto con bases de datos. Los códigos de barras no requieren ingreso manual por el ser humano, ya que pueden ser leídos automáticamente por los scanners y son virtualmente libres de error

Los Scanners de códigos de barras leen el patrón de barras blancas y negras (o mejor dicho, claras y oscuras) y decodifican el código, convirtiéndolo en un “string de caracteres” que generalmente se guarda en una base de datos.

#### **Ventajas:**

Algunas de sus ventajas sobre otros procedimientos de colección de datos son:

- Se imprime a bajos costos
- Permite porcentajes muy bajos de error

- Los equipos de lectura e impresión de código de barras son flexibles y fáciles de conectar e instalar.

### **Beneficios:**

El código de barras es una técnica de entrada de datos, como son la captura manual, el reconocimiento óptico y la cinta magnética.

Se considera que la tecnología de código de barras es la mejor tecnología para implementar un sistema de colección de datos mediante identificación automática, y presenta muchos beneficios. Entre otros:

- Virtualmente no hay retrasos desde que se lee la información hasta que puede ser usada
- Se mejora la exactitud de los datos
- Se tienen costos fijos de labor más bajos
- Se puede tener un mejor control de calidad, mejor servicio al cliente
- Se pueden contar con nuevas categorías de información.
- Se mejora la competitividad.

### **Aplicaciones:**

Las aplicaciones del código de barras cubren prácticamente cualquier tipo de actividad humana, tanto en industria, comercio, instituciones educativas, instituciones médicas, gobierno, etc.

- Control de material en proceso
- Control de inventario
- Control de tiempo y asistencia
- Punto de venta
- Control de calidad
- Control de inventario
- Embarques y recibos
- Control de documentos
- Facturación
- Bibliotecas
- Bancos de sangre
- Hospitales
- Control de acceso
- Control de tiempo y asistencia

### **Características de un código de barras:**

Un símbolo de código de barras puede tener, a su vez, varias características, entre las cuales podemos nombrar:

- Densidad:

Es la anchura del elemento (barra o espacio) más angosto dentro del símbolo de código de barras. Está dado en mils (milésimas de

pulgada). Un código de barras no se mide por su longitud física sino por su densidad.

- WNR: (Wide to Narrow Ratio)

Es la razón del grosor del elemento más angosto contra el más ancho. Usualmente es 1:3 o 1:2.

- Quiet Zone:

Es el área blanca al principio y al final de un símbolo de código de barras. Esta área es necesaria para una lectura conveniente del símbolo.

### **Simbologías:**

Un símbolo de código de barras es la impresión física de un código de barras.

Una Simbología es la forma en que se codifica la información en las barras y espacios del símbolo de código de barras.

Existen diferentes simbologías para diferentes aplicaciones, cada una de ellas con sus propias características. Las principales características que definen una simbología de código de barras son las siguientes:

- Numéricas o alfanuméricas
- De longitud fija o de longitud variable
- Discretas o continuas
- Número de anchos de elementos
- Autoverificación.

En las Figuras 2.8 a 2.13 se presentan ejemplos de diferentes simbologías de códigos de barras.

### **EAN/UPC**

Comercio detallista, autoverificable, numérico, longitud fija.

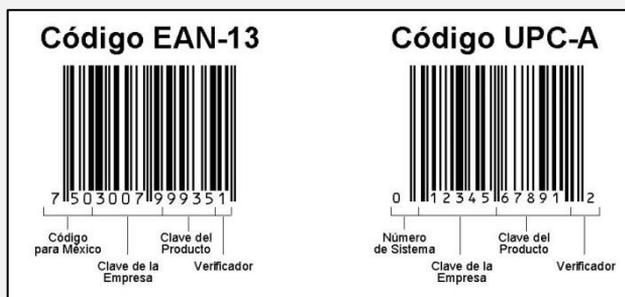


Fig. 2.8 Simbología EAN/UPC

Código 39

Industrial, alfanumérico, 44 caracteres



Fig. 2.9 Simbología Código 39

Codabar

Bancos de sangre, bibliotecas



Fig. 2.10 Simbología Codabar

I 2/5

Aplicaciones numéricas, aerolíneas, numérico



Fig. 2.11 Simbología I 2/5

### Código 93

Complementa al código 39, alfanumérico



Fig. 2.12 Simbología Código 93

### Código 128

Industrial, alfanumérico, 128 caracteres ASCII



Fig. 2.13 Simbología Código 128

## 4.2 Códigos QR

Un código QR es un código de barras bidimensional cuadrado que puede almacenar los datos codificados.

Hoy en día, los códigos QR se pueden ver en folletos, carteles, revistas, etc. Los códigos QR permiten interactuar con el mundo a través de smartphones.

Específicamente, un código QR extiende los datos a disposición de cualquier objeto físico y crean una medida digital para las operaciones de marketing. Esta tecnología permite y acelera el uso de servicios web para móviles: se trata de una herramienta digital muy creativa.



Fig. 2.14 Ejemplo de Código QR

Al escanear un código QR utilizando el teléfono inteligente, se obtiene un acceso inmediato a su contenido. El lector de código QR a continuación, puede realizar una acción, como abrir el navegador web para una URL específica. Además, pueden provocarse otras acciones, como el almacenamiento de una tarjeta de visita en la lista de contactos de su teléfono inteligente o conectarse a una red inalámbrica.

Los códigos QR se crearon en 1994 por Denso Wave, subsidiaria japonesa en el Grupo Toyota. El uso de esta tecnología es ahora libre y el más famoso de los códigos de barras 2D en el mundo. Se ha ganado su éxito en Japón desde la década de 2000, donde ahora es un estándar. En 2011, los japoneses escanearon diariamente más códigos QR que el número de SMS enviados. En 2010 los códigos QR comenzaron a expandirse en los EE.UU, y luego en Europa.

Los códigos QR se pueden personalizar, y por lo tanto, hacen posible que las marcas incorporen su identidad visual en sus códigos QR. Al personalizar, se deben seguir algunas reglas sobre la estructura de los códigos QR para que sigan siendo legibles.

En las Figuras 2.15 a 2.20 se presentan ejemplos de diferentes simbologías de códigos de barras.



Fig. 2.15 Ejemplo de Código QR usado en ticket de acceso a un evento

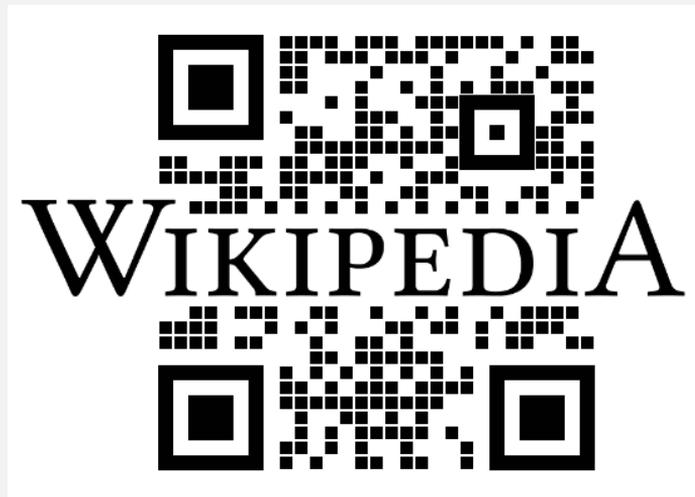


Fig. 2.16 Código QR usado por Wikipedia



Fig. 2.17 Ejemplo de Código QR usado en cartel comercial



Fig. 2.18 Ejemplo de Código QR



Fig. 2.19 Ejemplo de Código QR



Fig. 2.20 Ejemplo de Código QR

## 5 Ejercitación

### **Ejercicio 1:**

Realizar la tabla de un código Gray de 5 bits.

### **Ejercicio 2:**

Que cantidad de bits necesitaría en un código Gray; para codificar ángulos de 1 en 1 grados hasta 360 grados.

### **Ejercicio 3:**

Realizar la tabla de un código Jhonson de 6 bits. Indique que características presenta este código.

### **Ejercicio 4:**

Completar el cuadro, según los códigos indicados para la codificación de los números decimales enunciados. ¿Cuáles de los códigos son auto complementarios?.

Decimal	BCD 2421	BCD EXC3	BCD 3421	BCD 5421
7,25				
23,1				
67,5				
81				
95,8				
104,3				
237				
982,99				

**Ejercicio 5:**

Representar el número 927 en binario natural y en BCD EXS 3. Comentar el resultado luego de efectuar un análisis comparativo sobre la facilidad para obtener las representaciones y la longitud de bits necesarios para cada caso.

**Ejercicio 6:**

Indicar cuál es la distancia mínima del código BCD Aiken. Obtener a partir de él un código de paridad impar con la incorporación de un bit de paridad. ¿Cuál es la distancia mínima del código resultante ?.

**Ejercicio 7:**

Realice la tabla del código Hamming para la detección y corrección de un bit, tomando como código base de información el BCD 3421.

**Ejercicio 8:**

¿Cuántos bits tendrá el código Hamming para poder detectar y corregir un error si los datos originalmente se codifican con combinaciones de:

- a) 5 bits
- b) 8 bits
- c) 12 bits?

**Ejercicio 9:**

Indicar las distintas combinaciones binarias asignadas a cada uno de los siguientes números, caracteres ó símbolos especiales, en el código ASCII de 7 bits.

0 ; % ; , ; G ; ) ; 3 ; + ; & ; . ; T ; ç ; z

**Ejercicio 10:**

Indicar a que números, caracteres ó símbolos especiales pertenecen las combinaciones del código ASCII de 7 bits si las mismas se representan con los siguientes números en octal.

75 ; 12 ; 105 ; 62 ; 52 ; 13 ; 74 ; 132 ; 55 ; 27

**Ejercicio 11:**

Dado el siguiente texto sométalo a un algoritmo de cifrado

UNIVERSIDAD TECNOLOGICA NACIONAL

Suponemos por simplicidad que los únicos caracteres de datos que tenemos que manejar son las letras mayúsculas y los espacios en blanco. Y que clave de cifrado la cadena de caracteres:

ESTUDIAR

**Ejercicio 12:**

Dado el siguiente texto que se encuentra codificado con la palabra clave: AVANTI, descifrelo utilizando el esquema anterior.

ÑLTNÑNSANCNIFJAZUIDQNMN

---

# CAPÍTULO 3

---

## Álgebra de Boole

- 1 Visión General del Álgebra de Boole**
    - 1.1 Introducción**
    - 1.2 Postulados**
    - 1.3 Teoremas**
  - 2 Funciones Lógicas**
    - 2.1 Introducción**
    - 2.2 Teoremas de funciones lógicas**
  - 3 Minimización de Funciones Lógicas**
    - 3.1 Introducción**
    - 3.2 Método de simplificación de Karnaugh**
  - 4 Compuertas Lógicas**
  - 5 Ejercitación**
-

## Capítulo 3

# Álgebra de Boole

## 1 Visión General del Álgebra de Boole

### 1.1 Introducción

Un Sistema, en un aspecto amplio, puede definirse como un conjunto de elementos que guardan una relación entre sí. A su vez, un elemento del sistema puede ser otro sistema (subsistema).

Los Sistemas se clasifican en:

SISTEMAS

NATURALES

.

.

ARTIFICIALES

.

.

ELÉCTRICOS

.

.

ELECTRÓNICOS

ANALÓGICOS

DIGITALES

COMBINACIONALES

SECUENCIALES

Un Sistema digital es aquel cuyos elementos son digitales (sólo pueden adoptar valores discretos). En la Unidad 1 se llegó a la conclusión que la base 2, para la elección de un sistema de numeración, era la más adecuada desde el punto de vista de la confiabilidad y el costo. Por esta razón los Sistemas Digitales trabajan con elementos físicos binarios (sólo pueden adoptar dos valores).

Para poder realizar el estudio de los Sistemas Digitales se necesita estudiar un álgebra binaria. El Álgebra de George Boole, que data de 1854, es sin dudas la más apropiada para nuestro fin. Claude Shannon en 1938 adaptó esta álgebra para la aplicación en sistemas digitales.

Seguidamente se estudia brevemente el Álgebra de Boole, las funciones booleanas y su minimización, finalmente las compuertas lógicas.

## 1.2 Postulados

Dentro de las álgebras de Boole, es de utilidad definir la propiedad bivalente. Es decir, álgebras que están compuestas por sólo dos elementos. Así, el álgebra es un conjunto de elementos binarios relacionados entre sí mediante las operaciones lógicas producto  $[.]$  y suma  $[+]$ , que cumplen con los siguientes postulados (las letras  $a, b, c$ , etc., indican variables binarias):

- 1) Existe el elemento identidad

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

- 2) Las dos operaciones cumplen con la propiedad conmutativa

$$a + b = b + a$$
$$a \cdot b = b \cdot a$$

- 3) Propiedad distributiva

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

- 4) Complementación o inversión lógica

$$a + \bar{a} = 1$$
$$a \cdot \bar{a} = 0$$

## 1.3 Teoremas

Algunos teoremas importantes son:

- 1) Dualidad: Toda igualdad lógica sigue siendo válida si se intercambian los operadores  $(+ \text{ y } \cdot)$  y los elementos de identidad  $(0 \text{ y } 1)$ . La simetría de los postulados demuestra este teorema.

2) El álgebra es un conjunto cerrado; es decir, los resultados de aplicar las operaciones lógicas a las variables, pertenecen al álgebra.

3) En el álgebra se cumple que

$$\begin{aligned}a + 1 &= 1 \\ a \cdot 0 &= 0\end{aligned}$$

4) Ley de Idempotencia

$$\begin{aligned}a + a &= a \\ a \cdot a &= a\end{aligned}$$

5) Ley de involución

$$\bar{\bar{a}} = a$$

6) Las operaciones lógicas son asociativas

$$\begin{aligned}a + (b + c) &= (a + b) + c \\ a \cdot (b \cdot c) &= (a \cdot b) \cdot c\end{aligned}$$

7) Absorción:

$$\begin{aligned}a &= a + (a \cdot b) \\ a &= a \cdot (a + b)\end{aligned}$$

8) Leyes de De Morgan

$$\begin{aligned}\overline{a + b + c + \dots + n} &= \bar{a} \cdot \bar{b} \cdot \bar{c} \dots \bar{n} \\ \overline{a \cdot b \cdot c \dots n} &= \bar{a} + \bar{b} + \bar{c} + \dots + \bar{n}\end{aligned}$$

Con excepción del teorema 1, siempre aparecen dos expresiones; obsérvese que la segunda es la dual de la primera. Se recomienda al alumno demostrar estos teoremas en forma algebraica basándose en los postulados.

Aún cuando las operaciones  $+$  y  $\cdot$  son distributivas entre sí, de ahora en más prescindiremos de los paréntesis que encierran los productos lógicos. Además el símbolo del producto no se indicará en lo sucesivo. De esta forma, por ejemplo, la expresión

$$a + (b \cdot c) \cdot (b + e)$$

se escribirá

$$a + bc(b + d)$$

## 2 Funciones Lógicas

### 2.1 Introducción

Una función lógica es una variable binaria que depende de otras variables binarias relacionadas entre sí por las operaciones lógicas. Una función lógica se nota de la siguiente manera:

$$f(a, b, c, \dots, n) = \{\text{expresión lógica que involucra a las variables } a, b, c, d, \dots, n\}$$

La función adoptará el valor 0 o 1 de acuerdo a la expresión y al valor determinado de las variables. Por ejemplo:

$$f(a, b, c) = a\bar{b} + ac$$

$f(a,b,c)$  es una función de tres variables a la cual le corresponde la Tabla de Verdad de la Tabla 3.1. Puede decirse que la tabla de verdad es otra forma de expresar una función lógica.

c	b	a	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Tabla. 3.1 Tabla de verdad de la función  $f(a,b,c)$

### 2.2 Teoremas de funciones lógicas

*Teorema I:* En el Álgebra de Boole se cumple

$$f(a, b, c, \dots, n) = af(1, b, c, \dots, n) + \bar{a}f(0, b, c, \dots, n)$$

Para demostrar esta igualdad basta con reemplazar  $a = 1$  y  $a = 0$  en la expresión y verificar que la misma se cumple en ambos casos. También, considerando que la función en cuestión no tiene restricciones, se puede decir que también es válida su dual:

$$f(a, b, c, \dots, n) = [a + f(0, b, c, \dots, n)][\bar{a} + f(1, b, c, \dots, n)]$$

Y se trata de una función general.

Este teorema posee corolarios muy útiles a la hora de simplificar (obtener una expresión más simple de la misma función) funciones (expresiones en general) lógicas. Se obtienen efectuando el producto miembro a miembro de la primera expresión por  $a$  o por  $\bar{a}$ , como se indica a continuación:

$$af(a, b, c, \dots, n) = a[af(1, b, c, \dots, n) + \bar{a}f(0, b, c, \dots, n)]$$

Aplicando propiedad distributiva al segundo miembro, se obtiene:

$$af(a, b, c, \dots, n) = af(1, b, c, \dots, n) \quad \text{Primer Corolario}$$

Ahora si multiplicamos miembro a miembro por  $\bar{a}$

$$\bar{a}f(a, b, c, \dots, n) = \bar{a}[af(1, b, c, \dots, n) + \bar{a}f(0, b, c, \dots, n)]$$

Aplicando propiedad distributiva al segundo miembro, se obtiene:

$$\bar{a}f(a, b, c, \dots, n) = \bar{a}f(0, b, c, \dots, n) \quad \text{Segundo Corolario}$$

Aplicando dualidad a los corolarios anteriores, se obtienen:

$$a + f(a, b, c, \dots, n) = a + f(0, b, c, \dots, n) \quad \text{Tercer Corolario}$$

y

$$\bar{a} + f(a, b, c, \dots, n) = \bar{a} + f(1, b, c, \dots, n) \quad \text{Cuarto Corolario}$$

*Teorema II:* Toda función lógica puede expresarse en forma canónica, es decir:

- Como una sumatoria de términos en los cuales aparecen todas sus variables en forma de producto lógico (estos términos se llaman MINTERMS)

- Como una productoria de términos en los cuales aparecen todas sus variables en forma de suma lógica (estos términos se llaman MAXTERMS).

En ambos casos la función se dice expresada en forma canónica y sus términos (ya sean minterms o maxterms se llaman términos canónicos).

Se demostrará este teorema para una función de dos variables  $f(a, b)$ , luego se hará extensivo para  $n$  variables.

Aplicando el *Teorema I* a  $f(a, b)$ , se tiene:

$$f(a, b) = af(1, b) + \bar{a}f(0, b)$$

Aplicando nuevamente el *Teorema I* a  $f(1, b)$  y a  $f(0, b)$ , se tiene:

$$\begin{aligned} f(1, b) &= af(1, b) + \bar{b}f(1, b) \\ f(0, b) &= af(0, b) + \bar{b}f(0, b) \end{aligned}$$

Remplazando en la expresión inicial se obtiene

$$f(a, b) = abf(1,1) + a\bar{b}f(1,0) + \bar{a}bf(0,1) + \bar{a}\bar{b}f(0,0)$$

Se observa entonces que toda función puede expresarse como una sumatoria de todos sus minterms, afectados cada uno de ellos por un coeficiente que consiste en el valor de la función (que se calcula remplazando las variables por 1 o por 0 si, en el minterm que acompaña, la variable correspondiente se encuentra directa o negada respectivamente).

Teniendo en cuenta que  $f(a, b)$  es una función cualquiera del álgebra de Boole, su dual también lo será, por lo tanto:

$$f(a, b) = [a + b + f(0,0)][a + \bar{b} + f(0,1)][\bar{a} + \bar{b} + f(0,1)][\bar{a} + \bar{b} + f(1,1)]$$

Análogamente, toda función puede expresarse como una productoria de todos sus maxterms, afectados cada uno de ellos por un coeficiente que consiste en el valor de la función (que se calcula remplazando las variables por 0 o por 1 si, en el maxterm que acompaña, la variable correspondiente se encuentra directa o negada respectivamente).

La generalización de los resultados obtenidos para funciones de  $n$  variables, resulta evidente.

A fin de obtener una notación más sencilla de las funciones lógicas, se suele asignar a cada término canónico un número decimal que se obtiene dando pesos a las variables de acuerdo a si las mismas se encuentran expresadas en forma directa o negada. El convenio se observa en la Tabla 3.2.

Variable	Peso
a	1
b	2
c	4
d	8
e	16
f	32
----	----

Tabla. 3.2. Pesos de las variables booleanas.

Si la variable aparece en forma negada, el peso asignado es cero. Por ejemplo, usando este convenio, el término canónico cualquiera  $\bar{a}b\bar{c}d$  correspondiente a un minterm de una función de cuatro variables, tendrá el número decimal 10.

El convenio mencionado permite una nueva forma, llamada compacta, de notar una función, a saber:

$$f(a, b, c, \dots, n) = \sum_{i=0}^{2^n-1} i f(i) = \prod_{i=0}^{2^n-1} (2^n - 1 - i) + f(i)$$

De la expresión anterior se deduce una regla para pasar de una función canónica en minterms a una en maxterms y viceversa: Se buscan los términos canónicos que no están en la expresión de la función, y se los complementa a  $2^n - 1$ . Estos serán los términos de la función buscada.

Por ejemplo, sea la función de 4 variables en minterms:

$$f(a, b, c, d) = \sum_4 (0, 1, 3, 5, 6, 7, 10, 13, 14, 15)$$

El 4 abajo del símbolo sumatoria indica la cantidad de variables

Los términos canónicos que no están son: 2, 4, 8, 9, 11 y 12. Sus complementos a 15 son: 13, 11, 7, 6, 4 y 3. Por lo tanto, la expresión canónica en maxterms de la función es:

$$f(a, b, c, d) = \prod_4(3, 4, 6, 7, 11, 13)$$

Nótese que, a modo de verificación, la suma del número de minterms y maxterms de una función, siempre es igual a  $2^n - 1$ .

## 3 Minimización de Funciones Lógicas

### 3.1 Introducción

Es importante obtener la mínima expresión posible de una función, esto es la menor cantidad de variables y operaciones involucradas. Los métodos de minimización se basan en los postulados del álgebra y a la conveniencia de agregar oportunamente términos en la expresión de la función.

Para aplicar los métodos es necesario que la función esté expresada en forma canónica. Como se vio en el punto anterior, toda función lógica es expresable en forma canónica, ya sea en minterms o maxterms.

Supóngase que una función canónica de 4 variables posee en su expresión los siguientes términos canónicos:

$$\dots + \bar{a}bc\bar{d} + abc\bar{d} + \dots$$

Se observa que puede sacarse factor común de la siguiente forma:

$$\dots + bc\bar{d}(\bar{a} + a) + \dots$$

Según el postulado 4,  $\bar{a} + a = 1$ , por lo tanto:

$$\dots + bc\bar{d}1 + \dots = \dots + bc\bar{d} + \dots$$

Se ha perdido la variable a.

Este procedimiento se sistematiza detectando todos los términos canónicos de la función que difieran en el estado (directo o negado) de

sólo una variable. Se saca factor común entre ellos y se van eliminando variables.

Sea el siguiente ejemplo:

$$f(a, b, c, d) = \sum_4 (0, 4, 8, 12)$$

La expresión algebraica de la misma es:

$$f(a, b, c, d) = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}d + \bar{a}bcd$$

Se ve que los dos primeros son adyacentes, como así también los dos últimos. Puede sacarse factor común:

$$f(a, b, c, d) = \bar{a}\bar{b}\bar{d}(c + \bar{c}) + \bar{a}bd(c + \bar{c}) = \bar{a}\bar{b}\bar{d} + \bar{a}bd$$

Los dos términos que quedan, si bien no canónicos, son adyacentes, quedando finalmente:

$$f(a, b, c, d) = \bar{a}\bar{b}\bar{d} + \bar{a}bd = \bar{a}\bar{b}(\bar{d} + d) = \bar{a}\bar{b}$$

## 3.2 Método de Simplificación de Karnaugh

E. W. Veitch, en 1952, propuso un método gráfico para la identificación de los términos adyacentes de una función. Posteriormente Maurice Karnaugh lo modificó tal como se conoce actualmente. Consiste en mapas aplicables a funciones de dos, tres, cuatro y cinco variables. Este método no resulta práctico para funciones de más de 5 variables. Para estos últimos casos, se usa un método numérico que no se estudia en este libro.

Un ejemplo de mapa de Karnaugh se muestra en la Figura 3.1. Se trata de un mapa para funciones de 4 variables.

Los dos números binarios en las columnas y las filas, que siguen un código Gray de dos variables, se corresponden con las variables directas o negadas de cada cuadro, y los números decimales son los asignados a cada término canónico según la convención indicada con anterioridad. Esta tabla genérica puede particularizarse para una función determinada marcando en la misma con un 1 los términos canónicos que forman parte de la función. De esta forma es sencillo identificar los términos canónicos adyacentes que serán los que limitan por los lados.

Por ejemplo, el término canónico 14, posee cuatros términos adyacentes que son: 6, 10, 12 y 15.

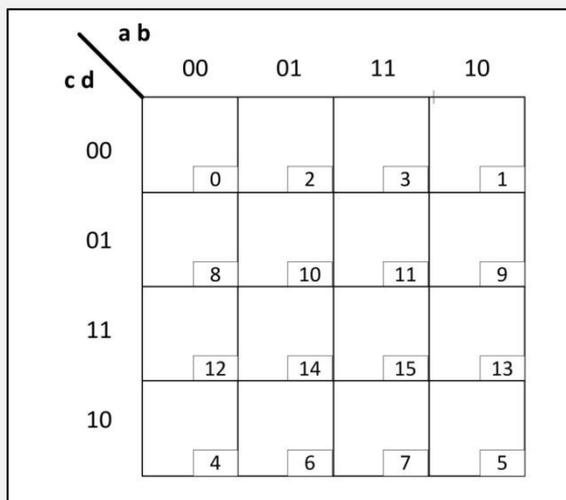


Fig. 3.1. Mapa de Karnaugh para funciones de 4 variables.

Formar un grupo entre dos unos colindantes en el mapa se corresponde con sacar factor común y perder la variable que cambia. Es de suponer la conveniencia de realizar los grupos que contengan mayor cantidad de unos en su interior. Pero esto debe seguir ciertas reglas.

Sea la función de 4 variables siguiente en minterms:

$$f(a, b, c, d) = \sum_4 (0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 14, 15)$$

El mapa que le corresponde es el indicado en la Figura 3.2.

El grupo 0-2 corresponde a sacar factor común con pérdida de la variable b. El grupo 3-1 pierde la variable b.

Se observa que estos dos grupos son adyacentes y se pueden juntar en un solo grupo 0-1-2-3, donde se pierden las variables a y b. El mismo razonamiento es válido para el grupo 8-9-10-11 que pierde las variables a y b. Estos grupos son adyacentes y podría formarse un solo grupo 0-1-2-3-8-9-10-11, donde sólo queda la variable c'. Para el grupo vertical de 8 unos se ha seguido el mismo procedimiento. Cabe aclarar que los términos canónicos 2, 3, 10 y 11 se han usado dos veces. Esto puede realizarse, ya que según el teorema 5, un término canónico podría repetirse cuantas veces se quiera sin alterar el valor de la función.

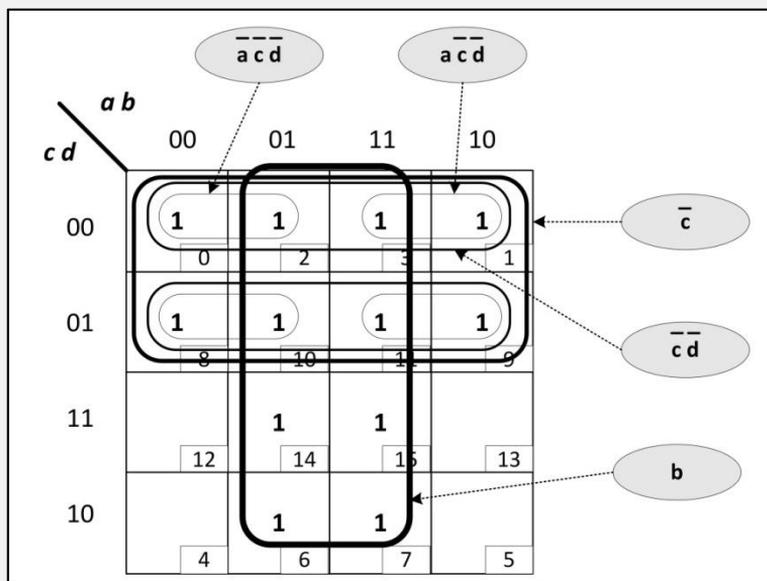


Fig. 3.2 Mapa de Karnaugh de la función ejemplo.

La función minimizada queda por lo tanto

$$f(a,b,c,d) = b + \bar{c}$$

Cabe aclarar que la última expresión es una suma porque la función inicial estaba en minterms, es decir, era una sumatoria.

De lo visto, pueden enunciarse la siguiente regla de formación de grupos:

- Se agrupan la mayor cantidad de unos posible, siempre que sean una potencia de dos y el grupo resultante pueda subdividirse en grupos menores.
- Se agrupan los unos restantes siguiendo la regla a), pudiendo usar (si es conveniente) un uno ya agrupado anteriormente
- Se repite b) hasta realizar todos los unos.

Para el caso de funciones de tres y de dos variables, las tablas son más pequeñas y la regla de formación de grupos es la misma. Se invita al alumno a sugerir cómo serían estas tablas y visitar el Práctico correspondiente resolviendo los ejercicios propuestos.

## 4 Compuertas Lógicas

La realización práctica (implementación) de las funciones lógicas se hace por medio de las compuertas lógicas, que son la base constructiva de la electrónica digital. No todas las funciones lógicas presentan interés práctico. En la Figura 3.3 se muestran las compuertas lógicas más comunes.

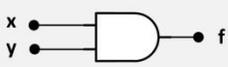
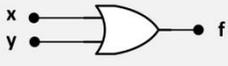
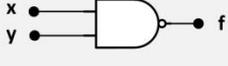
Nombre	Símbolo	Función	Tabla															
AND		$f=xy$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>f</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	f	0	0	0	0	1	0	1	0	0	1	1	1
x	y	f																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$f=x+y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>f</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	f	0	0	0	0	1	1	1	0	1	1	1	1
x	y	f																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NAND		$f=\overline{xy}$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>f</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	f	0	0	1	0	1	1	1	0	1	1	1	0
x	y	f																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$f=\overline{x+y}$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>f</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	f	0	0	1	0	1	0	1	0	0	1	1	0
x	y	f																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$f=x\bar{y}+\bar{x}y=x\oplus y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>f</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	y	f	0	0	0	0	1	1	1	0	1	1	1	0
x	y	f																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR		$f=xy+\bar{x}\bar{y}=x\odot y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>f</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	y	f	0	0	1	0	1	0	1	0	0	1	1	1
x	y	f																
0	0	1																
0	1	0																
1	0	0																
1	1	1																
NOT (inversor)		$f=\bar{x}$	<table border="1"> <thead> <tr> <th>x</th> <th>f</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	x	f	0	1	1	0									
x	f																	
0	1																	
1	0																	
Buffer		$f=x$	<table border="1"> <thead> <tr> <th>x</th> <th>f</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	x	f	0	0	1	1									
x	f																	
0	0																	
1	1																	

Fig. 3.3. Listado de compuertas lógicas más comunes.

En la figura aparecen compuertas de dos entradas. Existen compuertas de más entradas, disponibles comercialmente en circuitos integrados (chips) en SSI (Escala de Integración Pequeña). En función de

la cantidad de compuertas por chip, se suele clasificar a los CI en escalas de integración:

- SSI, escala de integración pequeña, hasta 10 compuertas por CI.
- MSI, escala de integración media, de 10 a 100 compuertas por CI.
- LSI, escala de integración grande, de 100 a 1000 compuertas por CI.
- VLSI, escala de integración muy grande, más de 1000 compuertas por CI.

A la hora de implementar una función lógica es cuando se torna importante la minimización. Por ejemplo, sea la función:

$$f(x, y, z) = \sum_3(2,4,5,6)$$

Si implementamos esta función sin minimizar, obtenemos el circuito de la Figura 3.4.

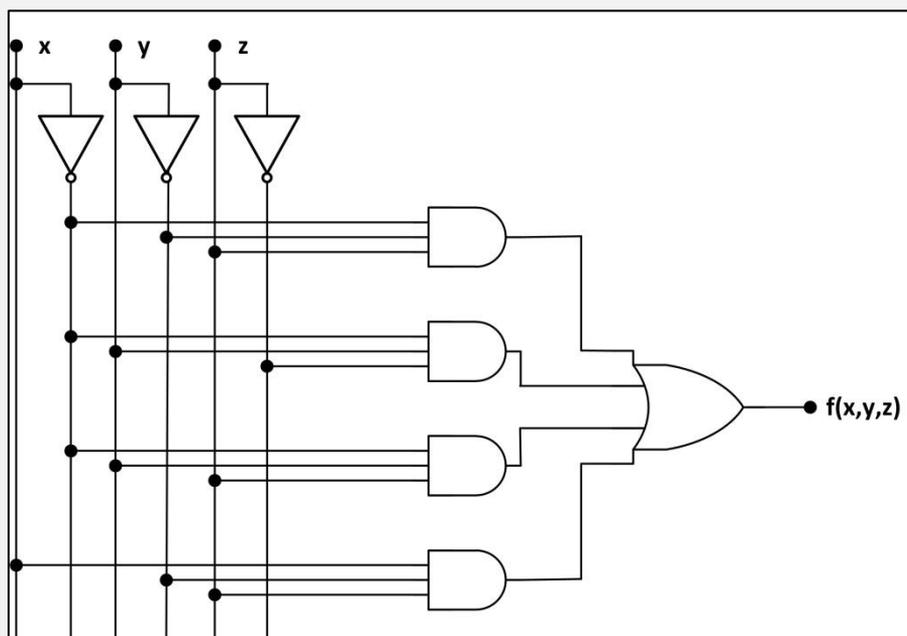


Fig. 3.4. Implementación con compuertas lógicas de la función sin minimizar.

Se invita al lector a minimizar la función y comparar los resultados obtenidos.

## 5 Ejercitación

### Ejercicio 1:

Hallar las expresiones canónicas de las siguientes funciones. Representar la tabla de verdad correspondiente a cada una de ellas.

$$f(a, b, c) = a\bar{c} + bc + a\bar{b}c$$

$$f(a, b, c, d, e) = a\bar{b}ce + \bar{a}de$$

$$f(a, b, c, d) = a + \bar{b}\bar{c} + c$$

### Ejercicio 2:

Simplificar las siguientes expresiones aplicando los teoremas del álgebra de Boole.

$$f(p, q, r) = pqr + p\bar{q} + \bar{p}qr + pqr + p\bar{q}$$

$$f(a, b, c, d) = b\bar{c} + ab\bar{c} + \bar{a}bd + a\bar{b}c + cb\bar{d} + \bar{a}\bar{b}d + cbd$$

### Ejercicio 3:

Dadas las siguientes funciones, representadas mediante la expresión canónica por comprensión de suma de productos y producto de sumas, obtener las representaciones de las mismas en la forma de producto de sumas y suma de productos respectivamente.

$$f(a, b, c) = \sum_3 (2, 5, 7)$$

$$f(a, b, c, d) = \prod_4 (1, 3, 4, 8, 12, 14)$$

### Ejercicio 4:

Obtener la tabla de verdad y la función canónica por comprensión en la forma de producto de sumas de una función de 4 variables que toma el valor 1 cuando 3 o más variables toman el valor 0.

### Ejercicio 5:

Obtener la tabla de verdad y la función canónica por comprensión y extensión en la forma de suma de productos y producto de sumas de una función de 4 variables que toma el valor 0 cuando la variable de menor peso vale 0, y la de mayor peso vale 1.

**Ejercicio 6:**

Demostrar las siguientes igualdades.

$$\overline{\overline{(p+a)}de\bar{c}} = \bar{p}\bar{a}d + \bar{e} + c$$

$$a \oplus b = \bar{a} \oplus \bar{b}$$

$$a(a \oplus c) = ab \oplus ac$$

**Ejercicio 7:**

Haciendo uso de las leyes de De Morgan, indicar cuál de las siguientes igualdades es correcta.

$$\overline{ab + \bar{a}c + \bar{c}b} = \overline{(\bar{a} + \bar{b})(a + \bar{c})(c + \bar{b})}$$

$$(a + \bar{b} + c)(\bar{a} + \bar{b} + \bar{c}) + ab = \overline{(\bar{a}\bar{b}\bar{c} + abc)}(\bar{a} + b)$$

**Ejercicio 8:**

Dadas las siguientes funciones, representadas mediante la expresión canónica por comprensión de suma de productos y producto de sumas, obtener las representaciones de las mismas en la forma de producto de sumas y suma de productos respectivamente.

$$f(a, b, c, d) = \sum_4 (4, 6, 8, 9, 14, 15)$$

$$f(a, b, c) = \prod_3 (1, 2, 5, 6)$$

**Ejercicio 9:**

Minimizar por el método de karnaugh las funciones expresadas en la forma canónica por extensión del Ejercicio 3.

**Ejercicio 10**

Las siguientes expresiones corresponden a funciones minimizadas expresadas en la forma de suma de productos. Obtener las funciones minimizadas expresadas en la forma de producto de sumas correspondientes.

$$f(a,b,c,d)=bcd + \bar{b}\bar{c}d + \bar{a}\bar{c} + \bar{a}bd$$

$$f(a,b,c,d)=\bar{a}b\bar{c} + bd + acd$$

$$f(a,b,c,d)=\bar{a}\bar{b} + bd + \bar{b}\bar{c}$$

**Ejercicio 11:**

En un sistema digital que opera con el código BCD EXC 3 se desea implementar un generador de paridad impar. Indicar la función más simple, ya sea en la forma de producto de sumas o suma de productos que satisfice el requisito.

**Ejercicio 12:**

En un registro de 4 bits, cuyas salidas están disponibles al exterior, se almacena información numérica decimal en el código BCD Natural. Se desea implementar un sistema digital que detecte cuando el número contenido en el registro es superior a 6 e inferior a 3. Indicar la función más simple, ya sea en la forma de producto de sumas o suma de productos que satisfice el requisito.



---

# CAPÍTULO 4

---

## Sistemas Combinacionales

- 1 Sistemas Digitales**
  - 2 Sistemas Combinacionales**
    - 2.1 Introducción**
    - 2.2 Sistemas combinacionales MSI**
  - 3 Casos Comunes de Sistemas Combinacionales MSI**
    - 3.1 Codificadores**
    - 3.2 Decodificadores**
    - 3.3 Multiplexores**
    - 3.4 Demultiplexores**
    - 3.5 Comparadores**
    - 3.6 Detectores/Generadores de paridad**
    - 3.7 Sumadores**
    - 3.8 Unidades aritméticas y lógicas**
  - 4 Ejercitación**
-

## Capítulo 4

# Sistemas Combinacionales

## 1 Sistemas Digitales

Un sistema digital es un conjunto de elementos binarios relacionados entre si de alguna manera. Se distinguen dos tipos de variables en un sistema digital. Las variables de entrada y las variables de salida que dependen de las de entrada. Funcionalmente, las variables de entrada se dividen en dos grupos: variables de proceso y variables de control (Figura 4.1).

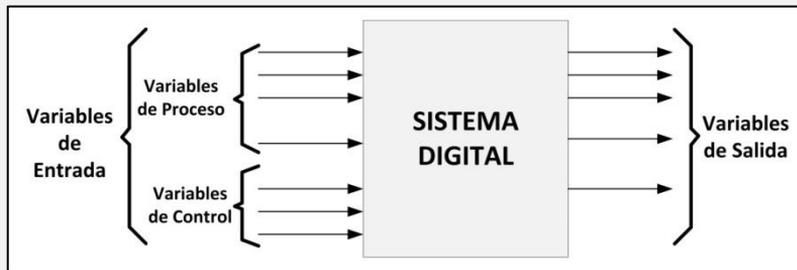


Fig. 4.1. Diagrama de un sistema digital.

Cuando cada combinación de las variables de entrada (Vector de entrada) se corresponde con una y sólo una combinación de las variables de salida (Vector de salida), se trata de un sistema combinacional. Dicho de otra manera, siempre que se repita un conjunto de valores de las variables de entrada, se repetirá la salida. En la Figura 4.2 se muestran las correspondencias entre entradas y salidas de un sistema combinacional.

Cuando a un mismo vector de entrada puede corresponder más de uno de salida, el sistema se llama secuencial. Dicho de otra manera, cuando se repite un conjunto de valores de las variables de entrada, no necesariamente se repetirá la salida. Los sistemas secuenciales deben poseer memoria interna, ya que sus salidas son consecuencia de la evolución anterior de sus entradas. En la Figura 4.3 se muestran las correspondencias de un sistema secuencial.

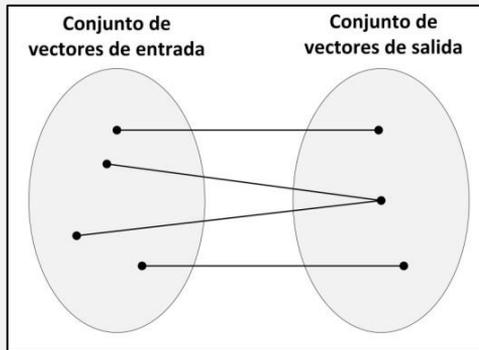


Fig. 4.2. Correspondencias para un Sistema Combinacional.

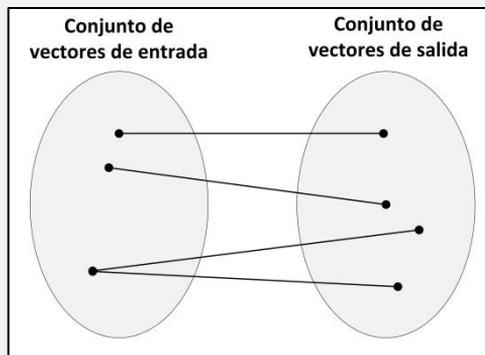


Fig. 4.3. Correspondencias para un Sistema Secuencial.

## 2 Sistemas Combinacionales

### 2.1 Introducción

De lo definido en el punto anterior se concluye que en un Sistema combinacional las salidas no son otra cosa que funciones lógicas de las entradas. En la Figura 4.4 se ve el diagrama en bloque de un combinacional con  $n$  entradas y  $m$  salidas. Se puede escribir que:

$$z_i = f_i(x_1, x_2, \dots, x_n)$$



Fig. 4.4. Diagrama en bloque de un sistema o circuito combinacional.

De lo anterior se deduce que para diseñar un circuito combinacional bastará con minimizar las funciones requeridas y finalmente implementar con compuertas lógicas.

## 2.2 Circuitos combinacionales MSI

Cuando las funciones lógicas son muy complejas, no siempre el diseño basado en la minimización, y posterior implementación con compuertas lógicas, es el más adecuado. Las técnicas de integración han permitido CI más complejos. Por ejemplo, en MSI se dispone de CI de hasta 100 puertas. Estos bloques funcionales MSI, si bien a veces tienen fines específicos, pueden aplicarse a la implementación de funciones lógicas de muchas variables. Las ventajas principales son: la disminución de los CI necesarios, del tiempo de diseño, del número de conexiones externas, y la facilidad del mantenimiento.

A continuación se describen brevemente los Combinacionales MSI más comunes.

## 3 Casos Comunes de Sistemas Combinacionales MSI

### 3.1 Codificadores

Permiten codificar las líneas de entrada. Generalmente codifican en binario o BCD. En la Figura 4.5 se muestra un codificador binario de 8 entradas y 3 salidas, su circuito interno y su tabla de verdad.

En este codificador se supone que sólo está activa una entrada por vez. En caso de no ser así, la salida debe calcularse como la función OR

bit a bit de las salidas correspondientes a las entradas activadas independientemente. Estos decodificadores se llaman sin prioridad.

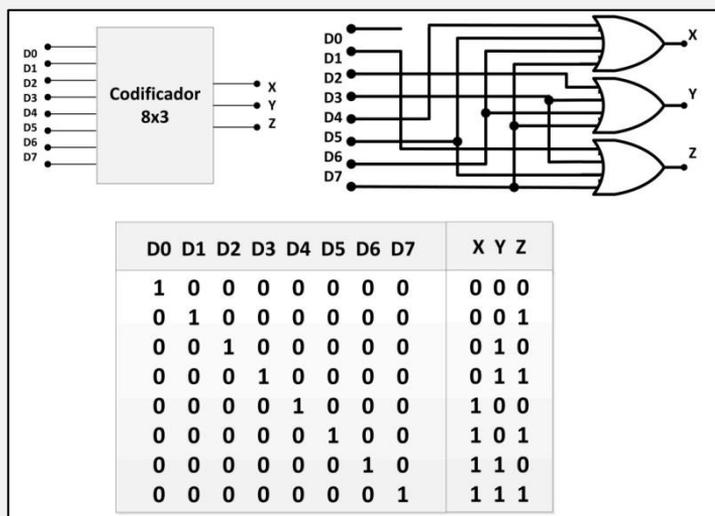


Fig. 4.5. Codificador binario de 8 entradas y 3 salidas.

Si en la tabla de verdad de la Figura 4.5 se rempazan con x los ceros a la izquierda de los unos de las entradas, se obtiene un codificador con prioridad. La entrada de mayor prioridad es la que define la salida.

Si ninguna entrada está activa las salidas son todas cero, igual que si estuviera activada la entrada D0. Para evitar este problema, los codificadores cuentan con una salida adicional que indica la ausencia de activación de las entradas.

Por último, los codificadores suelen contar con una entrada de habilitación. Cuando el chip está activado es válida la tabla de verdad, si no lo está el chip no funciona.

## 3.2 Decodificadores

Son combinacionales que poseen n entradas y m salidas. El orden adecuado de la salida se activa cuando la codificación correspondiente se inyecta a la entrada. Generalmente, son binarios o BCD. En el caso de un decodificador binario, si tiene n entradas poseerá  $m = 2^n$  salidas. Así, un decodificador realiza lo opuesto a un codificador. En la Figura 4.6 se muestra un decodificador de 3 x 8 y su tabla de verdad.

Los decodificadores, además de usarse para decodificar, son útiles para implementar funciones lógicas. Cada una de sus salidas es un minterm de una función de n variables. Aprovechando la entrada de habilitación que suelen tener, es posible aumentar el número de variables. En la Figura 4.7 se usa un decodificador de 3 x 8 para implementar la siguiente función:

$$F(z, y, x) = \sum (1,3,6,7)$$

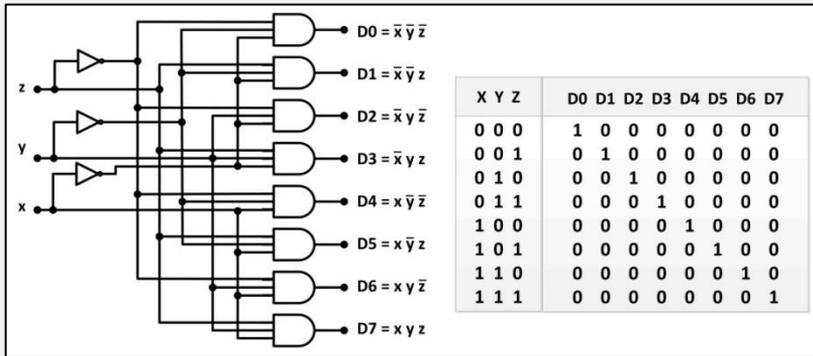


Fig. 4.6. Decodificador de 3 entradas y 8 salidas.

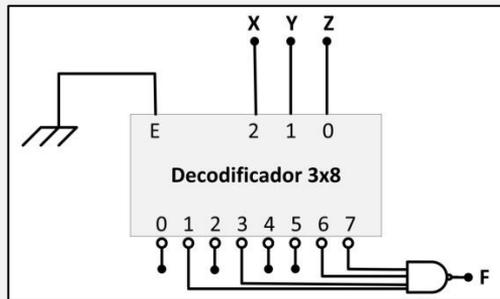


Fig. 4.7. Decodificador de 3x8 para implementar la función F.

En la Figura 4.7 se observa la entrada E de habilitación. Si E = 0 el decodificador está habilitado. Si E = 1, cualesquiera sean los valores de x, y, o z, ninguna salida se activará.

La Figura 4.8 muestra cómo obtener un decodificador de 4 x 16, partiendo de dos decodificadores 3 x 8.

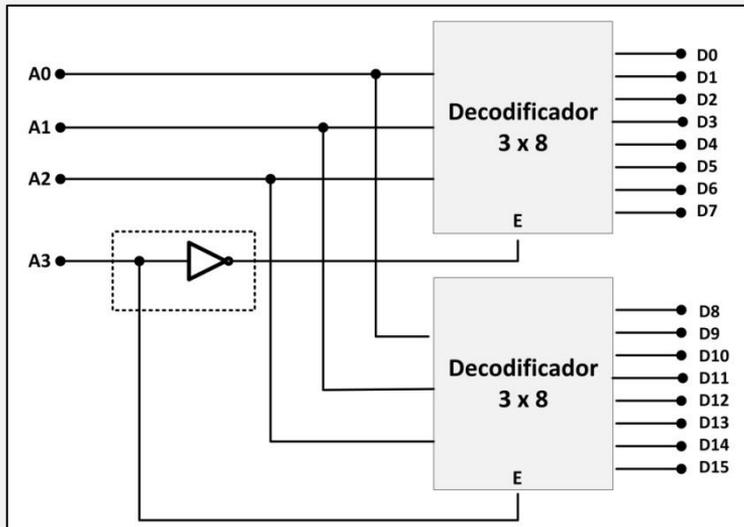


Fig. 4.8. Decodificador de 4x16 a partir de dos decodificadores de 3x8

### 3.3 Multiplexores

Disponen de  $m = 2^n$  líneas de entrada (canales), una línea de salida y  $n$  líneas de selección. En función de las líneas de selección, se determina qué entrada aparece en la salida. La Figura 4.9 indica la función de un multiplexor y la Figura 4.10 el circuito de un multiplexor de 4 canales.

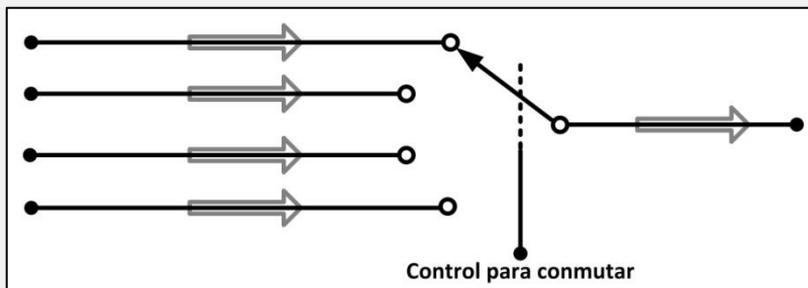


Fig. 4.9. Funcionamiento del multiplexor.

Los multiplexores, además de multiplexar, pueden usarse eficazmente para implementar funciones lógicas. Supongamos que la función a implementar sea:

$$F(a, b, c) = \sum (0, 1, 5, 6, 7)$$

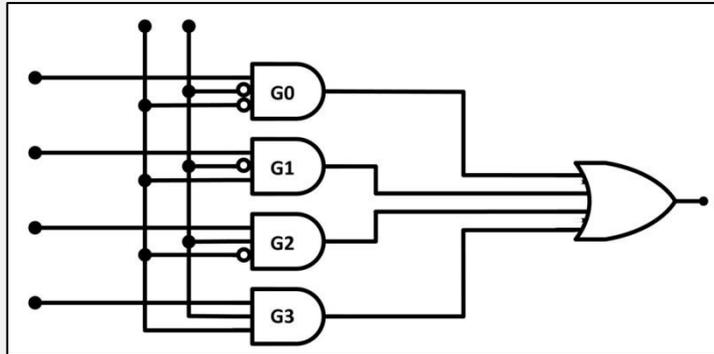


Fig. 4.10. Multiplexor de 4 canales.

Para implementar una función de 3 variables se necesita un multiplexor de  $2^3 - 1$  entradas de selección. Dos de las variables (por ejemplo a y b) se conectan a las líneas de selección. La tercer variable c, se conecta a los canales. A esta altura es conveniente contar con la tabla de verdad de la función, que en nuestro ejemplo es la presentada en la Tabla 4.1.

De la tabla de verdad de la función, se construye la tabla auxiliar presentada en la Tabla 4.2.

Esta tabla auxiliar se obtiene de la anterior verificando cuánto vale la función para las diferentes combinaciones de a y b, y permite determinar qué valores conectar a los canales del multiplexor (Figura 4.11).

c	b	a	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Tabla 4.1. Tabla de verdad de la función F.

<b>b</b>	<b>a</b>	<b>f(a,b,c)</b>
<b>0</b>	<b>0</b>	$\bar{c}$
<b>0</b>	<b>1</b>	<b>1</b>
<b>1</b>	<b>0</b>	<b>c</b>
<b>1</b>	<b>1</b>	<b>c</b>

Tabla 4.2. Tabla auxiliar.

El procedimiento puede generalizarse para  $n$  variables, todas menos una se conectan a las líneas de selección del multiplexor. La restante a los canales de acuerdo a la tabla auxiliar.

Para realizar multiplexores de muchos canales, pueden combinarse diferentes multiplexores. Por ejemplo en la Figura 4.12 se muestra un multiplexor de 32 canales a partir de dos de 16 canales y uno de dos canales. La entrada  $E$  (habilitación) se activa con un 0. Si  $E = 1$ , la salida del multiplexor es 0, independientemente del valor de las entradas.

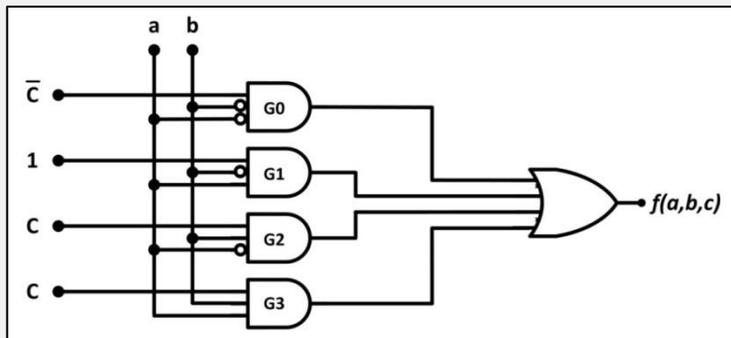


Fig. 4.11. Implementación de la función  $f$  usando un multiplexor.

### 3.4 Demultiplexores

Cumplen la función opuesta a los multiplexores. Tienen una entrada y  $m$  salidas, y  $n$  entradas de selección. La salida seleccionada tendrá el valor de la entrada. En la Figura 4.13 se muestra un demultiplexor de cuatro canales de salida.

El circuito de un demultiplexor es coincidente con un decodificador que posea entrada de habilitación. Por esta razón no se

encuentran demultiplexores específicos. En la Figura 4.14 se indica cómo obtener un demultiplexor de cuatro canales desde un decodificador de 2 x 4 con entrada de habilitación.

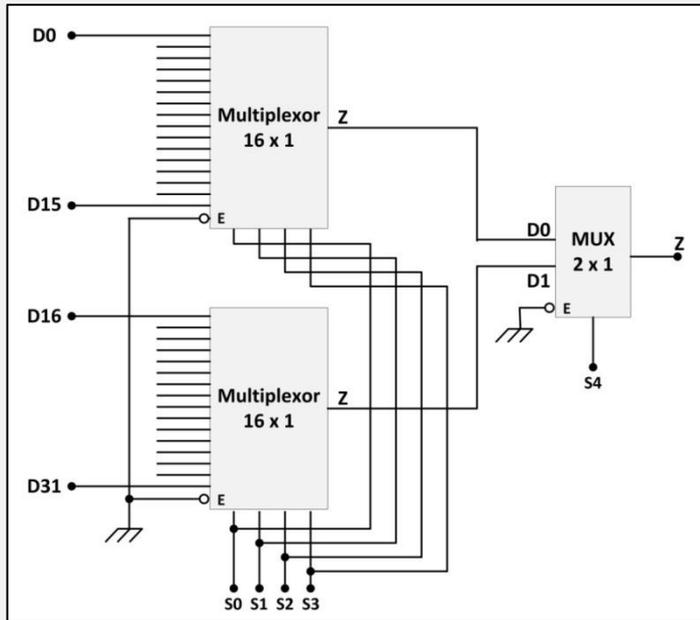


Fig. 4.12. Multiplexor de 32 canales usando dos multiplexores de 16 canales.

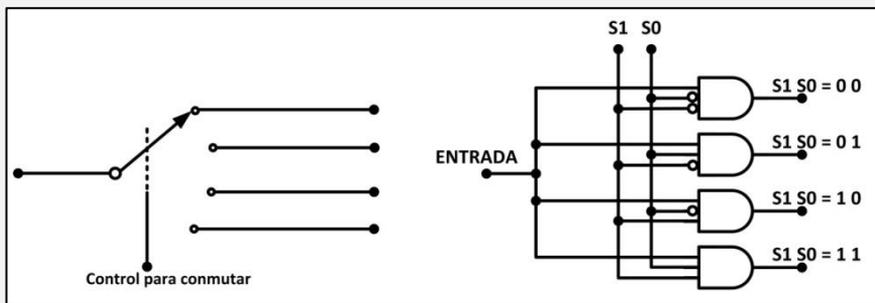


Fig. 4.13. Demultiplexor de 4 canales de salida.

Es usual encontrar en algunas familias lógicas multiplexores/demultiplexores. Estos circuitos pueden cumplir ambas funciones.

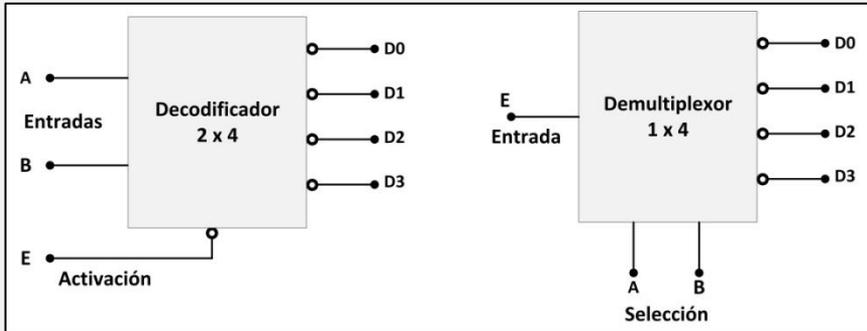


Fig. 4.14. Demultiplexor de 4 canales usando un decodificador de 2x4.

### 3.5 Comparadores

Realizan la comparación entre dos números binarios de  $n$  bits. El circuito básico que realiza la comparación de 1 bit, se indica en la Figura 4.15.

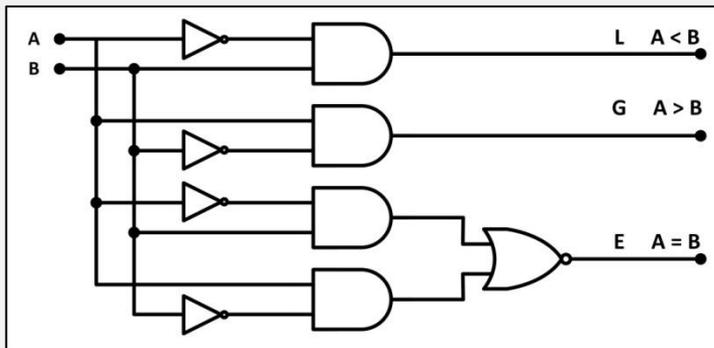


Fig. 4.15. Circuito básico de comparación de 1 bit.

Este circuito responde a la tabla de verdad de la Tabla 4.3.

Comparadores de más bits se diseñan de la misma manera. Los Comparadores poseen además entradas por  $=$ ,  $<$ , y  $>$ . Esto permite realizar comparadores de elevado número de bits, partiendo de comparadores menores. Por ejemplo, en la Figura 4.16 se muestra un comparador de 8 bits, partiendo de 2 comparadores de 4 bits.

B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Tabla 4.3. Tabla de verdad del comparador.

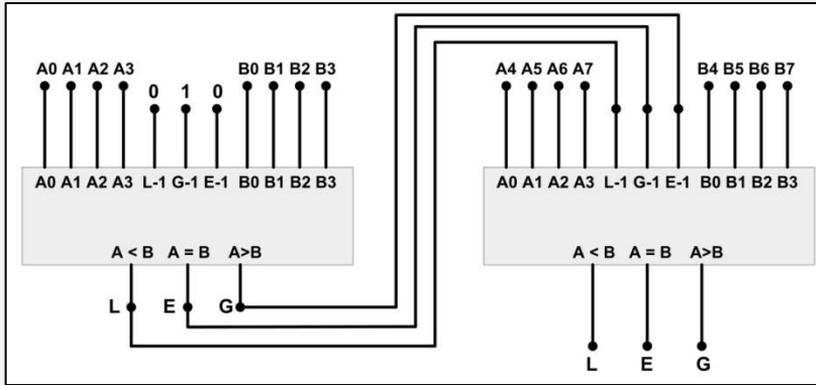


Fig. 4.16. Comparador de 8 bits usando 2 comparadores de 4 bits.

### 3.6 Detectores/Generadores de paridad

Son CI capaces de generar/detectar la paridad de un conjunto de bits. En la Figura 4.17 se muestra un generador/detector de paridad de 8 bits, su circuito.

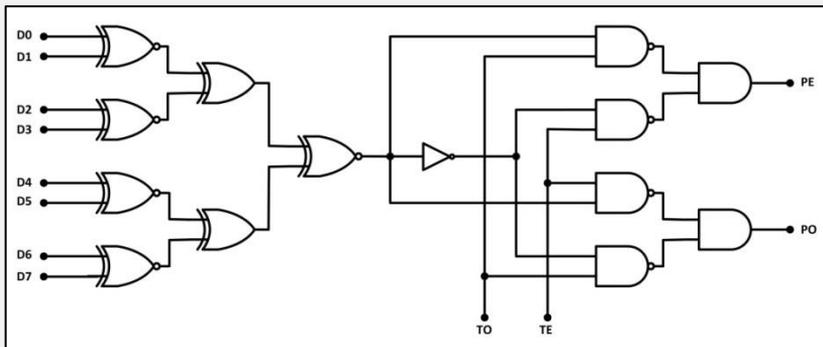


Fig. 4.17. Generador/Detector de paridad de 8 bits.

Las señales de control TO (paridad impar) y TE (paridad par) permiten seleccionar la paridad. Se recomienda al alumno obtener la tabla de verdad de este circuito y verificar su funcionamiento.

### 3.7 Sumadores

Son CI que realizan la suma aritmética de dos números de  $n$  bits. Antes de ver los sumadores disponibles en escala de integración MSI, estudiaremos la suma y resta binaria.

#### Suma binaria

Para indicar la suma aritmética utilizaremos el símbolo  $+$  para diferenciarlo del  $+$  usado para la suma lógica. Para sumar dos bits, se puede implementar el circuito de la Figura 4.18, llamado Semisumador, cuya tabla de verdad se observa en la Tabla 4.4.

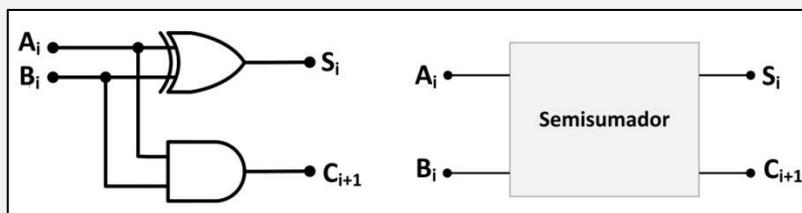


Fig. 4.18. Semisumador o sumador parcial.

$B_i$	$A_i$	$S_i$	$C_{i+1}$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabla 4.4. Tabla de verdad del semisumador.

Supóngase ahora que se desea sumar dos números binarios de cuatro bits  $A$  y  $B$ , entonces:

$$\begin{array}{r}
 C_4 \ C_3 \ C_2 \ C_1 \ C_0 \\
 \mathbf{A} = a_3 \ a_2 \ a_1 \ a_0 \\
 + \\
 \mathbf{B} = b_3 \ b_2 \ b_1 \ b_0 \\
 \hline
 \mathbf{S} = s_3 \ s_2 \ s_1 \ s_0
 \end{array}$$

Se observa que son necesarios cuatro circuitos, uno para cada columna, y cada uno debe ser capaz de sumar tres bits: ai, bi, y ci. Se implementa entonces el circuito de la Figura 4.19, llamado Sumador Total, cuya tabla de verdad se presenta en la Tabla 4.5.

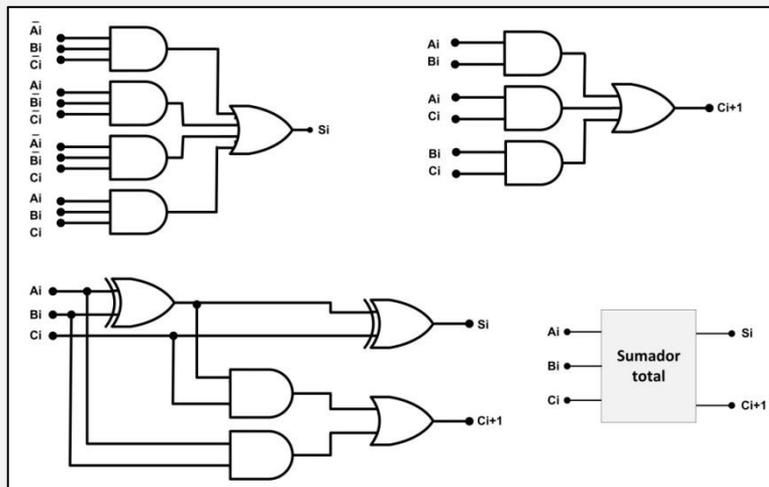


Fig. 4.19. Sumador total.

La interconexión de cuatro Sumadores Totales permite obtener un Cuádruple Sumador Total, capaz de realizar la suma aritmética de dos números binarios de cuatro bits (Figura 4.20).

## Resta binaria

Deben recordarse los convenios de representación de números negativos en binario. Se podría implementar un circuito para realizar la resta como una nueva operación. Sin embargo, se verá que es posible restar dos números realizando la suma de uno de ellos más el complemento a dos (o a uno) del otro.

$C_i$	$B_i$	$A_i$	$S_i$	$C_{i+1}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabla 4.5. Tabla de verdad del sumador total.

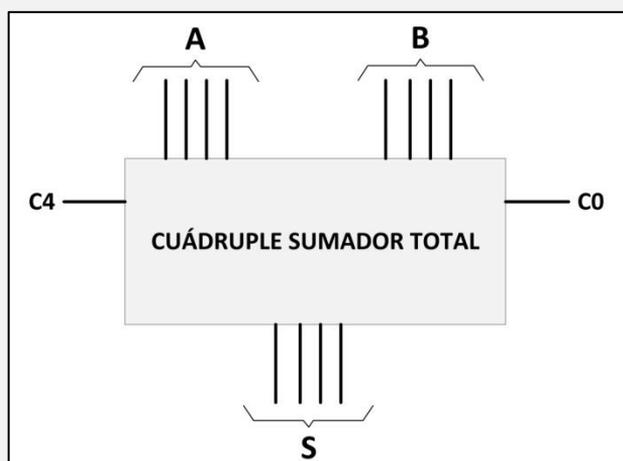


Fig. 4.20. Cuádruple sumador total.

- a) Para el caso de usar el convenio de complemento a dos.

Sean A y B dos números binarios signados en convenio de complemento a dos. Véase el siguiente desarrollo:

$$A - B / A + C_2(B) = A + 2^n - B = 2^n + (A - B)$$

En la expresión se observa que el resultado obtenido difiere del buscado en el valor  $2^n$ . Este resultado debe interpretarse como un acarreo a despreciar si el paréntesis (A - B) resulta positivo. En caso que el paréntesis resulte negativo, el resultado estará expresado en complemento a dos.

b) Para el caso de usar complemento a uno

Sean A y B dos números binarios signados en convenio de complemento a uno. El desarrollo en este caso sería:

$$A - B / A + C_1(B) = A + 2^n - 1 - B = 2^n + (A - B) - 1$$

En la expresión se observa que el resultado obtenido difiere del buscado en el valor  $2^n$  y además tiene un error en defecto de valor 1. Este resultado debe interpretarse como un acarreo que debe sumarse al resultado si el paréntesis (A - B) resulta positivo. En caso que el paréntesis resulte negativo el resultado estará expresado en complemento a uno.

Es conveniente que el alumno verifique el párrafo anterior par todos los casos, es decir:

- A > 0 ; B > 0
- A > 0 ; B < 0
- A < 0 ; B > 0
- A < 0 ; B < 0

En la Figura 4.21, se muestra un Sumador/Restador en complemento a dos de 4 bits a partir de un Cuádruple sumador total. Y en la Figura 4.22 se muestra un Sumador/Restador de 4 bits en complemento a uno a partir de un Cuádruple sumador total.

En estas dos últimas figuras aparece un circuito detector de rebasamiento. Su salida será 1 si se ha rebasado al sumador y cero en caso contrario. Nótese que las entradas a este circuito son los bits de signo de los números de entrada y del resultado.

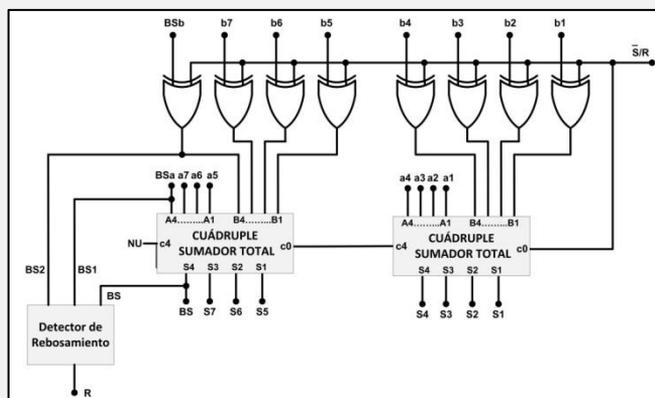


Fig. 4.21. Sumador/Restador de 4 bits en complemento a dos.

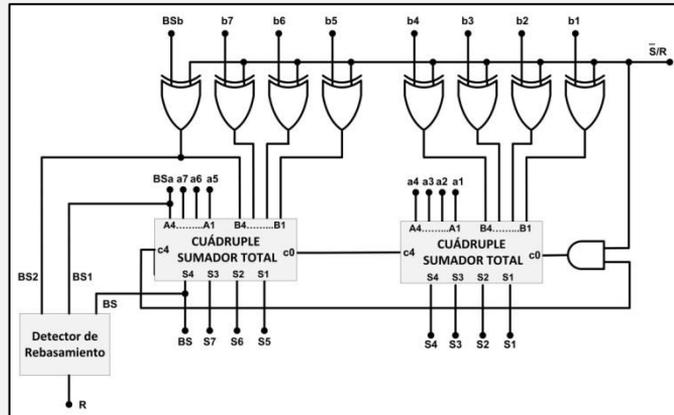


Fig. 4.22. Sumador/Restador de 4 bits en complemento a uno.

### 3.8 Unidades aritméticas y lógicas

Son bloques funcionales en escala MSI que permiten realizar operaciones lógicas y aritméticas sobre números binarios (generalmente de 4 bits). La operación a realizar se selecciona colocando los valores adecuados en las líneas de selección. En la Figura 4.23 se muestra una ALU típica de 4 bits.

Estos bloques funcionales pueden conectarse en cascada para realizar operaciones sobre números de mayor número de bits.

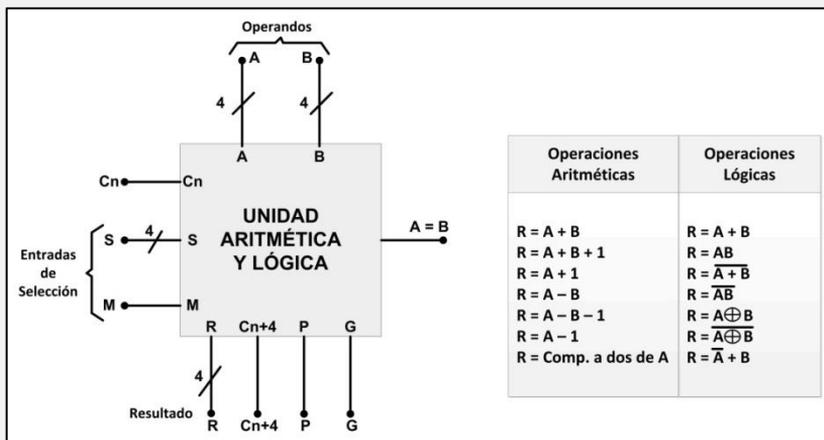


Fig. 4.23. ALU de 4 bits.

## 4 Ejercitación

### Ejercicio 1:

Diseñe un sistema combinacional que accione el motor de un limpiaparabrisas sabiendo que éste funciona cuando la llave general del auto está en contacto, y además, se activa el control del limpiaparabrisas, pero si desactiva este último, el motor sigue funcionando hasta que las escobillas lleguen al costado izquierdo. Obtener la solución más simple, e implementarla con las compuertas correspondientes.

### Ejercicio 2:

Realizar el diseño e implementación de dos llaves de luz colocadas al pie y cima de una escalera; de tal manera que pueda prender la luz de la llave de abajo y apagarla con la llave de arriba y viceversa.

### Ejercicio 3:

Utilizar decodificadores para implementar las siguientes funciones:

$$f(a, b, c, d) = \sum_4 (0, 6, 8, 10)$$

$$f(a, b, c, d) = \sum_4 (1, 5, 7, 8, 13)$$

### Ejercicio 4:

Realizar el diseño de un comparador de dos números de dos bits cada uno. Este sistema tiene que tener 3 salidas: la de mayor, menor e igual. Realizar el diseño con compuertas NAND.

### Ejercicio 5:

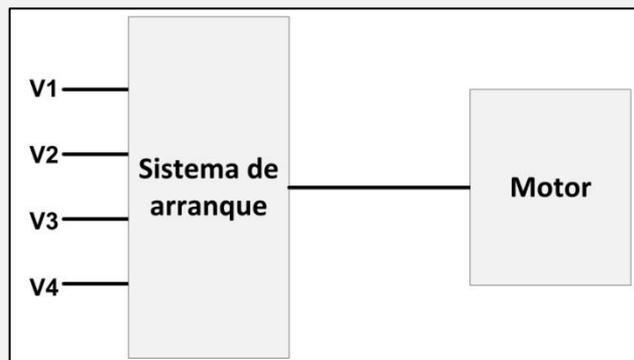
Diseñar un sistema combinacional que ante la excitación de 4 variables de entrada genere salidas que activen un display BCD 7 segmentos para la representación de los dígitos decimales sobre el mismo. Considerar que las combinaciones de entrada corresponden al código BCD 8421, y que las restantes a las 16 posibles a las letras del alfabeto a, b, c, d, e y f. Obtener la solución más simple, e implementarla con las compuertas NOR.

**Ejercicio 6:**

Un registro de salidas en paralelo A0-A3 contiene un carácter codificado en BCD 5421. Realizar un generador de paridad para agregar el correspondiente bit  $A_p$ , que permita elegir paridad par o impar con una llave selectora. Utilizar decodificadores y multiplexores

**Ejercicio 7:**

El diagrama de la figura indica el sistema de arranque de un motor eléctrico. Existen cuatro variables binarias de control las cuales permitirán el arranque del motor sólo cuando existe paridad impar entre ellas. Hay dos casos especiales, llamados de emergencia que permitirán arranque, y ocurren cuando todas las variables están en uno o cero simultáneamente. Realizar con compuertas.

**Ejercicio 8:**

Construir un multiplexor de 8 canales a partir de multiplexores de 2 canales.

**Ejercicio 9:**

Realizar un detector de desbordamiento aritmético. Tenga en cuenta que el intervalo de valores que se puede representar mediante números de  $n$  bits en el sistema numérico de complemento a dos es:

$$-2^{n-1} \leq N \leq 2^{n-1} - 1$$



---

# CAPÍTULO 5

---

## Sistemas Secuenciales

- 1 Visión General de los Sistemas Secuenciales**
    - 1.1 Introducción
    - 1.2 Un caso de estudio
  - 2 Biestables**
    - 2.1 Introducción
    - 2.2 Biestables asíncronos
    - 2.3 Biestables síncronos
  - 3 Tipos de Biestables**
    - 3.1 Introducción
    - 3.2 Biestables JK
    - 3.3 Biestables T
    - 3.4 Biestables D
  - 4 Aplicaciones de los Biestables**
    - 4.1 Registros de desplazamiento
    - 4.2 Transferencias entre registros
    - 4.3 Contadores
    - 4.4 Multiplicación y división binaria
  - 5 Ejercitación**
-

## Capítulo 5

# Sistemas Secuenciales

## 1 Visión General de los Sistemas Secuenciales

### 1.1 Introducción

Son aquellos Sistemas Digitales cuyas salidas no sólo dependen de sus entradas en un momento dado, sino también de cómo han evolucionado estas anteriormente.

El Sistema Secuencial tiene que ser capaz de memorizar la mencionada evolución. Puede decirse que las salidas de un Sistema Secuencial dependen de ellas mismas y de las entradas.

Este concepto es equivalente al anterior y permite plantear un esquema general de Sistema Secuencial partiendo de un Sistema Combinacional realimentado (las entradas a este combinacional están formadas por variables independientes y además por una o más salidas del mismo. Lo mencionado puede observarse en la Figura 5.1.

Se observa un nuevo tipo de variables llamadas variables internas. El bloque M, indica un circuito capaz de mantener el estado de sus entradas en su valor, por un cierto tiempo.

El lector puede deducir que el sistema evolucionará entre distintos estados internos hasta arribar a un estado estable. Efectivamente, para un valor de las variables de entrada determinado, las salidas del combinacional adoptarán cierto estado; como algunas de ellas se realimentan, las salidas del combinacional cambiarán nuevamente. Este proceso (llamado evolución automática del sistema) se repetirá hasta tanto el valor de las variables internas coincida con el anterior, este es el estado estable.

Si a las variables internas se las deja pasar de izquierda a derecha sólo en ciertos momentos, se obtiene un Sistema Secuencial Síncrono como se muestra en la Figura 5.2.

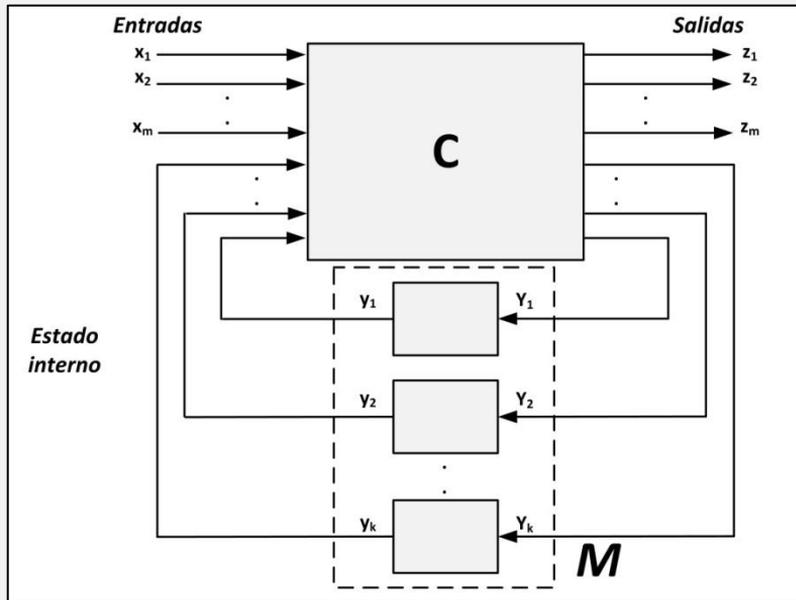


Fig. 5.1. Sistema Secuencial Asíncrono.

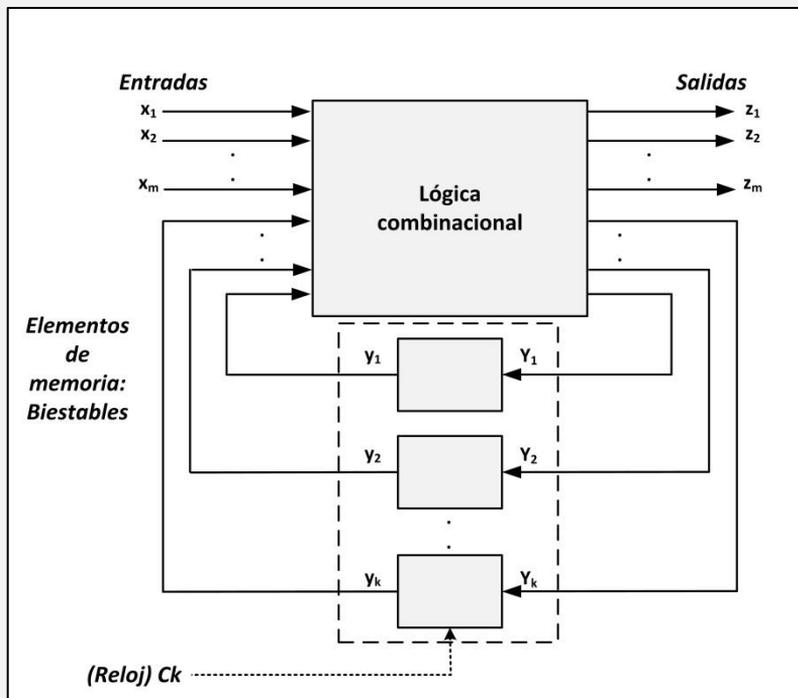


Fig. 5.2. Sistema Secuencial Síncrono.

El diseño básico de estos sistemas consiste en plantear una tabla de verdad en la cual se tenga en cuenta el concepto tiempo. Ahora, una función lógica no sólo depende de ciertas variables independientes sino que también depende de sí misma. Por ejemplo:

$$p = f(a, b, c, \dots, p, \dots)$$

Se observa que la función, indicada como p, aparece en ambos miembros de la expresión. Esto, para que no carezca de sentido, debe interpretarse de la siguiente manera:

$$p_{t+1} = f(a, b, c, \dots, p_t, \dots)$$

El subíndice t+1 se interpreta como el valor que adoptará p para el conjunto de valores que tenían las variables de las cuales depende en el instante t.

## 1.2 Un caso de estudio

A fin de aclarar los conceptos anteriores, sea plantea la resolución del siguiente problema:

Considere la construcción de un Sistema Digital para una alarma domiciliaria que posea:

- Dos variables de entrada:
  - Variable S: entrada de sensor. Por ejemplo, proviene del sensor de una puerta.
  - Variable R: entrada de inicialización.
- Una variable de salida:
  - Variable Q salida a la sirena

El sistema deberá funcionar de la siguiente manera:

- Cuando la variable S tome el valor 1, se activará la salida ( $Q = 1$ ), y si ya estaba activada permanecerá en esa condición. La salida quedará activada aún cuando la variable S pase a 0.
- Cuando la variable R toma el valor 1, se desactivará la salida ( $Q = 0$ ), y si ya estaba desactivada permanecerá en esa condición. La salida quedará desactivada aún cuando la variable R pase a 0
- Nunca S y R podrán valer 1 simultáneamente.

Se propone la tabla de verdad (Tabla 5.1) con el concepto de tiempo ya explicado:

R	S	Qt	Qt+1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	X
1	1	1	X

Tabla 5.1. Tabla de verdad del sistema secuencial de ejemplo.

Las entradas al combinacional serán las tres variables de la izquierda, las X indican que nunca se produce esa combinación de las variables de entrada (por lo tanto, no importa el valor que adopte Q t+1). Para implementar el circuito se realiza el mapa de Karnaugh para la función. La Figura 5.3 presenta el caso para la función en forma de minterms y la Figura 5.4 para el caso en de maxterms:

$$Q_{t+1} = \Sigma_3(1, 2, 3, 6, 7) \quad \text{o} \quad Q_{t+1} = \Pi_3(0, 1, 2, 3, 7)$$

Las X corresponden a los minterms 6 y 7, y a los maxterms 0 y 1. Se ha decidido incluirlos en las expresiones por cuanto podría obtenerse una minimización óptima.

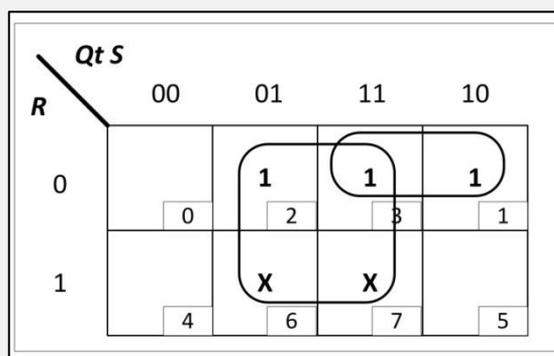


Fig. 5.3. Función en  $\Pi$ .

		$QtS$			
		00	01	11	10
$R$	0	X	1	1	X
	1			1	
		0	2	3	1
		4	6	7	5

Fig. 5.4. Función en  $\Sigma$ .

De la Figura 5.3:

$$Q_{t+1} = S + Q_t \bar{R} = \overline{\overline{S} \overline{Q_t R}}$$

Y resulta, entonces, el circuito de la Figura 5.5.

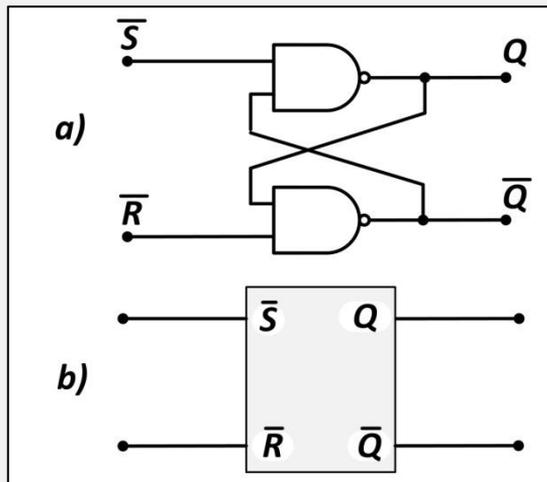


Fig. 5.5. (a) Circuito - (b) Biestable SR (NAND).

De la Figura 5.4 puede deducirse el circuito de la figura 5.6 formado por dos compuertas NOR realimentadas entre sí.

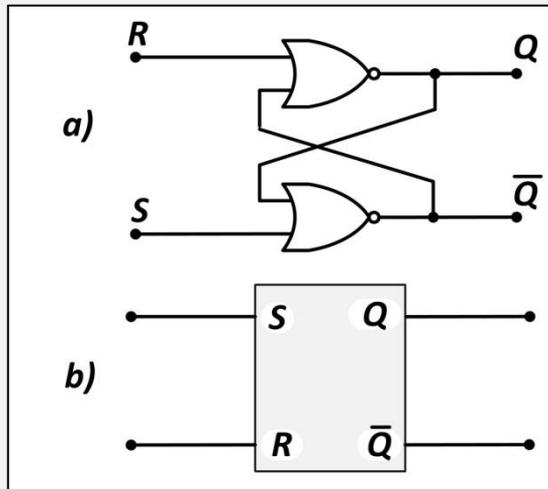


Fig. 5.6. (a) Circuito – (b) Biestable SR (NOR).

Dos conceptos útiles:

- I) Las compuertas lógicas reales se diferencian de las ideales en:
- Poseen un tiempo de retardo, es decir: la señal lógica tarda un tiempo no nulo para atravesar la compuerta.
  - Disipan calor.

La característica a) es de especial importancia en los Sistemas Secuenciales. Efectivamente, en la figura 5.1 aparecen unos elementos M necesarios para que el secuencial funcione. Si estos elementos no estuvieran, una misma línea lógica debería tener dos estados a la vez y esto no es posible. Sin embargo, en los Biestables de las Figuras 5.5 o 5.6, estos elementos M no aparecen. La razón por la cual funcionan es que están contruidos con compuertas reales y el retardo propio de las mismas cumple la función de los elementos M.

- II) En una señal lógica se pueden indicar las siguientes partes (Figura 5.7):

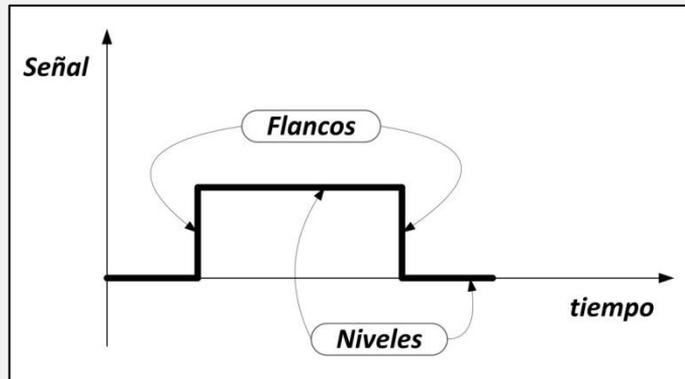


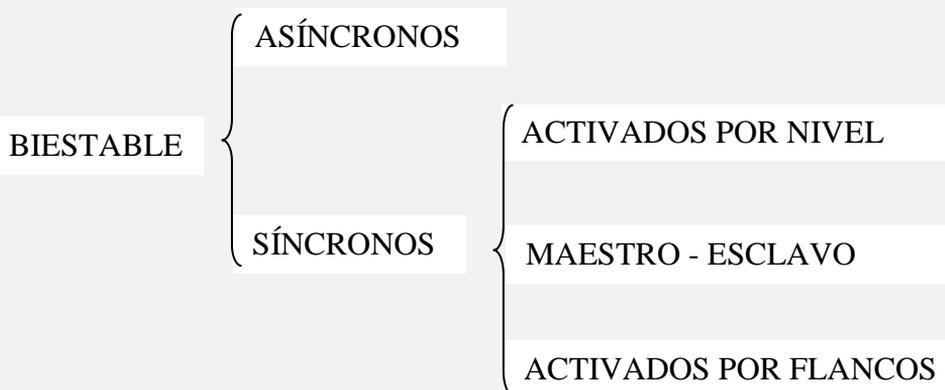
Fig. 5.7. Componentes de una señal lógica.

## 2 Biestables

### 2.1 Introducción

Como se vio en el problema anterior, los Biestables son secuenciales que poseen dos estados estables, es decir, que las variables internas pueden adoptar en este caso dos estados en los cuales permanecerán indefinidamente a menos que cambien las variables de entrada. Se trata entonces de los secuenciales más simples ya que poseen una sola variable interna. Los Biestables representan los circuitos base para la construcción de secuenciales más complejos.

Se puede clasificar a los Biestables de la siguiente manera:



## 2.2 Biestables asíncronos

Son aquellos en los cuales las entradas actúan directamente sobre el biestable. Son ejemplos de estos biestables los vistos en el problema de la alarma domiciliaria (Fig. 5.5 y Fig. 5.6). Puede decirse que la tabla de verdad de la Tabla 5.1 es válida en todo momento.

## 2.3 Biestables síncronos

Estos Biestables cuentan con una entrada adicional: La entrada de sincronismo o reloj. De acuerdo a cómo actúa esta señal, los Biestables síncronos se dividen en activados por nivel, maestro – esclavo y activados por flancos.

### a) Biestables Síncronos activados por nivel

Son aquellos biestables en los cuales la tabla de verdad es válida sólo en presencia de un nivel activo en la entrada de sincronismo. La figura 5.8 muestra un biestable SR síncrono por nivel.

Se observa que hay una parte de un RS asíncrono y se le agrega un circuito de disparo. La Figura 5.8 (a), muestra un SR activado con nivel 1, y la Figura 5.8 (b), un SR activado con nivel 0.

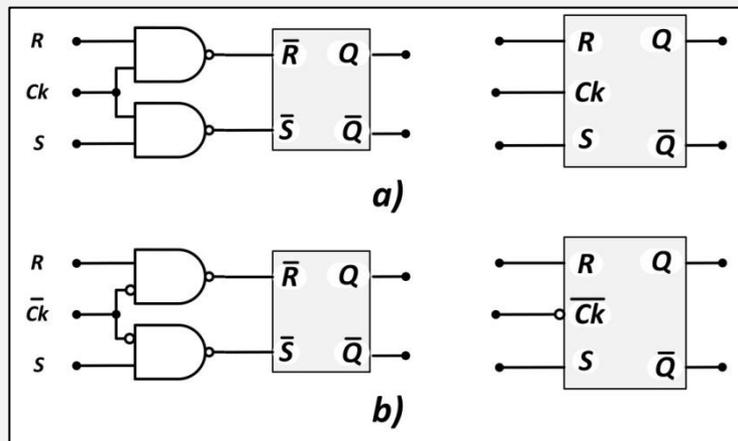


Fig. 5.8. Biestable SR síncrono por nivel.

b) Biestables síncronos maestro – esclavo

Están formados por dos biestables activados por nivel. La Figura 5.9 muestra un biestable SR maestro esclavo.

Se observa que mientras  $Ck = 1$ , se encuentra funcionando el primer biestable (maestro); en el momento que  $Ck = 0$ , la información del maestro pasa al esclavo. Este biestable actúa como si estuviera activado en el flanco de bajada de la señal de sincronismo, no obstante se diferencia de los activados por flancos en el hecho que las entradas actúan sobre el maestro durante el tiempo que  $Ck = 1$ .

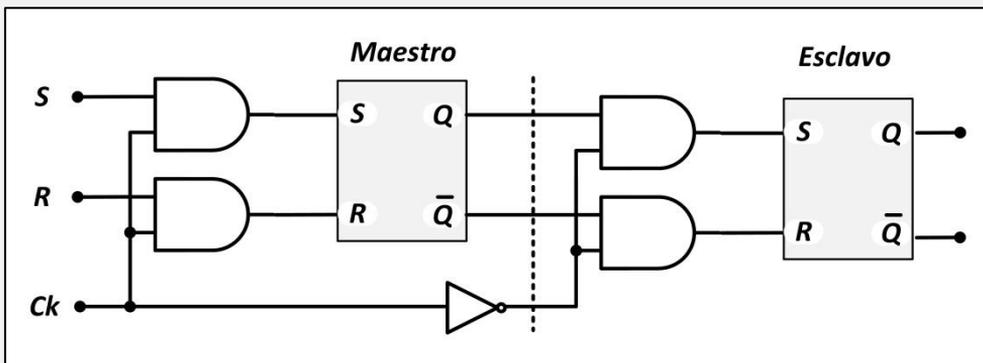


Fig. 5.9. Biestable SR maestro esclavo.

c) Biestables activados por flancos

En estos Biestables las entradas actúan sólo en presencia de un flanco (de subida o bajada) en la entrada de sincronismo. La tabla de verdad será válida sólo en esos instantes. En la Figura 5.10 se muestra un SR activado por flanco.

Se trata de un SR síncrono por nivel al cual se le agrega un circuito detecto de flancos. En la Figura 5.11, se muestra el símbolo utilizado para este tipo de biestables. La Figura 5.11 (a), muestra un SR sincronizado por flanco de subida. La Figura 5.11 (b), muestra un SR sincronizado por flanco de bajada.

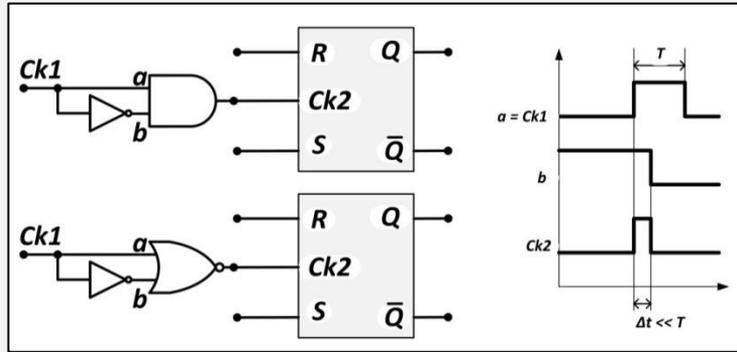


Fig. 5.10. Biestable SR activado por flanco.

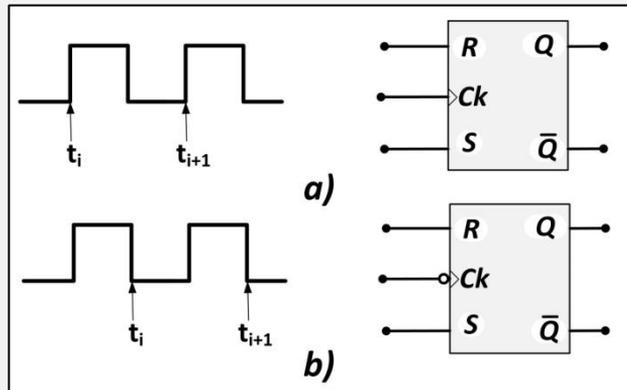


Fig. 5.11. Representación de los biestables activados por flanco

Todos los Biestables SR tienen la misma tabla de verdad. Dependiendo de la clase de SR, la tabla de verdad es válida siempre, durante el nivel activo o durante el flanco activo, según corresponda. Una forma reducida de indicar la tabla de verdad de un SR se indica en la Tabla 5.2.

R	S	Qt+1
0	0	Qt
0	1	1
1	0	0
1	1	X

Tabla 5.2. Tabla de verdad de un biestable SR.

## 3 Tipos de Biestables

### 3.1 Introducción

Existen otros tipos de Biestables diferentes al SR. Ellos son el biestable JK, el biestable T, y el biestable D. Estos no se encuentran disponibles en todas las clases (asíncronos, síncronos por nivel, etc.). Las tablas de verdad son las indicadas en la Tabla 5.3 a, b y c, respectivamente.

K	J	Qt+1	T	Qt+1	D	Qt+1
0	0	Qt	0	Qt	0	0
0	1	1	1	$\bar{Q}_t$	1	1
1	0	0				
1	1	$\bar{Q}_t$				
<b>Biestable JK</b>			<b>Biestable T</b>		<b>Biestable D</b>	

Tabla 5.3. Biestables JK, T y D.

### 3.2 Biestables JK

En la Tabla 5.3 (a), para  $J = K = 1$ ,  $Q_{t+1} = \bar{Q}_t$ , es decir, la salida adopta el valor opuesto al anterior. Por esta razón, sólo tienen aplicación práctica los biestables JK síncronos activados por flancos. En las Figuras 5.12 y 5.13 se muestran algunos biestables JK.

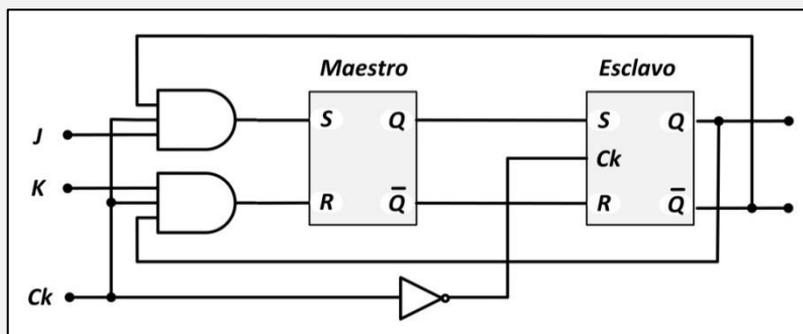


Fig. 5.12. Biestable JK Maestro – Esclavo a partir de dos SR por nivel.

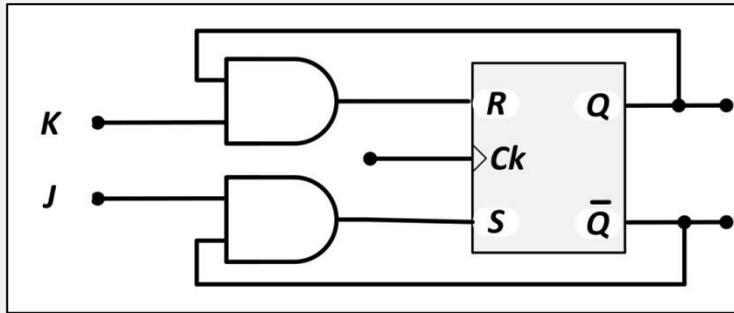


Fig. 5.13. Biestable JK por flanco ascendente a partir de un SR por flanco.

### 3.3 Biestables T

No están disponibles comercialmente. Se obtienen a partir de un biestable JK haciendo  $J=K=T$ . También pueden obtenerse a partir de un biestable D por flancos.

### 3.4 Biestables D

En la Tabla 5.3 se muestra la tabla de verdad de este biestable. Se concluye que carece de aplicación un biestable D asíncrono. Por lo tanto, los biestables D se disponen comercialmente como síncronos, ya sea por nivel (D Latch), Maestro – Esclavo, o por flanco. Pueden obtenerse a partir de un SR síncrono, haciendo  $S = R' = D$  (Figura 5.14 a). También desde un JK, haciendo  $J = K' = D$  (Figura 5.14 b).

En cuanto al sincronismo, en la Figura 5.14 a, puede usarse un SR por nivel, maestro–esclavo o por flanco y resultará un biestable D por nivel, maestro–esclavo o por flanco, respectivamente. En la Fig. 5.14 b, se supone un JK maestro-esclavo o por flanco.

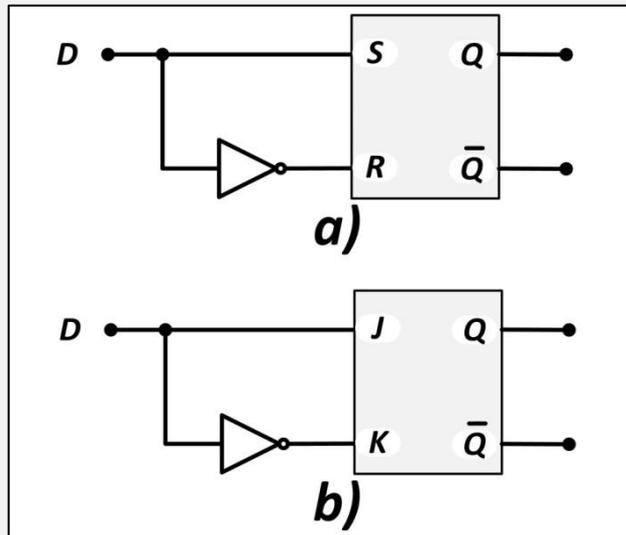


Fig. 5.14. Biestable D implementado con biestables SR y JK.

## 4 Aplicaciones de los Biestables

Los biestables son secuenciales básicos capaces de memorizar un bit. Existen infinidad de aplicaciones, entre las principales se encuentran:

- Memorias electrónicas (desarrolladas en el Capítulo 6)
- Registros, y
- Contadores.

### 4.1 Registros de desplazamiento

Es un sistema secuencial síncrono que almacena varios bits de información. El formato de la información puede ser de dos tipos: serie (cuando los bits se transfieren uno después del otro por la misma línea) o paralelo (cuando se transfieren simultáneamente). Los registros pueden clasificarse de la siguiente manera:

- I) Registros de Desplazamiento
  - Entrada serie, salida serie
  - Entrada serie, salida paralela
  - Entrada paralela, salida serie
- II) Registros propiamente dichos (o sólo Registros)
  - Entrada paralela, salida paralela

## Registros de desplazamiento serie - serie

Se considerará que los registros tienen una gran cantidad de bits almacenados, principalmente para el caso serie - serie. En las Figuras 5.15 y 5.16 se observan registros de algunos bits.

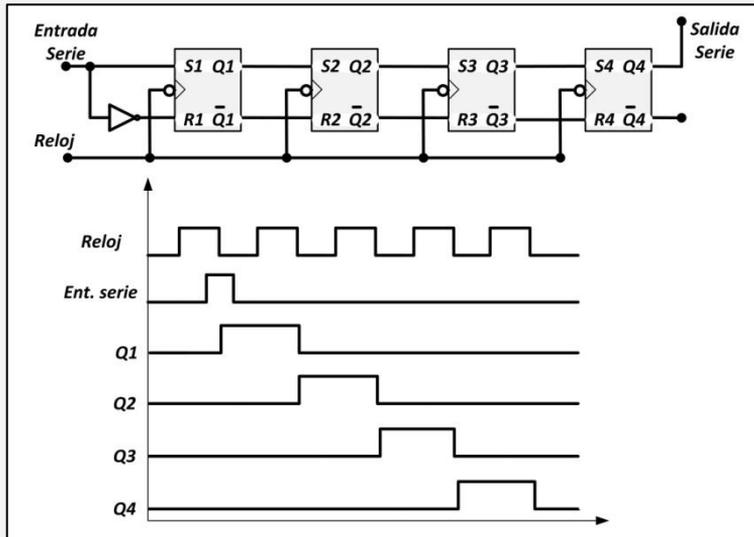


Fig. 5.15. Registro de desplazamiento de cuatro bits serie - serie.

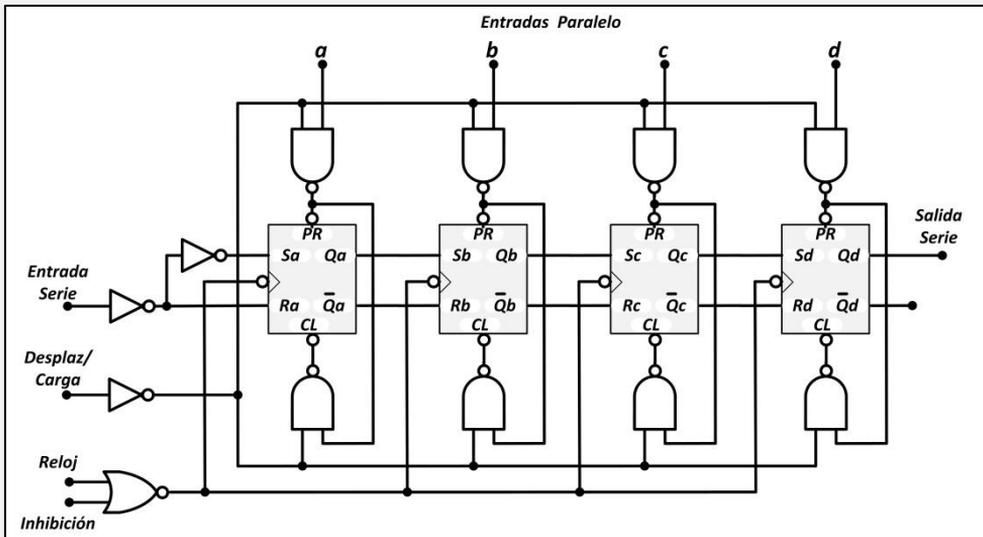


Fig. 5.16. Registro de desplazamiento paralelo - serie de cuatro bits.

## Registros propiamente dichos

Consisten en un conjunto de biestables sincronizados por nivel o por flancos, cuyas entradas de sincronismo se encuentran unidas. Son de uso extensivo en cualquier sistema digital. En la Figura 5.17 se muestra un registro de 8 bits cuyas salidas están provistas de inversores triestado a fin de conectarse a un bus.

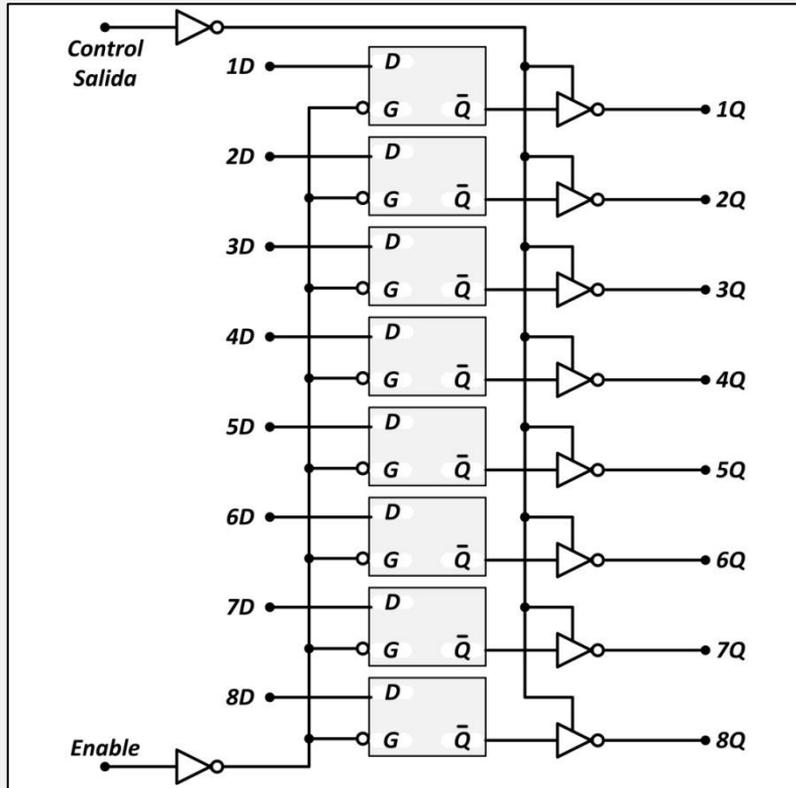


Fig. 5.17. Registro paralelo - paralelo de 8 bits.

## 4.2 Transferencias entre registros

Buena parte de la actividad de un Sistema Digital es la transferencia de los contenidos entre distintos registros. En la Guía Didáctica 5 se presenta una computadora elemental, y su funcionamiento se basa en la transferencia entre registros. Los Lenguajes de Programación de Hardware permiten diseñar sistemas digitales basándose en las transferencias entre registros. Es común ver

estructuras en las cuales aparece un bus (conjunto de líneas lógicas que transporta información) del cual se encuentran “colgados” registros. Estos registros pueden actuar como elementos de interconexión entre el bus y distintas unidades funcionales, o bien, ser registros de almacenamiento temporario de información exclusivamente.

En la Figura 5.18 se presenta una forma de interconexión entre registros llamada BUS COMÚN. Se trata de tres registros de dos bits cada uno. Las entradas de algunos de estos registros podrían estar conectadas a una Unidad Funcional (una ALU, por ejemplo), funcionando en este caso como registro de salida de la misma; o bien, las salidas de alguno de los registros podrían estar conectadas a otra Unidad Funcional (una Unidad de Memoria, por ejemplo) funcionando en este caso como registro de entrada a la misma. En la Figura se indica cómo deberían ser las señales de control para llevar a cabo la transferencia entre registros. Estas señales de control son generadas, en general, por la Unidad de Control del Bus, y de su eficiencia depende en gran medida las prestaciones (velocidad de procesamiento) del Sistema Digital.

Otra forma de construir un sistema de interconexión entre registros es usando registros con salida tri-estado (como el indicado en la Figura 5.17). Un ejemplo puede apreciarse en la Figura 5.19.

En la figura se aprecian tres registros con salida tri-estado. Cada registro, además de dos entradas y dos salidas de información, poseen dos entradas de control: Enable y Control de salida. Estas son manejadas por la Unidad de Control del Bus y por las Unidades Funcionales. Supóngase, por ejemplo, que la Unidad Funcional I ya ha procesado una información y es necesario transferirla al Registro C para que, finalmente, sea procesada por la Unidad Funcional II. Para este ejemplo, la Unidad de Control del Bus deberá realizar lo siguiente:

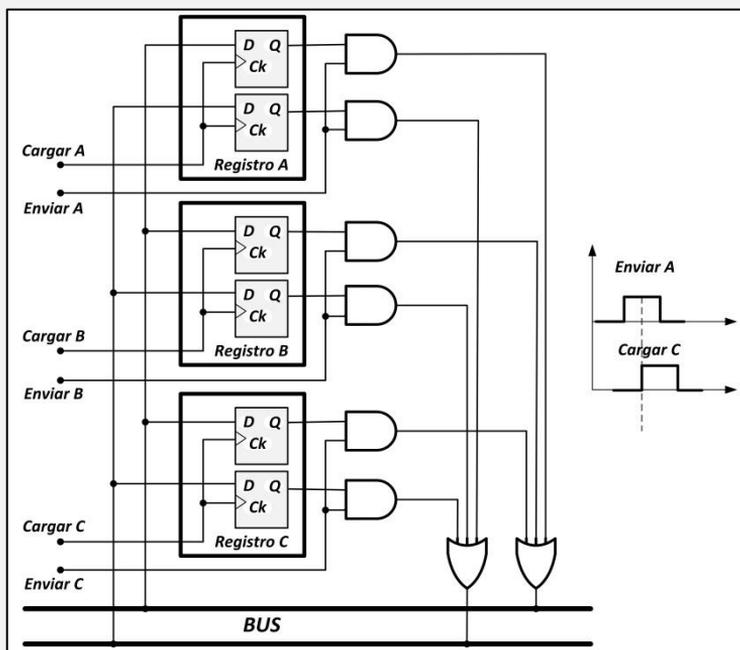


Fig. 5.18. Interconexión de registros por bus común.

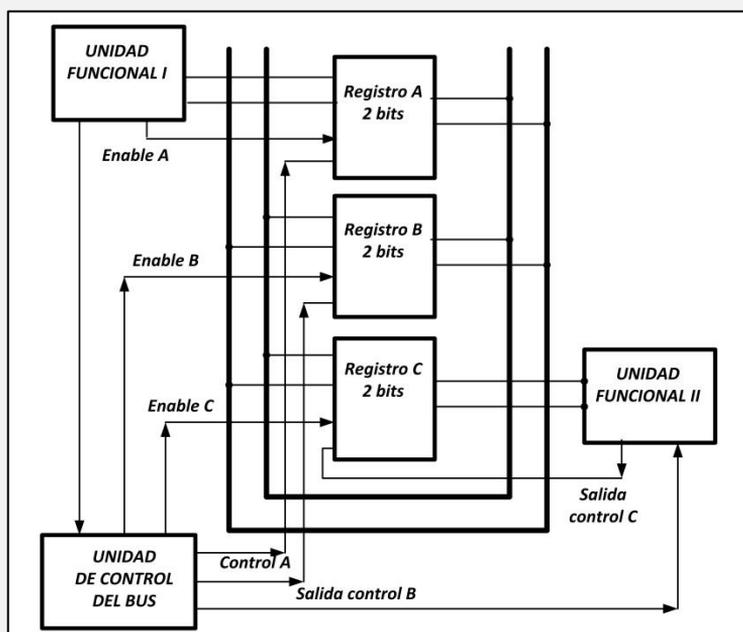


Fig. 5.19. Interconexión de registros usando registros tri-estado.

- Recibir desde la Unidad Funcional I la señal de listo (La Unidad Funcional I, una vez que dispone de la información, la carga en el Registro A con Enable A y luego envía la señal listo a la Unidad de Control del Bus).
- Activar la señal Salida Control A, con lo cual vuelca al bus el contenido del Registro A.
- Activar la señal Enable C, a fin de cargar el Registro C con la información presente en el Bus.
- Desactivar las señales Salida Control A y Enable C.
- Indicar a la Unidad Funcional II que, en el Registro C, existe información a procesar.

La Unidad Funcional II, al recibir la señal de la Unidad de Control del Bus, lee el contenido del Registro C mediante la señal Salida de Control C.

De lo visto se puede intuir la importancia de Bus en los Sistemas Digitales. En el Capítulo 7 se desarrolla un Sistema Digital basado en la estructura de Von Newman, en el que se considera al Bus como una unidad en sí mismo.

## 4.3 Contadores

Es un sistema secuencial formado por biestables y lógica combinacional, capaz de almacenar en binario u otro código, la cantidad de impulsos recibidos por su entrada de cuenta. Puede aplicarse como divisor de frecuencia, control de tiempos, generador de direcciones en sistemas de memoria, secuenciador en unidades de control, etc.

### Contadores Asíncronos

Son secuenciales síncronos formados por un conjunto de biestables síncronos por flancos. Su denominación de asíncrono no se refiere al tipo de secuencial, sino al hecho que las entradas de sincronismo de sus biestables no están unidas entre sí. Por lo general, la salida de un biestable sirve como entrada de sincronismo del siguiente. En la Figura 5.20 se muestra un contador binario de 4 bits asíncrono; obsérvese que las salidas de los biestables se conectan a las entradas de sincronismo del siguiente. También puede verse el diagrama de tiempo de este contador.

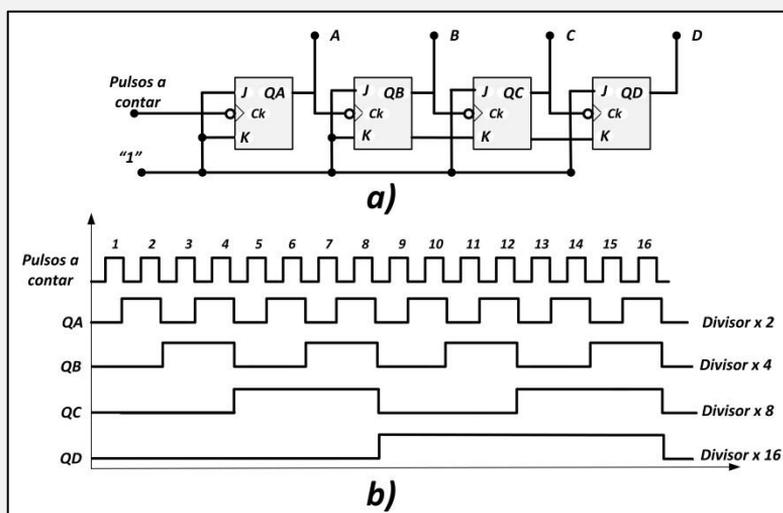


Fig. 5.20. Contador asíncrono de 4 bits. (a) Diagrama circuital (b) Diagrama de tiempo.

### Contadores Síncronos

Son similares a los anteriores, sólo que comparten la misma señal de reloj. Son más rápidos y complejos que los asíncronos. En la Figura 5.21 se muestra un contador binario natural síncrono de 4 bits. Nótese que, a diferencia de la Figura 5.20, este contador tiene todas las entradas de sincronismo de los biestables unidas. Además, es más complejo puesto que tiene más compuertas que el anterior.

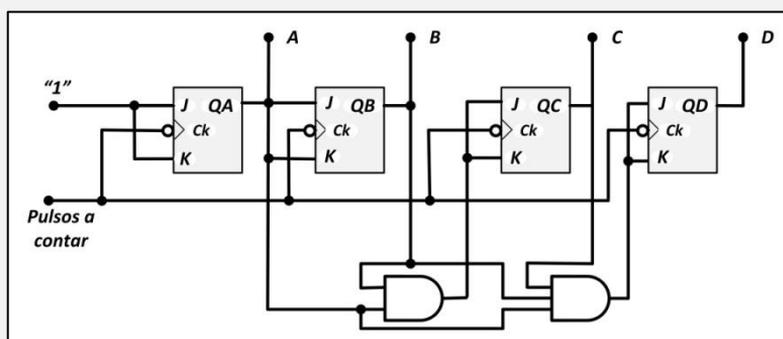


Fig. 5.21. Contador binario natural de 4 bits síncrono.

El diseño de contadores se realiza planteando una tabla de verdad temporal, y luego de obtener las funciones correspondientes, se minimizan teniendo en cuenta el biestable elegido. Los biestables utilizados en los contadores, como así también en los registros de

desplazamiento, son biestables síncrono maestro – esclavo o activados por flancos.

## 4.4 Multiplicación y división binaria

La multiplicación y la división binaria implican circuitos de naturaleza combinacional y secuencial por lo tanto es el momento adecuado para mencionar sus principales características.

### 4.4.1 Multiplicación binaria

La multiplicación es una sucesión de sumas. Por lo tanto, para multiplicar dos números (**MULTIPLICANDO** por **multiplicador**), para obtener el **PRODUCTO**, deberíamos sumar el MULTIPLICADOR tantas veces a si mismo, como indique el multiplicando. Veremos más adelante que la ALU de un procesador siempre incluye un sumador, y entonces, la multiplicación se podría resolver por software mediante un programa que procese las sumas mencionadas. Esta solución requiere un tiempo de producto variable y alto. Para resolver este problema de elevado tiempo de producto y dependiente del valor de los factores, se puede implementar un multiplicador por hardware. En este caso, la ALU incluye en su hardware un circuito que multiplica.

Existen dos alternativas para construir un multiplicador por hardware:

- Multiplicador paralelo o concurrente
- Multiplicador serie o secuencial

#### Multiplicador paralelo

Consiste en aplicar el algoritmo de multiplicación que conocemos desde la escuela primaria. Por ejemplo, para dos números binarios de 4 bits es:

$$\begin{array}{r} \phantom{x} \phantom{000} 1001 \\ x \phantom{000} 0110 \\ \hline \phantom{000} 0000 \\ \phantom{000} 1001 \\ \phantom{000} 1001 \\ \phantom{000} 0000 \\ \hline 0110110 \end{array}$$

Pueden hacerse dos observaciones:

- El resultado tiene 8 bits. Si pretendemos que el producto tenga 4 bits, deberemos chequear el rebasamiento. Es decir, que si multiplicamos dos números de 4 bits, el producto no debe superar la máxima cantidad representable con 4 bits.
- Si se consideran números con signo, los datos se deben convertir a números positivos, tratar el bit de signo separadamente y multiplicar sólo los valores absolutos. Finalmente, se deberá expresar el resultado según el convenio de representación de números negativos utilizado.

El circuito de la Figura 5.22 ejemplifica el hardware de un multiplicador paralelo de 4 bits con detector de rebasamiento (overflow).

La suma comienza cuando se inyectan al circuito el MULTIPLICANDO ( $M_3M_2M_1M_0$ ) y el multiplicador ( $m_3m_2m_1m_0$ ). El conjunto de compuertas AND realizan, en paralelo, los productos parciales que sumarán los 3 sumadores de 4 bits conectados en serie y la compuerta OR genera el overflow. El tiempo de producto ( $TP_p$ ) de este circuito es:

$$TP_p = T_{and} + 3T_{sumador}$$

dónde:

Tand: Es el tiempo de retardo de una compuerta AND

Tsumador: Es el tiempo de retardo del SUMADOR de 4 BITS

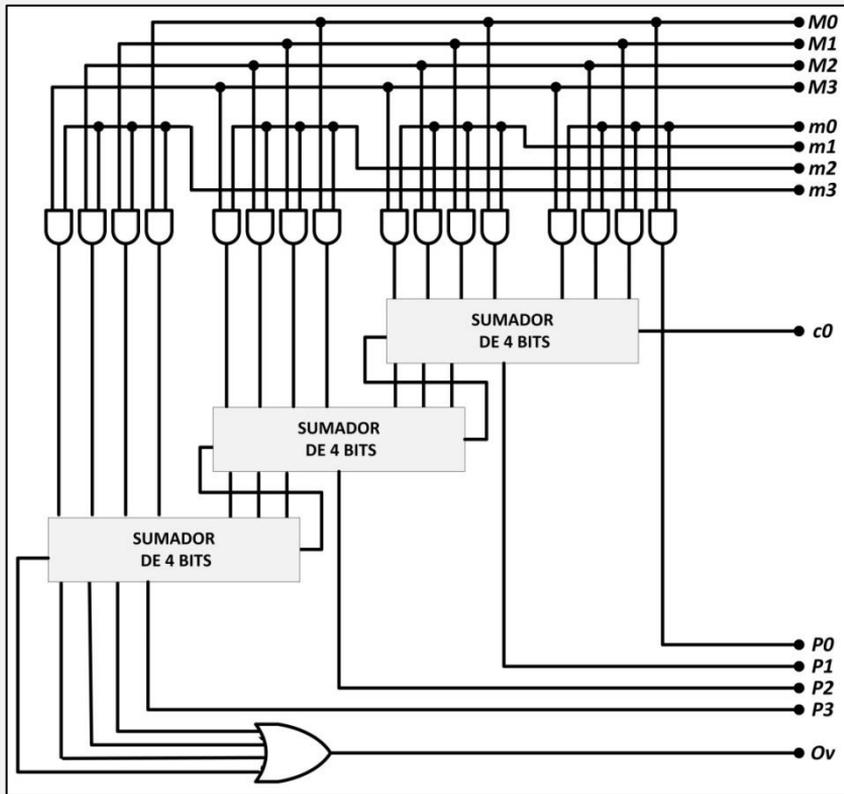


Fig. 5.22 Multiplicador paralelo de 4 bits con generación de overflow

Si tomamos como unidad de retardo el retardo de una AND y lo llamamos  $T$ , tenemos:

$$TP_p = T_{and} + 3.4T_{and}$$

Donde se ha supuesto que  $T_{sumador} = 4T_{and}$ , aproximadamente.

En el caso general de un multiplicador de  $n$  bits, el tiempo de producto es:

$$TP_p = T_{and} + 4(n - 1)T_{and} = (4n - 3)T_{and}$$

Cuando aumenta el número de bits, también aumenta significativamente la complejidad del hardware. Por ejemplo, consideremos multiplicadores de 32 bits. En este caso, se necesitan 31 sumadores de 32 bits. Además, para lograr los tiempos de suma

supuestos en el ejemplo para 4 bits, se requieren generadores de acarreo anticipado de 32 bits cada uno, lo que complica aún más el hardware.

### Multiplicador serie o secuencial

Una aproximación que simplifica el hardware y mejora el promedio de TP (cuando el número de bits es elevado) utiliza un único sumador, al cual se le cambian adecuadamente las entradas mediante el desplazamiento de una de ellas.

El funcionamiento implica una secuencia de señales de control. Por este motivo se lo llama multiplicador secuencial.

Cuando el multiplicador se usa como parte de una ALU de un procesador, esta secuencia de señales de control puede generarse con:

- Un circuito de control especial (en una Unidad de Control Cableada - Capítulo 7) y se llama multiplicación serie por hardware
- Un microprograma (en una Unidad de Control Microprogramada - Capítulo 7) y se llama multiplicación por firmware.

En la Figura 5.23 se observa un multiplicador serie de 4 bits.

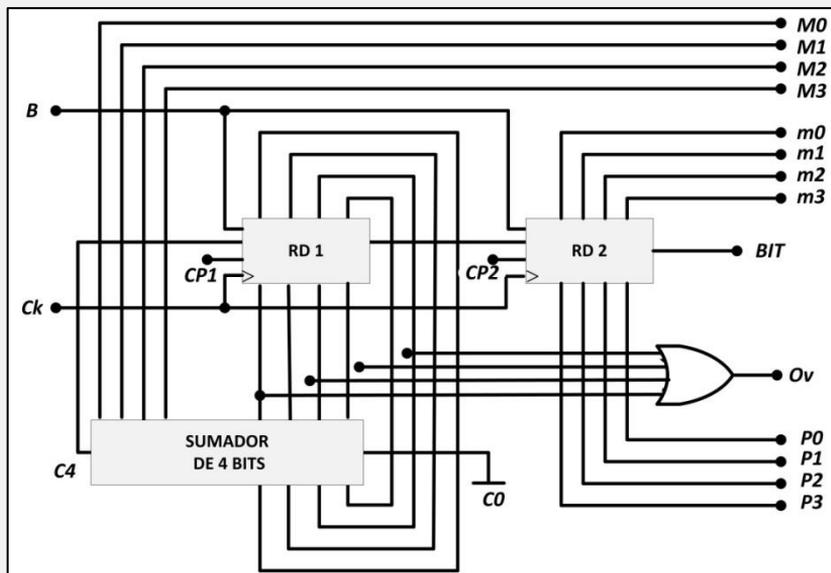


Fig. 5.23 Multiplicador serie de 4 bits con generador de overflow

RD1 y RD2 son registros de desplazamiento paralelo-serie de 4 bits cuyo contenido aparece en las salidas inferiores del diagrama. Las señales que controlan estos RD son:

- B: borra los RD
- CP1: carga el RD1
- CP2: carga el RD2
- Ck: desplaza un bit hacia la derecha los contenidos de RD1 y RD2

A continuación se describe el funcionamiento del multiplicador. Una vez borrados los RD se suministra el MULTIPLICANDO y el multiplicador. Este último se carga en RD2 mediante la señal CP2. Luego, se dispara 4 veces la siguiente secuencia:

- 1) Si BIT = 1 se espera al SUMADOR de 4 BITS y su resultado se carga en RD1 mediante la señal CP1. Si BIT = 0 no se hace nada y pasamos a 2).
- 2) Se desplaza hacia la derecha los contenidos de RD1 y RD2 mediante la señal Ck.
- 3) Terminar si se ha repetido 4 veces, si no saltar a 1).

El tiempo de la multiplicación ( $TP_s$ ) para este circuito es:

$$TP_s = (T_{cp1} + T_{sumador} + T_d)4$$

dónde:

$T_{cp1}$ : tiempo de carga de RD1

$T_{sumador}$ : tiempo de suma del SUMADOR de 4 BITS.

$T_d$ : tiempo necesario para desplazar los contenidos de RD1 y RD2.

Si aproximamos:

$$T_{cp1} = T_d = T$$

y

$$T_{sumador} = 4T$$

tenemos que:

$$TP_s = (2T + 4T)4$$

En el caso de un multiplicador de  $n$  bits, el tiempo de producto es:

- si el multiplicador tiene todos 1s:
  - $TP_s = 6nT$
- Si el multiplicador tiene todos 0s
  - $TP_s = nT$

Finalmente, si consideramos que en promedio los números tienen igual cantidad de 1s y 0s, el tiempo de multiplicación será la media:

$$TP_s = 3,5nT$$

Cuando aumenta el número de bits, la complejidad del hardware aumenta, pero no significativamente como en el caso del multiplicador paralelo. Por ejemplo, para construir un multiplicador serie de 32 bits, solo es necesario un único sumador de 32 bits y dos registros de desplazamiento de 16 bits.

En la Tabla 5.4 se comparan los tiempos de producto aproximados para las dos alternativas planteadas:

n	TPp	TPs
4	13T	14T
16	61T	56T
32	125T	112T
64	253T	224T

Tabla 5.4 Comparativa aproximada de los tiempos de producto

Se ve que los multiplicadores serie son más rápidos y menos complejos que los multiplicadores paralelo. Es válido comentar que si en un multiplicador paralelo paralelizamos los acarreo de salida de cada uno de los sumadores parciales que lo componen, lograríamos un tiempo de producto independiente del número de bits de cada factor. Este TPp estaría en el orden de:

$$TP_p = 5T + T_{ga}$$

dónde:

Tga: tiempo de propagación de un generador de acarreo anticipado de n bits

Si bien este tiempo es menor que los vistos anteriormente, resulta un hardware muy complejo.

## 4.4.2 División binaria

La DIVISIÓN es una sucesión de RESTAS. Por lo tanto, para DIVIDIR dos números (**DIVIDENDO** sobre **divisor**) para obtener un **COCIENTE** y un **RESTO**, deberíamos restar del **DIVIDENDO** el **divisor** tantas veces hasta obtener un **RESTO** menor que el **DIVISOR**. Como dijimos anteriormente, la ALU de un procesador siempre incluye un sumador, por lo tanto la división se podría resolver por software mediante un programa. Esta solución consume un tiempo de división variable y alto. Para resolver este problema algunas ALUs incluyen en su hardware un circuito que implementa la división.

También este caso, se aplica el algoritmo de división que se aprende tempranamente en la escuela. Por ejemplo, la división entera de dos números binarios sin signo de 4 bits, es así:

$$\begin{array}{r} 1101 \\ - 10 \\ \hline 10 \\ - 10 \\ \hline 01 \end{array} \quad \begin{array}{r} 0010 \\ \hline 0110 \end{array}$$

El algoritmo puede describirse de la siguiente manera:

- Tomamos el MSB del DIVIDENDO y lo comparamos con el divisor. Si es menor que el divisor, colocamos un 0 en el COCIENTE (representará el MSB del COCIENTE). Luego, tomamos los 2 MSB del DIVIDENDO y comparamos nuevamente. Si sigue siendo menor al divisor colocamos otro 0 en el COCIENTE. Seguimos así hasta que no sea menor y, en tal caso, restamos a los bits considerados del DIVIDENDO, el divisor. Además colocamos un 1 en el COCIENTE.
- Seguimos con este procedimiento hasta agotar los bits del DIVIDENDO, el resultado de la última resta será el RESTO de la división.

Para resolver el hardware de la división pueden construirse divisores paralelos (o concurrentes) y divisores serie (o secuenciales). Los primeros resultan en circuitos de gran complejidad. En cambio los divisores serie resultan circuitos relativamente simples.

En la Figura 5.24 se observa como ejemplo un divisor serie de 4 bits.

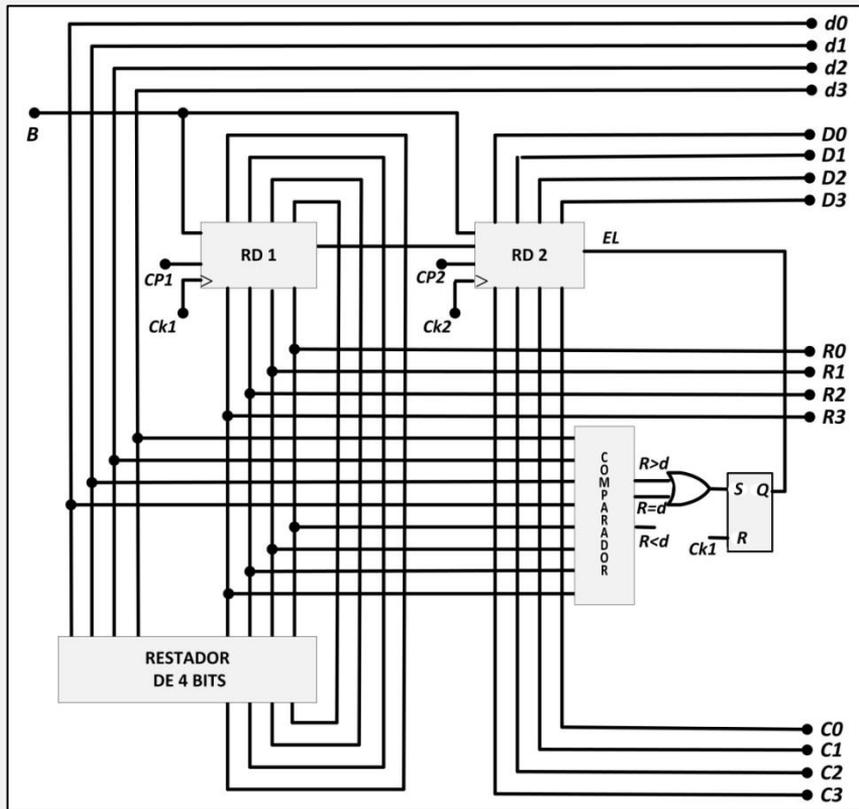


Fig. 5.24 Circuito divisor serie de 4 bits

RD1 y RD2 son registros de desplazamiento paralelo serie con disponibilidad de sus contenidos. El comparador compara el RESTO (contenido en RD1) con el divisor. La salida de la compuerta OR indica si el RESTO es mayor o igual al divisor y se usa para generar el cociente (a través del biestable S-R) y determinar si se realiza o no una resta.

Los operandos y el resultado están identificados de la siguiente forma:

- DIVIDENDO  $D = D_3D_2D_1D_0$  (entrada)
- Divisor =  $d_3d_2d_1d_0$  (entrada)
- COCIENTE =  $C_3C_2C_1C_0$  (salida)
- RESTO =  $R_3R_2R_1R_0$  (salida)

El funcionamiento del circuito se puede describir según la siguiente secuencia que debe repetirse tantas veces como bits tengan los números:

- 1) Se inicializa  $RD_1 = 0$ ,  $RD_2 = D$  y el biestable  $SR = 0$
- 2) Se desplaza un bit a la izquierda  $RD_1$  y  $RD_2$  (esto se hace con  $Ck_1=Ck_2=1$ ). Si es la última repetición sólo desplazar  $RD_1$  (esto se hace con  $Ck_1$ ) y terminar. El contenido de  $RD_2$  es el COCIENTE y el contenido de  $RD_1$  es el RESTO.
- 3) Si  $RD_1$  es mayor o igual que el divisor, resta y carga el resultado en  $RD_1$  (mediante  $Ck_1$ ). Si  $RD_1$  es menor que el divisor no hacer nada
- 4) Si faltan repeticiones saltar a 2)

En la tabla 5.5 se presenta una verificación del funcionamiento para  $D = 1101$  y  $d = 0010$ .

El tiempo de división de este circuito es similar al tiempo de producto  $TPs (3,5nT)$ .

Comparando las Figuras 5.23 y 5.24 se ve que los circuitos del Multiplicador y el Divisor son similares. Si disponemos de un sumador/restador y de registros de desplazamiento reversibles (que desplacen hacia la izquierda o la derecha según una señal de control) se puede usar el mismo hardware para multiplicar o dividir.

Como en el caso del multiplicador serie, un divisor serie en una ALU puede controlarse por señales de control generadas por hardware con una Unidad de Control cableada. En este caso, se dice que la ALU cuenta con división por hardware. Si las señales de control son generadas por un microprograma con una Unidad de Control microprogramada, se dice que la ALU tiene división por firmware.

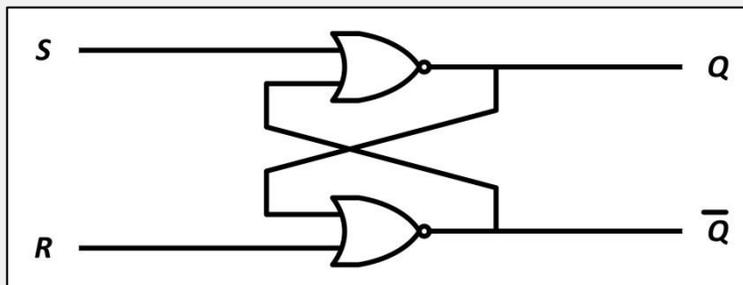
Evolución	Resto (RD1)	Dividendo y Cociente (RD2)		Comentarios
1	0000	1101		Inicialización EL = 0
2	0001	1010		EL = 0, desplaza RD1 y RD2
3	0001	1010		No hace nada, RD1 menor d
4(2)	0011	0100	1	EL = 0, desplaza RD1 y RD2
3	0001	0100		Resta, resultado en RD1
4(2)	0010	1001	2	EL = 1, desplaza RD1 y RD2
3	0000	1001		Resta, resultado en RD1
4(2)	0001	0011	3	EL = 1, desplaza RD1 y RD2
3	0001	0011		No hace nada, RD1 menor d
4(2)	0001	0110	4	EL = 0, solo desplaza RD2 y termina

Tabla 5.5 Secuencia de control del circuito divisor de la Figura 5.24

## 5 Ejercitación

### Ejercicio 1:

Realizar la tabla de verdad de este biestable.



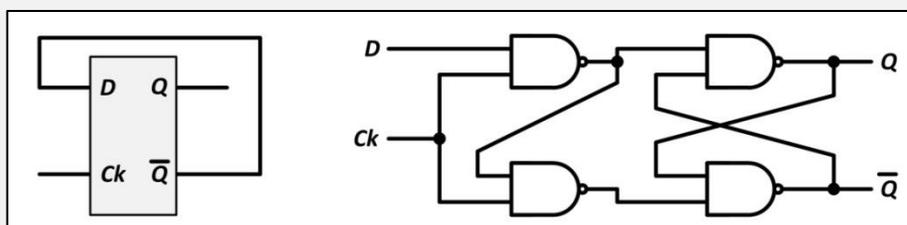
**Ejercicio 2:**

Explicar el significado del siguiente cuadro que con brevedad sintetiza la utilización del biestable J-K como R-S, T ó D.

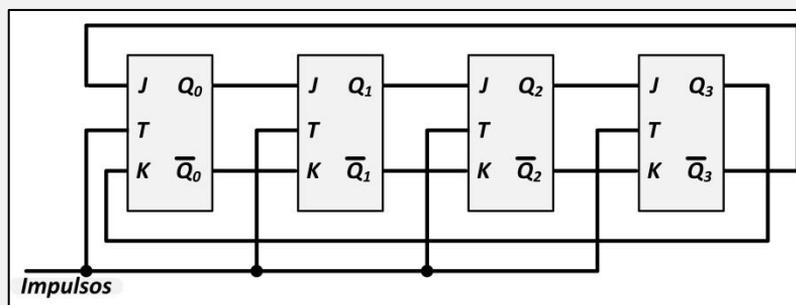
	JK	Qt+1	
J=K=T	00	Q	J=S; K=R
J=R=D	01 10	1 1	J=S; K=R
J=K=T	11	Q	

**Ejercicio 3:**

Deducir si el siguiente flip-flop es síncrono o asíncrono, y cuál es su funcionamiento.

**Ejercicio 4:**

Determinar en qué código se lleva la cuenta usando el siguiente registro de desplazamiento realizado con biestables J-K. Suponer que inicialmente  $Q_0=Q_1=Q_2=Q_3=0$ .



**Ejercicio 5:**

Realizar los diagramas en bloque de los cuatro registros de desplazamiento: entrada serie salida serie, entrada serie salida paralelo, entrada paralelo salida serie y entrada paralelo salida paralelo.

**Ejercicio 6:**

- a) A partir de un biestable SR, obtener un JK.
- b) A partir de un biestable JK, obtener un biestable T.
- c) A partir de un biestable SR, obtener un biestable D.

**Ejercicio 7:**

Determinar el Sistema combinacional necesario para indicar que los contadores binarios han alcanzado el valor equivalente decimal 52, e inicie el conteo nuevamente desde 0.

---

# CAPÍTULO 6

---

## Memorias Electrónicas

- 1 Visión General**
    - 1.1 Introducción
    - 1.2 Clasificación de las memorias electrónicas
  - 2 Memorias de Acceso Aleatorio (RAM)**
    - 2.1 Definición
    - 2.2 Memorias RAM de lectura/escritura
    - 2.3 Memorias RAM de sólo lectura (ROM)
    - 2.4 Extensión de longitud de palabra y capacidad
  - 3 Memorias de Acceso Serie**
    - 3.1 Definición
    - 3.2 Registros de desplazamiento
    - 3.3 Memorias FIFO
    - 3.4 Memorias LIFO
  - 4 Ejercitación**
-

## Capítulo 6

# Memorias Electrónicas

## 1 Visión General

### 1.1 Introducción

Las memorias son los dispositivos de almacenamiento de datos e instrucciones en una computadora. Llamamos sistema de memoria al conjunto de estos dispositivos y los algoritmos de hardware y/o software de control de los mismos. Diversos dispositivos son capaces almacenar información, lo deseable es que el procesador tuviese acceso inmediato e ininterrumpido a la memoria, a fin de lograr la mayor velocidad de procesamiento. Desafortunadamente, memorias de velocidades similares al procesador son muy caras. Por esta razón la información almacenada se distribuye en forma compleja en una variedad de memorias diferentes, con características físicas distintas.

Una clasificación funcional de las memorias es la siguiente:

- a) **Memoria interna:** Constituida por los registros internos de la CPU (Unidad Central de Procesos o Procesador). Este tipo de memoria se estudia en el Capítulo 9. Se caracteriza por su alta velocidad.
- b) **Memoria central (o principal):** Almacena programas y datos. Es relativamente grande, rápida, y es accedida directamente por la CPU a través de un bus. Este tipo de memoria es parte de esta Guía Didáctica.
- c) **Memoria secundaria:** Se usa para el almacenamiento de programas del sistema y grandes archivos. Su capacidad es mucho mayor que las anteriores, pero más lenta. El acceso a la misma por parte de la CPU es indirecto. Las principales tecnologías son la magnética y la óptica.

Se pueden definir algunos parámetros generales aplicables a todas las memorias:

- a) **Unidad de almacenamiento:** Bit.

- b) **Capacidad de almacenamiento:** Cantidad de bits que puede almacenarse. Si bien la unidad de almacenamiento es el bit, muchas veces se usa el byte. Así encontramos capacidades en Kb (1Kb = 1024 bytes), en Mb (1Mb = 1024 Kb), en Gb (1Gb = 1024 Mb), etc.. Las memorias se consideran organizadas en palabras. Cada palabra es un conjunto de bits a los cuales se accede simultáneamente.
- c) **Tiempo de acceso ( $t_a$ ):** Es el que se tarda en leer o escribir una palabra en la memoria desde el momento que se direcciona. La velocidad de acceso  $ba=1/t_a$  se mide en palabras/segundo (Figura 6.1).
- d) **Tipo de acceso:**
- Acceso aleatorio: cuando el tiempo de acceso es similar para cualquier posición, y
  - Acceso serie: cuando el tiempo de acceso depende de la posición que ocupa la palabra dentro de la memoria.
- e) **Tiempo de ciclo ( $t_c$ ):** Indica el mínimo tiempo entre dos accesos sucesivos a la memoria. El tiempo  $t_c$  es mayor que el tiempo  $t_a$ . El ancho de banda de una memoria se define como la inversa de  $t_c$  y es un indicativo de la cantidad de palabras procesables por unidad de tiempo.
- f) **Medio físico**
- Electrónicas: construidas con semiconductores.
  - Magnéticas: basadas en el fenómeno de histéresis de los materiales ferromagnéticos.
  - Ópticas: utilizan la tecnología láser.
- g) **Estabilidad**
- Volatilidad: el contenido de la memoria se pierde cuando se suspende la alimentación eléctrica.
  - Almacenamiento dinámico: El bit se almacena como carga de una capacidad parásita de un transistor MOS. La información se pierde cuando el capacitor se descarga, lo que hace necesario un refresco periódico para restaurar el contenido antes que se deteriore.
  - Lectura destructiva (DRO): Al efectuar la lectura se pierde la información, por lo cual dicho proceso debe acompañarse de una restauración (Tabla 6.1).

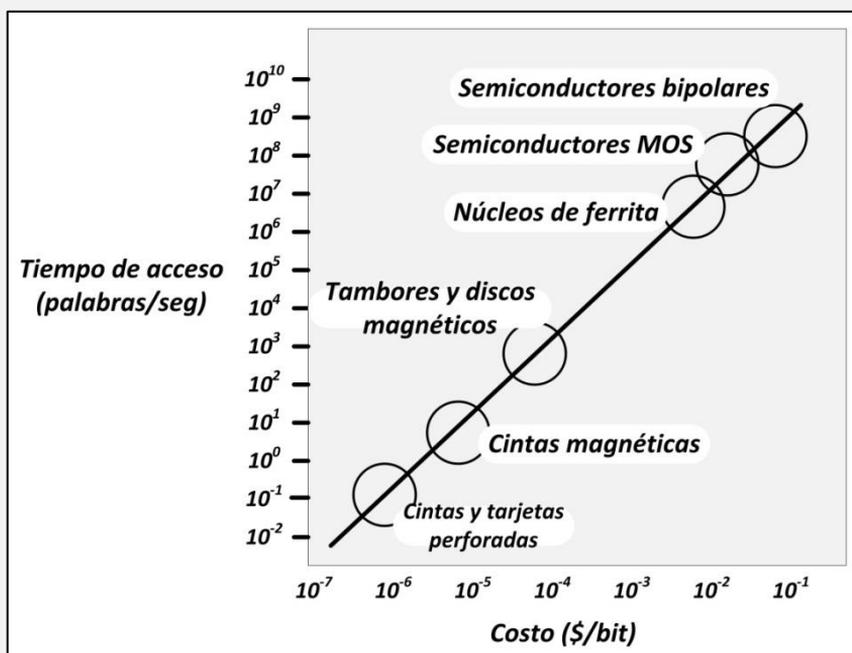


Fig. 6.1. Tiempo de acceso (palabras/seg) en función del costo (\$/bit).

Tecnología	Costo (\$/bit)	Tiempo acceso	Modo acceso	Alterabilidad	Estabilidad	Medio físico Almacenamiento
Bipolar semiconductor	$10^{-1}$	$10^{-8}$	aleatorio	Lectura/escritura	NDRO volátil	electrónico
Metal—Oxido Semicond (MOS)	$10^{-2}$	$10^{-7}$	aleatorio	Lectura/escritura	DRO o NDRO volátil	electrónico
Núcleos de ferrita	$10^{-2}$	$10^{-6}$	aleatorio	Lectura/escritura	DRO no volátil	magnético
Discos y tambores magnéticos	$10^{-4}$	$10^{-2}$	Aleatorio o semialeatorio	Lectura/escritura	NDRO no volátil	magnético
Cintas magnéticas	$10^{-5}$	$10^{-1}$	serie	Lectura/escritura	NDRO no volátil	magnético
Tarjetas y papel perforado	$10^{-6}$	10	serie	Solo lectura	NDRO no volátil	mecánico

Tabla 6.1. Clasificación de las memorias usando sus principales características.

## 1.2 Clasificación de las memorias electrónicas

Las memorias electrónicas pueden considerarse como un sistema digital mixto (combinacional y secuencial) capaz de almacenar información binaria el cual se puede acceder (introducir o extraer información) sólo parcialmente en un momento dado.

En función del tipo de acceso, estas memorias se clasifican en:

- a) **Memorias de acceso aleatorio (RAM)**, en las que ta es similar para cualquier posición. Se subdividen en:
  - **Memorias de lectura/escritura**, también llamadas activas. Se caracterizan por tener los ta de lectura y escritura similares, presentan volatilidad, pierden su contenido cuando dejan de estar alimentadas. Se subdividen en:
    - **Memorias estáticas (SRAM)**
    - **Memoria dinámicas (DRAM)**
  - **Memorias de sólo lectura (ROM)**, también llamadas pasivas. Se caracterizan por tener el ta de escritura en mucho mayor que el de lectura, presentan no volatilidad, no pierden su contenido sin alimentación. Se subdividen en:
    - **ROM**, se graban una vez por el fabricante.
    - **PROM**, se graban una vez por el usuario.
    - **EPROM**, se graban varias veces por el usuario, el borrado se realiza con luz ultravioleta.
    - **EEPROM**, se graban varias veces por el usuario, el borrado se realiza eléctricamente posición a posición.
    - **FLASH**, se graban varias veces por el usuario, el borrado se realiza eléctricamente de una sola vez.
- b) **Memorias de acceso serie**, en las que el tiempo de acceso ta depende de la posición de la palabra dentro de la memoria. Son memorias de lectura/escritura. Se subdividen en:
  - **Registros de desplazamiento**,
  - **Memorias pila (LIFO)**, última escritura, primera lectura
  - **Memorias cola (FIFO)**, primera escritura, primera lectura.

## 2 Memorias de Acceso Aleatorio (RAM)

### 2.1 Definición

Desde los 60 aparecen los circuitos integrados que permiten construir memorias de alta capacidad. Actualmente se encuentran memorias semiconductoras del orden de los Gb. Podemos considerar la memoria como un conjunto de posiciones, donde cada una de ellas está formada por una o más celdas (células elementales). El esquema general de una memoria de acceso aleatorio puede verse en la Figura 6.2 y el diagrama en bloque en la Figura 6.3.

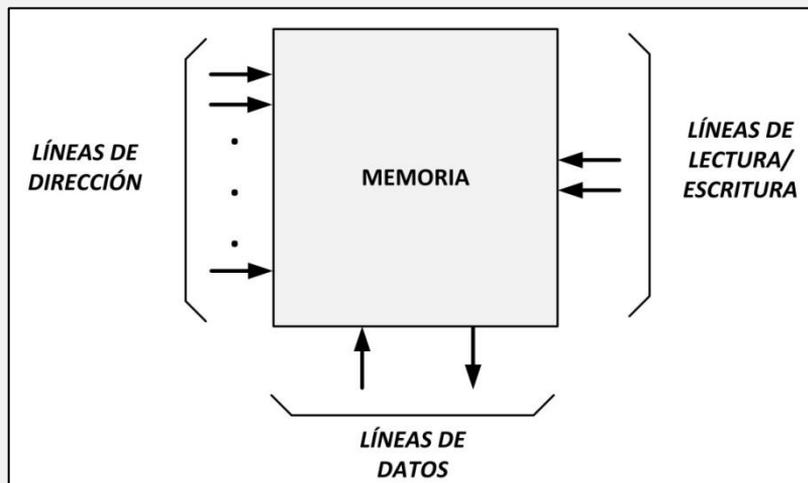


Fig. 6.2. Esquema general de una memoria RAM.

El tipo de celda depende de la clase de memoria que se trate y la tecnología utilizada. En las RAM de lectura/escritura, las celdas consisten en biestables asíncronos como los estudiados en el Capítulo 5. En las RAM de sólo lectura (ROM) consisten en diodos o transistores.

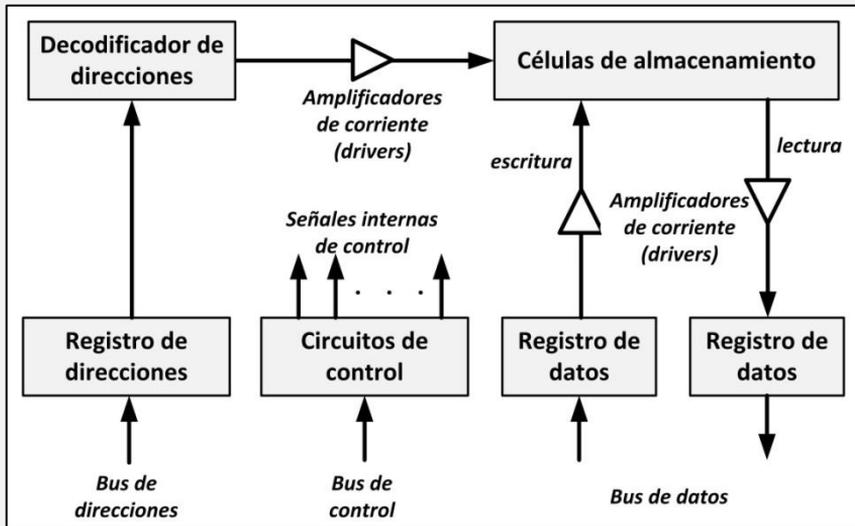


Fig. 6.3. Diagrama en bloques de una memoria RAM.

Las memorias RAM operan de la siguiente manera:

- Una dirección (conjunto de  $m$  bits) se transfiere al registro de direcciones,
- El decodificador de direcciones procesa la dirección y selecciona una posición de memoria,
- La posición seleccionada se lee o escribe en función de las señales de control,
- Si es una lectura, el contenido de la posición seleccionada se transfiere al registro de datos de salida (de  $n$  bits). Si es una escritura (para el caso de una RAM de lectura/escritura) se transfiere el registro de datos de entrada (que debe haber sido cargado anteriormente) a la posición seleccionada.

La organización interna de las memorias RAM puede ser 2D o 3D:

**Organización 2D (Bidimensional):** Las celdas se organizan en una matriz de dos dimensiones, en la que las filas vienen dadas por el número de palabras ( $N$ ) y las columnas por la longitud (cantidad de bits) de cada palabra (Figura 6.4). Cada celda binaria se accede por una sola línea de selección.

Esta organización se usa en memorias de pequeña capacidad.

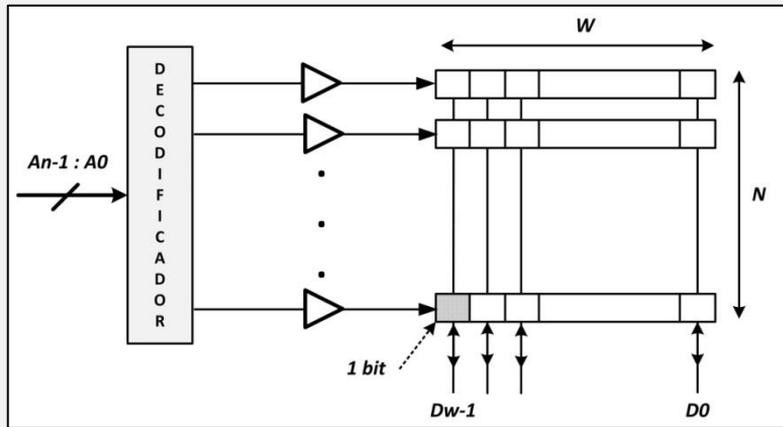


Fig. 6.4. Esquema de una memoria RAM 2D.

**Organización 3D (Tridimensional):** Cada celda binaria se accede por dos líneas de selección. La activación simultánea de ambas determina la selección de la celda. Así se logra reducir el tamaño de los decodificadores. (Figura 6.5).

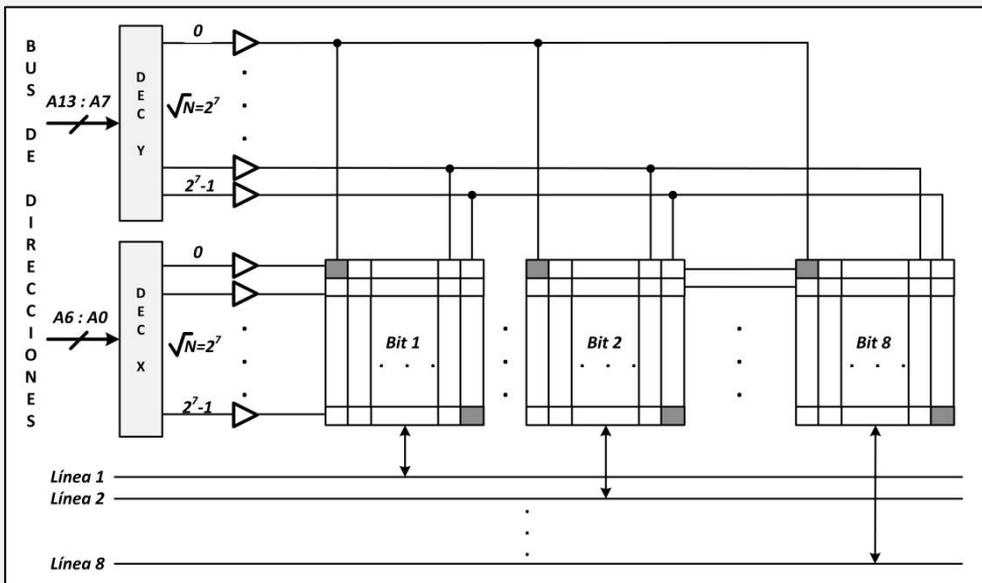


Fig. 6.5. Esquema de una memoria RAM 3D.

Se destaca que la cantidad líneas de salidas del decodificador de una organización 2D es igual a:

$LS2D = 2^m$ , donde m en la cantidad de líneas de direccionamiento

Este valor en la cantidad de líneas de salidas del decodificador se reduce en una organización 3D a:

$LS_{3D} = 2 \cdot 2^{m/2}$  considerando a los dos decodificadores iguales.

La reducción de líneas se logra a costa de agregar un decodificador y una compuerta AND por cada palabra.

## **2.2 Memorias RAM de lectura/escritura**

### **2.2.1 Memorias RAM de lectura/escritura estáticas**

El elemento básico (celda elemental) de estas memorias consiste en un biestable asíncrono como el estudiado en la Guía Didáctica, y algunas compuertas adicionales para manejar la selección y el control de la celda.

En la Figura 6.6 se muestra la celda básica, para el caso de una organización 2D.

Se observa que si la línea de selección está activa con un 1 lógico, se habilita la celda para lectura/escritura. Si  $L/\bar{E} = 1$  se trata de una operación de lectura, y las entradas al biestable se bloquean y se habilita la compuerta AND de salida. Si  $L/\bar{E} = 0$ , se bloquea la compuerta AND de salida y se habilita la entrada al biestable.

En la Figura 6.7 se muestra la celda básica, para el caso de una organización 3D.

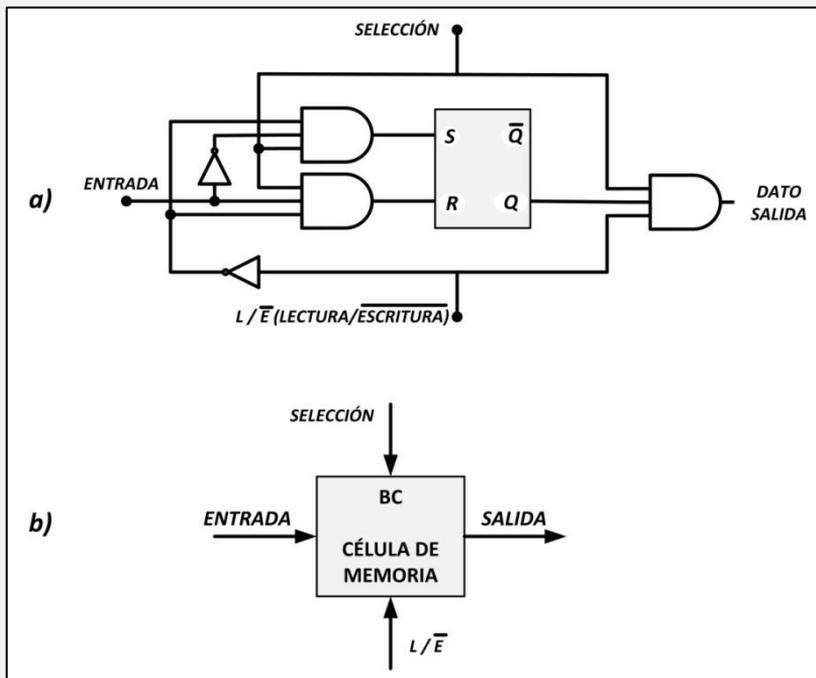


Fig. 6.6. Celda básica para una organización 2D.

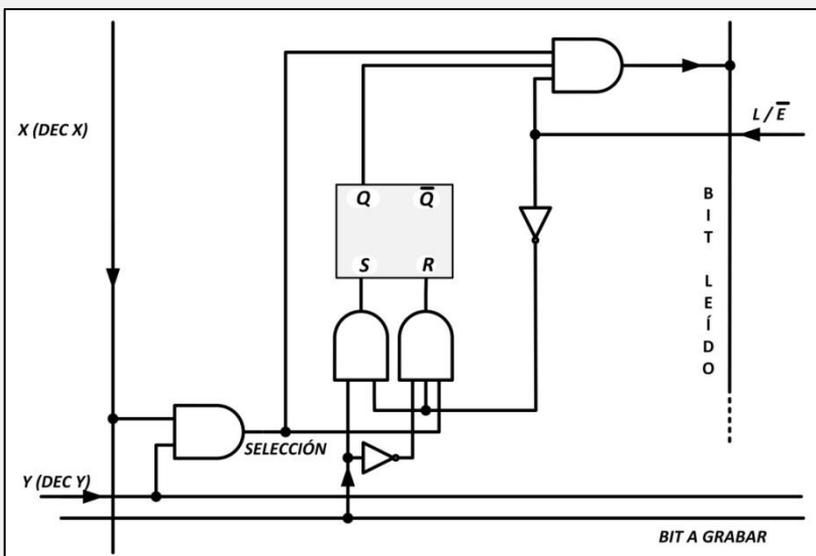


Fig. 6.7. Celda básica para una organización 3D.

Se observa que hay una compuerta AND de dos entradas que completa la selección.

En la Figura 6.8 se muestra una RAM de lectura/escritura con organización interna 2D (que usa la celda básica de la Figura 6.6) de 16 palabras de 4 bits.

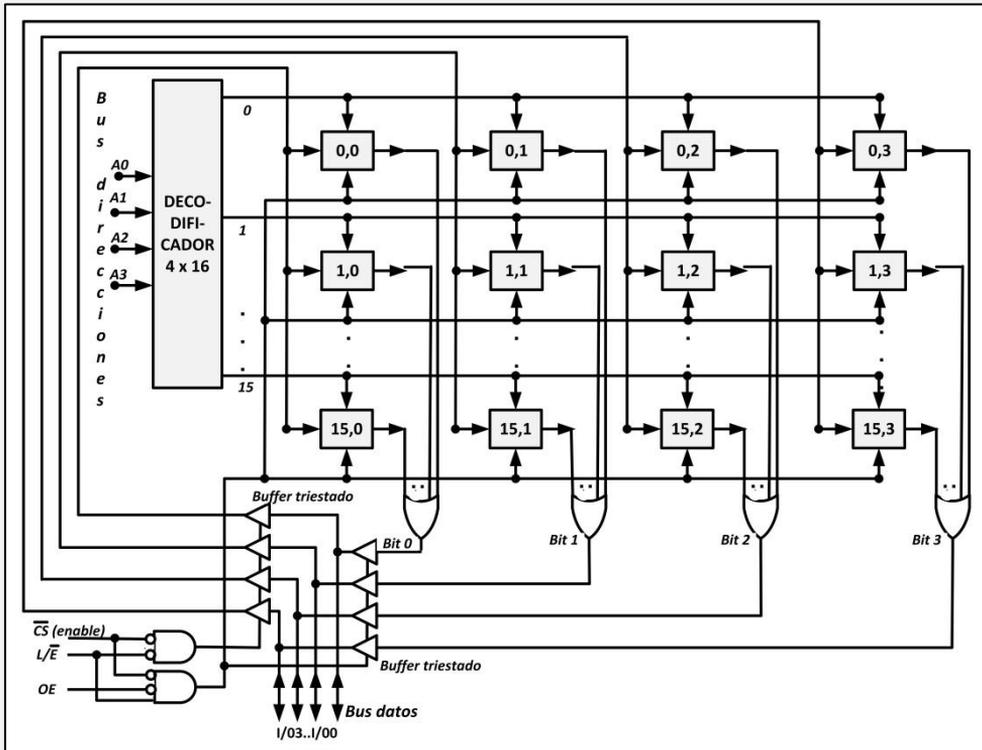


Fig. 6.8. RAM de lectura/escritura con una organización 2D.

Las líneas de acceso a la memoria son:

- a) **Líneas de direccionamiento A0:A3:** Se trata de 4 líneas unidireccionales para la selección de la palabra a acceder. Se usa una de las 16 combinaciones posibles de los 4 bits asociados a las 4 líneas.
- b) **Líneas de datos I/O 01:03:** Se trata de 4 líneas bidireccionales que pueden actuar como entradas o salidas. Para lograr líneas bidireccionales se usan los buffer triestado. De esta forma se evita usar líneas de entrada y salida independientes.

c) **Señales de control de lectura escritura ( $L/\bar{E}$ ):**

- Si  $L/\bar{E} = 1$  se lee la memoria, y
- Si  $L/\bar{E} = 0$  se escribe en la memoria.

d) **Señales de control:**

- Si  $\bar{CS} = 1$  las líneas de datos se colocan en alta impedancia
- Si  $\bar{CS} = 0$  y  $L/\bar{E} = 0$  las líneas de datos se conectan al bus, y el contenido de la posición direccionada se vuelca al bus.
- Si  $\bar{CS} = 0$  y  $L/\bar{E} = 1$  las líneas de datos de entrada se conectan al interior de la memoria y las líneas de datos de salida se colocan en alta impedancia.
- Si  $\bar{OE} = 1$  se deshabilitan los circuitos de salida de la memoria sin tener en cuenta el estado de las señales  $\bar{CS}$  y  $L/\bar{E}$ .

### Ciclo de lectura y ciclo de escritura

Para una correcta operación de la memoria se necesita una temporización adecuada de las señales aplicadas a sus líneas. En la Tabla 6.2 se presentan las combinaciones posibles de las líneas de control para la escritura o lectura, o habilitación de la memoria, representada esquemáticamente en la Figura 6.9.

$\bar{CE}$	$R/\bar{W}$	ACCIÓN
0	1	Operación de lectura
0	0	Operación de escritura
1	X	Memoria deshabilitada. Líneas de datos en alta impedancia

Tabla 6.2. Combinaciones de las líneas de control para escritura o lectura.

Existe una variedad de memorias, y cada una de ellas requiere de su propia temporización. El fabricante provee los diagramas de tiempo que involucran las señales de la memoria.

En la Figuras 6.10 se observa el diagrama de tiempo, para la operación de lectura o de escritura de una memoria más o menos general.

Un ciclo de lectura o escritura comienza con la aplicación de una dirección en las líneas de direccionamiento (bus de direcciones), la línea  $\overline{CS}$  ( $\overline{CE}$  como se ve en la Tabla 6.2) debe estar en cero desde aproximadamente al mismo momento.

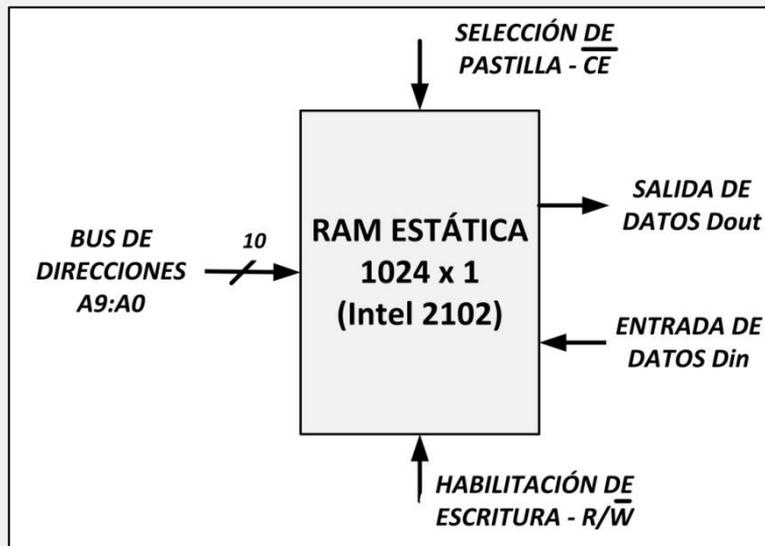


Fig. 6.9. Diagrama en bloques de una memoria RAM estática.

Si es una lectura,  $R/\overline{W}$  ( $L/\overline{E}$ ) debe colocarse en 1. Los datos a leer aparecerán en las líneas de salida de datos (Dout) al cabo del tiempo  $T_A$ . Este es el tiempo de lectura.

Si es una escritura,  $R/\overline{W}$  debe colocarse en 1 un tiempo mínimo  $t_{AW}$  (tiempo de fijación de la dirección), después del cual debe pasar a 0 para indicar una operación de escritura. Este valor debe mantenerse al menos el tiempo  $t_{WP}$  (ancho del pulso de escritura) para garantizar que los datos se hayan almacenado en la RAM. Los datos a escribir deben estar en las líneas de entrada de datos ( $D_{in}$ ) aproximadamente en el momento que aparece la nueva dirección y mantenerse hasta después del tiempo  $t_{WP}$ .

El tiempo  $t_C$  es el tiempo de ciclo, indicativo de la cantidad de operaciones sucesivas por unidad de tiempo. Se observa que  $t_C$  es siempre mayor que  $t_a$ .

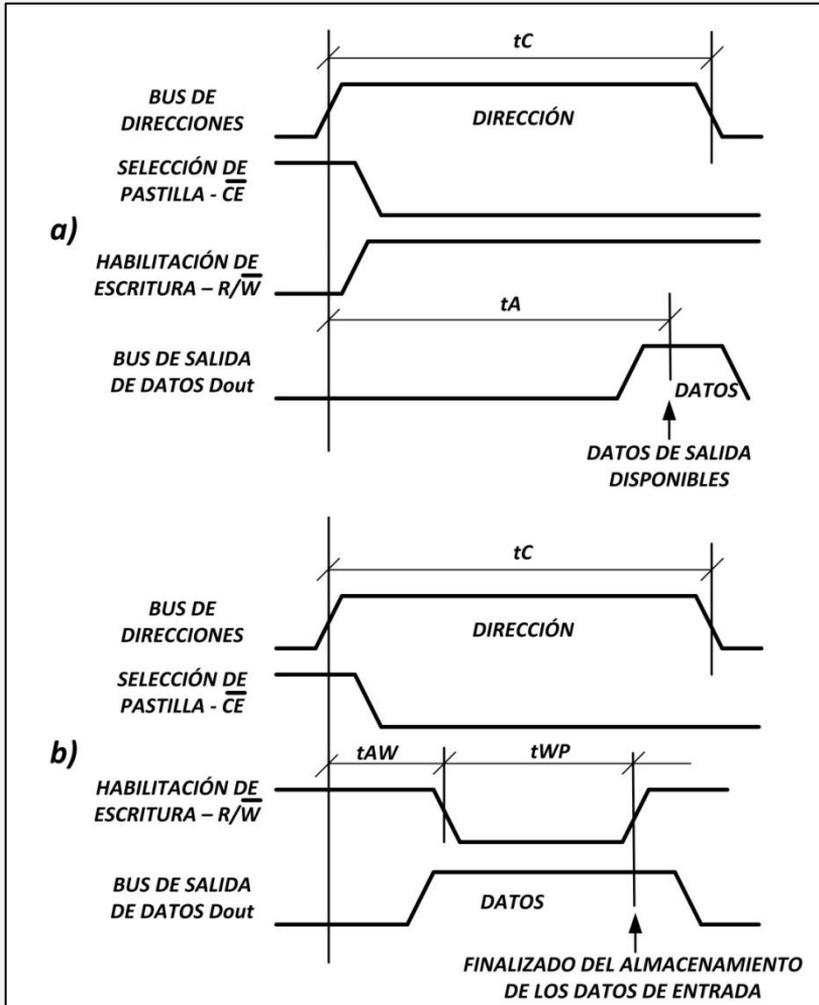


Fig. 6.10. Diagrama de tiempo para la operación de lectura y escritura.

### 2.2.2 Memorias RAM de lectura/escritura dinámicas

En estas memorias la celda básica consiste en la capacidad parásita de una compuerta de un transistor MOS y los circuitos asociados de control. Puesto que debido a las pérdidas inevitables el capacitor se descarga, es necesario restaurar periódicamente la información mediante un proceso que se llama refresco. Este refresco consiste en una lectura seguida de una escritura automática cada aproximadamente 2 ms (dependiendo de la memoria). Cuando está

actuando el proceso de refresco, no es posible acceder a la memoria. La velocidad de estas memorias es menor que las estáticas, sin embargo la densidad de integración es apreciablemente mayor.

La estructura interna de estas memorias es generalmente 3D, con el mismo número de filas (ROW) y columnas (COLUMN). Los  $m$  bits de dirección están divididos en  $m/2$  para las filas y  $m/2$  para las columnas (Figura 6.11). Generalmente, las líneas de dirección están multiplexadas en el tiempo, es decir, desde el exterior se suministran primero los  $m/2$  bits de filas que se cargan en el registro de filas con la señal RAS, luego se suministran los  $m/2$  bits de columnas que se cargan en el registro de columnas con la señal CAS. De esta manera se reduce el bus de direcciones a la mitad.

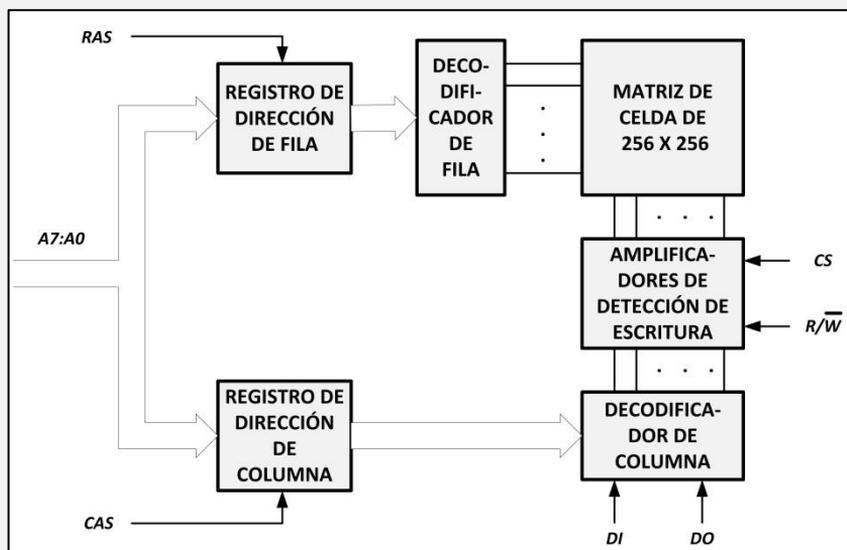


Fig. 6.11. Estructura interna de memorias RAM de escritura lectura dinámica (DRAM).

La aplicación de una dirección de fila provoca que todas las celdas de la fila se lean y restauren. Esto tiene lugar en cualquier operación de lectura o escritura. La operación de refresco se realiza por un Controlador de RAM dinámica, a veces incluido dentro de la memoria, y otras veces en el exterior.

Si bien estas memorias son más lentas que las estáticas, poseen una característica interesante. En muchos casos es necesario leer o escribir direcciones consecutivas de memoria. Si este es el caso, y además, pueden colocarse los datos en una misma fila, será necesario sólo cambiar la dirección de columna para realizar la operación. Esto se

traduce en velocidades aún mayores que las estáticas. Si bien lo anterior es posible, puede ser complejo aprovecharlo en la generalidad de los casos.

## 2.3 Memorias RAM de sólo lectura (ROM)

Estas memorias una vez programadas sólo realizan operaciones de lectura. No son volátiles, y pueden utilizarse para almacenar códigos, en generadores de caracteres, en funciones aritméticas complejas, en unidades de control microprogramadas, o en almacenamiento de partes del sistema operativo (BIOS), entre otras. La organización interna de estas memorias es similar a las RAM de lectura/escritura. La parte de entrada/salida es más sencilla por cuanto sólo es necesario considerar las salidas. De igual manera sucede con las líneas de control. A pesar que las ROM son memorias RAM, se suele utilizar este último término para hacer referencia a las memorias de lectura/escritura.

### 2.3.1 Memorias ROM

Para su construcción se utilizan diodos y transistores. Con las conexiones se indica un 1, y sin conexiones se indica un 0, como puede verse en la Figura 6.12. La presencia o no de un elemento acoplador (diodo) la realiza el fabricante, a quien hay que suministrarle la información requerida.

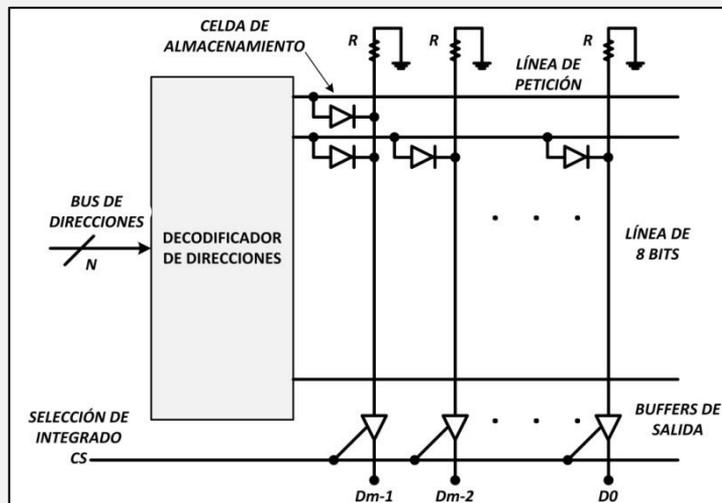


Fig. 6.12. Esquema de las conexiones en una memoria ROM.

Los diodos se utilizan como elementos acopladores. La conexión de varios diodos a una misma línea implementa la función OR de las señales de entrada. Puede decirse entonces que una ROM de  $2^n \times m$  bits, podría realizar cualquier combinacional de  $n$  variables de entrada y  $m$  funciones.

Las salidas del bus de datos de la Figura 6.12 son triestado para permitir la conexión de más de una memoria a un bus común.

### 2.3.2 Memorias PROM

Los elementos de conexión son diodos o transistores con un fusible en serie. Inicialmente la memoria presenta todas las conexiones establecidas. La programación consiste en destruir el fusible en aquellos lugares donde quiere almacenarse un 0. Esto se consigue direccionando la palabra deseada e inyectando una corriente adecuada en las salidas, así la conexión queda abierta y es como si no existiera el elemento acoplador. Se deduce que una vez programada la memoria ya no es posible volver a hacerlo.

Internamente estas memorias son similares a las ROM, como puede verse en la Figura 6.13. Para la grabación de estas memorias es necesario utilizar equipos de grabación especiales disponibles comercialmente.

### 2.3.3 Memorias R PROM

A diferencia de las anteriores, las memorias R PROM pueden ser reescritas por el usuario. Es necesario contar con equipos de grabación específicos para cada tipo de memoria. La grabación se realiza con la memoria fuera del circuito en el cual está conectada.

Se distinguen tres tipos de R PROM:

- EPROM: Las celdas están constituidas por puertas flotantes de transistores MOS. La descarga se realiza con luz ultravioleta exponiendo la celda a la misma por varios minutos. La reprogramación es eléctrica aplicando tensiones superiores a las de funcionamiento, y es permanente hasta que vuelva a grabarse.
- EEPROM: Similares a las anteriores con diferencia que el borrado es posición a posición, eléctricamente, y en algunas casos, puede realizarse con la memoria inserta en el circuito.
- FLASH: Similares a las anteriores sólo que el borrado se realiza simultáneamente a todas las posiciones.

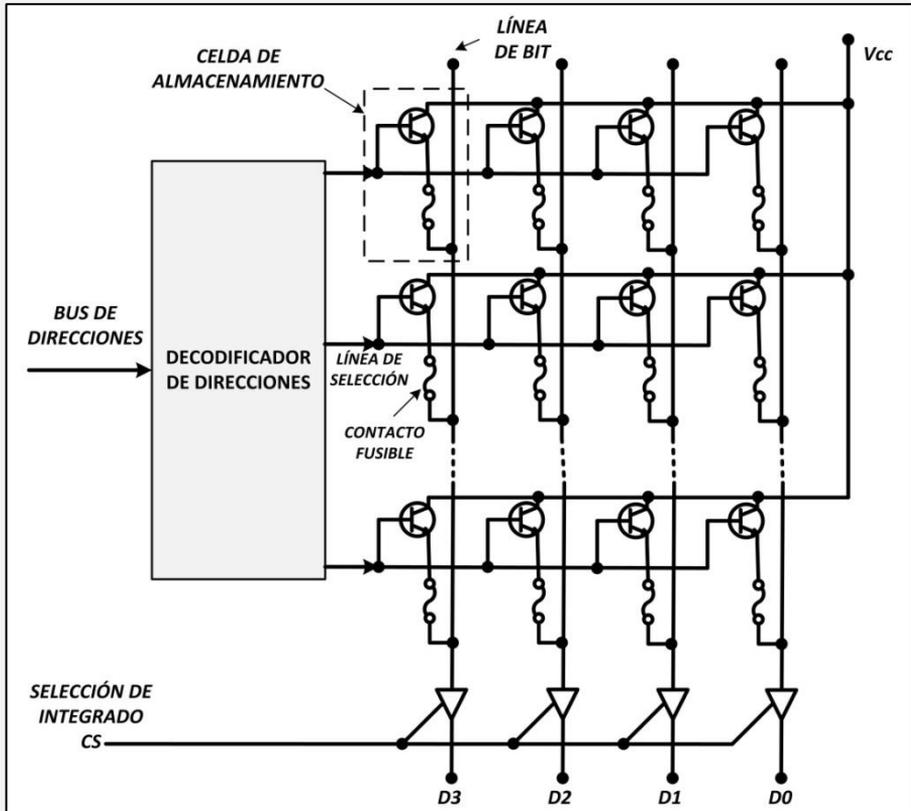


Fig. 6.13. Esquema de las conexiones en una memoria ROM.

## 2.4 Extensión de longitud de palabra y de capacidad

Es posible aumentar la capacidad de una memoria partiendo de circuitos integrados de menor capacidad. Esto puede lograrse aumentando la longitud de palabra o la cantidad de las mismas.

### 2.4.1 Extensión de longitud de palabra

En la Figura 6.14 se observa una memoria de N palabras de k.m bits, partiendo de un CI de N palabras de m bits. Las líneas de dirección y de control son compartidas por todos los CI. Las líneas de datos se amplían de m a k.m bits.

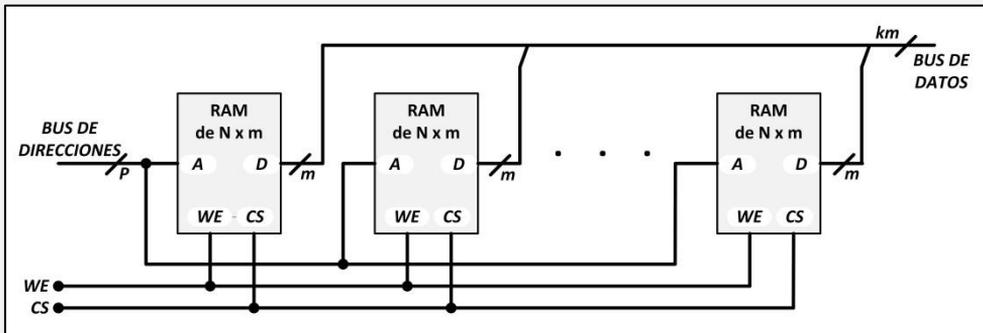


Fig. 6.14. Memoria de N palabras de k.m bits.

## 2.4.2 Extensión de número de palabras

En la Figura 6.15 se observa una memoria de  $2kN$  palabras de  $m$  bits, partiendo de un CI de  $N$  palabras de  $m$  bits. De las  $p + k$  líneas de dirección necesarias,  $p$  se interconectan a todos los CI a fin de seleccionar una de las  $N$  ( $2p$ ) palabras en cada CI. El resto de las  $k$  líneas de dirección se inyectan a un decodificador cuyas salidas se conectan a las líneas de selección (CS) de cada CI. La señal de  $R/\bar{W}$  es común para todos los CI.

El bus de datos es común para todos los CI. Esto es posible gracias a la tecnología triestados de los CI. Para ampliar la longitud de palabra y la cantidad de las mismas, se suman las técnicas indicadas.

Como ejemplo es interesante observar la Figura 6.16.

Finalmente, si disponemos de memorias RAM de  $N$  palabras de  $m$  bits y pretendemos una memoria de  $N'$  palabras de  $h \times m$  bits, se procede como sigue:

- Se calcula el número de Chips, dónde:
  - Número de CIs = Entero  $[N'/N] \times h$
- Se calcula el decodificador de  $k$  entradas, dónde:
  - $2^k \geq$  Número de CIs
- La parte baja de la dirección se conecta a las líneas de dirección de los CIs ( $p$  líneas, siendo  $2p=N$ ). La parte alta de la dirección ( $k$  bits), se conecta a las entradas del decodificador.
- Las salidas del decodificador se conectan a las entradas de selección (CS) de cada CI, y
- Las líneas de datos se conectan a un bus común de  $m + h$  bits.

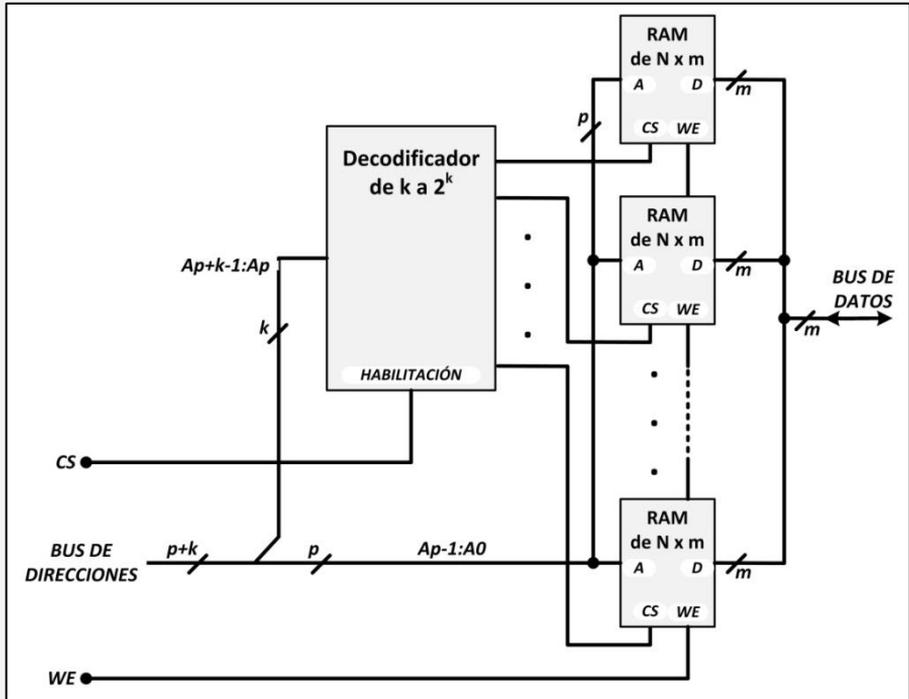


Fig.6.15. Memoria de  $2^k N$  palabras de  $k.m$  bits.

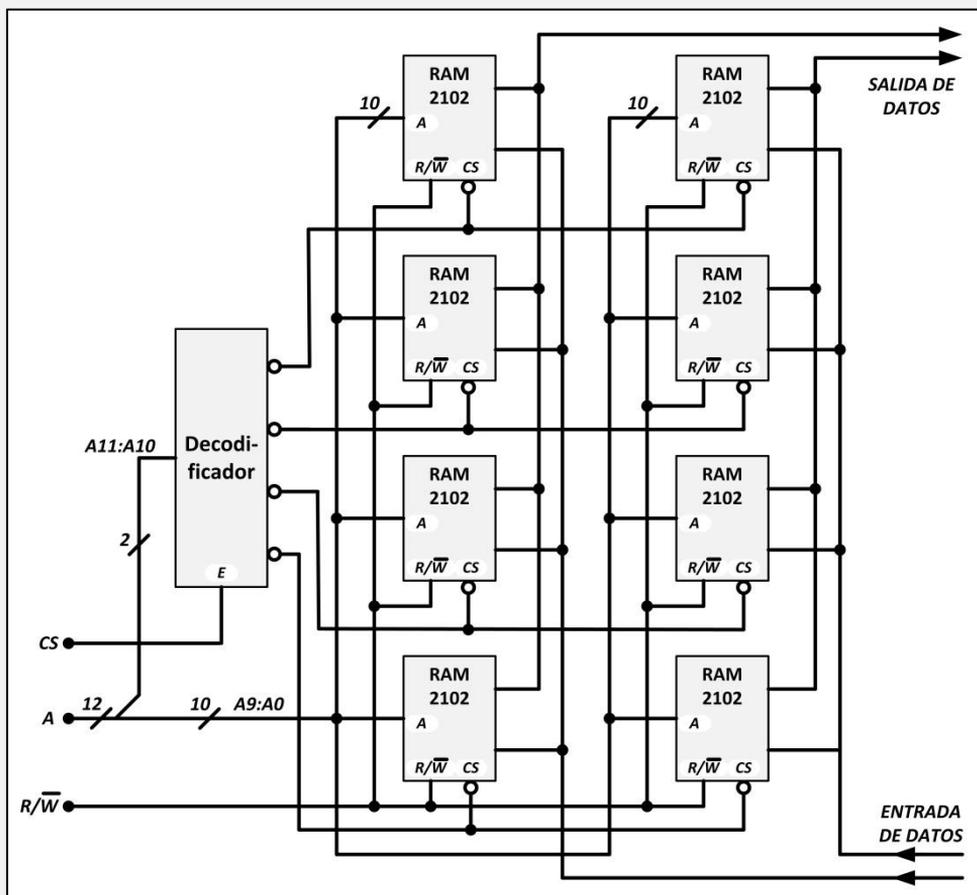


Fig.6.16. Memoria ejemplo.

## 3 Memorias de Acceso Serie

### 3.1 Definición

Son aquellas en las que el tiempo de escritura o lectura de una posición depende de la situación física de la misma en el interior de la memoria. Para escribir o leer en una de estas memorias es preciso pasar primero por todas las posiciones anteriores (Figura 6.17).



Fig. 6.17. Esquema de una memoria serie.

En las memorias serie, la información puede organizarse de dos formas:

- a) **Organización bit a bit:** Se disponen en serie tanto las palabras como los bits que las conforman (Figuras 6.18 y 6.19). La memoria posee una sola línea de entrada de información y una sola de salida. Además, dispone de líneas de control de lectura/escritura.



Fig. 6.18. Esquema de una memoria serie bit a bit.

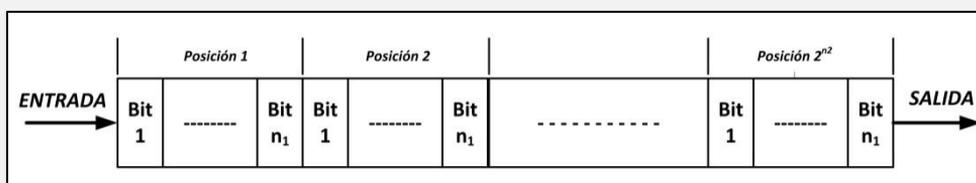


Fig. 6.19. Detalles de una memoria serie bit a bit.

- b) **Organización posición a posición:** Se disponen las palabras en serie pero los bits que las conforman en paralelo (Figuras 6.20 y 6.21). En este caso existen  $n$  entradas de información y  $n$  salidas de información. Las líneas de control también están presentes. Estas memorias pueden clasificarse en tres tipos: Registros de desplazamiento, memorias FIFO y memorias LIFO.



Fig. 6.20. Esquema de una memoria serie posición a posición.

Posición 1	Posición 2		Posición $2^{n_2}$
Bit 1	Bit 1	-----	Bit 1
Bit 2	Bit 2	-----	Bit 2
⋮	⋮	⋮	⋮
Bit $n_1$	Bit $n_1$	-----	Bit $n_1$

Fig. 6.21. Detalles de una memoria serie posición a posición.

### 3.2 Registros de desplazamiento

Son aquellos en los que la información se desplaza una posición en un sentido, con cada orden de lectura o escritura. La orden externa de desplazamiento está constituida por los impulsos de un generador (reloj o clock). Existen dos tipos de registros de desplazamiento: estáticos y dinámicos.

### 3.2.1 Registros de desplazamiento estáticos

Son aquellos en los que los impulsos de desplazamiento pueden anularse por tiempo indefinido sin que la información almacenada se pierda. Están constituidos por biestables síncronos activados por flancos (Figura 6.22).



Fig. 6.22. Esquema de un registro de desplazamiento estático.

### 3.2.2 Registros de desplazamiento dinámicos

Son aquellos en los que los impulsos de desplazamiento no pueden anularse porque la información se perdería. La celda elemental en este caso es la capacidad parásita de la puerta de un transistor MOS (similar a las DRAM).

La sencillez de estas celdas y su alta densidad de integración ha permitido construir CI de gran capacidad y bajo costo

A fin de restaurar la información en estos registros las salidas se conectan a las entradas, de esta forma la información circula permanentemente en todo el registro sincrónicamente a los impulsos del reloj. El circuito de la Figura 6.23 funciona correctamente siempre y cuando no se realice ninguna operación de lectura o escritura. Se puede concluir que es necesario agregar circuitos que permitan las operaciones de lectura y escritura. En ambos casos, se deberá adicionar un circuito exterior para permitir las operaciones mencionadas.

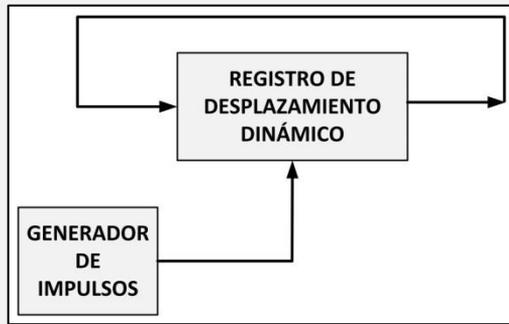


Fig. 6.23. Esquema de un registro de desplazamiento dinámico.

Para el caso de la lectura se propone el circuito de la Figura 6.24, y para la escritura el circuito de la Figura 6.25.

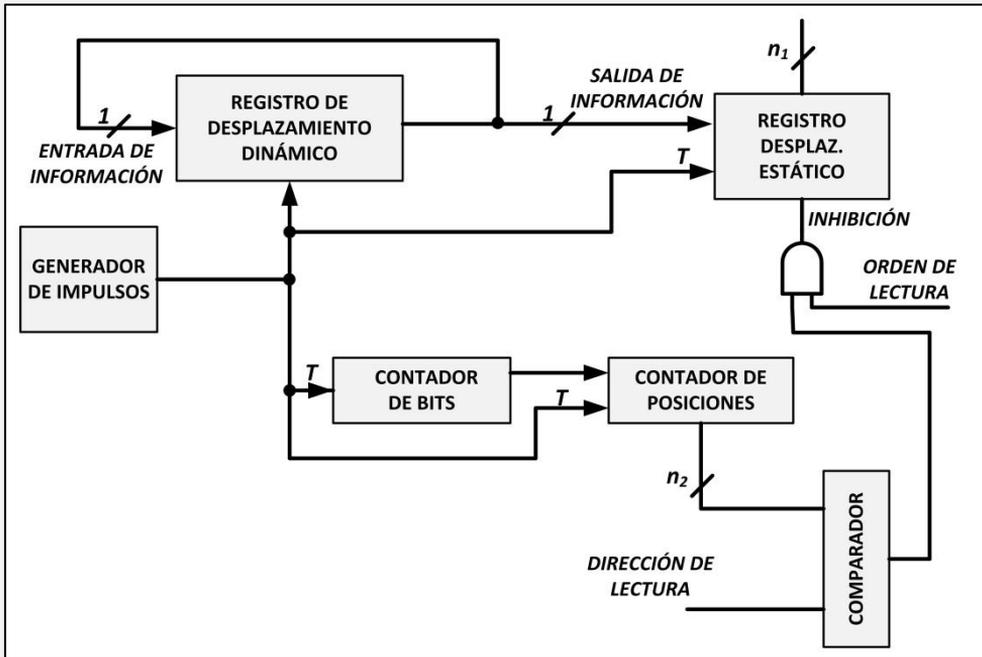


Fig. 6.24. Circuito para la operación de lectura.

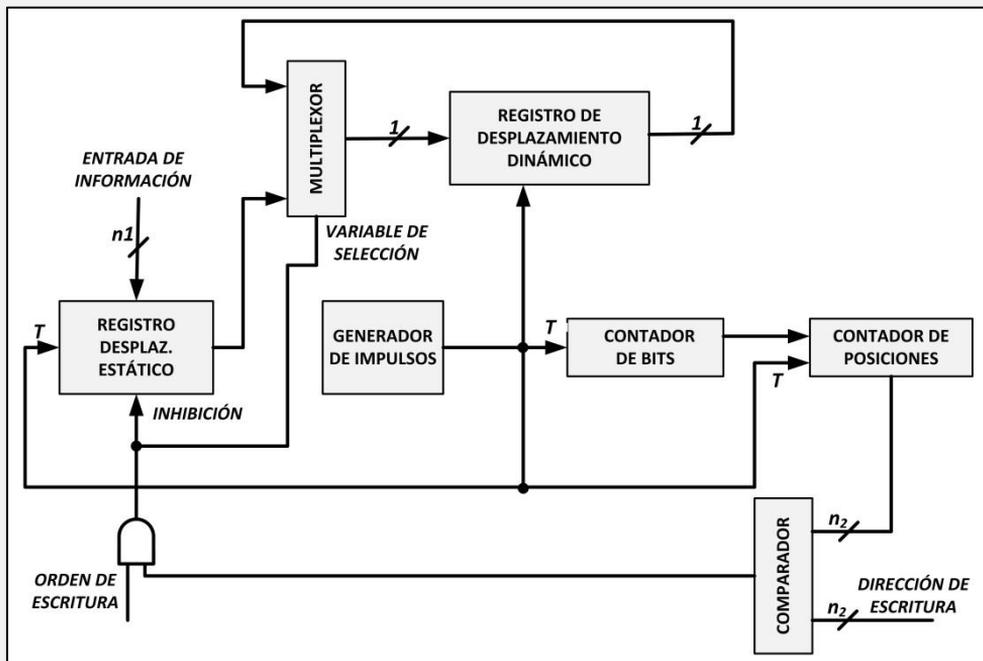


Fig. 6.25. Circuito para la operación de escritura.

Se invita al alumno a interpretar el funcionamiento de ambos circuitos y realizar los ciclos de lectura/escritura.

### 3.3 Memorias FIFO

Son memorias serie en las que la primera información que entra es la primera que sale (First Input First Output). La Figura 6.26 indica una FIFO en bloque y su funcionamiento se observa en la Figura 6.27.

Las memorias FIFO pueden implementarse con registros de desplazamiento estáticos y una unidad de control. Esta última debe tener en cuenta las siguientes características de este tipo de memoria.

- La lectura es destructiva. Es decir, que al leer, el dato leído ya no está más en la memoria.
- Cada operación de lectura o escritura debe producir un desplazamiento del resto de la memoria.
- Cuando la memoria está llena no podrá escribirse, por lo tanto, la Unidad de Control deberá ser capaz de generar una señal de Memoria llena.

- Generar las señales de control necesarias para que el primer dato escrito esté disponible para la primera lectura.
- Deberá aceptar al menos tres entradas exteriores: señal de lectura/escritura, señal de inicio de ciclo y señal de sincronismo.

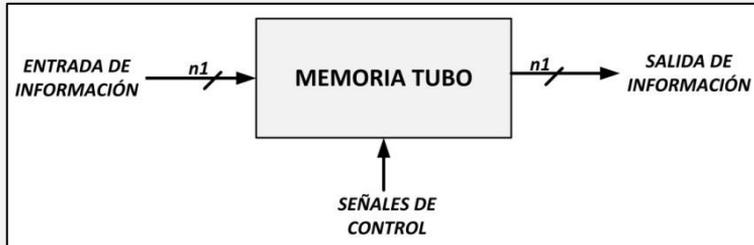


Fig. 6.26. Esquema de una memoria FIFO.

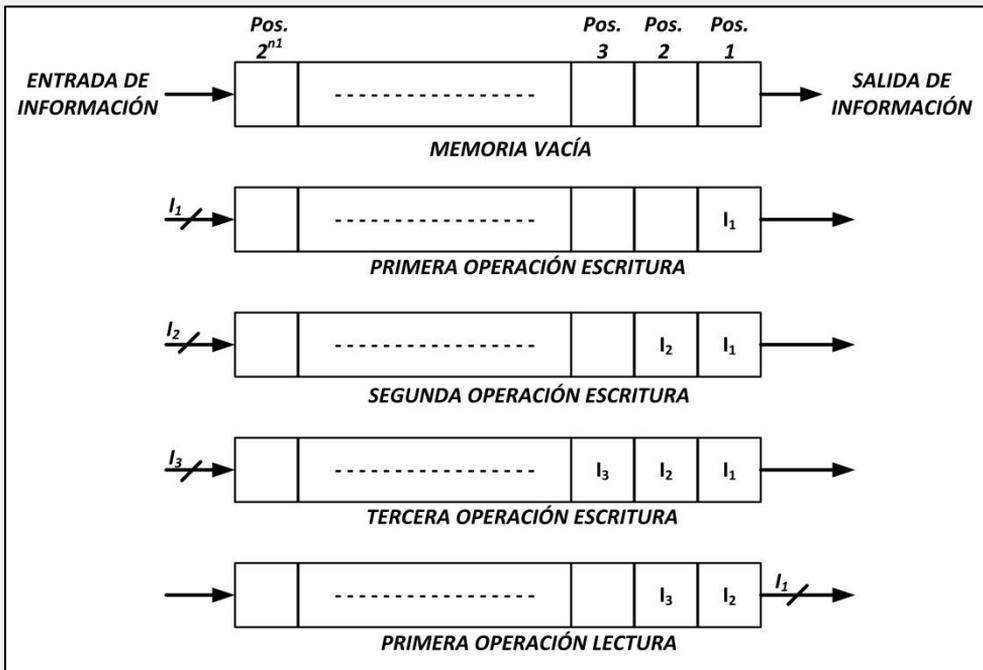


Fig. 6.27. Funcionamiento de una memoria FIFO.

En la Figura 6.28 se muestra un diagrama en bloques simplificado de una memoria FIFO implementada con un registro de desplazamiento estático.

Las FIFO se encuentran en CI de LSI y una de sus aplicaciones es acoplar sistemas digitales con velocidades de procesamiento diferentes (Figura 6.29). El sistema rápido va llenando la FIFO mientras que el lento

la va vaciando. La capacidad de la memoria debe estar acorde con la diferencia de velocidades y el tamaño del bloque a transferir.

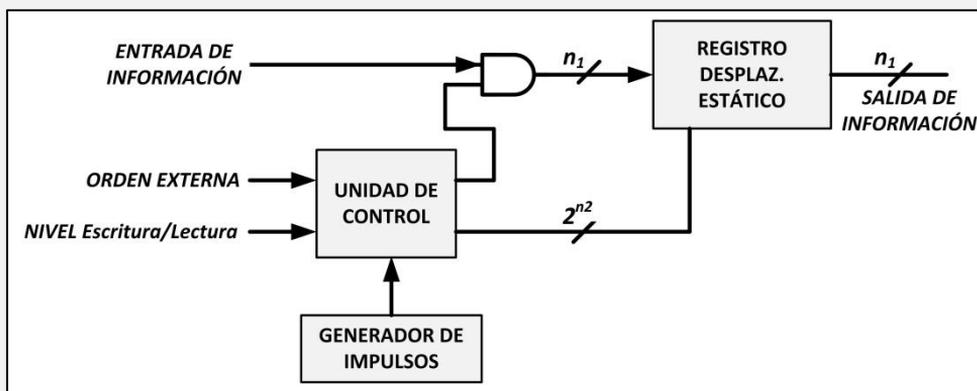


Fig. 6.28. Diagrama en bloques de una memoria FIFO implementada con un registro de desplazamiento estático.

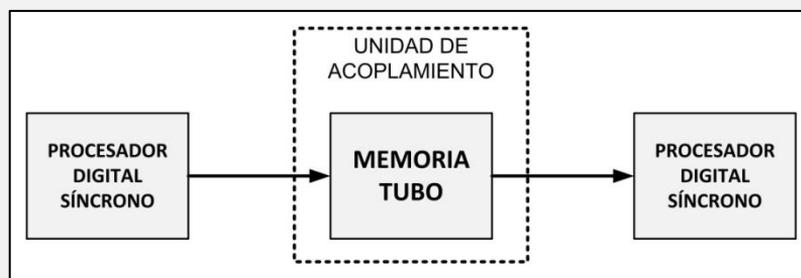


Fig. 6.29. Ejemplo de aplicación de una memoria FIFO en sistemas digitales.

### 3.4 Memorias LIFO

En estas memorias, la última información que entra es la primera que sale (Last Input First Output). En la Figura 6.30 se indica su diagrama en bloque y en la Figura 6.31 el funcionamiento de una memoria LIFO.

Una LIFO puede implementarse con un registro de desplazamiento reversible, según puede verse en la Figura 6.32.

Un registro de desplazamiento reversible está formado por biestables síncronos y multiplexores. La salida de la memoria es la salida del primer biestable y la entrada es el segundo canal del primer

multiplexor. Se sugiere al alumno interpretar la Figura 6.32 y realizar los ciclos de lectura/escritura.

El diagrama en bloque de una LIFO es el indicado en la Figura 6.33

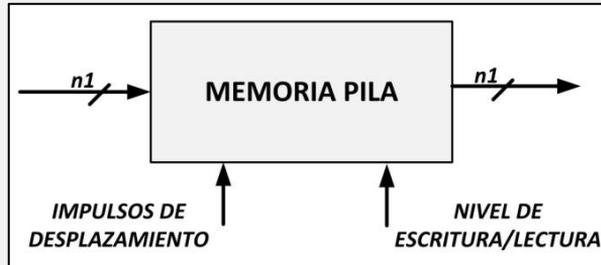


Fig. 6.30. Esquema de una memoria LIFO.

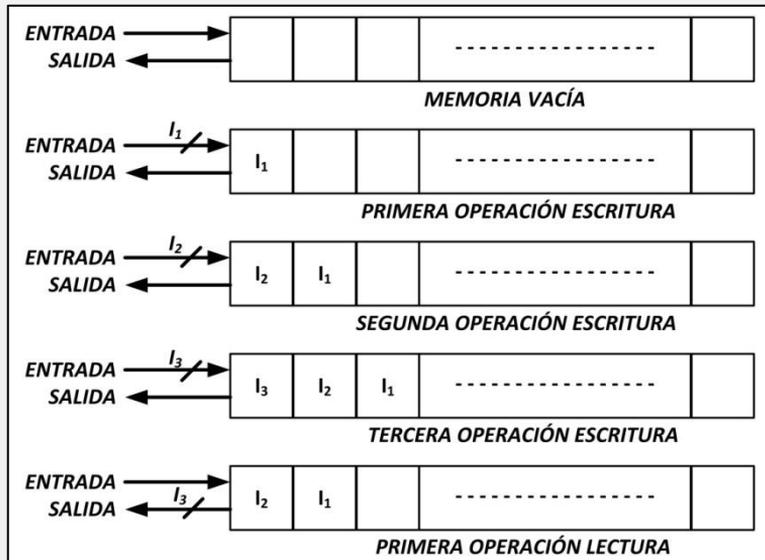


Fig. 6.31. Funcionamiento de una memoria LIFO.

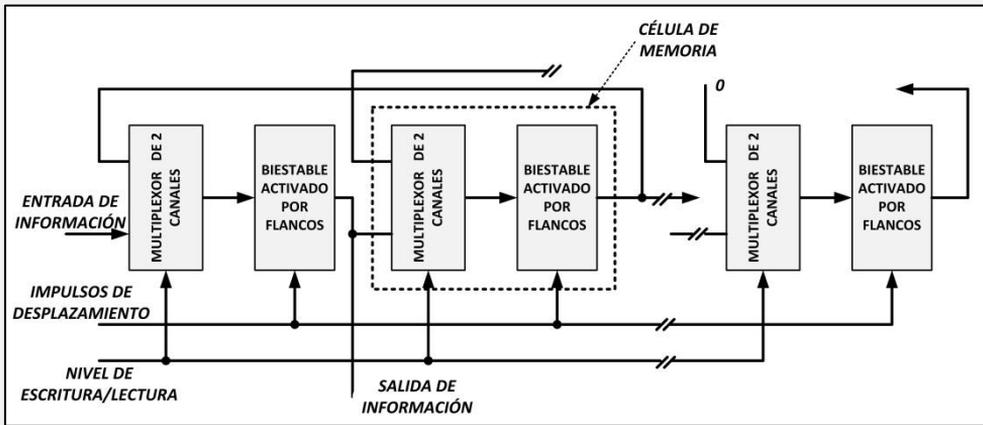


Fig. 6.32. Diagrama en bloques de una memoria LIFO implementada con un registro de desplazamiento reversible.

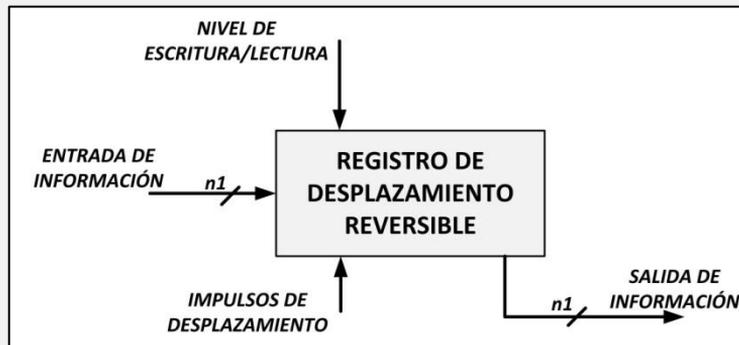
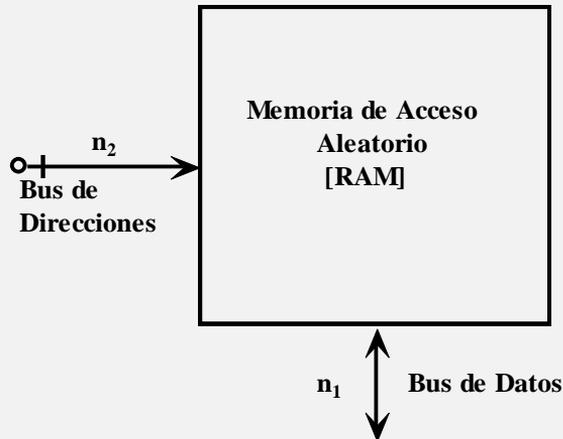


Fig. 6.33. Diagrama en bloque de una LIFO.

## 4 Ejercitación

### Ejercicio 1:

Determinar cuál es el valor de  $n_1$  y  $n_2$ , si la memoria es de: 4 Kbytes, 64 Kbytes, 1Mbytes, 4096 palabras de 32 bits, 65536 palabras de 16 bits o 524288 palabras de 64 bits.



**Ejercicio 2:**

Cuantos bits de direccionamiento son necesarios para la localización de una palabra en una memoria de 256 posiciones y de una de 1024.

**Ejercicio 3:**

Cuantos bits posee una memoria de 32Kx8, siendo 32K la cantidad de posiciones a direccionar y 8 la cantidad de bits de cada posición o palabra? ¿Cuantos bits de direccionamiento son necesarios?

**Ejercicio 4:**

Se tienen dos memorias RAM de 256 posiciones cada una. ¿Cuantos bits se necesitarán para direccionar a ambas de manera de obtener una memoria de 512 posiciones? Realizar una posible conexión.

**Ejercicio 5:**

Realizar una memoria de 2K X 8, partiendo de pastillas de 1K X 4.

**Ejercicio 6:**

Indique la cantidad de biestables de una memoria RAM de 4 Gbytes.

**Ejercicio 7:**

Suponga una memoria de 1 Mbytes. ¿De qué tamaño serán los decodificadores si se usan las técnicas 2D y 3D, respectivamente?.

**Ejercicio 8:**

Suponga que en un determinado sistema de control, se debe almacenar la temperatura de 2 sensores 1 vez cada hora durante las 24 hs (la temperatura viene dada por cuatro dígitos, p. ej. 100.6, o 028.7 grados centígrados) y se debe llevar un registro de las temperaturas de los últimos 2 meses, cuyos valores máximo y mínimo esperables son 150 y 10 °C? ¿Qué capacidad de memoria se necesitaría para llevar este registro? Considere que cada lectura se puede codificar con dígitos BCD, en forma binaria, o en ASCII, y que se puede seleccionar la forma que demande menos capacidad de memoria.

**Ejercicio 9:**

Teniendo en cuenta el tamaño de memoria definida en el ejercicio anterior, ¿en cuánto tiempo se llenará dicha memoria si los sensores registran valores 1 vez cada 3 minutos?

**Ejercicio 10:**

Un circuito combinacional acepta un número de entrada de tres bits codificado en binario natural y genera su cuadrado en sus salidas correspondientes. Escribir la tabla de verdad de este circuito. ¿Cómo se obtendría el mismo funcionamiento reemplazando el circuito combinacional por una memoria ROM? ¿De qué tamaño sería la memoria ROM? Mostrar el esquema de conexiones pertinente.

---

# CAPÍTULO 7

---

## Arquitectura Básica de una Computadora

- 1 Arquitectura de Von Neumann**
  - 2 Máquina Elemental**
    - 2.1 Introducción**
    - 2.2 Arquitectura de computadora elemental**
    - 2.3 El conjunto de instrucciones**
    - 2.4 El ciclo de máquina**
    - 2.5 Flujo de información**
    - 2.6 Unidad de control**
    - 2.7 Unidad de control microprogramada**
    - 2.8 Bus en la máquina elemental**
    - 2.9 Unidad aritmética y lógica**
  - 3 Ejercitación**
-

## Capítulo 7

# Arquitectura Básica de una Computadora

## 1 Arquitectura de Von Neumann

La estructura básica de una computadora está compuesta por cinco elementos (Figura 7.1):

- 1) **La Unidad de Proceso Central (CPU),**
- 2) **La Unidad de Memoria (UM),**
- 3) **La Unidad de Entrada/Salida (UE/S),**
- 4) **La Unidad de Buses (UB), y**
- 5) **El Programa Almacenado en UM (PA)**

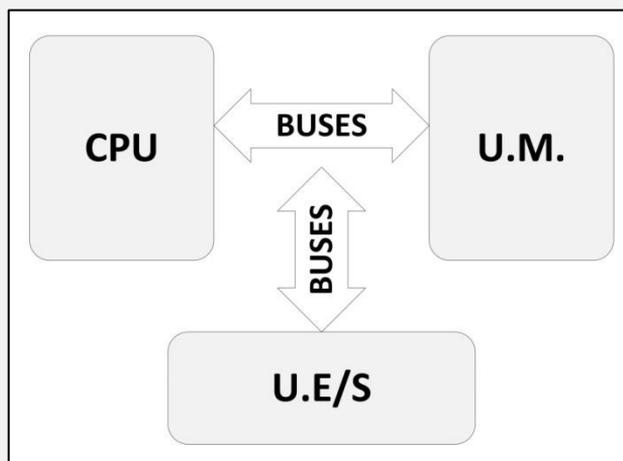


Fig. 7.1. Estructura básica de una computadora.

Estos cinco elementos y su funcionamiento reciben el nombre de Arquitectura de Computadoras de John von Neumann, en recuerdo como uno de los principales propulsores (1903-1957), y la utilizaremos como definición de COMPUTADORA.

Es común referenciar a los primeros cuatro elementos como **HARDWARE** y al quinto como **SOFTWARE**.

El funcionamiento de la Computadora implica una fuerte cooperación entre las unidades que la componen y supone que, en la memoria, reside un conjunto ordenado de **INSTRUCCIONES** llamado **PROGRAMA**.

### **Funcionamiento:**

Las **INSTRUCCIONES** del **PROGRAMA** son buscadas y ejecutadas secuencialmente (por la CPU) hasta que el programa finaliza. Durante la ejecución de una instrucción es posible que sea necesario obtener datos (o guardar resultados) desde o hacia la **UM**, o también obtener datos (o enviarlos) desde o hacia la **UE/S**, por lo tanto, será necesario coordinar el movimiento de datos a la **UM** y **UE/S**.

- 1) **CPU:** La **UNIDAD DE PROCESAMIENTO CENTRAL**, Unidad Central de Procesos o Procesador cuenta con:
  - Varios **REGISTROS**. Los **REGISTROS** son usados principalmente para almacenar temporalmente la información (datos, direcciones, instrucciones).
  - **Unidad Aritmética y Lógica (ALU)**. La **ALU** realiza las operaciones lógico-aritméticas necesarias, eventualmente, en la ejecución de una instrucción.
  - **Unidad de Control (UC)**. La **UC** es la parte de la **CPU** que realiza concretamente el control del sistema, para lo cual genera un conjunto de señales de control (órdenes) que se diseminan en todo el Sistema.
- 2) **UM:** La **UNIDAD DE MEMORIA** almacena programas y datos. Es, principalmente, una memoria **RAM** de lectura/escritura ya que es necesario leer y escribir en la misma. No obstante, una porción de la **UM** podría ser **ROM**, para almacenar programas o algún tipo de datos que no cambien y deberían permanecer aún que se quite la alimentación.
- 3) **UE/S:** La **UNIDAD DE ENTRADA/SALIDA** se encarga de interconectar la computadora con los dispositivos externos (periféricos) accesibles al usuario, como son el teclado, el monitor, la impresora, etc., por los cuales podemos ingresar o extraer información a o desde la **CPU**. Las **UE/S** cuentan con **INTERFACES** (que generalmente realizan la adaptación de niveles eléctricos de las señales) y **CONTROLADORES** que son sistemas digitales para controlar periféricos específicos.

- 4) **UB:** La UNIDAD DE BUSES cumple la función de transportar información entre las unidades del Sistema. Por el tipo de información, podrían clasificarse los buses en:
- **BUS de DATOS:** Transporta Operandos o Instrucciones
  - **BUS de DIRECCIONES:** Transporta Direcciones
  - **BUS de CONTROL:** Transporta señales de Control.
- 5) **PA:** El PROGRAMA ALMACENADO es un conjunto de INSTRUCCIONES almacenadas correlativamente en la Unidad de Memoria. Los programas tienen un comienzo y un fin. Las instrucciones son propias de cada máquina y se conocen como Set de Instrucciones.

## 2 Máquina Elemental

### 2.1 Introducción

A fin de estudiar la arquitectura de una computadora como la definida, presentaremos una Máquina Elemental utilizando el planteo de Caxton Foster, de su libro Arquitectura de Computadoras.

Previo al desarrollo es necesario distinguir entre las máquinas asíncronas y las síncronas. En las primeras, las órdenes emitidas por la UC se suceden secuencialmente, una después de la anterior. En cambio, en las síncronas las órdenes son emitidas por la UC en forma síncrona con una señal de sincronismo externa llamada reloj.

Las máquinas asíncronas son más rápidas que las síncronas, ya que las acciones ordenadas por la UC tardan lo que tarda el hardware involucrado, mientras que en las síncronas todas las acciones tardan un periodo de reloj. A pesar de la ventaja mencionada, las máquinas se construyen síncronas por la simplicidad en el hardware. Se puede inferir que la UC de una máquina asíncrona deberá contar con hardware adicional que detecte la finalización de las acciones ordenadas. Esto implica una complicación que no justifica la ganancia en tiempo que resultaría.

Las características principales de esta computadora son:

- Arquitectura de von Neumman,
- Usa sistema binario y aritmética en complemento a 2,

- Su memoria es de 4096 x 16 (4096 posiciones de memoria de 16 bits cada una), y
- Usa punto fijo y sus datos son de 16 bits (15 bits de mantisa y un bit de signo) como se observa en la Figura 7.2

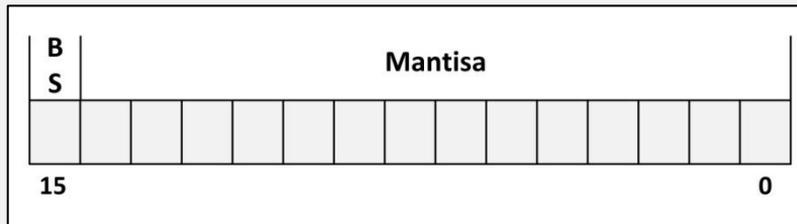


Fig. 7.2. Formato de los datos numéricos en punto fijo.

- Usa instrucciones de formato fijo de 16 bits (4 bits para el código de operación y 12 bits para el campo de dirección) tal como se presentan en la Figura 7.3.

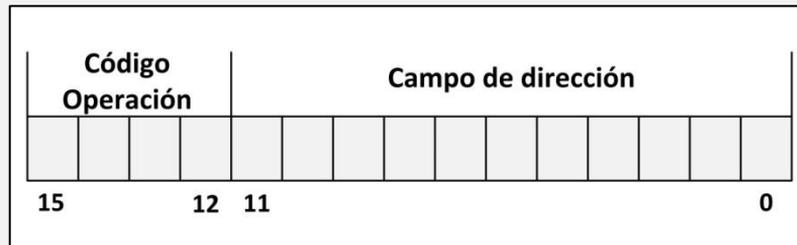


Fig. 7.3. Formato de las instrucciones.

- Usa un BUS común de 16 bits multiplexado (datos y direcciones).
- La ALU realiza, sobre 1 o 2 operandos, las siguientes operaciones:
  - ADD
  - OR
  - XOR
  - AND
  - RAL
  - NOT
- Capacidad para manejar hasta 128 periféricos.

Externamente tiene una consola con llaves, pulsadores y luces, que permite al operador comunicarse con la Máquina. El aspecto se muestra en la Figura 7.4.

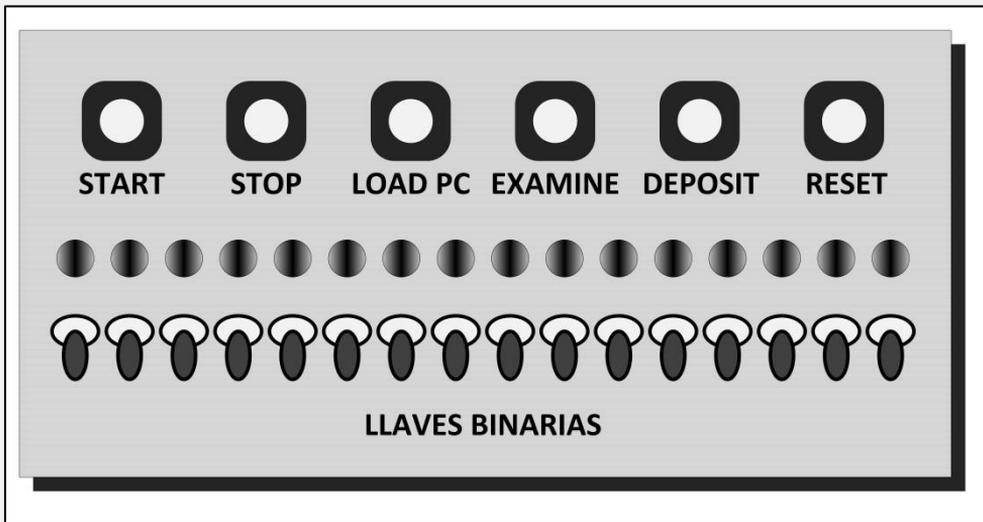


Fig. 7.4. Consola de la máquina elemental.

El Diagrama en Bloques de la Máquina Elemental se presenta en la Figura 7.5.

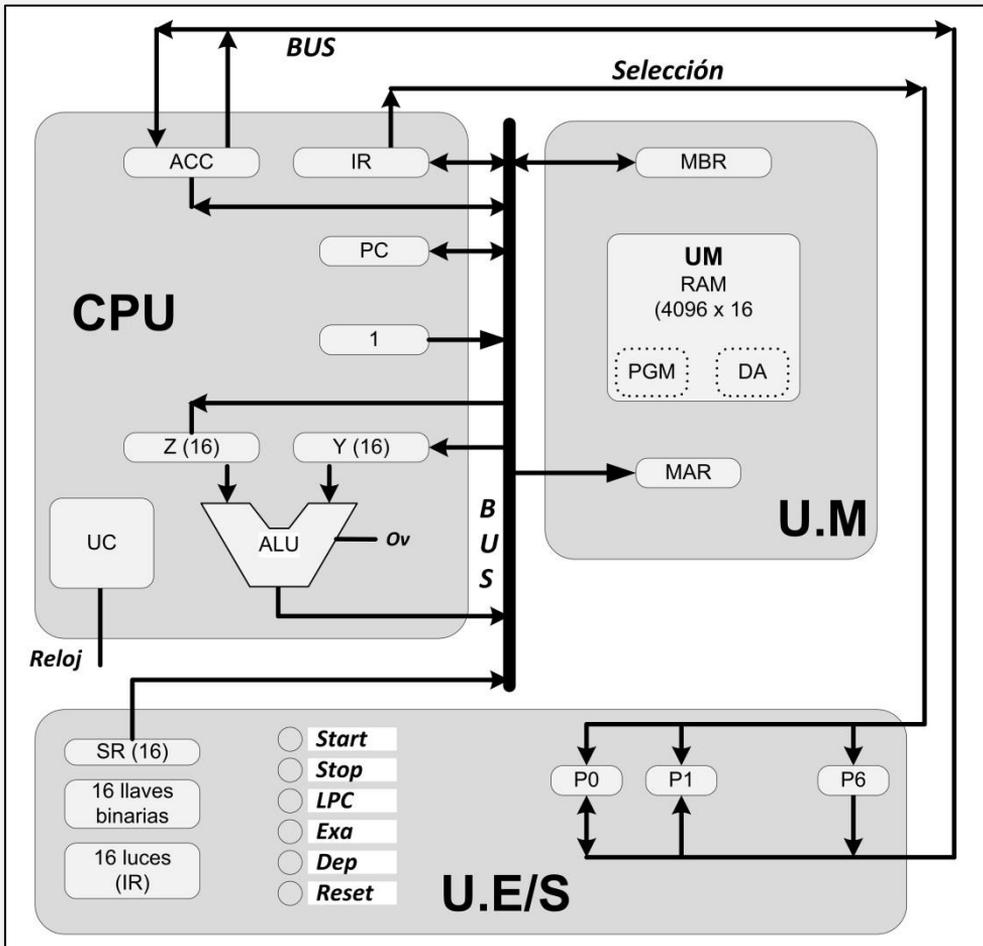


Fig. 7.5. Diagrama en bloques de la máquina elemental.

## 2.2 Arquitectura de la computadora elemental

### 2.2.1 Unidad de procesamiento central

La CPU cuenta con:

- **REGISTROS:**
  - **ACC** (16 bits): El registro Acumulador es un registro de propósitos generales, se utiliza para distintos fines.

- **IR** (16 bits): El registro de Instrucciones es un registro de propósito especial. Se utiliza para guardar la instrucción que se ha buscado desde la memoria.
  - **PC** (12 bits): El registro Contador de Programa es un registro de propósitos especiales. Se utiliza para guardar la dirección de memoria de la próxima instrucción a buscar.
  - **Registro Z** (16 bits) y **Registro Y** (16 bits): Los Registros de Operandos de entrada a la ALU son registros de propósito especial y no son accedidos por el programador.
  - **1** (1 bit): Es un registro de propósito especial. Permite generar un 1.
- **ALU:** realiza las operaciones lógico-aritméticas ya mencionadas sobre los operandos ubicados en los registros Z e Y. Genera la señal OV (overflow) si el resultado de la suma aritmética supera la capacidad de representación.
  - **UC:** La Unidad de Control tiene una entrada de sincronismo y genera las órdenes en sincronismo con los flancos del Reloj. Existen dos tipos de Unidades de Control: la UC Cableada y la UC Microprogramada.

### 2.2.2 Unidad de memoria

La UM es una memoria RAM de lectura/escritura como las vistas anteriormente. Posee un registro asociado a sus líneas de datos llamado MBR (16 bits), a través del cual pasará la información que se lea o escriba en la memoria. También posee otro registro asociado a las líneas de dirección llamado MAR (12 bits), cuyo contenido deberá ser la dirección de memoria a la que se pretende acceder. La RAM que se usa tiene un Tiempo de Acceso = 400 ns.

### 2.2.3 Unidad de Entrada/Salida

Como dispositivos de entrada/salida se observan dos casos:

- La **CONSOLA:** Está conectada directamente del BUS. En su interior posee el hardware que posibilita su comunicación con la CPU a través de los 6 pulsadores, las 16 llaves, las 16 luces y el Registro de Llaves (16 bits), que reflejan el estado de las 16 llaves binarias. Las 16 luces indican el contenido del IR.  
Los pulsadores definidos son:
  - Pulsador de **ARRANQUE** (Start)
  - Pulsador de **PARADA** (Stop)

- Pulsador de **CARGAR PC** (Load PC)  
Acción: Transfiere el contenido del SR al PC
  - Pulsador de **DEPOSITAR** (Deposit)  
Acción: Transfiere el contenido del SR a la posición de memoria indicada por el PC, luego incrementa el PC
  - Pulsador de **EXAMINAR** (Examine)  
Acción: Transfiere el contenido de la posición de memoria indicada por PC al IR, luego incrementa el PC
  - Pulsador de **RESET** (Reset)  
Acción: Borra la RAM y los Registros
- Los **DISPOSITIVOS EXTERNOS**: Son los llamados periféricos (P0 a P63 de entrada y P0 a P63 de salida)). Tienen destinado un BUS bidireccional dedicado a E/S (8 bits) para los Datos y un BUS (6 bits) para selección de periférico (el hardware de interfaz no se muestra en la Figura 7.5). Las transferencias de entrada - salida en la Blue son realizadas bajo control de programa (veremos instrucciones para tal fin) y a través del acumulador. Cuando se ejecuta una instrucción de entrada/salida, la Unidad de Control genera la señal Transferencia (TRA) hacia los dispositivos externos. Sólo el dispositivo seleccionado responderá con una señal Ready (R) cuando haya completado su tarea. Este tipo de control se llama HANDSHAKING.

## 2.3 Conjunto de instrucciones

Como anticipamos, esta computadora posee un tipo de instrucción de longitud fija de 16 bits con el siguiente formato (Figura 7.6):

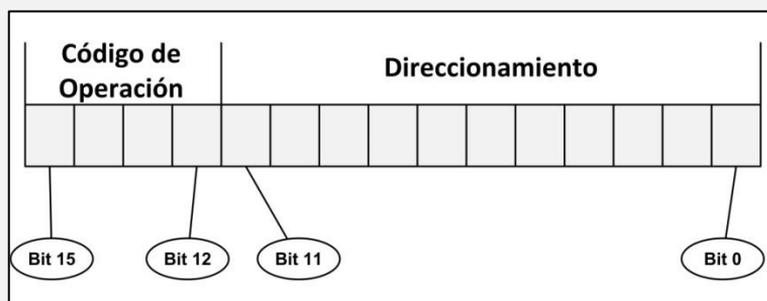


Fig. 7.6. Formato de las instrucciones.

Debido a que tenemos solamente 4 bits para el código de operación, se dispone a lo sumo de 16 instrucciones que se presentan en la Tabla 7.1.

<b>Código binario</b>	<b>Código octal</b>	<b>Nombre</b>	<b>Mnemónico</b>
0000XXXXXXXXXXXX	00XXXX	HALT (PARAR)	HLT XXXX
0001XXXXXXXXXXXX	01XXXX	ADD (SUMAR)	ADD XXXX
0010XXXXXXXXXXXX	02XXXX	OR-EXCLUSIVA	XOR XXXX
0011XXXXXXXXXXXX	03XXXX	AND	AND XXXX
0100XXXXXXXXXXXX	04XXXX	OR	IOR XXXX
0101XXXXXXXXXXXX	05XXXX	NOT (COMPLEM. A 1)	NOT XXXX
0110XXXXXXXXXXXX	06XXXX	LOAD AC (CARGAR AC)	LDA XXXX
0111XXXXXXXXXXXX	07XXXX	STORE AC (ALMAC. AC)	STA XXXX
1000XXXXXXXXXXXX	10XXXX	SALTO SUBROUTINA	SRJ
1001XXXXXXXXXXXX	11XXXX	SALTO CONDICIONAL	JMA XXXX
1010XXXXXXXXXXXX	12XXXX	SALTO INCONDICIONAL	JMP XXXX
1011XXXXXXXXXXXX	13XXXX	ENTRADA DE DATOS	INP XXXX
1100XXXXXXXXXXXX	14XXXX	SALIDA DE DATOS	OUT XXXX
1101XXXXXXXXXXXX	15XXXX	ROTACION DE BITS	RAL XXXX
1110XXXXXXXXXXXX	16XXXX	COPIAR LLAVES	CSA XXXX
1111XXXXXXXXXXXX	17XXXX	NO OPERACION	NOP XXXX

Tabla 7.1 Conjunto de instrucciones

La descripción de cada una de las instrucciones es la siguiente:

- HLT XXXX: Detiene el funcionamiento de la computadora. Presionando el pulsador START de la consola principal la computadora arranca nuevamente siguiendo con la ejecución de la instrucción siguiente al HALT. El campo de direcciones XXXX se ignora.

- **ADD XXXX:** Realiza la suma aritmética en complemento a dos de los operandos ubicados en el acumulador y en la dirección de memoria expresada en el campo XXXX, y deja el resultado en el acumulador. El contenido de la dirección de memoria XXXX no cambia, mientras que si lo hace el dato que se encontraba en el acumulador. Si el resultado de la suma es mayor que 215-1 o menor que -215 la computadora se detiene.
- **XOR XXXX:** Realiza la OR-Exclusiva bit a bit de los operandos ubicados en el acumulador y en la dirección de memoria expresada en el campo XXXX, y deja el resultado en el acumulador. El contenido de la dirección de memoria XXXX no cambia, mientras que si lo hace el dato que se encontraba en el acumulador.
- **AND XXXX:** Realiza la AND bit a bit de los operandos ubicados en el acumulador y en la dirección de memoria expresada en el campo XXXX, y deja el resultado en el acumulador. El contenido de la dirección de memoria XXXX no cambia, mientras que si lo hace el dato que se encontraba en el acumulador.
- **IOR XXXX:** Realiza la OR bit a bit de los operandos ubicados en el acumulador y en la dirección de memoria expresada en el campo XXXX, y deja el resultado en el acumulador. El contenido de la dirección de memoria XXXX no cambia, mientras que si lo hace el dato que se encontraba en el acumulador.
- **NOT XXXX:** Cada bit del dato en el acumulador es remplazado por su complemento lógico. La dirección de memoria XXXX se ignora.
- **LDA XXXX:** El contenido de la ubicación de memoria XXXX se copia en el acumulador. El contenido de la dirección de memoria XXXX no cambia, mientras que si lo hace el dato que se encontraba en el acumulador.
- **STA XXXX:** El contenido del acumulador se copia en la dirección de memoria XXXX. El contenido del acumulador no cambia, mientras que si lo hace el dato que se encontraba en la dirección de memoria.

- **SRJ XXXX:** Sirve para hacer un salto del programa a una subrutina. Para esto realiza una copia del contador del programa en los 12 bits más bajos del acumulador (en los 4 bits más altos del acumulador se ponen ceros). Luego, se copia el número XXXX en el contador del programa para que la próxima instrucción sea tomada de dicha dirección.
- **JMA XXXX:** Produce un salto a otra dirección de programa si el bit de signo del acumulador es uno (es decir si el acumulador contiene un número negativo). Si se cumple dicha condición, copia el número XXXX en el contador de programa y la próxima instrucción se toma de esta dirección. Si no se cumple la condición (el bit de signo del acumulador es cero, es decir, que el dato en el acumulador es positivo o cero), esta instrucción no realiza nada y el programa sigue normalmente.
- **JMP XXXX:** Produce un salto incondicional a otra parte del programa, dado que se copia el número XXXX en el contador de programa, y la próxima instrucción a ejecutar se toma de la dirección XXXX.
- **INP XXY:** Los 8 bits de mayor peso del acumulador se colocan a cero, y el próximo carácter de 8 bits que viene del dispositivo externo YY se coloca en la parte baja del acumulador. La parte XX del campo de dirección se ignora. La próxima instrucción no se ejecuta hasta que la transferencia del dato se haya completado.
- **OUT XXY:** Lo 8 bits más significativos del acumulador se envían al dispositivo externo YY. La parte XX del campo de dirección se ignora. Si el dispositivo externo no puede aceptar el dato en ese momento, la computadora espera hasta que se haya podido realizar la transferencia.
- **RAL XXXX:** Los bits del acumulador se rotan un lugar hacia la izquierda. El bit AC15 se coloca en ACo, de modo que el desplazamiento es cíclico. El campo de direcciones XXXX se ignora.
- **CSA XXXX:** El número que está en el registro de llaves (introducido por las llaves de la consola) se copia en el acumulador. El campo de direcciones XXXX se ignora.

- NOP XXXX: Esta instrucción no hace nada. El campo de direcciones XXXX se ignora.

## 2.4 El Ciclo de instrucción

Se llama “ciclo de instrucción” de una computadora, al procedimiento que consta de todas las tareas necesarias para poder buscar y ejecutar completamente una instrucción del programa almacenado en memoria, que podemos sintetizar de la siguiente forma:

- Búsqueda de una instrucción a memoria
- Lectura e interpretación de esa instrucción
- Ejecución de la misma.
- Almacenamiento de resultados
- Preparación para leer la próxima instrucción.

Esta computadora elemental (BLUE) tiene un ciclo de instrucción básico compuesto por dos partes:

- CICLO DE BÚSQUEDA
- CICLO DE EJECUCIÓN

Durante el Ciclo de Búsqueda, la instrucción almacenada en la memoria y apuntada por el Contador de Programa (PC) se localiza en la memoria y se copia en el Registro de Instrucciones (RI). Luego, el número almacenado en el PC se incrementa en uno, logrando así que ahora apunte a la próxima celda de memoria (o sea, a la siguiente instrucción).

Al completar el Ciclo de búsqueda, la instrucción que está en el IR se analiza, decodifica y ejecuta. Si la presente instrucción no necesita hacer una nueva búsqueda a memoria (de algún dato u operando) el ciclo de máquina termina en ese momento.

Si es necesario buscar un operando a memoria, entonces comienza el Ciclo de Ejecución, para realizar un nuevo acceso a memoria para traer al operando necesario y completar así la instrucción.

## 2.5 Flujo de información

Sin analizar las instrucciones en detalle todavía, mostraremos los movimientos de información entre registros dentro de la máquina:

- **El Flujo de direcciones (addresses)** en la Máquina Elemental son movimientos entre registros de 12 bits (Figura 7.7):

- Load PC: envía los 12 bits más bajos del registro de llaves (SR) al PC.
  - Saltos (JMP, JMA, SRJ): envían los 12 bits más bajos del IR al PC.
  - SRJ (salto a subrutina): envía los 12 bits del PC al Acumulador (ACC).
  - Búsqueda de una instrucción: envía los 12 bits del PC al MAR.
  - Búsqueda de un Operando: envía los 12 bits más bajos de RI al MAR.
- **El Flujo de instrucciones y operandos** en la Máquina Elemental son movimientos entre registros de 16 bits (Figura 7.8):
    - CSA: copia los 16 bits del SR al ACC.
    - Deposit: copia los 16 bits del SR al MBR.
    - Instrucciones: se copian del MBR al RI.
    - LDA: copia los 16 bits del MBR al ACC.
    - STA: copia los 16 bits del ACC al MBR.
    - Operaciones de la ALU (en el ciclo de ejecución):
      - Copia los 16 bits del ACC al Registro Z de la ALU.
      - Copia los 16 bits del MBR al Registro Y de la ALU.
      - El Resultado (o salida de la ALU) se copia al ACC.

Para poder realizar las posibles transferencias vistas es necesario implementar algún esquema de relación entre los registros, memoria y unidades de E/S, como por ejemplo:

- BUS COMUN (elegido para la Máquina Elemental)
- PUNTO A PUNTO

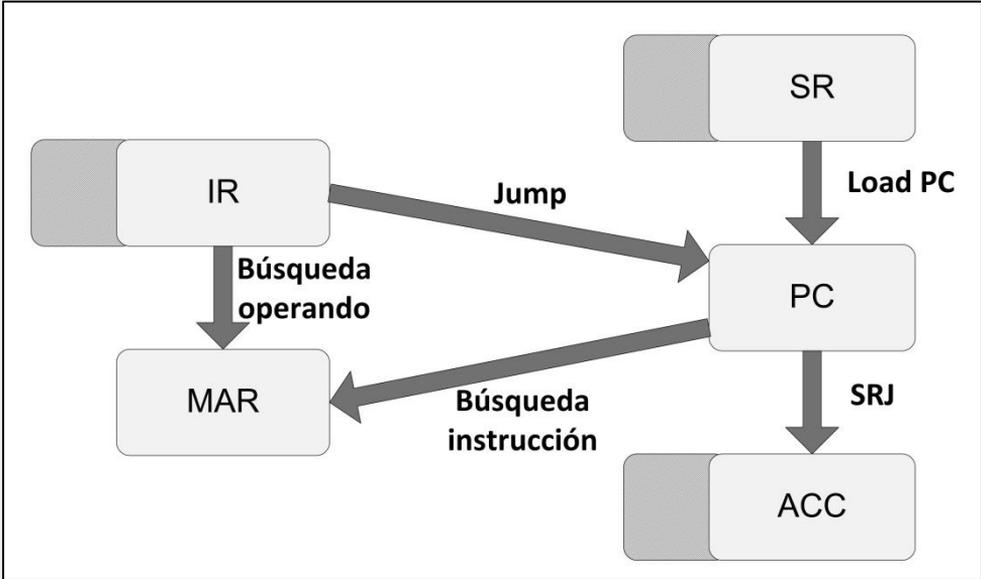


Fig. 7.7. La transmisión de direcciones en la Blue.

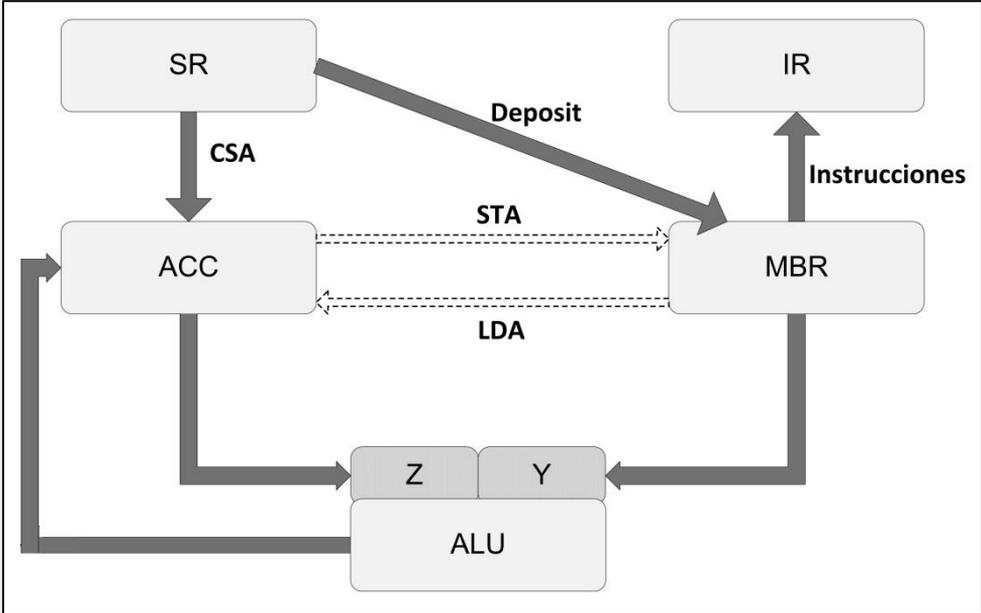


Fig. 7.8. La transmisión de instrucciones y operandos en la Blue.

## 2.6 Unidad de control

Las máquinas síncronas tienen Unidades de Control Síncronas, es decir, que tienen una entrada de sincronismo (reloj) que emite pulsos en periodos de tiempos fijos (frecuencia de reloj), y en cada pulso se realizan una o más tareas. Esto tiene como ventaja que mantiene las distintas tareas de cada dispositivo en orden y secuencia, y con un hardware sencillo. La desventaja es que ninguna tarea se puede realizar en menos tiempo que la duración de un ciclo de reloj.

La tarea de la unidad de control es coordinar todas las acciones de la máquina. Para este trabajo se necesita una secuencia de pulsos y señales que deben generarse sincrónicamente al reloj.

Existen dos maneras de diseñar la Unidad de Control:

- Unidad de Control Cableada, y
- Unidad de Control Microprogramada.

Ambas pueden verse como una caja negra con exactamente las mismas entradas y las mismas salidas. La diferencia está en su implementación interna.

### 2.6.1 Unidad de Control Cableada

En la Figura 7.9 se presenta el Diagrama en Bloques de la Unidad de Control Cableada. Está integrada por los siguientes componentes:

- a) **DECODIFICADOR DE INSTRUCCIONES:** Es un Sistema Combinacional. Un decodificador binario de 4 entradas y 16 salidas, que se utiliza para determinar qué instrucción contiene el IR. Por lo tanto, sus entradas serán IR12 a IR15.

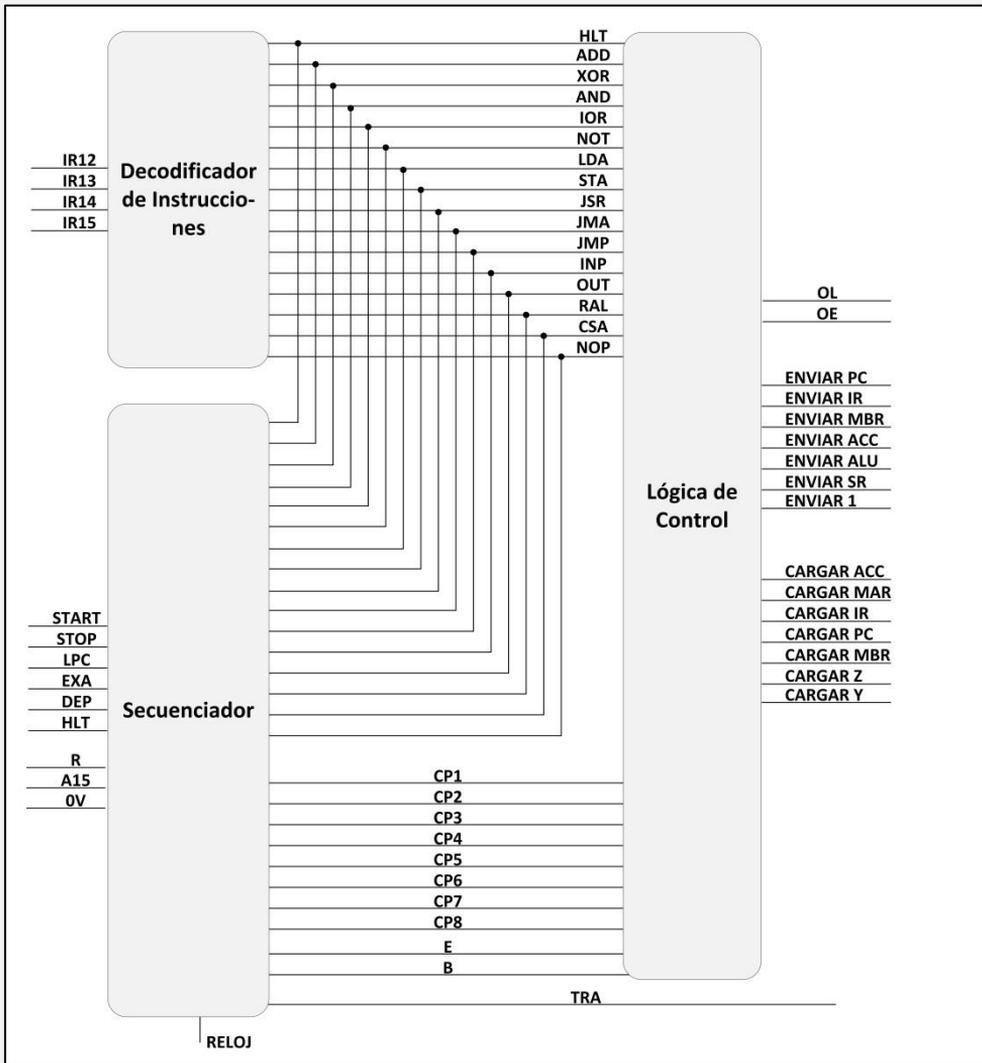
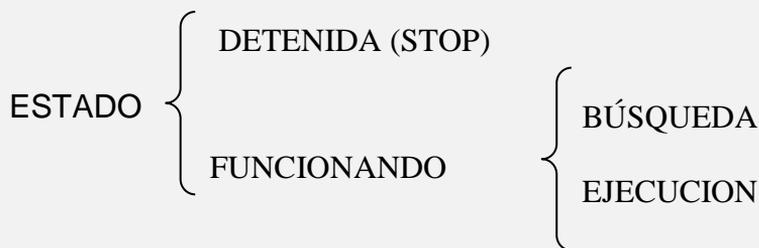


Fig. 7.9. Diagrama en bloques de la Unidad de Control Cableada.

- b) **SECUENCIADOR:** Es un Sistema Secuencial. El estado de la Máquina puede ser:



El SECUENCIADOR consta de (Figura 7.10):

- 2 (dos) biestables para definir los distintos estados mencionados:
    - **RUN** (arranque): es un biestable Set-Reset que arranca (pone en “1” su salida Q) por el botón de START o se para (pone en “0” su salida Q) por el botón de STOP, por la instrucción HALT o por la detección de un overflow aritmético.
    - **STATE** (estado): es un biestable D tiene dos condiciones (sus salidas) para determinar el ciclo de búsqueda (Fetch) o el ciclo de ejecución (Execute).
  - 1 (un) biestable relacionado con E/S:
    - **TRA** (transferencia): Es un biestable Set-Reset que se pone en “1” cuando se inicia una transferencia de E/S y se pone en “0” cuando se recibe la señal R (Ready) desde algún periférico. La E/S se discute más adelante.
  - 1 (un) biestable para tareas relacionadas con los Pulsadores EXA y DEP.
  - 1 (un) un reloj externo de 8 Mhz.
- c) **LÓGICA DE CONTROL:** Sistema Combinacional que consiste en un conjunto de compuertas que generan las señales de control.

## 2.6.2 Secuenciador

La señal de sincronismo es un oscilador de 8 Mhz llamado RELOJ que se divide en una secuencia de 8 pulsos, llamado CICLO DE MEMORIA, en líneas distintas separadas en tiempo por 125 ns.

Si RUN = 1 (máquina funcionando) el contador cuenta los pulsos de reloj, comenzando por 1 (ya que inicialmente estaba en cero). Las salidas del contador están conectadas a las entradas de un decodificador binario cuyas salidas son los pulsos de 125 ns.

Cuando la cuenta llega a 1001 (CP9), el contador se borra y comienza un nuevo CICLO DE MEMORIA. Además, este pulso CP9 se usa para sincronizar el biestable ESTADO y borrar el biestable SR sin nombre. Si la instrucción en proceso es de un ciclo, D toma el valor 0; en el caso de instrucciones de más de un ciclo, D toma el valor 1. De esta forma la máquina pasa de CICLO DE BÚSQUEDA a CICLO DE EJECUCIÓN automáticamente después del CP8.

Nótese que la máquina arranca con  $RUN = 1$ , al presionar el pulsador START y se detiene al presionar STOP, con un OV o con la instrucción HLT, siempre y cuando  $ESTADO = 0$  (BÚSQUEDA).

El biestable SR E/S genera la señal TRA hacia los periféricos en las condiciones que se ven en la Figura 7.10. Además, recibe la señal R (READY) desde los periféricos y se pone en cero.

Los pulsadores EXA y DEP funcionan sólo si  $RUN = 0$  y  $ESTADO = 0$ , y disparan un único CICLO DE MEMORIA. En la Figura 7.11 se muestra el diagrama de tiempo del secuenciador, y aparecen los CP1 a CP8. El CP9, que no se usa en el CICLO DE MEMORIA, es de menor duración.

### **El Ciclo de Búsqueda**

Si el biestable  $RUN = 1$ , arranca el reloj, el biestable de  $STATE = 0$  está en Búsqueda y se inicia el ciclo de búsqueda de la máquina, en el cual la máquina carga la instrucción cuya dirección está en el PC, en el registro de instrucciones. Se necesita que el operador haya cargado un PROGRAMA en Memoria y la dirección de la primera instrucción en el PC desde la Consola.

La secuencia del ciclo de búsqueda se presenta en la Tabla 7.2.

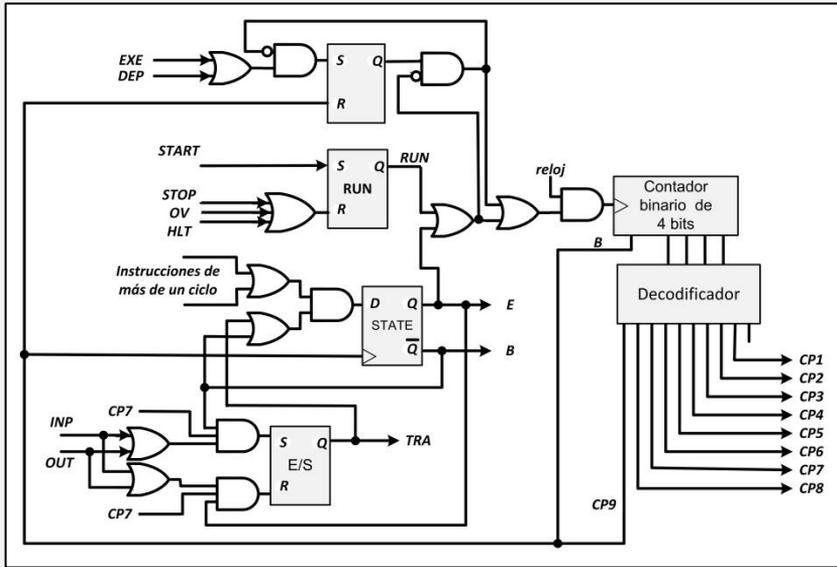


Fig. 7.10. Diagrama en bloques del secuenciador.

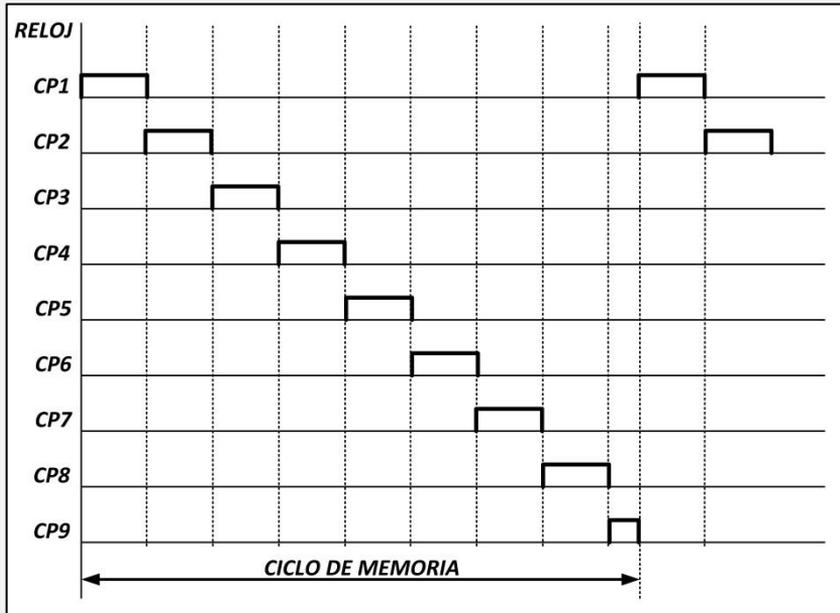


Fig. 7.11. Diagrama de tiempos del secuenciador.

En el pulso de reloj 1 se copia el contenido del contador de programa (PC) al registro de direcciones de memoria (MAR) y al registro Z de la ALU, y se inicia el ciclo de lectura de la memoria. En el pulso de reloj 2 se coloca el número +1 en el registro Y de la ALU. En el pulso de reloj 3 esperamos que se realice la suma  $PC + 1$ . En el pulso 4 se copia el resultado de la suma (la salida de la ALU) al contador de Programa (esto incrementa el PC y está listo para indicar la próxima instrucción). En el pulso 5 se copia el dato de la memoria al registro buffer de memoria (MBR). En el pulso 6 se copia el contenido del MBR al registro de instrucción (IR). Así culmina el Ciclo de Búsqueda.

Si la instrucción en cuestión no requiere, en su ejecución, acceder a la memoria por un dato, se pueden utilizar los pulsos de reloj 7 y 8 para ejecutar algunas instrucciones de la BLUE. Este es el caso de las instrucciones HALT, NOP, JMP, JMA, SRJ, CSA, NOT, RAL como indica la Tabla 7.2. Las mismas finalizan en el ciclo de memoria asignado a la Búsqueda (no necesitan otro ciclo de memoria) y las llamamos instrucciones de un ciclo. Es decir, la búsqueda y ejecución de estas instrucciones se realizan en un mismo ciclo de memoria. Al finalizar el ciclo de búsqueda, empieza un nuevo ciclo que debe asignarse nuevamente a Búsqueda.

CP	ACCIÓN	COMENTARIO
1	PC → MAR PC → Z	Transfiere el PC al MAR y a Z, y comienza a leer la instrucción
2	+1 → Y	Coloca +1 en Y
3		Tiempo de espera
4	ALU → PC	Hace la suma de $PC + 1$ y coloca el resultado en el PC
5	M → MBR	Coloca el dato leído de la memoria en el MBR
6	MBR → IR	Transfiere el contenido del MBR al IR y comienza la decodificación de la instrucción
7		Disponible para decodificación y ejecución
8		Se define si el próximo ciclo es de búsqueda o de ejecución

Tabla 7.2. Ciclo de búsqueda.

Sin embargo, si la instrucción requiere en su ejecución acceder a la memoria, o si se trata de las instrucciones INP o OUT, será necesario asignar el próximo ciclo a Ejecución.

### El Ciclo de Ejecución

Las instrucciones que requieren de un dato a memoria para realizar una operación lógica o matemática, o realizar una transferencia de datos hacia o desde la memoria, requieren un segundo ciclo de memoria. Estas son: LDA, STA, ADD, XOR, AND, IOR. Para acceder a memoria es necesario un segundo ciclo de memoria como se observa en la Tabla 7.3a y 7.3b.

Pero también existen instrucciones que requieren más de un ciclo y no acceden a memoria en su ejecución. Es el caso particular de las instrucciones de E/S: INP y OUT. Utilizan nuevos ciclos para esperar al periférico involucrado. Y serán tantos ciclos como el tiempo de espera (Tabla 7.4).

CP	HALT	NOP	JMP	JMA	SRJ	CSA	NOT	RAL
7	1 → TRA	-----	-----	-----	PC → A	-----	A → Z	A → Z
8	Off → RUN	-----	IR → PC	If A <sub>15</sub> = 1 IR → PC	IR → PC	SR → A	Z → A	2*Z → A

a)

CP	LDA	STA	ADD	XOR	AND	IOR
8	Al finalizar este pulso del ciclo de búsqueda, el biestable de ESTADO cambia a EJECUCIÓN					
1	IR → MAR; OL	IR → MAR	IR → MAR; OL	IR → MAR; OL	IR → MAR; OL	IR → MAR; OL
2	-----	A → MBR; OE	A → Z	A → Z	A → Z	A → Z
3	-----	-----	-----	-----	-----	-----
4	-----	-----	-----	-----	-----	-----
5	Mem → MBR	-----	Mem → MBR	Mem → MBR	Mem → MBR	Mem → MBR
6	MBR → A	-----	MBR → Y	MBR → Y	MBR → Y	MBR → Y
7	-----	-----	-----	-----	-----	-----
8	STATE → F	STATE → F	SUM → A STATE → F	XOR → A STATE → F	AND → A STATE → F	OR → A STATE → F

b)

Tabla 7.3. Ejecución de las instrucciones de uno y dos ciclos.

CP	INPUT	OUTPUT	COMENTARIO
7	1 → TRA	1 → TRA	Los bits IR <sub>5-0</sub> seleccionan el dispositivo de E/S
8	E → STATE	E → STATE	Fin ciclo de búsqueda y comienza el de ejecución
<hr style="border-top: 1px dashed black;"/>			
1	-----	-----	
2	-----	-----	
3	-----	-----	
4	-----	-----	
5	-----	-----	
6	-----	-----	
7	Si R = 1 Inp → A 0 → TRA	Si R = 1 0 → TRA	Si el Flag R=1 en INP se copian los datos del periférico a A, y en ambos se pone TRA=0
8	Si TRA = 0 F → STATE	Si TRA = 0 F → STATE	Si se completó la transferencia se inicia un nuevo ciclo de búsqueda
	Si TRA = 1 E → STATE	Si TRA = 1 E → STATE	Si NO se completó la transferencia se inicia un nuevo ciclo de ejecución

Tabla 7.4. Ciclo de búsqueda de las instrucciones INP y OUT.

En la Figura 7.12 se puede ver el hardware relacionado con la Entrada/Salida con cierto detalle.

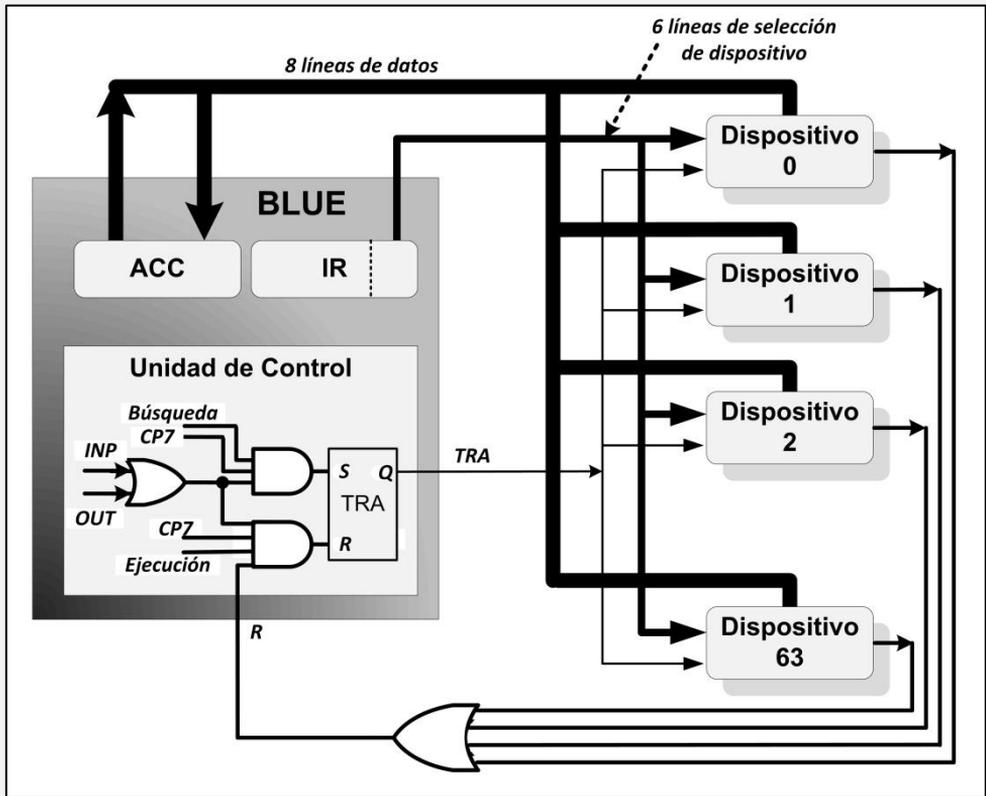


Fig. 7.12. Lógica para el funcionamiento de las E/S.

### 2.6.3 Lógica de control

Como se observa en la Figura 7.13, las entradas a este bloque combinacional son los códigos de operación decodificados, los pulsos de reloj y el estado de Búsqueda o Ejecución. Las salidas deben ser los órdenes concretos que emite la unidad de control. A fin de aclarar en qué consiste el hardware dentro de este bloque LÓGICA DE CONTROL veamos un ejemplo:

¿En qué casos la UC debe emitir la orden CARGAR MAR?

- 1) Si la máquina está parada y se presiona el pulsador EXA o DEP durante el CP1,
- 2) Si la máquina está funcionando, y está en Búsqueda y durante el CP1, o
- 3) Si la máquina está funcionando y está en Ejecución de las instrucciones LDA, STA, ADD, IOR, AND o XOR durante el CP1.

Entonces el circuito de control deberá tener en cuenta si la máquina está parada (con el biestable Estado = 0; esto significa que B = 1 y E = 0) y que se producirá una única secuencia de pulsos CP1 a CP8 sólo si el operador presionó EXA o DEP.

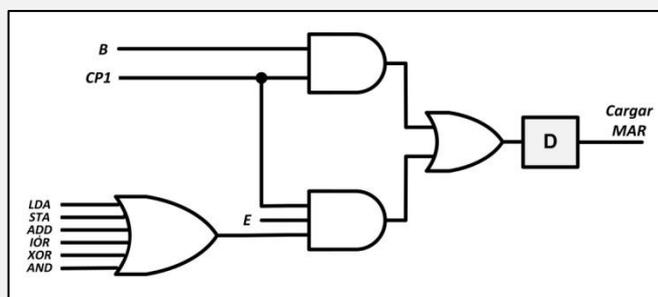


Fig. 7.13. Vista parcial de la lógica control.

El recuadro D indica un retardo de 50 ns a fin de permitir que se establezca el BUS como se verá posteriormente. Circuitos similares al planteado se corresponderán con todas las señales emitidas por la Unidad de Control.

## 2.7 Unidad de control microprogramada

El esquema general de la máquina elemental con unidad de control microprogramada o simplemente microprogramada, se observa en la Figura 7.14.

Obsérvese que lo único que ha cambiado en la máquina elemental es la Unidad de Control que, por su parecido a la arquitectura de von Neumann, se la ha llamado MÁQUINA INTERIOR.

La búsqueda y ejecución de cada macroinstrucción son realizadas por los microprogramas residentes en la micro-ROM (al contenido de la ROM, es decir, los microprogramas y la propia ROM se le llama FIRMWARE). La dirección de la primera microinstrucción a ejecutar, es proporcionada por el código de operación de la macroinstrucción, es decir, será alguna de las 16 primeras posiciones.

Cada microinstrucción está compuesta de 45 bits, divididos en seis campos: Acción, Test, Envíe, Reciba, Falso, y Éxito.

El significado de cada uno puede verse en las Tabla 7.5. El campo Acción está relacionado con las órdenes que debe dar la Unidad de

Control (leer la memoria, escribir la memoria, etc.), el campo Test se relaciona con la necesidad de chequear el estado de la máquina en un momento dado (Bit 15 del acumulador, señal de overflow, etc.). Los campos Envíe y Reciba tienen que ver con enviar los contenidos de los registros al bus o levantarlos del mismo. Por último los campos falso y éxito están relacionados con el resultado del chequeo indicado por el campo Test y definen la próxima microinstrucción a ejecutar. Nótese que la máquina interior no posee contador de programa. En la Tabla 7.5 puede verse el contenido de la ROM, es decir, los microprogramas correspondientes a cada instrucción.

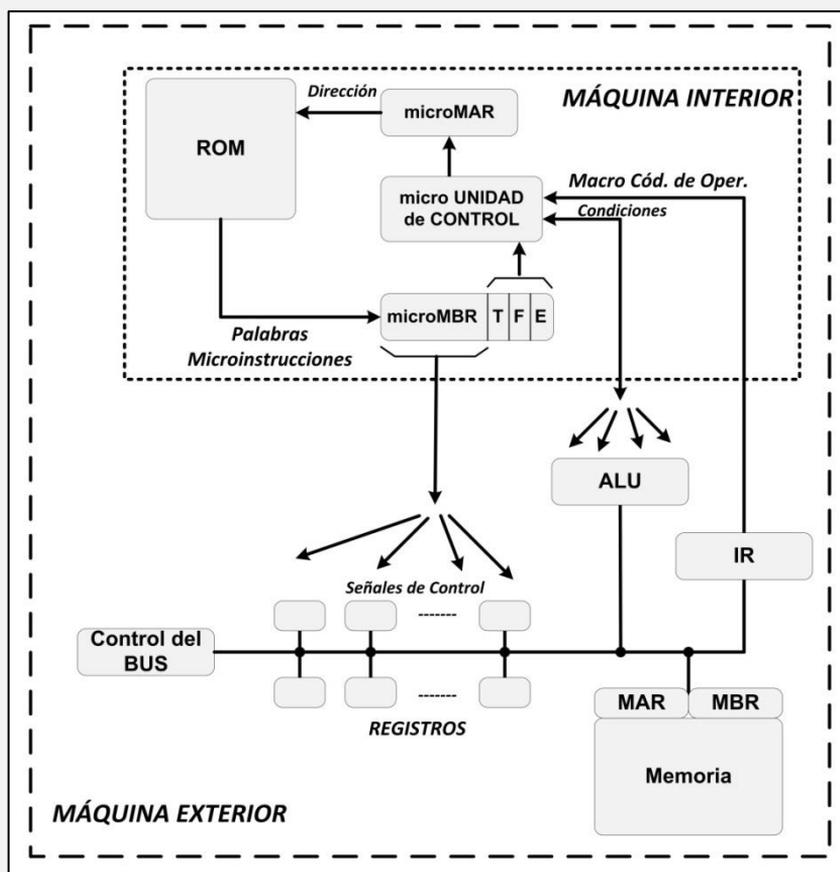


Fig. 7.14. Máquina elemental con unidad de control microprogramada.

## Microinstrucción de la Máquina Microprogramada

Acción (5)	Test (4)	Enviar (13)	Recibir (7)	Falso (8)	Éxito (8)
<b>Campo ACCIÓN: (5 BITS): Realiza una de las siguientes acciones:</b>					
1° bit	Inicie ciclo de lectura (LEER)				
2° bit	Inicie ciclo de escritura (ESCRIBIR)				
3° bit	TRA = 1				
4° bit	TRA = 0				
5° bit	BOOC				
<b>Campo ENVIAR: (13 BITS): Envía al BUS uno de los siguientes Registros:</b>					
1° bit	Acumulador (A)				
2° bit	MBR				
3° bit	PC				
4° bit	SR				
5° bit	+1				
6° bit	Campo de direccionamiento de IR				
7° bit	Líneas de entrada de datos (LED)				
8° bit	SUM				
9° bit	OR				
10° bit	XOR				
11° bit	AND				
12° bit	2 * Z				
13° bit	C <sub>i</sub> (z)				
<b>Campo RECIBIR: (7 BITS): Carga uno de los siguientes Registros desde el BUS:</b>					
1° bit	Acumulador (A)				
2° bit	MBR				
3° bit	PC				
4° bit	Z				
5° bit	IR				
6° bit	Y				
7° bit	MAR				
<b>Campo TEST: (4 bits): Si la condición especificada es acertada, toma la próxima microinstrucción desde la dirección especificada por el campo ÉXITO. Sino la toma desde el campo FALSO. Las condiciones son las siguientes:</b>					
0000	No realizar TEST				
0001	A <sub>15</sub> = 1 Acumulador negativo)				
0010	R = 1 (Dispositivo de E/S listo)				
0011	Memoria LIBRE (la memoria dispone de una línea que indica la finalización de una lectura o una escritura)				
0100	Botón STOP				
0101	Botón START				
0110	Botón Load PC				
0111	Botón EXAMINE				
1000	Botón Deposit				
1001	OVERFLOW				
1010 al 1111	NO DEFINIDOS				

Tabla 7.5. Formato de la microinstrucción

de la máquina elemental microprogramada.

El circuito de la UNIDAD de CONTROL MICROPROGRAMADA se observa en la Figura 7.15. Obsérvese que el contenido del micro-MAR (los 8 biestables D de la Figura) puede tener cuatro orígenes:

- Contenido del campo FALSO, en caso que la salida del multiplexor sea cero.
- Contenido del campo ÉXITO, en caso que la salida del multiplexor sea 1.
- El código de operación de la macroinstrucción residente en el registro de instrucciones, en el caso que el bit 5 del campo acción sea 1
- Cero, en el caso que RESET sea 1 (esta señal proviene del botón Master Reset en la consola del operador).

Los dos primeros casos se dan cuando la Unidad de Control está ejecutando un microprograma, ya sea correspondiente a una macroinstrucción o al ciclo de búsqueda.

En las Tablas 7.6a, b y c se muestra el contenido de la ROM (256 x 45). A modo de ejemplo, se describirá el ciclo de búsqueda en nuestra nueva máquina. El microprograma correspondiente comienza en la dirección rotulada RNI. El campo Acción no tiene contenido indicando que ninguna acción es necesaria, El campo Envíe envía el PC al bus, el campo Reciba carga el MAR y el Z desde el bus, el campo test verifica si alguien ha apretado el botón STOP de la consola, si está apretado la próxima microinstrucción es la que está en la dirección rotulada HALT, si no está apretado, la próxima microinstrucción es RNI1. De RNI1 a RNI5, se realiza el incremento del PC y la carga del IR con el contenido de la posición de memoria direccionada. La RNI6 da la orden de cargar el micro-MAR con el código de operación de la macroinstrucción ya almacenada en el IR, esto lo hace con el bit 5 en 1 (ver el circuito de control del micro-MAR).

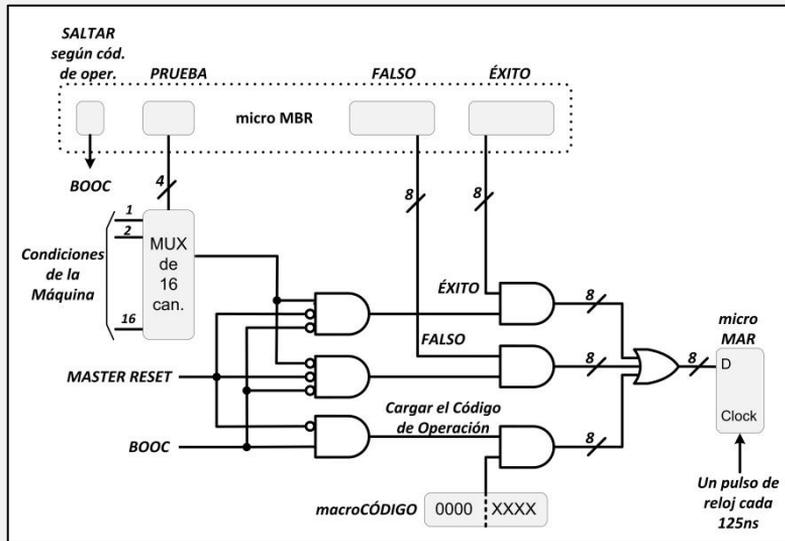


Fig. 7.15. Unidad de control microprogramada.

Dirección	Acción	Envíe	Recibe	Test	Falso	Éxito
0	----	----	----	----	HALT	----
1	----	----	----	----	ADD	----
2	----	----	----	----	XOR	----
3	----	----	----	----	AND	----
4	----	----	----	----	IOR	----
5	----	----	----	----	NOT	----
6	----	----	----	----	LDA	----
7	----	----	----	----	STA	----
10	----	----	----	----	SRJ	----
11	----	----	----	----	JMA	----
12	----	----	----	----	JMP	----
13	----	----	----	----	INP	----
14	----	----	----	----	OUT	----
15	----	----	----	----	RAL	----
16	----	----	----	----	CSA	----
17	----	----	----	----	NOP	----
<b>Microprograma de Búsqueda</b>						
RNI	LEER	PC	MAR, Z	STOP	RNI 1	HALT
RNI 1	----	+1	Y	----	RNI 2	----
RNI 2	----	----	----	----	RNI 3	----
RNI 3	----	SUM	PC	LIBRE	RNI 4	RNI 5
RNI 4	----	----	----	LIBRE	RNI 4	RNI 5
RNI 5	----	MBR	IR	----	RNI 6	----
RNI 6	BOOC	----	----	----	----	----
<b>Microprograma de la instrucción HALT</b>						
HALT	TRA = 0	----	----	START	H 1	RNI
H 1	----	----	----	EXAM	H 2	EXAM
H 2	----	----	----	DEPOSIT	H 3	DEPOSIT
H 3	----	----	----	Load PC	HALT	LPC
<b>Microprograma del Botón EXAMINAR</b>						
EXA	LEER	PC	MAR, Z	----	E 1	----
E 1	----	+1	Y	----	E 2	----
E 2	----	----	----	----	E 3	----
E 3	----	SUM	PC	LIBRE	E 4	E 5
E 4	----	----	----	LIBRE	E 4	E 5
E 5	----	MBR	IR	----	E 6	----
E 6	----	----	----	EXAM	H 6	E 6

**a)**

Dirección	Acción	Envíe	Recibe	Test	Falso	Éxito
<b>Microprograma del Botón DEPOSITAR</b>						
DEP	----	PC	MAR, Z	----	D 1	----
D 2	----	+ 1	Y	----	D 2	----
D 2	----	SUM	PC	----	D 3	----
D 3	ESCRIBIR	SR	MBR	----	D 4	----
D 4	----	----	----	LIBRE	D 4	D 5
D 5	----	----	----	DEPOSIT	H 3	D 5
<b>Microprograma del Botón CARGUE PC</b>						
LPC	----	SR	PC	----	L 1	----
L 1	----	----	----	LOAD PC	HALT	L 1
<b>Microprograma de la instrucción NOP</b>						
NOP	----	----	----	----	RNI	----
<b>Microprograma de la instrucción CSA</b>						
CSA	----	SR	A	----	RNI	----
<b>Microprograma de la instrucción RAL</b>						
RAL	----	A	Z	----	RAL 1	----
RAL 1	----	2*Z	A	----	RAL 2	----
RAL 2	----	----	----	----	RNI	----
<b>Microprograma de la instrucción JMP</b>						
JMP	----	IR	PC	----	RNI	----
<b>Microprograma de la instrucción JMA</b>						
JMA	----	----	----	A 15	RNI	JMP
<b>Microprograma de la instrucción SRJ</b>						
SRJ	----	----	PC	A	JMP	----
<b>Microprograma de la instrucción NOT</b>						
NOT	----	A	Z	----	NOT 1	----
NOT 1	----	C,(Z)	A	----	NOT 2	----
NOT 2	----	----	----	----	RNI	----
<b>Microprograma de la instrucción INP</b>						
INP	TRA = 1	----	----	R = 1	INP	INP 1
INP 1	----	LED	A	----	INP 2	----
INP 2	TRA = 0	----	----	----	RNI	----
<b>Microprograma de la instrucción OUT</b>						
OUT	TRA = 1	----	----	R = 1	OUT	OUT 1
OUT 1	TRA = 0	----	----	----	RNI	----

**b)**

Dirección	Acción	Envíe	Recibe	Test	Falso	Éxito
<b>Microprograma de la instrucción LDA</b>						
LDA	LEER	IR	MAR	----	LDA 1	----
LDA 1	----	----	----	LIBRE	LDA 1	LDA 2
LDA 2	----	MBR	A	----	RNI	----
<b>Microprograma de la instrucción STA</b>						
STA	----	IR	MAR	----	STA 1	----
STA 1	ESCRIBIR	A	MBR	----	STA 2	----
STA 2	----	----	----	LIBRE	STA 2	RNI
<b>Microprograma de la instrucción ADD</b>						
ADD	LEER	IR	MAR	----	ADD 1	----
ADD 1	----	A	Z	LIBRE	ADD 2	ADD 3
ADD 2	----	----	----	LIBRE	ADD 2	ADD 3
ADD 3	----	MBR	Y	----	ADD 4	----
ADD 4	----	----	----	----	ADD 5	----
ADD 5	----	SUM	A	----	RNI	----
<b>Microprograma de la instrucción IOR</b>						
IOR	LEER	IR	MAR	----	IOR 1	----
IOR 1	----	A	Z	LIBRE	IOR 2	IOR 3
IOR 2	----	----	----	LIBRE	IOR 2	IOR 3
IOR 3	----	MBR	Y	----	IOR 4	----
IOR 4	----	----	----	----	IOR 5	----
IOR 5	----	OR	A	----	RNI	----

**c)**

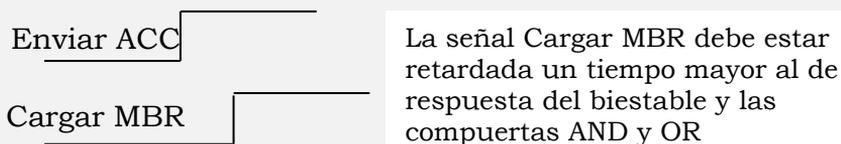
Tabla 7.6. Contenido de la ROM (256 x 45).

Es conveniente realizar el análisis de todos los microprogramas a fin de comprender el funcionamiento completo. Al final de la Guía Didáctica se discuten las ventajas y desventajas de las máquinas microprogramadas que, debido a la facilidad que da el firmware, poseen gran cantidad de instrucciones. En contraposición, las máquinas no microprogramadas poseen, entre otras características, pocas instrucciones.

## 2.8 Bus en la máquina elemental

Si analizamos los órdenes que emite la UC (Cableada o Microprogramada), podemos concluir que la gran mayoría implica una transferencia entre registros. Con exactitud son 20 señales entre enviar al BUS y cargar desde el BUS. Sólo 4 órdenes no implican transferencias: LEER, ESCRIBIR, TRA = 0 y TRA = 1. Gran parte de la actividad de la máquina se resuelve transfiriendo registros.

Por lo mencionado, los registros deben estar eficazmente interconectados y esto puede lograrse con la arquitectura de BUS COMÚN. Este término se refiere a que existirá un único BUS que interconecta a las distintas partes de la Máquina. Este BUS COMÚN puede transportar un dato (16 bits), una instrucción (16 bits) o una dirección (12 bits) en distintos momentos, razón por la cual se lo llama bus multiplexado. Las señales de control son diseminadas por la Máquina por un BUS especial a ese fin, que llamamos BUS de CONTROL. En la Figura 7.16 se observa un circuito que resuelve los requerimientos y las señales de control.



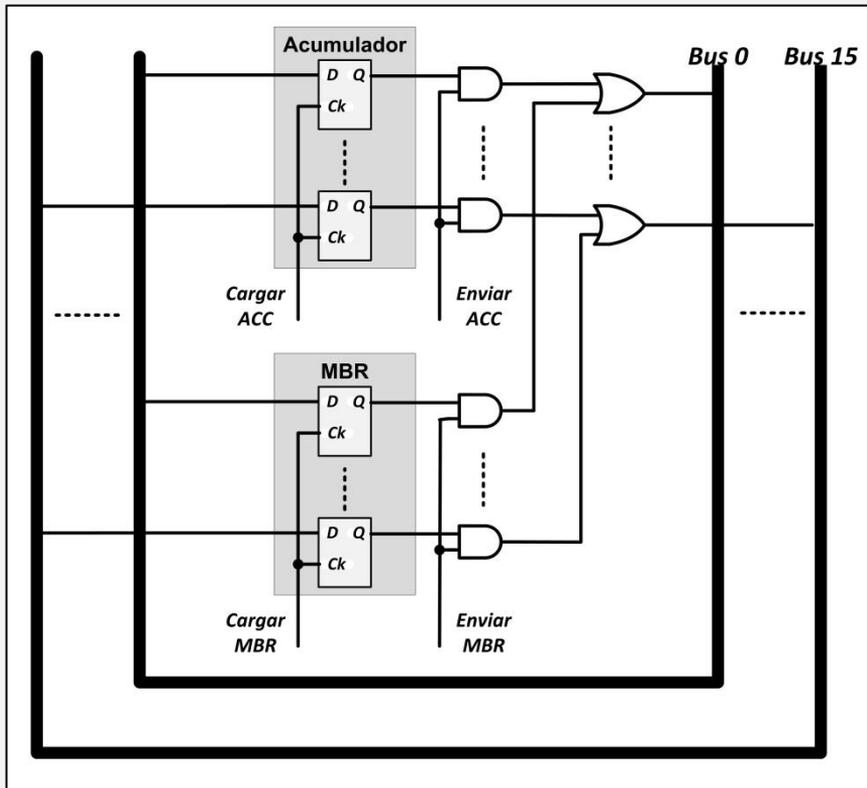


Fig. 7.16. Circuito que resuelve los requerimientos y las señales de control.

Obsérvese que sólo se debe enviar un registro al BUS. Sin embargo, es posible cargar más de un registro desde el BUS simultáneamente, un ejemplo de esto es la orden:

ENVÍE PC, CARGUE MAR y Z

La conexión del Registro MBR al BUS requiere un circuito adicional por cuanto este Registro también está conectado la memoria. La Figura 7.17 indica esta conexión.

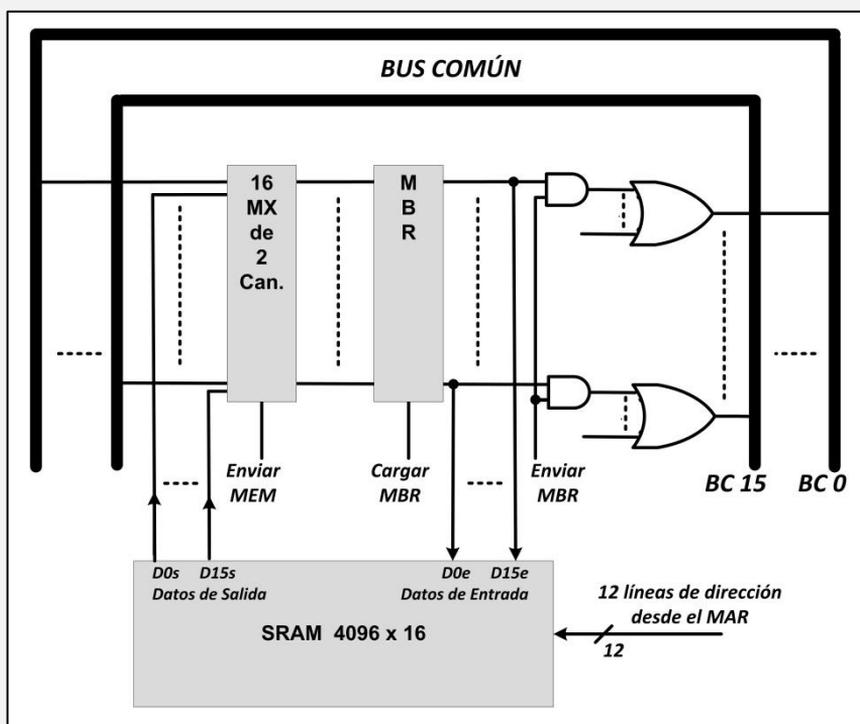


Fig. 7.17. Circuito asociado al Registro MBR.

## 2.9 Unidad aritmética y lógica

La ALU de la Máquina Elemental realiza las operaciones AND, OR, IOR, XOR y ADD sobre los operandos contenidos en los registros Z e Y. Las operaciones lógicas se realizan bit a bit y la suma aritmética se realiza con convenio de Complemento a 2 (el acarreo se desprecia). Adicionalmente, la ALU realiza la operación NOT (C1) y RAL (rotación de bits hacia la izquierda) sobre el contenido de Z.

Las operaciones mencionadas las realiza simultánea y permanentemente, es decir, que en todo momento está calculando los resultados sobre el contenido de los registros Z e Y. La unidad de control sólo tiene que cargar los registros Z e Y con los operandos y seleccionar cual es la salida que necesita. El diagrama en bloques de la ALU se observa en la Figura 7.18.

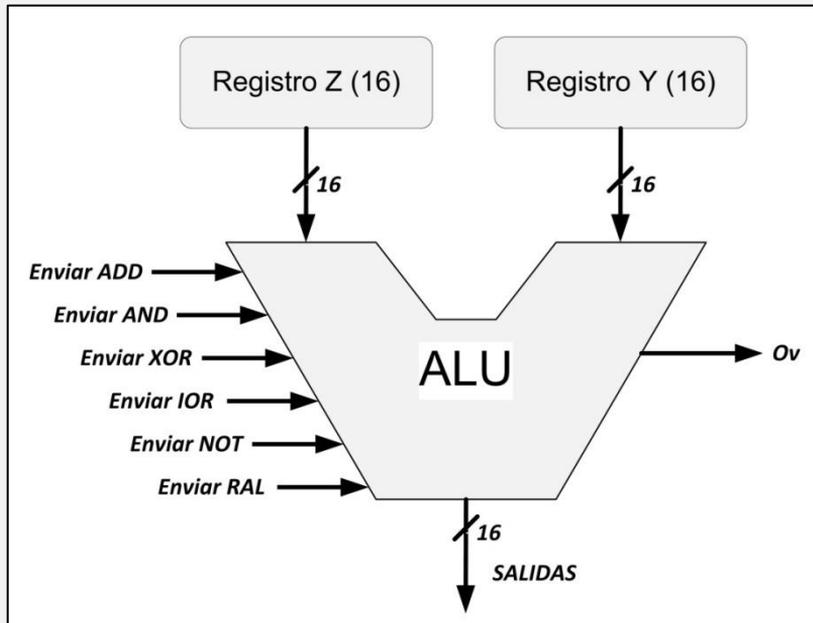


Fig. 7.18. Diagrama en bloques de la unidad aritmética y lógica.

Se recomienda al lector realizar el circuito de la ALU para ejercitar lo aprendido. La ALU tendrá:

- 16 Compuertas AND,
- 16 Compuertas XOR,
- 16 Compuertas IOR,
- 16 Compuertas NOT,
- 1 Sumador de 16 bits con circuito de rebasamiento (OV) sin acarreo (C2), y
- Compuertas AND y OR para implementar la selección de la salida.

Es conveniente mencionar que el tiempo que tarda la ALU para realizar la suma aritmética (ADD), es mayor que el necesario para las operaciones lógicas, ya que estas últimas requieren menos hardware. Para los ciclos de máquina se ha supuesto que la ALU tarda para sumar (ADD), como máximo, 200 ns y para el resto de las operaciones 80 ns.

Por lo tanto, cuando la ALU suma, la UC debería esperarla un pulso de reloj (para el caso de UC Cableada) o una microinstrucción (para el caso de UC Microprogramada). Observe las Figuras 7.10 y 7.15.

## 3 Ejercitación

### Ejercicio 1:

¿Cuál es el efecto de las siguientes instrucciones en octal?. Indique el mnemónico correspondiente.

177777	137701
140277	013150
042322	125000
075303	104210
063501	

### Ejercicio 2:

Complete el siguiente cuadro con las restantes formas de representación.

Mnemónico	octal	binario
ADD 50	--	--
--	150510	--
--	--	1111000001001000
NOT 33	--	--
--	--	0011000001010110
--	110350	--
XOR 276	--	--

### Ejercicio 3:

Escriba un programa comenzando en la ubicación 400 que intercambie los contenidos de las ubicaciones 550 y 551. Expresarlo en representación octal y mnemónica.

### Ejercicio 4:

Escriba la representación octal del siguiente programa. ¿Cuál es el contenido del acumulador y de las ubicaciones 3007, 3010, 3011 3012 después de la ejecución del programa?.

3001	LDA	3007
3002	IOR	3010
3003	AND	3011
3004	STA	3012
3005	OUT	01
3006	HLT	
3007	110771	
3010	145735	

3011	074000
3012	77177

**Ejercicio 5:**

Escriba un programa que comience en la ubicación 550 que efectúe la operación lógica AND entre el contenido de la ubicación 560 y 563. Depositar el resultado en 560 e indicar el valor resultante en octal a partir de los siguientes datos en decimal:

$$(560) = +1307$$
$$(563) = +2431$$
**Ejercicio 6:**

Proyecte una secuencia de instrucciones que efectúe un salto a la ubicación 255 si el contenido de la dirección 300 es cero, y a la ubicación 333 si el contenido es distinto de cero.

**Ejercicio 7:**

Suponga que la máquina elemental debe ser dotada del doble de instrucciones. ¿Cómo modificaría la estructura de la palabra, manteniendo su longitud con dicho objeto?. ¿Cuántas palabras pueden direccionarse directamente según su propuesta?.

**Ejercicio 8:**

Escriba un programa que efectúe una multiplicación entre dos números usando el método de las sumas sucesivas.

**Ejercicio 9:**

Escriba en instrucciones de la blue el siguiente código, parte de un programa de lenguaje de alto nivel tipo C++, considerando que todas las variables son enteras, y proponiendo posiciones de memoria para el almacenamiento de los datos y el programa.

```
for (i = 0; i <= 10; i = i + 1)
    a[i] = b[i] + c;
```

**Ejercicio 10:**

Con instrucciones de la Máquina Elemental escriba el siguiente código:

```
i=010
while i<0 do
    a[i]= b[i]
    i=i+1
break a[i]= 170
done
```

**Ejercicio 11:**

Escriba en instrucciones de la blue la siguiente expresión algebraica, parte de un programa de lenguaje de alto nivel tipo C++, considerando que todas las variables son enteras, y proponiendo posiciones de memoria para el almacenamiento de los datos y el programa. Realizarlo como subrutina; indicar expresamente el paso y retomo de la subrutina al programa principal.

$$f = (g + h) - (i + j)$$

**Ejercicio 12:**

Realice el circuito de la Unidad de Control Cableada usando un contador y un decodificador para generar los 8 pulsos (CP1 a CP8) del Ciclo de Máquina de la Máquina Elemental.

---

# CAPÍTULO 8

---

## Arquitectura Convencional

- 1 Visión General**
  - 1.1 Formato de instrucciones**
  - 1.2 Modos de direccionamiento**
- 2 Nuevo Hardware y Nuevo Software**
  - 2.1 Registros nuevos**
  - 2.2 Máquina elemental indexada**
  - 2.3 Hacia una estructura convencional**
- 3 Microprocesador 8088**
  - 3.1 Introducción**
  - 3.2 Diagrama en bloques**
- 4 Ejercitación**



## Capítulo 8

# Arquitectura Convencional

## 1 Visión General

En este Capítulo veremos inicialmente dos conceptos: el formato de las instrucciones y los modos de direccionamiento. Luego, haremos modificaciones a la Máquina Elemental para obtener nuestra Máquina Elemental Indexada (MEI), que se parece más a la arquitectura actual de una computadora. Después, agregaremos Interrupciones a la Máquina Elemental Indexada (MEI) y, finalmente, estudiaremos el microprocesador INTEL 8088 como ejemplo de máquina (CPU) convencional.

### 1.1 Formato de instrucciones

En sentido general, cualquier instrucción que involucre una operación diódica (aquellas que tienen dos operandos y un resultado), requiere cuatro piezas de datos además del código de operación, a saber:

- Ubicación del primer operando
- Ubicación del segundo operando
- Ubicación del resultado
- Ubicación de la próxima instrucción

#### 1.1.1 Formato de cuatro direcciones

Las primeras máquinas solían tener estas cuatro piezas en la propia instrucción (Figura 8.1). Por ejemplo:

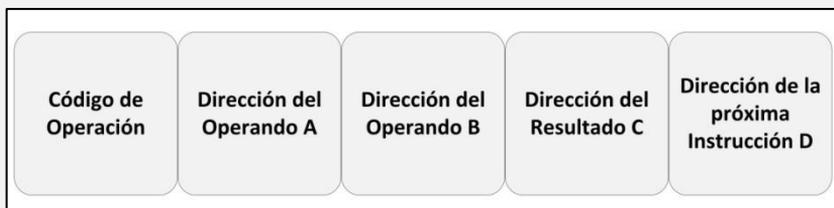


Fig. 8.1. Formato general de las instrucciones con cuatro direcciones.

La instrucción ADD A,B,C,D implica:

sumar  $A + B$ , colocar el resultado en C y buscar la próxima instrucción en D.

### 1.1.2 Formato de tres direcciones

La primera pieza a eliminar es la dirección de la próxima instrucción. Las instrucciones se colocan en forma sucesiva en la memoria. La próxima instrucción se obtiene entonces con ayuda de un registro llamado contador de programa (Fig. 8.2).

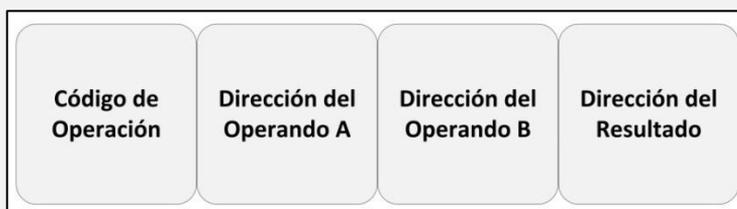


Fig. 8.2. Formato general de las instrucciones con tres direcciones.

La instrucción ADD A,B,C implica:

sumar  $A + B$ , colocar el resultado en C y buscar la próxima instrucción de acuerdo al PC.

### 1.1.3 Formato de dos direcciones

La siguiente pieza a eliminar es la ubicación del resultado (Fig. 8.3). Por ejemplo:

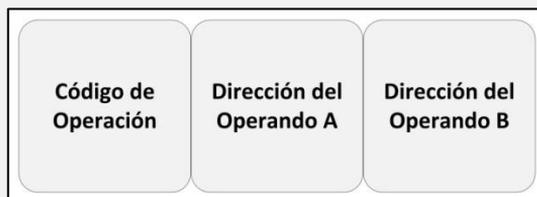


Fig. 8.3. Formato general de las instrucciones con dos direcciones.

La instrucción ADD A,B implica:

sumar  $A + B$ , colocar el resultado en B y buscar la próxima instrucción de acuerdo al PC. La dirección del operando B no se utiliza.

### 1.1.4 Formato de una dirección

La siguiente pieza a eliminar es la dirección del primer operando. Este operando se encuentra en un registro de la CPU (Figura 8.4). Por ejemplo:

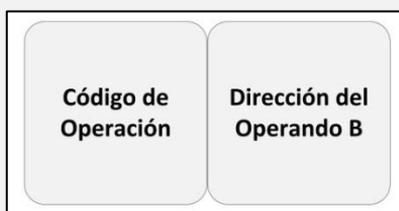


Fig. 8.4. Formato general de las instrucciones con una dirección.

La instrucción ADD B implica:

sumar el contenido del Acc + B, colocar el resultado en el Acc, y buscar la próxima instrucción de acuerdo al PC. Este es el caso de la máquina elemental vista en el Capítulo 7.

## 1.2 Modos de direccionamiento

Son medios que facilitan la tarea de programación, permitiendo el acceso a los datos. Estos Modos de Direccionamiento indican al procesador cómo calcular la dirección absoluta (real o efectiva) para determinar donde se encuentran los datos. Es decir, especifican la manera de obtener un operando.

El operando puede estar ubicado en:

- **la Memoria**,
- **un Registro** de la CPU,
- **la propia Instrucción** (el dato se encuentra dentro de la instrucción almacenada en el IR, y por lo tanto, en la CPU).

Es importante aclarar que, en general, en una instrucción que realiza alguna operación lógico-aritmética, están involucrados dos

operandos (sobre los cuales se realiza la operación). Pueden indicarse como OPER1 y OPER2, así:

MNEMÓNICO OPER2, OPER1

En el caso de instrucciones que realizan operaciones sobre un único operando, el OPER2 no aparece:

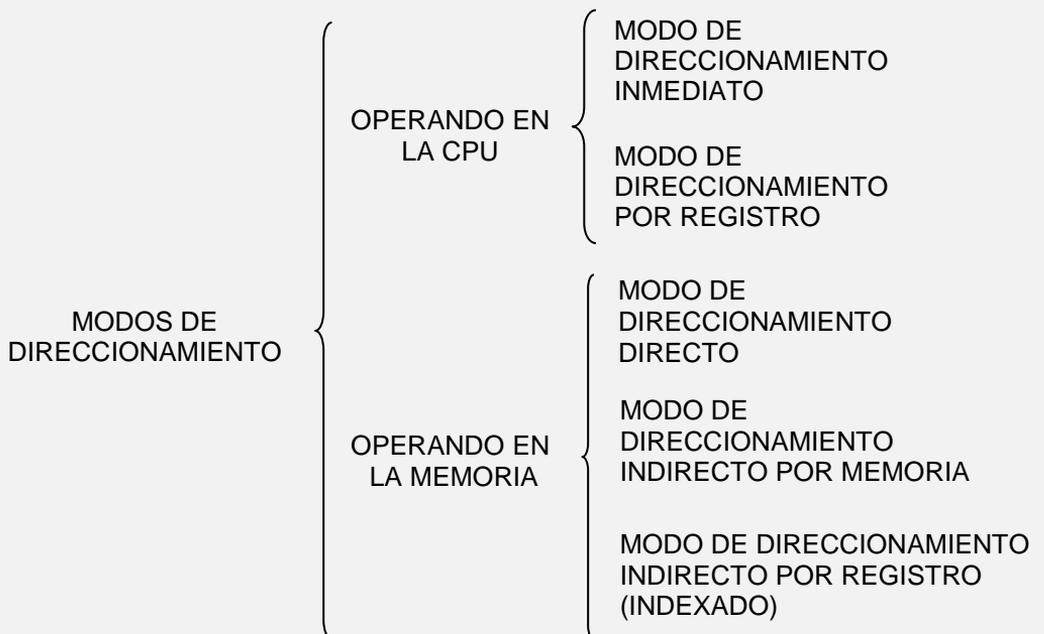
MNEMÓNICO OPER1

En el caso de instrucciones que realizan movimientos de datos, se suele indicar a los operandos como: OPERfuente y OPERdestino, indicando la ubicación origen y destino del movimiento, así:

MNEMÓNICO OPERdestino, OPERfuente

De lo mencionado, queda claro que si se trata de una instrucción que usa dos operandos, es posible que coexistan en la misma dos Modos de Direccionamiento.

De acuerdo a donde está ubicado el operando y a cómo se especifica, tenemos los siguientes modos de direccionamiento:



## 1.2.1 Operando en la CPU

### Modo de direccionamiento inmediato

El operando está en la instrucción. En la Máquina Elemental ya vista, no disponemos de este Modo.

### Modo de direccionamiento por registro

El operando está en un Registro de la CPU. En la Máquina Elemental este Modo está ejemplificado en las instrucciones:

- NOT: El Operando está en el ACC. Se trata de una operación sobre un único operando en el ACC.
- RAL: El Operando está en el ACC. Es un caso similar al anterior.
- CSA: El OPERfuente está en el SR y el OPERdestino está en el ACC. Se trata de una operación de movimiento de datos de un operando.

## 1.2.2 Operando en la memoria

### Modo de direccionamiento directo

La dirección del operando está en la instrucción. Ejemplo de este Modo son las siguientes instrucciones de la Máquina Elemental:

- LDA XXXX: El OPERfuente está en la dirección XXXX de Memoria (Modo de direccionamiento directo) y el OPERdestino en el ACC (Modo de direccionamiento registro).
- STA XXXX: El OPERfuente está en el Acc (Modo de direccionamiento por registro) y el OPERdestino en la dirección XXXX de Memoria (Modo de direccionamiento directo).

También son ejemplos de Modo de direccionamiento directo las instrucciones:

- ADD XXXX
- IOR XXXX
- XOR XXXX

En estos casos, el OPER2 está en la Memoria y su dirección (XXXX) se indica en la instrucción (Modo de direccionamiento directo) y el

OPER1 está en un registro de la CPU: el ACC (Modo de direccionamiento por Registro).

### **Modo de direccionamiento indirecto por memoria**

En la instrucción se especifica la dirección de la dirección del operando. Este modo de direccionamiento no está en la máquina elemental. El soporte del mismo implica ampliar el hardware y el software de la Máquina Elemental como veremos posteriormente.

### **Modo de direccionamiento indirecto por registro**

La dirección del operando (o parte de ella) está en Registros destinados a tal fin. Este modo de direccionamiento no está en la Máquina Elemental. También en este caso sería necesario ampliar los componentes de la Máquina Elemental. Este Modo de Direccionamiento es en realidad indirecto, ya que la dirección del operando no está en la instrucción sino en un Registro, razón por la cual también se lo suele llamar Indirecto por Registro.

Los modos de direccionamiento indirecto son muy útiles para el programador en relación con tareas muy comunes en el ámbito de la programación. Estas tareas son:

- Modificar direcciones,
- Generar contadores a fin de implementar lazos repetitivos,
- Reducir el espacio ocupado en memoria por las instrucciones,
- Permitir la reubicación del código, y
- Facilitar el manejo de las estructuras de datos, entre otras.

A continuación se presentan un par de ejemplos facilitados por este modo de direccionamiento:

- a) Supongamos que en la máquina elemental deseamos sumar sucesivamente un conjunto de 100 números ubicados correlativamente en la memoria. Deberíamos tomar los números uno a uno, sumarlos, colocar la suma parcial en alguna posición de memoria, hasta llegar al número 100.
- b) Otro problema común es mover un bloque de datos desde un sector a otro de la memoria.

Sería interesante contar con hardware y software que faciliten el incremento de la dirección de memoria y conteo de las operaciones realizadas. Surge así la idea de mejorar el hardware a fin de facilitar tareas como las mencionadas.

## 2 Nuevo Hardware y Nuevo Software

### 2.1 Registros nuevos

Vamos a agregar nuevos registros a la máquina elemental con el objeto de obtener algunas mejoras operativas.

#### 2.1.1 Registros índices

Vamos a agregar Registros Índice a la máquina elemental. Serán dos registros de 16 bits:

- Registro Índice Fuente (RIF)
- Registro Índice Destino (RID)

Para poder trabajar con Registros Índice, también modificaremos el formato de instrucción, a fin de poder indicar de qué modo de direccionamiento se trata. Planteamos el formato de la Figura 8.5.

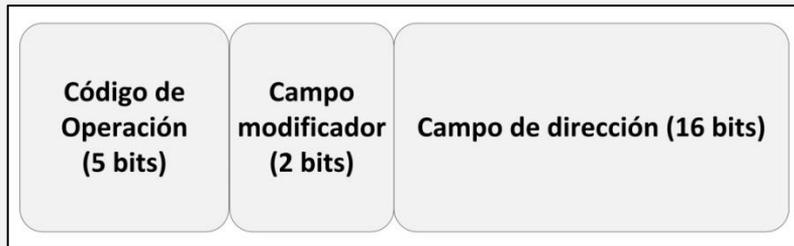


Fig. 8.5. Formato de instrucción modificado.

Con 5 bits de Código de Operación, podremos tener hasta 32 instrucciones. Y usando 16 bits en el Campo de Dirección podremos direccionar hasta 65536 posiciones de memoria. Mientras que con los dos bits en el Campo Modificador podremos seleccionar hasta 4 Modos de Direccionamiento:

- **Directo:**

Si el contenido del Campo Modificador es 0, usa el Campo de Dirección como la dirección de un operando, por ejemplo:

LDA 0, 1000

Carga el Acc con el contenido de la posición 1000

- **Indirecto por memoria:**

Si el contenido del Campo Modificador es 1, el Campo de Dirección apunta a una posición de memoria cuyo contenido no es el operando sino la dirección de éste. Por ejemplo, suponiendo que el contenido de la posición 1000 es 3400:

LDA 1, 1000

Carga el Acc con el contenido de la posición 3400

- **Indirecto por registro (Indexado):**

- Si el contenido del Campo Modificador es 2, el contenido del RIF se suma al Campo de Dirección obteniendo así la dirección del operando. Por ejemplo: Suponiendo que el contenido de RIF es 5000:

LDA 2, 1000

Carga el Acc con el contenido de la posición 6000 (1000 + 5000)

- Si el contenido del Campo Modificador es 3, el contenido del RID se suma al Campo de Dirección, obteniendo así la dirección del operando.

## Sumador de Direcciones

Para realizar la suma indicada para el direccionamiento indexado, agregamos un sumador en la CPU dedicado sólo a esta tarea. De esta forma, no usamos tiempo de la ALU. Este Sumador tiene dos registros de entrada: el Registro W y el Registro X de 16 bits cada uno. Además, siempre suma al resultado el contenido de un tercer registro que agregaremos: el Registro Base de 16 bits.

## Decodificador del Campo Modificador

Para poder decodificar el campo Modificador implementamos en la Unidad de Control un nuevo decodificador: el DECODIFICADOR DEL CAMPO MODIFICADOR, además del Decodificador de Instrucciones ya existente. Esto permite realizar simultáneamente la decodificación del Código de Operación de la instrucción y del Campo Modificador.

### 2.1.2 Registros base

Cuando un programador escribe un programa debe definir la dirección de comienzo del mismo, y la memoria que necesita para su programa y datos asociados. Una idea arbitraria, aunque comúnmente usada, es que comience en la dirección CERO. Sin embargo, hay razones por las cuales un programa no puede comenzar en la dirección CERO.

Una de ellas es que puede existir más de un programa residente en memoria en un momento dado (**Multiprogramación**) y no todos pueden comenzar en la dirección CERO. Otra, es que las primeras posiciones de memoria se reservan para otras tareas, como el manejo de entrada salida (se discutirá posteriormente). Aquí, se puede imaginar la necesidad de un programa administrador del sistema (que podríamos llamar **Monitor**). Este programa debería, entre otras tareas, encargarse de decidir la ejecución de los programas de usuario, asignándole direcciones de comienzo y área de memoria a utilizar (Proceso llamado **ensamblado del programa**).

El programador no tiene forma de conocer de antemano cual será la dirección de comienzo de su programa disponible en el momento de ejecución. Peor aún, una vez que el programa ha sido cargado, el Monitor puede necesitar reubicarlo a fin de consolidar áreas vacías en la memoria (desfragmentación).

Un Programa Cargador Reubicable (Relocating Loader) podría ser una solución a fin de reubicar programas en la memoria. Este programa del Monitor debería ser capaz de distinguir entre las constantes del programa de usuario (que no debe modificar) y direcciones reubicables del programa (que debe modificar) para luego ensamblarlo.

Una alternativa que permite reubicar programas utilizando una herramienta de hardware es agregar un registro nuevo llamado REGISTRO BASE de 16 bits. El Registro Base es como el Registro Índice en el sentido que su contenido se suma al campo de dirección de la instrucción a fin de obtener la dirección efectiva. La diferencia consiste en que:

- El usuario común no tiene control sobre el contenido del Registro Base, y
- El contenido del Registro Base se suma siempre a la dirección efectiva cuando se realiza una referencia a memoria en la Ejecución de la instrucción.

Aclaremos mediante un ejemplo. Supongamos que el usuario escribe un programa que comienza en la dirección CERO:

```
0000  ENI 3, 1000    coloque el valor 1000 en el Registro
                   Índice RID
0001  LDA 3, 1234   cargue el Acc con el contenido de la
                   posición 2234 (1234+1000)
                   .....
                   .....
```

Supongamos ahora que el Relocating Loader, por razones ya mencionadas, carga el programa a partir de la dirección 5000. El resultado de su ejecución introduce las siguientes modificaciones:

5000	ENI RID, 1000	coloque el valor 1000 en el Registro Índice RID
5001	LDA 3, 6234	cargue el Acc con el contenido de la posición 7234 (6234+1000)
	.....	
	.....	

Se observa que no se ha modificada la instrucción en la dirección 5000, porque se trata de una constante. Si se modifica la instrucción en la dirección 5001, ya que se trata de una dirección.

Si la máquina cuenta con un Registro Base, la tarea del Relocating Loader puede simplificarse. El desplazamiento de dirección (cargado en el Registro Base mediante una nueva instrucción privilegiada) se suma a la dirección efectiva sólo cuando se realiza una referencia a memoria. Las constantes están entonces inmunes a tal influencia. Cualquier referencia a memoria se modifica automáticamente en una cantidad igual a la dirección inicial del programa.

## 2.2 Máquina elemental indexada

Se propone una nueva Máquina Elemental Indexada (MEI) incluyendo los Registros Índice, el Registro Base y el Sumador de Direcciones con sus registros de entrada W, X, y RB. El diagrama en bloques de la máquina se muestra en la Figura 8.6.

En la figura se indica en *cursiva* el nuevo hardware. Obsérvese que el Sumador de Direcciones realiza la suma aritmética de direcciones sobre 3 operandos, que son:

- El Registro W de 16 bits
- El Registro X de 16 bits
- El Registro base de 16 bits

Siempre que se genera una dirección en tiempo de Ejecución se realiza esta suma. Nótese que esta suma no influye para la Búsqueda, ya que la dirección de la instrucción buscada es el contenido del Registro PC. El acarreo de este sumador no se tiene en cuenta.

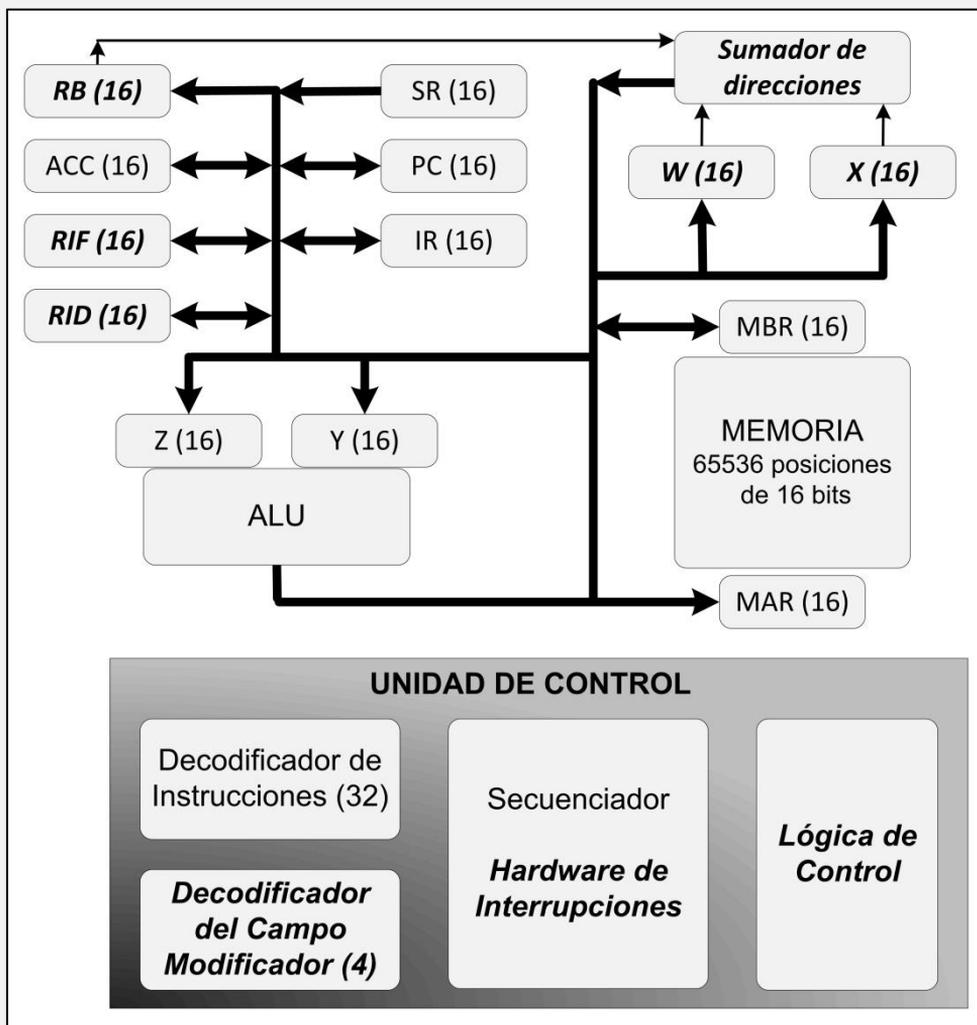


Fig. 8.6. Diagrama en bloques de la Máquina Elemental Indexada.

### 2.2.1 Conjunto de instrucciones de la máquina elemental indexada

En la Tabla 8.1 se presentan los detalles del conjunto de instrucciones de la Máquina Elemental Indexada.

Código Operación (Octal)	Mnemónico	Campo Modif. (Octal)	Campo de Dirección (Octal)	Acción	
00 (1)	HLT			La computadora se detiene. Si se presiona START comienza desde la instrucción siguiente a HLT	
01 (2)	ADD	00	0	XXXXXX	La suma aritmética del contenido del Acumulador con el contenido de la posición XXXXXX, se carga en el Acumulador (ACC). Si hay overflow la máquina se para.
		01	1	XXXXXX	La suma aritmética del contenido del Acumulador con el contenido de la posición indicada en XXXXXX, se carga en el Acumulador (ACC). Si hay overflow la máquina se para.
		10	2	XXXXXX	La suma aritmética del contenido del Acumulador con el contenido de la posición XXXXXX + RIF, se carga en el Acumulador (ACC). Si hay overflow la máquina se para.
		11	3	XXXXXX	La suma aritmética del contenido del Acumulador con el contenido de la posición XXXXXX + RID, se carga en el Acumulador (ACC). Si hay overflow la máquina se para.
02 (2)	XOR	00	0	XXXXXX	La XOR bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		01	1	XXXXXX	La XOR bit a bit del contenido del Acumulador con el contenido de la posición indicada en XXXXXX, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		10	2	XXXXXX	La XOR bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX + RFI, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		11	3	XXXXXX	La XOR bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX + RID, se carga en el Acumulador (ACC). Su contenido anterior se pierde
03 (2)	AND	00	0	XXXXXX	La AND bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		01	1	XXXXXX	La AND bit a bit del contenido del Acumulador con el contenido de la posición indicada en XXXXXX, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		10	2	XXXXXX	La AND bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX + RFI, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		11	3	XXXXXX	La AND bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX + RID, se carga en el Acumulador (ACC). Su contenido anterior se pierde
04 (2)	IOR	00	0	XXXXXX	La IOR bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		01	1	XXXXXX	La IOR bit a bit del contenido del Acumulador con el contenido de la posición indicada en XXXXXX, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		10	2	XXXXXX	La IOR bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX + RFI, se carga en el Acumulador (ACC). Su contenido anterior se pierde
		11	3	XXXXXX	La IOR bit a bit del contenido del Acumulador con el contenido de la posición XXXXXX + RID, se carga en el Acumulador (ACC). Su contenido anterior se pierde

Código Operación (Octal)		Mnemónico	Campo Modif. (Octal)	Campo de Dirección (Octal)	Acción
05 (1)		NOT			El contenido del Acumulador (ACC) se reemplaza por su complemento a uno.
06 (2)	00	LDA	0	XXXXXX	Carga el Acumulador (ACC) con el contenido de la posición XXXXXX
	01		1	XXXXXX	Carga el Acumulador (ACC) con el contenido de la posición indicada en la posición XXXXXX
	10		2	XXXXXX	Carga el Acumulador (ACC) con el contenido de la posición XXXXXX + RIF
	11		3	XXXXXX	Carga el Acumulador (ACC) con el contenido de la posición XXXXXX + RID
07 (2)	00	STA	0	XXXXXX	Guarda el contenido del Acumulador (ACC) en la posición de memoria XXXXXX
	01		1	XXXXXX	Guarda el contenido del Acumulador (ACC) en la posición de memoria indicada en la posición XXXXXX
	10		2	XXXXXX	Guarda el contenido del Acumulador (ACC) en la posición de memoria XXXXXX + RIF
	11		3	XXXXXX	Guarda el contenido del Acumulador (ACC) en la posición de memoria XXXXXX + RID
10 (2)		SRJ	XXXXXX	Sirve para hacer un salto del programa a una subrutina. Para esto realiza una copia del contenido del Contador de Programa (PC) en el Acumulador (ACC). Luego se copia el número XXXXXX en el contador del programa (PC) para que la próxima instrucción sea tomada de la dicha dirección.	
11 (2)		JMA	XXXXXX	Si el bit más significativo del Acumulador (ACC) es uno; el número XXXXXX se copia en el Contador de Programa (PC)	
12 (2)		JMP	XXXXXX	El número XXXXXX se copia en el Contador de Programa (PC)	
13 (1)		INP		XY	Si la Máquina no está atendiendo una Interrupción, inicia una transferencia E/S. Si la Máquina está atendiendo una Interrupción, carga el ACC <sub>0-7</sub> con el dato del PYY.
14 (1)		OUT		XY	Los 8 bits más significativos del acumulador ACC <sub>8-15</sub> se envían al dispositivo externo PYY.
15 (1)		RAL			Los bits del acumulador se rotan un lugar hacia la izquierda. El bit ACC <sub>15</sub> se coloca en ACC <sub>0</sub> , de modo que el desplazamiento es cíclico.
16 (1)		CSA			El número que está en el registro de llaves (introducido por las llaves de la consola) se copia en el acumulador.

Código Operación (Octal)		Mnemónico	Campo Modif. (Octal)	Campo de Dirección (Octal)	Acción
17 (1)		NOP			Esta instrucción no hace nada.
20 (1)	00	CRR	0		Copia el contenido del Acumulador (ACC) al Registro Índice Fuente (RIF)
	01		1		Copia el contenido del Acumulador (ACC) al Registro Índice Destino (RID)
	10		2		Copia el contenido del Registro Índice Fuente (RIF) al Acumulador (ACC)
	11		3	XXXXXX	Copia el contenido del Registro Índice Destino (RID) al Acumulador (ACC)
21 (2)	10	SIX	2	XXXXXX	Guarda el contenido del Registro Índice Fuente (RIF) en la posición de memoria XXXXXX
	11		3	XXXXXX	Guarda el contenido del Registro Índice Destino (RID) en la posición de memoria XXXXXX
22 (2)	10	LIX	2	XXXXXX	Carga al Registro Índice Fuente (RIF) con el contenido de la posición de memoria XXXXXX
	11		3	XXXXXX	Carga al Registro Índice Destino (RID) con el contenido de la posición de memoria XXXXXX
23 (2)	00	ENI	0	XXXXXX	Carga al Acumulador (ACC) con el número entero XXXXXX
	01		1	XXXXXX	Carga al Registro Base (RB) con el número entero XXXXXX. Sólo debe ser usada por el administrador del Sistema. (Instrucción privada).
	10		2	XXXXXX	Carga al Registro Índice Fuente (RIF) con el número entero XXXXXX
	11		3	XXXXXX	Carga al Registro Índice Destino (RID) con el número entero XXXXXX
24 (1)	10	INC	2	XXXXXX	Incrementa en XXX al Registro Índice Fuente (RIF). XXX es un número signado de 9 bits.
	11		3	XXXXXX	Incrementa en XXX al Registro Índice Destino (RID). XXX es un número signado de 9 bits.
25 (2)	10	JRI	2	XXXXXX	Si el contenido del Registro Índice Fuente (RIF) no es cero, salta a posición XXXXXX. Si el contenido es cero sigue en la siguiente instrucción
	11		3	XXXXXX	Si el contenido del Registro Índice Destino (RID) no es cero, salta a posición XXXXXX. Si el contenido es cero sigue en la siguiente instrucción

Código Operación (Octal)	Mnemónico	Campo Modif. (Octal)	Campo de Dirección (Octal)	Acción
26 (1)	SKF		XYX	Omite la siguiente instrucción si la BDYY = 0
27 (1)	ION			Habilita Interrupciones
30 (1)	IOF			Inhabilita Interrupciones
31 (1)	RTI			Retorno de Interrupción y habilitación del Sistema de Interrupciones

Tabla 8.1. Conjunto de instrucciones de la Máquina Elemental Indexada.

### 2.2.2 Ciclos de máquina

Los cambios en la Máquina Elemental también se reflejan en el ciclo de búsqueda y ejecución. A continuación se presentan desde la Tabla 8.2 a la Tabla 8.6 los casos para algunas de las nuevas instrucciones de la Blue:

**1) LDA 3, XXXXXX** Cargue el Acc con el contenido de la posición de memoria (XXXXXX + RID)

El aspecto de LDA 3, XXXXXX en la Memoria y los ciclos de búsqueda y ejecución se observan en la Tabla 8.2.

Código de Operación de 5 bits (LDA)	Campo Modificador de 2 bits (3)	Campo sin uso de 9 bits
Campo de Dirección de 16 bits (XXXXXX)		
<b>Búsqueda</b>		
Pulso de Reloj	Acción	
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura	
CP2	Envíe 1, Cargue Y	
CP3	-	
CP4	Envíe SUMA, Cargue PC	
CP5	-	
CP6	Envíe MBR, Cargue IR	
CP7	-	
CP8	Pasar a Ejecución	
<b>Ejecución</b>		
Pulso de Reloj	Acción	
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura	
CP2	Envíe 1, Cargue Y	
CP3	-	
CP4	Envíe SUMA, Cargue PC	
CP5	Envíe RID, Cargue X	
CP6	Envíe MBR, Cargue W	
CP7	-	
CP8	Seguir en Ejecución	
<b>Ejecución</b>		
Pulso de Reloj	Acción	
CP1	Envíe $\Sigma$ , Cargue MAR, Orden de Lectura	
CP2	-	
CP3	-	
CP4	-	
CP5	-	
CP6	Envíe MBR, Cargue Acc	
CP7	-	
CP8	Estado Búsqueda	

Tabla 8.2. Formato, y ciclos de búsqueda y ejecución de la instrucción LDA 3, XXXXXX.

El ciclo de búsqueda es igual al de la Máquina Elemental. El ciclo de ejecución consta de dos ciclos de memoria. En CP1 del primer ciclo de ejecución, se direcciona la posición siguiente para obtener el campo de dirección de la instrucción (tener en cuenta que esta instrucción ocupa dos palabras en Memoria). Luego, entre CP2 y CP4 se incrementa nuevamente al PC de forma tal que en el siguiente ciclo de búsqueda se apunte a la próxima instrucción. En CP5 y CP6 se cargan los registros de entrada al sumador de direcciones, para calcular la dirección efectiva del operando.

En CP1 del segundo de ejecución se direcciona al operando y se da orden de lectura. Después, entre CP2 y CP5 se espera a la memoria, y en CP6 se envía el dato leído al Acumulador. Finalmente, en CP8 se coloca a la máquina en búsqueda para la siguiente instrucción.

Se observa que para esta instrucción se utilizan 3 ciclos de máquina.

**2) LDA 1, XXXXXX**

Carga el ACUMULADOR con el contenido de la dirección indicada en el contenido de XXXXXX

El aspecto de LDA 1, XXXXXX en la Memoria, y los ciclos de búsqueda y ejecución se observan en la Tabla 8.3.

Código de Operación de 5 bits (LDA)	Campo Modificador de 2 bits (1)	Campo sin uso de 9 bits
Campo de Dirección de 16 bits (XXXXXX)		
<b>Búsqueda</b>		
<b>Pulso de Reloj</b>	<b>Acción</b>	
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura	
CP2	Envíe 1, Cargue Y	
CP3	-	
CP4	Envíe SUMA, Cargue PC	
CP5	-	
CP6	Envíe MBR, Cargue IR	
CP7	-	
CP8	Pasar a Ejecución	
<b>Ejecución</b>		
<b>Pulso de Reloj</b>	<b>Acción</b>	
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de lectura	
CP2	Envíe 1, Cargue Y	
CP3	-	
CP4	Envíe SUMA, Cargue PC	
CP5	Envíe 1, Cargue Z	
CP6	Envíe MBR, Cargue W	
CP7	Envíe ALU(XOR), cargue X	
CP8	Seguir en Ejecución	
<b>Ejecución</b>		
<b>Pulso de Reloj</b>	<b>Acción</b>	
CP1	Envíe Sumador, Cargue MAR, Orden de Lectura	
CP2	-	
CP3	-	
CP4	-	
CP5	-	
CP6	Envíe MBR, Cargue W	
CP7	-	
CP8	Seguir en Ejecución	

Búsqueda	
Pulso de Reloj	Acción
CP1	Envíe Sumador, Cargue MAR, Orden de Lectura
CP2	-
CP3	-
CP4	-
CP5	-
CP6	Envíe MBR, Cargue Acc
CP7	-
CP8	Pasar a Búsqueda

Tabla 8.3. Formato, y ciclos de búsqueda y ejecución de la instrucción LDA 1, XXXXXX.

Se observa que el direccionamiento indirecto requiere 4 ciclos de máquina

**3) LDA 0, XXXXXX** Cargue el Acc con el contenido de la posición XXXXXX

El aspecto de LDA 0, XXXXXX en la Memoria, y los ciclos de búsqueda y ejecución se observan en la Tabla 8.4.

Código de Operación de 5 bits (LDA)		Campo Modificador de 2 bits (0)	Campo sin uso de 9 bits
Campo de Dirección de 16 bits (XXXXXX)			
<b>Búsqueda</b>			
Pulso de Reloj	Acción		
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura		
CP2	Envíe 1, Cargue Y		
CP3	-		
CP4	Envíe SUMA, Cargue PC		
CP5	-		
CP6	Envíe MBR, Cargue IR		
CP7	-		
CP8	Pasar a Ejecución		
<b>Ejecución</b>			
Pulso de Reloj	Acción		
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura		
CP2	Envíe 1, Cargue Y		
CP3	-		
CP4	Envíe SUMA, Cargue PC		
CP5	Envíe 1, Cargue Z		
CP6	Envíe MBR, Cargue W		
CP7	Envíe ALU(XOR), Cargue X		
CP8	Seguir en Ejecución		
<b>Ejecución</b>			
Pulso de Reloj	Acción		
CP1	Envíe Sumador, Cargue MAR, Orden de Lectura		
CP2	-		
CP3	-		
CP4	-		
CP5	-		
CP6	Envíe MBR, Cargue Acc		
CP7	-		
CP8	Pasar a Búsqueda		

Tabla 8.4. Formato, y ciclos de búsqueda y ejecución de la instrucción LDA 0, XXXXXX.

#### 4) INC 2, XXX

Incrementar el Registro Índice RIF en el valor XXX (número signado de 9 bits)

El aspecto de INC2, XXX en la Memoria, y los ciclos de búsqueda y ejecución se observan en la Tabla 8.5.

Código de Operación de 5 bits (INC)	Campo Modificador de 2 bits (2)	Campo sin uso de 9 bits
<b>Búsqueda</b>		
<b>Pulso de Reloj</b>	<b>Acción</b>	
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura	
CP2	Envíe 1, Cargue Y	
CP3	-	
CP4	Envíe SUMA, Cargue PC	
CP5	-	
CP6	Envíe MBR, Cargue IR	
CP7	-	
CP8	Pasar a Ejecución	
<b>Ejecución</b>		
<b>Pulso de Reloj</b>	<b>Acción</b>	
CP1	Envíe RIF, Cargue Z	
CP2	Envíe IR <sub>0-8</sub> , Cargue Y	
CP3	-	
CP4	Envíe ALU(ADD), Cargue RIF	
CP5	-	
CP6	-	
CP7	-	
CP8	Pasar a Búsqueda.	

Tabla 8.5. Formato, y ciclos de búsqueda y ejecución de la instrucción INC2 valor XXXXXX.

**5) ENI 3, XXXXXX**

Cargar el RID con el valor XXXXXX

El aspecto de ENI RID, XXXXXX en la Memoria, y los ciclos de búsqueda y ejecución se observan en la Tabla 8.6.

Código de Operación de 5 bits (ENI)	Campo Modificador de 2 bits (3)	Campo sin uso de 9 bits
Número signado de 16 bits		
<b>Búsqueda</b>		
Pulso de Reloj	Acción	
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura	
CP2	Envíe 1, Cargue Y	
CP3	-	
CP4	Envíe SUMA, Cargue PC	
CP5	-	
CP6	Envíe MBR, Cargue IR	
CP7	-	
CP8	Pasar a Ejecución	
<b>Ejecución</b>		
Pulso de Reloj	Acción	
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura	
CP2	Envíe 1, Cargue Y	
CP3	-	
CP4	Envíe SUMA, Cargue PC	
CP5	-	
CP6	Envíe MBR, Cargue RID	
CP7	-	
CP8	Pasar a búsqueda.	

Tabla 8.6. Formato, y ciclos de búsqueda y ejecución de la instrucción ENI3, XXXXXX.

Comentarios:

- La Máquina Elemental Indexada no tiene formato fijo de instrucción,
- No todas las instrucciones que hacen referencia a memoria deben tener todos los modos de direccionamiento. Por ejemplo, la instrucción LIX 2, XXXXXX no posee modo de direccionamiento indexado ya que se estaría autoindexando.
- Las instrucciones pueden tener de 1 a 4 ciclos de máquina para ejecutarse, dependiendo de la instrucción y de su modo de direccionamiento. Se sugiere que el lector deduzca los ciclos de máquina para todas las instrucciones de Máquina Elemental Indexada (MEI) que se indican al final del Capítulo.

### 2.2.3 Interrupciones

Todas las entradas-salidas en la Máquina Elemental se llevan a cabo bajo control de la CPU, usando las instrucciones INP y OUT. Sin embargo, existen otras formas de manejar las operaciones de entrada-salida que no inmovilizan la CPU mientras ocurre una transferencia de datos. Esto tiene que ver con la velocidad de los periféricos que, en general, es inferior a la de la CPU. Por otro lado, existen eventos que se producen con poca frecuencia, o bien por su importancia, sería conveniente atenderlos en el momento que ocurren. Estos eventos son externos, es decir, no se generan directamente por el programa sino por algún periférico.

El modo más conveniente de manejar estos problemas es arreglar el hardware de tal forma que cuando estos eventos ocurren, se produzca una suspensión (interrupción) automática del programa corriente, y se transfiera el control temporariamente a una rutina diseñada especialmente para manejar estos eventos.

Así planteadas las cosas podríamos clasificar la transferencia de Entrada/Salida de la siguiente forma:



## **Manejo de Entrada/Salida Bajo Control del Procesador**

En general, las operaciones de entrada/salida son iniciadas por el procesador. Una vez iniciada la operación, el Procesador espera que la misma se complete, y luego, continúa con el programa principal. Este es el caso de la máquina elemental. Obsérvese que si la velocidad del dispositivo es inferior a la del Procesador (como es el caso general) este está inmovilizado por mucho tiempo. Peor aún, si el dispositivo sufre algún desperfecto, el Procesador puede esperar eternamente.

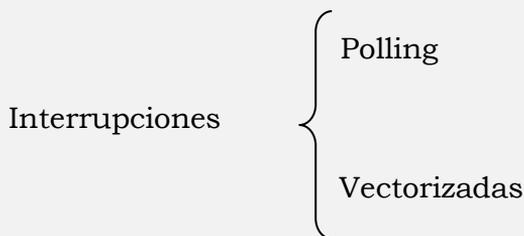
## **Manejo de Entrada/Salida con Interrupciones**

La otra opción, como se mencionó al principio, es iniciar la operación de entrada/salida y continuar con el programa principal. Cuando el dispositivo en cuestión termina, solicita un pedido de atención (solicitud de interrupción) que, al ser atendido por el Procesador, finaliza con la operación.

Este esquema es más eficaz que el anterior por cuanto no hay pérdida de tiempo esperando al periférico, pero requiere hardware e instrucciones especiales que lo soporten. Además, si existe más de un dispositivo que solicite atención, es necesario identificarlo y asignar prioridades.

Este esquema de interrupciones es útil también para el caso de dispositivos que requieren una operación de entrada/salida por sí mismos (no necesariamente iniciadas por el Procesador anteriormente).

Se verá más adelante que, de acuerdo a cómo se identifica al dispositivo que solicita la Interrupción, las interrupciones se pueden clasificar en:



Por último, antes de describir un Sistema Elemental de Interrupciones para la Máquina Elemental, podemos mencionar que el concepto de interrupción se extiende a eventos no necesariamente externos. Este es el caso de las llamadas interrupciones internas cuyo

origen proviene del interior del Procesador (desborde de registros, división por cero, código de operación no válido, etc.). Y las llamadas interrupciones por software, cuando se dispara un proceso de interrupción mediante instrucciones especiales.

## 2.2.4 Sistema elemental de interrupciones

Se define un Biestable en la Unidad de Control que se llama Sistema de Interrupción (SI). El valor o estado de dicho bit SI puede controlarse por medio de dos nuevas instrucciones: ION e IOF. La UC lo desactiva cuando se está procesando una interrupción, y lo activa cuando el proceso ha concluido.

- Si SI está en 1 el sistema de interrupciones está "habilitado"
- Si SI está en 0 el sistema de interrupciones está "deshabilitado"

Cada dispositivo con capacidad de interrumpir posee una línea de solicitud de interrupción que pone a 1 cuando necesita atención. Estas Banderas de Dispositivo (BD) van a una compuerta OR cuya salida, será una línea única de pedido IRQ de interrupción a la CPU.

Si las interrupciones están habilitadas (SI=1) e IRQ pasa a 1, se produce una interrupción en la ejecución del programa corriente, justo antes del comienzo del próximo ciclo de búsqueda. Es decir, la instrucción actual termina de ejecutarse.

Tres eventos ocurren cuando la CPU acepta una interrupción:

- El Biestable SI se coloca en 0, inhabilitando el sistema de interrupciones.
- El contenido del PC se guarda en la posición CERO.
- Se carga el PC con el valor 1 y se dispara un ciclo de búsqueda.

Estos eventos los lleva a cabo la Unidad de Control según el esquema de la Figura 8.7. El biestable PRO indica que la CPU está atendiendo una interrupción. Se verá que su estado condiciona las acciones que realiza la instrucción INP YY. En la figura se observan el Biestable Estado, el biestable PRO, el Biestable SI (ya mencionado) y un tercer Biestable auxiliar I (encargado de generar la señal I). Esta señal I dispara un ciclo de INTERRUPCIÓN.

Ahora existen tres estados en la máquina:

- Búsqueda (F)
- Ejecución (E)
- Interrupción (I)

El estado de Interrupción ocurre, según se ve en la Figura, cuando la línea IRQ está en 1 (indicando que algún dispositivo requiere atención), el sistema de interrupciones está habilitado (SI = 1), y con CP8 en el estado de Ejecución o al final de un ciclo de Búsqueda, si la instrucción corriente es de 1 ciclo (esto asegura la finalización de la ejecución de la instrucción corriente).

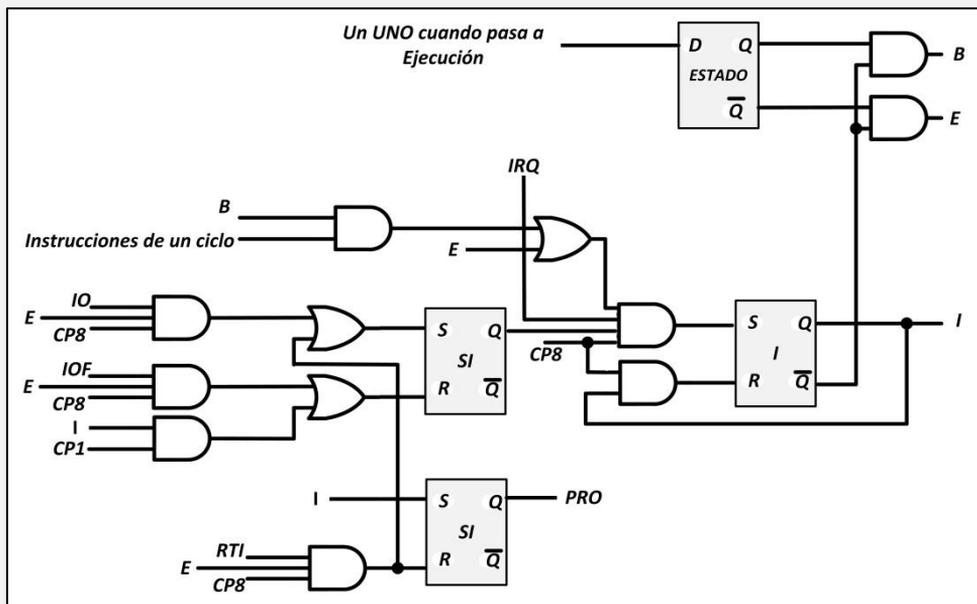


Fig. 8.7. Unidad de Control incluyendo sistema elemental de interrupciones.

### Ciclo de Interrupción

La siguiente Tabla 8.7 indica el ciclo de interrupción.

<b>Pulso de Reloj</b>	<b>Acción</b>
CP1	Envíe 1, cargue Z, Cargue Y, SI = 0
CP2	Envíe PC, Cargue MBR
CP3	Envíe ALU (XOR), Cargue MAR, OE
CP4	Envíe 1, Cargue PC
CP5	-
CP6	-
CP7	-
CP8	Biestable I = 0

Tabla 8.7. Ciclo de interrupción.

### **Rutina de Interrupción**

En la posición de memoria 1 debe haber una instrucción de salto a la rutina de interrupción cuya tarea consiste en la secuencia de acciones de la Figura 8.8.



Fig. 8.8. Secuencia de acciones de la rutina de interrupción.

- "Salvar" (guardar) en la memoria los contenidos de todos los Registros (Contexto), excepto el PC, a fin de restituirlos una vez terminado el proceso. Esta acción podría implementarse automáticamente en el ciclo de interrupción, el cual en este caso podría implicar más de un ciclo de máquina.
- Determinar quién causó la interrupción. Esta acción puede realizarse por POLLING o por los llamados VECTORES DE INTERRUPCIÓN.
- Saltar a la Subrutina de Atención del periférico determinado. Esta Subrutina se especifica para cada periférico y es la que realiza la transferencia de Entrada/Salida.
- Al finalizar, todas las Subrutinas de Atención de Periféricos saltan a la porción de código que restaura el contexto. Esto es esencial a fin de hacer transparente este proceso de interrupción al programa principal. El Programa principal continuará como si

nada hubiera pasado. Restaurar contexto implica volver a cargar los registros de la CPU con los valores que tenían antes.

- La acción de habilitar el SISTEMA de INTERRUPCIONES y retornar al Programa Principal debe realizarse en una misma instrucción llamada RTI: Retorno de Interrupción (a fin de evitar que, antes de retornar, se produzca otra interrupción pendiente). El ciclo de máquina de RTI se muestra en la Tabla 8.8.

<b>Pulso de Reloj</b>	<b>Acción</b>
CP1	Envíe PC, Cargue MAR, Orden de Lectura
CP2	-
CP3	-
CP4	-
CP5	-
CP6	Envíe MBR, IR
CP7	Envíe 1, Cargue Z, Cargue Y
CP8	Pasar a Ejecución

<b>Pulso de Reloj</b>	<b>Acción</b>
CP1	Envíe ALU(XOR), Cargue MAR, Orden de Lectura
CP2	-
CP3	-
CP4	-
CP5	-
CP6	Envíe MBR, Cargue PC
CP7	-
CP8	PRO = 0, SI = 0, Pasar a Búsqueda

Tabla 8.8. Ciclo de máquina de la instrucción RTI.

## Polling

A fin de determinar qué periférico solicitó interrupción puede aplicarse la técnica POLLING.

Se recuerda que cada dispositivo con capacidad de interrumpir posee una "Bandera de Dispositivo" (BD). Si BD está en 1 indica que solicita interrupción, y 0 en caso contrario.

En la técnica de polling se pregunta secuencialmente por las BD con un cierto orden. La primera BD =1 resulta en un salto a la subrutina de atención del periférico que corresponda. Se puede proponer una nueva instrucción que verifique estas banderas:

SKF XX	Omitir la próxima instrucción si XX es CERO, donde XX representa la bandera que se corresponde con los dispositivos que solicitan interrupción
--------	--

Esta parte de la rutina de interrupción (llamada POLLING), tendrá entonces este aspecto:

SKF 1	Si la bandera 1 es 0, omitir la próxima instrucción
JMP Rutina1	Salte a la Rutina de Atención del Dispositivo 1
SKF 2	Si la bandera 2 es 0, omitir la próxima instrucción
JMP Rutina2	Salte a la Rutina de Atención del Dispositivo 2
SKF 3	Si la bandera 3 es 0, omitir la próxima instrucción
JMP Rutina3	Salte a la Rutina de Atención del Dispositivo 3
SKF 4	Si la bandera 4 es 0, omitir la próxima instrucción
JMP Rutina4	Salte a la Rutina de Atención del Dispositivo 4
-	
-	

El orden en que se realiza la verificación de las banderas de dispositivo, en la Rutina de Interrupción, determina la prioridad del dispositivo que solicita atención (en caso de existir más de una solicitud a la vez). Con este esquema es probable que el último periférico nunca sea atendido (o difícilmente sea atendido). Una primera solución para este aspecto, es cambiar las prioridades alterando el orden de las instrucciones. Esto es poco práctico.

Una alternativa, que no se implementa en la Máquina Elemental Indexada, es agregar un nuevo registro llamado Registro Máscara (RM) cuyo contenido puede modificarse con una nueva instrucción: EMR Valor.

Este registro permite enmascarar (evitar dar curso a) solicitudes de interrupción. En la Figura 8.9 se presenta el sistema de interrupciones descrito:

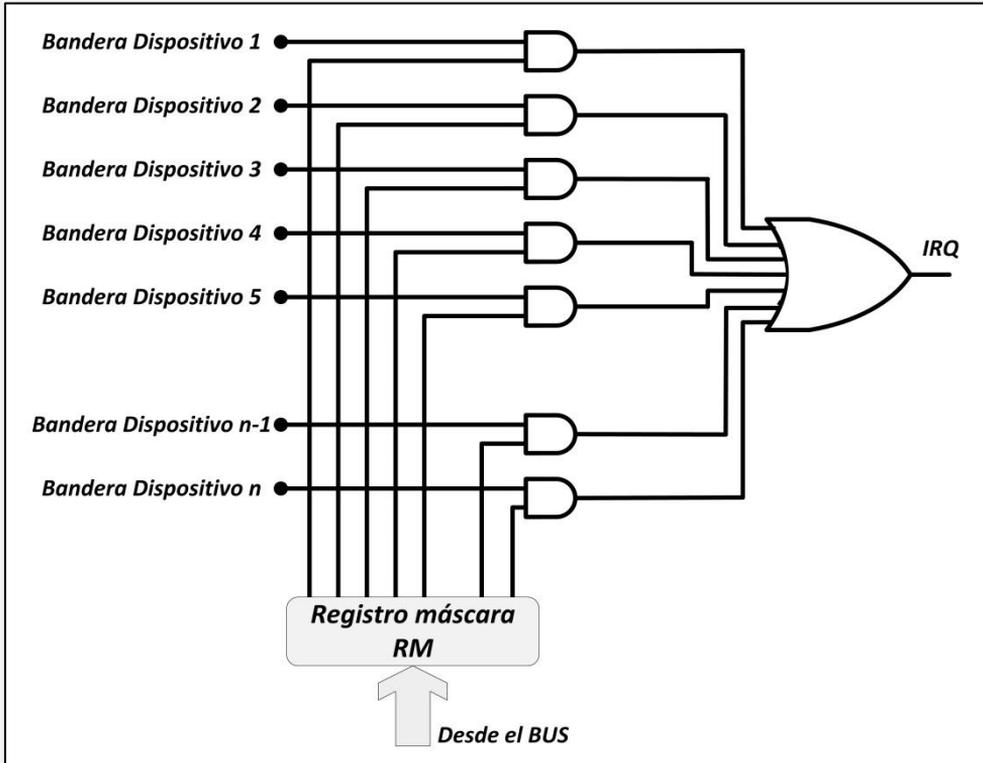


Fig. 8.9. Sistema de interrupciones usando Banderas de Dispositivos y un Registro de Máscaras para las prioridades.

Las Banderas de Dispositivo (BD), que son bits que provienen de los dispositivos que solicitan atención, se borran por la propia instrucción SKF XX después que lee la bandera.

## **Vector de Interrupciones**

Otro método (que no se implementa en la Máquina Elemental Indexada) que se usa para determinar quién interrumpe es el llamado VECTORIZADO. En este esquema, a cada dispositivo que puede interrumpir se le asigna una dirección llamada vector de interrupción (Trap Vector Address). Esta dirección es suplida por el dispositivo que interrumpe y en la misma, el programador debe almacenar la dirección de comienzo de la Rutina de Atención que corresponda. De esta forma toma lugar un salto indirecto a través del vector a la Rutina de Atención (Figura 8.10).

Como puede inferirse, este método es más rápido que el POLLING ya que no es necesario preguntar bandera a bandera. Por otro lado, es conveniente considerar una señal de reconocimiento de interrupción ACK, generada por la Unidad de Control, cuando ha reconocido una solicitud de interrupción y comienza el proceso de su atención.

La Figura 8.11 indica el conexionado externo a la CPU, y la señal de reconocimiento ACK que se encadena del Dispositivo 1 a los siguientes. La prioridad la tiene el Dispositivo de la izquierda.

La señal I (similar a la señal BD) es la bandera de solicitud de interrupción. Se pone en 1 si el dispositivo correspondiente necesita atención. Cuando el dispositivo recibe la señal de reconocimiento ACK vuelca al BUS (no dibujado) su dirección y pone a 0 su bandera I.

La señal M es un bit del registro máscara (ahora distribuido en los periféricos) y se pone a 0 o 1 por programa.

El hardware y software necesario para la implementación de la verificación vectorizada no se desarrolla aquí. Se invita al lector al planteo del mismo.

### **2.2.5 Inicio de una transferencia**

Para comprender cómo se realiza una transferencia de Entrada/Salida es necesario realizar algunas consideraciones respecto a los Periféricos.

Los periféricos son dispositivos de varias características, externos a la MEI. Básicamente, debemos considerar que los periféricos pueden poseer:

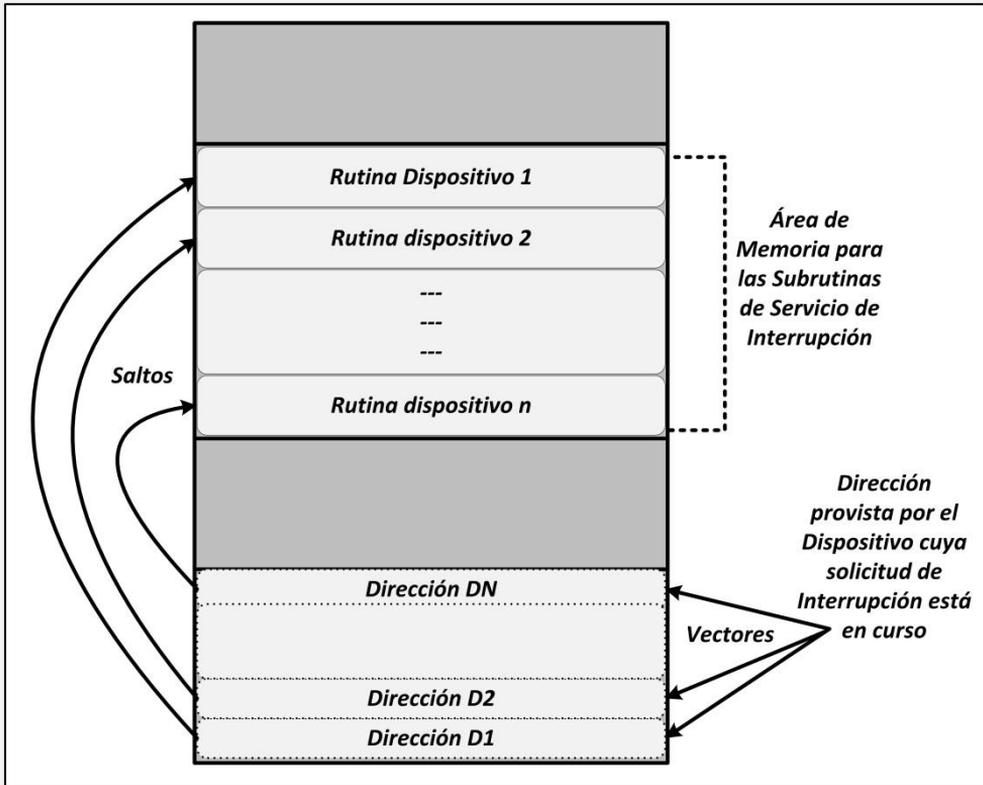


Fig. 8.10. Vector de Interrupciones.

- Elementos de micromecánica (servomotores, relay, motores paso a paso, sensores ópticos, sensores magnéticos, indicadores luminosos, componentes electrónicos de media potencia, etc.),
- Circuitos electrónicos que permiten controlar a los diferentes elementos constitutivos del periférico (CONTROLADOR), y
- Circuitos electrónicos que permiten la adaptación de los niveles de tensión y corriente (INTERFASE), para poder conectarlos a la MEI.

En general, las acciones de control de un periférico se realizan por los Controladores. Se trata de sistemas electrónicos dedicados específicamente al periférico en cuestión, de forma tal que el control a este nivel no es tarea de la CPU. Así, la CPU ve a los Periféricos como registros con los cuales se comunica. La mayoría de los periféricos (sus Controladores) dejan ver a la CPU los siguientes registros:

- **REGISTRO DE DATOS:** a través del cual fluyen los datos de la transferencia.
- **REGISTRO DE CONTROL:** a través del cual la CPU envía los comandos.
- **REGISTRO DE ESTADO:** refleja el estado del periférico.

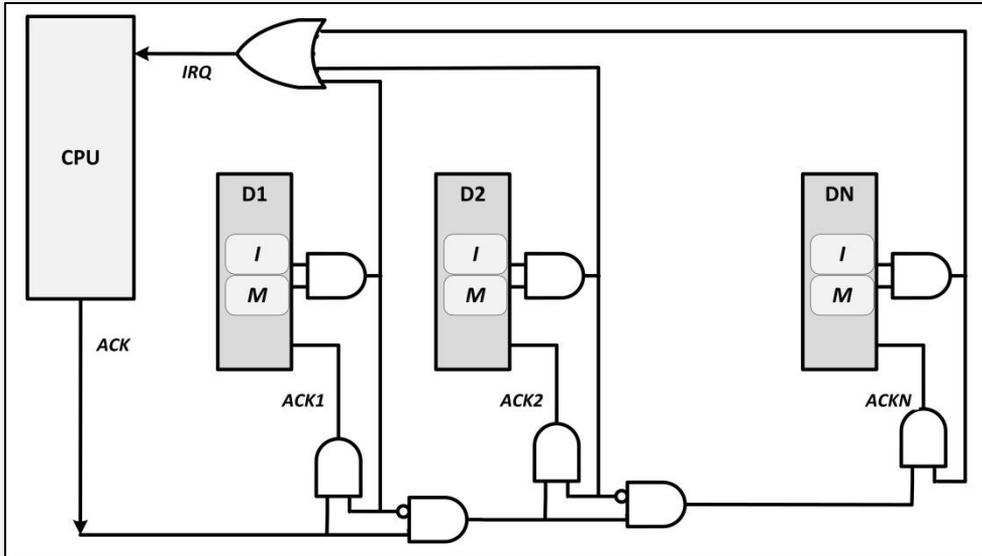


Fig. 8.11. Sistema de interrupciones usando bit I y M en los Dispositivos y la señal ACK de la CPU para las prioridades.

Los periféricos se clasifican en función de la dirección de los datos que fluyen por el Bus de E/S, en:

- Periféricos de Entrada
- Periféricos de salida.

Si los datos se dirigen hacia el Periférico se considera de Salida y viceversa.

A fin de simplificar la explicación vamos a considerar que los periféricos sólo tienen el REGISTRO DE DATOS.

### **Ciclo de las instrucciones de entrada salida con interrupciones**

Las transferencias de entrada salida, en general, son iniciadas por el programa, a través de las instrucciones:

- INP YY
- OUT YY

Estas instrucciones tienen ahora ciclos de máquina diferentes a los vistos en la máquina Elemental. La diferencia consiste en que no realizan la verificación de la señal R (Ready). Simplemente hacen  $TRA = 1$  durante un pulso de reloj, y luego,  $TRA = 0$ .

OUT YY ya no espera la señal R desde el Periférico, sólo genera un 1 (de un pulso de reloj de duración) que le indica al Periférico YY el comienzo de una transferencia de salida (Tabla 8.9). El periférico, al recibir  $TRA = 1$ , carga su Registro de Entrada con el valor de los 8 bits menos significativos del ACC a través del BUS de E/S, y comienza a realizar la acción correspondiente (por ejemplo imprimir). El programador deberá cargar el ACC con el dato que desea transferir, antes de la instrucción OUT YY.

Pulso de Reloj	Acción
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura
CP2	Envíe 1, Cargue Y
CP3	-
CP4	Envíe SUMA, Cargue PC
CP5	-
CP6	Envíe MBR, Cargue IR
CP7	$TRA = 1$
CP8	$TRA = 0$ , Pasar a Búsqueda

Tabla 8.9. Ciclo de la instrucción OUT YY.

Mientras que INP YY genera un pulso  $TRA = 1$  (igual que la instrucción OUT YY) sólo si  $PRO = 0$ , es decir, si la CPU no está involucrada en un Proceso de Interrupción (Tabla 8.10).

En el caso que la CPU esté en Proceso de Interrupción (en algún lugar de la Rutina de Interrupción), no se genera  $TRA$  y en CP8 se cargan los 8 bits más significativos del ACC con el valor del Registro de Salida del periférico YY (PYY) a través del Bus de E/S.

Pulso de Reloj	Acción
CP1	Envíe PC, Cargue MAR, Cargue Z, Orden de Lectura
CP2	Envíe 1, Cargue Y
CP3	-
CP4	Envíe SUMA, Cargue PC
CP5	-
CP6	Envíe MBR, Cargue IR
CP7	Si PRO = 0, TRA = 1
CP8	Si PRO = 1, Envíe PYY, Cargue ACC TRA = 0, Pasar a Búsqueda

Tabla 8.10. Ciclo de la instrucción INP YY.

Se sugiere al lector, como ejercitación, escribir una Subrutina para manejar una impresora (Periférico de salida), que es capaz de escribir un carácter ASCII por vez. Se desean escribir 100 caracteres ubicados en la Memoria desde la posición 1000.

## Controladores elementales de periféricos

### PERIFÉRICO DE SALIDA

Si se trata de un Periférico de Salida y se ha ejecutado una instrucción OUT YY, su Registro de Datos se cargará con el dato que esté en el BUS de E/S cuando el periférico seleccionado (YY) reciba la señal TRA e iniciará acciones.

Ejemplo para Periférico de Salida P32: OUT 32 (Figura 8.12)

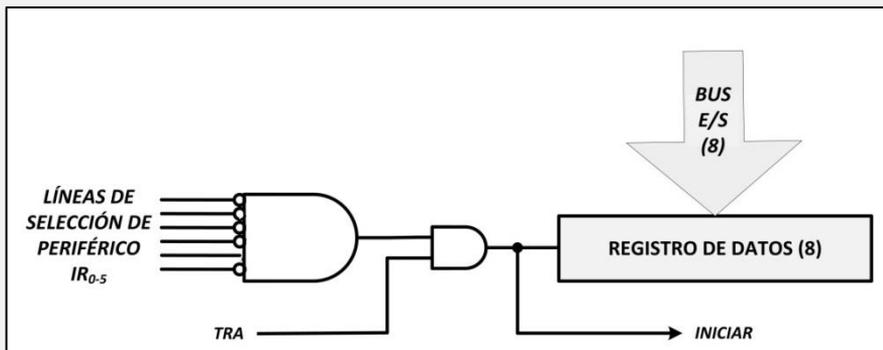


Fig. 8.12. Transferencia de salida sobre el periférico 32.

## PERIFÉRICO DE ENTRADA

Si se trata de un Periférico de Entrada y se ha ejecutado una instrucción INP YY pueden haber dos opciones:

- Si  $PRO = 0$ , la Máquina no está atendiendo una interrupción. Por lo tanto, cuando se ejecuta INP YY, el periférico YY recibe TRA e inicia las acciones para obtener el dato solicitado.
- Si  $PRO = 1$ , la Máquina está atendiendo una interrupción y su Registro de Datos tendrá el dato solicitado y estará disponible en el Bus de E/S cuando el periférico sea seleccionado (CP7 de la instrucción INP YY). La señal TRA no se genera en este caso por la instrucción INP YY.

Ejemplo para Periférico de Entrada P16: INP 16 (Figura 8.13)

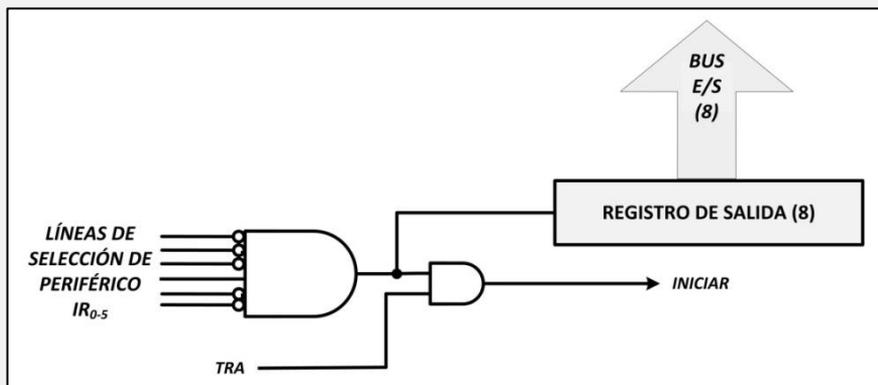


Fig. 8.13. Transferencia de entrada sobre el periférico 16.

## Registro puntero de pila

El Sistema de Interrupciones descrito no permite que la CPU sea interrumpida cuando está atendiendo una interrupción, ya que la dirección de retorno guardada en la posición CERO se perdería.

Si el contexto de la máquina (incluido el PC) se guardara en la memoria en una estructura LIFO (pila), cuya dirección inicial se indicara en un nuevo Registro, sería posible atender interrupciones dentro de una interrupción (interrupciones anidadas). A este registro se le llama REGISTRO PUNTERO DE PILA (SP). Por cada nueva interrupción, sólo debe incrementarse este Registro en la cantidad adecuada, y su contenido debe decrementarse por cada retorno de interrupción. Además, el biestable SI debería ponerse en 1 justo al comienzo de la Rutina de Atención de Dispositivo.

El programador debe escribir la Rutina de Interrupción de manera que los dispositivos de menor prioridad no puedan interrumpir un proceso de interrupción corriente de mayor prioridad. Y habría que agregar nuevo hardware y más instrucciones.

Se deja al lector, bosquejar la modelación de la Máquina Elemental Indexada para Interrupciones anidadas.

## 2.3 Hacia una estructura convencional

Nuestra nueva máquina, en lo esencial, se parece bastante a cualquier procesador actual.

La generalización de los conceptos introducidos previamente nos llevaría a la organización de la memoria en una máquina convencional.

Imaginemos que existe más de un programa en memoria (multiprogramación). En ese contexto, es necesario que:

- Cada programa (posiblemente de distintos usuarios) deba poseer un sector de memoria propio para su código y para los datos con los que opera (datos locales),
- Exista un sector destinado a los datos compartidos por varios programas (datos globales),
- Esté previsto un sector destinado al Sistema de Interrupciones,
- Haya un sector destinado a la PILA, y que
- Exista un sector destinado a un programa Monitor que administre el Sistema.

Como se observa, la memoria debe estar organizada en al menos cinco sectores y el Monitor debe cumplir básicamente con las siguientes tareas:

- Organizar la Memoria,
- Administrar la ejecución de los Programas (Reubicar Programas, Prioridades, Memoria disponible, Uso de Datos Globales),
- Administrar las Interrupciones (Habilitación, Prioridades), y
- Definir el tamaño de la PILA en función del estado actual del Sistema

En resumen, el Monitor debe administrar los recursos del Sistema adecuadamente. De él dependerá, en gran parte, el desenvolvimiento de la máquina ante los distintos requerimientos del usuario.

En el siguiente apartado se verá el microprocesador Intel 8088 que sirvió de CPU a las primeras PC (Computadora Personal). Su

arquitectura fundamental, dentro del alcance inicial de estas Guías Didácticas, se conserva en los procesadores actuales de nuestra máquina convencional.

## 3 Microprocesador Intel 8088

### 3.1 Introducción

El término microprocesador se refiere a una CPU contenida en un solo circuito integrado. Existen en el mercado varios microprocesadores que pueden ser utilizados como CPU de una máquina convencional. Se ha elegido el Intel 8088 (Figura 8.14) por ser el iniciador de las PC (IBM PC XT), tener una estructura similar a otros procesadores Intel (8086, 80186, 80286, 80386, 80486 y Pentium) y estar muy difundido.

<i>GND</i>	1		40	<i>VCC</i>
<i>A14</i>	2		39	<i>A15</i>
<i>A13</i>	3		38	<i>A16/S3</i>
<i>A12</i>	4		37	<i>A17/S4</i>
<i>A11</i>	5		36	<i>A18/S5</i>
<i>A10</i>	6		35	<i>A19/S6</i>
<i>A9</i>	7	<b>8</b>	34	<i>SS0</i>
<i>A8</i>	8	<b>0</b>	33	<i>MN/MX</i>
<i>AD7</i>	9	<b>8</b>	32	<i>RD</i>
<i>AD6</i>	10	<b>0</b>	31	<i>HOLD</i>
<i>AD5</i>	11	<b>8</b>	30	<i>HLDA</i>
<i>AD4</i>	12	<b>8</b>	29	<i>WR</i>
<i>AD3</i>	13	<b>8</b>	28	<i>I/O/M</i>
<i>AD2</i>	14	<b>8</b>	27	<i>DT/R</i>
<i>AD1</i>	15		26	<i>DEN</i>
<i>AD0</i>	16		25	<i>ALE</i>
<i>NMI</i>	17		24	<i>INTA</i>
<i>INTR</i>	18		23	<i>TEST</i>
<i>CLK</i>	19		22	<i>READY</i>
<i>GND</i>	20		21	<i>RESET</i>

Fig. 8.14. Configuración de pines del 8088.

El Intel 8088 tiene las siguientes características generales:

- Interfase al bus de datos de 8 bits,

- Arquitectura interna de 16 bits,
- Capacidad de direccionamiento de 1 Mbyte,
- Compatibilidad de Software con el 8086 CPU,
- Variedad en modos de direccionamiento,
- Operaciones en bloques, palabras y bytes,
- Aritmética signada y no-signada de 8 y 16 bits, en binario, decimal, incluyendo multiplicación y división, y
- 14 registros de 16 bits.

Algunas de las líneas (pines) del 8088 se describen en la Tabla 8.11. El bus de datos es de 8 bits, si bien el 8088 es un microprocesador de 16 bits. Además, está multiplexado en el tiempo, es decir, algunas líneas son de datos en un momento o direcciones de memoria en otro.

Símbolo	Tipo	Función
AD7-AD0	E/S	BUS DE DATOS Y DIRECCIONES. Estas líneas están multiplexadas en el tiempo.
A15-A8	S	BUS DE DIRECCIONES
A19-A16 S6-S3	S	BUS DE DIRECCIONES Y ESTADO. Estas líneas están multiplexadas en el tiempo.
RD WR	S	BUS DE CONTROL. Líneas de control de lectura y escritura
INTR NMI	E	Líneas de pedido de interrupción.
READY TEST RESET MN/MX	E	BUS DE CONTROL. Líneas de control de entrada
IO INTA ALE DT/R DEN S2, S1, S0	S	BUS DE CONTROL. Líneas de control de salida
VCC	E	Alimentación
GND	E	Tierra

Tabla 8.11. Símbolos, tipo y función de algunos pines del 8088.

### 3.2 Diagrama en Bloques (Figura 8.15)

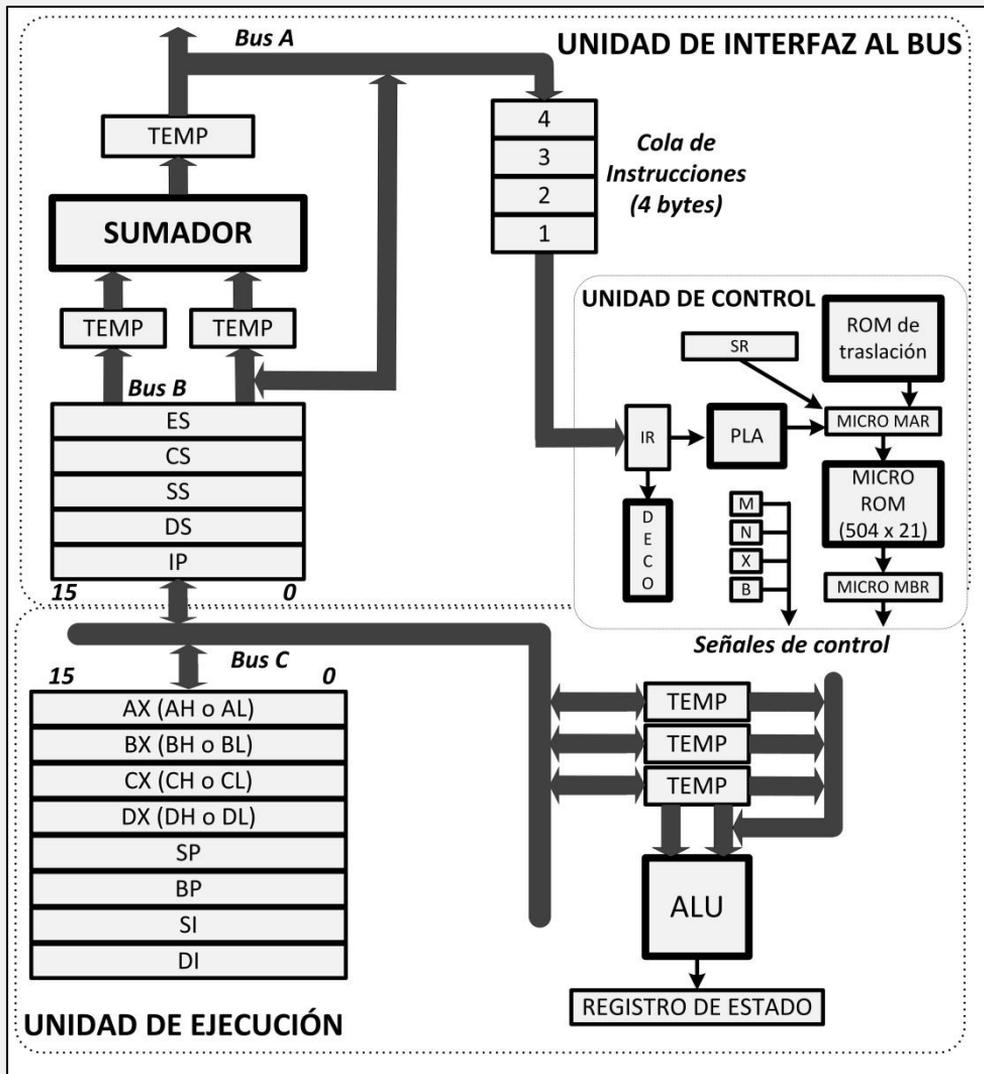


Fig. 8.15. Diagrama en bloques del microprocesador 8088.

### 3.2.1 BIU Y EU

Las funciones internas del 8088 están divididas lógicamente en dos unidades de procesamiento: la BIU y la EU. Estas unidades pueden interactuar directamente, realizando las operaciones asincrónicamente.

- 1) La **Unidad de Interfaz al bus (BIU)** cumple las siguientes funciones:
  - a) Busca las instrucciones en la memoria.
  - b) Llena la cola de instrucciones que consta de 4 bytes (FIFO).
  - c) Proporciona el control del bus.
  - d) Proporciona a la EU los operandos a procesar.

La BIU está formada por un sumador, un conjunto de 5 registros de 16 bits y la cola de instrucciones (4 bytes). Los circuitos relacionados con el control del bus y tareas de pre-búsqueda de instrucciones que realiza la BIU, no están indicados en la Figura 8.15.

- 2) La **Unidad de ejecución (EU)** recibe la instrucción buscada previamente por la BIU y se encarga de ejecutarla. La EU consta de la Unidad de Control (CU), la Unidad Lógico-Aritmética (ALU) y un conjunto de 9 registros de 16 bits.

En la Figura 8.15 se observa que el 8088 posee tres buses internos (A, B, C). La ALU posee tres registros temporarios de entrada (estando limitada una entrada a uno solo de ellos). La salida de la ALU (sin registro temporario) puede fluir por el bus a cualquier registro, incluidos los de su propia entrada. Puede realizar operaciones de 8 o 16 bits, y las condiciones resultantes son almacenadas en el registro de condiciones.

Cuando la BIU detecta que el bus externo está ocioso, envía una solicitud a la memoria para leer el siguiente byte en el flujo de instrucciones. Los bytes leídos son almacenados temporariamente en la cola de instrucciones. Cuando la EU requiere un nuevo byte de instrucción, lo toma de esta cola. La dimensión de cuatro bytes de la cola responde al compromiso de que la EU no tenga que estar esperando por un nuevo byte por un lado. Y por otro lado, colas demasiado largas ocuparían mucho al bus llenándose con bytes que podrían no utilizarse (por ejemplo, cuando se ejecuta una instrucción de salto).

La Unidad de Control del 8088 es microprogramada, y posee una ROM de 504 palabras de 21 bits (se recomienda leer la máquina elemental microprogramada) que almacena, aproximadamente, 90 microprocedimientos. Cada instrucción, para ejecutarse, requerirá de al menos un microprocedimiento.

La instrucción que se encuentra almacenada en la cola de instrucciones, es transferida al registro de instrucciones (IR), el decodificador la extrae y la disemina por la Unidad de Control. La información relacionada con la fuente y destino de los operandos se transfiere a los registros M y N. El código de operación se transfiere al registro X (para indicar a la ALU la operación a realizar) y al registro B (para indicar a la ALU si se trata de una operación de 8 o 16 bits). El código de operación también se transfiere a un combinacional (PLA) a fin de obtener la dirección de comienzo del microprocedimiento correspondiente. Cada microprocedimiento tiene como máximo 16 microinstrucciones, y no son exclusivos de una instrucción. Por ejemplo, existen microprocedimientos comunes a todas las instrucciones como los relacionados con el cálculo de direcciones.

El formato de una microinstrucción del 8088 se observa en la Figura 8.16, e incluye los siguientes campos:



Fig. 8.16. Formato de una microinstrucción del 8088.

- Campo Fuente: Indica el registro fuente de una operación,
- Campo Destino: Indica el registro destino de una operación,
- Campo Tipo: Indica el tipo de microinstrucción:
  - Operación de la ALU
  - Operación de la memoria
  - Salto corto
  - Salto largo
  - Llamada a microprocedimiento
  - Contabilidad
- Campo ALU: Indica la operación que debe realizar la ALU,

- Campo Registro: Proporciona el operando,
- Campo de Condición. Indica la activación de registro.

Las microinstrucciones se ejecutan una por ciclo de reloj. La dirección de la primera microinstrucción a ejecutar la proporciona el código de operación de la instrucción a través de la PLA, cargando el microMAR. Se observa en la Figura 8.14 que el microMAR puede ser cargado también desde la ROM de traslación, que mapea direcciones de 5 bits y desde el registro SR. La ROM de traslación se usa cuando el microprograma requiere de un salto, y el registro SR se utiliza para guardar la dirección de retorno de una micro-subrutina.

El estudio en detalle del funcionamiento de la Unidad de Control del 8088 escapa al alcance de esta Guía Didáctica, no obstante, se puede considerar similar a la máquina elemental microprogramada.

El microprograma del 8088, como el de otros microprocesadores, no es accesible al usuario. Su estructura en particular en cada micro es consecuencia de consideraciones tecnológicas y de mercado. Aunque, posteriormente se harán consideraciones generales respecto a las ventajas y desventajas de la microprogramación.

### 3.2.2 Registros del 8088

El 8088 tiene catorce registros que se agrupan en las siguientes tres categorías:

**1) Registros generales:** Ocho registros generales de 16 bits se dividen en dos grupos:

- **Cuatro registros direccionables como de 16 bits u 8 bits:**

**AX** (Acumulador), usado para almacenar resultados de operaciones, lectura/escritura desde o hacia la memoria o los puertos. (AH, AL)

**BX** (Base), usado en direccionamiento. (BH, BL)

**CX** (Contador), usado en interacciones como contador. (CH, CL)

**DX** (Datos), usado para el acceso de datos. (DH, DL)

- **Cuatro registros índice y registros base:**

**BP** (Puntero de base), usado en direccionamiento.

**SI** (Índice), usado en direccionamiento.

- DI** (Índice), usado en direccionamiento.
- SP** (Puntero de pila), apunta a la última dirección de pila utilizada.

**2) Registros de segmento:** Cuatro registros de 16 bits de propósitos especiales, están relacionados con la segmentación de la memoria.

- CS** Selecciona el área de memoria destinada al código del programa
- DS** Selecciona el área de memoria destinada a los datos.
- SS** Selecciona el área de memoria destinada a la pila (STACK).
- ES** Selecciona el área de memoria destinada

**3) Registros de control y estado:** Dos registros de 16 bits de propósitos especiales:

- F** Registro de condiciones
  - Bit 0: C Acarreo, C=1 si hay acarreo.
  - Bit 1: No usado
  - Bit 2: P Paridad, P=1 si el resultado tiene un número par de unos.
  - Bit 3: No usado
  - Bit 4: AC Acarreo auxiliar, AC = 1 si hay acarreo del bit 3 al bit 4 del resultado.
  - Bit 5: No usado
  - Bit 6: ZCero, Z=1 si el resultado en cero.
  - Bit 7: SSigno, S=1 si el resultado es negativo.
  - Bit 8: TTrap, T=1 si ocurre alguna interrupción.
  - Bit 9: I Interrupción, I=1 indica que las interrupciones están habilitadas.
  - Bit 10: D Dirección, D=1 indica sentido de alto a bajo en el procesamiento de cadenas.
  - Bit 11: O Overflow, O=1 si hay overflow.
  - Bit 12: No usado
  - Bit 13: No usado
  - Bit 14: No usado
  - Bit 15: No usado
  
- IP** **Puntero de instrucciones**, indica la dirección de la próxima instrucción a ejecutar.

### 3.2.3 Organización de la memoria

La memoria está organizada como un conjunto de cuatro segmentos, cada uno como una secuencia lineal de hasta 64 kbytes. La

memoria se direcciona usando dos componentes de dirección consistentes en un selector de segmento de 16 bits y un offset de 16 bits (Figura 8.17). El primero indica el segmento seleccionado y el segundo indica el byte deseado dentro del segmento. Todas las instrucciones que direccionan operandos en memoria deberán especificar el segmento y el offset. Sin embargo, en la mayoría de los casos, no es necesario indicar en forma explícita el segmento en uso. El registro de segmento correcto es elegido automáticamente de acuerdo a la Tabla 8.12. Existen Instrucciones especiales que permiten especificar el segmento para casos determinados.

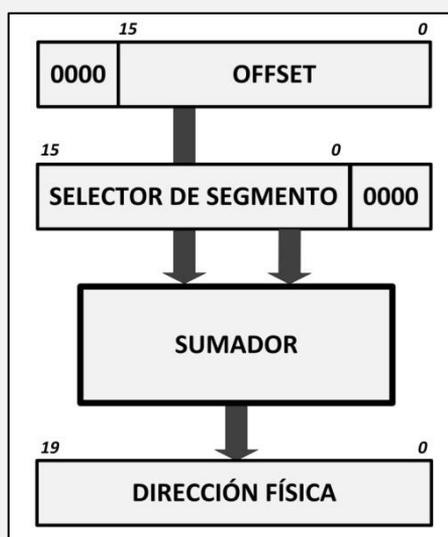


Fig. 8.17. Suma del segmento y offset.

Cabe mencionar que no toda la memoria se encuentra disponible para cualquier finalidad. Las primeras 03FFh posiciones (como se verá posteriormente) se encuentran destinadas para el proceso de interrupciones. Las últimas Fh posiciones (desde la FFF0h a la FFFFh) están reservadas para operaciones que tienen que ver con la carga del programa inicial. Cuando el 8088 recibe una señal en el pin RESET inicializa al IP de modo que apunte a la posición FFF0h, donde debería estar una instrucción de salto a una rutina de inicio. De aquí se desprende que no toda la memoria debe ser RAM. Necesariamente algunos sectores deben de ser memoria ROM.

REF. NECESARIA	REGISTRO USADO	REGLA
Instrucción	CS	Automáticamente con la pre-búsqueda de la instrucción
Stack	SS	Operaciones de Push y Pop y ref. a memoria que usan el registro. BP
Dato local	DS	Todas las referencias a datos excepto los relacionados al stack u operaciones de string.
Dato externo	ES	Segmento de datos alternativo y destino de operaciones de string.

Tabla 8.12. Selección del registro de segmento.

### 3.2.4 Modos de direccionamiento

En el 8088 son posibles ocho modos de direccionamiento para especificar operandos. Dos de ellos son provistos para instrucciones que operan sobre registros o con operandos inmediatos:

**Modo de operando en registro:** El operando está en uno de los registros generales (8 o 16 bits). Por ejemplo:

MOV AX, DX                      (DX) → (AX)

**Modo de operando inmediato:** El operando está incluido en la instrucción. Por ejemplo:

MOV AH, 14                      14 → AH

Los seis modos de direccionamiento restantes permiten especificar operandos ubicados en un segmento de memoria. Como se observó, una dirección de operando en memoria posee dos componentes de 16 bits: el selector de segmento y el offset. El selector de segmento se sufre por alguno de los registros de segmento, mientras que el offset se calcula por la suma de diferentes combinaciones de los siguientes tres elementos de dirección:

- **el desplazamiento:** valor de 8 o 16 bits incluido en la instrucción.

- **la base:** contenido de cualquier registro base BP o BX.
- **el índice:** contenido de cualquier registro índice SI o DI.

La combinación de estos elementos de dirección define los siguientes seis modos:

**Modo directo:** el offset está contenido en la instrucción como un desplazamiento. Por ejemplo:

MOV [14], AL                      (AL) → (DS) \* 10h + 14h

**Modo indirecto por registro:** el offset está contenido en uno de los registros BX, BP, SI o DI. Por ejemplo:

MOV [BX], CX.                      (CX) → (DS) \* 10h + (BX)

**Modo basado:** el offset resulta de la suma de un desplazamiento y el contenido de BP o BX. Por ejemplo:

MOV [BP + 3], 2A                      2A → (DS) \* 10h + (BP) + 3h

**Modo indexado:** el offset resulta de la suma de un desplazamiento y el contenido de SI o DI. Por ejemplo:

MOV [SI + 3], 2A                      2A → (DS) \* 10h + (SI) + 3h

**Modo basado indexado:** el offset resulta de la suma del contenido de un registro base y de un registro índice. Por ejemplo:

MOV [BP + SI], 2A                      2A → (DS) \* 10h + (BP) + (SI)

**Modo basado indexado con desplazamiento:** el offset resulta de la suma del contenido de un registro base más un registro índice y un desplazamiento. Por ejemplo:

MOV [BP + SI + 3], 2A                      2A → (DS) \* 10h + (BP) + (SI) + 3h

Cabe aclarar que cualquier acarreo en las sumas se ignora y que el desplazamiento es un valor signado. En el caso de ser un valor de 8 bits se extiende a 16 bits con el MSB como bit de signo.

También cabe aclarar que el offset puede obtenerse del registro IP en la búsqueda de una instrucción o en las instrucciones de salto. En este último caso, el contenido de IP puede modificarse de tres formas:

- salto relativo: al contenido de IP se le suma un desplazamiento.

- salto directo: al contenido de IP se lo cambia por un desplazamiento.
- salto indirecto: el contenido de IP es cambiado por el offset obtenido por cualquiera de los modos indirectos vistos anteriormente.

### 3.2.5 Conjunto de instrucciones

Las instrucciones del 8088 se dividen en siete categorías:

- Transferencia de datos
- Aritméticas
- Lógicas
- De desplazamiento y rotación
- De manipulación de cadenas.
- De control de programa
- De control del procesador

Una instrucción puede referirse a cero, uno o dos operandos, donde un operando puede residir en un registro, en la propia instrucción o en la memoria. Las instrucciones con cero operando (NOP, HLT) tienen un byte de longitud. Las de un operando (INC, DEC) tienen usualmente dos bytes de longitud. Las de dos operandos (MOV, ADD) usualmente tienen una longitud de tres a seis bytes y pueden hacer referencia a un registro o a una locación de memoria. Así, estas instrucciones permiten los siguientes cinco tipos de operaciones:

- registro a registro,
- memoria a registro,
- inmediato a registro,
- registro a memoria, e
- inmediato a memoria

Los tipos de datos que soporta el 8088 son:

- Entero: Valor numérico signado de 8 o 16 bits. Todas las operaciones asumen la representación en complemento a dos.
- Ordinal: Valor numérico sin signo de 8 o 16 bits.
- Puntero: Una cantidad de 32 bits compuesto por un selector de segmento y un offset, cada uno de 16 bits.
- Cadena: Secuencia continua de bytes o palabras, puede contener desde 1 a 64 Kbytes.
- BCD: Un byte que representa un dígito BCD.
- BCD empaquetado: Un byte que representa dos dígitos BCD, uno en cada nibble.

### 3.2.6 Direccionamiento de Entrada/Salida

Las operaciones de entrada salida pueden direccionar hasta 64K registros de E/S. La dirección de la E/S aparece en el bus de direcciones de la misma forma que una dirección de memoria.

Las instrucciones de entrada/salida de puerto variable usan al registro DX para contener la dirección del registro E/S (puerto), y tienen la capacidad total de direccionamiento. En cambio, las instrucciones de puerto fijo sólo pueden direccionar los primeros 256 locaciones de E/S en la página cero.

### 3.2.7 Interrupciones del 8088

El 8088 está provisto con un sistema de interrupciones vectorizado. Se pueden clasificar en:

- **Interrupciones iniciadas por hardware**

Externas

- enmascarables (a través del pin INTR)
- no enmascarables (a través del pin NMI)

Internas

- **Interrupciones iniciadas por software** (a través de la instrucción INT XX)

Una interrupción resulta en la transferencia del control a un nuevo programa. En las primeras 1024 (03FF) posiciones de memoria reside una tabla de 256 elementos, que contiene punteros a programas de servicio de interrupción. Cada elemento (puntero) es de 4 bytes y se corresponde con lo que se llama tipo de interrupción. El dispositivo que interrumpe suministra, durante el proceso de reconocimiento de interrupción, un valor de 8 bits que se usa como un vector hacia el tipo de interrupción que corresponda.

Si la interrupción se inicia desde el pin NMI (que tiene prioridad sobre el pin INTR), se producirá una interrupción de tipo 2, que se usa, en general, para activar rutinas de falla de alimentación. El vector es suplido internamente. Cuando ocurre una interrupción de este tipo, automáticamente se borra el bit I del registro de estado, a fin de enmascarar posibles interrupciones iniciadas por el pin INTR.

Si la interrupción es requerida desde el pin INTR y las interrupciones se encuentran habilitadas (bit I=1 del registro de estado) se inicia el proceso de atención de interrupción. Se termina de ejecutar la instrucción corriente, el bit I se borra, se salva el contexto actual (contenido del conjunto de registros) en el stack, se emite la señal de reconocimiento de interrupción (INTA), al recibirla el dispositivo que interrumpe vuelca en el bus su vector. Este vector se utiliza para determinar el puntero a la rutina de atención de interrupción, y se transfiere el control a dicha rutina. Una vez terminada esta rutina se restituye el contexto y se continúa con la ejecución del programa interrumpido. En el caso de existir más de un dispositivo externo que pueda interrumpir, se necesita adicionar al hardware un controlador de interrupciones (por ejemplo, el controlador 8258 es capaz de manejar hasta 8 dispositivos).

Las interrupciones por software reciben igual tratamiento que las iniciadas desde INTR. El vector, en este caso, lo suministra el campo XX de la instrucción INT XX.

No todos los vectores están disponibles para ser utilizados. El 8088 se reserva algunos para usos específicos.

En el Microprocesador 8088, como en la mayoría de CPUs de PCs, las Rutinas de Servicio no forman parte del programa que se ha interrumpido, sino que son aportadas por el sistema operativo (se cargan del disco) o bien directamente están en la memoria ROM (BIOS – Basic Input Output System). Una rutina de servicio de interrupción es una porción de código que se ocupa de llevar a cabo una tarea específica, propia del tipo de interrupción que soporta.

Durante la ejecución de un programa, todos los registros internos del procesador (PC, AX, SP, etc.) contienen información vinculada al mismo. Como la rutina de servicio utiliza los mismos registros para su ejecución, es necesario que al momento de la interrupción, la información contenida en los registros se resguarde en memoria y se recupere después que finaliza la ejecución de la rutina de servicio. Esto permite que el programa original pueda continuar. Se comprende la importancia de preservar el contenido del contador de programa o puntero de instrucciones que contiene siempre la dirección de la próxima instrucción a ejecutar.

Los contenidos de los registros se almacenan en una zona de memoria principal denominada Pila (Stack), que trabaja en modalidad LIFO (Last In First Out).

El retorno desde la rutina de servicio estará determinado por una última instrucción (IRET) que rescata los registros de la Pila, salvaguardados durante el ciclo de reconocimiento de interrupción, cargándolos en la CPU.

## 4 Ejercitación

### Ejercicio 1:

Represente la estructura de la CPU 8088, indicando la composición interna de las subunidades BIU y EU, y la conexión con la memoria principal de 1 Mbytes con los buses correspondientes.

### Ejercicio 2:

¿Cuál es el efecto de las siguientes instrucciones en octal?. Indique el mnemónico correspondiente. Acceda a un set de instrucciones o un simulador del 8088 para la resolución de este ejercicio

F4	BE503A
BF028B	B80100
01D8	F7D2
35AF00	F7D9
E900FF	E822FF
3400	A21300

### Ejercicio 3:

Complete el siguiente cuadro con las restantes formas de representación.

Mnemónico	hexadecimal	binario
MOV CX,DX	--	--
--	09D8	--
--	--	0111010111111010
XOR AX, [BX]	--	--
--	--	0111001111110110
--	F7D9	--
INC SI	--	--

### Ejercicio 4:

Indique la cantidad de bytes de almacenamiento y los modos de direccionamiento utilizados en cada una de las instrucciones del cuadro del Ejercicio 3.

**Ejercicio 5:**

Indique la posición de memoria a la que se accederá para leer una instrucción, si los contenidos de los registros IP y CS son 1F1A y F341, respectivamente.

**Ejercicio 6:**

Suponiendo que los contenidos de los registros DS (Segmento de Datos) y CS (Segmento de Código) son 024B y B000, respectivamente, indicar la posición física de inicio y final de los segmentos de datos y de código.

**Ejercicio 7:**

Suponiendo que los contenidos de los registros DS (Segmento de Datos) y CS (Segmento de Código) son C000 y B246, respectivamente, indicar la posición física de inicio y final de los segmentos de datos y de código. ¿Observa algún problema?.

**Ejercicio 8:**

Indicar el valor que queda almacenado en el operando destino después de ejecutar cada una de las siguientes instrucciones.

- a) ADD AX,BX
- b) ADD AX, [BX]
- c) ADD BX,0124
- d) ADD AL, [1A1B]
- e) ADD BX, [SI+1A]

Datos:

AX=1A1B

BX=147A

SI=2682

AL=1B

DS:147A=AB

DS:147B=DC

DS:1A1B=12

DS:269C=21

DS:269D=A2

**Ejercicio 9:**

Escriba en instrucciones del 8088 el siguiente código, que es parte de un programa de lenguaje de alto nivel tipo C++, considerando que todas las variables son enteras, y proponiendo posiciones de memoria para el almacenamiento de los datos y el programa.

---

# CAPÍTULO 9

i

=

---

---

## **Arquitectura Avanzada**

- 1 Visión General**
- 2 Pipeline**
  - 2.1 Introducción**
  - 2.2 Predicción de la dirección de salto**
  - 2.3 Pipeline en la máquina elemental**
- 3 Memoria Caché**
  - 3.1 Principio de localidad**
  - 3.2 Manejo de la caché**
- 4 DMA**
  - 4.1 Controlador DMA**
  - 4.2 Scanner**
- 5 Evolución de las Arquitecturas**
  - 5.1 CISC**
  - 5.2 RISC**
  - 5.3 Comparación entre RISC y CISC**
- 6 Evolución desde el Procesador 8088**
  - 6.1 Funcionamiento básico del procesador Intel 80486**
  - 6.2 Funcionamiento básico del procesador Intel Pentium**
- 7 Ejercitación**

## Capítulo 9

# Arquitectura Avanzada

## 1 Visión General

Un objetivo en el diseño de una computadora es que sea lo más rápida posible. Esto se logra haciendo más veloz el hardware, pero esto tiene sus limitaciones. Una de ellas es que la velocidad de propagación de las señales eléctricas en un conductor de cobre está en el orden de 20 cm/ns (en 1 ns recorre 20 cm). Si se pretende construir una máquina con un ciclo de instrucción de algunos nanosegundos, el bus debería tener una longitud significativamente menor de 20 cm. Esto presenta una brecha tecnológica.

Otra limitación es que a medida que aumentamos la velocidad de una máquina, también aumenta el calor generado, y el construirlas en un reducido espacio dificulta la tarea de disiparlo.

Construir máquinas rápidas y confiables es costoso. La producción en serie implica elevados costos que hacen impracticable construir máquinas rápidas basándose en tecnologías rápidas. Sin embargo, es posible aumentar la velocidad de otra manera, con el argumento de que varios gestores que trabajan simultáneamente realizan una gestión en menor tiempo que un solo gestor. Obviamente algún gestor deberá tener la tarea de organizar las acciones de los demás gestores. Además, debería preverse que cualquier gestor pueda realizar tareas de organización para el caso que el gestor encargado de ello falle. Este tipo de procesamiento se denomina PROCESAMIENTO PARALELO e implica varias CPUs trabajando coordinada y simultáneamente bajo el comando de una de ellas.

Otra forma de aumentar la velocidad de procesamiento, que definimos como,

$$V_p = \frac{\text{Instrucciones}}{\text{Unidad de tiempo}} \left[ \frac{\text{Instrucciones}}{\text{Segundo}} \right]$$

es adicionar el hardware necesario para que la CPU busque y comience a ejecutar una instrucción aun cuando no ha terminado de ejecutar la

instrucción corriente. Este mecanismo se llama PROCESAMIENTO PIPELINE.

Además de lo mencionado, a fin de aumentar la velocidad de procesamiento podríamos agregar una memoria rápida, de menor capacidad que la UM, cercana a la CPU con un bus privado, para almacenar datos de uso frecuente evitando tener que usar la UM. A esta memoria rápida se la conoce como MEMORIA CACHEÉ.

Finalmente, también se podría agregar hardware que permita realizar transferencias de datos directamente desde la Unidad de Memoria hacia los periféricos y viceversa con una intervención mínima de la CPU. De esta forma se aumentaría la velocidad de procesamiento y dejaría disponible la CPU mientras dure la transferencia. Este recurso es conocido como Acceso Directo a Memoria DMA.

## 2 Pipeline

### 2.1 Introducción

La Unidad de Control sólo puede hacer una tarea a la vez. Según se analizó previamente, las acciones que realiza secuencialmente son:

- Búsqueda, y
- Ejecución, que podemos subdividir en:
  - o Decodificación,
  - o Cálculo de Direcciones,
  - o Búsqueda de Operandos,
  - o Ejecución propiamente dicha, y
  - o Guardar el resultado

Se hace notar que es posible subdividir la Ejecución de la instrucción en tareas que tarden tiempos similares. Si se asignan Unidades Funcionales para cada tarea mencionada, y estas Unidades Funcionales operan en forma simultánea y coordinada, se podría aumentar la velocidad de procesamiento en un factor importante.

La Unidad de Control extrae una instrucción y la dirige a las unidades funcionales para su ejecución, luego la Unidad de Control procede a extraer la próxima instrucción y la envía a las unidades funcionales. Y así sucesivamente hasta que todas las unidades funcionales estén ocupadas.

Supongamos que la Unidad de Control cuenta con 5 Unidades Funcionales como se indica en la Tabla 9.1. Horizontalmente está considerado el tiempo y están indicados los CICLOS de MEMORIA con números 1, 2, 3 hasta 13. La Unidad de Control coordina las acciones a fin de suministrar a las sucesivas Unidades Funcionales las Instrucciones indicadas como: I1, I2, IS, etc. La IS indica una instrucción de Salto.

Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13
Búsqueda de la instrucción	I1	I2	IS				I4	I5	I6	I7	I8	I9	I10
Decodificación		I1	I2	IS				I4	I5	I6	I7	I8	I9
Cálculo de direcciones			I1	I2	IS				I4	I5	I6	I7	I8
Búsqueda de operandos				I1	I2	IS				I4	I5	I6	I7
Ejecución					I1	I2	IS				I4	I5	I6

Tabla 9.1. Comportamiento de las 5 Unidades Funcionales en 13 ciclos de memoria.

En la Figura 9.1 se presenta un ejemplo de Pipeline. Consiste en cinco Unidades Funcionales dentro de la Unidad de Control dedicadas a funciones específicas. En la Tabla 9.1 se observa el proceso. En el primer ciclo se procede a buscar la instrucción 1, en el segundo ciclo se decodifica la instrucción 1 y se busca la instrucción 2, y así sucesivamente

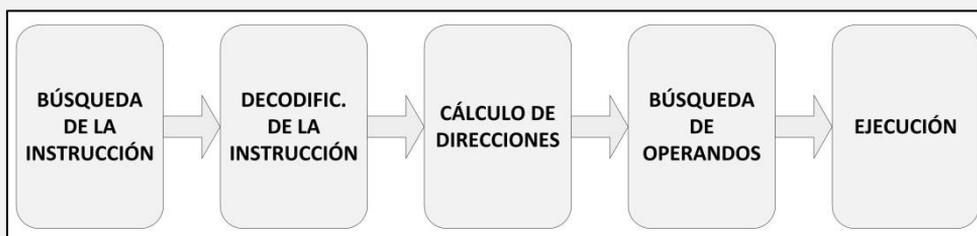


Fig. 9.1. Ejemplo de pipeline.

## 2.2 Predicción de la dirección de salto

La instrucción S representa un salto. Se observa en la Tabla 9.1 que en el ciclo 4 no se procede a buscar la próxima instrucción, ya que hasta no ejecutar la anterior no se sabe cuál es. Teniendo en cuenta que

estadísticamente se verifica que las instrucciones de salto representan aproximadamente el 30 % del programa, el incremento de velocidad que podría lograrse con el pipeline (un factor de cinco en este caso) se ve reducido en el porcentaje indicado. Vale entonces la pena estudiar los tipos de saltos posibles a fin de disminuir el porcentaje. Existen tres categorías de saltos:

- saltos incondicionales.
- saltos condicionales.
- saltos iterativos.

Una forma de disminuir el porcentaje de penalización por salto condicional es **ejecutar la próxima instrucción sin considerar el posible salto**. En este caso, si efectivamente el salto no se concreta, nada se ha perdido y se continúa el proceso normal de pipeline. Si el salto se produce, entonces deben eliminarse las instrucciones actualmente en línea y volver a comenzar. Se infiere que para que esta técnica sea posible, habrá que dotar al procesador de registros auxiliares que salven el contexto del sistema al momento del salto, de forma tal de restituirlos en el caso que el salto se produzca.

Por otro lado, esta técnica no es eficaz para el caso de los saltos incondicionales ya que estos se producen siempre.

Otra técnica utilizada es la **predicción de la dirección del salto**.

El programador raramente escribe el programa fuente en lenguaje ensamblador, es decir, con instrucciones de máquina. Generalmente lo escribe en un lenguaje de nivel superior (C, Pascal, Visual, HTML, etc). Como la CPU sólo entiende el programa escrito en lenguaje de máquina (que es el programa en lenguaje ensamblador traducido a 1s y 0s) es necesario “traducir” el programa fuente escrito en lenguaje superior a lenguaje de máquina. Este proceso se llama COMPILACIÓN y al programa que lo realiza COMPILADOR.

Existen dos clases de predicciones: las **estáticas** (al momento de la compilación) y las **dinámicas** (al momento de la ejecución).

En el primer caso el Compilador estima una dirección para cada una de las instrucciones de salto que genera. Por ejemplo, en los saltos iterativos lo más probable es que el salto se produzca al inicio de la iteración; en los incondicionales se conoce la dirección; y en los condicionales es posible suponer una probabilidad de ocurrencia de la condición, y en función de ella estimar la dirección más probable.

En el segundo caso la Unidad de Control construye una tabla de saltos y guarda el comportamiento de los mismos a fin de decidir cuál es la dirección de la próxima instrucción más probable. Esto requiere agregar hardware.

Con la utilización de estas técnicas se ha logrado reducir el porcentaje de penalización a un 10 %. Por lo tanto, en un PIPELINE como el del ejemplo, que cuente con técnicas de predicción de la dirección del salto, es posible aumentar la  $V_p$  según la siguiente expresión:

$$V_{p(\text{pipeline})} = \frac{V_p}{5} \cdot 0,9 \cdot k$$

dónde

5: es cantidad de Unidades Funcionales de la UC, y

k: factor que depende del tiempo de acceso a la memoria y el reloj de la máquina.

## 2.3 Pipeline en la máquina elemental

En los Ciclos de Máquina de la Máquina Elemental, se observó que el uso del BUS y de la UM se reparte en:

- Pulsos que usan el BUS y, eventualmente, se dan órdenes OL o OE,
- Pulsos que esperan a la UM, y
- Pulsos que dan orden puntual (por ej. TRA=1)

La idea es aprovechar la mayor cantidad de pulsos haciendo alguna acción, de forma tal que el BUS esté ocupado la mayor parte del tiempo. Esto puede lograrse reestructurando la UC en dos Unidades Funcionales (Búsqueda y Ejecución), que puedan funcionar en forma simultánea. De esta forma cuando se está esperando a la UM puede usarse el bus. Además, se puede alcanzar un uso permanente de la UM disparando ciclos de memoria permanentemente (siempre que sea posible).

### 2.3.1 Unidad de control con pipeline

La Unidad de Control Cableada consta ahora de dos Unidades Funcionales:

- Unidad Funcional de Búsqueda (UFB)
- Unidad Funcional de Ejecución (UFE)

Estas unidades pueden funcionar simultáneamente. Además, el ciclo contará ahora con sólo seis pulsos (CP1 a CP6). Esto es así porque suponemos que la UM tiene un tiempo de acceso de 5 pulsos. El sexto pulso es necesario para rescatar el dato eventualmente leído desde la UM.

Debido a que las instrucciones de la Máquina Elemental son simples y con formato fijo, la decodificación del código de operación de las mismas es muy simple. La decodificación se puede realizar cuando la instrucción buscada se carga en el Registro de Instrucciones (en el CP6).

El diagrama en bloque de la UC con PIPELINE se ve en la Figura 9.2. Las órdenes para búsqueda y ejecución se emiten sincrónicamente con los pulsos CP1 a CP6. El Secuenciador habilita a la UFB y/o a la UFE mediante las señales HB y HE.

En la Unidad de Control ya no se observará el biestable Estado. A diferencia vemos el biestable UFB, cuya salida HB se usa para habilitar la Unidad Funcional de Búsqueda. Y el biestable UFE, cuya salida HE se usa para habilitar la Unidad Funcional de Ejecución.

Para simplificar se mantiene habilitada la Unidad Funcional de Ejecución siempre que  $RUN = 1$ . Para generar CP1 a CP6 se usa un contador binario natural de 3 bits y un decodificador binario de  $3 \times 8$  (ver UC Cableada en la Guía Didáctica 5). El Ciclo de Memoria es de 6 pulsos de reloj.

En la Tabla 9.2 del PIPELINE, el tiempo transcurre de arriba hacia abajo. La columna de la izquierda es el acumulado de los pulsos requeridos para ejecutar las distintas instrucciones. Y la columna ACCIONES indica lo que está haciendo la Unidad de Control.

PREGUNTA: ¿Agregar un incrementador a la Máquina Elemental resultaría en un aumento de la velocidad de procesamiento?

### **2.3.2 Secuenciador de la UC con pipeline**

La Figura 9.3 presenta el secuenciador de la Unidad de Control con Pipeline. Cuando se arranca la Máquina (START) la UC pone  $HB = 1$  con la señal  $RUN$  y  $HE = 1$  al final del 1er ciclo.

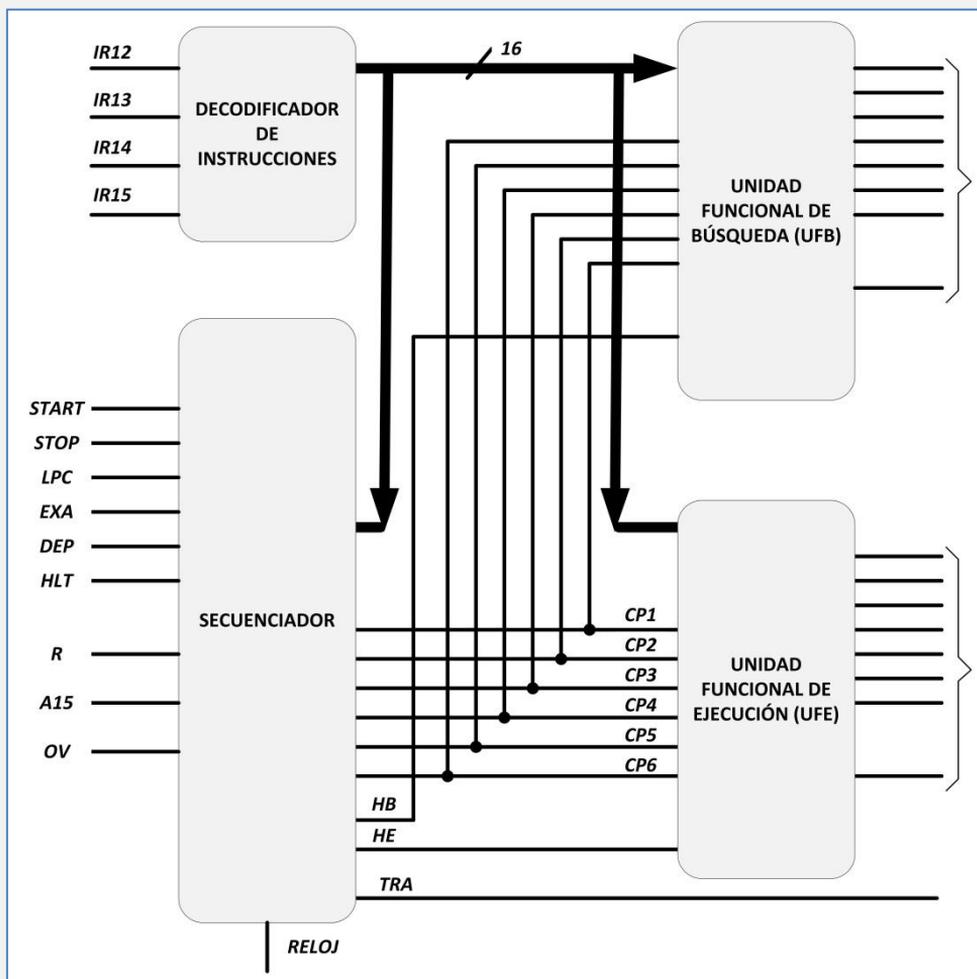


Fig. 9.2. Diagrama en bloque de la Unidad de Control.

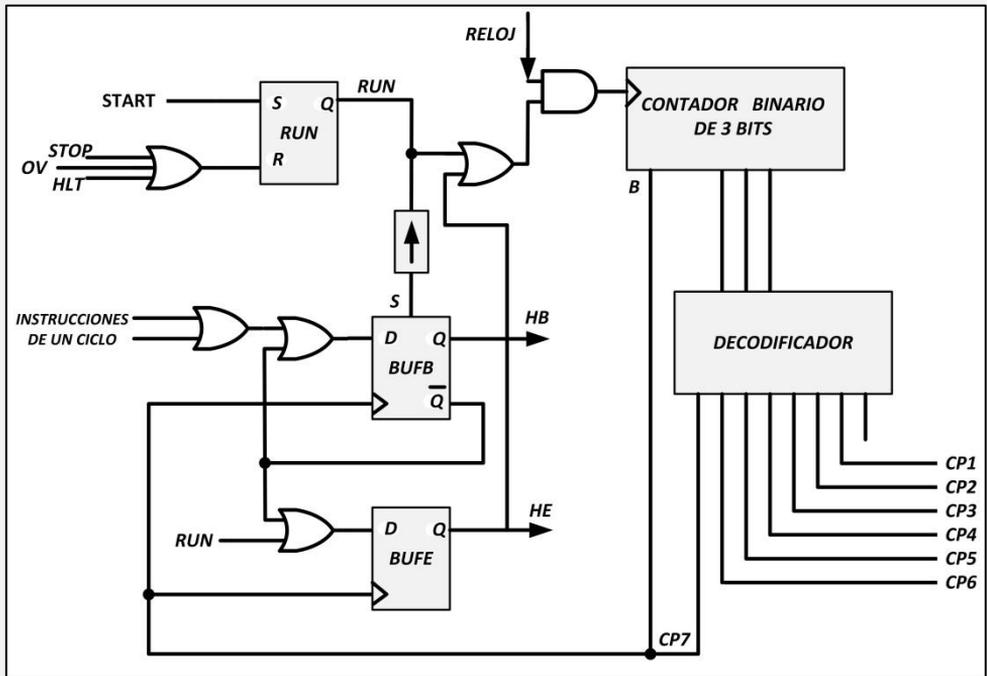


Fig. 9.3. Secuenciador de la Unidad de Control.

Cada vez que finaliza un ciclo de la Unidad Funcional de Búsqueda se define, en función de la instrucción contenida en el IR, el estado del BUFB. Si la instrucción en el IR es alguna de las siguientes:

ADD, XOR, AND, IOR, LDA, STA, INP o OUT

la UC hace BUFB = 0. Si la instrucción en el IR es cualquiera de las restantes, HB queda en 1. Cada vez que finaliza un ciclo de la Unidad Funcional de Ejecución, si BUFB = 0, se pone BUFB= 1

### 2.3.3 Ciclos de Pipeline

	Pulso de Reloj	Búsqueda	Ejecución	Acciones
1	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, PC OL		BÚSQUEDA DE ADD XXXX
2	CP2	-		
3	CP3	-		
4	CP4	Envíe PC+1, Cargue PC		
5	CP5	Envíe mem, Cargue MBR		
6	CP6	Envíe MBR, Cargue IR		
7	CP1		Envíe IR <sub>0-11</sub> , Cargue MAR OL	EJECUCIÓN DE ADD XXXX
8	CP2		Envíe ACC, Cargue Y	
9	CP3		-	
10	CP4		-	
11	CP5		Envíe Mem, Cargue MBR	
12	CP6		Envíe MBR, Cargue Z	
13	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, PC OL	-	BÚSQUEDA DE XOR XXXX
14	CP2	-	Envíe ALU(Suma), Cargue ACC	
15	CP3	-	-	
16	CP4	Envíe PC+1, Cargue PC	-	
17	CP5	Envíe Mem, Cargue MBR	-	
18	CP6	Envíe MBR, Cargue IR	-	
19	CP1		Envíe IR <sub>0-11</sub> , Cargue MAR OL	EJECUCIÓN DE XOR XXXX
20	CP2		Envíe ACC, Cargue Y	
21	CP3		-	
22	CP4		-	
23	CP5		Envíe Mem, Cargue MBR	
24	CP6		Envíe MBR, Cargue Y	
25	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, PC OL	-	BÚSQUEDA DE AND XXXX
26	CP2	-	Envíe ALU (xor), Cargue ACC	
27	CP3	-	-	
28	CP4	Envíe PC+1, Cargue PC	-	
29	CP5	Envíe Mem, Cargue MBR	-	
30	CP6	Envíe MBR, Cargue IR	-	

**a)**

	Pulso de Reloj	Búsqueda	Ejecución	Acciones
31	CP1		Envíe IR <sub>0-11</sub> , Cargue MAR OL	EJECUCIÓN DE AND XXXX
32	CP2		Envíe ACC, Cargue Y	
33	CP3		-	
34	CP4		-	
35	CP5		Envíe Mem, Cargue MBR	
36	CP6		Envíe MBR, Cargue Z	
37	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, PC OL	-	BÚSQUEDA DE IOR XXXX
38	CP2	-	Envíe ALU (and), Cargue ACC	
39	CP3	-	-	
40	CP4	Envíe PC+1, Cargue PC	-	
41	CP5	Envíe mem, Cargue MBR	-	
42	CP6	Envíe MBR, Cargue IR	-	
43	CP1		Envíe IR <sub>0-11</sub> , Cargue MAR OL	EJECUCIÓN DE IOR XXXX
44	CP2		Envíe ACC, Cargue Y	
45	CP3		-	
46	CP4		-	
47	CP5		Envíe Mem, Cargue MBR	
48	CP6		Envíe MBR, Cargue Z	
49	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, PC OL	-	BÚSQUEDA DE NOT
50	CP2	-	Envíe ALU (or), Cargue ACC	
51	CP3	-	-	
52	CP4	Envíe PC+1, Cargue PC	-	
53	CP5	Envíe mem, Cargue MBR	-	
54	CP6	Envíe MBR, Cargue IR	-	
55	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, PC OL	-	BÚSQUEDA DE LDA XXXX
56	CP2	-	Envíe ACC, Cargue Z	
57	CP3	-	Envíe ALU (not), Cargue ACC	
58	CP4	Envíe PC+1, Cargue PC	-	
59	CP5	Envíe mem, Cargue MBR	-	
60	CP6	Envíe MBR, Cargue IR	-	

**b)**

	Pulso de Reloj	Búsqueda	Ejecución	Acciones
61	CP1		Envíe IR <sub>0-11</sub> , Cargue MAR OL	EJECUCIÓN DE LDA XXXX
62	CP2		Envíe ACC, Cargue Y	
63	CP3		-	
64	CP4		-	
65	CP5		Envíe Mem, Cargue Z	
66	CP6		Envíe ALU(or), Cargue ACC	
67	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, PC OL		BÚSQUEDA DE STA XXXX
68	CP2	-		
69	CP3	-		
70	CP4	Envíe PC+1, Cargue PC		
71	CP5	Envíe mem, Cargue MBR		
72	CP6	Envíe MBR, Cargue IR		
73	CP1		Envíe ACC, Cargue MBR	EJECUCIÓN DE STA XXXX
74	CP2		Envíe IR <sub>0-11</sub> , Cargue MAR OE	
75	CP3		-	
76	CP4		-	
77	CP5		-	
78	CP6		-	
79	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, PC OL		BÚSQUEDA DE SRJ XXXX
80	CP2	-		
81	CP3	-		
82	CP4	Envíe PC+1, Cargue PC		
83	CP5	Envíe mem, Cargue MBR		
84	CP6	Envíe MBR, Cargue IR		
85	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, OL	-	BÚSQUEDA DE JMP XXXX
86	CP2	-	Envíe PC, Cargue ACC	
87	CP3	-	-	
88	CP4	Envíe PC+1, Cargue PC	-	
89	CP5	Envíe mem, Cargue MBR	-	
90	CP6	Envíe MBR, Cargue IR	-	

**c)**

	Pulso de Reloj	Búsqueda	Ejecución	Acciones
91	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, OL		BÚSQUEDA DE JMA
92	CP2	-		
93	CP3	-		
94	CP4	Envíe PC+1, Cargue PC		
95	CP5	Envíe mem, Cargue MBR		
96	CP6	Envíe MBR, Cargue IR		
97	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, OL		BÚSQUEDA DE INP YY
98	CP2	-		
99	CP3	-		
100	CP4	Envíe PC+1, Cargue PC		
101	CP5	Envíe mem, Cargue MBR		
102	CP6	Envíe MBR, Cargue IR		
103	CP1		TRA = 1	EJECUCIÓN DE INP YY
104	CP2		-	
105	CP3		-	
106	CP4		-	
107	CP5		Si R=1,TRA=0,Envíe PYY,Cargue A	
108	CP6		Si TRA = 1	
109	CP1	Si J=0 :Envíe PC, Cargue MAR OL; Si J=1 :Envíe IR <sub>0-11</sub> , Cargue MAR, OL		BÚSQUEDA DE OUT YY
110	CP2	-		
111	CP3	-		
112	CP4	Envíe PC+1, Cargue PC		
113	CP5	Envíe mem, Cargue MBR		
114	CP6	Envíe MBR, Cargue IR		
115	CP1		TRA = 1	EJECUCIÓN DE OUT YY
116	CP2		-	
117	CP3		-	
118	CP4		-	
119	CP5		Si R=1 TRA=0,Envíe PYY,Cargue A	
120	CP6		Si TRA = 1	

**d)**

Tabla 9.2. Ciclos de pipeline.

### 2.3.4 Incrementador del contador de programa

Se ha adicionado en la UC un incrementador del PC (Figura 9.4), para evitar el uso de la ALU. De esta manera se ahorran los pulsos.

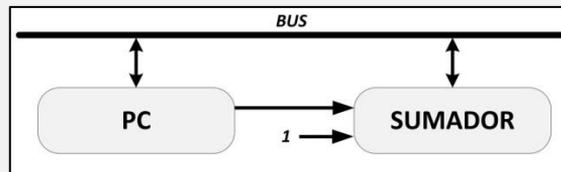


Fig. 9.4. Incrementador del PC.

### 2.3.5 Predicción de la dirección de salto

La predicción de la dirección del salto la implementamos por hardware mediante el circuito de la Figura 9.5.

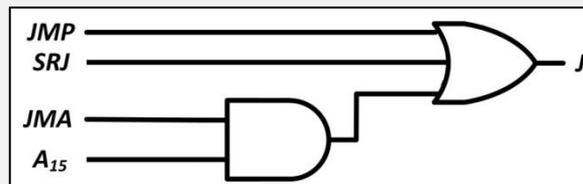


Fig. 9.5. Circuito de predicción de salto.

Si la instrucción que se ha buscado es una instrucción de salto, la próxima instrucción se buscará en la dirección que contiene el registro  $IR_{0-12}$  si  $J = 1$ , o en la dirección que indica el PC si  $J = 0$ .

Este es un ejemplo de la predicción dinámica, es decir, que se estima la dirección del salto al momento de ejecución.

### 2.3.6 Comparación con la máquina elemental

En la Tabla 9.2 se observa que para ejecutar todas las instrucciones de la Máquina Elemental con PIPELINE se necesitan 119 pulsos de reloj. Comparando con la cantidad necesaria en la Máquina Elemental sin PIPELINE (el lector puede calcular en 160 pulsos), vemos

una mejora en la velocidad de procesamiento de aproximadamente el 35 %.

## 3 Memoria Caché

### 3.1 Principios de localidad

Cuando se realiza una referencia a memoria, lo más probable es que las próximas referencias se realicen en las cercanías de la anterior. Además, buena parte del tiempo de ejecución de un programa se emplea en iteraciones entre un número limitado de instrucciones.

El **principio de localidad** es un comportamiento de la máquina por el cual se verifica que, en un lapso arbitrario, el 90% de las referencias a memoria caen en un área relativamente pequeña de la misma. Este principio representa la base de los sistemas con memoria caché.

La velocidad de la memoria principal de un sistema es menor que la CPU. Entonces, y atendiendo a lo mencionado, sería conveniente disponer de una memoria adicional (caché), que esté cerca de la CPU (controlada por ella), pequeña (para no encarecer el sistema), con un bus privado y definir una técnica apropiada para el llenado de la misma, teniendo en cuenta el principio de localidad.

La idea general del llenado de la caché es que cuando la Unidad de Control realiza un ciclo de lectura de memoria, se dispare simultáneamente un ciclo de lectura a la Caché, como se indica en la Figura 9.6.

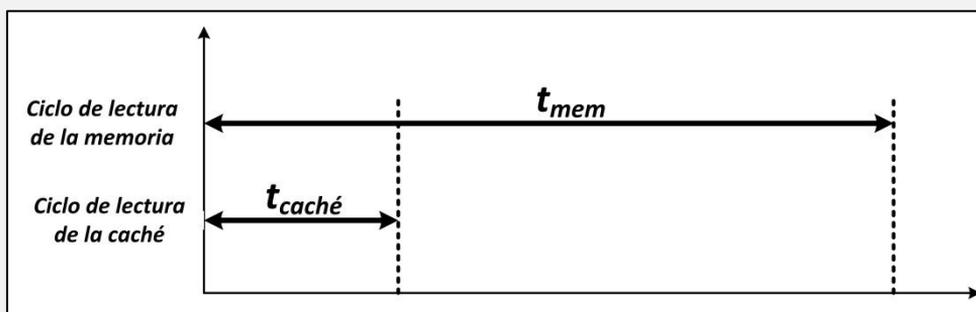


Fig. 9.6. Simultaneidad del ciclo de lectura a la memoria y a la caché.

Si al culminar el Ciclo de lectura de la Caché el dato no se encuentra en la misma, se continúa con el ciclo de lectura de la memoria, y una vez concluido, se lo escribe en la Caché. Y, además, obviamente se lo usa. Y por lo tanto, si al culminar el Ciclo de lectura de la Caché el dato se encuentra en la misma, no se continúa con el ciclo de lectura de la memoria y se usa el dato de la caché.

La mejora de velocidad puede formalizarse de la siguiente manera.

- $t_{caché}$ : tiempo de acceso de la memoria caché
- $t_{mem}$ : tiempo de acceso a la memoria principal.
- $h$ : proporción de aciertos (fracción de todas las lecturas que pueden ser satisfechas en la caché)

$$\text{Tiempo medio de acceso} = t_{caché} + t_{mem}(1 - h)$$

## 3.2 Manejo de la caché

Es necesario que el contenido de la caché y en la memoria sean *consistentes*, es decir, que coincidan. Esto es importante a la hora de mover un bloque de datos de la memoria, o bien, cuando otros programas necesiten usar datos de un área de memoria que haya sido "cacheado".

Existen dos técnicas para la *consistencia*:

- **Escritura en memoria:** Cada vez que se escribe en la caché, también se escribe en la memoria. Esta técnica asegura que coincidan en todo momento.
- **Retrocopiado:** La memoria se actualiza cuando sea necesario; por ejemplo, cuando hace falta transferir un bloque de datos desde la memoria hacia un periférico. A fin de evitar la transferencia completa de la Caché, se adiciona un bit a cada palabra de la memoria caché que indica si la misma ha sido modificada desde que se cargó desde la memoria.

## 4 DMA

### 4.1 Controlador DMA

Cuando ocurre una transferencia de Entrada/Salida en la Máquina Elemental (por ejemplo, transferir un bloque de datos desde la

memoria hacia un periférico o viceversa) se realiza a través del ACC. Si es de salida el programador deberá cargar el ACC, y luego, usar una instrucción OUT YY. En el caso de entrada será a través de una instrucción INP YY que cargará al ACC con el dato del periférico.

Una buena idea para aumentar la velocidad de la transferencia es evitar que intervenga la CPU en el proceso. Si contáramos con un “procesador auxiliar” que pueda controlar el bus y la memoria cuando se lo ceda la CPU ante una transferencia de E/S, y devuelva el control a la CPU una vez finalizada la misma, estaríamos evitando la intervención de la CPU y podríamos aumentar la velocidad. Se supone que el “procesador auxiliar” cuenta con el hardware y software especialmente diseñados para tal función. Además, la UC podría realizar algunas tareas compartiendo el BUS cuando el “procesador auxiliar” esté esperando a la UM o al Periférico. Al “procesador auxiliar” se lo llama Controlador DMA (Direct Access Memory).

El controlador DMA debe contar con una Unidad de Control capaz de comunicarse con la CPU, y controlar la UM y el BUS. Además tendrá registros para almacenar la Dirección Inicial de Memoria del Bloque a transferir, un registro para almacenar la Cantidad de Palabras a transferir y un Registro de Datos.

El esquema del soporte de hardware para un canal de datos DMA se indica en la Figura 9.7.

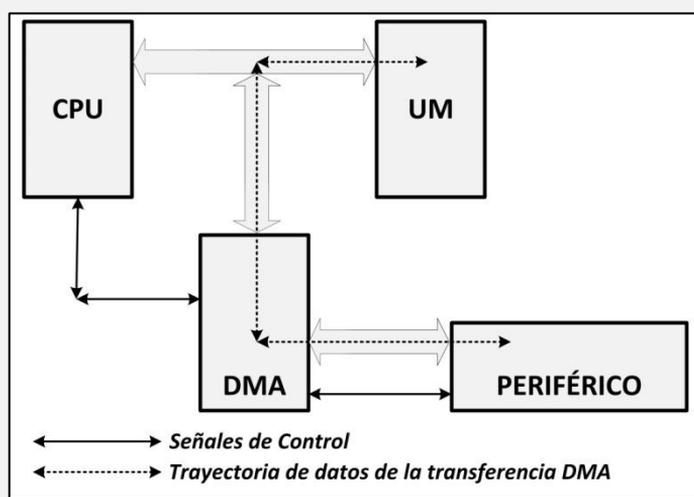


Fig. 9.7. Diagrama en bloque de un canal DMA.

Observar que la trayectoria de los datos en una transferencia DMA, no pasa por la CPU. Por esta razón la transferencia es más rápida.

El Controlador DMA se maneja mediante interrupciones. La CPU inicia una transferencia DMA enviándole al Controlador DMA la dirección inicial del bloque de datos a enviar hacia el periférico (o recibir desde el mismo), la cantidad de palabras a transmitir y cediéndole el control del BUS. Cuando el Controlador DMA culmina la transferencia, solicita INTERRUPCIÓN y la transferencia termina recuperando la CPU el control del BUS.

## **4.2 Scanner**

Según lo explicado, el controlador DMA es casi una computadora independiente con su propio programa fijo (almacenado en su propia memoria ROM). Debe ser capaz de generar ciclos de memoria proporcionando la dirección de memoria y las órdenes de lectura y escritura según sea el caso. Además, debe ser capaz de coordinar sus acciones con la CPU y peticionar la atención de la UM.

Si bien la CPU puede realizar algunas tareas, seguramente no podrá usar la UM durante la transferencia DMA. El BUS está ocioso mientras se está esperando que el dato se escriba o lea desde la UM. En esos lapsos la CPU puede usar el BUS sin conflictos. Sin embargo, si la CPU necesita acceder a la UM, no podrá hacerlo mientras dure la transferencia.

Esto representa un problema en el caso de transferencias muy extensas, que efectivamente, pueden demorar inaceptablemente la ejecución de un programa. En muchos casos es conveniente seguir ejecutando el programa, aunque sea lentamente, mientras está en proceso una transferencia DMA. La implementación de un sistema que permita realizarlo recibe el nombre de ROBO DE CICLO.

### **4.2.1 Robo de ciclo**

Hasta ahora, la UM solo podía ser requerida por la CPU. En Máquinas con DMA la UM también puede ser requerida por el controlador DMA (o los controladores DMA en caso de ser más de uno). Una técnica utilizada para lograr un mejor aprovechamiento de la UM, apuntando a aumentar la velocidad de procesamiento, es agregar hardware que administre la UM.

Este hardware administrador llamado “scanner” funciona como una llave selectora que gira constantemente en busca de algún procesador que necesite la UM. Supongamos que la UM tiene que ser compartida por la CPU y un DMA, el Scanner sería un dispositivo que contaría con dos Puertos, el Scanner examinaría el primer puerto, si hay algún requerimiento de servicio, el Scanner se detiene y atiende el requerimiento. Si no hay requerimiento o después de haber atendido uno anterior, el Scanner avanza al otro puerto y así sucesivamente.

La velocidad de avance de un puerto a otro es menor al tiempo de un ciclo de memoria. Existen variaciones al esquema planteado que permiten asignar prioridades a ciertos procesadores con respecto a otros a fin de asegurar el recurso a periféricos más rápidos.

## 5 Evolución de las Arquitecturas

### 5.1 Introducción

Con el invento de la microprogramación de Wilkes, al principio de los cincuenta, la idea de hacer más y más complejo el microcódigo fue poco menos que irresistible. Las razones eran de validez. Por aquellos años las memorias eran significativamente más lentas que la CPU, de forma tal que la posibilidad de poner las bibliotecas de las aplicaciones de uso frecuente como residentes en el microcódigo (en una ROM rápida) en lugar de la memoria principal representaba una solución ideal. Por otro lado, elevar el nivel del lenguaje de máquina incorporando instrucciones más complejas, parecía no tener discusión. Además, la microprogramación permitía modificar y/o agregar instrucciones cambiando el microprograma (lo que significa simplemente cambiar una memoria ROM).

Ya que la microprogramación no tenía discusión, a lo largo de los años esta arquitectura siguió un proceso de perfeccionamiento. La micromemoria es costosa (recordemos que debe ser rápida) lo que la limita en su dimensión. Una micromemoria ancha (como la de 45 bits de la máquina elemental microprogramada) se llama **horizontal**, lo que implica pocas microinstrucciones para ejecutar una instrucción, es decir, una máquina más rápida. Como contrapartida, se puede pensar en una micromemoria angosta (microprogramación **vertical**), con más microinstrucciones por instrucción, pero menor costo de la micromemoria.

Puede verse en la UC microprogramada que existen secuencias de microinstrucciones comunes a más de una instrucción. Esto dió la idea

de escribir microrutinas (usables por muchas instrucciones) que podrían residir en una microROM angosta y combinarla con una microprogramación horizontal.

Existen alternativas siempre con el compromiso de costo-performance. Esta arquitectura de máquinas se llama **CISC** (Complex Instruction Set Computer).

## 5.2 CISC

A partir de los computadores 360 y 370 de IBM, surgidos en la década del 70, la mayoría de los procesadores (CPU) de las computadoras, incluidos los de minicomputadoras y PC personales (Pentium y el P6 de Intel, y los 680x0 de Motorola han sido CISC (Complex Instruction Set Computer).

Esta denominación se debe a que pueden ejecutar desde instrucciones muy simples (como las que ordenan sumar, restar dos números que están en registros de la CPU y el resultado asignarlo a uno de esos registros), hasta instrucciones muy complejas (como los tan usados movimientos de cadenas de caracteres de gran longitud variable, en procesamiento de textos).

Las instrucciones simples, luego de su decodificación, pueden ejecutarse en un pulso reloj, mientras que las complejas requieren un número de pulsos que depende de la secuencia de pasos necesarios para su ejecución.

Como se describió, cada paso se lleva a cabo mediante combinaciones binarias (*microinstrucciones*), que aparecen en las líneas de control de la UC con cada pulso de reloj, el cual activa los circuitos que intervienen en ese paso.

Las sucesivas microinstrucciones que requiera la ejecución de una instrucción compleja, deben ser provistas por la ROM (**firmware**) de Control que las almacena, y que forma parte de la UC.

Por lo tanto, **un procesador CISC necesariamente debe contener una ROM** con las microinstrucciones para poder ejecutar las instrucciones complejas. Esta es una de las características CISC.

En general, cada operación que ordena una instrucción de un procesador CISC presenta variantes para ser aplicadas a diversas estructuras de datos, desde simples constantes y variables, hasta matrices y otras. Así, una instrucción que ordena sumar, tiene muchas variantes (códigos) en función de la estructura de datos sobre la cual opera. Es como si existieran tantas instrucciones que ordenan una misma operación como estructuras de datos típicas se han definido para operar. Este concepto fue planteado previamente bajo el nombre de "**modos de direccionamiento**" de una instrucción.

La existencia de muchos "**modos de direccionamiento**" para realizar la operación que ordena una instrucción, es otra de las características de complejidad de los procesadores CISC. Esto se manifiesta en que el repertorio ("set") de instrucciones de una máquina CISC presente un **número elevado de códigos de operación**. Así, una IBM 370 tiene 210 instrucciones, 300 la VAX, y 230 el 80486. Asimismo, lo anterior exige instrucciones que ocupan **distinta** cantidad de bytes en memoria. Resulta así, que por tener instrucciones ejecutables en diferente cantidad de pulsos de reloj, un procesador CISC no puede aprovechar eficazmente su "pipeline" en la producción de instrucciones.

## 5.3 RISC

Ya para los setenta, la velocidad de las memorias se acercó a la de la CPU y resultaba difícil escribir, depurar y mantener los microprogramas. Además, algunos especialistas comenzaron a analizar qué tipo de instrucciones eran las más usadas en los programas. Los resultados fueron sorprendentes:

- El 85% de las instrucciones son de asignación, condicionales y de llamadas a procedimientos,
- El 80% de las instrucciones de asignación son de un solo término,
- El 41% de los procedimientos no tienen argumentos, y
- El 80% de los procedimientos tienen 4 o menos variables locales.

Como conclusión, podemos decir que si bien teóricamente es posible escribir programas complicados, la mayoría de los programas reales consisten en simples asignaciones, declaraciones condicionales y llamadas a procedimientos con un número reducido de parámetros.

Esta conclusión es de extrema importancia en la tendencia a agregar más y más funciones al microcódigo. Mientras que el lenguaje de máquina se hace más complicado, el microprograma se hace más grande y lento. Un número elevado de modos de direccionamiento significa que

su decodificación no puede realizarse en línea (lo que implicaría repetir cientos de veces el mismo microcódigo). Peor aún, se ha sacrificado velocidad a fin de incorporar instrucciones que en la práctica rara vez se usan.

Se podría afirmar que una buena idea sería eliminar el microcódigo y que los programas se corran directamente por el hardware residiendo en una rápida memoria principal. Surgen así las computadoras con un número reducido de instrucciones, llamadas máquinas RISC (Reduced Instruction set Computer). Antes de Wilkes todas las máquinas eran RISC, y luego con la microprogramación las computadoras se hicieron más complejas y menos eficientes. Ahora, la industria está volviendo a sus raíces, construyendo máquinas sencillas y rápidas.

La denominación RISC no está del todo bien aplicada. Si bien es cierto que tienen pocas instrucciones, la característica más importante es que éstas se completan en un sólo ciclo usando pipeline intensivamente (entendiendo por ciclo la extracción de los operandos de un registro, colocarlos en el bus, ejecutarlos a través de la ALU y guardar el resultado en un registro).

Cualquier operación que no se lleve a cabo en un ciclo, no puede formar parte del conjunto de instrucciones.

### **5.3.1 Arquitectura de carga/almacenamiento**

Dado que cada instrucción debe ejecutarse en un ciclo, resulta claro que aquellas que hacen referencia a memoria representan un problema.

Las instrucciones ordinarias sólo pueden tener operandos en registros (sólo está permitido el direccionamiento por registros). Las únicas instrucciones que hacen referencia a memoria son LOAD y STORE. Para lograr que éstas se ejecuten en un ciclo se recurre a exigir que cada instrucción se inicie en cada ciclo, sin importar cuando termine. Si se logra, entonces, comenzar  $n$  instrucciones en  $n$  ciclos, se habrá alcanzado un promedio de una instrucción por ciclo.

Todas las RISC poseen procesamiento en línea (pipeline). Por ejemplo, una RISC con tres unidades funcionales tendría el aspecto de la Tabla 9.5. En el ejemplo, se puede apreciar que las instrucciones ordinarias se ejecutan en dos ciclos. La instrucción LOAD (indicada con una L) se ejecuta en tres ciclos. Se observa que la instrucción 4 termina de ejecutarse antes que termine de hacerlo L, que es la anterior. Lo

mismo ocurre con la instrucción 8 y la S. En estos casos, el Compilador verifica si la instrucción 4 es afectada por la instrucción L. Si no lo es, el proceso continúa sin problemas. En el caso contrario, algo debe hacerse. Por ejemplo, el Compilador puede reemplazar a la instrucción 4 por una NOP, lo que implica una degradación de la velocidad. En el caso de las instrucciones de salto (JUMP) se producirá un problema similar y la solución es la misma. La instrucción que sigue a una de salto siempre comienza a ejecutarse independientemente si el salto se lleva a cabo. En todos los casos el Compilador es el responsable de colocar una instrucción útil después de una de salto. En caso de no encontrar nada adecuado, se coloca una instrucción NOP.

Ciclo	1	2	3	4	5	6	7	8	9	10
Extracción de Instrucciones	1	2	L	4	5	6	S	8	9	10
Ejecución de Instrucciones		1	2	L	4	5	6	S	8	9
Referencia a Memoria					L				S	

Tabla 9.3. Procesador RISC con 3 etapas.

Las instrucciones generadas por el Compilador son ejecutadas directamente por el hardware, es decir, no son interpretadas por el microcódigo. Esta es la razón de la velocidad de las RISC. La complejidad que soporta el microcódigo en las CISC, se traslada al código de usuario en las RISC. Si una instrucción compleja implica  $n$  microinstrucciones en una CISC, en una RISC implicaría un número similar de instrucciones, pero ocuparía más memoria. Sin embargo, también debe tenerse en cuenta que las instrucciones complejas representan un porcentaje menor en un programa real. Se recomienda comparar los tiempos de ciclo de instrucción de la máquina elemental con la máquina elemental microprogramada.

La razón del reducido número de instrucciones responde a la idea de simplificar el decodificador de instrucciones. En cuanto a los modos de direccionamiento conviene reducirlos al mínimo. En el formato de una instrucción RISC (Figura 9.8), puede apreciarse que es posible generar distintos modos de direccionamiento.

7	1	5	5	1	13
CÓD. OPER.	C	DESTINO	ORIGEN	I	DESPLAZAM.
COD.OPER.:	Código de operación de la instrucción				
C:	Activa o no los códigos de condición				
DESTINO:	Registro destino de la operación				
ORIGEN:	Registro origen de la operación				
I:	El campo de desplazamiento es tal o un registro				
DESPLAZAMIENTO:	Valor de desplazamiento o un registro				

Fig. 9.8. Formato típico de una instrucción.

El primer operando de cualquier operación se toma del registro origen.

Para instrucciones ordinarias como ADD, los operandos dependen de cuánto vale I. Si  $I = 0$ , el segundo operando lo especifican los últimos cinco bits del campo desplazamiento (representa un direccionamiento por registro). Si  $I = 1$ , el segundo operando es el campo desplazamiento (representa un direccionamiento inmediato).

Para la instrucciones LOAD y STORE, el desplazamiento es sumado al registro origen para obtener la dirección de memoria (representa un direccionamiento indexado). Si el desplazamiento es cero representa un direccionamiento indirecto por registro.

### 5.3.2 Registros múltiples

A fin de reducir el número de cargas y almacenamientos (LOAD Y STORE), una buena parte del chip de una RISC se usa para registros (no es raro encontrar RISC con 500 registros), aprovechando que carece de firmware. La organización de estos registros es un aspecto muy importante en las RISC.

El hecho que, como se mencionó anteriormente, una parte importante del tráfico a memoria es consecuencia de los llamados a procedimientos (que implica transmitir parámetros, salvar registros, etc.), dio lugar a los diseñadores de las RISC a plantear una organización de registros llamada **traslape de registros**.

El traslape de registros consiste, en general, que en un momento dado la CPU accede a sólo un subgrupo de ellos (por lo general de 32 bits). divididos en 4 grupos de 8 cada uno (ver Figura 9.9). Los primeros

ocho registros se encuentran en todo momento accesibles a la CPU y representan los registros globales utilizables por cualquier procedimiento. En cambio desde el R8 en adelante, el grupo de 24 registros serán accesibles por la CPU dependiendo del valor de un puntero de registros. Este puntero se ajusta cuando algún procedimiento es invocado desde el actual. Esta organización permite intercambiar valores entre procedimientos sin necesidad de referir a la memoria.

R0 . . R7	<b>Registro origen con valor cero</b> VARIABLES GLOBALES
R8 . . R15	<b>PARÁMETROS DE ENTRADA</b>
R16 . . R23	<b>VARIABLES LOCALES</b>
R24 . . R31	<b>PARÁMETROS DE SALIDA</b>

Fig. 9.9. Organización de registros usados para el traslape.

## 5.4 Comparación entre RISC y CISC

A partir de los conceptos introducidos en los apartados anteriores, es posible establecer una comparación de las arquitecturas RISC y CISC, considerando las características particulares de cada una.

### 5.4.1 Desde la semántica de los programas de alto nivel

Buscando optimizar la performance de los procesadores, se realizaron estadísticas de las instrucciones de máquina más usadas. Resultó que las instrucciones *más simples* -que sólo son el 20% del repertorio de instrucciones de un procesador CISC- constituían el 80% de programas típicos ejecutados. El 91% de las sentencias más usadas en lenguajes de alto nivel (Fortran, Pascal, Basic, C, etc.) son las del tipo *Asignar* (un valor a una variable), *IF* (condicional), *Call* llamar a

procedimiento), "Loop" (repetir una secuencia), que en promedio constituyen el 47%, 23%, 15%, y 6%, respectivamente.

En la concepción CISC se busca una menor disparidad entre los lenguajes de alto nivel y el lenguaje de máquina lo que se da en llamar "salto semántico". Recordar al respecto, que una sentencia como  $Z = P + P - Q$  se debe traducir a una secuencia de instrucciones de máquina como I1, I2, I3, I4.

Suponiendo que en un cierto lenguaje de alto nivel dicha sentencia (u otra más común) se usara frecuentemente, se podría tener un CISC que hiciera corresponder a esa sentencia una sola instrucción Ix de máquina, que reemplazara a las 4 instrucciones citadas. Esto se conseguiría escribiendo en la ROM de Control una extensa secuencia de microcódigos para poder ejecutar Ix.

De existir muchas equivalencias entre sentencias en alto nivel e instrucciones de máquina, el programa traductor entre ambos niveles (compilador) sería más sencillo de fabricar, y los tiempos de compilación disminuirían, objetivos de las arquitecturas CISC.

Se comprende que esta concepción puede llegar al extremo de fabricar un CISC con instrucciones de máquina que sean equivalentes a sentencias muy usadas en un cierto lenguaje de alto nivel, pero que no se usarían si se programa en otro lenguaje de alto nivel que no las utiliza.

La información anterior sirvió para planificar procesadores con un **repertorio de instrucciones simples** (operar dos números que están en registros de la CPU, y el resultado asignarlo a uno de esos registros) que presentan muy pocos modos de direccionamiento. Por ser simples, estas instrucciones se ejecutan en un solo pulso reloj, luego de haber sido decodificada.

A fin de poder traducir un lenguaje de alto nivel a este tipo de instrucciones, empleando un mínimo de ellas, se requiere para RISC un programa **compilador inteligente**, muy elaborado. O sea que es necesario un compilador (software) complejo, como contrapartida de un hardware más simple.

#### 5.4.2 Considerando la transferencia de datos entre la CPU y la memoria

Para mover datos de memoria a registro, y en sentido contrario, fue necesario la existencia de instrucciones que ordenen esos

movimientos, que en el lenguaje assembler de un RISC se denominan LOAD y STORE, respectivamente. Estas instrucciones se trata de usarlas lo menos posible (a través de programas compiladores "inteligentes"), puesto que requieren dos pulsos para ser ejecutadas, luego de que fueron decodificadas.

A diferencia, en un CISC cualquier instrucción tiene la opción de requerir un dato de memoria.

El número total de instrucciones del repertorio de un procesador RISC es reducido, entre 70 y 150 instrucciones según el modelo. Puesto que la mayoría de las instrucciones RISC se ejecutan en un pulso reloj, resulta un "pipeline" eficaz, terminándose de ejecutar en promedio, una instrucción por pulso reloj.

En promedio pues las instrucciones tipo LOAD y STORE requieren dos pulsos luego de ser decodificadas. Todas las instrucciones RISC son de formato fijo, por ejemplo de 4 bytes, siendo que en un CISC ocupan distinta cantidad de bytes. Esto redundante en una mayor sencillez y velocidad de procesamiento.

Por constar de instrucciones cuya fase de ejecución demanda mayormente un pulso, o a lo sumo dos, no se requiere una ROM de Control para generar el microcódigo que debe aparecer en las salidas de la UC con cada pulso reloj para comandar el procesador. Esto es, los bits del código de operación de cada instrucción que llega al registro de instrucción RI para ser ejecutado por un procesador RISC, sirven de base para que un circuito de la UC los convierta directamente en la combinación de bits que deben aparecer en las salidas de la UC (microcódigo), para que en próximo pulso reloj se ejecute la instrucción en cuestión.

Por lo tanto, la UC de un RISC **no contiene ROM de control (de microcódigos)**. Esto, por un lado, permite ganar en velocidad, pues se evita el acceso a una ROM. Por otro lado, beneficia mucho el diseño del chip que contiene un procesador RISC, dado que la superficie que ocupa una ROM de Control de un CISC, en un RISC es aprovechada para aumentar -como ser a 32- el número de registros de uso general de la CPU.

Un mayor número de registros permite utilizar menos instrucciones LOAD STORE, lo cual redundante en menos accesos a memoria principal. Por otro lado, el hecho de que la mayoría de las instrucciones sean de igual complejidad trae aparejado un mejor rendimiento del "pipeline". Aún, aunque se mantenga la dependencia del

resultado de una instrucción para poder ejecutar la siguiente, cuando un mismo recurso (por ejemplo un registro de la CPU) es requerido por varias etapas de un "pipeline". Este factor influye menos en un RISC que en un CISC, resultando así una mayor velocidad de procesamiento.

Por ejemplo, si el resultado de una instrucción es el dato de la siguiente, puesto que la mayoría requiere dos pulsos (fases) para ejecutarse, cuando una instrucción en el "pipeline" pasa a la fase de ejecución, las anteriores que entraron al "pipeline" ya completaron dicha fase. Sólo existe una espera, cuando el dato que opera una instrucción se cargó en un registro desde memoria en la instrucción anterior. Se trata de un "pipeline" al cual llegan instrucciones de máquina planificadas por un compilador inteligente.

### **5.4.3 Comportamiento en el salto a subrutinas e interrupciones**

Los procesadores CISC pierden mucho tiempo en las instrucciones de llamado a subrutina y en las interrupciones, dado los consiguientes accesos a la pila de memoria principal que requieren. Las estadísticas indican que en el llamado a procedimientos, el 98% de las veces se utilizan menos de 6 argumentos, y el 92% de las veces menos de 6 variables locales.

Asimismo, sólo en el 1% de los casos se llega a 8 llamadas sucesivas o niveles de anidamiento en el que un procedimiento llame a otro, este a un tercero, el tercero a un cuarto, etc..

Dado que los RISC poseen un número elevado de registros, éstos pueden usarse para el manejo de llamados, en lugar de perder tiempo para escribir y leer la pila ubicada en memoria.

### **5.4.4 Cuadro comparativo entre RISC y CISC**

Los aspectos discutidos previamente pueden resumirse en una tabla comparativa como la que se observa en la Tabla 9.4.

	RISC	CISC
1	Instrucciones sencillas en un ciclo	Instrucciones complejas en varios ciclos
2	Sólo LOAD/STORE hacen referencia a memoria	Cualquier instrucción puede referencia memoria
3	Procesamiento serie de varias etapas	Poco procesamiento en serie
4	Instrucciones ejecutadas por hardware	Instrucciones interpretadas por el microprograma
5	Instrucciones de formato fijo	Instrucciones de formato variable
6	Pocas instrucciones y modos de direccionamiento	Muchas instrucciones y modos de direccionamiento
7	La complejidad está en el compilador	La complejidad está en el microprograma
8	Varios conjuntos de registros	Un sólo conjunto de registros
9	Mayor uso del Bus en Búsqueda	Menor uso del Bus en Búsqueda
10	Menor uso del Bus en Ejecución	Mayor uso del Bus en Ejecución

Tabla 9.4. Comparación de las características RISC y CISC.

Una comparación entre RISC y CISC para determinar cuál es mejor, entendiendo por mejor a la más rápida, es en extremo compleja. Varios factores deben tenerse en cuenta.

Aunque podríamos orientarnos por medio de algunas preguntas:

- ¿Qué tipo de programas corren en la máquina? Programas con pocas llamadas a procedimientos y muchos saltos se corren mejor en una CISC. Por otro lado, programas cortos y recursivos corren mejor en las RISC.
- ¿Qué tipo de compilador se usó? Es entendible que un buen compilador implica programas más rápidos.
- ¿Deberían usarse programas en punto flotante? Hemos visto que las RISC, de no contar con hardware adicional, no son buenas para este tipo de cálculo.
- ¿Deben tenerse en cuenta los recursos del sistema además de la CPU, como unidades de entrada/salida, sistema operativo, etc.?
- ¿Qué tecnología se usa para construir la CPU? Las máquinas que podríamos comparar usan distintas tecnologías, distintos tiempos de reloj, distintos buses, etc.. ¿Cómo tener en cuenta estas cosas?
- ¿Debe considerarse la cantidad de memoria utilizada? En este sentido, las RISC usan más memoria que las CISC, teniendo en

cuenta que las memorias son cada vez más baratas ¿es de considerar este aspecto?

- ¿Debe medirse el tráfico de memoria? Es decir, una máquina que realice un mismo programa con menos referencias a memoria es mejor que otra, aun cuando tarde tiempos similares.

En general los cálculos de desempeño favorecen a las máquinas RISC. Pero esta conclusión no debe tomarse para desestimar la arquitectura CISC. Simplemente significa que las máquinas RISC se desempeñan mejor que las CISC ya construidas y que, en su diseño tuvieron que respetar la compatibilidad con procesadores anteriores (por ejemplo si el 80486 no hubiera tenido que ser compatible con el 8088, seguramente su arquitectura CISC habría sido diferente).

## 6 Evolución desde el Procesador 8088

La evolución de la tecnología ha permitido que cada vez se construyan procesadores más veloces y con mayor rendimiento.

Los principales ítems que se han adicionado y/o mejorado son:

- mayor capacidad de memoria,
- el aumento del tamaño del dato a 32 bits (y hasta 64 bits actualmente),
- el uso de frecuencias del reloj que superan los 3.000 Mhz,
- el «pipeline»,
- la obtención anticipada de las próximas instrucciones a ejecutar,
- la memoria "caché",
- el mayor número de registros de la CPU, y
- operación multitarea (multitasking).

Al mismo tiempo, estas mejoras han permitido que los nuevos procesadores de una misma familia puedan ejecutar las instrucciones de modelos anteriores, sin tener que cambiar el software desarrollado para éstos, logrando una necesaria compatibilidad.

En relación a las mejoras citadas, por ejemplo Intel las ha incorporado en sus procesadores como se observa en la Tabla 9.5.

El **8086** utiliza un tipo de "pipeline", y realiza la obtención anticipada de próximas instrucciones a ejecutar.

El procesador **80286** aumenta su frecuencia de reloj hasta 25 MHz, presenta la opción de un coprocesador matemático externo opcional (80287), y puede realizar "**multitasking**" en modo protegido si usa un

sistema operativo preparado para ello. Opera en memoria y en la ALU con 16 bits a la vez. Con sus 24 líneas de dirección puede acceder a  $2^{24} = 16$  MB de memoria principal. Opera hasta 25 Mhz.

El **80386** además de perfeccionar las innovaciones del 80286, tiene también como opcional externo una memoria caché. Maneja 32 bits a la vez, por lo cual sus registros internos son también de 32 bits. Las líneas de dirección son 32, pudiendo así direccionar  $2^{32} = 4$  GB de memoria. Opera hasta 40 MHz.

	8086	80286	80386	80486	80586 (Pentium)
Memoria en Mbytes	<b>1</b>	<b>16</b>	<b>4000</b>	<b>4000</b>	<b>4000</b>
Bus direcciones	<b>20</b>	<b>24</b>	<b>32</b>	<b>32</b>	<b>32</b>
Tamaño del dato	<b>16</b>	<b>16</b>	<b>32</b>	<b>32</b>	<b>32</b>
Frecuencia de reloj		<b>25</b>	<b>40</b>	<b>100</b>	<b>200</b>
Pipeline					<b>Doble</b>
Memoria caché	<b>No</b>	<b>No</b>	<b>externa</b>	<b>8kb + externa</b>	
Registros	<b>14 (16 bits)</b>	<b>19 (16 bits)</b>	<b>28 (32 bits)</b>	<b>28 (32 bits)</b>	<b>28 (32 bits)</b>
Bus de datos	<b>16</b>	<b>16</b>	<b>32</b>	<b>32</b>	<b>64</b>
Otras mejoras		<b>Multitasking</b>			<b>2 ALUs</b>

Tabla 9.5. Mejoras de los procesadores Intel desde el 8086 al 80586.

El procesador **80486DX** opera con datos de 32 bits, y 32 líneas de dirección, como el 80386. Presenta un "pipeline" más elaborado, y en su interior se tiene un coprocesador matemático y un caché de 8 KB. De éste se obtienen en forma simultánea, en promedio, las 5 próximas instrucciones a ejecutar. Como en los procesadores RISC, muchas de las instrucciones (incluidas las usadas para resguardar datos en la pila) del 486 se ejecutan en un solo pulso reloj, o sea, utilizan un solo microcódigo de la ROM de Control.

El **Pentium** opera internamente con 32 bits, y se comunica con el exterior a través de 64 líneas de datos y 32 de dirección. Contiene dos "pipeline" con dos ALU, por lo cual puede ejecutar simultáneamente dos instrucciones en un pulso reloj, si ambas son simples, por lo cual es un

procesador "super escalar" (el modelo de Von Neumann es "escalar") con muchas concepciones RISC y "predicción de saltos condicionales". En punto flotante es cuatro veces más rápido que el 486.

El **P6**, así designado por Intel, opera interna y externamente con igual número de bits que el Pentium. Presenta 3 "pipeline", con la posibilidad de ejecutar 3 instrucciones simples por pulsos reloj. Permite ejecutar las instrucciones fuera del orden establecido por el programa, las que luego son reordenadas. Su hardware convierte las instrucciones 80x86 en operaciones simples, tipo RISC. Incorpora 8 nuevos registros de 32 bits, en relación con los registros clásicos de los 80x86. A 150 MHz puede llegar a ser hasta un 50% más rápido que el Pentium sólo si el computador tiene un sistema operativo totalmente de 32 bits (como el OS/2 o el Windows NT; no así el Windows 95). El chip del P6, de 5.5 millones de transistores, viene adosado con otro chip que contiene un caché externo o de nivel 2 ("level 2"-L2) de 256 o 512 KB (con 15.5 y 31 millones de transistores, respectivamente). Como en el Pentium, el caché interno (L1) está separado en uno de 8 KB para instrucciones y otro de 8 KB para datos. Otra característica del P6 es que puede conectarse directamente a otros 3 procesadores P6 para multiprocesamiento.

Los requerimientos actuales de velocidad de procesamiento hicieron necesario el desarrollo de máquinas designadas "no Von Neumann", en el sentido de que existen varios procesadores operando juntos, en paralelo, de modo de poder ejecutar, en forma independiente, varias instrucciones de un mismo programa, o varios programas independientes, u operar con diversos datos a un mismo, tiempo.

Esto se conoce como "**multiprocesamiento**", contrapuesto al "**uniprocesamiento**" de Von Neumann. En las arquitecturas "no Von Neumann", varias CPU pueden terminar de ejecutar juntas varias instrucciones por pulso reloj.

No debe confundirse multiprocesamiento con "**multiprogramación**". **Multitasking**" (también traducible como "**multitarea**"), consiste en la ejecución alternada por una CPU de varios programas que están en memoria principal. Dada la velocidad de procesamiento, **puede parecerle al usuario como simultánea la ejecución de dos o más programas cuya ejecución en realidad se alterna muy rápidamente.**

## 6.1 Funcionamiento básico del procesador Intel 80486

A continuación describiremos los principales bloques que están en el interior de un procesador 486, y las funciones que cumplen.

En la Figura 9.10 aparecen los siguientes sub-bloques y bloques:

- Los registros de direcciones (RDI) y de datos (RDA) pertenecen a la **"Unidad de Interconexión con el Bus"** (*BIU* en inglés), encargada de la comunicación con el exterior a través de las 32 líneas de datos y 32 líneas de direcciones del bus. Las instrucciones y datos leídos en memoria pasan al caché interno de 8 KB del procesador.
- La **Unidad de caché** de 8 KB guarda las instrucciones y datos que seguramente serán requeridos próximamente. Por una parte, a través de un bus de 128 líneas, se pueden leer del caché 16 bytes que pasan a un buffer de la Unidad de pre-carga de instrucciones, correspondientes en promedio a unas 5 instrucciones a ejecutar, que así llegan juntas para entrar al "pipeline". Por otra, el caché puede ser leído para que se envíen 32 bits de datos a la ALU, o a un registro de la CPU o 64 bits de datos a la Unidad de Punto Flotante (*FPU* en inglés). En una escritura van hacia el caché 32 o 64 bits, respectivamente
- La **Unidad de Pre-carga** proporciona las direcciones de las próximas instrucciones a ejecutar y guarda las mismas en orden en dos buffers de 16 bytes, para que luego cada una sea decodificada.
- La **Unidad de Decodificación** realiza dos decodificaciones de cada instrucción.
- La **Unidad de Control** (*UC*) mediante líneas que salen de ella, activa las operaciones que con cada pulso reloj deben realizar los distintos bloques de la CPU, conforme lo establecen microcódigos de la ROM de Control.
- La **Unidad de segmentación, paginación y protección de memoria**, conocida como "Unidad de manejo de memoria (*MMU* en inglés) se encarga de proporcionar las direcciones físicas de memoria que utiliza un programa. Para tal fin, esta unidad convierte la referencia a la dirección del dato - que viene con la instrucción - en la correspondiente dirección física. Puesto que la memoria de una PC se divide en segmentos, y éstos - de ser necesario - pueden subdividirse en páginas; esta unidad se encarga de ello, así como de la protección contra escrituras no permitidas en zonas reservadas de memoria.

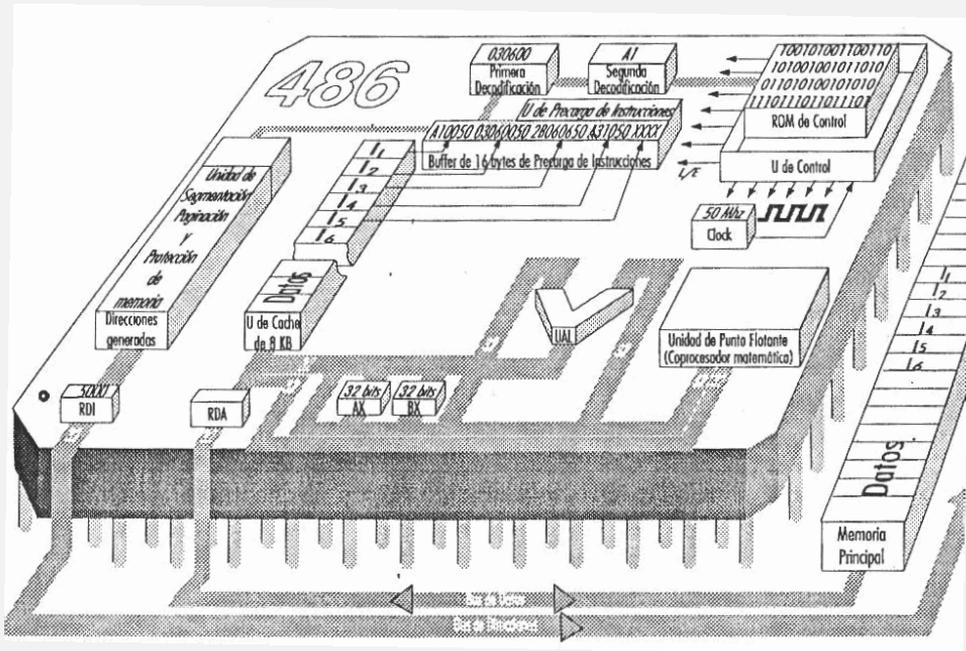


Fig. 9.10. Bloques y sub-bloques del Intel 80486.

Todas estas unidades participan en el "pipeline" de instrucciones, que en el 486 consta de 5 etapas, que progresan con cada pulso reloj, al compás de sus millones de ciclos por segundo:

**1. Pre-carga** ("pre-fetch") consiste en la llegada de los códigos de las próximas instrucciones que entrarán al "pipeline" a dos buffers (de 16 bytes cada uno) de la Unidad de Pre-carga, para formar una "cola".

**2. Primera Decodificación:** a la Unidad de Decodificación llegan los primeros 3 bytes de cada instrucción, para separar - entre todos los bytes que forman su código de máquina - su código de operación, del número que hace referencia a la dirección del dato (los códigos de operación pueden tener de 1 a 3 bytes).

**3. Segunda Decodificación:** el código de operación identificado en el paso anterior es ahora decodificado. Esto permite determinar la secuencia de **microcódigo contenida** en la ROM de Control, merced a la cual la UC generará las señales de control, que enviará por las líneas que salen de ella, para que cada unidad que controla, ejecute una parte de la instrucción con cada pulso reloj. Si la instrucción es simple se ejecuta en un solo pulso. Al mismo tiempo que pasa esto por esta etapa del

"pipeline", otros tres bytes del código entran a la etapa de primera codificación.

**4. Ejecución:** se ejecuta una operación en la ALU por ejemplo, leyendo el dato a operar en el caché. Paralelamente con la acción recién descrita se van ejecutando nuevas tareas en la primera y segunda decodificación.

**5. Almacenamiento de resultados:** esta es la etapa final del "pipeline" completándose la ejecución de la instrucción. En ésta, el resultado de la ALU se almacena, así como los "flags" que ella también genera, resultantes de la operación, en el registro de estado.

## 6.2 Funcionamiento básico del procesador Intel Pentium

La Figura 9.11 da cuenta de un esquema básico de un Pentium basado en el del 486.

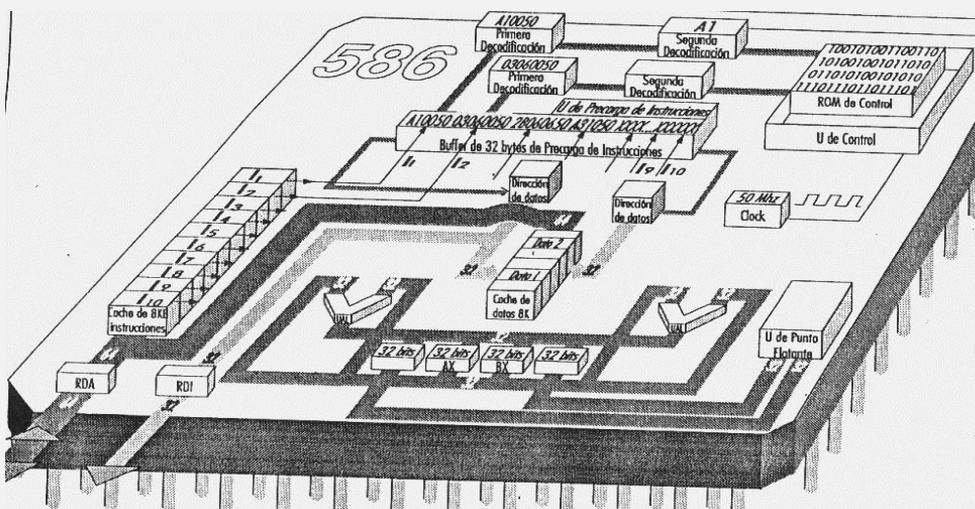


Fig. 9.11. Bloques del Intel Pentium.

Como en el 386 y el 486, en el Pentium las instrucciones para enteros, siguen un "pipeline" de 5 etapas.

Para la etapa de pre-carga se supone que en un caché interno de 8 KB se encuentran las próximas instrucciones a ejecutar, las cuales en el Pentium pasan en promedio de a diez juntas hacia un buffer de la Unidad

de pre-carga, que puede almacenar 32 bytes (existen dos de estos buffers). O sea, en dicho caché se leen 32 bytes en un solo acceso. Los datos están en otro caché de 8 KB.

Tener dos memorias caché separadas permite que mientras por un lado se accede a las próximas instrucciones a ejecutar en un caché, al mismo tiempo en el otro, se accede a datos, sin tener que esperar.

Por tener el Pentium un bus de datos externo e interno de 64 bits que llega a cada caché, se posibilita que en cada acceso al caché externo (para leer un dato o instrucción contenido en éste, y si no lo está se accede a memoria principal), cada caché reciba el doble de datos o instrucciones - según cual sea - que en el 486.

El Pentium, contiene dos "pipeline" para instrucciones, que operan con números enteros, a fin de poder procesar dos instrucciones en forma independiente como una fábrica de autos con dos líneas de montaje. Esto lo hace "superescalar", capaz de terminar de ejecutar dos instrucciones en un pulso, como los procesadores RISC, por lo cual también como en éstos, se requiere un caché para datos y otro para instrucciones. Asimismo, deben existir por duplicado: la unidad decodificadora (de modo de poder decodificar dos instrucciones por vez), la unidad de segmentación generadora de direcciones de datos, y la ALU.

Puesto que dos instrucciones en proceso simultáneo pueden necesitar acceder juntas al caché de datos para leer cada una su dato a operar, este caché tiene duplicado el número de líneas de datos y de direcciones.

A la primer decodificación entran dos instrucciones al mismo tiempo. Durante la misma se determina si ambas se procesarán juntas una (en cada "pipeline") o si sólo seguirá una por el "pipeline". También se identifica la porción de cada instrucción que permite formar la dirección del dato (que pasa a la unidad de segmentación correspondiente), y cada código de operación (que pasará a la segunda codificación).

Una instrucción para números en punto flotante opera con datos de 64 bits, que ocupan los dos "pipelines" para números enteros (de 32 bits cada uno), por lo que ella no puede procesarse junto con otra instrucción.

Estas instrucciones pasan por las cinco etapas correspondientes a instrucciones para enteros, y además requiere 3 etapas de un "pipeline" exclusivo para punto flotante. Puede decirse que el Pentium presenta un

"pipeline" de 8 etapas, siendo que las instrucciones para enteros se ejecutan en 5 etapas.

Las denominadas instrucciones "simples" para enteros, luego de haber pasado por la pre-carga, y las dos decodificaciones (pasos 1, 2 y 3) se ejecutan en uno, dos o tres pulsos reloj, según sea su complejidad.

Las instrucciones muy simples, por ejemplo con datos a operar en registros de la UCP, y el resultado de la operación asignado a otro registro de la UCP, se ejecutan en un solo pulso reloj, luego de la segunda decodificación.

Si en la primer decodificación se determina que el par de instrucciones identificadas son simples, y que la segunda en **orden no depende** del resultado de la primera, cada una sigue su ejecución en uno de los dos "pipelines", o sea, que se procesan en paralelo.

Y si además, ambas se ejecutan en igual cantidad de pulsos reloj, al cabo del último de ellos, las mismas se terminan de ejecutar simultáneamente. Esta es la forma en que el Pentium puede ejecutar dos instrucciones en un pulso reloj, lo cual significa que los resultados de las operaciones ordenadas se obtienen a un mismo tiempo.

## 7 Ejercitación

### **Ejercicio 1:**

¿Por qué es imposible que un pipeline de instrucciones de dos etapas reduzca el tiempo de ciclo de instrucciones a la mitad, en comparación con un diseño sin pipeline?. Justifique.

### **Ejercicio 2:**

¿Qué diferencia hay entre localidad espacial y localidad temporal?. Desarrolle.

### **Ejercicio 3:**

Desarrolle dos programas en pseudocódigo, uno que favorezca la localidad espacial, pero no la temporal, y el otro lo contrario.

### **Ejercicio 4:**

Cuando apareció la memoria caché, cada sistema disponía de una única caché. Pero los procesadores actuales disponen de diversos niveles de memoria caché. ¿Cuál es el número de niveles apropiados? Justifique.

### **Ejercicio 5:**

¿Cómo se relaciona el principio de localidad con el uso de múltiples niveles o una jerarquía de memoria en una computadora?. Desarrolle.

**Ejercicio 6:**

Los diseños de los primeros sistemas que incluyeron memoria caché disponían de una única caché para almacenar las referencias tanto a operandos como a instrucciones. ¿Por qué los procesadores actuales tienden a tener cachés separadas o divididas: una dedicada a las instrucciones y la otra a los datos u operandos. Considere los conceptos de pipeline y la búsqueda de los datos asociados que le correspondan.

**Ejercicio 7:**

¿Los procesadores RISC tienen unidades de control microprogramadas o cableadas? Justifique.

**Ejercicio 8:**

¿Por qué resulta importante el desarrollo de los compiladores para máquinas RISC? Justifique.

---

# CAPÍTULO 10

---

## Entradas y Salidas (E/S)

- 1 Módulos, Canales y Procesadores de E/S**
    - 1.1 Introducción**
    - 1.2 Módulo de E/S**
    - 1.3 Diagrama en bloques de un módulo de E/S**
    - 1.4 Técnicas para las operaciones de E/S**
    - 1.5 Ejemplo de módulo de E/S**
    - 1.6 Canales y procesadores de E/S**
  - 2 Datos, Señales e Interfaces**
    - 2.1 Datos y señales**
    - 2.2 Ancho de banda**
    - 2.3 Señalización**
    - 2.4 Interface serie y paralela**
    - 2.5 Configuraciones punto-a-punto y multipunto**
    - 2.6 Medios de enlace**
  - 3 Casos de Estudio**
    - 3.1 Puerto USB**
    - 3.2 Puerto Ethernet**
    - 3.3 Puerto Bluetooth**
    - 3.4 Puerto HDMI**
    - 3.5 Puerto Wi-Fi**
  - 4 Ejercitación**
-

## Capítulo 10

# Entradas y Salidas (E/S)

## 1 Módulos, Canales y Procesadores de E/S

### 1.1 Introducción

El tercer componente fundamental de una computadora es la Unidad de Entradas y Salidas (UE/S), además de la CPU y la memoria. En realidad se trata de un conjunto de módulos de E/S especializados que se conectan habitualmente al sistema de buses (bus de direcciones, bus de datos y bus de control) y controlan la comunicación con uno o más dispositivos periféricos (o simplemente periféricos). Un módulo de E/S tiene la capacidad necesaria para permitir la comunicación entre el periférico y el sistema de buses.

Los periféricos no se conectan directamente al sistema de buses, para alcanzar a la CPU y la memoria (salvo algún caso especial), por diversos motivos. Los periféricos son muy diferentes entre sí, y debería disponerse de una lógica adicional dentro de la CPU para cada dispositivo. Además, la mayoría de los periféricos son lentos respecto a la CPU y la memoria, por lo que se desaprovecharía tiempo y prestaciones conectándolos a un sistema de buses de alta velocidad. También es inconveniente que la CPU y la memoria gestionen las transferencias de algunos periféricos de muy altos rendimientos. Finalmente, los periféricos son distintos constructivamente, y utilizan datos con formatos y tamaños distintos a los de la computadora con la que se comunican.

Los módulos de E/S son la interface o nexo entre el procesador y la memoria, mediante el conjunto de buses por un lado, y los dispositivos periféricos usando enlaces de datos específicos (Figura 10.1).

Los periféricos se comunican a la computadora a través de un enlace a un módulo de E/S específico. El enlace permite el intercambio de señales de control, de estado y los datos entre el módulo de E/S y el dispositivo externo. Con las señales de control se determina la función que debe realizar el periférico. Mientras que con las señales de estado se indica el estado del dispositivo, por ejemplo, para señalar si está preparado o no para una transferencia. Finalmente, con los datos se produce el efectivo intercambio de la información digital que se envía o recibe desde el módulo de E/S.

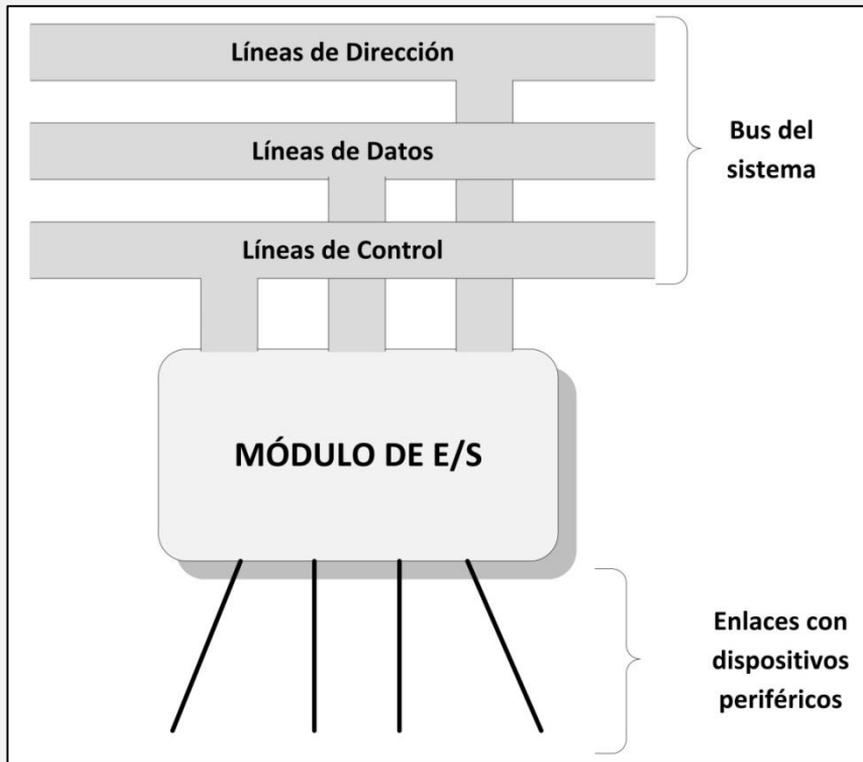


Fig. 10.1. Módulo de E/S.

Por lo expuesto, cada periférico deberá tener una lógica de control que controla la operación con el módulo de E/S. Las señales eléctricas de los datos pueden requerir una transducción o adecuación en el caso de una salida o una entrada dependiendo de las características operativas del periférico. Finalmente, el dispositivo puede disponer de una memoria o buffer para almacenar temporalmente los datos que van o vienen desde el módulo de E/S hacia el periférico. Estos aspectos se muestran en la Figura 10.2.

Los periféricos se clasifican según:

- El **sentido de la transferencia de los datos** en: Periféricos de entrada, Periféricos de salida y Periféricos de entrada-salida.
- **Con quién interactúan** en: de interacción con humanos y de interacción con máquinas

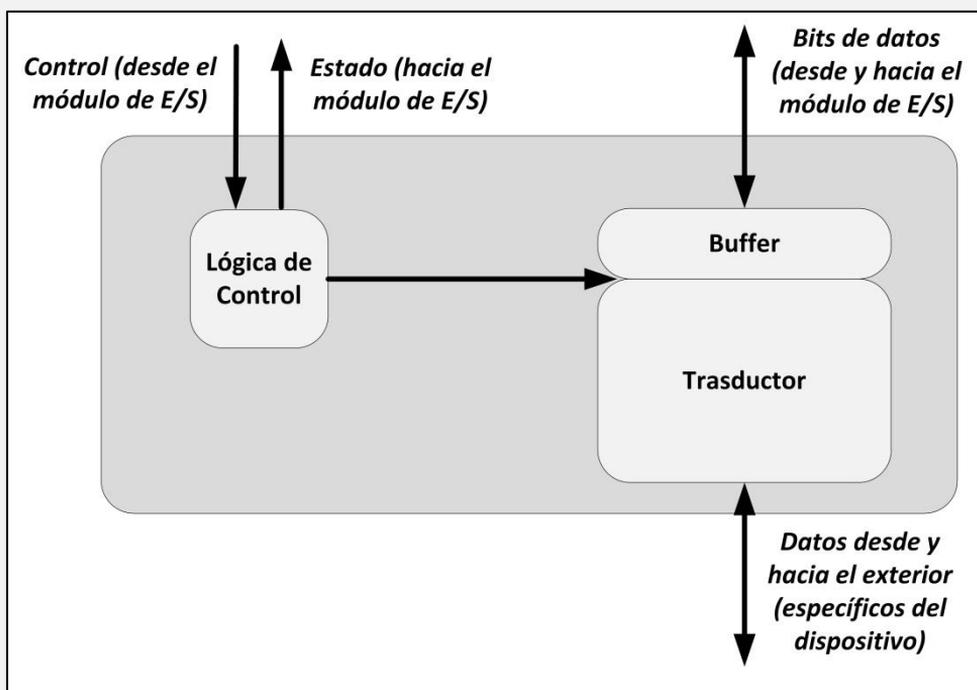


Fig. 10.2 Diagrama en bloques de un dispositivo externo o periférico

## 1.2 Módulos de E/S

La CPU puede requerir comunicación con un periférico según las necesidades de un programa. La CPU, la memoria principal y los buses deben compartirse con las transferencias de datos hacia o desde los periféricos. Por lo tanto, los módulos deberán incluir funciones de **control y temporización** que coordinen el tráfico entre los recursos internos y externos de la computadora.

Además, los módulos de E/S deben **comunicarse con el procesador**, del que recibirá órdenes e intercambiará datos e información de estado, a través del bus de control y del bus de datos, respectivamente. Y lógicamente, también habrá **comunicación con el periférico** mediante los enlaces externos, que también implica órdenes, información de estado y los datos.

Una función común de un módulo de E/S es el **almacenamiento temporal de datos**. Esta capacidad resuelve la diferencia de velocidades entre los dispositivos internos y externos. Por ejemplo, una ráfaga de datos que proviene de la CPU y la memoria principal pueden almacenarse

en un buffer del módulo de E/S, y luego enviarse al periférico a la velocidad de este. Y el mismo criterio puede seguirse para una transferencia en el sentido contrario, que libera a la memoria principal de estar mucho tiempo ocupada cuando se trata de una operación de transferencia lenta.

Finalmente, los problemas mecánicos y eléctricos de los periféricos deben informarse a través de una función de **detección de errores** y comunicarse a la CPU.

### 1.3 Diagrama en bloques de un módulo de E/S

La Figura 10.3 presenta un diagrama en bloques de un módulo de E/S. Como se indicó previamente, el módulo se conecta a la CPU y la memoria a través de las líneas del bus del sistema (indicadas a la izquierda de la figura). Los datos se almacenan temporalmente en uno o más registros de datos internos del módulo, y otros tantos registros de estado. La lógica de control del módulo interactúa con la CPU mediante las líneas de control. Además, el módulo de E/S debe estar preparado para reconocer las direcciones que provienen de la CPU y generar las direcciones asociadas a los periféricos que controla (si son más de uno). Por ese motivo, cada módulo de E/S tiene una dirección única, o un conjunto único de direcciones si controla más de un dispositivo externo. Finalmente, posee la lógica dedicada de interfaz con cada uno de los dispositivos que controla

En estos términos, cada módulo oculta los detalles de temporización, formatos de datos y aspectos constructivos de los dispositivos externos. El procesador observa una versión simplificada del periférico al que le aplica órdenes de lectura y escritura.

Los módulos de E/S pueden clasificarse de acuerdo a los detalles de procesamiento presentados a la CPU, en:

- **Canal o procesador de E/S:** cuando el módulo se encarga de la mayoría de los detalles de procesamiento, presentando a la CPU una interface de alto nivel. Este esquema se usa en grandes computadoras (mainframes)
- **Controlador de E/S o del dispositivo:** cuando el módulo es bastante sencillo y requiere de la CPU un control más detallado.

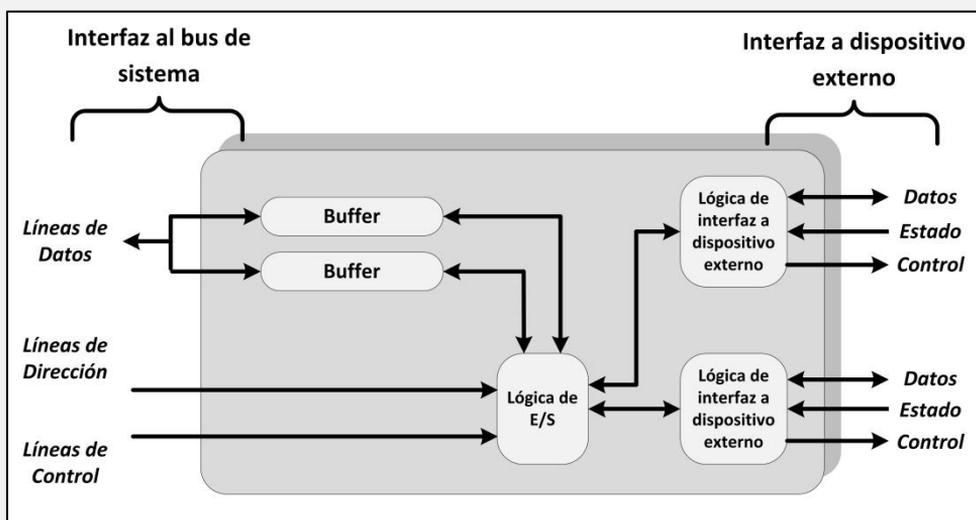


Fig. 10.3. Diagrama de bloques de un módulo de E/S.

## 1.4 Técnicas para las operaciones de E/S

Para las operaciones de E/S (como fue discutido en el Capítulo 8), se pueden utilizar tres técnicas:

- **E/S programada:** Los datos se intercambian entre el procesador y el módulo de E/S. Con un programa que se ejecuta en la CPU se controla directamente la operación de E/S. Cuando el procesador envía una orden al módulo de E/S debe esperar que esta operación concluya. La gestión de la operación de E/S por parte de un programa ejecutándose en una CPU rápida genera un gran desperdicio de tiempo.
- **E/S mediante interrupciones:** La CPU envía la orden de E/S, pero continúa ejecutando otras instrucciones y es interrumpida por el módulo de E/S cuando ha terminado la operación.
- **Acceso directo a memoria (DMA):** El módulo de E/S y la memoria principal intercambian datos directamente sin la intervención de la CPU.

En las dos primeras técnicas, la CPU es responsable de almacenar los datos leídos desde el módulo de E/S en la memoria en una operación de entrada, y de leerlos desde la memoria en una operación de salida.

## 1.5 Ejemplo de módulo de E/S

Un ejemplo de módulo de E/S utilizado tanto para la E/S programada como para la E/S mediante interrupciones es la interface programable Intel 82C55A. Se trata de un módulo de E/S de propósito general integrado en un solo chip y diseñado para usarse en conjunto con la CPU 8088/8086. La Figura 10.4 muestra su diagrama en bloques general.

A la derecha se observa la interface externa, con 24 líneas de E/S salidas digitales programables por el 8088/86 mediante un registro de control. A través de este registro se pueden establecer diversos modos de operación y configuraciones operativas. Las 24 líneas se dividen en tres grupos o registros de 8 bits cada uno, llamados A, B y C. Cada registro puede funcionar como un puerto de E/S de 8 bits. Alternativamente, el registro C puede subdividirse para usarse como dos grupos de 4 bits asociados a los registros de E/S A y B. En esta configuración, esos subgrupos contienen las señales de control y estado para los registros A y B.

Hacia la izquierda se encuentra la interface interna que se conecta con el bus de datos del 8088/86. Se trata de 8 líneas de datos bidireccional (D0-D7) que se usa para transferir los datos desde y hacia los puertos de E/S, y la información del registro de control. Las dos líneas de direcciones (A0-A1) seleccionan cada uno de los 3 puertos de E/S o el registro de control.

El 82C55 puede usarse para controlar diversos dispositivos simples. La Figura 10.5 muestra el diagrama en bloques para controlar una terminan con teclado y pantalla. El teclado proporciona 8 bits de entrada (6 de datos, y los bits SHIFT y CONTROL). Y dos líneas adicionales para la sincronización del teclado. La pantalla también está conectada a un puerto de datos de 8 bits. Dos de esos bits tienen un significado específico para la pantalla, y se usan 4 líneas para el control y sincronización.

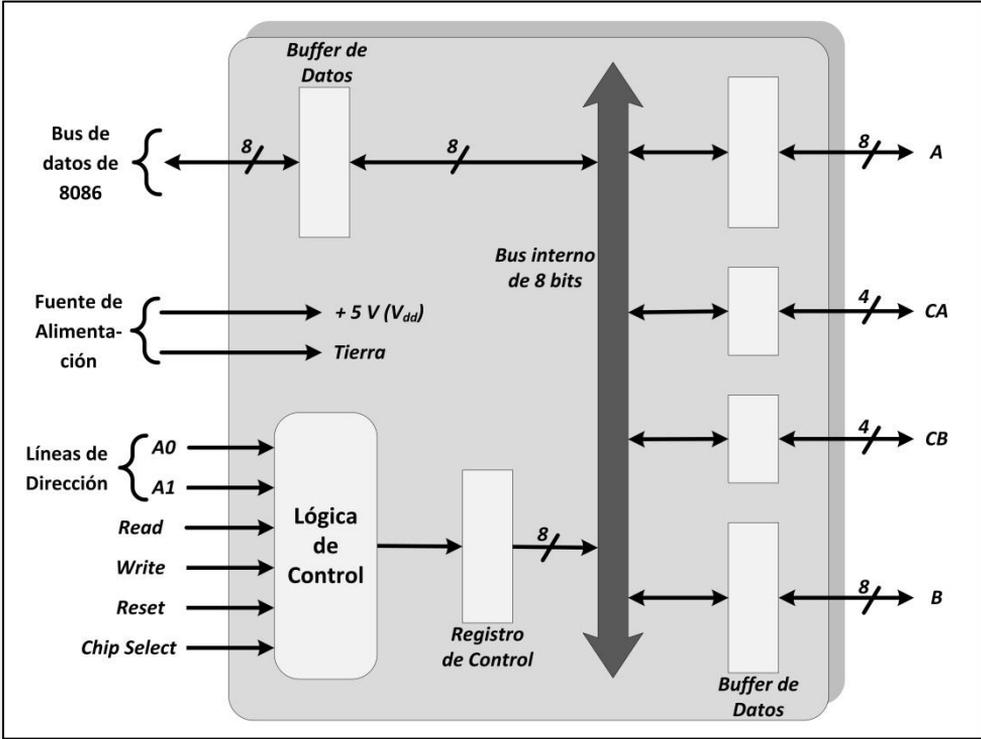


Fig. 10.4. Diagrama en bloques de la interface programable de periféricos 802C55A de Intel.

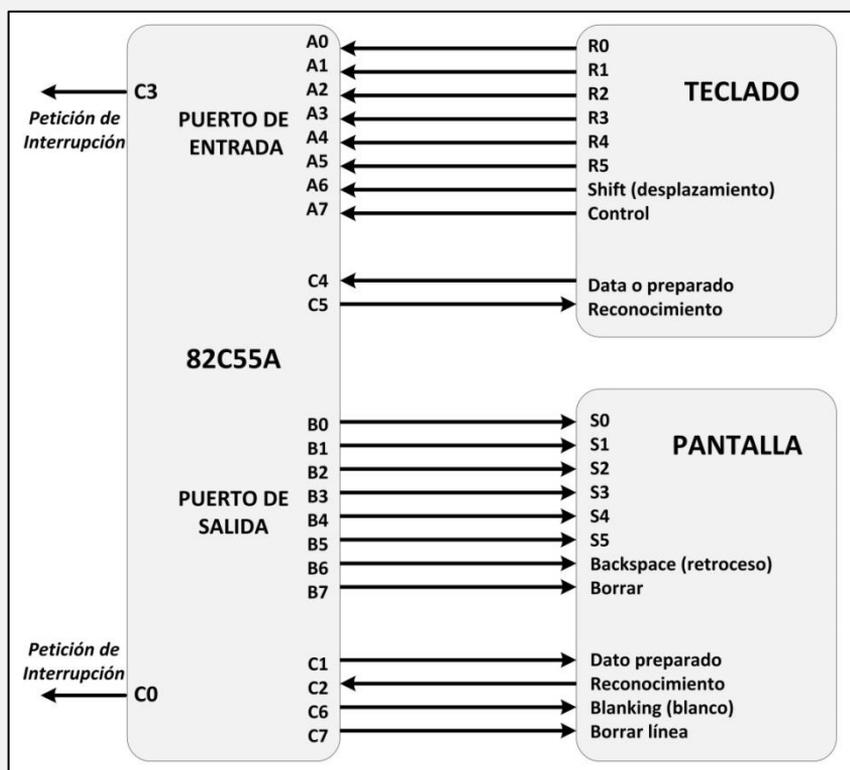


Fig. 10.5. Interface teclado/pantalla usando el 802C55A.

## 1.6 Canales y procesadores de E/S

La evolución de las computadoras y la creciente complejidad de las computadoras también se han manifestado en sus componentes, incluidos los módulos de E/S. Esa evolución puede describirse con la siguiente secuencia:

- En los dispositivos simples, la CPU controla directamente al periférico.
- La CPU usa E/S programada sin interrupciones con la incorporación de un controlador o módulo de E/S. La CPU se independiza de aspectos específicos de las interfaces de los dispositivos externos.
- La CPU no necesita esperar a que termine la operación de E/S, excepto al comienzo y final de la transferencia. Es el mecanismo anterior con interrupciones.

- La CPU permite el acceso directo de un módulo de E/S a la memoria a través del DMA. Se transfiere un bloque de datos, desde o hacia, la memoria sin la participación de la CPU, excepto al comienzo o al final de la transferencia.

Cada vez más funciones de E/S se realizan sin la participación de la CPU. Este aspecto releva a la CPU de ciertas tareas mejorando las prestaciones generales de la computadora.

La complejidad del módulo de E/S ha crecido de tal forma que incluye mejoras significativas, como las siguientes:

- El módulo de E/S puede comportarse como un procesador en sí, con su propio repertorio de instrucciones orientado a las E/S. La CPU hace que el procesador ejecute un programa de E/S en memoria, y es interrumpida cuando se completa la secuencia entera. En este caso, el módulo de E/S recibe el nombre de canal de E/S.
- Cuando además incluye un memoria local propia, transformando el módulo de E/S en una computadora en sí misma. Con esta arquitectura, el módulo de E/S puede controlar un conjunto grande de dispositivos de E/S con mínima intervención de la CPU. Y en este caso, al módulo se le llama procesador de E/S.

Aunque habitualmente los nombres de canal de E/S y procesador de E/S se usan de manera intercambiada.

Un canal es una extensión del concepto de DMA. Un canal puede ejecutar instrucciones de E/S (relevando a la CPU de dichas operaciones), en dos modos operativos:

- **Canal selector** (Figura 10.6 a) que controla varios dispositivos de velocidad elevada, salvo que en un instante se dedica a transferir datos a uno de estos dispositivos.
- **Canal multiplexor** (Figura 10.6 b) que puede controlar las E/S de varios dispositivos al mismo tiempo.

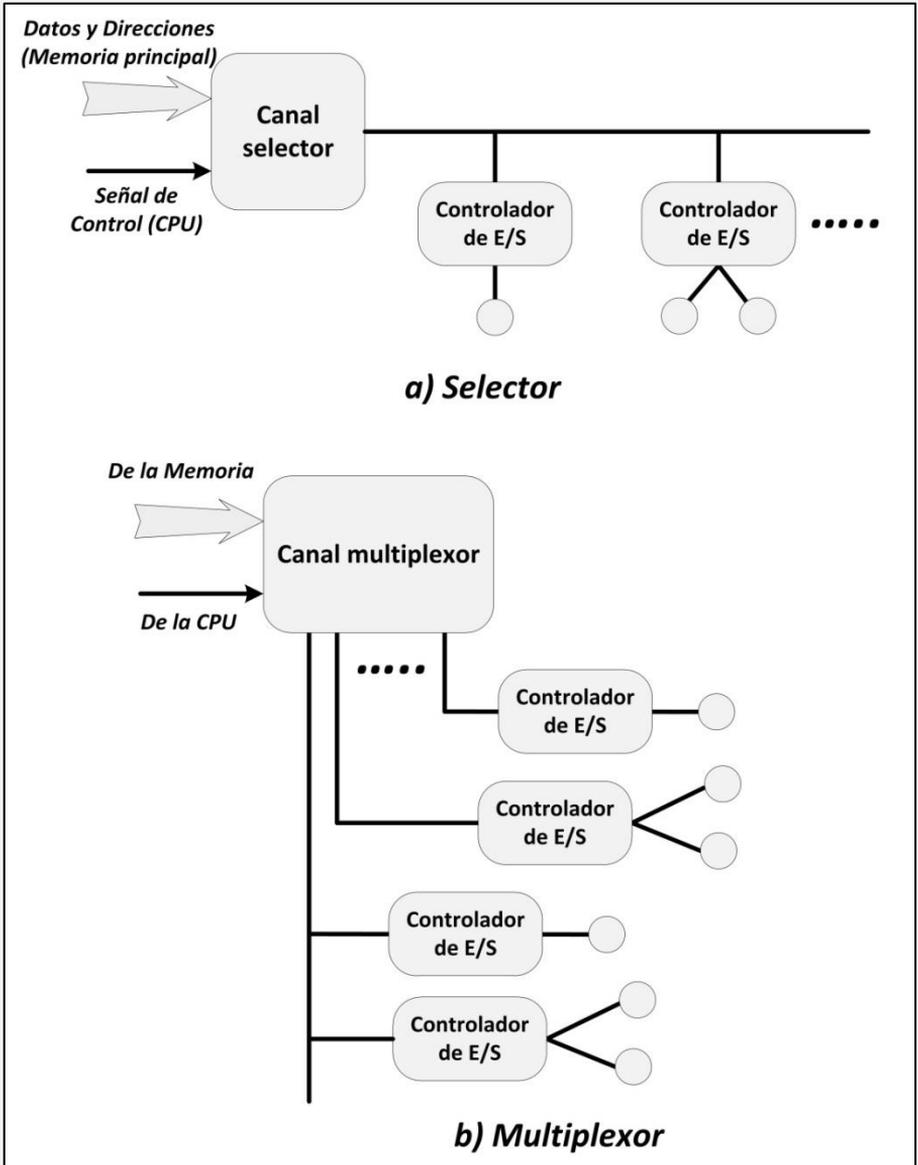


Fig. 10.6 Arquitectura de un canal de E/S

## 2 Datos, Señales e Interfaces

### 2.1 Datos y señales

Se considera a un DATO como un ente abstracto que contiene información. Para lograr su representación física utilizamos las ondas electromagnéticas. A esta representación la llamamos SEÑAL. Las señales electromagnéticas pueden ser CONTÍNUAS cuando su intensidad varía suavemente en el tiempo, o DISCRETAS cuando su intensidad cambia bruscamente en el tiempo. En este último caso, si la cantidad de valores que puede adoptar una señal en el tiempo son 2 (nivel alto y nivel bajo), la señal se llama señal binaria. La Figura 10.7a indica una señal continua y la figura 10.7b una señal discreta.

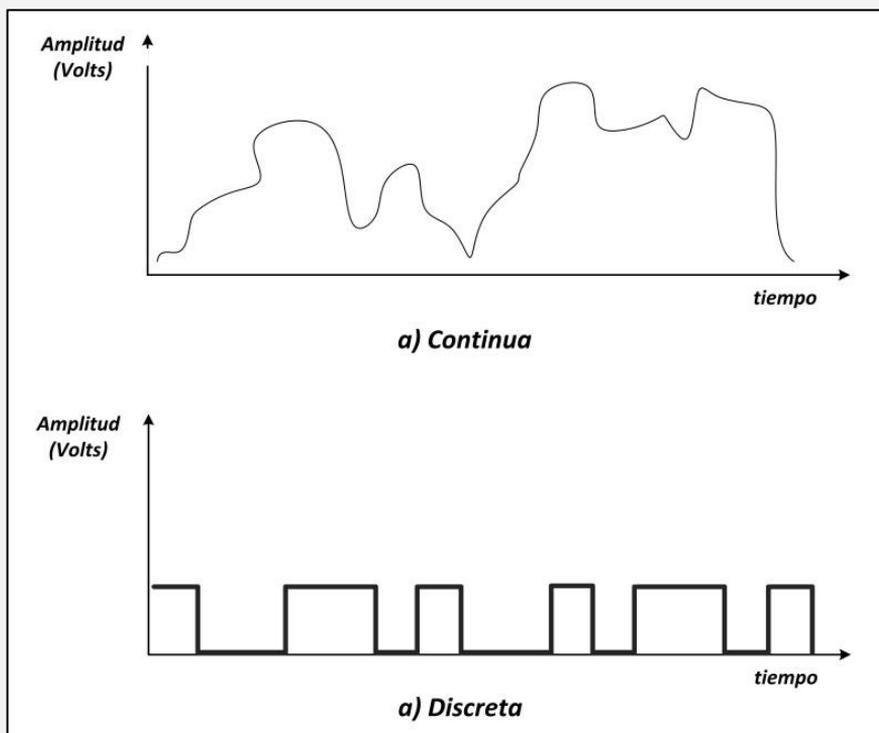


Fig. 10.7. Señales continuas y discretas

Los datos, como las señales, pueden ser continuos o discretos. A los datos continuos se los llama ANALÓGICOS y a los discretos se los llama DIGITALES.

Tanto los datos analógicos como los digitales pueden transportarse desde una fuente a un destino. Este transporte mediante la propagación y procesamiento de señales recibe el nombre de TRANSMISIÓN.

## 2.2 Ancho de banda

Se puede demostrar (a través de la serie de Fourier) que cualquier señal está formada por componentes sinusoidales de distintas frecuencias. En el dominio de las frecuencias, estas componentes conforman lo que llamamos ESPECTRO de la señal. El ancho del espectro se llama ANCHO DE BANDA ABSOLUTO de la señal. Cualquier señal discreta tiene un espectro con infinitas componentes sinusoidales y, por lo tanto, su ancho de banda es infinito. Sin embargo, observando con detalle el espectro de señales discretas se ve que la mayor parte de la energía se concentra en un sector estrecho del mismo. A esta banda se la conoce como ANCHO DE BANDA EFECTIVO o simplemente ANCHO DE BANDA (AB) de la señal. Su unidad es el Hz.

El ancho de banda (AB) es un concepto esencial en la transmisión de datos ya que se verifica que los sistemas de transmisión (transmisor, medio y receptor) solo pueden transferir eficazmente una banda limitada de frecuencias. A esta banda se la define como ancho de banda del sistema de transmisión. Mayor ancho de banda implica mayor costo del sistema.

El ancho de banda (AB) también está relacionado con la velocidad de transmisión, que mide la cantidad de datos que pueden transmitirse por unidad de tiempo.

En las transmisiones digitales la velocidad de transmisión (VT) se mide en bits por segundo (bps), e indica la cantidad de bits que se pueden transmitir en un segundo.

Cuanto mayor es el AB de un sistema de transmisión, mayor es la velocidad con que se pueden transmitir los datos en el mismo. En general, puede comprobarse que con un AB de X Hz para un sistema de transmisión, es posible transmitir con una velocidad de transmisión VT de C bps = 2 X Hz como máximo (dependiendo de las características del canal de comunicaciones: atenuación, ruido, distorsión y tasa de errores).

## 2.3 Señalización

La transformación de un dato en su representación electromagnética con el fin de su propagación física por un medio, recibe el nombre de señalización.

Si el dato es analógico se puede señalar y obtener una señal analógica (por ejemplo: la voz en el teléfono) u obtener una señal digital (por ejemplo: la voz en un CODEC).

Si el dato es digital se puede señalar y obtener una señal analógica (por ejemplo: usando un MODEM) u obtener una señal digital (por ejemplo: transmisor digital).

Las señales analógicas o digitales pueden usar transmisión analógica o transmisión digital.

La transmisión digital ha avanzado desplazando a la transmisión analógica en muchísimas aplicaciones. Lo mismo sucede con la señalización digital sobre la analógica. Las razones son:

- Evolución del soporte digital (hardware): CIs con mayores prestaciones
- Integridad de los datos: Repetidoras en lugar de amplificadores
- Multiplexado: Mayor eficacia con técnicas digitales (mx en tiempo)
- Seguridad e integración: Encriptación

### Señalización digital

También se suele mencionar a señalización digital como codificación.

Esta codificación, entonces, realiza la transformación de datos digitales en señales digitales. Las señalizaciones más utilizadas son:

- NRZ
- NRZI
- BIPOLAR AMI
- PSEUDOTERNARIO
- MANCHESTER
- MANCHESTER DIFERENCIAL

La Figura 10.8 muestra estas codificaciones

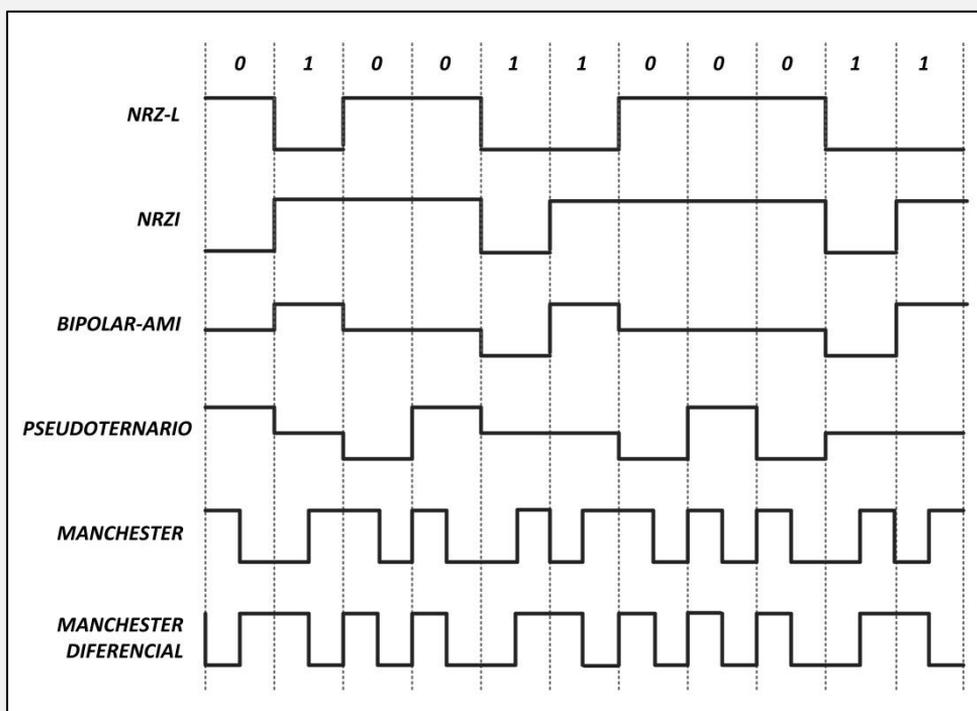


Fig. 10.8. Señalización Digital

Las transmisiones que se realizan entre la CPU, la UM y la UE/S presentan señalizaciones digitales, generalmente NRZ-L a través del BUS que representa un medio guiado.

Las transmisiones entre la computadora y el exterior (periféricos) también utilizan este tipo de señalización. Los controladores de periféricos, ver Unidad 4, pueden estar conectados al BUS COMÚN o bien a un BUS de E/S. La arquitectura de una Máquina actual incluye algunos controladores estandarizados para facilitar y generalizar la transmisión de datos desde y hacia la máquina. Estos controladores reciben el nombre de "puertos" y se corresponden con estándares que incluyen los protocolos de transmisión.

## 2.4 Interface serie y paralela

La interface entre el periférico y el módulo de E/S debe tener en cuenta las características y el funcionamiento del periférico. Una de las principales características de la interface es si es **serie o paralela** (Figura 10.9).

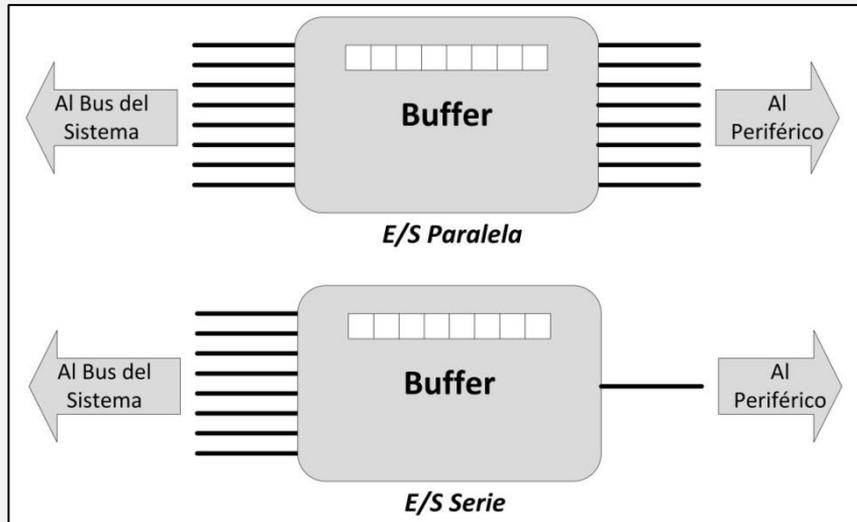


Fig. 10.9. E/S paralela y serie.

En una **interface paralela** hay varias líneas conectadas al módulo de E/S y el dispositivo periférico, y se transfieren varios bits simultáneamente a través del bus de datos. En una **interface serie** hay solo una línea para transmitir los datos, y por lo tanto, los bits deben transmitirse uno a uno. Las interfaces paralelas se utilizaron usualmente para los dispositivos de alta velocidad. Sin embargo, con la nueva generación de interfaces series de alta velocidad, las interfaces paralelas son cada vez menos comunes.

Cualquiera sea el caso, el módulo de E/S debe establecer un diálogo con el periférico. Ese diálogo para una operación de escritura requiere que el módulo de E/S envíe una señal de control solicitando permiso para enviar datos. Luego, el periférico debe reconocer la solicitud para que el módulo de E/S inicie la transferencia de los datos. Y finalmente, el periférico debe reconocer la recepción de los datos.

#### 2.4.1 Transmisión paralela

En este caso se transmiten varios bits a la vez y requerirá tantas líneas como bits se transmitan, y líneas adicionales a fin de resolver el sincronismo. Este tipo de transmisión es usual entre las unidades internas de una computadora a través del BUS, por ejemplo la comunicación entre la CPU, la UM y UE/S. También se usó en algunas

transmisiones entre la computadora y periféricos a través de una interface y un BUS dedicado.

La cooperación entre Transmisor y Receptor da lugar a dos técnicas bien diferenciadas:

- Transmisión paralela con control Estroboscópico
- Transmisión paralela con Handshaking

### Transmisión paralela con control estroboscópico

Además de las líneas de datos, este tipo de transmisión paralelo tiene una línea de sincronismo. En la figura 10.10 se indica este caso.



Fig. 10.10. Transmisión Paralela estroboscópica.

Cuando el Receptor detecta un flanco de bajada en la línea de sincronismo, lee y guarda el estado de las líneas de datos (Figura 10.11).

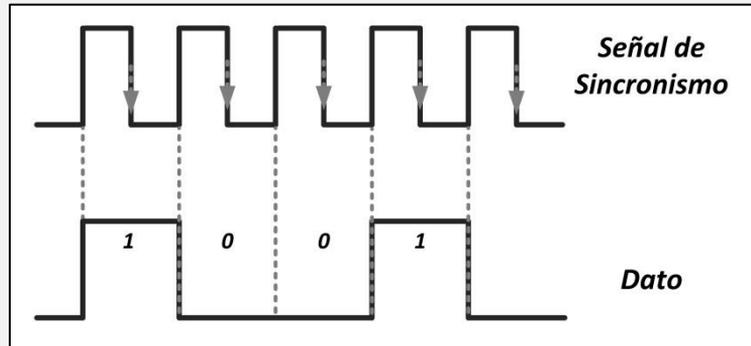


Fig. 10.11 Diagrama de tiempo para transmisión paralela estroboscópica.

## Transmisión paralela con handshaking

En este caso, además de las líneas de datos tiene dos líneas de control que resuelven el sincronismo. Estas líneas son:

- **Señal de Envío:** el Transmisor genera esta señal mediante la cual le avisa al Receptor que en las líneas de datos está el dato enviado.
- **Señal de Aceptación:** el Receptor genera esta señal mediante la cual le avisa al Transmisor que ha recibido y guardado el dato.

Cuando el Transmisor recibe la señal de aceptación borra la Señal de envío; asimismo, cuando el receptor ve la señal de envío borrada, borra la señal de aceptación. De esta forma culmina el handshaking (apretón de manos).

En la Figura 10.12 se aprecia esta lógica y en la Figura 10.13 se puede ver el diagrama de tiempo del handshaking.

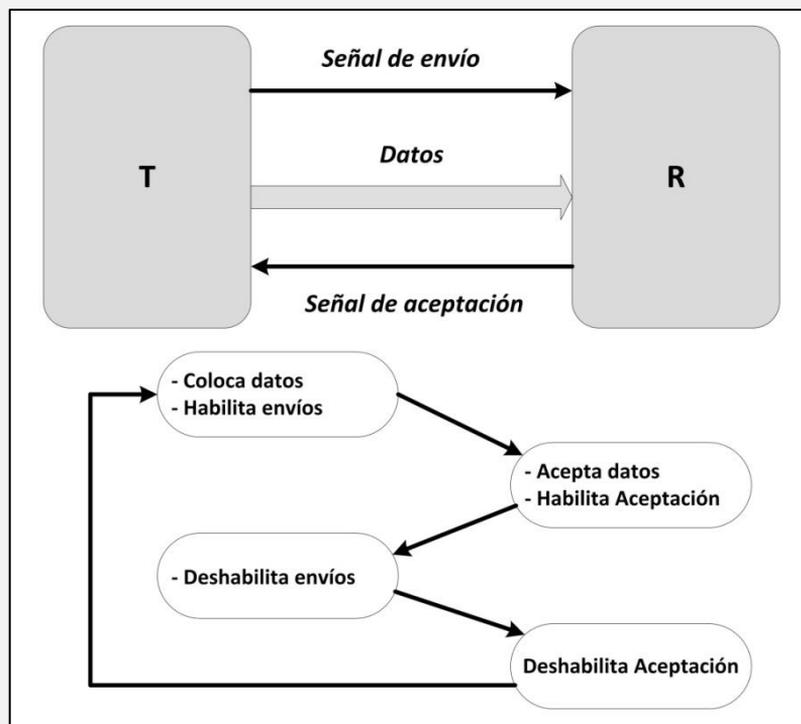


Fig. 10.12. Lógica de funcionamiento de la transmisión

paralela con handshaking.

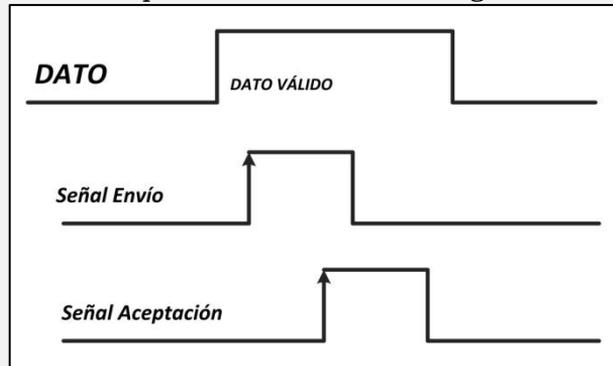


Fig. 10.13. Diagrama de tiempo que ejemplifica el handshaking.

La transmisión paralela es sensible a perturbaciones electromagnéticas externas e internas (del propio cable), y la señal de sincronismo se degrada a los pocos metros; por lo tanto, para distancias mayores se utiliza la transmisión serie.

## 2.4.2 Transmisión serie

En este tipo de transmisión los bits se envían uno a uno y se utilizan distintas señalizaciones. Existen dos tipos de transmisión serie:

- Transmisión Asíncrona
- Transmisión Síncrona

### Transmisión asíncrona

En este caso los datos se organizan carácter (un carácter tiene de 5 a 8 bits) a carácter. Es decir, los bits de un carácter se envían en serie, luego otro carácter y así sucesivamente a través de una única línea (Figura 10.14). El Receptor tiene la oportunidad de sincronizarse carácter a carácter.

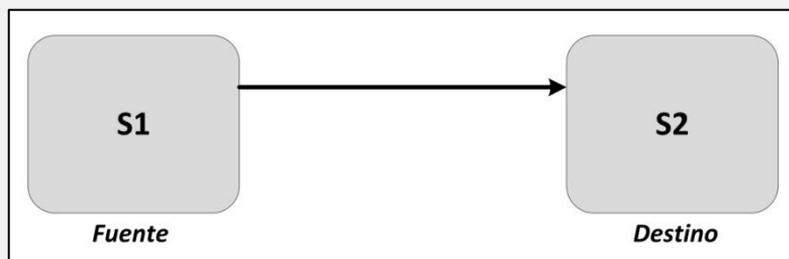


Fig. 10.14. Transmisión serie asíncrona.

Cuando no se transmite ningún carácter, la línea de comunicación está en reposo y el receptor se encuentra a la espera de recibir un bit de comienzo.

Una vez que el receptor detectó el bit de comienzo, empieza a leer el carácter. Una vez leído el carácter (es posible que después del carácter existe un bit de paridad) el receptor espera leer 1, 1.5 o 2 bits de parada. Este proceso se repite carácter a carácter logrando de esta forma la sincronización. Es necesario que el receptor conozca la velocidad del transmisor, y si se incluye o no el bit de paridad.

La Figura 10.15 muestra el caso de transmisión de un carácter de  $n$  bits en transmisión serie asíncrona

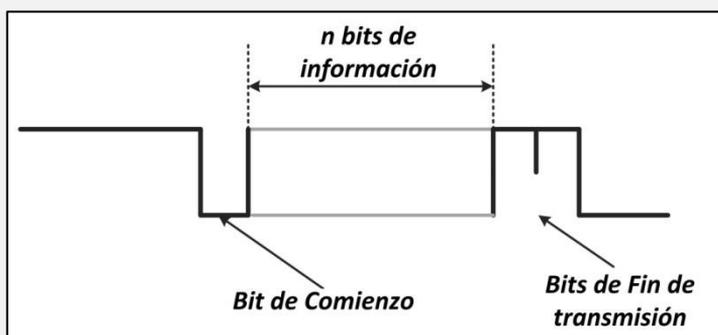


Fig. 10.15. Diagrama de tiempo para transmisión serie asíncrona.

## Velocidad de Modulación

La velocidad de modulación  $D$ , se define como:

$$D = VT / b$$

dónde:

$D$ : Velocidad de Modulación (BAUDIOS)

$VT$ : Velocidad de Transmisión (bps)

$b$ : cantidad de bits por elemento de señal

Nota: El lector puede verificar que  $b = \frac{1}{2}$  para la señalización Manchester y  $b = 1$  para la señalización NRZ-L

## Ejemplo de transmisión serie asíncrona

La figura 10.16 indica como ejemplo la norma RS232C

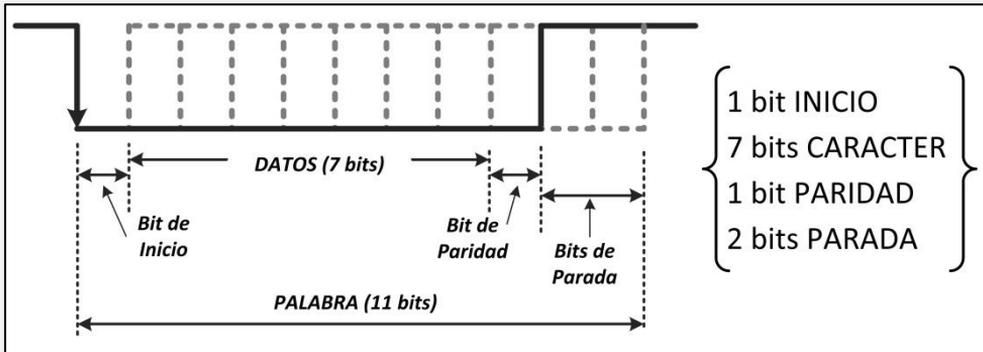


Fig. 10.16 RS232 C como ejemplo de transmisión asíncrona

Un esquema con las interfaces incluidas, puede verse en la Figura 10.17.

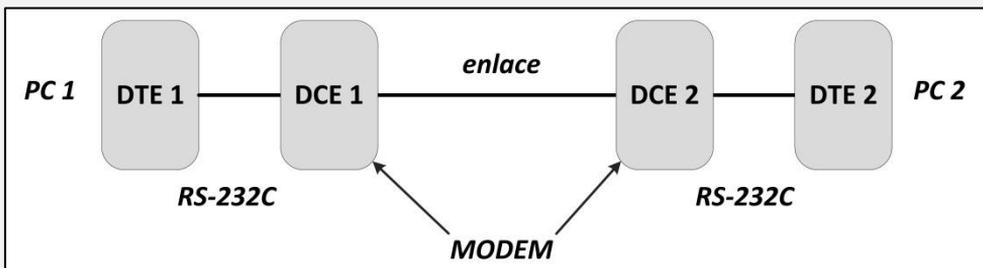


Fig. 10.17 Esquema completo de una comunicación RS232 C

Supongamos los siguientes valores:

- $VT = 110 \text{ bps} = 110 \text{ baudios}$
- $\text{Tiempo de palabra} = 11 \text{ bits} \times 1 / 110 \text{ bits} = 0,1 \text{ s}$
- $VT [\text{palabras/s}] = 1 / 0,1 \text{ s} = 10 \text{ palabras/s}$
- $VT [\text{carga útil}] = 10 \text{ palabras/s} \times 7 \text{ bits / palabra} = 70 \text{ bps}$

En este ejemplo la señalización utilizada implica un elemento de señal por bits. Por lo tanto, los baudios y los bps son iguales.

La siguiente relación nos da la eficiencia del enlace:

$$\text{Eficiencia [\%]} = (\text{VT}(\text{carga útil})/\text{VT}) \cdot 100$$

$$\text{Eficiencia [\%]} = (70/110) \cdot 100 = 63 \%$$

En la transmisión serie asíncrona, como acabamos de ver, la sincronización se logra cuando el receptor (que debe conocer la VT) detecta el bit de comienzo. Seguramente el reloj del receptor tendrá un corrimiento con respecto al reloj del transmisor. Este desfase inevitable no representa un problema como puede verse en la Figura 10.18.

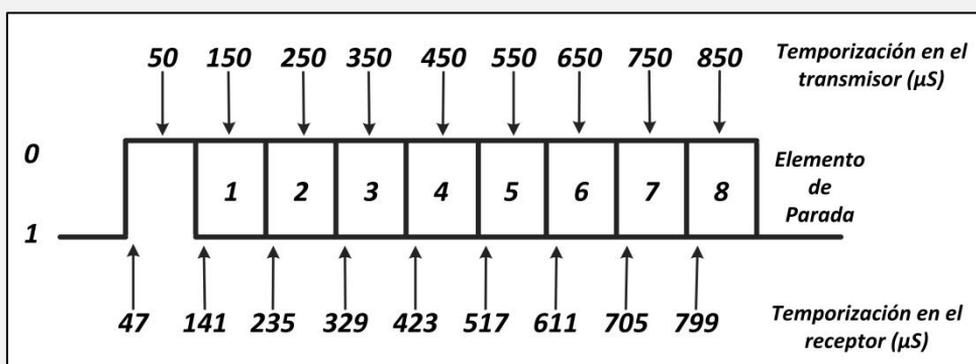


Fig. 10.18 Desfase en la temporización entre transmisor y receptor

### Transmisión síncrona

En este caso se transmite un bloque de una cantidad importante de bits sin utilizar los bits de comienzo y parada del caso asíncrono. Existen dos tipos de transmisión serie síncrona:

- Transmisión síncrona con línea de sincronización
- Transmisión síncrona autosincronizada.

El primer caso usa una línea adicional como un reloj de sincronización (Figura 10.19). Será utilizado por el receptor para leer el bit y guardarlo. La sincronización puede establecerse en el flanco descendente del reloj. Esta técnica funciona para distancias cortas ya que para mayores distancias la señal de sincronismo sufre una degradación importante.

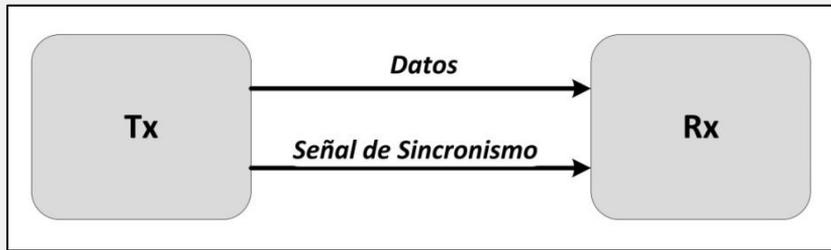


Fig. 10.19. Transmisión serie síncrona con reloj independiente.

La segunda alternativa transmite el sincronismo en la propia señal de datos (Figura 7.20). Esta técnica ahorra una línea de sincronismo especial y evita la degradación del mismo. Como contrapartida requiere un mayor ancho de banda. Por ejemplo, es el caso de la señalización Manchester usada en las Redes LAN Cableadas Ethernet (la señalización demanda 2 elementos de señal por bit).

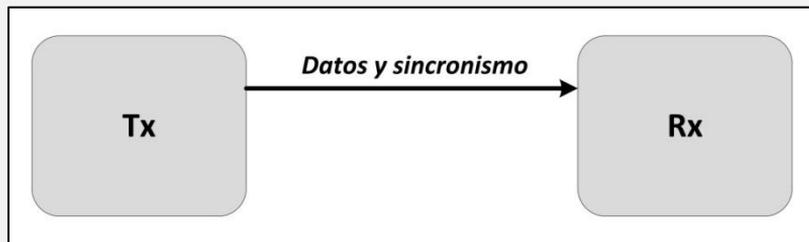


Fig. 10.20. Transmisión serie síncrona con reloj incorporado en los datos.

Además de cualquiera de las técnicas de sincronización mencionadas, en este tipo de transmisión, se necesita una sincronización adicional a fin que el receptor pueda identificar el comienzo o el fin del bloque de bits o trama. Consiste en adicionar un patrón de bits conocido (por ejemplo, 8 unos) llamado delimitador de trama. Una vez que el receptor detecta el delimitador, sabe que comienza o termina una trama. Una trama comienza con un conjunto de bits en una cabecera de control que incluye metadatos que dependen del protocolo de comunicación (por ejemplo, direcciones origen y destino, tipo prioridad de tráfico, tamaño de la trama o del bloque de datos, etc.), sigue con el bloque de datos y termina con otro conjunto de bits de una cola de control, con bits de verificación o checksum.

La figura 10.21 representa una trama.

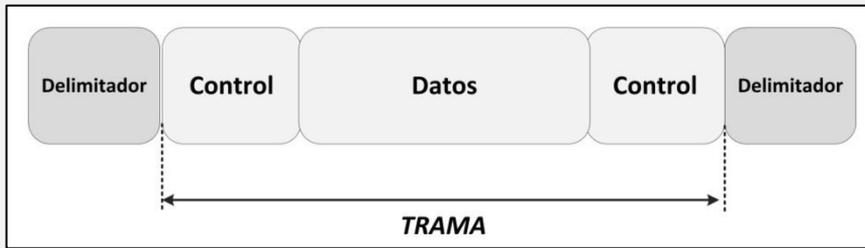


Fig. 10.21. Trama en transmisión serie síncrona.

La eficiencia de una transmisión síncrona, como la relación entre la cuenta de los bits efectivos de datos respecto a la cuenta total de bits, es significativamente mayor que para una asíncrona. Debe tenerse en cuenta que el bloque de datos contiene una cantidad de bits muy superior a los otros campos. Por ejemplo, para protocolos reales, los campos de control más el delimitador pueden ser de 48 bits, mientras que el bloque de datos puede ser del orden de los 8000 bits. La eficiencia en este caso sería entonces de:

$$\text{Eficiencia [\%]} = (8000/8048) \cdot 100 = 99,4 \%$$

Este resultado es muy superior al caso asíncrono.

## 2.5 Configuraciones punto-a-punto y multipunto

La conexión entre un módulo de E/S del computador y los periféricos pueden ser punto-a-punto o multipunto.

Una interface punto-a-punto proporciona una línea específica entre el módulo de E/S y el dispositivo externo (Figura 10.22). En las computadoras personales, la comunicación con el teclado, el monitor, entre otros dispositivos, es del tipo punto-a-punto.

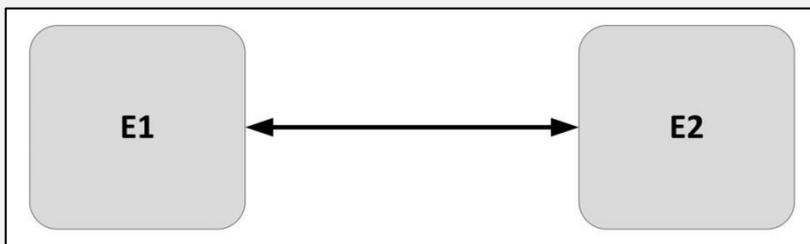


Fig. 10.22. Topología punto a punto.

Las interfaces externas multipunto son utilizadas para soportar varios dispositivos a través del mismo enlace. Estas interfaces multipunto son de hecho buses externos (Figura 10.23).

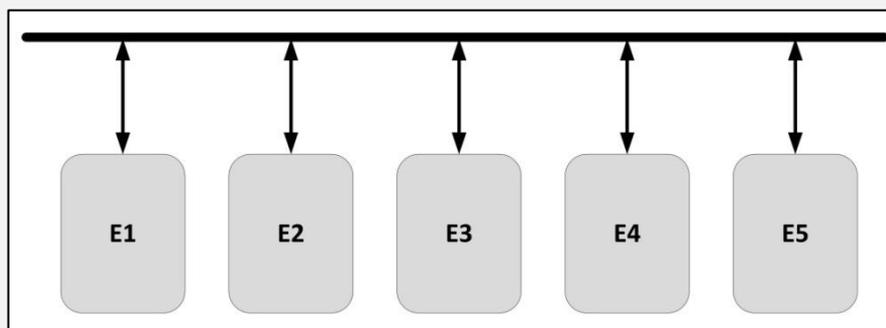


Fig. 10.23. Topología Multipunto.

## 2.6 Medios de transmisión

Los medios de transmisión utilizados para las comunicaciones de las computadoras con sus periféricos, y con otras computadoras a través de las redes de datos LAN Cableadas (Ethernet) o Inalámbricas (Wi-Fi), pueden clasificarse en

- Medios Guiados
- Medios no Guiados (Inalámbricos)

En ambos casos, la comunicación de la computadora con otros dispositivos se concreta con ondas electromagnéticas. Las características y calidad de la transmisión están determinadas tanto por el tipo de señal como las características del medio utilizado.

### 2.6.1 Medios guiados

Los medios guiados proporcionan un camino físico a través del cual la señal se propaga. En la actualidad, una computadora podría comunicarse básicamente con cualquiera de los siguientes medios guiados:

- Par Trenzado
- Fibra Óptica

## Par trenzado

El par trenzado consiste en dos cables embutidos en un aislante, entrecruzados en forma de espiral. Cada par de cables establece sólo un enlace de comunicación. En algunas aplicaciones se agrupan varios pares mediante una envoltura protectora (Figura 10.24).

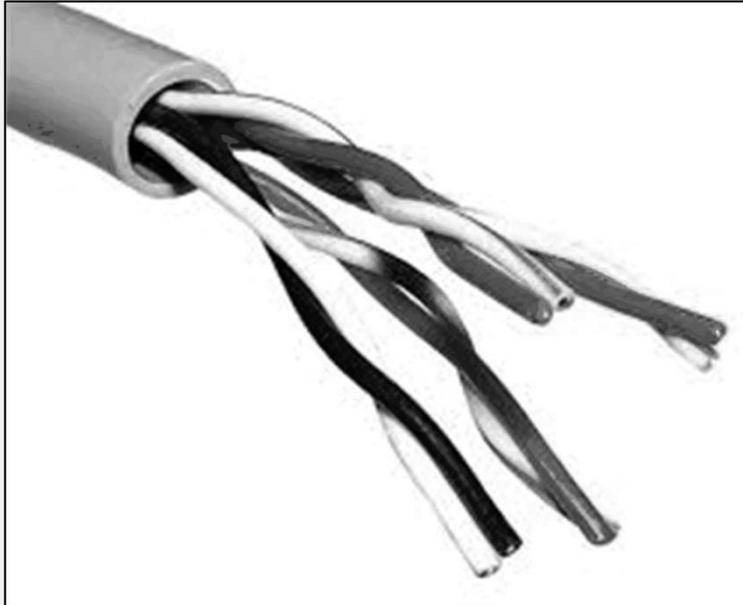


Fig. 10.24. Cable con varios pares trenzados.

Cuando el par trenzado se usa para llevar datos digitales, en distancia cortas, puede alcanzar velocidades de transmisión del orden de Gbps. Para largas distancias puede llegar a velocidades de transmisión del orden de Mbps.

Es el medio más barato y puede usarse en topologías punto a punto y multipunto.

Pueden usarse pares trenzados apantallados y sin apantallar. El no apantallado (UTP) se utiliza habitualmente en ambientes computacionales (Figura 10.25) (y en telefonía). Para evitar las interferencias electromagnéticas externas y de otros pares trenzados próximos, se usa una malla metálica que cubre todos los pares (STP) que mejora los resultados en ambientes críticos.



Fig. 10.25. Cable de par trenzado apantallado.

### **Fibra óptica**

La fibra óptica es un medio flexible capaz de transportar información digital mediante un haz de luz. También puede utilizar para las comunicaciones computacionales. En la Figura 10.26 se observa un patch cord de fibra.

El medio que transporta la luz (núcleo) es una (o varias) fibra de cristal o plástico de algunas micras de diámetro. La velocidad de transmisión puede llegar a cientos de Gbps para distancias de decenas de kilómetros. Es un cable liviano, y tiene menor atenuación y mejor aislamiento electromagnético, respecto a los pares trenzados.

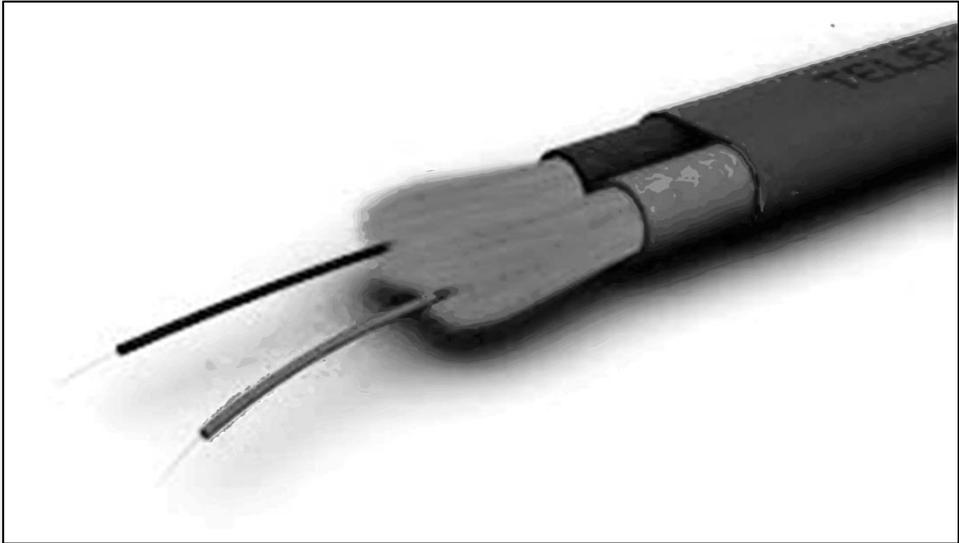


Fig. 10.26. Cable de fibra óptica.

### **2.6.2 Medios no guiados (inalámbricos)**

Los medios no guiados o inalámbricos utilizan una antena para transmitir y recibir a través del aire, el vacío, o aún en el agua.

En la transmisión, la antena radia energía electromagnética en el medio circundante, y en la recepción capta las ondas electromagnéticas del medio que la rodea.

Las comunicaciones inalámbricas se separan o clasifican en bandas de frecuencias, como se muestra en la Tabla 10.1.

Clasificación	Rango de frecuencias	Velocidades promedio
LF (frecuencia baja)	30 – 300 KHz	100 bps
MF (frecuencia media)	300 – 3000 KHz	1000 bps
HF (frecuencia alta)	3 – 30 MHz	3000 bps
VHF (frecuencia muy alta)	30 – 300 MHz	100 Kbps
UHF (frecuencia ultra alta)	300 – 3000 MHz	10 Mbps
SHF (frecuencia super alta)	3 – 30 GHz	100 Mbps
EHF (frecuencia extremadamente alta)	30 – 300 GHz	1000 Mbps

Tabla 10.1. Características de las bandas de comunicaciones.

Las comunicaciones inalámbricas también son usadas para las comunicaciones de una computadora, notebook, netbook, u otros dispositivos móviles, para transferencias de información entre ellos o para acceder a una red de datos (Figura 10.27). Las bandas de frecuencias normalmente están ubicadas en la banda SHF, también conocida como de microondas.



Fig. 10.27. Comunicaciones inalámbricas.

## 3 Casos de Estudio

### 3.1 Puerto USB

#### Visión general

El **Universal Serial Bus (USB)** es un estándar industrial desarrollado a mediados de los años 1990 que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre ordenadores y periféricos, y dispositivos electrónicos. En 1996 se lanzó la primera especificación (USB 1.0), la cual no fue popular, hasta 1998 con (USB 1.1).

USB fue diseñado para estandarizar la conexión de periféricos, como mouse, teclados, memorias USB (Figura 10.28), joysticks, escáneres, cámaras digitales, teléfonos móviles, reproductores multimedia, impresoras, dispositivos multifuncionales, sistemas de adquisición de datos, módems, tarjetas de red, tarjetas de sonido, tarjetas sintonizadoras de televisión y grabadora de DVD externa, discos duros externos y disquetera externas. Ha desplazado a conectores como el puerto serie, puerto paralelo, puerto de juegos, el Apple Desktop Bus o PS/2.



Fig. 10.28. Memoria USB.

Su campo de aplicación se extiende en la actualidad a cualquier dispositivo electrónico o con componentes, desde los automóviles a reproductores o los modernos juguetes. Se han implementado variaciones para su uso industrial e incluso militar. Pero en donde más se nota su influencia es en los teléfonos inteligentes (Europa ha creado una norma por la que todos los móviles deberán venir con un cargador microUSB), tabletas, PDAs y videoconsolas, donde ha reemplazado a conectores propietarios casi por completo.

### Aspectos técnicos

Un puerto USB permite conectar hasta 127 dispositivos y ya es un estándar en los ordenadores de última generación, que incluyen al menos cuatro puertos USB 3.0 en los más modernos, y algún USB 1.1 en los más anticuados.

Se trata de un puerto totalmente plug and play. Con sólo conectar el dispositivo es reconocido e instalado de manera inmediata.

A través del cable USB no sólo se transfieren datos; además es posible alimentar dispositivos externos. Una de las limitaciones de este tipo de conexiones es que la longitud del cable no debe superar los 5 metros, y que éste debe cumplir las especificaciones del Standard USB iguales para la 1.1 y la 2.0 (Figura 10.29).



Fig. 10.29. Cables USB.

Los dispositivos USB se clasifican en cuatro tipos según su velocidad de transferencia de datos:

- **Baja velocidad (1.0):** Tasa de transferencia de hasta 1,5 Mbit/s (188 kB/s). Utilizado en su mayor parte por dispositivos de interfaz humana como los teclados, los ratones o mouse, las cámaras web, etc.
- **Velocidad completa (1.1):** Tasa de transferencia de hasta 12 Mbit/s (1,5 MB/s) según este estándar. Ésta fue la más rápida antes de la especificación USB 2.0.
- **Alta velocidad (2.0):** Tasa de transferencia de hasta 480 Mbit/s (60 MB/s) pero con una tasa real práctica máxima de 280 Mbit/s (35 MB/s). El cable USB 2.0 dispone de cuatro líneas, un par para datos, y otro par de alimentación. Casi todos los dispositivos fabricados en la actualidad trabajan a esta velocidad
- **Superalta velocidad (3.0):** Tiene una tasa de transferencia de hasta 4,8 Gbit/s (600 MB/s). La velocidad del bus es diez veces más rápida que la del USB 2.0, debido a que han incluido 5 contactos adicionales, y es compatible con los estándares anteriores.

Las especificaciones USB 1.0, 1.1 y 2.0 definen dos tipos de conectores para conectar dispositivos al servidor: A y B. Sin embargo, la capa mecánica ha cambiado en algunos conectores. Por ejemplo, algunos fabricantes mantienen las señales y protocolos característicos del USB.

## 3.2 Puerto Ethernet

### Visión general

Es un puerto que viene integrado en la tarjeta principal o motherboard de una computadora, o bien en una tarjeta o placa de red (NIC - Network Interface Card). Se utiliza para interconectar la computadora con otras computadoras y dispositivos de red en una red LAN (Local Area Network) cableada. La Figura 10.30 muestra un puerto Ethernet para usar en una red LAN cableada con cable UTP.

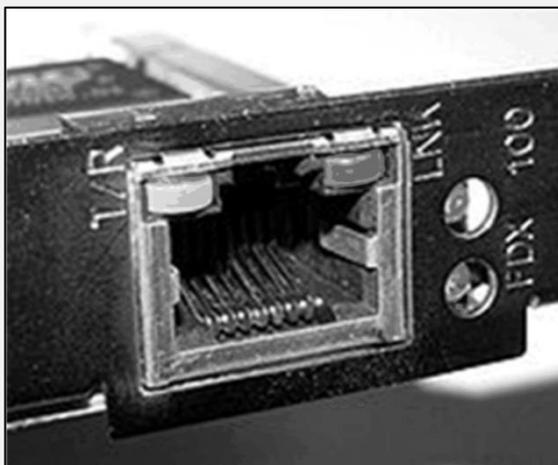


Fig. 10.30. Puerto Ethernet para cable UTP en placa de red.

El puerto Ethernet RJ-45 (Registered Jack 45) es una interfaz física que posee ocho pines o conexiones eléctricas, que se usan como extremos de cables de par trenzado UTP o STP (Figura 10.31).

Una variante es que el puerto Ethernet sea una interfaz física para cables de fibra óptica.



Fig. 10.31. Cable de par trenzado para puerto Ethernet.

## Aspectos técnicos

El puerto Ethernet es parte de una tarjeta de red o adaptador de red. La Figura 10.27 muestra el puerto Ethernet para cableado de par trenzado UTP. Permite la comunicación con aparatos conectados entre sí y también compartir recursos entre dos o más computadoras (discos duros, CD-ROM, impresoras, etc).

A las tarjetas de red también se les llama NIC (Network Interface Card). Hay diversos tipos de adaptadores en función del tipo de cableado o arquitectura que se utilice en la red, pero actualmente el más común es del tipo Ethernet utilizando una interface o conector RJ-45.

Aunque el término tarjeta de red se suele asociar a una tarjeta de expansión insertada en una ranura interna de un computador (como se observa en la Figura 10.32), también se suele utilizar para referirse a una placa integrada en la placa principal o madre de una computadora, notebook, netbook o tablets.

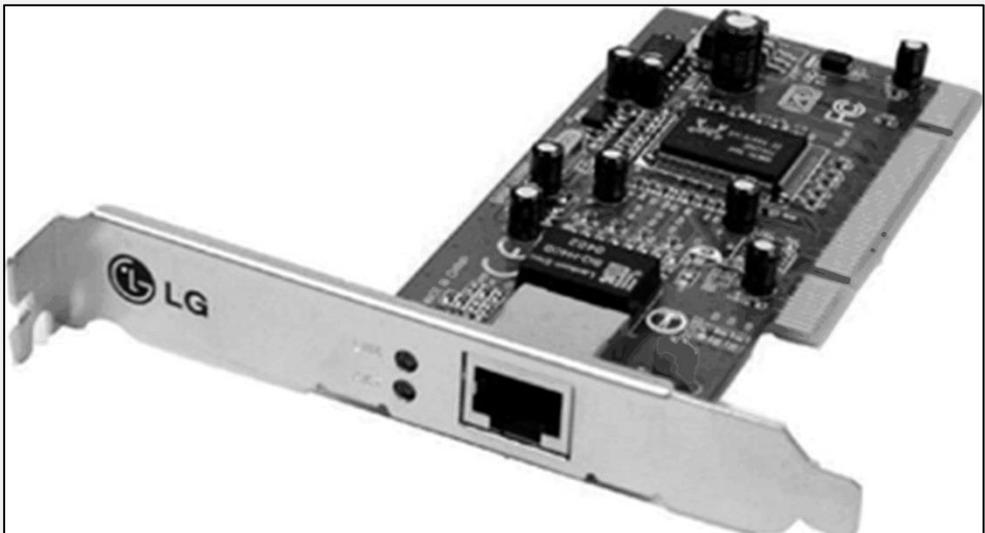


Fig. 10.32. Placa de red Ethernet con un puerto de par trenzado UTP.

Las tarjetas de red Ethernet para cable UTP utilizan conectores RJ-45, para comunicaciones a 10/100/1000 Mbps, aunque actualmente se están empezando a utilizar las 10 Gigabit Ethernet.

Alternativamente, se pueden usar puertos Ethernet para Fibra Óptica de 1000 Mbps, o superior (Figura 10.33). Uno de los conectores usuales para las aplicaciones de fibra óptica es el MM ST (Straight Tip fibra MultiModo).

Se trata de una solución simple para conectar una computadora directamente a una red de fibra óptica de alta velocidad, que ofrece una conexión de fibra directa que no es susceptible a interferencia electromagnética.

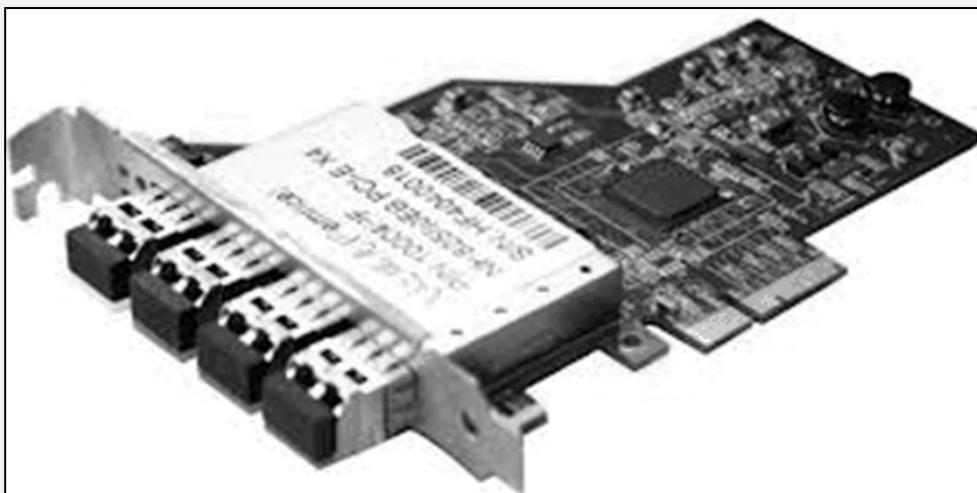


Fig. 10.33. Placa de red con un puerto Ethernet de fibra óptica.

### 3.3 Puerto Bluetooth

#### Visión general

El puerto BlueTooth es un puerto inalámbrico que viene integrado en la tarjeta principal o motherboard de una computadora. Se ha estandarizado su uso en los dispositivos móviles, y por lo tanto, también en notebooks, netbooks o tablets. Es posible que algunas computadoras también tengan este recurso de conectividad inalámbrica, ya sea de manera integrada o bien en una tarjeta o placa de red (NIC – Network Interface Card). Se utiliza para interconectar la computadora con otras computadoras y dispositivos de red en una red WPAN (Redes Inalámbricas de Área Personal).

Bluetooth es una especificación industrial que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un

enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles,
- Eliminar los cables y conectores entre éstos, y
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Los dispositivos que con mayor frecuencia utilizan esta tecnología son los siguientes dispositivos: PDA, teléfonos móviles, computadoras portátiles, computadoras, impresoras o cámaras digitales.

### Aspectos técnicos

Bluetooth fue diseñado especialmente para dispositivos de bajo consumo, que requieren corto alcance de emisión y basados en transceptores (transmisores y receptores) de bajo costo.

Los dispositivos que incorporan este protocolo pueden comunicarse entre ellos cuando se encuentran dentro de su alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión es suficiente. Estos dispositivos se clasifican como "Clase 1", "Clase 2" o "Clase 3" en referencia a su potencia de transmisión (Tabla 10.2).

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBm)	Alcance (aproximado)
Clase 1	100 mW	20 dBm	~ 30 metros
Clase 2	2,5 mW	4 dBm	~ 10-5 metros
Clase 3	1 mW	0 dBm	~ 1 metro

Tabla 10.2. Clases de Bluetooth.

En la mayoría de los casos, la cobertura efectiva de un dispositivo de clase 2 se extiende cuando se conecta a un transceptor de clase 1. Esto es así gracias a la mayor sensibilidad y potencia de transmisión del dispositivo de clase 1.

Los dispositivos con Bluetooth también pueden clasificarse según su capacidad de canal, como se observa en la Tabla 10.3.

<b>Versión</b>	<b>Ancho de banda</b>
<b>Versión 1.2</b>	<b>1 Mbit/s</b>
<b>Versión 2.0 + EDR</b>	<b>3 Mbit/s</b>
<b>Versión 3.0 + HS</b>	<b>24 Mbit/s</b>
<b>Versión 4.0</b>	<b>24 Mbit/s</b>

Tabla 10.3. Versiones y velocidad en Bluetooth.

La utilidad Bluetooth fue desarrollada en 1994 por Jaap Haartsen y Mattisson Sven, como reemplazo de cable, que estaban trabajando para Ericsson en Lund, Suecia.

Las prestaciones fueron publicadas por el Bluetooth Special Interest Group (SIG). El SIG las anunció formalmente el 20 de mayo de 1998. Hoy cuenta con una membresía de más de 20.000 empresas en todo el mundo. Fue creado por Ericsson, IBM, Intel, Toshiba y Nokia, y posteriormente se sumaron muchas otras compañías. Todas las versiones de los estándares de Bluetooth están diseñadas para la retro compatibilidad, que permite que el último estándar cubra todas las versiones anteriores. Las versiones más recientes son la 3.0 y la 4.0

La versión 3.0 + HS de la especificación Core Bluetooth fue aprobada por el Bluetooth SIG en 2009. Soporta velocidades teóricas de transferencia de datos de hasta 24 Mbit/s entre sí, aunque no a través del enlace Bluetooth propiamente dicho. La conexión Bluetooth nativa se utiliza para la negociación y el establecimiento mientras que el tráfico de datos de alta velocidad se realiza mediante un enlace Wi-Fi. Justamente, su principal novedad es AMP (Alternate MAC/PHY), es decir, la adición de Wi-Fi como transporte de alta velocidad.

En la especificación, la incorporación de la transmisión a alta velocidad no es obligatoria, y por lo tanto, los dispositivos marcados con "+ HS" incorporan el enlace Wi-Fi de alta velocidad de transferencia de datos. Mientras que un dispositivo Bluetooth 3.0, sin el sufijo "+ HS" no soporta alta velocidad.

El SIG de Bluetooth completó en 2010 la especificación de un nuevo núcleo de Bluetooth en su versión 4.0, que incluye al Bluetooth clásico, el Bluetooth de alta velocidad y los protocolos Bluetooth de bajo consumo. El bluetooth de alta velocidad se basa en Wi-Fi, y el Bluetooth clásico consta de protocolos Bluetooth preexistentes. El bluetooth de baja energía (Bluetooth Low Energy o BLE) es un subconjunto de Bluetooth v4.0 con una pila de protocolos completamente nueva para desarrollar rápidamente enlaces sencillos. Como alternativa a los protocolos estándar de Bluetooth que se introdujeron en Bluetooth v1.0 a v4.0, está dirigido a aplicaciones de muy baja potencia alimentados con una pila botón.

### 3.4 Puerto HDMI

#### Visión general

El puerto HDMI (High-Definition Multimedia Interface), o Interface multimedia de alta definición, es una norma de audio y vídeo digital cifrado sin compresión. HDMI provee una interface (Figura 10.34) entre cualquier fuente de audio y vídeo digital como podría ser una computadora, un sintonizador TDT, un reproductor de Blu-ray, una Tablet PC, y un monitor de audio/vídeo digital compatible, como un televisor digital (DTV).

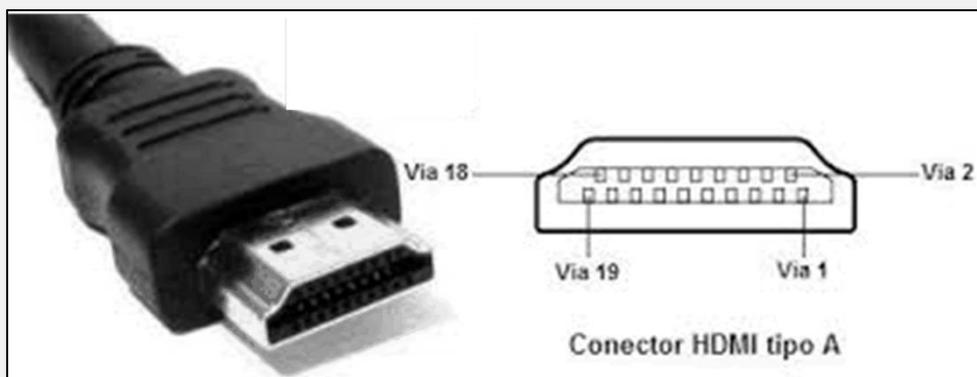


Fig. 10.34. Conector de la interface HDMI.

HDMI permite el uso de vídeo computarizado, mejorado o de alta definición, así como audio digital multicanal en un único cable.

## Aspectos técnicos

El conector estándar de HDMI tipo A tiene 19 pines (Tabla 10.4). Se ha definido también una versión de mayor resolución -tipo B-, pero su uso no se ha generalizado. El tipo B tiene 29 pines, permitiendo llevar un canal de vídeo expandido para pantallas de alta resolución. Este último fue diseñado para resoluciones más altas, es decir, mayor tamaño de imagen.

El HDMI tipo A es compatible hacia atrás con un enlace simple DVI, usado por los monitores de computadoras y tarjetas gráficas modernas. Esto quiere decir que una fuente DVI puede conectarse a un monitor HDMI, o viceversa, por medio de un adaptador o cable adecuado (el audio y las características de control remoto HDMI no estarán disponibles). El HDMI tipo B es, de forma similar, compatible hacia atrás con un enlace DVI.

<b>Pin</b>	<b>Asignación de señal</b>	<b>Pin</b>	<b>Asignación de señal</b>
<b>1</b>	<b>TMDS Data2+</b>	<b>1</b>	<b>TMDS Data2 Shield</b>
<b>3</b>	<b>TMDS Data2-</b>	<b>3</b>	<b>TMDS Data1+</b>
<b>5</b>	<b>TMDS Data1 Shield</b>	<b>5</b>	<b>TMDS Data1-</b>
<b>7</b>	<b>TMDS Data0+</b>	<b>7</b>	<b>TMDS Data0 Shield</b>
<b>9</b>	<b>TMDS Data0-</b>	<b>9</b>	<b>TMDS Clock+</b>
<b>11</b>	<b>TMDS Clock Shield</b>	<b>11</b>	<b>TMDS Clock-</b>
<b>13</b>	<b>CEC</b>	<b>13</b>	<b>Reservado</b>
<b>15</b>	<b>SCL</b>	<b>15</b>	<b>SDA</b>
<b>17</b>	<b>Tierra DDC/CEC</b>	<b>17</b>	<b>+5V Power</b>
<b>19</b>	<b>Detección de conexión “en caliente” (Hot Plug)</b>	<b>19</b>	

Tabla 10.4. Asignación de pines de un conector HDMI tipo A.

El conector HDMI ha sido diseñado para que los equipos que lo utilicen impidan al usuario realizar copia del contenido de audio-vídeo transmitido, mediante el cifrado de dichos datos.

Desde la aparición de la primera versión de HDMI en 2002, se han sucedido sucesivas revisiones que se resumen en la Tabla 10.5.

La especificación HDMI no define una longitud máxima del cable. Al igual que con todos los cables, la atenuación de la señal se hace demasiado alta a partir de una determinada longitud. En lugar de ello, HDMI especifica un mínimo nivel de potencia. Diferentes materiales y calidades de construcción permitirán cables de diferentes longitudes. Además, el mejor cumplimiento de los requisitos técnicos de los cables permitirá soportar los formatos de vídeo de mayor resolución. La atenuación de la señal y la interferencia causada por los cables pueden ser compensadas mediante la utilización de ecualizadores.

En la norma HDMI 1.3 fueron definidas dos categorías de cables llamados Categoría 1 (Estándar de HDTV) y Categoría 2 (de alta velocidad o superior que la HDTV) para reducir la confusión acerca de cuáles son los cables que dan soporte a distintos formatos de vídeo. Usando conductores de calibre 28 AWG, un cable de 5 metros se puede fabricar de manera fácil y económica para las especificaciones de la categoría 1. Un cable con conductores de mayor grosor, como 24 AWG, de construcción más estricta en cuanto a tolerancias y otros factores, puede alcanzar longitudes de 12 a 15 metros. Además, otros cables (fibra óptica o de doble cable Cat-5 en vez del estándar de cobre) se pueden utilizar para ampliar HDMI a 100 metros o más. Algunas compañías también ofrecen amplificadores, ecualizadores y repetidores que pueden encadenar varios cables HDMI.

Revisión HDMI	1.0	1.1	1.2/1.2a	1.3/1.3a/1.3b	1.4/1.4a/1.4b
Máximo ancho de banda de señal (MHz)	165	165	165	340	340
Máximo ancho de banda TMDS total (Gbit/s)	4.95	4.95	4.95	10.2	10.2
Máximo ancho de banda de video (Gbit/s)	3.96	3.96	3.96	10.2	10.2
Máximo ancho de banda de audio (Mbit/s)	36.86	36.86	36.86	36.86	36.86
Resoluciones posibles sobre una señal simple HDMI a 24bits por pixel	1920x1080p @ 60	1920x1080p @ 60	1920x1080p @ 60	2560x1440p @ 75	4096x2160p @ 24
Máxima profundidad de color (bits por pixel)	24	24	24	48*	48*
8 canales/ 192KHz/ capacidad de 24-bit de audio	si	si	si	si	si

Tabla 10.5. Resumen de las sucesivas revisiones de HDMI.

## 3.5 Puerto Wi-Fi

### Visión general

El puerto Wi-Fi es un puerto inalámbrico que viene integrado en la tarjeta principal o motherboard de una computadora. Se ha estandarizado su uso en los dispositivos móviles, y por lo tanto, también en notebooks, netbooks, tablets y smartphones. Las computadoras pueden tener este recurso de conectividad inalámbrica, ya sea de manera integrada, en una tarjeta o placa de red (NIC – Network Interface Card). Aunque también existen placas de red Wi-Fi con conexión USB. Se utiliza para interconectar la computadora con otras computadoras y dispositivos de red en una red WLAN (Wireless LAN).

Las NIC Wi-Fi vienen en diferentes variedades dependiendo de la norma a la cual se ajustan. Usualmente son 802.11a, 802.11b, 802.11g y 802.11n. En el tiempo, las más populares han sido la 802.11b que transmite a 11 Mbit/s (1,375 MB/s), y la 802.11g que transmite a 54

Mbit/s (6,75 MB/s). Actualmente, el protocolo que se viene utilizando es 802.11n que es capaz de transmitir 600 Mbit/s. Actualmente la capa física soporta una velocidad de teórica de hasta 300 Mbit/s.

### Aspectos técnicos

Esta tecnología surgió por la necesidad de establecer un mecanismo de conexión inalámbrica que fuese compatible entre distintos dispositivos. Buscando esa compatibilidad fue que en 1999 importantes empresas se reunieron para crear la Wireless Ethernet Compatibility Alliance, o WECA, actualmente llamada Wi-Fi Alliance. El objetivo de la misma fue designar una marca que permitiese fomentar más fácilmente la tecnología inalámbrica y asegurar la compatibilidad de equipos.

La norma inalámbrica IEEE 802.11 fue diseñada para sustituir el equivalente cableado IEEE 802.3 (Ethernet). Esto quiere decir que en lo único que se diferencia una red Wi-Fi de una red Ethernet es en cómo se transmiten las tramas o paquetes de datos; el resto es idéntico. Por tanto, una red local inalámbrica 802.11 es completamente compatible con todos los servicios de las redes locales (LAN) de cable 802.3 (Ethernet).

Existen diversos tipos de Wi-Fi, basado cada uno de ellos en el estándar original IEEE 802.11. Son los siguientes:

- Los estándares IEEE 802.11b, IEEE 802.11g e IEEE 802.11n disfrutaron de una aceptación internacional debido a que la banda de 2.4 GHz está disponible casi universalmente, con una velocidad de hasta 11 Mbit/s, 54 Mbit/s y 300 Mbit/s, respectivamente.
- El estándar IEEE 802.11a, conocido como WIFI 5, que opera en la banda de 5 GHz y que disfruta de una operatividad con canales relativamente limpios. La banda de 5 GHz ha sido recientemente habilitada y, además, no existen otras tecnologías (Bluetooth, microondas, ZigBee, WUSB) que la estén utilizando (por lo tanto, existen muy pocas interferencias). Su alcance es algo menor que el de los estándares que trabajan a 2.4 GHz (aproximadamente un 10%), debido a que la frecuencia es mayor (a mayor frecuencia, menor alcance).

Existen otras tecnologías inalámbricas como Bluetooth que también funcionan a una frecuencia de 2.4 GHz, por lo que puede presentar interferencias con la tecnología Wi-Fi. Debido a esto, en la versión 1.2 del estándar Bluetooth por ejemplo se actualizó su especificación para que no existieran interferencias con la utilización simultánea de ambas tecnologías.

Uno de los problemas a los cuales se enfrenta actualmente la tecnología Wi-Fi es la progresiva saturación del espectro radioeléctrico, debido a la masificación de usuarios. Esto afecta especialmente en las conexiones de larga distancia (mayor de 100 metros). Aunque debe tenerse en cuenta que Wi-Fi fue diseñado para conectar computadoras (u otros dispositivos móviles) a la red a distancias reducidas.

Existen varios dispositivos Wi-Fi, los cuales se pueden dividir en dos grupos:

- Dispositivos de Distribución o Red, entre los que destacan los routers, puntos de acceso (APs) y Repetidores Wi-Fi (Figura 10.35), para dar servicio Wi-Fi a los dispositivos terminales, y
- Dispositivos Terminales que son aquellos equipos que tienen una NIC Wi-Fi (placa de red Wi-Fi), ya sean internas (tarjetas PCI) o bien USB, como el caso de las computadoras personales u otros dispositivos móviles.



Fig. 10.35. Router Wi-Fi.

- **Dispositivos de Distribución o Red:**
  - **Los puntos de acceso o APs** son dispositivos que generan un "set de servicios", que podría definirse como una "Red Wi-Fi" a la que se pueden conectar otros dispositivos. Los APs permiten conectar dispositivos en forma inalámbrica a una

red existente. Pueden agregarse más APs a una red para generar redes de cobertura más amplia, o conectar antenas más grandes que amplifiquen la señal.

- **Los repetidores inalámbricos** son equipos que se utilizan para extender la cobertura de una red inalámbrica. Éstos se conectan a una red existente que tiene señal más débil y crean una señal limpia a la que se pueden conectar los equipos dentro de su alcance. Algunos de ellos funcionan también como punto de acceso.
- **Los router inalámbricos** son dispositivos compuestos, especialmente diseñados para redes pequeñas (hogareñas o una pequeña oficina). Estos dispositivos incluyen, un Router (encargado de interconectar redes, por ejemplo, nuestra red del hogar con internet), un punto de acceso (explicado más arriba) y generalmente un switch que permite conectar algunos equipos vía cable (Ethernet y USB). Su tarea es tomar la conexión a internet, y brindar a través de ella acceso a todos los equipos que conectemos, sea por cable o en forma inalámbrica.
- **Los dispositivos terminales** se clasifican según el tipo de tarjeta Wi-Fi con la que consiguen servicio de red Wi-Fi. Existen tres tipos mayoritarios: tarjetas PCI, tarjetas PCMCIA y tarjetas USB:
  - **Con tarjetas PCI** para Wi-Fi se agregan (o vienen de fábrica) en las computadoras personales. Están perdiendo terreno debido a las tarjetas USB. Dentro de este grupo también pueden agregarse las tarjetas MiniPCI que vienen integradas en casi cualquier computadora portátil disponible hoy en el mercado.
  - **Con tarjetas PCMCIA** son un modelo que se utilizó mucho en las primeras computadoras personales, aunque están cayendo en desuso, debido a la integración de tarjeta inalámbricas internas en estas computadoras.
  - **Con tarjetas USB** para Wi-Fi son el tipo de tarjeta más común que existe para conectar a una PC, ya sea personal o portátil, haciendo uso de todas las ventajas que tiene la tecnología USB. Hoy en día pueden encontrarse incluso tarjetas USB con el estándar 802.11n que es el último estándar liberado para redes inalámbricas.

También existen impresoras, cámaras Web y otros periféricos que funcionan con la tecnología Wi-Fi, permitiendo un ahorro de mucho cableado en las instalaciones de redes y especialmente, gran movilidad.

## 4 Ejercitación

### **Ejercicio 1:**

Una computadora capaz de ejecutar  $10^7$  instrucciones por segundo. Se desea conectar al computador, únicamente un periférico con una velocidad de transferencia de 20.000 bytes/sg. y sobre el que se realizan operaciones de lectura de bloques de 1.024 bytes. Se pretende ver el comportamiento de la pareja computador-periférico ante las diferentes técnicas de entrada-salida (programada, mediante interrupciones y por DMA)

Se sabe que:

- La rutina de transferencia de E/S programada consta de 10 instrucciones.
- La rutina de tratamiento de interrupción en la E/S mediante interrupciones consta de 20 instrucciones.
- La rutina de inicialización del DMA consta de 8 instrucciones. Y en cada operación de escritura de un dato en memoria el controlador ocupa los buses durante 500 ns.

Se pide:

Indicar el número de instrucciones de otros procesos que puede realizar el computador durante cada uno de los tipos de E/S previstos.

### **Ejercicio 2:**

Realizar el ejercicio anterior suponiendo que la velocidad de transferencia del periférico es de 100.000 bytes por segundo y los bloques de 512 bytes.

### **Ejercicio 3:**

Si tenemos una computadora que puede ejecutar 100.000 instrucciones en el tiempo que se tarda en leer un bloque de 2048 bytes, indicar cuantas de esas instrucciones quedarían disponibles para otros procesos si le conectamos un sistema de E/S mediante DMA. Se supone que la rutina de inicialización del DMA consta de 8 instrucciones, y que en cada operación de escritura de un dato en memoria el controlador ocupa los buses durante 750 nseg.

### **Ejercicio 4:**

Un sistema controlado por un operador a través de una serie de comandos que se introducen desde un teclado. En cada intervalo de ocho horas se introducen un promedio de sesenta comandos.

- a) Suponga que el procesador comprueba el teclado cada 100 ms.  
¿Cuántas veces se chequea en un periodo de ocho horas?
- b) ¿En qué porcentaje se reduciría el número de comprobaciones de teclado si se utilizase E/S por interrupciones.

**Ejercicio 5:**

Un módulo de DMA transfiere caracteres a memoria mediante robo de ciclo desde un dispositivo que transmite a 9600 bps. El procesador ejecuta instrucciones a un ritmo de un millón por segundo. ¿Cuánto disminuye la velocidad del procesador debido al DMA?

**Ejercicio 6:**

Considere un sistema en el que una transferencia a través de un bus necesita 500 ns. La transferencia de control del bus en uno u otro sentido, entre el procesador y un dispositivo de E/S, necesita 250 ns. Uno de los dispositivos de E/S tiene una velocidad de transferencia de 50 Kbytes/s y utiliza DMA. Los datos se transfieren byte a byte.

- a) Suponga que se emplea DMA en modo ráfaga. Es decir, la interface de DMA adquiere el control de bus antes de empezar la transferencia de un bloque y mantiene dicho control durante la transferencia completa. ¿Durante cuánto tiempo el dispositivo tiene el bus ocupado si se transfieren 128 bytes?
- b) Repita el cálculo si se utiliza el modo de robo de ciclo.

**Ejercicio 7:**

¿Cuántos puertos de E/S puede direccionar el 8088?. Desarrolle.

**Ejercicio 8:**

¿Cuáles son las ventajas del 8088 por tener las E/S mapeadas en memoria? Desarrolle y analice cuál es la opción que no utiliza el 8088.

## Índice de contenidos:

### Capítulo 1 Representación Numérica

<b>1 Sistemas de Numeración</b> .....	12
1.1 Introducción .....	12
1.2 Confiabilidad .....	14
1.3 Costo .....	15
<b>2 Sistema de Numeración Binario</b> .....	16
2.1 Introducción .....	16
2.2 Conversión entre números de distintas bases .....	18
2.3 Complementos binarios .....	19
2.4 Representación de números negativos en binario .....	21
<b>3 Punto Fijo y Punto Flotante</b> .....	23
3.1 Introducción .....	23
3.2 Operaciones aritméticas .....	24
3.3 Norma IEEE 754 .....	25
<b>4 Ejercitación</b> .....	27

### Capítulo 2 Códigos Numéricos y Alfanuméricos

<b>1 Códigos</b> .....	30
1.1 Introducción .....	30
1.2 Códigos binarios .....	30
1.3 Códigos BCD.....	33
1.4 Códigos alfanuméricos .....	35
<b>2 Códigos detectores y correctores de error</b> .....	37
2.1 Introducción .....	37
2.2 Distancia mínima.....	38
2.3 Códigos detectores de error .....	40
2.4 Códigos correctores de error .....	41
2.4.1 Código de Hamming .....	42
2.4.2 Verificación de redundancia LRC y CRC.....	44
2.4.3 Códigos bidimensionales .....	47
<b>3 Encriptación o cifrado</b> .....	49
<b>4 Otros códigos</b> .....	<b>51</b>
4.1 Códigos de barras .....	51
4.2 Códigos QR.....	55
<b>5 Ejercitación</b> .....	60

### Capítulo 3 Álgebra de Boole

<b>1 Visión General del Álgebra de Boole</b> .....	64
1.1 Introducción .....	64
1.2 Postulados .....	65

1.3 Teoremas .....	65
<b>2 Funciones lógicas</b> .....	67
2.1 Introducción .....	67
2.2 Teoremas de Funciones Lógicas.....	67
<b>3 Minimización de Funciones Lógicas</b> .....	71
3.1 Introducción .....	71
3.2 Método de Simplificación de Karnaugh .....	72
<b>4 Compuertas Lógicas</b> .....	75
<b>5 Ejercitación</b> .....	77

## Capítulo 4 Sistemas Combinacionales

<b>1 Sistemas Digitales</b> .....	82
<b>2 Sistemas Combinacionales</b> .....	83
2.1 Introducción .....	83
2.2 Circuitos Combinacionales MSI .....	84
<b>3 Casos Comunes de Sistemas Combinacionales MSI</b> .....	84
3.1 Codificadores .....	84
3.2 Decodificadores.....	85
3.3 Multiplexores .....	87
3.4 Demultiplexores .....	89
3.5 Comparadores .....	91
3.6 Detectores/Generadores de Paridad .....	92
3.7 Sumadores .....	93
3.8 Unidades Aritméticas y Lógicas .....	97
<b>4 Ejercitación</b> .....	98

## Capítulo 5 Sistemas Secuenciales

<b>1 Visión general de los sistemas secuenciales</b> .....	102
1.1 Introducción .....	102
1.2 Un Caso de Estudio .....	104
<b>2 Biestables</b> .....	108
2.1 Introducción .....	108
2.2 Biestables Asíncronos .....	109
2.3 Biestables Asíncronos .....	109
<b>3 Tipos de Biestables</b> .....	112
3.1 Introducción .....	112
3.2 Biestables JK.....	112
3.3 Biestables T .....	113
3.4 Biestables D.....	113
<b>4 Aplicaciones de los Biestables</b> .....	114
4.1 Registros de Desplazamiento .....	114
4.2 Transferencias entre Registros.....	116
4.3 Contadores .....	119
4.4 Multiplicación y División Binaria .....	121

4.4.1 Multiplicación binaria .....	121
4.4.2 División binaria.....	127
<b>5 Ejercitación</b> .....	130

## Capítulo 6 Memorias Electrónicas

<b>1 Visión general</b> .....	134
1.1 Introducción .....	134
1.2 Clasificación de las memorias electrónicas .....	137
<b>2 Memorias de acceso aleatorio (RAM)</b> .....	138
2.1 Definición .....	138
2.2 Memorias RAM de lectura/escritura .....	141
2.2.1 Memorias RAM de lectura/escritura estáticas .....	141
2.2.2 Memorias RAM de lectura/escritura dinámicas .....	146
2.3 Memorias RAM de sólo lectura (ROM) .....	148
2.3.1 Memorias ROM .....	148
2.3.2 Memorias PROM .....	149
2.3.3 Memorias R PROM .....	149
2.4 Extensión de longitud de palabra y de capacidad .....	150
2.4.1 Extensión de longitud de palabra.....	150
2.4.2 Extensión de número de palabras .....	151
<b>3 Memorias de acceso serie</b> .....	153
3.1 Definición .....	153
3.2 Registros de desplazamiento .....	155
3.2.1 Registros de desplazamiento estáticos.....	156
3.2.2 Registros de desplazamiento dinámicos .....	156
3.3 Memorias FIFO .....	158
3.4 Memorias LIFO .....	160
<b>4 Ejercitación</b> .....	162

## Capítulo 7 Arquitectura Básica de una Computadora

<b>1 Arquitectura de Von Neumann</b> .....	166
<b>2 Máquina elemental</b> .....	168
2.1 Introducción .....	168
2.2 Arquitectura de la Computadora Elemental .....	171
2.2.1 Unidad de Procesamiento Central .....	171
2.2.2 Memorias RAM de lectura/escritura dinámicas .....	172
2.2.3 Unidad de Entrada/Salida.....	172
2.3 Conjunto de Instrucciones.....	173
2.4 El Ciclo de Máquina .....	177
2.5 Flujo de Información .....	177
2.6 Unidad de Control.....	180
2.6.1 Unidad de Control Cableada.....	180
2.6.2 Secuenciador .....	182
2.6.3 Lógica de Control .....	189
2.7 Unidad de control microprogramada.....	190

2.8 Bus en la máquina elemental .....	198
2.9 Unidad aritmética y lógica .....	200
<b>3 Ejercitación</b> .....	<b>202</b>

## **Capítulo 8 Arquitectura Convencional**

<b>1 Visión General</b> .....	<b>206</b>
1.1 Formato de Instrucciones .....	206
1.1.1 Formato de cuatro direcciones .....	206
1.1.2 Formato de tres direcciones .....	207
1.1.3 Formato de dos direcciones .....	207
1.1.4 Formato de una dirección .....	208
1.2 Modos de direccionamiento .....	208
1.2.1 Operando en la CPU .....	210
1.2.2 Operando en la memoria .....	210
<b>2 Nuevo Hardware y Nuevo Software</b> .....	<b>212</b>
2.1 Registros nuevos .....	212
2.1.1 Registros índices .....	212
2.1.2 Registros base .....	213
2.2 Máquina Elemental Indexada .....	215
2.2.1 Conj. instrucciones de la máquina elemental indexada .....	216
2.2.2 Ciclos de máquina .....	220
2.2.3 Interrupciones .....	228
2.2.4 Sistema elemental de interrupciones .....	230
2.2.5 Inicio de una transferencia .....	237
2.3 Hacia una Estructura Convencional .....	243
<b>3 Microprocesador Intel 8088</b> .....	<b>244</b>
3.1 Introducción .....	244
3.2 Diagrama en Bloques .....	246
3.2.1 BIU Y EU .....	247
3.2.2 Registros del 8088 .....	249
3.2.3 Organización de la memoria .....	250
3.2.4 Modos de direccionamiento .....	252
3.2.5 Conjunto de instrucciones .....	254
3.2.6 Direccionamiento de Entrada/Salida .....	255
3.2.7 Interrupciones del 8088 .....	255
<b>4 Ejercitación</b> .....	<b>257</b>

## **Capítulo 9 Arquitectura Avanzada**

<b>1 Visión general</b> .....	<b>260</b>
<b>2 Pipeline</b> .....	<b>261</b>
2.1 Introducción .....	261
2.2 Predicción de la Dirección de Salto .....	262
2.2 Pipeline en la Máquina Elemental .....	264
2.3.1 Unidad de control con pipeline .....	264

2.3.2	Secuenciador de la UC con pipeline .....	265
2.3.3	Ciclos de Pipeline .....	268
2.3.4	Incrementador del contador de programa.....	272
2.3.5	Predicción de la dirección de salto .....	272
2.3.6	Comparación con la máquina elemental.....	272
<b>3</b>	<b>Memoria Caché .....</b>	<b>273</b>
3.1	Principios de localidad .....	273
3.2	Manejo de la caché.....	274
<b>4</b>	<b>DMA.....</b>	<b>274</b>
4.1	Controlador DMA .....	274
4.2	Scanner .....	276
4.2.1	Robo de ciclo.....	276
<b>5</b>	<b>Evolución de las Arquitecturas .....</b>	<b>277</b>
5.1	Introducción .....	277
5.2	CISC .....	278
5.3	RISC .....	279
5.3.1	Arquitectura de carga/almacenamiento .....	280
5.3.2	Registros múltiples.....	282
5.4	Comparación entre RISC y CISC.....	283
5.4.1	Desde la semántica de los programas de alto nivel .....	283
5.4.2	Considerando la transferencia de datos entre la CPU y la memoria.....	284
5.4.3	Comportamiento en el salto a subrutinas e interrupciones .....	286
5.4.4	Cuadro comparativo entre RISC y CISC .....	286
<b>6</b>	<b>Evolución desde el procesador 8088 .....</b>	<b>287</b>
6.1	Funcionamiento básico del procesador Intel 80486 .....	290
6.2	Funcionamiento básico del procesador Intel Pentium .....	293
<b>7</b>	<b>Ejercitación.....</b>	<b>295</b>

## **Capítulo 10 Entradas y Salidas (E/S)**

<b>1</b>	<b>Módulos, Canales y Procesadores de E/S .....</b>	<b>298</b>
1.1	Introducción .....	298
1.2	Módulos de E/S .....	300
1.3	Diagrama en bloques de un módulo de E/S .....	301
1.4	Técnicas para las operaciones de E/S.....	302
1.5	Ejemplo de módulo de E/S.....	303
1.6	Canales y procesadores de E/S .....	305
<b>2</b>	<b>Datos, Señales e Interfaces .....</b>	<b>308</b>
2.1	Datos y Señales .....	308
2.2	Ancho de Banda.....	309
2.3	Señalización.....	310
2.4	Interface serie y paralela .....	311
2.4.1	Transmisión paralela.....	312
2.4.2	Transmisión serie.....	315
2.5	Configuraciones punto-a-punto y multipunto .....	320

2.6 Medios de transmisión .....	321
2.6.1 Medios guiados .....	321
2.6.2 Medios no guiados (inalámbricos) .....	324
<b>3 Casos de Estudio</b> .....	326
3.1 Puerto USB .....	326
3.2 Puerto Ethernet .....	328
3.3 Puerto Bluetooth .....	331
3.4 Puerto HDMI .....	334
3.5 Puerto Wi-Fi .....	337
<b>4 Ejercitación</b> .....	341
Índice de contenidos .....	343
Índice de Figuras .....	349
Índice de Tablas .....	355
Índice de Cuadros .....	357

## Índice de Figuras:

### Capítulo 1 Representación Numérica

Figura 1.1 Sistemas físicos que representan números .....	14
Figura 1.2 Representación de números reales en punto flotante .....	23
Figura 1.3 Representaciones de punto flotante de la Norma 754 .....	26
Figura 1.4 Casos especiales de números en punto flotante .....	26

### Capítulo 2 Códigos Numéricos y Alfanuméricos

Figura 2.1 Relación biunívoca de un código .....	30
Figura 2.2 Esquema sobre la capacidad de detección y corrección de error de un código .....	37
Figura 2.3 Estructura general de mensaje de un protocolo .....	38
Figura 2.4 Esquema sobre la capacidad de detección y corrección de error de un código .....	39
Figura 2.5 Figura Esquema de un enlace de datos .....	44
Figura 2.6 Organización de códigos bidimensionales .....	47
Figura 2.7 Organización de códigos bidimensionales usando Hamming y bit de paridad .....	48
Figura 2.8 Simbología EAN/UPC .....	53
Figura 2.9 Simbología Código 39 .....	54
Figura 2.10 Simbología Codabar .....	54
Figura 2.11 Simbología I 2/5 .....	54
Figura 2.12 Simbología Código 93 .....	55
Figura 2.13 Simbología Código 128 .....	55
Figura 2.14 Ejemplo de Código QR .....	56
Figura 2.15 Ejemplo de Código QR en ticket de acceso a un evento ....	57
Figura 2.16 Código QR usado por Wikipedia .....	57
Figura 2.17 Ejemplo de Código QR usado en cartel comercial .....	58
Figura 2.18 Ejemplo de Código QR .....	59
Figura 2.19 Ejemplo de Código QR .....	59
Figura 2.20 Ejemplo de Código QR .....	60

### Capítulo 3 Álgebra de Boole

Figura 3.1 Mapa de Karnaugh para funciones de 4 variables .....	73
Figura 3.2 Mapa de Karnaugh de la función ejemplo .....	74
Figura 3.3 Listado de compuertas lógicas más comunes .....	75
Figura 3.4 Implementación con compuertas lógicas de la función sin minimizar .....	76

## Capítulo 4 Sistemas Combinacionales

Figura 4.1 Diagrama de un sistema digital .....	82
Figura 4.2 Correspondencias para un Sistema Combinacional .....	83
Figura 4.3 Correspondencias para un Sistema Secuencial .....	83
Figura 4.4 Diagrama en bloque de un sistema o circuito Combinacional .....	84
Figura 4.5 Codificador binario de 8 entradas y 3 salidas .....	85
Figura 4.6 Decodificador de 3 entradas y 8 salidas .....	86
Figura 4.7 Decodificador 3x8 para implementar la función F .....	86
Figura 4.8 Decodificador de 4x16 a partir de dos decodificadores de 3x8 .....	87
Figura 4.9 Funcionamiento del multiplexor .....	87
Figura 4.10 Multiplexor de 4 canales .....	88
Figura 4.11 Implementación de la función f usando un multiplexor ...	89
Figura 4.12 Multiplexor de 32 canales usando dos multiplexores de 16 canales .....	90
Figura 4.13 Demultiplexor de 4 canales de salida .....	90
Figura 4.14 Demultiplexor de 4 canales usando un decodificador de 2x4 .....	91
Figura 4.15 Circuito básico de comparación de 1 bit .....	91
Figura 4.16 Comparador de 8 bits usando 2 comparadores de 4 bits..	92
Figura 4.17 Generador/Detector de paridad de 8 bits .....	92
Figura 4.18 Semisumador o sumador parcial .....	93
Figura 4.19 Sumador total .....	94
Figura 4.20 Cuádruple sumador total.....	95
Figura 4.21 Sumador/Restador 4 bits complemento a dos .....	96
Figura 4.22 Sumador/Restador 4 bits complemento a uno .....	97
Figura 4.23 ALU de 4 bits .....	97

## Capítulo 5 Sistemas Secuenciales

Figura 5.1 Sistema Secuencial Asíncrono .....	103
Figura 5.2 Sistema Secuencial Síncrono .....	103
Figura 5.3 Función en $\Pi$ .....	105
Figura 5.4 Función en $\Sigma$ .....	106
Figura 5.5 (a) Circuito – (c) Biestable SR (NAND) .....	106
Figura 5.6 (a) Circuito – (c) Biestable SR (NOR) .....	107
Figura 5.7 Componentes de una señal lógica .....	108
Figura 5.8 Biestable SR síncrono por nivel .....	109
Figura 5.9 Biestable SR maestro esclavo.....	110
Figura 5.10 Biestable SR activado por flanco .....	111
Figura 5.11 Representación de biestables activados por flanco .....	111
Figura 5.12 Biestable JK Maestro – Esclavo a partir de dos SR por nivel.....	112

Figura 5.13 Biestable JK por flanco ascendente a partir de un SR por flanco .....	113
Figura 5.14 Biestable D implementado con biestables SR y JK.....	114
Figura 5.15 Registro de desplazamiento cuatro bits serie-serie.....	115
Figura 5.16 Registro de desplazamiento paralelo-serie de cuatro bits ..	115
Figura 5.17 Registro paralelo – paralelo de 8 bits .....	116
Figura 5.18 Interconexión de registros por bus común.....	118
Figura 5.19 Interconexión registros usando registros tri-estado .....	118
Figura 5.20 Contador asíncrono de 4 bits. (a) Diagrama circuital (b) Diagrama de tiempo .....	120
Figura 5.21 Contador binario natural de 4 bits síncrono .....	120
Figura 5.22 Multiplicador paralelo de 4 bits con generación de overflow .....	123
Figura 5.23 Multiplicador serie de 4 bits con generación de overflow ..	124
Figura 5.24 Circuito divisor serie de 4 bits.....	128

## Capítulo 6 Memorias Electrónicas

Figura 6.1 Tiempo de acceso (palabras/seg) en función del costo (\$/bit).....	136
Figura 6.2 Esquema general de una memoria RAM .....	138
Figura 6.3 Diagrama en bloques de una memoria RAM .....	139
Figura 6.4 Esquema de una memoria RAM 2D.....	140
Figura 6.5 Esquema de una memoria RAM 3D.....	140
Figura 6.6 Celda básica para una organización 2D .....	142
Figura 6.7 Celda básica para una organización 2D .....	142
Figura 6.8 RAM de lectura/escritura con una organización 2D .....	143
Figura 6.9 Diagrama en bloques de memoria RAM estática .....	145
Figura 6.10 Diagrama de tiempo para la operación de lectura y escritura .....	146
Figura 6.11 Estructura interna de memorias RAM de escritura lectura dinámicas (DRAM) .....	147
Figura 6.12 Esquema de las conexiones en una memoria ROM .....	148
Figura 6.13 Esquema de las conexiones en una memoria ROM .....	150
Figura 6.14 Memoria de N palabras de k.m bits .....	151
Figura 6.15 Memoria de 2kN palabras de k.m bits .....	152
Figura 6.16 Memoria ejemplo .....	153
Figura 6.17 Esquema de una memoria serie .....	154
Figura 6.18 Esquema de una memoria serie bit a bit .....	154
Figura 6.19 Detalles de una memoria serie bit a bit .....	154
Figura 6.20 Esquema de memoria serie posición a posición .....	155
Figura 6.21 Detalles de memoria serie posición a posición .....	155
Figura 6.22 Esquema de registro de desplazamiento estático .....	156
Figura 6.23 Esquema de registro de desplazamiento dinámico .....	157
Figura 6.24 Circuito para la operación de lectura .....	157
Figura 6.25 Circuito para la operación de escritura .....	158

Figura 6.26 Esquema de una memoria FIFO.....	159
Figura 6.27 Funcionamiento de una memoria FIFO .....	159
Figura 6.28 Diagrama en bloques de una memoria FIFO implementada con un registro de desplazamiento estático.....	160
Figura 6.29 Ejemplo de aplicación de una memoria FIFO en sistemas digitales.....	160
Figura 6.30 Esquema de una memoria LIFO.....	161
Figura 6.31 Funcionamiento de una memoria LIFO .....	161
Figura 6.32 Diagrama en bloques de una memoria LIFO implementada con un registro de desplazamiento reversible.....	162
Figura 6.33 Diagrama en bloque de una LIFO.....	162

## **Capítulo 7 Arquitectura Básica de una Computadora**

Figura 7.1 Estructura básica de una computadora .....	166
Figura 7.2 Formato de los datos numéricos en punto fijo .....	169
Figura 7.3 Formato de las instrucciones .....	169
Figura 7.4 Consola de la máquina elemental .....	170
Figura 7.5 Diagrama en bloques de la máquina elemental.....	171
Figura 7.6 Formato de las instrucciones .....	173
Figura 7.7 La transmisión de direcciones en la Blue .....	179
Figura 7.8 La transmisión de instrucciones y operandos en la Blue....	179
Figura 7.9 Diagrama en bloques de la Unidad de Control Cableada....	181
Figura 7.10 Diagrama en bloques del secuenciador .....	184
Figura 7.11 Diagrama de tiempos del secuenciador .....	184
Figura 7.12 Lógica para el funcionamiento de las E/S .....	189
Figura 7.13 Vista parcial de la lógica control .....	190
Figura 7.14 Máquina elemental con unidad de control Microprogramada.....	191
Figura 7.15 Unidad de control microprogramada .....	194
Figura 7.16 Circuito que resuelve los requerimientos y las señales de control.....	199
Figura 7.17 Circuito asociado al Registro MBR .....	200
Figura 7.18 Diagrama en bloques unidad aritmética y lógica.....	201

## **Capítulo 8 Arquitectura Convencional**

Figura 8.1 Formato general de las instrucciones con cuatro direcciones.....	206
Figura 8.2 Formato general de las instrucciones con tres direcciones.....	207
Figura 8.3 Formato general de las instrucciones con dos direcciones.....	207
Figura 8.4 Formato general de instrucciones con una dirección .....	208
Figura 8.5 Formato de instrucción modificado .....	212

Figura 8.6 Diagrama en bloques de la Máquina Elemental Indexada.....	216
Figura 8.7 Unidad de Control incluyendo sistema elemental de Interrupciones .....	231
Figura 8.8 Secuencia de acciones de la rutina de interrupción .....	233
Figura 8.9 Sistema de interrupciones usando Banderas de Dispositivos y un Registro de Máscaras para las prioridades .....	236
Figura 8.10 Vector de Interrupciones.....	238
Figura 8.11 Sistema de interrupciones usando bit I y M en los Dispositivos y la señal ACK de la CPU para las prioridades .....	239
Figura 8.12 Transferencia de salida sobre el periférico 32 .....	241
Figura 8.13 Transferencia de entrada sobre el periférico 16 .....	242
Figura 8.14 Configuración de pines del 8088 .....	244
Figura 8.15 Diagrama en bloques del microprocesador 8088.....	246
Figura 8.16 Formato de una microinstrucción del 8088 .....	248
Figura 8.17 Suma del segmento y offset .....	251

## **Capítulo 9 Arquitectura Avanzada**

Figura 9.1 Ejemplo de pipeline .....	262
Figura 9.2 Diagrama en bloque de la Unidad de Control .....	266
Figura 9.3 Secuenciador de la Unidad de Control .....	267
Figura 9.4 Incrementador del PC .....	272
Figura 9.5 Circuito de predicción de salto.....	272
Figura 9.6 Simultaneidad del ciclo de lectura a la memoria y a la caché .....	273
Figura 9.7 Diagrama en bloque de un canal DMA .....	275
Figura 9.8 Formato típico de una instrucción .....	281
Figura 9.9 Organización de registros usados para el traslape .....	283
Figura 9.10 Bloques y sub-bloques del Intel 80486.....	292
Figura 9.11 Bloques del Intel Pentium .....	293

## **Capítulo 10 Entradas y Salidas (E/S)**

Figura 10.1 Módulo de E/S.....	299
Figura 10.2 Diagrama en bloques de un dispositivo externo o Periférico .....	300
Figura 10.3 Diagrama de bloques de un módulo de E/S .....	302
Figura 10.4 Diagrama en bloques de la interface programable de periféricos 802C55A de Intel .....	304
Figura 10.5 Interface teclado/pantalla usando el 802C55A .....	305
Figura 10.6 Arquitectura de un canal de E/S .....	307
Figura 10.7 Señales continuas y discretas .....	308
Figura 10.8 Señalización Digital .....	311

Figura 10.9 E/S paralela y serie .....	312
Figura 10.10 Transmisión Paralela Stroboscópica .....	313
Figura 10.11 Diagrama de tiempo para transmisión paralela Stroboscópica .....	313
Figura 10.12 Lógica de funcionamiento de la transmisión paralela con handshaking .....	314
Figura 10.13 Diagrama de tiempo que ejemplifica el handshaking.....	315
Figura 10.14 Transmisión serie asíncrona .....	315
Figura 10.15 Diagrama de tiempo para transmisión serie Asíncrona ..	316
Figura 10.16 RS323 C como ejemplo de transmisión asíncrona.....	317
Figura 10.17 Esquema completo de una comunicación RS232 C.....	317
Figura 10.18 Desfasaje en la temporización entre transmisor y receptor .....	318
Figura 10.19 Transmisión serie síncrona con reloj independiente.....	319
Figura 10.20 Transmisión serie síncrona con reloj incorporado en los datos .....	319
Figura 10.21 Trama en transmisión serie síncrona .....	320
Figura 10.22 Topología punto a punto .....	320
Figura 10.23 Topología Multipunto.....	321
Figura 10.24 Cable con varios pares trenzados .....	322
Figura 10.25 Patch cord de cable UTP .....	323
Figura 10.26 Patch cord de fibra .....	324
Figura 10.27 Comunicaciones inalámbricas .....	325
Figura 10.28 Memoria USB .....	326
Figura 10.29 Cables USB .....	327
Figura 10.30 Puerto Ethernet para cable UTP en placa de red .....	329
Figura 10.31 Cable de par trenzado para puerto Ethernet.....	329
Figura 10.32 Placa de red Ethernet con un puerto de par trenzado UTP .....	330
Figura 10.33 Placa de red con puerto Ethernet de fibra óptica .....	331
Figura 10.34 Conector de la interface HDMI .....	334
Figura 10.35 Router Wi-Fi.....	339

## Índice de Tablas:

### Capítulo 1 Representación Numérica

Tabla 1.1 Correspondencia entre sistemas de numeración .....	17
Tabla 1.2 Secuencia de conversión de decimal a binario .....	18
Tabla 1.3 Representación de números enteros usando una representación de 3 bits .....	22

### Capítulo 2 Códigos Numéricos y Alfanuméricos

Tabla 2.1 Código binario natural .....	31
Tabla 2.2 Representación del código de Gray en 2, 3 y 4 bits .....	33
Tabla 2.3 Representación del código de Johnson de 5 bits .....	33
Tabla 2.4 Tipos de códigos BCD .....	34
Tabla 2.5 Representación del número 926 en distintos códigos BCD .....	35
Tabla 2.6 Código ASCII estándar de 7 bits .....	36
Tabla 2.7 Ejemplos de códigos de peso constante.....	41
Tabla 2.8 Tabla correctora del código de Hamming .....	42

### Capítulo 3 Álgebra de Boole

Tabla 3.1 Tabla de verdad de la función $f(a,b,c)$ .....	67
Tabla 3.2 Pesos de las variables booleanas .....	70

### Capítulo 4 Sistemas Combinacionales

Tabla 4.1 Tabla de verdad de la función $F$ .....	88
Tabla 4.2 Tabla auxiliar .....	89
Tabla 4.3 Tabla de verdad del comparador.....	92
Tabla 4.4 Tabla de verdad del semisumador .....	93
Tabla 4.5 Tabla de verdad del sumador total.....	95

### Capítulo 5 Sistemas Secuenciales

Tabla 5.1 Tabla de verdad del sistema secuencial de ejemplo .....	105
Tabla 5.2 Tabla de verdad de un biestable SR .....	111
Tabla 5.3 Biestables JK, T y D.....	112
Tabla 5.4 Comparativa aproximada de los tiempos de productos .....	126
Tabla 5.5 Secuencia de control del circuito divisor de la Figura 5.24 ..	130

### Capítulo 6 Memorias Electrónicas

Tabla 6.1 Clasificación de las memorias usando sus principales características .....	136
Tabla 6.2 Combinaciones de las líneas de control para	

escritura o lectura.....	144
--------------------------	-----

## **Capítulo 7 Arquitectura Básica de una Computadora**

Tabla 7.1 Conjunto de Instrucciones .....	174
Tabla 7.2 Ciclo de búsqueda .....	185
Tabla 7.3 Ejecución de las instrucciones de uno y dos ciclos.....	187
Tabla 7.4 Ciclo de búsqueda de las instrucciones INP y OUT.....	188
Tabla 7.5 Formato de la microinstrucción de la máquina elemental microprogramada .....	192
Tabla 7.5 Contenido de la ROM (256 x 45).....	197

## **Capítulo 8 Arquitectura Convencional**

Tabla 8.1 Conjunto de instrucciones de la Máquina Elemental Indexada .....	220
Tabla 8.2 Formato, y ciclos de búsqueda y ejecución de la instrucción LDA 3,XXXXXX.....	221
Tabla 8.3 Formato, y ciclos de búsqueda y ejecución de la instrucción LDA 1,XXXXXX.....	224
Tabla 8.4 Formato, y ciclos de búsqueda y ejecución de la instrucción LDA 0,XXXXXX.....	225
Tabla 8.5 Formato, y ciclos de búsqueda y ejecución de la instrucción INC2 valor XXXXXX .....	226
Tabla 8.6 Formato, y ciclos de búsqueda y ejecución de la instrucción ENI3, XXXXXX.....	227
Tabla 8.7 Ciclo de interrupción .....	232
Tabla 8.8 Ciclo de máquina de la instrucción RTI .....	234
Tabla 8.9 Ciclo de la instrucción OUT YY.....	240
Tabla 8.10 Ciclo de la instrucción INP YY .....	241
Tabla 8.11 Símbolos, tipo y función de algunos pines del 8088 .....	245
Tabla 8.12 Selección del registro de segmento.....	252

## **Capítulo 9 Arquitectura Avanzada**

Tabla 9.1 Comportamiento de las 5 Unidades Funcionales en 13 ciclos de memoria .....	262
Tabla 9.2 Ciclos de pipeline.....	271
Tabla 9.3 Procesador RISC con 3 etapas .....	281
Tabla 9.4 Comparación de las características RISC y CISC .....	286
Tabla 9.5 Mejoras de los procesadores Intel desde el 8086 al 80586 ...	289

## **Capítulo 10 Entradas y Salidas (E/S)**

Tabla 10.1 Características de las bandas de comunicaciones .....	325
Tabla 10.2 Clases de Bluetooth .....	332
Tabla 10.3 Versiones y velocidad en Bluetooth.....	333
Tabla 10.4 Asignación de pines de un conector HDMI tipo A .....	335

Tabla 10.5 Resumen de las sucesivas revisiones de HDMI.....337

## **Índice de Cuadros:**

### **Capítulo 2 Códigos Numéricos y Alfanuméricos**

Cuadro 2.1 Códigos continuos y cíclicos .....32  
Cuadro 2.2 Ejemplo de códigos con bit de paridad usando BCD Nat...40  
Cuadro 2.3 Ejemplo de código Hamming .....44

