CAPITULO 1

INTRODUCCION A LOS ORDENADORES

1.1 ESTRUCTURA BASICA DE UN ORDENADOR

1.1.1 EL SISTEMA DE MEMORIA

1.1.2 LA UNIDAD CENTRAL DE PROCESO

1.1.3 LA UNIDAD DE INTERCAMBIOS

1.2 ESTRUCTURA REAL DE UN ORDENADOR DIGITAL

1.1 ESTRUCTURA BASICA DE UN ORDENADOR

Expresado de la forma más simple, un ordenador es un dispositivo que ejecuta automáticamente una secuencia de operaciones sobre unos datos dados. El propósito de tales operaciones puede ser la resolución de problemas matemáticos, el control de ciertas funciones o dispositivos o, bastante a menudo, una combinación de ambas tareas. Si los datos se expresan en forma continua se habla de un *ordenador analógico*; si los datos se expresan de forma discreta, se habla de un *ordenador digital*.

En los ordenadores digitales, las cantidades numéricas son representadas por niveles de tensión o pulsos eléctricos discretos. La presencia o ausencia de un pulso, o de la tensión apropiada, define un dígito binario o *bit* (BI*nary digi*T). Un dígito binario es un número que sólo puede tomar uno de dos valores: 0 ó 1. Lo que hace útil al dígito binario es que puede ser representado físicamente por cualquier dispositivo *biestable*, es decir, aquél que posee dos estados estables (en "conducción" o en "alta" y en "corte" o en "baja") y la posibilidad de conmutar, cuando se le ordena, de un estado estable a otro. En definitiva, un dispositivo biestable puede representar a un 0 en uno de los estados y un 1 en el otro, tal como muestra la **Figura 1.1**.

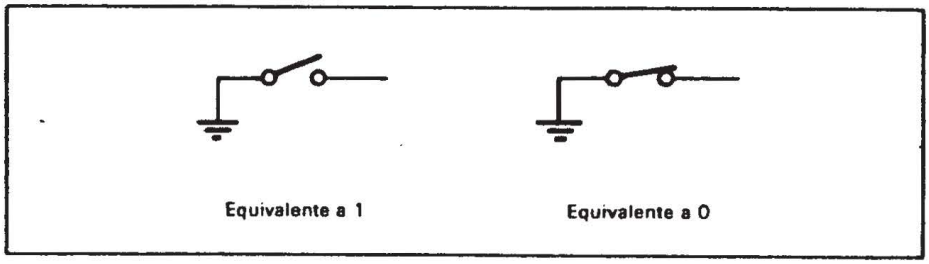


Figura 1.1
Representación de los dígitos binarios
por un dispositivo biestable

Por su parte, un grupo de bits que se manejan conjuntamente recibe el nombre de *palabra* (*word*). Así, una palabra está formada por un conjunto de bits, que se expresan como 0 ó 1, y que quedan físicamente representados por pulsos eléctricos apropiadamente definidos.

Una palabra puede representar una cantidad numérica, llamada *operando*, o una *instrucción*, que obliga a la máquina a manipular ciertos operandos de una forma específica. El juego de instrucciones que gobierna la máquina se llama *programa*. El procedimiento por el cual se desarrollan los *cálculos aritméticos* o las *operaciones lógicas* se llama *algoritmo*.

Para calcular o controlar automáticamente, el ordenador debe realizar varias funciones internas fundamentales, las más sencillas de las cuáles son las operaciones aritméticas de adición y sustracción sobre dos operandos. Incluso estos procesos tan simples requieren varias funciones que son realizadas por diferentes secciones de la máquina, tal como muestra la **Figura 1.2**:

* Sistema de memoria
* Unidad central de proceso
* Unidad de intercambios

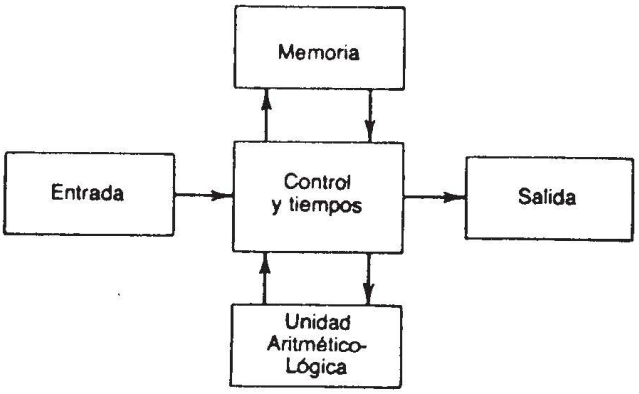


Figura 1.2
Bloques funcionales principales de un ordenador digital

1.1.1 EL SISTEMA DE MEMORIA

El sistema de memoria del ordenador es un conjunto de elementos biestables que permiten almacenar datos binarios en forma de operandos o de instrucciones.

La porción de memoria destinada al almacenamiento de operandos se llama *memoria de datos*. La unidad de proceso puede llamar o insertar una palabra en cualquier parte de la memoria de datos durante la ejecución de un programa, y

está diseñada de tal modo que puede acceder a cualquier posición aleatoriamente.

La parte de la memoria que almacena las instrucciones se llama *memoria de programa*. Cada instrucción de esta memoria es suministrada al ordenador en una secuencia preestablecida o programa. Una vez cargado éste, el ordenador decodifica cada instrucción e inicia el proceso específico, obligado por ésta. Al contrario de la memoria de datos, en la memoria de programa no se producen, normalmente, operaciones de escritura durante la ejecución.

1.1.2 LA UNIDAD CENTRAL DE PROCESO

Se suele aplicar el término de *unidad central de proceso* (*Central Processing Unit*) al conjunto de dos bloques funcionales: la *unidad aritmético -lógica* (*Arithmetic-Logic Unit*) y el *bloque de control y tiempos*.

La ALU realiza las operaciones aritméticas y los procesos lógicos sobre los operandos, suministrando resultados completos o parciales, dependiendo de los problemas a resolver. Los resultados parciales tienen aplicaciones internas en el ordenador y raramente se presentan al mundo exterior, a menos que sean requeridos especialmente.

La unidad de control coordina a todas las otras unidades funcionales del ordenador en una secuencia lógica y en el tiempo adecuado. Esto último se consigue gracias a un reloj central muy preciso que suministra unos pulsos básicos (*secuencia básica de tiempos*) con los que están sincronizados también el resto de los circuitos. La unidad de control recibe las instrucciones del programa almacenado en la memoria, las decodifica y las dirige al resto de los subsistemas funcionales para ejecutarlas.

En definitiva, la CPU es el corazón del ordenador, ya que controla todo el resto de las funciones realizadas por el sistema: llama a las instrucciones de la memoria, las decodifica y las ejecuta, utilizando además la memoria y la unidad de intercambio de la forma que la ejecución de las instrucciones lo requiera. La CPU también reconoce y responde a unas señales exteriores de demanda de servicios, denominadas *interrupciones*, emitidas por dispositivos externos o por el operador de la máquina.

1.1.3 LA UNIDAD DE INTERCAMBIOS

Esta unidad le permite al ordenador comunicarse con varios tipos de dispositivos llamados *periféricos* o *dispositivos E/S* (*Entrada/Salida* ó *Input/Output*) y está formada por los *puertos E/S* (*I/O ports*). Los puertos de entrada permiten a la CPU la adquisición de datos de otros dispositivos externos. Los puertos de salida capacitan a la CPU para comunicar los resultados de sus cálculos o procesos lógicos a los periféricos exteriores. También pueden utilizarse para suministrar señales de control de proceso para dirigir a otros sistemas, como sistemas de instrumentación o regulación.

La mayoría de los ordenadores disponen de más de un puerto de E/S; en este caso, los puertos deben ser direccionables, es decir, deben poder ser activados selectivamente bajo el control de un programa.

1.2 ESTRUCTURA REAL DE UN ORDENADOR DIGITAL

La Figura 1.2 es sólo un diagrama de bloques generalizado de un ordenador digital, que sólo pretende dar una idea de los bloques funcionales más importantes; por lo tanto, es irreal. Cuando un ordenador resuelve un problema que le ha sido planteado, la CPU debe ser controlada de acuerdo con un programa preestablecido. La secuencia de las distintas operaciones la determina el programa almacenado en la memoria de programa, juntamente con la unidad de tiempos y control de la CPU. También es necesaria una memoria temporal o borrador para almacenar los resultados intermedios de los cálculos y procesos lógicos.

La unidad de control de un ordenador detecta las distintas instrucciones que le presenta el programa y suministra las señales de control apropiadas para las diversas partes del ordenador. La CPU también responde a señales externas, entre ellas, las del operador humano. La comunicación del operador con el ordenador se hace generalmente a través de una consola de operador, que normalmente consta de un *teclado* (*keyboard*) y una *pantalla CRT* (*Cathode Ray Tube*).

Si se incluyen todas estas características, se obtiene una organización del ordenador más real, tal como muestra la Figura 1.3.

Obsérvese que los distintos bloques mostrados en la Figura 1.3 están conectados por líneas de distintas características. Cada una de estas líneas

representa físicamente un grupo de conductores en paralelo que permiten la transferencia de palabras entre varios orígenes y destinos, y se denomina *bus*. Estos buses pueden ser de tres tipos:

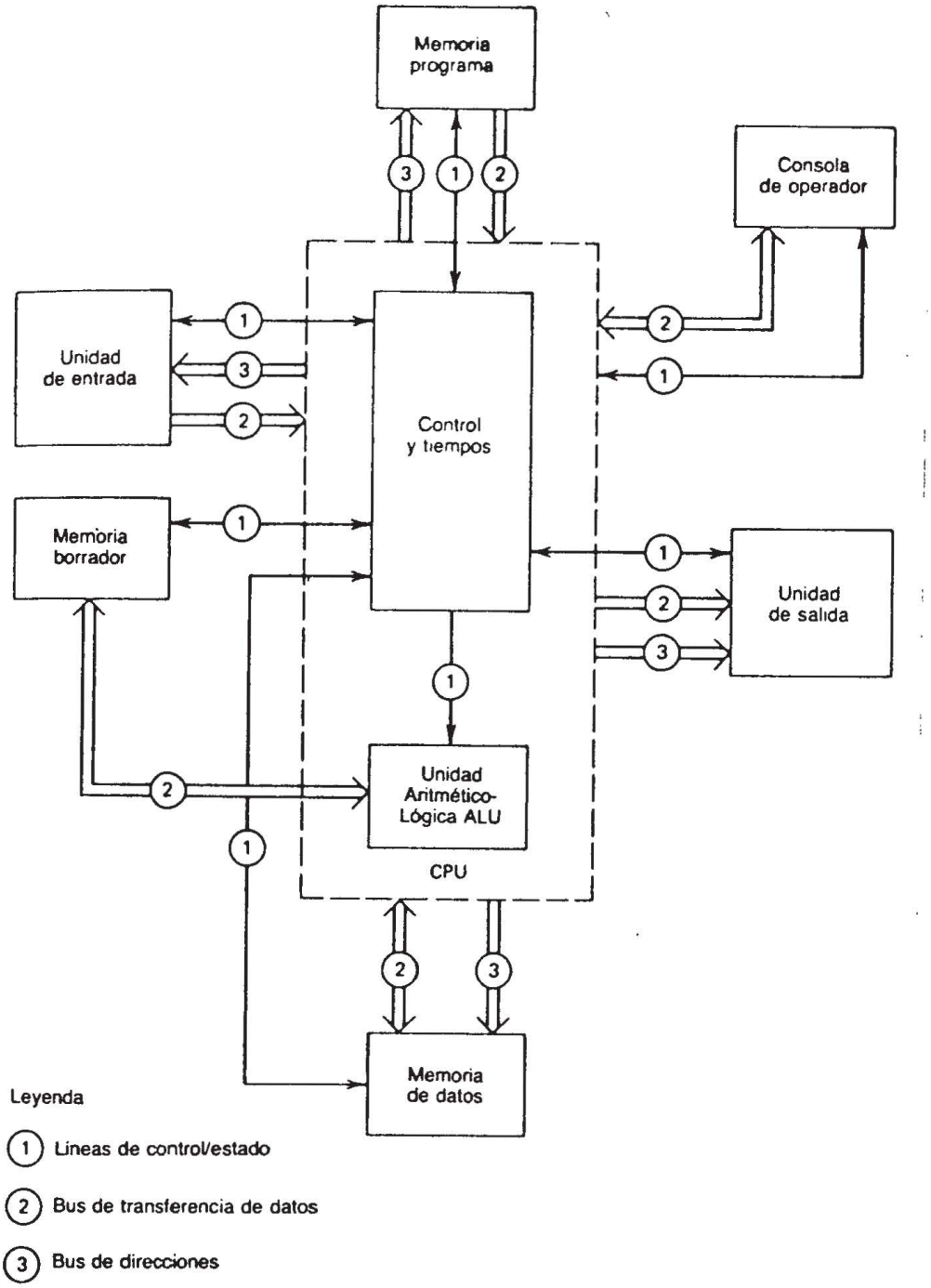


Figura 1.3

Organización real de un ordenador

* *Bus de direcciones:* Tiene la misión de comunicar a la memoria la dirección de la palabra que ha de ser leída o escrita y de comunicar a la unidad de intercambio la dirección del dispositivo periférico que debe ser atendido por la CPU; como la CPU sólo emite esos bits y a su vez no contiene elementos que puedan ser direccionados, este bus debe ser exclusivamente unidireccional.

* *Bus de datos:* Es el medio de intercambio de los datos entre la CPU, la memoria y las unidades de intercambio, indistintamente; cada uno de estos elementos puede ser fuente o destino, lo que obliga a que este bus sea bidireccional.

* *Bus de control:* Está formado por señales independientes que genera la unidad de control o que actúan sobre la misma; aunque son todos unidireccionales, por su naturaleza independiente unos son de entrada y otros de salida de la CPU.

También se puede observar en la **Figura 1.3** que, para simplificar el diagrama, sólo se muestran dos bloques en la CPU, que está indicada a trazos. Por supuesto, la CPU contiene más elementos aparte de los indicados. Por esta razón, todos los buses con excepción de los que unen la ALU con la memoria borrador, se representan entrando o saliendo de la línea de trazos de la CPU.

Ahora bien, los grandes avances de la tecnología de semiconductores han permitido reorganizar el esquema estudiado hasta aquí. Por ejemplo, por motivos de organización y direccionamiento, se combinaron las memorias de datos y de programa en una única memoria que las contiene a ambas y que puede estar formada por *circuitos integrados* (*chips*) de *memoria de acceso aleatorio* (*Random Access Memory*) y de *memoria de sólo lectura* (*Read-Only Memory*), pudiéndose asignar a diferentes partes de la memoria general el almacenamiento de datos y programas. De la misma forma, las funciones de entrada y salida también han podido ser combinadas en un solo circuito integrado y en un único dispositivo E/S.

La configuración resultante se muestra en la **Figura 1.4**, una aproximación muy básica de los ordenadores modernos. Por simplicidad, en la Figura 1.4 sólo se han representado los buses de datos y las líneas de control.

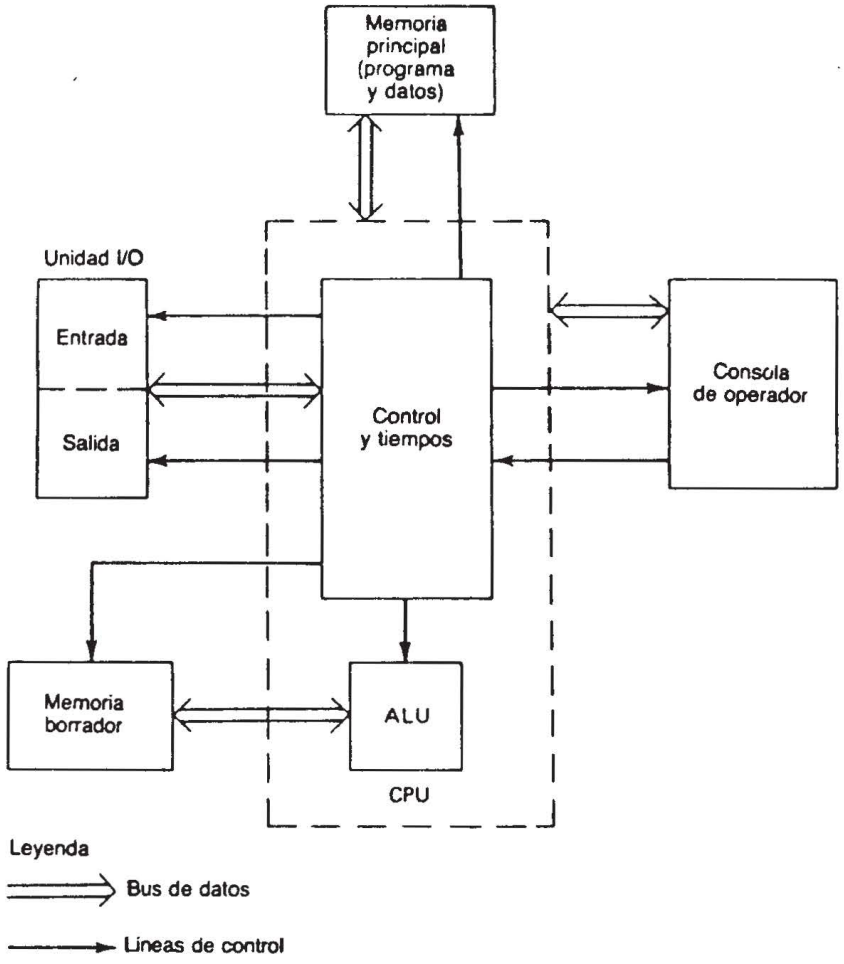


Figura 1.4

Organización básica de un ordenador moderno

Finalmente, posteriores avances enla tecnología de semiconductores han permitido incluir en un mismo circuito integrado de *gran escala de integración* (*Large Scale Integration*), la unidad de control y de tiempos, la ALU y la memoria borrador. El circuito integrado resultante es el llamado *microprocesador*, que se considera la unidad fundamental dentro de la configuración del moderno ordenador, cuyo esquema general se muestra en la Figura 1.5.

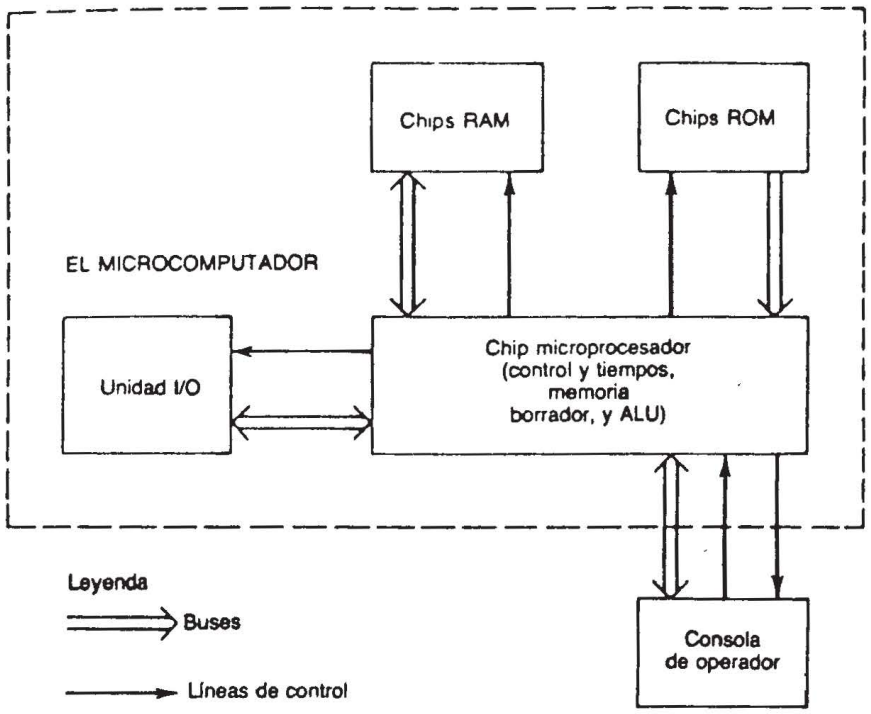


Figura 1.5

Esquema de un ordenador actual

BIBLIOGRAFIA

Khambata, A.J.; *"Microprocesadores/microcomputadores. Arquitectura, software y sistemas"*, Gustavo Gili, Barcelona (1987).

Osborne, A.; *"Introducción a las microcomputadoras"*, Vol. 1, 2ª ed., Osborne/McGraw-Hill, Madrid (1986).

CAPITULO 2

CODIFICACION DE DATOS EN LOS ORDENADORES

2.1 INTRODUCCION

2.2 SISTEMAS DE NUMERACION

- 2.2.1 NUMEROS BINARIOS PUROS
- 2.2.2 CONVERSION ENTRE NUMEROS BINARIOS Y DECIMALES
- 2.2.3 NUMEROS BINARIOS CON SIGNO: COMPLEMENTO A DOS
  - 2.2.3.1 OPERACIONES CON PALABRAS BINARIAS: ACARREO Y REBOSE
- 2.2.4 NUMEROS BINARIOS EN MULTIPLE PRECISION
- 2.2.5 NUMEROS BINARIOS FRACCIONARIOS: PUNTO FIJO
- 2.2.6 NUMEROS BINARIOS EN NOTACION CIENTIFICA: PUNTO FLOTANTE
- 2.2.7 VARIANTES DE LOS NUMEROS BINARIOS
  - 2.2.7.1 SISTEMA OCTAL
  - 2.2.7.2 SISTEMA BCD
  - 2.2.7.3 SISTEMA HEXADECIMAL
- 2.2.8 CODIGOS DE CARACTERES
  - 2.2.8.1 CODIGO ASCII
  - 2.2.8.2 EL OCTAVO BIT: BIT DE PARIDAD
  - 2.2.8.3 EL OCTAVO BIT: CODIGO ASCII EXTENDIDO

2.3 ORGANIZACION DE LOS DATOS EN LOS ORDENADORES

2.4 LA LOGICA DEL ORDENADOR

- 2.4.1 COMBINACIÓN DE OPERACIONES LOGICAS
- 2.4.2 OPERACIONES DE DESPLAZAMIENTO

2.1 INTRODUCCION

Como se ha visto anteriormente, el punto de partida de los datos en un ordenador es el *bit*, el elemento de información más pequeño posible. Este concepto es el que hace posible manejar la información de los ordenadores, debido a que ha sido posible construir sistemas electrónicos que funcionan con señales *on/off* (*encendido/apagado*) a gran velocidad, pero todo depende de la capacidad que se tenga para construir información real a partir de los simples bits 0 y 1 que representan, respectivamente, *off* y *on*, *falso* y *verdadero*, *no* y *si*.

Pero por sí solos los bits no son de mucho interés; sólo cuando se unen bits en conjuntos o *palabras* se puede conseguir representar una cantidad numérica, una carácter alfabético o una instrucción.

2.2 SISTEMAS DE NUMERACION

Un sistema de numeración es el conjunto de reglas que se emplean para representar todos los números utilizando un conjunto reducido de dígitos o guarismos. En todos los sistemas de numeración se distinguen dos características básicas:

* La *base* del sistema es la cantidad de dígitos distintos que se utilizan para representar un número.

* La *posición de un dígito* representa el valor que tiene dicho dígito en el número.

Ambas características quedan contempladas en el siguiente teorema: "*Todo número natural puede expresarse como un polinomio respecto a la base* B *(número natural mayor que 1) con coeficientes que son números naturales menores que* B". Es decir, si los dígitos de cualquier sistema de numeración se representan por d_i, d_j, d_k, d_l, etc., cualquier número puede expresarse mediante la ecuación:

$$d_i d_j d_k d_l = d_l \cdot B^0 + d_k \cdot B^1 + d_j \cdot B^2 + d_i \cdot B^3 \qquad [2.1]$$

2.2.1 NUMEROS BINARIOS PUROS

Un grupo de bits sin ninguna definición preestablecida, puede representar un número en base 2 o *número binario puro* (*straight binary*).

Dependiendo del número de bits y de la combinación de los mismos en el grupo, se pueden representar diferentes cantidades de valores. Así, por ejemplo, una palabra de 4 bits (*cuarteto* o *"nibble"*) puede representar 2^4 = 16 valores; una palabra de 8 bits (*octeto* o *"byte"*) puede representar hasta 2^8 = 256 valores, y una palabra de 16 bits (= 2 bytes) puede representar hasta 2^{16} = 65.536 valores.

Generalmente se utiliza el byte como agrupamiento básico de una serie de bits que van a ser procesados como una unidad. Dentro de un byte (o, en general, de una palabra), el bit situado más a la izquierda se denomina *bit más significativo* (*Most Significant Bit*) o *bit de mayor peso*, ya que tiene el mayor valor, mientras que el bit situado más a la derecha se denomina *bit menos significativo* (*Least Significant Bit*) o *bit de menor peso*.

2.2.2 CONVERSION ENTRE NUMEROS BINARIOS Y DECIMALES

La conversión de números de una base a otra se basa en la ecuación [2.1] y queda convenientemente explicada en la **Figura 2.1** (paso de binario a decimal) y en la **Figura 2.2** (paso de decimal a binario).

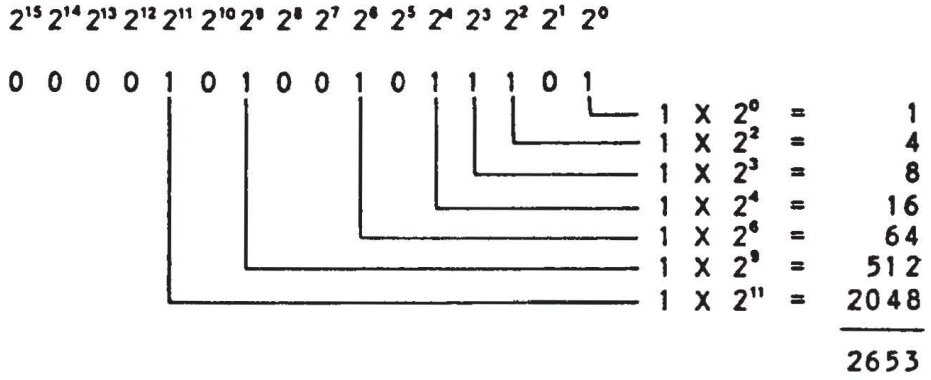


Figura 2.1
Paso de binario a decimal

2.2.3 NUMEROS BINARIOS CON SIGNO: COMPLEMENTO A DOS

Los números binarios puros no permiten la representación, de cantidades negativas, por lo que ha de establecerse algún método de identificación de cantidades positivas y negativas. La convención generalmente aceptada es usar un 0 en la posición del MSB para indicar una cantidad positiva y un 1 en esa

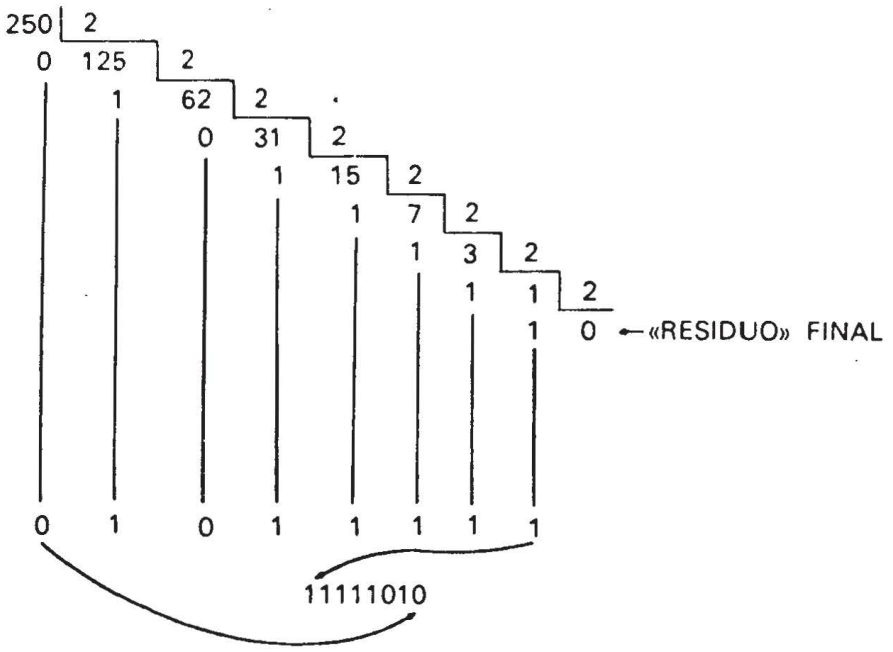


Figura 2.2

Paso de decimal a binario

misma posición para indicar una cantidad negativa. El resto de los bits representan el número. Si éste es positivo (MSB = 0), se representa como binario puro; si es negativo (MSB = 1) se representa en el denominado "complemento a dos". Así, un número de 8 bits utiliza un bit para el signo y 7 bits para el valor, tal como muestra la **Figura 2.3.**

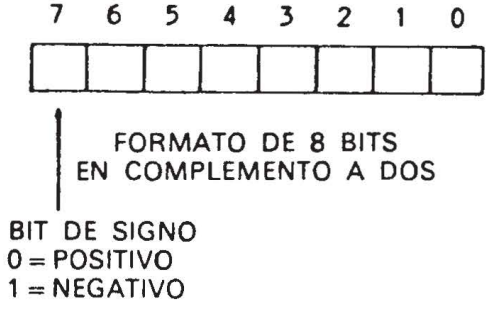


Figura 2.3

Representación binaria de un número con signo

Para obtener el *complemento a dos* (*two's complement*) de un número, se obtiene su complemento a uno (se cambian los 0 por 1 y los 1 por 0) y se le suma 1. Nemotécnicamente esta operación es idéntica a la de dejar el número de derecha a izquierda igual al original hasta que aparezca un 1; éste se deja tal como está y desde la siguiente posición hasta el extremo izquierdo (respetando el bit de signo) se cambian los 0 por 1 y los 1 por 0.

El siguiente ejemplo muestra la codificación de los números binarios con signo, tal como se ha indicado. El punto se escribe sólo a efectos de separar el bit de signo, y se opera con números de 1 byte:

Número a representar (decimal):	112
Es positivo (binario puro en 7 bits):	0.111 0000

Número a representar (decimal):	-112
Bit de signo:	1
Binario puro en 7 bits:	111 0000
Complemento a uno:	000 1111
Sumar 1:	000 0001
Resultado (complemento a dos):	001 0000
(confirmar la regla nemotécnica)	
Representación completa:	1.001 0000
(binario codificado)	

Obsérvese que si no se considerase la representación en complemento a dos, el binario puro que resulta (001 0000) tendría un valor decimal de 16.

De forma análoga, para obtener el valor decimal de un número binario cuyo MSB sea 1 (será un número negativo), se obtiene primero su complemento a dos (exceptuando el bit de signo, y luego se transforma en decimal, tal como muestra el siguiente ejemplo:

Número binario codificado:	1.110 1100
Bit de signo:	1
Bits de valor:	110 1100
Complemento a dos:	001 0100

Decimal equivalente:	20
Número decimal resultante:	-20

Obsérvese finalmente que con este tipo de notación se pueden representar con 8 bits valores desde -128 a +127. En efecto, el número positivo más grande que se podría representar sería:

Número binario:	0.111 1111
Bit de signo:	+
Bits de valor:	111 1111
Decimal equivalente:	127
Número decimal:	+127

Por su parte, el número más pequeño sería:

Número binario:	1.000 0000
Bit de signo:	-
Bits de valor:	000 0000
Complemento a dos:	1000 0000
Decimal equivalente:	128
Número decimal:	-128

De la misma forma, trabajando con 16 bits se pueden representar números desde -32.768 a +32.767, como se puede demostrar de idéntica manera a la utilizada con la palabra de 8 bits.

2.2.3.1 OPERACIONES CON PALABRAS BINARIAS: ACARREO Y REBOSE

Se puede efectuar cualquier tipo de operación aritmética con números binarios, de la misma forma que se hace con los números decimales. Sin embargo, cuando se opera con dígitos binarios agrupados en palabras, se pueden presentar dos problemas, que es necesario conocer para poder preverlos en el diseño del microprocesador. Estos dos problemas son el acarreo y el rebose.

El *acarreo* o *arrastre* (*carry*) es la cantidad resultante al realizar una

suma de dos dígitos cuyo resultado implica una posición de mayor peso en el número ("llevarse uno"), es decir:

1 + 1 = 10

o bien:

1 + 1 = 0 acarreo = 1

Cuando se opera con una longitud de palabra fija (por ejemplo, 8 bits), los acarreos producidos en los LSB no plantean dificultades; sin embargo, el que se produce a continuación del MSB tiene el problema de que no puede representarse en un nuevo bit, ya que el número de éstos es fijo. Como consecuencia es necesario prever un "bit de acarreo", que toma el valor 1 cuando se produce dicho acarreo, con el fin de tenerlo en cuenta al realizar otra operación o dar el resultado. Ello se logra diseñando en el microprocesador un *indicador de acarreo* (*carry flag*), como se verá más adelante.

El *rebose* o *desbordamiento* (*overflow*) se produce cuando el resultado de una operación aritmética excede del número de bits utilizado para definir una palabra. Así, empleando una notación binaria codificada y una palabra de 8 bits, no se puede representar un número mayor de +127 ni menor de -128, ya que se produciría un rebose en cualquiera de ambos casos. Como consecuencia se prevé un "bit de rebose", que toma el valor 1 cuando se produce dicho rebose, con el fin de indicar que el resultado obtenido, representado por la palabra de 8 bits, no es correcto. Ello se consigue diseñando en el microprocesador un *indicador de rebose* (*overflow flag*), como también se verá más adelante.

2.2.4 NUMEROS BINARIOS EN MULTIPLE PRECISION

Muchas veces es necesario trabajar con cantidades más grandes que el número de dígitos significativos que permite el formato de palabra, o bien se necesita hacer operaciones de alta velocidad u operaciones que necesiten la obtención de un elevado número de dígitos significativos (*precisión*). Es decir, no basta con la *simple precisión* que da un byte (7 dígitos significativos) o la *doble precisión* que dan dos bytes (15 dígitos significativos), por lo que se utiliza la *múltiple precisión*, que consiste en representar un número con varios bytes. En este caso, los bits se numeran

según la representación normal de potencias de 2, utilizando el MSB como bit de signo. De esta manera el número se considera un grupo único de n bits, aunque esté físicamente separado en m bytes.

Las operaciones aritméticas de suma de dos números en esta precisión implican tomar los bytes de cada uno de ellos de derecha (menos significativo) a izquierda (más significativo) y sumarlos todos. En caso de acarreo de la suma del primero, el acarreo debe sumarse al LSB del siguiente byte.

Dado que los acarreos se propagan al bit siguiente dentro de cada byte como resultado normal de la suma, cualquier acarreo que se produzca como consecuencia de la suma del MSB, afectará al byte siguiente y, por tanto, debe guardarse y añadirsele. Para ello se utiliza el indicador de acarreo y se realiza una llamada "suma con acarreo" al hacer esta operación, es decir, una suma que usa los dos sumandos y el indicador de acarreo.

2.2.5 NUMEROS BINARIOS FRACCIONARIOS: PUNTO FIJO

Hasta ahora sólo se han considerado valores enteros, pero también será necesario prever el manejo de números binarios fraccionarios por parte de un ordenador. Para ello es necesario definir una notación que permita representar cifras decimales, lo que se consigue de una forma similar al caso de los números decimales, estableciendo el formato denominado de *punto fijo* (*fixed point*).

El formato de punto fijo consiste en mantener separadas las partes fraccionaria y entera de un número mixto. La parte entera se representa en dos bytes (signo y complemento a dos) y la parte fraccionaria se representa en un byte, cuyas posiciones toman el valor de los inversos de las potencias de 2, haciendo el MSB igual a 2^{-1}. Entre ambas partes se sitúa el punto binario en posición fija y de modo ficticio, ya que el ordenador sólo operará con los 24 bits correspondientes al número. Ambas partes se procesan de forma separada.

En la Figura 2.4 se muestra el esquema de este tipo de representación, mientras que en la Figura 2.5 se puede observar un ejemplo del peso de las posiciones binarias del byte de fracción.

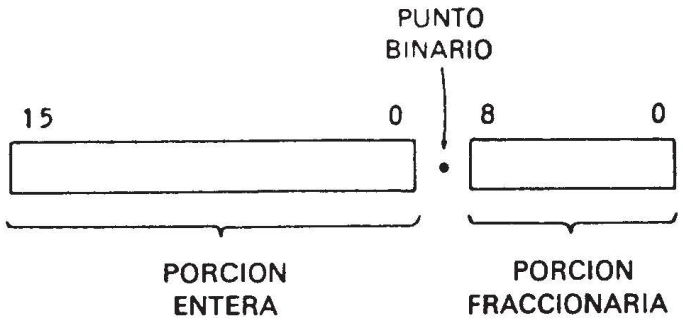


Figura 2.4
Representación binaria de números fraccionarios en formato de punto fijo

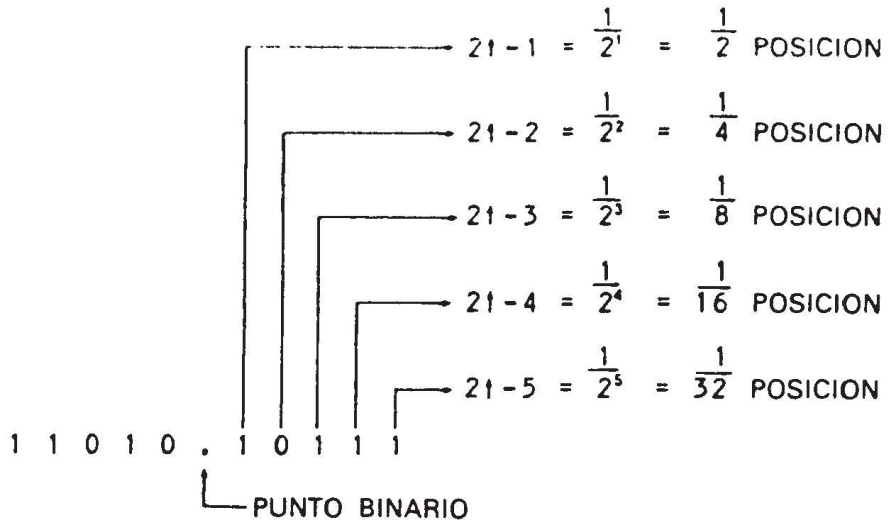


Figura 2.5
Valor de los bits fraccionarios en formato de punto fijo

El máximo número positivo que puede almacenarse en este esquema será:

Número binario:	0.111 1111 1111 1111 . 1111 1111
Signo:	+
Valor:	111 1111 1111 1111 . 1111 1111
Parte entera:	111 1111 1111 1111
Valor decimal:	32.767
Parte fraccionaria:	1111 1111

Valor decimal:	0,996 093 75 ...
Número completo:	+32.767,996 093 75 ...

Por su parte, el máximo número negativo que se puede almacenar de esta forma será:

Número binario:	1.000 0000 0000 0000 . 0000 0000
Signo:	-
Valor:	000 0000 0000 0000 . 0000 0000
Complemento a dos:	1000 0000 0000 0000 . 0000 0000
Parte entera:	1000 0000 0000 0000
Valor decimal:	32.768
Parte fraccionaria:	0000 0000
Valor decimal:	0,0
Número completo:	-32.768

Obsérvese que para encontrar la magnitud de un número negativo se aplica el complemento a dos desde el LSB de la parte fraccionaria y se considera cualquier acarreo de la misma a la siguiente posición superior de la parte entera. En general, el número se considera como un valor de 24 bits dividido en dos partes, cuando se ha de sumar. La parte fraccionaria se procesa primero y cualquier acarreo se lleva a la parte entera.

2.2.6 NUMEROS BINARIOS EN NOTACION CIENTIFICA: PUNTO FLOTANTE

Las aplicaciones científicas utilizan frecuentemente números que varían entre grandes extremos de magnitud, aunque generalmente no requieren una precisión de dígitos equivalente a dicha magnitud. La versión binaria de la notación científica es la denominada representación en *punto flotante* (*floating point*) y permite representar números muy grandes y muy pequeños en un formato uniforme que simplifica las operaciones.

Como es sabido, en la notación científica un valor se representa por un número mixto (un dígito y una fracción) y una potencia de 10. En la notación en punto flotante se representará un valor por un número en binario (que se denomina *mantisa*), y una potencia de 2, también en binario (que se denomina

exponente).

La *mantisa*, más que definir un rango, define una precisión. En efecto, dos bytes permiten almacenar 65.536 (2^{16}) números enteros o unos 5 dígitos decimales ($2^{-16} \simeq 1{,}5 \cdot 10^{-5}$) de precisión en notación fraccionaria, lo que no parece bastante amplio para muchos problemas. Si se utiliza otro byte más, será posible almacenar 16.777.215 (2^{24}) números enteros o unos 8 dígitos decimales ($2^{-24} \simeq 6{,}0 \cdot 10^{-8}$) en notación fraccionaria, lo que es probablemente una solución intermedia para resolver el conflicto entre las necesidades de almacenamiento y la precisión.

Si se desea obtener la máxima precisión, se debe operar en formato fraccionario, considerando el orden de magnitud en el exponente, al que se le asigna un byte (2^8 = 256 valores enteros). Para ello se deben guardar en la mantisa tantos bits significativos como sea posible. Puesto que existe un número limitado de bits para almacenar la mantisa (en el caso considerado, 24) será necesario eliminar los bits despreciables y conservar sólo los significativos, lo que se lleva a cabo "normalizando" la mantisa. Así, el primer 1 se sitúa inmediatamente a la derecha de un supuesto punto binario sitado a la izquierda de los 3 bytes de la mantisa, tal como muestra la **Figura 2.6**. De esta forma se asegura la máxima precisión, ya que no hay ceros despreciables a la izquierda del primer 1, ni parte truncada de la mantisa al final. El primer bit de la mantisa será, por tanto, siempre un 1 y, evidentemente, esta normalización se reflejará en el exponente, ya que eliminar un cero a la izquierda de una cifra fraccionaria binaria equivale a multiplicarla por 2, por lo que el exponente ha de ser disminuido en una unidad.

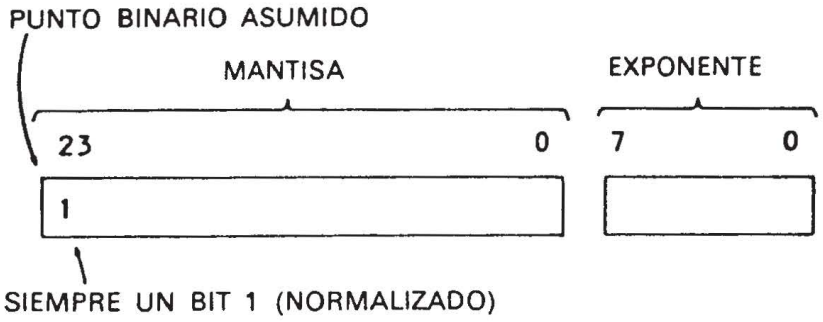


Figura 2.6

Representación normalizada del formato en punto flotante

Ahora bien, aún es necesario considerar un bit de signo, para permitir la representación de números negativos, lo que se lleva a cabo como sigue. Como todo número en punto flotante va a estar normalizado, se sabe que el MSB es un 1, por lo que se puede utilizar este mismo bit para representar el signo, igual que se ha hecho para representar los números enteros con signo, pero no olvidando nunca que el bit de signo (0 = positivo; 1 = negativo) "oculta" a su vez el primer dígito significativo (un 1) de la mantisa.

En definitiva, el formato final de la mantisa queda de la forma mostrada en la **Figura 2.7**, donde asimismo se ha considerado un exponente de un byte.

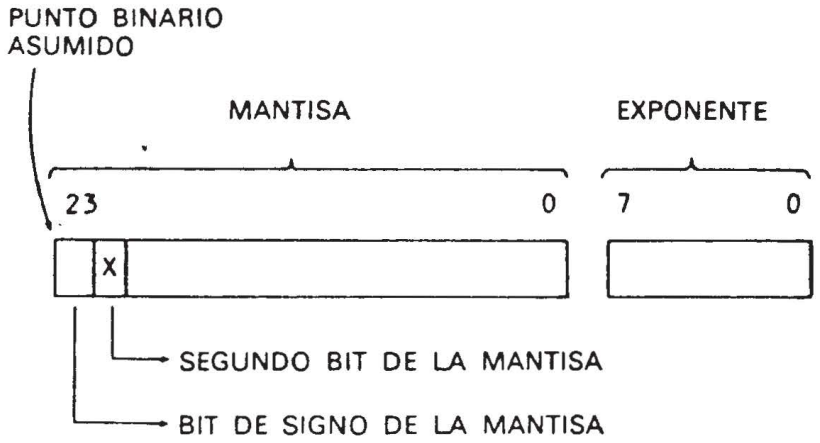


Figura 2.7
Formato de punto flotante

El *exponente* viene representado por un byte independiente que permitirá representar 256 (2^8) valores, es decir, de 0 a 255, lo que daría la posibilidad de representar números equivalentes a $5{,}79 \cdot 10^{76}$ (2^{255}). Sin embargo, si se desea poder representar también exponentes decimales negativos, habrá que codificar el exponente binario de alguna manera. Para ello se convierten los 8 bits del exponente binario en un *código exceso a 128* (*excess code 128*), consistente en añadir 128 a la potencia de 2 y luego almacenarlo como binario. Ello implica dividir el intervalo en dos partes, siendo su origen el número 128 (1000 0000): números mayores (hasta 256 = 1111

1111) representan exponentes positivos y números menores (hasta 0 = 0000 0000) representan exponentes negativos, tal como muestra el ejemplo de la **Figura 2.8.**

VALOR	DESPUES DE AJUSTAR POR RESTA DE 128	POTENCIA DE DOS
11111111	01111111	+127
11010000	01010000	+80
10000101	00000101	+5
10000000	00000000	0
01111111	11111111	-1
01111110	11111110	-2
00001111	10001111	-112
00000000	10000000	-128

Figura 2.8

Formato y valores del exponente en formato de punto flotante

El código exceso a 128 se aplica para simplificar el manejo de números en punto flotante como una simple unidad y permitirá obtener una serie de números desde $2{,}9 \cdot 10^{-39}$ (2^{-128}) hasta $1{,}7 \cdot 10^{+38}$ (2^{127}).

Las consideraciones realizadas acerca de esta codificación podrán quedar convenientemente fijadas considerando los ejemplos mostrados en las **Figuras 2.9 y 2.10.**

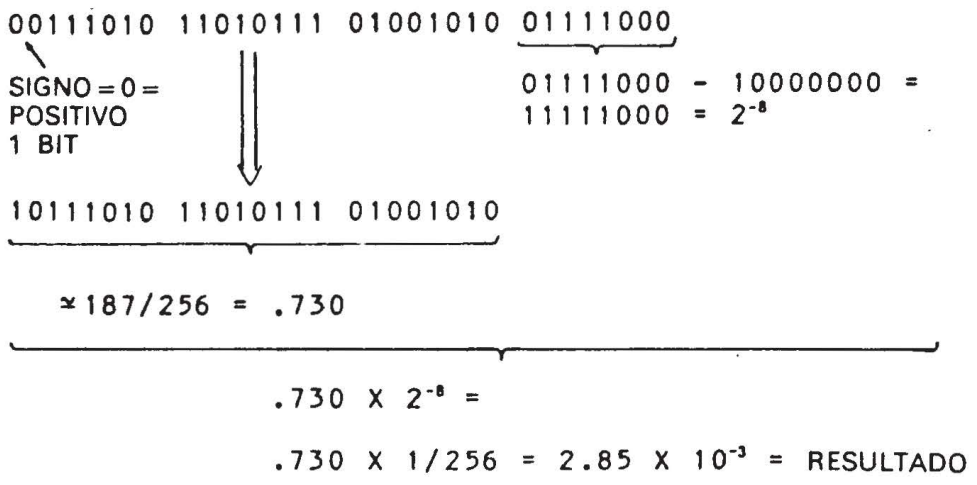


Figura 2.9

Ejemplo de número positivo codificado en punto flotante

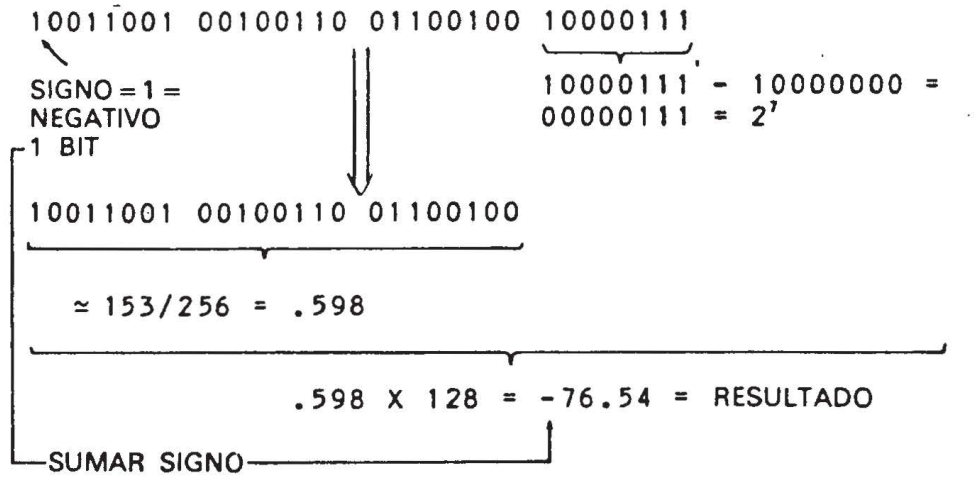


Figura 2.10

Ejemplo de número negativo codificado en punto flotante

2.2.7 VARIANTES DE LOS NUMEROS BINARIOS

Debido a que los números binarios tienden a ser muy largos, los dígitos binarios a menudo se asocian en grupos de tres o cuatro. Los números se representan entonces en base 8 (octal), en decimal codificado en binario (BCD) o en base 16 (hexadecimal).

2.2.7.1 SISTEMA OCTAL

Comparado con el sistema decimal, el sistema binario requiere muchos

más dígitos para representar un número, lo que lo hace incómodo y expuesto a errores cuando tiene que ser manejado por el hombre. Por ello se pensó en un sistema más simple, como el octal, que aún se utiliza en algunas máquinas.

El sistema octal utiliza la base 8, por lo que sólo emplea los dígitos del 0 al 7. Como además el dígito de mayor valor (el 7) coincide con el número binario de mayor valor representable con 3 bits (111), la conversión de octal a decimal es inmediata: basta agrupar el número binario de tres en tres bits, empezando por el LSB y convertir cada uno de los grupos resultantes en un dígito octal, tal como muestra el ejemplo de la **Figura 2.11.**

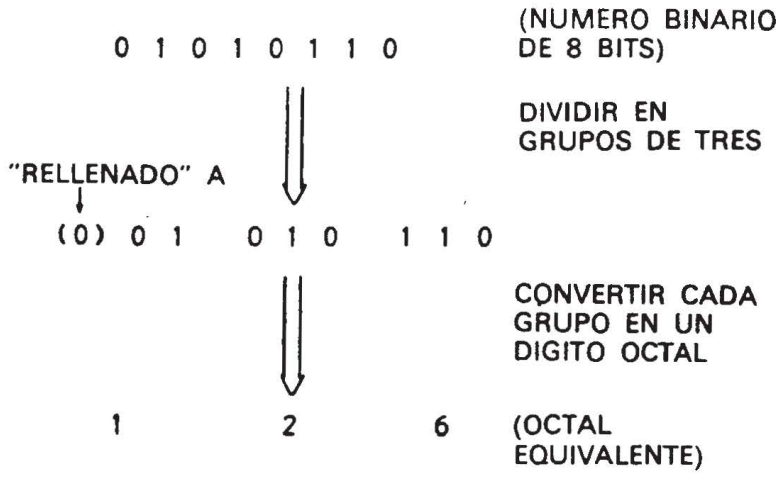


Figura 2.11

Paso de binario a octal

La conversión inversa, es decir, de octal a binario, es evidente si se observa asimismo la **Figura 2.11.**

Por su parte, las conversiones entre los sistemas octal y decimal se rigen por las mismas reglas utilizadas en las conversiones entre los sistemas binario y decimal.

2.2.7.2 SISTEMA BCD

Mientras que el ordenador sólo puede trabajar con números binarios, el operador del mismo lo haría más cómodamente utilizando números decimales.

Existen algunos dispositivos de E/S que podrían ser diseñados de manera que del lado humano recibieran y transmitieran información en sistema decimal y del lado del ordenador lo hicieran en sistema binario. Como tales dispositivos son complejos, caros y lentos, se diseñó el sistema *decimal-codificado-en-binario* (*Binary Coded Decimal*), una forma más simple de resolver el problema, y que aún utilizan algunos ordenadores.

En el sistema BCD, cada dígito decimal se representa con un grupo de 4 dígitos binarios, siendo realmente un código binario puro, limitado cada 4 bits a representar sólo hasta el número decimal 9. La **Figura 2.12** ilustra esta definición.

Decimal	*Binario puro*	*Código BCD*
0	0	0 0 0 0
1	0 1	0 0 0 1
2	1 0	0 0 1 0
3	1 1	0 0 1 1
4	1 0 0	0 1 0 0
5	1 0 1	0 1 0 1
6	1 1 0	0 1 1 0
7	1 1 1	0 1 1 1
8	1 0 0 0	1 0 0 0
9	1 0 0 1	1 0 0 1

Figura 2.12
Sistema de codificación BCD

Tanto la conversión de un código BCD a una cadena de dígitos binarios como la aritmética en BCD son complejos, por lo que no se tratarán aquí, más teniendo en cuenta que los ordenadores más populares no utilizan este sistema de codificación.

2.2.7.3 SISTEMA HEXADECIMAL

En el sistema BCD cada dígito se representa por un grupo de 4 dígitos binarios, con los cuáles se podría disponer de una capacidad de 16 combinaciones de bits. Las últimas 6 combinaciones (del 1010 al 1111) no se usan, de hecho, son ilegales en BCD y, por tanto, no se aprovechan.

El sistema hexadecimal es, en realidad, una extensión del sistema BCD

que utiliza estas 6 combinaciones para representar unos caracteres específicos. Por supuesto, se pueden utilizar estas combinaciones de cualquier forma que se desee, asignándose en las aplicaciones actuales a las 6 primeras letras del alfabeto (de la A a la F). Este sistema en base 16 es la codificación de datos binarios más utilizada actualmente, y se muestra en la **Figura 2.13**.

Decimal	*Binario puro*	*Hexadecimal*
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
10	1 0 1 0	A
11	1 0 1 1	B
12	1 1 0 0	C
13	1 1 0 1	D
14	1 1 1 0	E
15	1 1 1 1	F

Figura 2.13
Sistema de codificación hexadecimal

Pasar de binario a hexadecimal es tan simple como pasar de binario a octal. Empezando por el LSB, se divide el número binario en grupos de 4 bits. La falta de bits a la izquierda para completar un grupo de 4 se suple con ceros. A continuación se convierte cada grupo de 4 bits en un dígito hexadecimal y el resultado será la representación hexadecimal del valor binario, que es la cuarta parte de larga en caracteres.

La **Figura 2.14** muestra un ejemplo de este procedimiento y permite deducir que la conversión de hexadecimal a binario es el procedimiento equivalente a la inversa.

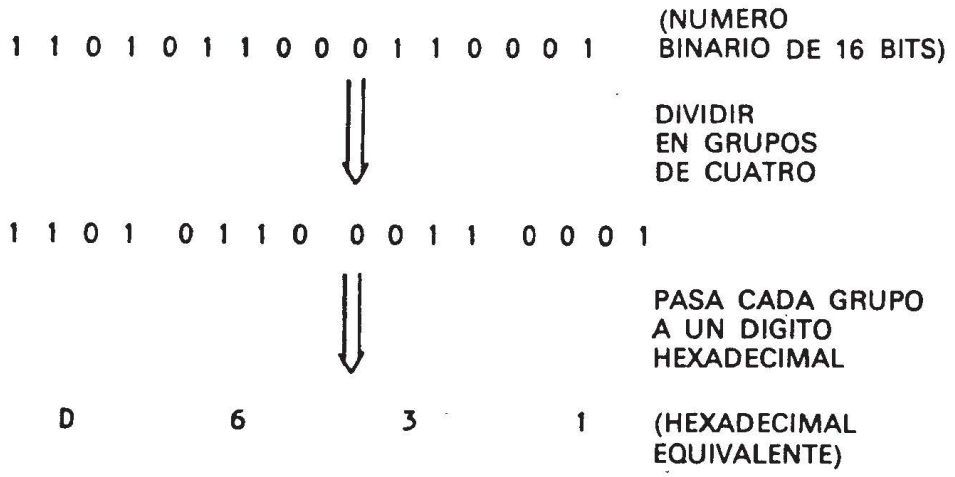


Figura 2.14

Paso de binario a hexadecimal

Las conversiones hexadecimal/decimal y viceversa siguen las reglas convencionales de cambio de base, "sumar las potencias de la base" y "dividir por la base y guardar los restos".

2.2.8 CODIGOS DE CARACTERES

Un ordenador no sería muy útil si exigiera que los datos a introducir fuesen secuencias de dígitos binarios, o si las soluciones las diera en un formato binario, codificado o sin codificar. Debería ser posible para un ordenador procesar textos y otra información numérica o no numérica en su interacción con el operador.

Si se tiene en cuenta que las combinaciones de dígitos binarios de cada palabra pueden utilizarse de muchas maneras, entonces todos los códigos binarios que se han utilizado hasta ahora para representar datos numéricos podrían también usarse para representar letras del abecedario, caracteres de dígitos, o cualesquiera otros caracteres impresos. Con tal que una instrucción o conjunto de instrucciones (programa) interprete correctamente los dígitos binarios de una palabra, no deben suscitarse confusiones o ambigüedades. Si la lógica del programa no interpreta correctamente el contenido de las palabras, dicha lógica será errónea y deberá ser corregida.

Así pues, la mayoría de los ordenadores disponen de un juego de caracteres adecuadamente codificados que permiten interaccionar con el

operador de forma sencilla. Esta capacidad de codificación se divide normalmente en los siguientes conjuntos de caracteres necesarios:

* 10 dígitos numéricos
* 26 letras minúsculas
* 26 letras mayúsculas
* 25 símbolos especiales.

Estos cuatro conjuntos suman un total de 87 caracteres, por lo que será necesario utilizar, como mínimo, una palabra de 7 bits, que permite 128 (2^7) combinaciones binarias, lo que sería suficiente, en principio, para las necesidades expuestas.

Se han establecido así dos códigos de caracteres. El primero, implantado por IBM para grandes ordenadores es el código EBCDIC (*Extended Binary Coded Decimal Interchange Code*), por lo que no está muy extendido entre los usuarios de ordendadores personales. Sin embargo, prácticamente todos los demás fabricantes de ordenadores han adoptado el denominado código ASCII (*American Standard Code for Information Interchange*), lo que lo ha hecho considerablemente popular, tanto en su versión básica como en su versión extendida.

2.2.8.1 CODIGO ASCII

El juego de caracteres ASCII, mostrado en la **Figura 2.15**, posee dos partes claramente diferenciadas: los códigos 0-31 o "códigos de control" y los códigos "32-127 o "caracteres convencionales".

Los *códigos de control*, representados por los 32 primeros caracteres ASCII (5 LSB) se reservan para pasar información especial, es decir, no se utilizan para pasar datos, sino que proporcionan instrucciones de acción, señales de formato y controles de comunicación entre periféricos. Hay una amplia gama de funciones que pueden realizar estos 32 códigos y sus usos pueden variar en circunstancias diferentes. Aunque dichos usos no se detallarán aquí, un somero análisis de la **Figura 2.16** permitirá adquirir una idea aproximada de la aplicación de estos códigos.

Ahora bien, si estos caracteres no se interpretan como códigos de control, tienen una representación gráfica, tal como la mostrada en la **Figura 2.17**. Lo que determina que el mismo código se interprete como una instrucción

de control o como un carácter gráfico es su utilización por medio del programa correspondiente.

b7 →					0	0	0	0	1	1	1	1
b6 →					0	0	1	1	0	0	1	1
b5 →					0	1	0	1	0	1	0	1
b4	b3	b2	b1	Columna / Fila	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	;
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

Figura 2.15
Codigo ASCII básico

Los *caracteres convencionales* están agrupados en varios bloques, como se puede observar en la **Figura 2.15**.

El primer bloque de 16 códigos (a partir del sexto bit), del 32 (20 h) al 47 (2F h), comprende signos especiales y de puntuación.

El segundo bloque de 16 códigos, del 49 (30 h) al 63 (3F h), contiene los 10 caracteres numéricos y otros signos de puntuación.

A partir del séptimo bit están comprendidos los caracteres del alfabeto: primero las letras mayúsculas, del 65 (41 h) al 90 (5A h), dentro de un bloque de 32 códigos [64 (40 h) a 95 (5F h)] y luego las letras minúsculas, del 97 (61 h) al 122 (6A h), dentro de otro bloque de 32 códigos [96 (60 h) a 127 (7F h)]. Estos dos bloque se completan ambos con caracteres de puntuación.

Código decimal	*Código hex*	*Tecla-control*	*Nombre*	*Descripción*
0	00	^@	NUL	Carácter nulo
1	01	^A	SOH	Inicio de cabecera
2	02	^B	STX	Inicio de texto
3	03	^C	ETX	Fin de texto
4	04	^D	EOT	Fin de transmisión
5	05	^E	ENQ	Requerimiento
6	06	^F	ACK	Reconocimiento
7	07	^G	BEL	Campana
8	08	^H	BS	Espacio atrás
9	09	^I	HT	Tabulación horizontal
10	0A	^J	LF	Avance de línea
11	0B	^K	VT	Tabulación vertical
12	0C	^L	FF	Página nueva
13	0D	^M	CR	Retorno de carro
14	0E	^N	SO	*Shift* desactivado
15	0F	^O	SI	*Shift* activado
16	10	^P	DEL	Borrar
17	11	^Q	DC1	Control dispositivo 1
18	12	^R	DC2	Control dispositivo 2
19	13	^S	DC3	Control dispositivo 3
20	14	^T	DC4	Control dispositivo 4
21	15	^U	NAK	Reconocimiento negativo
22	16	^V	SYN	Sincronización
23	17	^W	ETB	Fin de bloque de texto
24	18	^X	CAN	Cancelar
25	19	^Y	EM	Fin de medio
26	1A	^Z	SUB	Sustituir
27	1B	^[	ESC	*Escape*
28	1C	^/	FS	Separador de fichero
29	1D	^]	GS	Separador de grupo
30	1E	^^	RS	Separador de registro
31	1F	^_	US	Separador de unidad

Figura 2.16

Caracteres ASCII de control

Obsérvese, como curiosidad, que la tabla ASCII está diseñada de tal forma, que todos los códigos *X*1 h en los que comienza un bloque, significan un inicio o un primer elemento de un conjunto:

01 h = Inicio de cabecera (SOH)

11 h = Control dispositivo 1 (DC1) [no es comienzo de bloque]

21 h = ! (de todos los signos especiales, el más parecido a un 1)

31 h = 1

41 h = A

51 h = Q [no es comienzo de bloque]

61 h = a

71 h = q [no es comienzo de bloque]

Figura 2.17

Caracteres ASCII de control interpretados gráficamente

De la misma forma, parecen haberse relacionado los códigos *X*0 h de comienzo de un bloque con el valor nulo:

00 h = Carácter nulo (NUL)

10 h = Borrar (DEL) [no es comienzo de bloque]

20 h = Espacio

30 h = 0

40 h = @ (Control-@ = NUL)

50 h = P [no es comienzo de bloque]

60 h = ` (acento grave, único signo especial que no corre espacio en impresión)

70 h = p [no es comienzo de bloque]

2.2.8.2 EL OCTAVO BIT: BIT DE PARIDAD

Ya que sólo son necesarios 7 bits para códigos de carácter, pero se

utilizan normalmente palabras de 8 bits, el octavo bit se puede utilizar para detectar errores en la transmisión de palabras, denominándose en este caso *bit de paridad* (*parity bit*). Este bit se usa para asegurar que desde que se genera un byte de carácter hasta que se lee de nuevo, no se ha alterado erróneamente ningún bit. Para ello se pone este bit a 1 o a 0 para que el número de bits 1 en el byte sea siempre impar o siempre par (selección de paridad).

Si se selecciona *paridad impar* (*odd parity*), el bit de paridad será puesto a 1 o a 0 para conseguir que el número total de bits que son 1 sea siempre impar, como se muestra en los dos ejemplos siguientes:

```
Código de 7 bits:       X100 1011
Bits 1:                 4
Bit de paridad para
 que sea impar:         1
Código de 8 bits:       1100 1011

Código de 7 bits:       X101 0100
Bits 1:                 3
Bit de paridad para
 que sea impar:         0
Código de 8 bits:       0101 0100
```

De igual manera, si se selecciona *paridad par* (*even parity*), el bit de paridad será puesto a 1 o a 0 para conseguir que el número total de bits que son 1 sea siempre par, como se muestra en los siguientes ejemplos:

```
Código de 7 bits:       X100 1011
Bits 1:                 4
Bit de paridad para
 que sea par:           0
Código de 8 bits:       0100 1011
```

Código de 7 bits:	X101 0100
Bits 1:	3
Bit de paridad para que sea par:	1
Código de 8 bits:	1101 0100

De esta forma, si se selecciona por ejemplo paridad par, entonces siempre que se detecte un número impar de bits 1 en un byte de carácter, dicho byte será erróneo. Análogamente, si se selecciona paridad impar, entonces siempre que se detecte un número par de bits 1 en un byte, éste será erróneo.

2.2.8.3 EL OCTAVO BIT: CODIGO ASCII EXTENDIDO

La segunda utilización del octavo bit de una palabra de código de carácter, y quizá la más utilizada últimamente, es la ampliación de la tabla de caracteres con 128 códigos más para formar el denominado *código ASCII extendido*. Hablando con propiedad, sólo los códigos 0-127 son los caracteres ASCII, pero cada vez más frecuentemente se da este nombre a los códigos 0-255.

No existe coincidencia general de por qué se utilizaron estos códigos de caracteres 128-255, pero sí de que estos caracteres se diseñaron especialmente para los *ordenadores personales* (*Personal Computers*). Debido a la importancia y popularidad de estos equipos, estos caracteres ASCII extendidos se han adoptado ya en el juego de muchos ordenadores que están relativamente muy distantes del PC. Este particular grupo de caracteres sigue el camino de convertirse en una norma pero, de momento, no oficial, ya que aún existen muchos equipos que no pueden trabajar con ellos.

Los caracteres ASCII extendidos, mostrados en la Figura 2.18, están organizados en tres grupos principales: caracteres extranjeros, caracteres de dibujo y caracteres científicos.

Los *caracteres extranjeros* (*foreign characters*) ocupan un bloque de 48 (16 x 3) códigos, del 128 (90 h) al 175 (AF h), e incluyen todo lo necesario para acomodar los principales idiomas europeos, además del inglés: caracteres alfabéticos con signos diacríticos, puntuaciones especiales y símbolos de

monedas.

128	80	Ç	144	90	É	160	A0	á	176	B0	▒	192	C0	└	208	D0	╨	224	E0	α	240	F0	≡
129	81	ü	145	91	æ	161	A1	í	177	B1	▒	193	C1	┴	209	D1	╤	225	E1	β	241	F1	±
130	82	é	146	92	Æ	162	A2	ó	178	B2	▒	194	C2	┬	210	D2	╥	226	E2	Γ	242	F2	≥
131	83	â	147	93	ô	163	A3	ú	179	B3	│	195	C3	├	211	D3	╙	227	E3	π	243	F3	≤
132	84	ä	148	94	ö	164	A4	ñ	180	B4	┤	196	C4	─	212	D4	╘	228	E4	Σ	244	F4	⌠
133	85	à	149	95	ò	165	A5	Ñ	181	B5	╡	197	C5	┼	213	D5	╒	229	E5	σ	245	F5	⌡
134	86	å	150	96	û	166	A6	ª	182	B6	╢	198	C6	╞	214	D6	╓	230	E6	µ	246	F6	÷
135	87	ç	151	97	ù	167	A7	º	183	B7	╖	199	C7	╟	215	D7	╫	231	E7	τ	247	F7	≈
136	88	ê	152	98	ÿ	168	A8	¿	184	B8	╕	200	C8	╚	216	D8	╪	232	E8	Φ	248	F8	°
137	89	ë	153	99	Ö	169	A9	⌐	185	B9	╣	201	C9	╔	217	D9	┘	233	E9	Θ	249	F9	∙
138	8A	è	154	9A	Ü	170	AA	¬	186	BA	║	202	CA	╩	218	DA	┌	234	EA	Ω	250	FA	·
139	8B	ï	155	9B	¢	171	AB	½	187	BB	╗	203	CB	╦	219	DB	▉	235	EB	δ	251	FB	√
140	8C	î	156	9C	£	172	AC	¼	188	BC	╝	204	CC	╠	220	DC	▄	236	EC	∞	252	FC	n
141	8D	ì	157	9D	¥	173	AD	¡	189	BD	╜	205	CD	═	221	DD	▌	237	ED	φ	253	FD	²
142	8E	Ä	158	9E	₧	174	AE	«	190	BE	╛	206	CE	╬	222	DE	▐	238	EE	ε	254	FE	■
143	8F	Å	159	9F	ƒ	175	AF	»	191	BF	┐	207	CF	╧	223	DF	▀	239	EF	∩	255	FF	

Figura 2.18

Código ASCII extendido

Los *caracteres gráficos* (*graphic characters*) ocupan el siguiente bloque de 48 códigos, del 176 (B0 h) al 223 (CF h), y están diseñados para permitir que algunos programas realicen dibujos utilizando el juego de caracteres del ordenador. Hay tres subgrupos de caracteres gráficos: los de *sombreo* con grado de intensidad variable (176-178), los de *dibujar rectángulos* (179-218) y los de *máxima densidad* (219-223).

Los *caracteres científicos* (*scientific characters*) ocupan el último bloque de 32 códigos, del 224 (E0 h) al 255 (FF h), e incluyen las letras griegas utilizadas normalmente por los científicos y algunos símbolos matemáticos especiales. Aunque estos símbolos no cubren todas las necesidades matemáticas, científicas y de ingeniería, realmente solventan algunas de las necesidades más inmediatas.

2.3 ORGANIZACION DE LOS DATOS EN LOS ORDENADORES

Los ordenadores digitales están diseñados para trabajar con dígitos binarios. Independientemente del significado que el programador le dé a estos bits, la máquina los interpreta únicamente como 1 y 0. El hombre es el que

asigna un cierto significado a un bit o a un grupo de bits, para cumplir los requerimientos de un cierto problema. Resulta evidente, pues, que una misma combinación de unos y ceros puede tener varios significados en función de cómo se les agrupa, interpretándolos mediante distintos sistemas de codificación.

Así pues, se puede deducir que es el sistema de codificación el que le da significado a los dígitos binarios. Cabe considerar entonces que los grupos de dígitos binarios que se interpretarán por un sistema específico son los *datos* (*data*), y que éstos son los que serán procesados por los ordenadores digitales. Por otro lado, se denominará *información* a aquéllos datos que tienen significado e interpretación.

Es decir, internamente el microprocesador procesa datos, pero el sistema opera con información. Los dispositivos de entrada al ordenador toman la información externa, la configuran para su interpretación y la presentan al microprocesador, que los procesa como datos. El ordenador presenta los datos procesados al dispositivo de salida, el cual añade a los mismos el significado que los convierte en información.

El proceso descrito se muestra en la **Figura 2.19** para un teclado, una pantalla y un ordenador, dejando patente la importancia de la codificación de los datos que se ha estudiado hasta ahora.

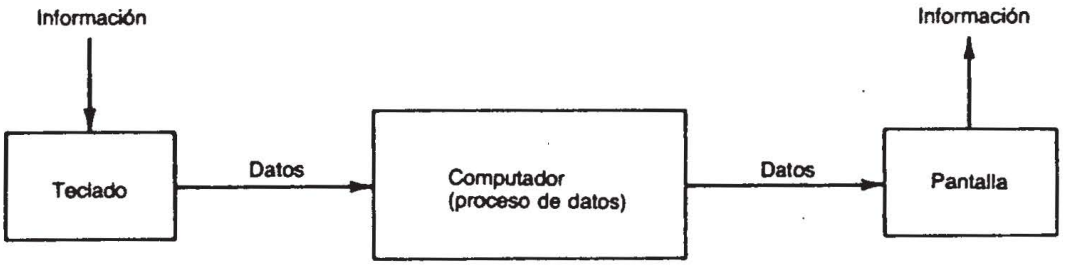


Figura 2.19
Relación entre datos e información

Por otro lado, ya se ha visto anteriormente que en los microprocesadores actuales se utiliza profusamente el byte de 8 bits y las palabras multibyte, siendo estas últimas el resultado de concatenar dos o más

bytes, que se tratan como una palabra única. Tales palabras multibyte a menudo contienen bits que representan la información en dos o más formas de codificación. El proceso de incluir dos o más elementos de información en una sola palabra de ordenador, utilizando grupos de bits diferentes para especificar cada elemento y con sistemas de codificación diferente se conoce como *empaquetado* (*packing*) de los datos, y se suele realizar para dotar al sistema de mayor rendimiento de almacenamiento y velocidad.

Sin embargo, una palabra empaquetada con bits de diferentes sistemas de codificación no puede ser tratada por el microprocesador para dar un resultado significativo, por lo que debe ser desempaquetada antes de ser procesada. Así pues, se define el *desempaquetado* (*unpacking*) como el proceso de separar los distintos elementos de una palabra empaquetada con objeto de que pueda ser procesada o memorizada. Evidentemente, de las dos operaiones descritas ha de encargarse un juego de instrucciones (programa) diseñado a tal fin, y que gestiona los distintos elementos físicos del ordenador.

2.4 LA LOGICA DEL ORDENADOR

La aritmética es la base de gran cantidad de funciones que realiza un ordenador. La suma y la resta son las operaciones más sencillas y, con mucho, las más normales, porque se utilizan para muchos fines. Hasta hace poco tiempo los ordenadores no podían hacer directamente las operaciones de multiplicación y división, por lo que era necesario diseñar los algoritmos apropiados para realizar estas operaciones. Hoy día, sin embargo, los modernos microprocesadores que constituyen la base de los ordenadores personales pueden efectuar las cuatro operaciones básicas.

No obstante, los microprocesadores no sólo pueden efectuar operaciones, aunque la aritmética conforma una parte importante de las funciones del ordenador. Lo que dota a estos sistemas de mayor potencia que a las simples calculadoras es una variedad de instrucciones, conocidas como *lógica del ordenador*, que permiten al ordenador ajustar lo que se está haciendo a la situación práctica, es decir, proporciona la base para la toma de decisiones y la comprobación de condiciones.

El fundamento de las operaciones lógicas es el *álgebra de Boole*, que usa los dígitos binarios 1 y 0 para definir decisiones lógicas. Todas las

operaciones lógicas trabajan "bit a bit", por lo que no hay acarreos a otras posiciones. Siempre hay dos operandos y un resultado, y el análisis de la operación se hace con ayuda de una *tabla de verdad* (*truth table*), tabla que define los resultados de los operandos y que contiene todos los posibles estados binarios de éstos.

Las operaciones lógicas se realizan físicamente en un circuito integrado compuesto de varios biestables y llamado *puerta lógica*, elemento lógico cuya salida binaria queda determinada en función de las entradas, de acuerdo con una operación lógica específica. Las operaciones lógicas que son capaces de realizar los ordenadores vienen representadas por los *operadores booleanos* OR, AND, XOR y NOT.

La *operación OR* (*O*) se define para dos enteros I y J por la frase: *"Si I o J, o ambos, son iguales a 1, entonces el resultado es 1; en cualquier otro caso, el resultado es 0"*. Para representar OR se usa el signo más (+) y su tabla de verdad es la representada en la **Figura 2.20.**

Entradas		Salida =
I	J	I + J
0	0	0
0	1	1
1	0	1
1	1	1

Figura 2.20
Tabla de verdad para la operación OR

La operación OR se suele utilizar para "poner a 1" un bit. Así, si se tiene un byte con el MSB a 0 y se desea que éste tome el valor 1, se diseña la *máscara* (*mask*) adecuada para que sólo cambie este bit después de una operación OR, como muestra el siguiente ejemplo:

```
Byte original:          0100 0111
Máscara para el MSB:    1000 0000
OR
Resultado:              1100 0111
```

La *operación AND* (*Y*) se define para dos enteros I y J por la frase: *"Si I y J son ambos 1, entonces el resultado es 1; en cualquier otro caso el resultado es 0"*. Para representar AND se usa el punto (·) y su tabla de verdad es la representada en la **Figura 2.21.**

Entradas		Salida =
I	J	I · J
0	0	0
0	1	0
1	0	0
1	1	1

Figura 2.21
Tabla de verdad para la operación AND

La operación AND se utiliza fundamentalmente para "poner a 0" un bit o "quitar" ciertas partes de un byte. Así, si se quiere poner a cero los 4 MSB de un byte, se diseña la *máscara* adecuada para que el resultado de la operación AND sólo mantenga los 4 LSB, como muesta el siguiente ejemplo:

```
Byte original:              1001 1010
Máscara para los 4 LSB:     0000 1111
AND
Resultado:                  0000 1010
```

La *operación XOR* (*O-exclusiva*) diferencia entre dígitos binarios de entrada iguales y dígitos binarios de entrada diferentes. La salida es 1 cuando las entradas son diferentes y 0 cuando son iguales. Para representar XOR se usa el símbolo ⊕ y su tabla de verdad es la representada en la **Figura 2.22.**

Resulta evidente por su propia definición, que la utilización principal de la operación XOR es la comparación de dos bits o de dos palabras.

La *operación NOT* (*NO*) realiza el complemento a uno de un dígito binario, es decir, cambia 1 por 0 y 0 por 1. Para representar NOT se coloca una barra sobre el dígito.

Entradas		Salida =
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Figura 2.22
Tabla de verdad para la operación XOR

La representación de los componentes lógicos que realizan estas operaciones (puertas lógicas) se muestra en la **Figura 2.23.**

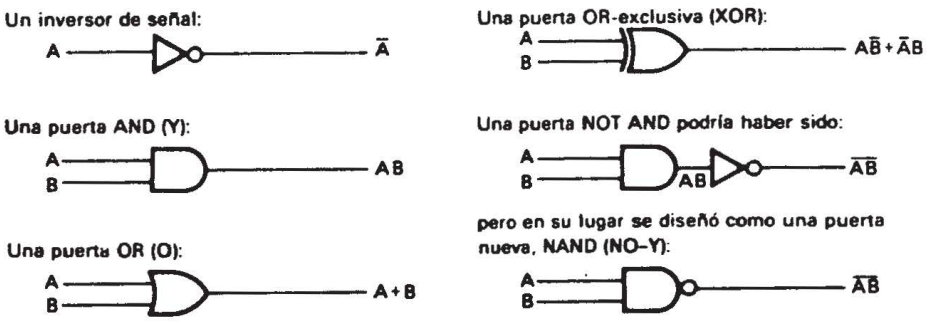


Figura 2.23
Representación de las puertas lógicas

2.4.1 COMBINACION DE OPERACIONES LOGICAS

Las operaciones booleanas pueden combinarse para producir cualquier salida deseada a partir de un grupo de entradas conocido, mediante el *Teorema de Morgan*, que puede escribirse en cualquiera de las dos formas siguientes:

$$\overline{A \cdot B} = \bar{A} + \bar{B} \qquad [2.2]$$

$$\overline{A + B} = \bar{A} \cdot \bar{B} \qquad [2.3]$$

Por tanto, un microprocesador sólo necesita dos operadores booleanos, OR y NOT, para generar todos los demás, ya que la siguiente operación genera AND a partir de OR y NOT:

$$A \cdot B = \overline{\overline{A} + \overline{B}} \qquad [2.4]$$

y análogamente, la siguiente operación genera XOR a partir de OR y NOT:

$$A \oplus B = \overline{(\overline{A} + B)} + \overline{(A + \overline{B})} \qquad [2.5]$$

Finalmente, a veces se utilizan operaciones combinadas de las anteriores. Así, combinando AND con NOT se genera *NAND* (*NO-Y*), que es la NOT de AND, y combinando OR con NOT se genera *NOR* (*NO-O*), que es la NOT de OR. A veces también se utiliza la operación *XNOR* (*NO-O-exclusiva*), que es la NOT de XOR.

2.4.2 OPERACIONES DE DESPLAZAMIENTO

Se denominan operaciones de desplazamiento aquéllas en las que los bits de una palabra son movidos conjuntamente hacia la derecha o hacia la izquierda respecto a la posición que ocupaban inicialmente. Estas operaciones se pueden utilizar para diseñar algoritmos de multiplicación y división binarias o para otras funciones que se estudiarán en su momento. Estas operaciones pueden ser clasificadas en rotaciones, desplazamientos lógicos y desplazamientos aritméticos.

La *rotación* (*rotation*) es el desplazamiento de los bits de tal forma, que el que sale del límite de la palabra por un extremo entra de nuevo en dicho límite por el extremo opuesto, o bien va al indicador de acarreo. Si los datos pasan a través del acarreo, se trata de una "rotación de 9 bits". Si los datos pasan por alto el acarreo, se tratará de una "rotación de 8 bits". En cualquier caso, el bit desplazado siempre va al acarreo, como muestra la **Figura 2.24.**

La rotación simplemente es útil para comprobar un bit de una vez, ya que en el sentido aritmético, el resultado no se puede nunca predecir.

El *desplazamiento lógico* (*logical shift*) es aquel en el que el bit desplazado va al acarreo, de forma que éste siempre almacena el resultado del desplazamiento; el límite opuesto de la palabra se rellena con ceros a medida que se desplaza cada bit. La **Figura 2.25** muestra dos ejemplos de desplazamiento lógico.

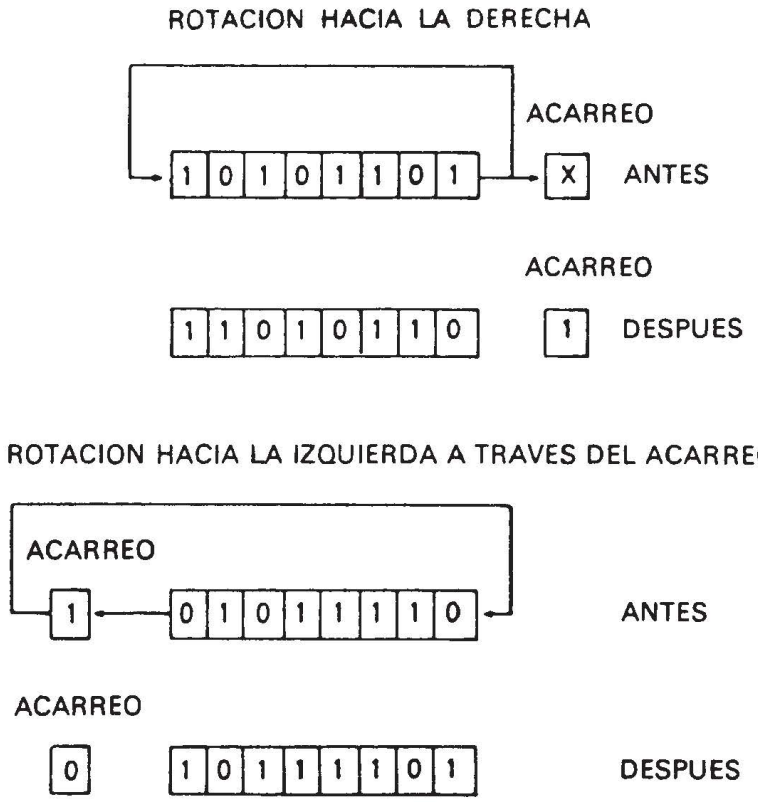


Figura 2.24

Rotaciones

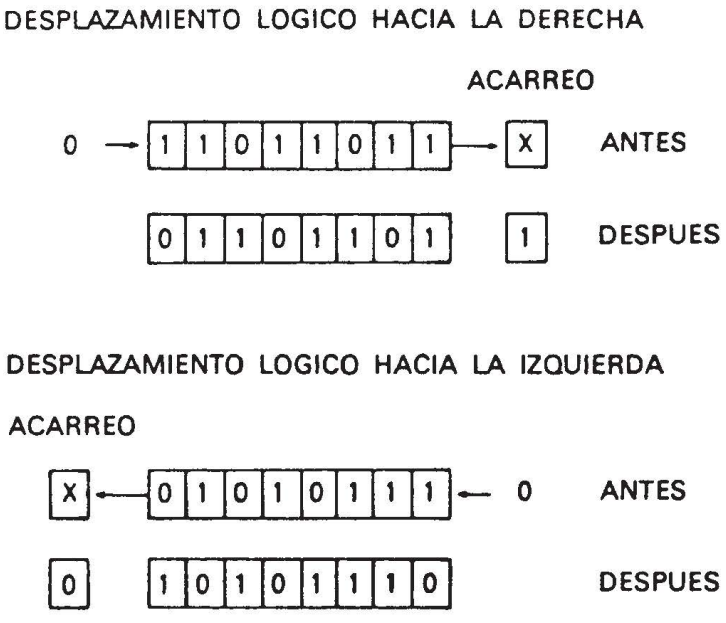


Figura 2.25

Desplazamientos lógicos

Obsérvese que un desplazamiento lógico tiene resultados plenamente previsibles: un desplazamiento lógico a la derecha divide el número original por 2 y pone el resto en el acarreo; un desplazamiento lógico a la izquierda multiplica el número original por 2, si se considera como MSB el acarreo. De esta manera la utilidad principal del desplazamiento lógico es multiplicar y dividir un número por potencias de 2, pero sólo cuando el número es binario puro, ya que si se utiliza un desplazamiento lógico de números con signo, el resultado no será correcto.

El *desplazamiento aritmético* (*arithmetic shift*) es análogo al desplazamiento lógico, pero conservando el signo, lo que permite resolver el problema planteado anteriormente. Para ello el desplazamiento a la derecha se hace de la misma forma que el lógico, pero en el acarreo quedaría representado el resto de la división, cambiado de signo, como se muestra en el ejemplo de la **Figura 2.26.**

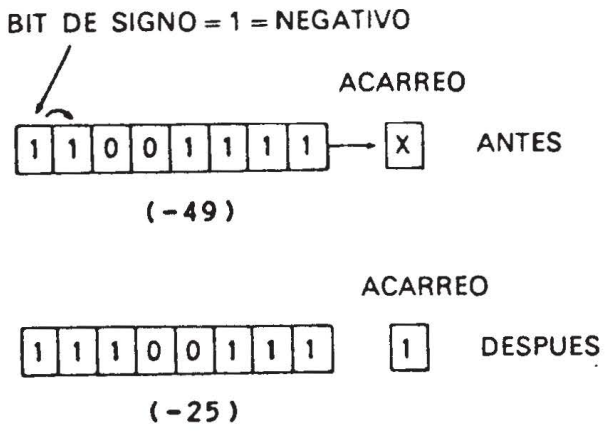


Figura 2.26
Desplazamiento aritmético a la derecha

Por su parte, el desplazamiento aritmético a la izquierda normalmente mantiene el bit de signo y desplaza el siguiente MSB al acarreo, tal como muestra la **Figura 2.27.**

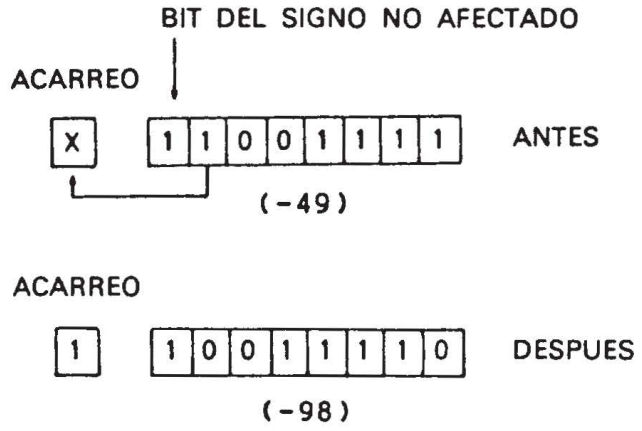


Figura 2.27

Desplazamiento aritmético a la izquierda

BIBLIOGRAFIA

Barden, W; *"Matemáticas para programadores. Sistemas de numeración y aritmética binaria"*, Anaya Multimedia, Madrid (1982).

Khambata, A.J.; *"Microprocesadores/microcomputadores. Arquitectura, software y sistemas"*, Gustavo Gili, Barcelona (1987).

Norton, P.; *"El IBM PC a fondo. Técnicas de programación avanzada"*, Anaya Multimedia, Madrid (1987).

Osborne, A. y **Bunnell, D.;** *"Introducción a las microcomputadoras"*, Vol. 0, 3ª ed., Osborne/McGraw-Hill, Madrid (1986).

Osborne, A.; *"Introducción a las microcomputadoras"*, Vol. 1, 2ª ed., Osborne/McGraw-Hill, Madrid (1986).

CAPITULO 3

EL SISTEMA DE MEMORIA

3.1 INTRODUCCION AL ALMACENAMIENTO DE LA INFORMACION

3.2 DISPOSITIVOS FISICOS DE MEMORIA

3.2.1 TIPOS DE ALMACENAMIENTO EN MEMORIAS DE SEMICONDUCTORES

3.3. TIPOS DE MEMORIA

3.4 ORGANIZACION FISICA DE LA MEMORIA

3.4.1 ORGANIZACION FISICA DE LA MEMORIA ROM

3.4.2 ORGANIZACION FISICA DE LA MEMORIA RAM

3.5 ORGANIZACION LOGICA DE LA MEMORIA RAM

3.6 GESTION DE MEMORIA

3.7 LA SEGMENTACION EN LOS ORDENADORES PERSONALES

3.1 INTRODUCCION AL ALMACENAMIENTO DE LA INFORMACION

El concepto de *memoria* (*memory*) implica el almacenamiento de información, y funciones de almacenamiento hay en todos los bloques funcionales del ordenador y fuera del mismo. Sin embargo, se suele utilizar este término para designar lo que es la *memoria principal* del ordenador, es decir, el lugar donde se almacena la información que será necesaria con carácter inmediato para el funcionamiento del sistema. Este tipo de *almacenamiento interno* se caracteriza por sus cortos tiempos de acceso, que varían desde 500 ns (10^{-9} s) a 2 μs (10^{-6} s).

Existen elementos de almacenamiento aún más rápidos en el bloque de control de la CPU, pero éstos son de muy baja capacidad. Por otro lado, al hacerse más grandes y potentes los ordenadores, aumenta la capacidad de almacenamiento necesaria. Sin embargo, como la velocidad a la que un ordenador puede utilizar la información suele estar limitada por la velocidad de procesamiento de la instrucciones, la necesidad de cantidades masivas de memoria con reducido tiempo de acceso no suele ser grande. Por ello se utiliza para almacenar grandes cantidades de información una *memoria auxiliar* o dispositivo de *almacenamiento externo*, cuyos componentes físicos principales son los *discos (disks)* y las *cintas* (*tapes*), y cuyos tiempos de acceso son considerablemente mayores.

También el tiempo de acceso depende de la forma en que se toma la información de la memoria o se entrega a ella.

Así, el método de *acceso aleatorio* (*random access*) permite acceder a cualquiera de las posiciones de almacenamiento sin tener que pasar por las demás, y una de sus características es el tiempo de acceso fijo. Este método es el que se utiliza principalmente para gestionar la memoria interna y la memoria auxiliar en forma de discos.

Por su parte, el método de *acceso secuencial* (*sequential access*) requiere recorrer todas las posiciones previas antes de llegar a la información buscada, siendo su característica principal el tiempo de acceso variable, ya que el tiempo necesario para obtener una información determinada dependerá de su posición en el dispositivo de almacenamiento. La memoria auxiliar en forma de cinta es el ejemplo más característico de este tipo de acceso.

En la **Figura 3.1** se muestra un esquema de clasificación de los distintos tipos de dispositivos de almacenamiento mencionados, especificándose sus características de volumen de almacenamiento, así como los tiempos y métodos de acceso.

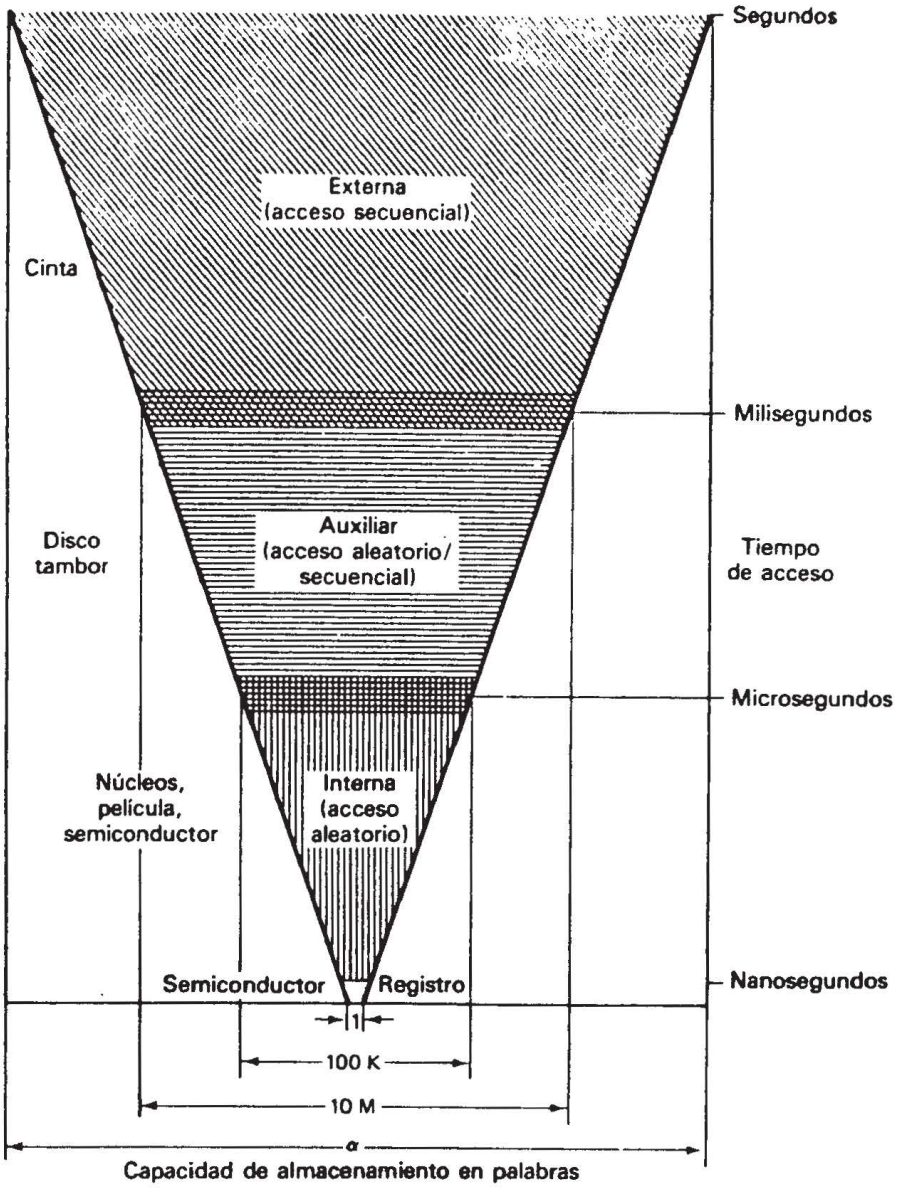


Figura 3.1

Clasificación de los dispositivos de almacenamiento

Es necesario destacar que en todo el estudio que sigue a continuación se utilizará el concepto de *memoria* para hacer referencia a la memoria principal o de almacenamiento interno.

3.2 DISPOSITIVOS FISICOS DE MEMORIA

Los dígitos binarios se almacenan en elementos de memoria, formados por agrupaciones de dispositivos biestables, cuya configuración física varía según la tecnología empleada.

Las *memorias de núcleos magnéticos* (*magnetic nuclei memories*), **Figura 3.2**, pertenecen a la primera técnica de almacenamiento masivo y rápido utilizada en los ordenadores, aunque hoy día prácticamente ya no se usan. El núcleo magnético para almacenar información binaria es un minúsculo toroide o anillo, de menos de 0,5 mm de diámetro, de material magnético, arrollado en torno a su centro con un conductor por el que circula corriente eléctrica y con otro conductor que permite detectar los cambios que puedan producirse en el campo magnético del núcleo. Según la dirección de la corriente eléctrica, el campo magnético tiene dos direcciones y el núcleo es capaz de conservar memoria del hecho de que la corriente haya circulado en un sentido o en otro, debido a que su comportamiento depende de un estado anterior (histéresis). Como la corriente sólo puede fluir en dos sentidos, el núcleo almacenará información binaria.

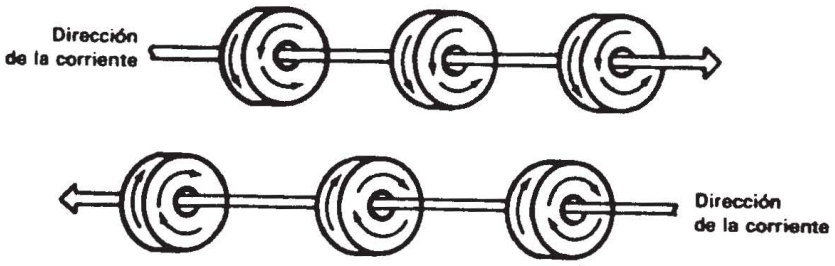


Figura 3.2
Memoria de núcleos magnéticos

Las *memorias de semiconductores* (*semiconductor memories*), **Figura 3.3**, son las memorias más utilizadas en la actualidad. Están formadas por conjuntos de puertas lógicas secuenciales, integradas y encapsuladas en un circuito integrado (*chip*) con muy alta escala de integración (*Very Large Scale Integration*), que permite integrar hasta 10.000 puertas lógicas en un solo circuito. Como las puertas pueden estar en conducción (*on*) o en corte

(*off*), se consigue almacenar en ellas información binaria.

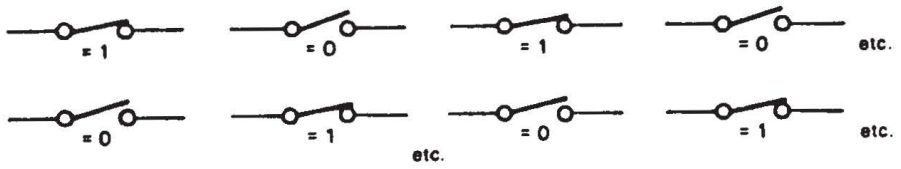


Figura 3.3
Memoria de semiconductores

Las *memorias de burbujas magnéticas* (*magnetic bubble memories*), **Figura 3.4**, constituyen la última innovación en dispositivos de memoria, aunque todavía no se usan ampliamente, debido a su alto coste y a que son relativamente lentas. Esta tecnología emplea una película delgada de material magnético dentro de la que pueden crearse zonas microscópicas magnéticas (burbujas) que tienen polaridad opuesta al resto de la película magnética. Haciendo aumentar o disminuir el campo magnético, se pueden crear y destruir a voluntad las burbujas, con lo que, si la existencia de una burbuja en una posición conocida se utiliza para representar el 1 binario, su ausencia puede servir para representar el 0.

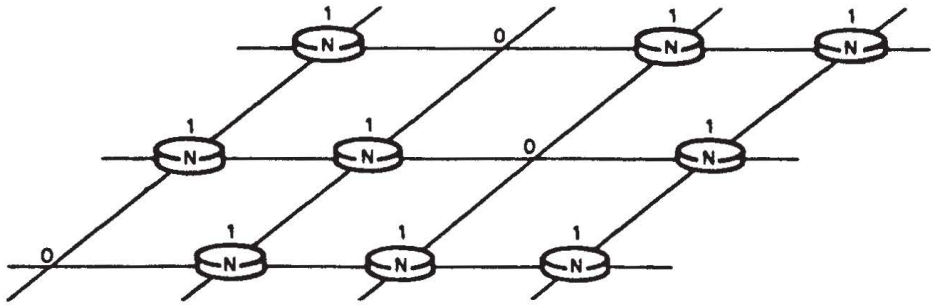


Figura 3.4
Memoria de burbujas magnéticas

Una característica fundamental de los dispositivos físicos de memoria es la *perdurabilidad* del almacenamiento. Se distingue así la memoria

"volátil" de la "no volátil".

La *memoria volátil* es aquélla en la que la información binaria se destruye cuando se desconecta la alimentación eléctrica, o incluso cuando se interrumpe momentáneamente. Cuando se restituye la alimentación, todos los bits de memoria pueden estar en estado 1 o 0, pero la combinación de los mismos es aleatoria y, por tanto, impredecible. El ejemplo más característico de este tipo de memoria lo constituyen las memorias de semiconductores, que pierden todos los datos almacenados al cortar su fuente de alimentación, lo que hay que tener en cuenta a la hora de diseñar el sistema que vaya a hacer uso de ellas.

La *memoria no volátil* es aquélla que mantiene la información almacenada, incluso cuando se interrumpe la alimentación. Cuando ésta se restituye, permanece la misma información almacenada previamente. Tanto las memorias de núcleos como las de burbujas retienen su carga magnética cuando se desconecta su suministro eléctrico, por lo que son ejemplos característicos de memorias no volátiles.

3.2.1 TIPOS DE ALMACENAMIENTO EN MEMORIAS DE SEMICONDUCTORES

La forma física de almacenar un dato en una memoria de semiconductores, la más ampliamente utilizada en la actualidad, varía según las características de los circuitos usados para construir el elemento de memoria considerándose, generalmente, tres tipos: estáticas, dinámicas y de sólo lectura.

Las *memorias estáticas* (*static memories*) almacenan la información en uno de dos estados estables. Se puede almacenar el dígito binario y también extraerlo, caracterizándose esta extracción por ser no destructiva, es decir, la información almacenada permanece en el elemento de memoria aunque sea transferida a otras partes del ordenador; por supuesto, mientras permanezca la alimentación, ya que la memoria de semiconductores en volátil.

Las *memorias dinámicas* (*dynamic memories*) almacenan la información utilizando el concepto de ausencia o presencia de carga eléctrica en un condensador. Como es sabido, un condensador es un dispositivo que almacena carga eléctrica en función de la energía disponible y del período de tiempo en que ésta se aplica al condensador. Como la carga de un condensador tiende

a desaparecer con el tiempo, la información almacenada como carga tiene que ser renovada periódicamente (normalmente, cada 2 ms) para que no se pierda antes de su extracción. Será necesario, pues, disponer de unos circuitos externos que permitan esta renovación, lo que encarece el sistema de memoria ya que, en principio, los elementos de memoria dinámica son más baratos que los de memoria estática. A pesar de ello, en la actualidad las memorias dinámicas suelen ser las más utilizadas como memorias para el almacenamiento de datos, debido a que permiten un grado de integración mucho mayor que las estáticas. Evidentemente las memorias dinámicas, al ser de semiconductores, también son volátiles.

Las *memorias de solo lectura* (*Read-Only-Memories*) son las únicas memorias de semiconductores no volátiles, ya que el estado de los dígitos binarios almacenados en ellas puede extraerse, pero no cambiarse. Así, la información se almacena en la ROM durante un proceso físico especial llamado "programación" o "reprogramación" y posteriormente, durante el proceso de funcionamiento normal, cualquier intento de introducir información en la ROM no la afecta en absoluto.

La información a escribir en cada posición del circuito integrado de memoria ROM la determina el usuario, suministrando al fabricante las instrucciones que debe contener. Estas instrucciones codificadas en binario se introducen de tal forma, que el valor 0 equivale a fundir un fusible en un elemento de memoria, lo que hace el proceso irreversible. Si la memoria ROM puede ser grabada por el usuario, se suele denominar *PROM* (*Programable ROM*).

Si es necesario borrar esta memoria y reprogramarla, se suelen utilizar otras técnicas, como el almacenamiento por carga, fenómeno de gran complejidad física que no se tratará aquí, pero que no es destructivo, como lo es la técnica del fusible. Así, se puede borrar una ROM de estas características exponiéndola a una luz ultravioleta de gran intensidad, para reprogramarla posteriormente; a este tipo de memoria se la conoce como *EPROM* (*Ereasable Programable ROM*). Por otra parte, la tecnología más nueva en este aspecto se refiere a las memorias ROM alterables eléctricamente, las *EAROM* (*Electrically Alterable ROM*), que se pueden reprogramar en milisegundos directamente, sin retirarlas del sistema. Como su uso es aún limitado, los costes de este último tipo de memoria impiden aún una amplia utilización.

3.3 TIPOS DE MEMORIA

Cualquier dispositivo de memoria debe poseer dos propiedades absolutamente necesarias:

* Cada posición que almacena un dígito binario debe ser direccionable unívocamente, es decir, debe ocupar un solo lugar en la memoria.

* Debe ser posible extraer el estado de cualquier dígito binario almacenado.

Con estas premisas pueden existir dos tipos de memoria, bajo el punto de vista lógico, y que son las utilizadas en todos los ordenadores:

* Memorias de lectura/escritura (*Read/Write Memory*)

* Memorias de sólo lectura (*Read-Only Memory*)

Las *memorias de lectura/escritura* son aquéllas en las que se puede leer (extraer) y escribir (almacenar) información, mientras que en las *memorias de solo lectura*, como ya se ha visto anteriormente, sólo se puede leer (extraer) información durante el proceso de funcionamiento normal del ordenador. Físicamente ambos tipos de memoria son de acceso aleatorio, es decir, se puede acceder directamente a cualquiera de sus elementos especificando su dirección.

Ahora bien, a las memorias de lectura/escritura se les ha dado en la práctica un nombre muy poco acertado: se las denomina *memorias RAM* (*Random Access Memories*), aunque tanto éstas como las memorias ROM son de acceso aleatorio. Así pues, en lo que sigue, y con objeto de mantener la nomenclatura que aparece en toda la bibliografía, se hablará de memoria RAM exclusivamente cuando se haga referencia a una memoria interna de lectura/escritura (es decir, la memoria de "trabajo"), teniendo siempre presente que el concepto de acceso aleatorio es mucho más amplio que el utilizado para esta definición. Obsérvese asimismo que como, en general, las memorias de lectura/escritura son de tipo dinámico, se podrá encontrar en algunas referencias el concepto *DRAM* (*Dynamic RAM*), absolutamente equivalente, pero haciendo referencia a una propiedad física del elemento de memoria: su carácter dinámico.

3.4 ORGANIZACION FISICA DE LA MEMORIA

Se entiende por organización física de la memoria a la disposición de los circuitos integrados que la componen sobre la placa base del ordenador. En general, las memorias de semiconductores se pueden organizar de la misma forma, independientemente de su uso como ROM o como RAM. Sin embargo, una memoria RAM requiere generalmente más lógica externa al elemento de memoria que la memoria ROM. La razón es evidente: las ROM requieren sólo circuitos lógicos para las operaciones de escritura. Por este motivo generalmente ambos tipos de memoria se organizan de forma diferente.

Gracias a los avances tecnológicos actuales en la integración de circuitos, existen circuitos integrados (*chips*) de memoria cada vez de mayor capacidad. Como es sabido, en función de la organización global del ordenador, cada circuito integrado de memoria ineracciona con el resto del sistema a través de los buses de control, dirección y datos. Dependiendo del modelo de circuito integrado, los buses de direcciones y de datos tienen distintos tamaños. El bus de direcciones tiene entre 16 líneas de señal (en los microprocesadores de 8 bits: 2^{16} = 65.536 direcciones) hasta 24 líneas de señal (en los microprocesadores de 16 bits: 2^{24} = 16.777.216 direcciones). Por su parte, el bus de datos puede tener 1, 4, 8 ó 16 líneas, cada una correspondiente a un bit.

Si el bus de datos del circuito integrado de memoria tiene 1 bit de anchura, se necesitan 8 circuitos de memoria conectados en paralelo para poder acceder a 1 byte ó 16 circuitos para poder acceder a una palabra de 2 bytes. Si el bus de datos tiene 4 bits, sólo se necesitarán 2 circuitos de memoria para acceder a 1 byte ó 4 para acceder a una palabra de 2 bytes. Finalmente, si el bus de datos tiene 8 bits de anchura, será suficiente 1 circuito de memoria para acceder a 1 byte y 2 circuitos para acceder a palabras de 2 bytes. Todo ello se muestra en el esquema de la Figura 3.5.

Según estas consideraciones, las memorias ROM se organizan por bytes, mientras que las memorias RAM se organizan por bits.

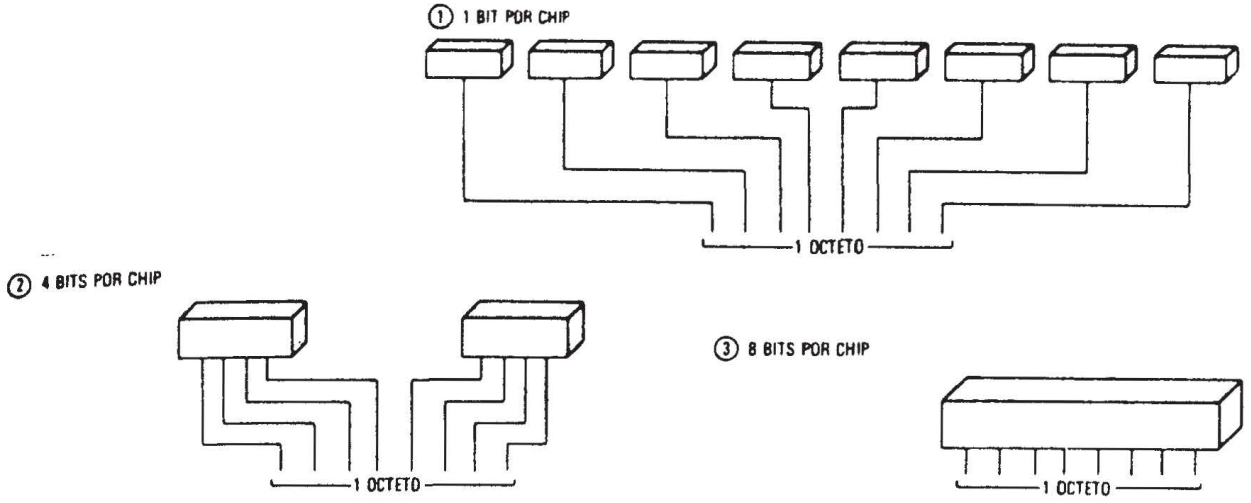


Figura 3.5

Acceso a 1 byte mediante circuitos de memoria de distintos tamaños

3.4.1 ORGANIZACION FISICA DE LA MEMORIA ROM

La cada vez menor necesidad de memoria ROM de los ordenadores actuales (sólo necesitan un programa de arranque) y la imposibilidad de escribir sobre ella, han hecho que la organización de este tipo de memoria se haga por bytes o, lo que es lo mismo, en una matriz de 2 dimensiones. Es decir, una palabra de 8 bits se encuentra en un eje **x** de la matriz de memoria y cada posición se estructura de acuerdo con el eje **y** de la matriz de memoria, tal como se muestra en el ejemplo de la **Figura 3.6**, donde se ha representado una memoria ROM de 1024 palabras de 8 bits, es decir, con una capacidad de 1.024 *x* 8 = 8.192 bits = 8 kbits (1 kbit = 1.024 bits).

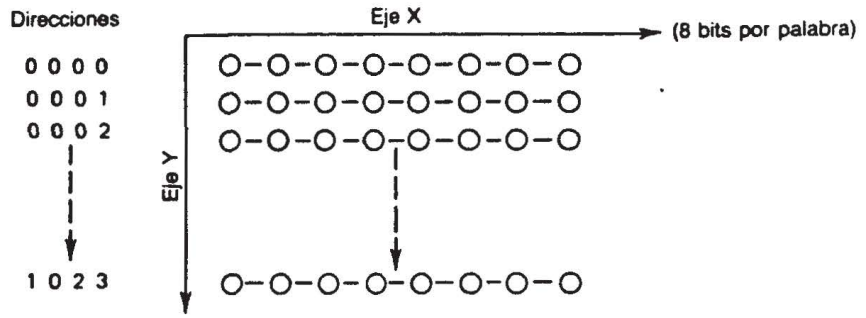


Figura 3.6

Matriz ROM bidimensional organizada en bytes, de 1.024 x 8

Si se desea trabajar con una longitud de palabra de 16 bits se suelen utilizar dos circuitos integrados de memoria ROM en paralelo, tal como muestra la **Figura 3.7**. En este caso la memoria sigue quedando organizada en bytes, porque el proceso de direccionamiento solamente activa una línea de las 1.024 posibles. De esta forma, a pesar de utilizar dos circuitos de memoria, desde el punto de vista de su organización, son como un único circuito de 1.024 palabras de 16 bits.

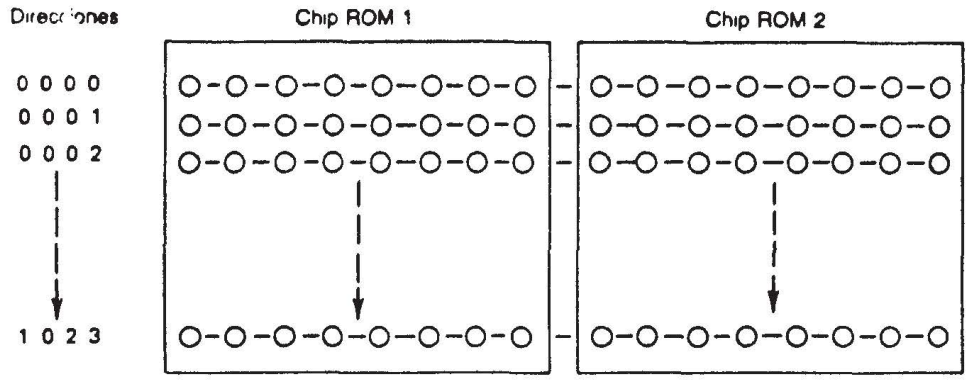


Figura 3.7
Memoria ROM bidimensional utilizando 2 circuitos

Para acceder a una ROM de 1.024 palabras, se necesita otra palabra que tenga capacidad de direccionar 1.024 (2^{10}) posiciones, es decir, de 10 bits, por lo que generalmente se utiliza como mínimo una palabra de 16 bits para el direccionamiento, lo que proporciona la posibilidad de acceder a 65.536 (2^{16}) posiciones de memoria, ó 64 kbytes. Aunque hasta hace unos años esto se lograba disponiendo varios circuitos en paralelo, lo que exigía la definición de bits para direccionar cada uno de los circuitos y bits para direccionar la palabra dentro del circuito, actualmente la capacidad de los circuitos de memoria ROM es mucho más elevada.

Así, hoy día un típico programa de arranque y configuración de un ordenador personal puede estar escrito en 2 circuitos ROM en paralelo, cada uno de 8 bits y con una capacidad total de hasta 256 kbits, lo que daría una capacidad de 32 kbytes (256/8) por circuito o una capacidad total de la memoria de 64 kbytes, acceso máximo que permite un bus de direcciones de 16

bits.

3.4.2 ORGANIZACION FISICA DE LA MEMORIA RAM

Como la memoria RAM es de lectura/escritura y ha de tener gran capacidad, se suele organizar utilizando un circuito integrado por bit, o configuración en tres dimensiones, lo que reduce considerablemente su coste, al poder prescindir de la electrónica externa que necesitaría si estuviese organizada por bytes.

Así por ejemplo, si se considera una longitud de palabra en la memoria de 8 bits, se utilizan 8 circuitos RAM que forman un denominado, *módulo*, *bloque* o *banco* RAM, tal como muestra el esquema de la **Figura 3.8.** En este caso, cada circuito de memoria contiene un solo bit de la palabra. Si un circuito puede almacenar 1.024 bits, todos los bits del circuito número 1 contendrán el bit de la posición 0 de las 1.024 palabras del módulo, el

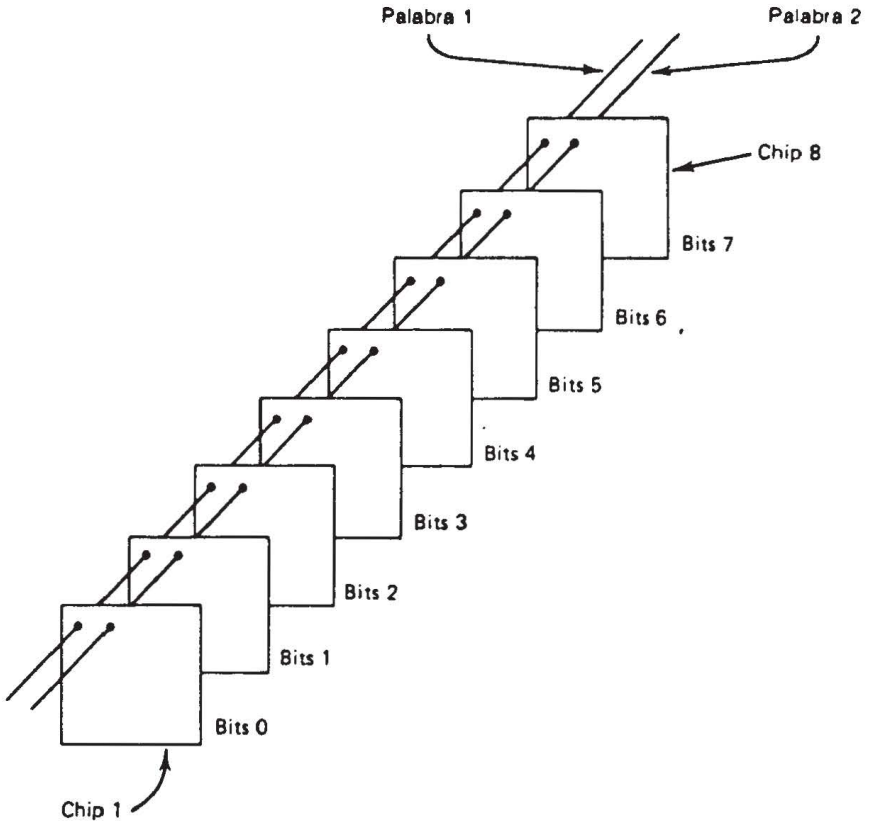


Figura 3.8

Módulo RAM organizado en bits, tridimensional, de 8 bits por palabra

circuito número 2 contendrá el bit de la posición 1 de las 1.024 palabras, etc.

De esta forma se puede construir una placa de memoria RAM combinando la capacidad de almacenamiento binario con el número adecuado de módulos, siempre y cuando el bus de direcciones sea capaz de direccionar el número de palabras resultante. Por ejemplo, con circuitos de 256 kbits dispuestos en 2 módulos se obtienen 512 kbytes de memoria (obsérvese que hay 8 circuitos en cada módulo, lo que transforma 1 bit en 1 byte). Esto requeriría, sin embargo, un bus de direcciones de, al menos, 19 bits (2^{19} = 524.288 bits = 512 kbits) para obtener las 512 k-direcciones que exigen las 512 k-palabras que se pueden almacenar en la memoria, lo que sólo es posible con un microprocesador de 16 bits, que tiene un bus de direcciones de 24 bits.

Ahora bien, para construir la dirección de memoria de una determinada palabra habrá que combinar la dirección del circuito de memoria en que se encuentra dicha palabra con la dirección de ésta dentro del mismo. Así, cada dirección de memoria se representa como un conjunto de *bits de selección de circuito* y de *bits de dirección de palabra.* Los bits de selección de circuito representan a uno o más circuitos que constituyen un módulo de memoria; los bits de dirección de palabra identifican a una palabra de memoria dentro del módulo direccionado.

Supóngase en estas condiciones que las palabras de una memoria se construyen mediante 8 circuitos separados de m *x* 1 bits. Si los bits de selección de circuito representan un módulo de memoria de 8 bits y los de dirección de palabra identifican la palabra dentro del módulo, la dirección de memoria se representará de la forma que se muestra en la **Figura 3.9.**

El número de bits de dirección de palabra considerado dependerá de la capacidad del circuito de memoria. Por ejemplo, un circuito de memoria que pueda contener 16.384 palabras de 8 bits (16 kbytes = 128 kbits) se ha de representar por una dirección de palabra de 14 bits (2^{14} = 16.384). Por su parte, el número de bits utilizados para la selección del circuito estará entonces en función de la arquitectura del microprocesador, de tal forma que si éste sólo permite un bus de direcciones de 16 bits, quedarían 2 bits libres para el direccionamiento del circuito. De esta forma, para el ejemplo considerado la memoria máxima constará de 4 módulos (2^2) de memoria, con

16.384 palabras por módulo, lo que daría un almacenamiento total de 16.384 *x* 4 = 65.536 bytes (64 kbytes).

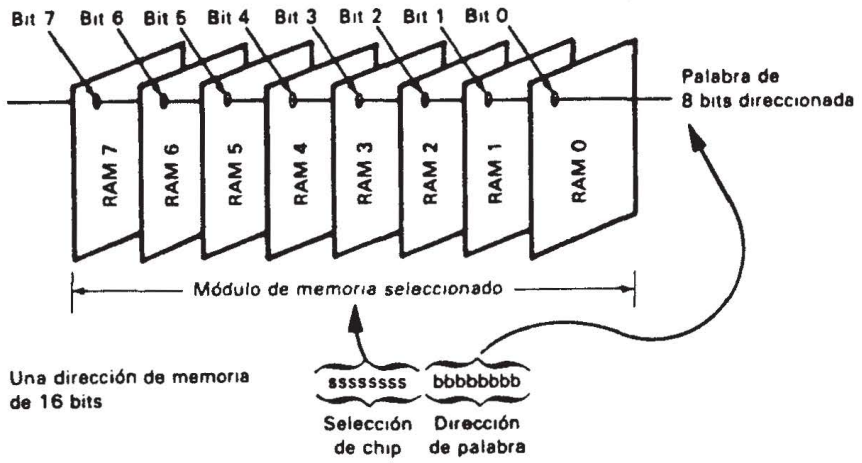


Figura 3.9

Dirección de memoria de 16 bits

Cabe indicar en este punto que 64 kbytes es, en principio, el límite físico de memoria que pueden direccionar los microprocesadores de 8 bits, cuyo bus de direcciones es de 16 bits (2^{16} = 65.536). Si el sistema tiene más memoria real de la que puede acomodar su rango de direcciones, se puede utilizar dicha memoria dividiéndola en "páginas", cada una de las cuáles sí tiene una correspondencia directa con la memoria real. Con un conjunto adicional de líneas de direccionamiento se selecciona una de las páginas cada vez. Este esquema de direccionamiento recibe el nombre de *paginación física* (*hardware paging*). Este esquema se raliza normalmente enviando la información de selección de página a través de un puerto de E/S a la memoria del sistema, donde se decodifica dicha información y se activan o desactivan los circuitos de memoria de acuerdo con ella.

Finalmente es necesario indicar que no hay ninguna razón por la que las direcciones de memoria disponibles tengan que ser continuas o incluso comenzar por cero, por lo que a efectos de programación se puede organizar la memoria de forma diferente a la estudiada hasta aquí, es decir, se puede acometer una organización lógica.

3.5 ORGANIZACION LOGICA DE LA MEMORIA RAM

La organización física de la memoria es la forma en que los otros dispositivos del sistema "ven" a la memoria. Sin embargo, para facilitar la programación de un sistema se suele organizar la memoria desde el punto de vista del programador, conociéndose este método como *organización lógica* de la memoria.

En el caso más sencillo, la organización lógica de la memoria coincide con la física. Esta es la situación más frecuente en los ordenadores de 8 bits y en el caso de las memorias ROM, debido a su propio diseño. Sin embargo, los ordenadores de 16 bits, con su mayor espacio de direccionamiento (bus de direcciones de 24 bits con posibilidad de direccionar 2^{24} = 16.384 kbytes) y sus aplicaciones multitarea, requieren esquemas de organización más elaborados como los denominados "segmentación" y "paginación".

La *segmentación* es un método de acceso a la memoria por el cual toda dirección física se compone de dos cantidades: un identificador de segmento y un desplazamiento.

* El *identificador de segmento* es un número que indica la posición inicial de un área general de memoria: el segmento.

* El *desplazamiento* es un número que indica la dirección de una palabra dentro de un segmento de memoria.

El método de la segmentación implica que la memoria física esté organizada de forma lineal, es decir, cada elemento de memoria tiene asignada una única dirección física, que se calcula a partir del identificador de segmento y del desplazamiento (gestión de memoria).

En los microprocesadores de 8 bits toda la memoria es un único y gran segmento, por lo que cada dirección de acceso a memoria corresponde a un desplazamiento. En los microprocesadores de 16 bits, el espacio de direccionamiento es tan grande, que la segmentación permite especificar distintos rangos de direccionamiento dentro de la memoria, es decir, se puede definir el comienzo y el final de un segmento. Todo ello hace que la segmentación sea un esquema muy flexible de organización de la memoria, que posee las siguientes características:

* El tamaño de los segmentos puede ser variable

* Los segmentos pueden comenzar y terminar en cualquier posición física

de la memoria

* Los segmentos se pueden superponer
* La segmentación obliga al programador a estar atento a los límites del segmento

La *paginación* es un método de acceso a la memoria por el cual toda dirección física se compone de dos cantidades: un identificador de página y un desplazamiento.

* El *identificador de página* es un número que indica la posición inicial de un área de memoria: la página lógica.

* El *desplazamiento* es un número que indica la dirección de una palabra dentro de una página de memoria.

Evidentemente, la paginación es un concepto completamente análogo a la segmentación, ya que cada acceso a una posición de memoria se hace a partir de un identificador de página, que relaciona la página deseada, y un desplazamiento, que localiza la posición del elemento de memoria en la página. Sin embargo, su interés especial radica en la aplicación de este concepto a una memoria físicamente organizada en páginas. En efecto, es este caso se utilizan las mismas páginas físicas que lógicas para evitar retardos innecesarios que se pueden originar al tener que cambiar de página física varias veces dentro de una misma página lógica.

Las características principales que diferencian a este método de la segmentación se pueden resumir en los siguientes puntos:

* Las páginas tienen un tamaño fijo
* Las páginas deben estar comprendidas entre unas fronteras de bloque rígidas
* Las páginas no se pueden superponer
* Los límites de la página vienen determinados automáticamente por la paginación física

Todo ello hace que la paginación sea un método mucho menos flexible de organización de la memoria que la segmentación, pero imprescindible cuando existe una paginación física de la memoria.

3.6 GESTION DE MEMORIA

La gestión de la memoria es el procedimiento que relaciona el esquema

lógico de direccionamiento de memoria y la organización física de ésta. Se encarga de calcular la dirección física real de memoria a partir de la dirección lógica definida en un programa, tanto en la segmentación como en la paginación.

La gestión de memoria en los microprocesadores de 8 bits se realiza, como ya se ha indicado anteriormente, por paginación. Por su parte, los microprocesadores de 16 bits incorporan un sistema de gestión de memoria por segmentación muy simple: se le suministra al sistema un nombre lógico para el segmento y otro para el desplazamiento, que son traducidos por la CPU a dos números de 16 bits, los identificadores correspondientes. El procesador multiplica el identificador de segmento por 16 y le suma el identificador de desplazamiento, obteniendo así la dirección física del elemento de memoria como una palabra de 20 bits, tal como muestra la **Figura 3.10.**

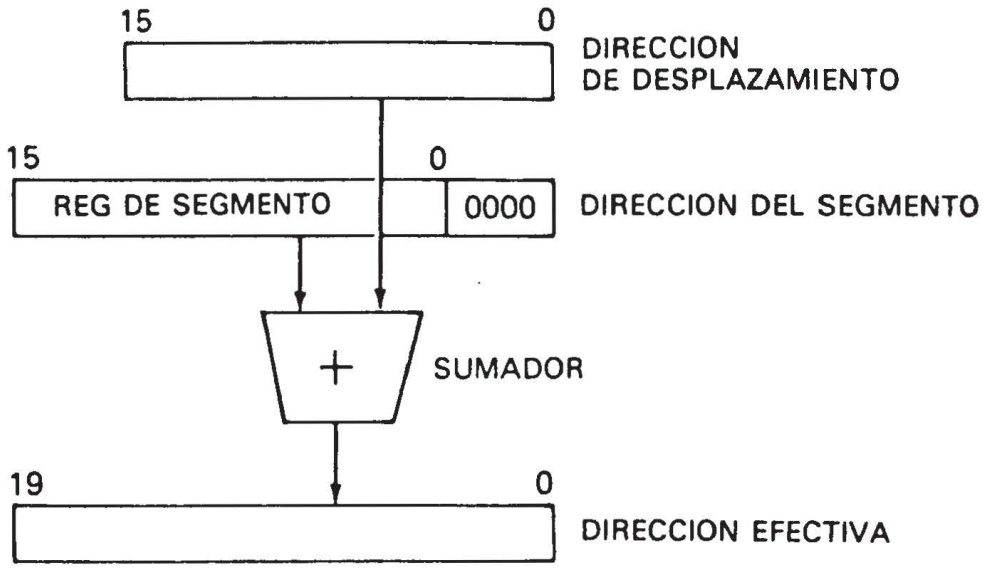


Figura 3.10
Gestión de memoria por segmentación

En realidad, el microprocesador no multiplica por 16 el identificador de segmento, sino que efectúa un desplazamiento de 4 bits a la izquierda, operación completamente equivalente, como ya se ha visto con anterioridad.

Así, con una dirección de 20 bits a su disposición, la CPU puede acceder a 1.048.576 (2^{20}) posiciones de memoria o palabras de 1 byte, por lo

que se suele decir que tiene una capacidad de direccionamiento de 1 Mbytes (1 Mbyte = 1.024 kbytes). Obsérvese no obstante que, como el identificador de segmento sólo puede representar direcciones que finalicen en 0000 (después de haber sido multiplicado por 16), los segmentos sólo pueden comenzar en cualquier elemento de memoria múltiplo de 16 (2^4) bits. Asimismo, como el identificador de desplazamiento está formado por 16 bits, la longitud máxima de un segmento será de 65.535 (2^{16}) bytes, ó 64 kbytes.

Cabe destacar finalmente que la gran ventaja del sistema de gestión de memoria es que el ordenador puede trabajar en multiproceso, es decir, ejecutando varios trabajos simultáneamente. Ello se logra asignando segmentos distintos a cada uno de los trabajos, de tal forma que éstos utilicen direcciones relativas (desplazamientos) como si no existiesen otros usuarios, sin preocuparse de qué segmentos de programa hay realmente en memoria. Este esquema se muestra en la **Figura 3.11** y necesita de una buena planificación de las tareas a realizar, asignándoles y actualizando prioridades, lo que aumenta considerablemente la eficacia del sistema.

3.7 LA SEGMENTACION EN LOS ORDENADORES PERSONALES

Los modernos ordenadores personales, dentro de los que destacan aquéllos basados en los microprocesadores de 16 bits Intel 80*i*86, requieren la definición de cuatro tipos de segmentos de memoria: uno para las instrucciones, dos para los datos y uno para cierto tipo de almacenamiento temporal.

El *segmento de código* (*Code Segment*) contiene el conjunto de instrucciones (programa) que el microprocesador está ejecutando en ese instante.

El *segmento de datos* (*Data Segment*) contiene los valores de las variables que utiliza el programa.

El *segmento adicional* (*Extra Segment*) es un segmento de datos de reserva, que se usa principalmente en operaciones en las que están involucradas cadenas de caracteres.

El *segmento de pila* (*Stack Segment*) guarda un tipo especial de estructura de datos, llamada *pila* (*stack*), que actúa como depósito temporal para datos y direcciones. Mientras se está ejecutando una subrutina de un

programa (parte del programa que se puede llamar desde un programa principal tantas veces como sea necesario), el microprocesador usa la pila para guardar una dirección de retorno, es decir, una marca que lleva al procesador al lugar original de donde partió desde dentro del programa. La pila también se puede utilizar para realizar otras funciones, como se verá más adelante.

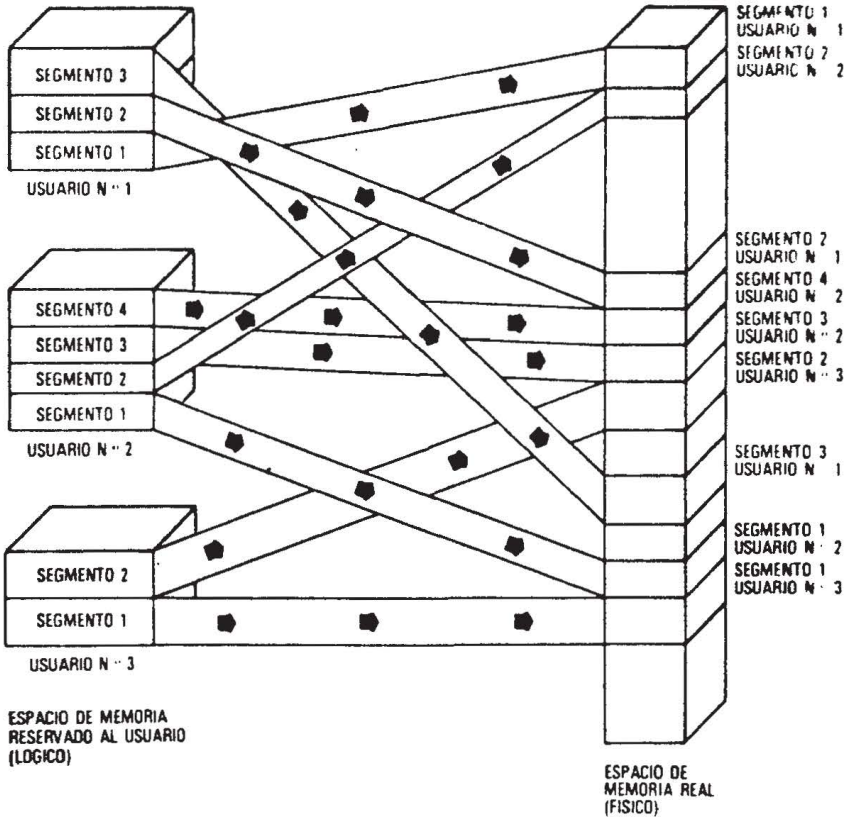


Figura 3.11

Gestión de memoria en multiproceso

El uso de estos diferentes segmentos significa que hay "áreas de trabajo" separadas para el programa, los datos y la pila. Como ya se ha indicado, cada segmento tiene un tamaño máximo de 64 kbytes (2^{16} direcciones de desplazamiento), por lo que el área de trabajo puede llegar hasta 256 kbytes (4 *x* 64) en un momento dado. Aunque no es necesario, es posible y, a veces, aconsejable, colocar los 4 segmentos en la misma área, como se verá al estudiar los programas .COM. En este caso , programa, datos y pila residirían en la misma área de trabajo de 64 kbytes.

El programador puede determinar la posición de estos segmentos, especificando el identificador de segmento de 16 bits con la dirección de segmento apropiada. Esto se realiza normalmente al inicio del programa, pero se puede hacer también fácilmente en cualquier momento mientras el programa se ejecuta, cambiando dinámicamente la dirección de estos segmentos.

BIBLIOGRAFIA

Boyce, J.C.; *"Microordenadores"*, Alhambra, Madrid (1986).

Khambata, A.J.; *"Microprocesadores/microcomputadoras. Arquitectura, software y sistemas"*, Gustavo Gili, Barcelona (1987).

Morgan, C.L. y Waite, M.; *"Introducción al microprocesador 8086/8088 (16 bits)"*, McGraw-Hill, México (1984).

Norton, P.; *"El IBM PC a fondo. Técnicas de programación avanzada"*, Anaya Multimedia, Madrid (1987).

Osborne, A.; *"Introducción a las microcomputadoras"*, Vol. 1, 2ª ed., Osborne/McGraw-Hill, Madrid (1986).

Scanlon, L.J.; *"80286. Programación ensamblador en entorno MS DOS"*, Anaya Multimedia, Madrid (1988).

CAPITULO 4

LA UNIDAD CENTRAL DE PROCESO

4.1 SISTEMAS FUNCIONALES DE UN MICROPROCESADOR

4.2 MEMORIA BORRADOR: REGISTROS, CONTADORES E INDICADORES

4.2.1 REGISTROS

4.2.1.1 REGISTROS GENERALES

4.2.1.2 REGISTROS DE SEGMENTOS

4.2.1.3 REGISTROS SE SEÑALIZACION Y DE INDICE

4.2.2 CONTADORES

4.2.3 INDICADORES

4.2.3.1 INDICADORES DE ESTADO

4.2.3.1 INDICADORES DE CONTROL

4.2.3.3 INDICADORES ESPECIALES

4.3 UNIDAD ARITMETICO-LOGICA

4.4 UNIDAD DE CONTROL Y TIEMPOS

4.5 LAS CONEXIONES EXTERNAS DE LA CPU: EL BUS

4.6 FLUJO DE INFORMACION EN LA CPU

4.1 SISTEMAS FUNCIONALES DE UN MICROPROCESADOR

Ya se ha visto anteriormente que los avances en la tecnología de semiconductores han permitido integrar en un solo circuito, denominado *microprocesador* (μP) los tres sistemas funcionales cuyo conjunto se conoce como CPU:

* Memoria borrador
* Unidad aritmético-lógica
* Unidad de control y tiempos

Estos tres bloques funcionales controlan todo el resto de las funciones realizadas por el ordenador: llaman las instrucciones de la memoria, las decodifican, las ejecutan e intercambian información con el resto del sistema. Al tratarse de la estructura básica de un ordenador, será necesario conocer, al menos, sus principios generales de funcionamiento, por lo que se describirán someramente cada uno de los bloques funcionales citados con objeto de mostrar los requerimientos que tiene cada uno de ellos para poder cumplir su tarea.

4.2 MEMORIA BORRADOR: REGISTROS, CONTADORES E INDICADORES

La *memoria borrador* (*draft memory*) es un grupo de posiciones de memoria internas de la CPU, que se utilizan para almacenar resultados parciales durante la ejecución de operaciones lógicas o aritméticas. Según la utilización que se haga de esta memoria de acceso rápido que, debido a ello, es muy limitada, estas posiciones de memoria se pueden tipificar como registros, contadores o indicadores.

A continuación se acomete el análisis de esta memoria especial. Se tratará de forma general su utilización, que siempre estará en función del diseño del μP. No obstante, también se incidirá en los aspectos concretos de la serie de μP más popular en la actualidad, Intel 80*i*86, con objeto de clarifiar los conceptos de utilización de esta memoria de acceso rápido por parte de un μP de 16 bits.

4.2.1 REGISTROS

Los registros son posiciones de memoria que se utilizan para el almacenamiento temporal de números binarios en la CPU. Los bits pueden ser

introducidos y extraidos de los registros en serie, es decir, un bit cada vez (rotaciones y desplazamientos a derecha o izquierda), o en paralelo, es decir, todos simultáneamente. En los sistemas digitales, los registros pueden ser de *propósito general,* cuando pueden utilizarse para varias funciones bajo control de un programa, o de *propósito específico,* cuando sólo pueden realizar ciertas funciones, ya que están cableados en la máquina y no pueden controlarse por programa.

De forma general, se pueden distinguir tres tipos de registros, según la utilización que le da el μP:

* Registros generales
* Registros de segmentos
* Registros de señalización y de índice

4.2.1.1 REGISTROS GENERALES

Los *registros generales* (*general registers*) están disponibles para ser usados por un programa, aunque algunas instrucciones específicas también los usan implícitamente. Estos registros son los siguientes:

* Registro acumulador (AX)
* Registro base (BX)
* Registro contador (CX)
* Registro de datos (DX)

En los μP Intel 80*i*86 estos registros se pueden tratar como de 16 bits o como de 8 bits, dependiendo de si se está operando con palabras de 2 bytes o de 1 byte, respectivamente. Los registros de 16 bits se llaman *AX, BX, CX* y *DX*. Dentro de estos registros "X" hay registros de 8 bits que se identifican sustituyendo "X" por "H" (*High*), para representar el byte más significativo, y registros de 8 bits que se identifican sustituyendo "X" por "L" (*Low*), para representar el byte menos significativo de los registros de 16 bits, tal como muestra la **Figura 4.1.**

El *registro acumulador* (*accumulator register*), *AX,* es el principal registro de trabajo del μP y a veces también es el puerto de E/S de la CPU. Usualmente almacena uno de los operandos que usa la ALU al realizar operaciones aritméticas o lógicas y el resultado de la operación.

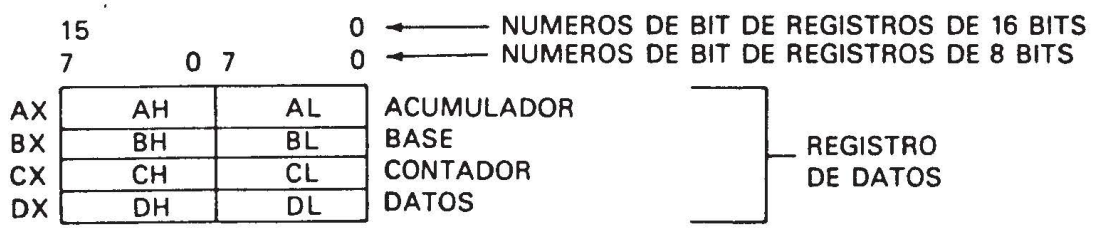


Figura 4.1

Registros generales

El *registro base* (*base register*), *BX*, es también de propósito general para realizar operaciones, pero se puede utilizar como registro de direcciones de memoria. En este caso mantiene la dirección base o comienzo de una tabla en la que las direcciones se determinan sumándole valores de desplazamiento.

El *registro contador* (*counter register*), *CX*, se usa a menudo como contador de repetición para las operaciones de bucles y como contador de elementos para las operaciones con cadenas. Su parte menos significativa, CL, se emplea para almacenar desplazamientos y rotaciones de múltiples bits.

El *registro de datos* (*data register*), *DX*, sirve como extensión del acumulador para las operaciones de multiplicación y división en múltiple precisión. También puede proporcionar el número de puerto en las operaciones de E/S.

4.2.1.2 REGISTROS DE SEGMENTOS

Los registros de segmentos contienen las direcciones de comienzo de los cuatro tipos de segmentos de memoria definidos anteriormente y se muestran en la **Figura 4.2**:

* Segmento de código (CS)
* Segmento de pila (SS)
* Segmento de datos (DS)
* Segmento adicional (ES)

El *registro de segmento de código* (*Code Segment*), *CS*, apunta al segmento de código, es decir, contiene la dirección de memoria inicial del segmento que guarda el programa que se está ejecutando en ese momento.

El *registro de segmento de pila* (*Stack Segment*), *SS*, apunta al segmento

de pila, es decir, contiene la dirección de memoria donde comienza el segmento que se está utilizando como pila de datos.

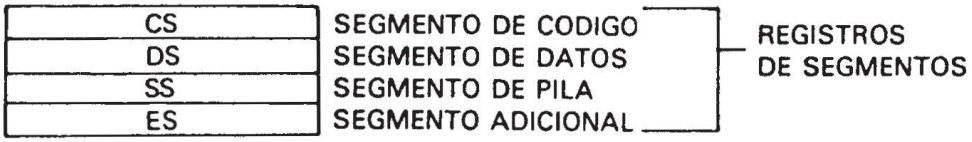


Figura 4.2
Registros de segmentos

El *registro de segmento de datos* (*Data Segment*), *DS*, apunta al segmento de datos, es decir, contiene la primera dirección de memoria del segmento que se utiliza para almacenar los valores de las variables del programa.

El *registro de segmento adicional* (*Extra Segment*), *ES*, apunta al segmento de datos adicional, conteniendo la dirección de memoria donde éste comienza. Cuando se necesita copiar información del segmento de datos al segmento adicional por falta de espacio (máximo 64 kbytes), es necesario utilizar ES junto con DS para poder identificar simultáneamente el segmento de destino y el segmento fuente de los datos para proceder a su traslado.

4.2.1.3 REGISTROS DE SEÑALIZACION Y DE INDICE

Los registros de señalización (punteros) y los registros de índice guardan los valores del desplazamiento empleados para el acceso a determinadas posiciones de memoria dentro de un segmento. Una de sus características más importantes es que pueden ser utilizados en operaciones aritméticas y lógicas, de modo que los valores de desplazamiento que almacenan pueden ser el resultado de operaciones previas. Los registros incluidos en este grupo son los siguientes (**Figura 4.3**):

* Puntero de pila (SP)
* Puntero base (BP)
* Indice fuente (SI)
* Indice destino (DI)

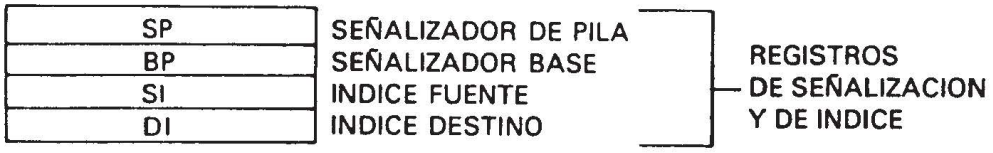


Figura 4.3

Registros de señalización y de índice

El *puntero de pila* (*Stack Pointer*), *SP*, es el registro que contiene el valor del desplazamiento de la última posición de memoria dentro del segmento de pila, es decir, la dirección del extremo de la pila. Así, la dirección de memoria efectiva del último dato almacenado en la pila viene dado por la pareja de registros SS:SP, como se verá más adelante con mayor profundidad al estudiar las operaciones con la pila.

El *puntero base* (*Base Pointer*), *BP*, es un registro de direccionamiento que contiene el valor del desplazamiento que se va a utilizar como punto de partida en el direccionamiento de datos. Aunque se puede combinar con cualquier segmento de datos (DS, ES y SS), normalmente se usa en el segmento de pila. De esta forma, se puede establecer dentro de ella una dirección de referencia mediante la pareja de registros SS:BP y manipular los datos contenidos en las posiciones de memoria contiguas mediante las direcciones SS:[BP ± n], siendo n el número de posiciones de memoria.

El *índice fuente* (*Source Index*), *SI*, es un registro de direccionamiento que contiene el valor del desplazamiento relativo a una posición de memoria definida previamente, bien por un comienzo de segmento o bien por una dirección base dentro de un segmento. Este registro índice, combinado con otros registros, permite realizar diversos modos de direccionamiento, como se estudiará más adelante, pero su principal utilidad es el acceso a matrices de datos.

El *índice destino* (*Destination Index*), *DI*, se define de forma idéntica al índice fuente, siendo su principal misión la que deriva de su posibilidad de utilizarlo con éste en operaciones de traslado de datos de un lugar a otro de la memoria. Así, la dirección de los datos que se desea trasladar se determina con ayuda de SI, mientras que con ayuda de DI se hace referencia a la nueva zona de memoria donde se desea almacenar los mismos datos.

Cabe resaltar finalmente que los tres últimos registros estudiados, BP, SI, DI, junto con el registro base, BX, son los denominados *registros de direccionamiento de memoria* (*Memory Addressing Registers*). Su uso combinado permite acceder de diferentes formas a cada posición de memoria y la importancia de estas operaciones hacen recomendable su estudio más detallado, que se abordará más adelante.

4.2.2 CONTADORES

Los *contadores* (*counters*) son posiciones de memoria que registran y mantienen la cuenta de un número de bits. Un contador puede operar en sentido positivo, igualando todos sus bits a 0 e incrementando su valor cada vez que le llega un bit, o en sentido negativo, igualando todos sus bits a 1 y decrementando su valor cada vez que le llega un bit. La lectura de los contadores se hace siempre en paralelo, es decir, todos sus bits a la vez.

El contador más importante es el *contador de programa* (*program counter*), que en los µP de la serie 80*i*86 se denomina *puntero de instrucción* (*Instruction Pointer*), *IP*. Este contador contiene la dirección de memoria, dentro del segmento de código en curso, de la instrucción que el programa va a ejecutar. Como el registro CS contiene la dirección de memoria donde comienza el segmento de código, IP representará un desplazamiento dentro de dicho segmento, con lo cual la lectura de la instrucción se lleva a cabo en la dirección efectiva indicada por la pareja registro:contador CS:IP.

4.2.3 INDICADORES

En todos los ordenadores digitales, durante algunas operaciones aritméticas o lógicas se llega a ciertas condiciones que requieren un tratamiento especial. Es importante conocer estas condiciones y mantener una indicación temporal de las mismas, ya que las siguientes operaciones pueden depender de estas condiciones. Ello se logra con los denominados *indicadores*, *registros de estado* o *"banderas"* (*flags*), almacenados en un registro de la CPU en el que cada bit refleja el estado de ciertos dispositivos lógicos del µP.

En los µP 80*i*86 el registro de indicadores está formado por 16 bits de uso individual, tal como muestra la **Figura 4.4.**

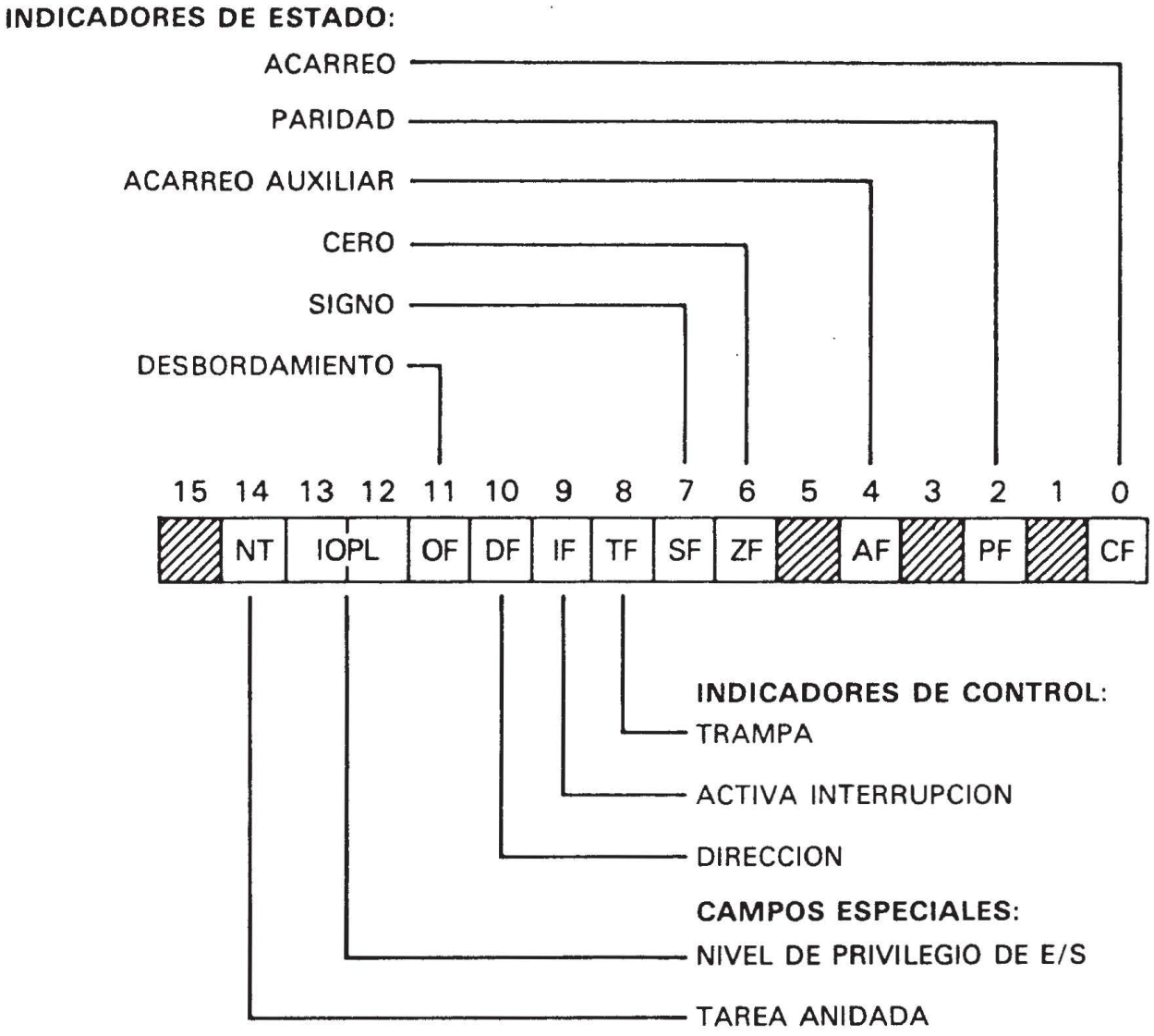


Figura 4.4
Registro de indicadores

Como se puede apreciar en la **Figura 4.4**, no todos los bits están definidos, por lo que sirven de reserva para usos posteriores en los μP más avanzados. Por otro lado, los indicadores se pueden dividir en tres grupos:

* Indicadores de estado
* Indicadores de control
* Indicadores especiales

4.2.3.1 INDICADORES DE ESTADO

Los indicadores de estado, contenidos en el byte menos significativo y en el bit 11 del registro de indicadores, se activan a través de las operaciones aritméticas o lógicas que se llevan a cabo en el procesador.

El *indicador de acarreo* (Carry Flag), CF, es el enlace del μP con la aritmética de múltiple precisión. Cuando una suma produce un acarreo o cuando una resta produce un acarreo negativo, su valor es 1; en caso contrario es 0. También actúa como indicador del resultado de multiplicaciones en combinación con el indicador de rebose. Asimismo CF puede contener el valor de un bit que ha sido desplazado o rotado fuera de una palabra o reflejar el resultado de una operación de comparación.

El *indicador de paridad* (*Parity Flag*), *PF*, es un bit que permite comprobar el valor de otros bits. Su valor es puesto a 1 si el resultado de una operación tiene un número par de bits 1 y es puesto a 0 si el resultado tiene un número impar de bits 1. Este indicador tiene poco uso en las operaciones aritméticas y lógicas, pero es muy importante en la transmisión de datos.

El *indicador auxiliar de acarreo* (*Auxiliary carry Flag*), *AF*, es completamente similar al CF, pero sólo se utiliza para señalar un acarreo por exceso o por defecto en los 4 LSB del resultado de operaciones en el sistema BCD.

El *indicador de cero* (*Zero Flag*), *ZF*, indica si el resultado de la última operación es cero o no lo es. El indicador se pone a 1 si el resultado es cero, mientras que un resultado distinto de cero pone el indicador a 0.

El *indicador de signo* (*Sign Flag*), *SF*, indica si el resultado de la última operación aritmética o lógica es positiva o negativa. Si una operación produce un resultado negativo, el indicador es puesto a 1; en caso contrario es 0.

El *indicador de rebose* (*Overflow Flag*), *OF*, es el único indicador del byte más significativo que es accionado por operaciones aritméticas normales, y sirve para detectar un error durante las operaciones de números con signo. Este indicador es puesto a 1 si la operación entre dos números produce un resultado que el acumulador no puede almacenar; también se pone a 1 si el MSB (el signo) del operando ha cambiado durante una operación de desplazamiento

aritmético. En los casos contrarios es 0. En combinación con CF indica la longitud del resultado de una multiplicación: si la mitad superior del producto es un número distinto de cero, OF y CF son 1; en caso contrario, ambos indicadores son 0.

4.2.3.2 INDICADORES DE CONTROL

Los indicadores de control están contenidos en los bits 8 a 10 y registran el modo de funcionamiento del procesador, influyendo en la forma en que se ejecuta un programa.

El *indicador de intercepción* o *"trampa"* (*Trap Flag*), *TP*, permite al μP seguir paso a paso los programas, cuando se procede a su depuración (*debug*) o corrección. Cuando este indicador es puesto a 1 se produce una interrupción después de cada instrucción; para que el programa se ejecute normalmente el indicador debe ser puesto a 0.

El *indicador de interrupción* (*Interrupt Flag*), *IF*, permite al μP reconocer solicitudes de interrupción de dispositivos externos al sistema. Cuando no se desea permitir una interrupción se debe poner el indicador a 0; cuando un programa pone IF a 1, los dispositivos externos pueden generar interrupciones.

El *indicador de dirección* (*Direction Flag*), *DF*, se utiliza para indicar el sentido del movimiento de los registros de índice SI y DI. Cuando DF se pone a 0, las operaciones con bloques de datos incrementan los registros de índice; cuando DF se pone a 1, los registros de índice son decrementados.

4.3.3.3 INDICADORES ESPECIALES

Los indicadores especiales ocupados por los bits 12 a 14 son utilizados por el procesador Intel 80287 cuando opera en el denominado "modo protegido". Los bits 12 y 13 representan conjuntamente el *nivel de privilegio de E/S* (*Input/Output Privilege Level*) y el bit 14 indica una *tarea anidada* (*Nested Task*). Debido a la especificidad de estos indicadores y a su uso poco generalizado hasta el momento, no se procede aquí a su estudio.

4.3 UNIDAD ARITMETICO-LOGICA

Las manipulaciones efectivas de datos en el interior de la CPU se llevan a cabo en una sección denominada globalmente *unidad aritmético-lógica (Arithmetic-Logic Unit)*, dispositivo combinacional que permite realizar operaciones aritméticas (suma, resta y, actualmente, multiplicación y división), de desplazamiento, rotación y complemento (NOT). La **Figura 4.5** ilustra funcionalmente algunos de los contenidos de la ALU.

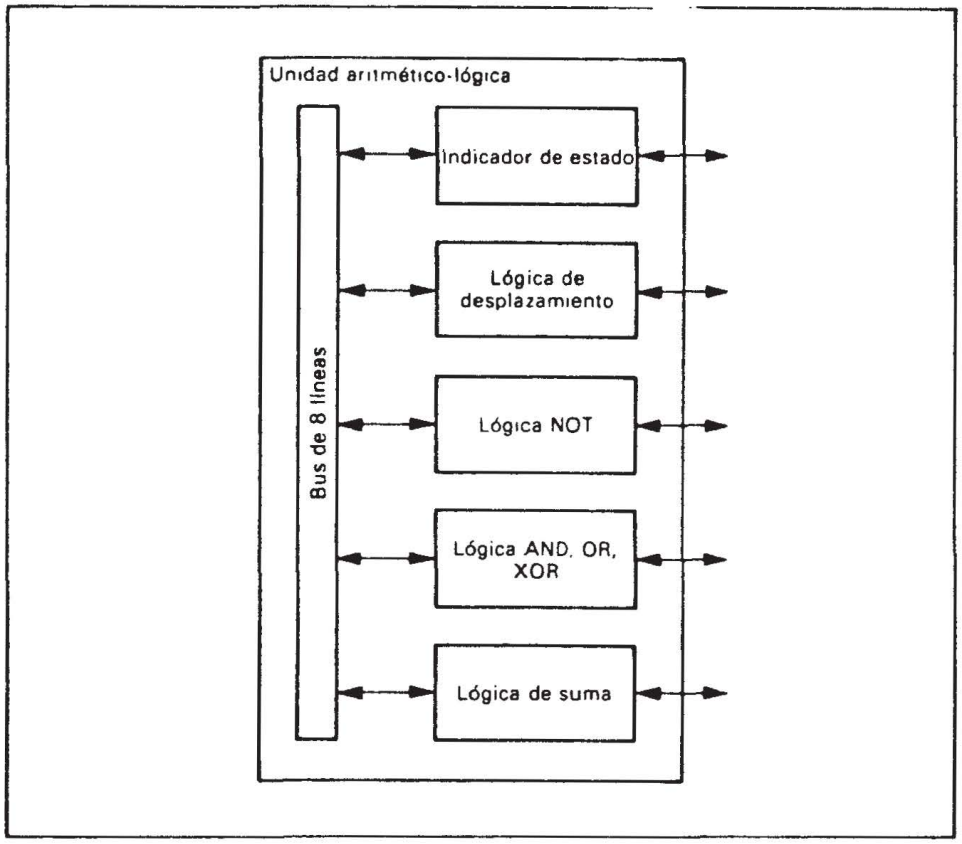


Figura 4.5
La unidad aritmético-lógica

Para que ocurran las citadas manipulaciones de datos, éstos deben transferirse desde los registros de la CPU a la ALU a través de los buses apropiados, y los resultados han de efectuar el recorrido inverso. Además, será necesario que la unidad de control actúe sobre la ALU para determinar qué parte de ésta se va a utilizar y cómo se va a hacer. Ello hace que exista una íntima relación de la ALU con los demás elementos de la CPU, tal como

muestra el esquema de la **Figura 4.6.**

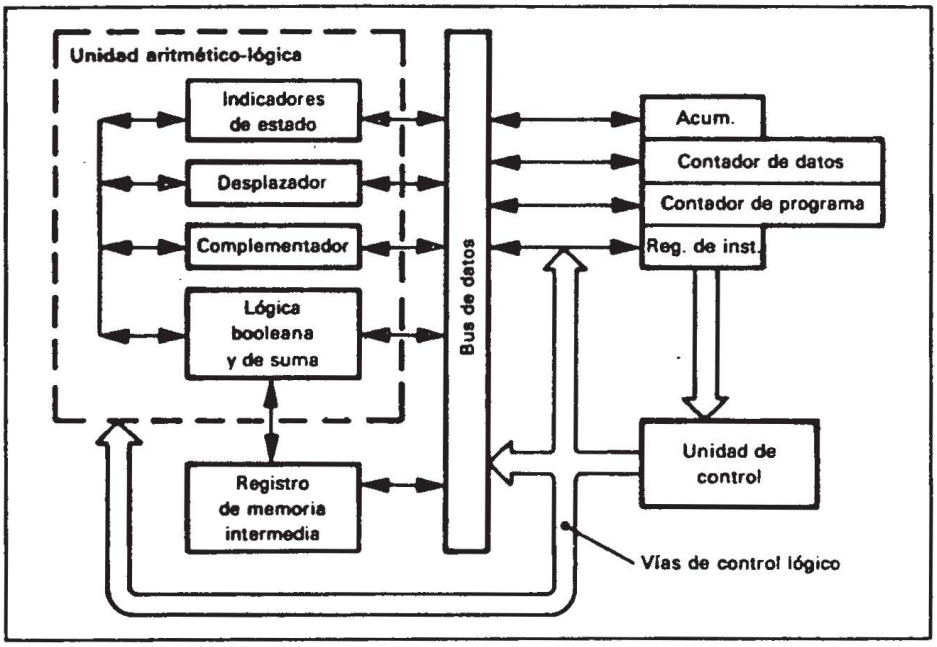


Figura 4.6
Relación de la ALU con el resto de la CPU

Los datos con los que trabaja la ALU están almacenados en la memoria, y son el resultado de pasos de programa que requieren operaciones aritméticas y lógicas. El primer paso en el proceso es la obtención de una palabra de ordenador que contenga el código que define la operación a ejecutar y la dirección de la información con la que se va a operar. La decodificación de la operación por parte de la unidad de control produce una señal enviada por aquélla a la ALU, que activa consecuentemente las unidades funcionales necesarias. A continuación se extraen los datos de la dirección de memoria contenida en la palabra de ordenador y se entrega también a la ALU. Se ejecuta la operación y luego la unidad de control obtiene la palabra siguiente, que en última instancia dará lugar a la extracción del siguiente paquete de datos con el que se va a operar. Este ciclo continúa bajo el control del programa hasta que se requiera otro tipo de operación.

4.4 UNIDAD DE CONTROL Y TIEMPOS

La CPU debe asegurar la ejecución adecuada de una instrucción específica, es decir, debe mantener una secuencia de sucesos y acometer las acciones necesarias para llevar a cabo cada operación. Dependiendo, pues, de la instrucción en proceso, ciertos circuitos suministran las señales apropiadas para las unidades internas y externas de la CPU que permiten iniciar las acciones de procesado. Estos circuitos se agrupan funcionalmente en la denominada *unidad de control* (*Control Unit*), que es el sistema encargado de la sincronización, decodificación y activación de operaciones para todas las demás partes del ordenador.

El funcionamiento de la CU está basado en la recepción de una señal de reloj generada a partir de la frecuencia suministrada por un oscilador de cuarzo externo a la CPU. EL *reloj* (*clock*) es un generador de pulsos que necesita un oscilador de cristal de cuarzo como fuente de frecuencia y, utilizando un oscilador u otro, la frecuencia obtenida puede variar dentro de un amplio rango. Asimismo, cada frecuencia implica un período, inverso de la frecuencia, que es el tiempo que transcurre entre dos formas de onda de reloj idénticas. El período equivalente a una frecuencia determinada se suele denominar *ciclo de reloj* (*clock cycle*) y se utiliza como unidad básica para medir el tiempo de ejecución de las instrucciones. Por su parte, se denomina *señal de reloj* (*clock signal*) a la onda cuadrada que éste genera, caracterizada por tener dos niveles de tensión perfectamente definidos, cada uno de los cuáles representa los estados lógicos 0 y 1. La **Figura 4.7** muestra una típica señal de reloj, donde se ha definido su período correspondiente.

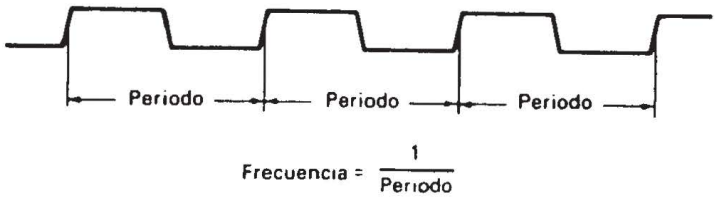


Figura 4.7
Señal de reloj

Obsérvese que dentro de un mismo ciclo, los dos niveles de tensión pueden tener duraciones diferentes. Aunque lo normal hasta ahora ha sido mantener ambos niveles el 50 % del período, en los modernos μP se mantiene la tensión alta sólo durante el 33 % del ciclo, para conseguir un rendimiento óptimo del procesador. Ello se consigue fácilmente utilizando un oscilador de triple frecuencia que la de salida de reloj.

Por lo indicado hasta ahora se puede deducir que un reloj que genera una señal de frecuencia de 4,77 MHz (1 MHz = 10^6 ciclos/s) proporciona un ciclo de reloj de 210 ns (1 ns = 10^{-9} s), mientras que si el reloj genera una frecuencia de 13 MHz se obtiene un ciclo de reloj de 77 ns.

El *generador de sincronismo* (*syncronism generator*) es la unidad que genera señales de una cierta duración y frecuencia, utilizadas para secuenciar en el tiempo las diferentes acciones que debe ejecutar la CPU. Estas señales se obtienen a partir de la señal básica proporcionada por el reloj y, evidentemente, su período es un múltiplo del ciclo de reloj. Las señales de sincronismo se aplican a diversas secciones funcionales del μP y determinan la velocidad básica de funcionamiento del mismo. La misión principal del generador de sincronismo es activar el decodificacdor de instrucciones y el generador de control, cuya combinación permite desarrollar las diversas señales de control que utiliza el μP. Como ejemplo de su funcionamiento se muestra el esquema de la **Figura 4.8.**

Por su parte, las funciones de una típica unidad de control y tiempos quedan esquemáticamente representadas en la **Figura 4.9**, y se pueden resumir en los siguientes puntos:

* Decodificación de una instrucción procedente de la memoria
* Determinación de los ciclos necesarios para ejecutarla
* Activación de las señales de control requeridas para cada ciclo
* Ejecución de los distintos ciclos en una secuencia ordenada
* Toma de decisisones en función de los estados de las distintas señales internas y externas.
* Direccionamiento de cada uno de los dispositivos de E/S, cuando sea necesario

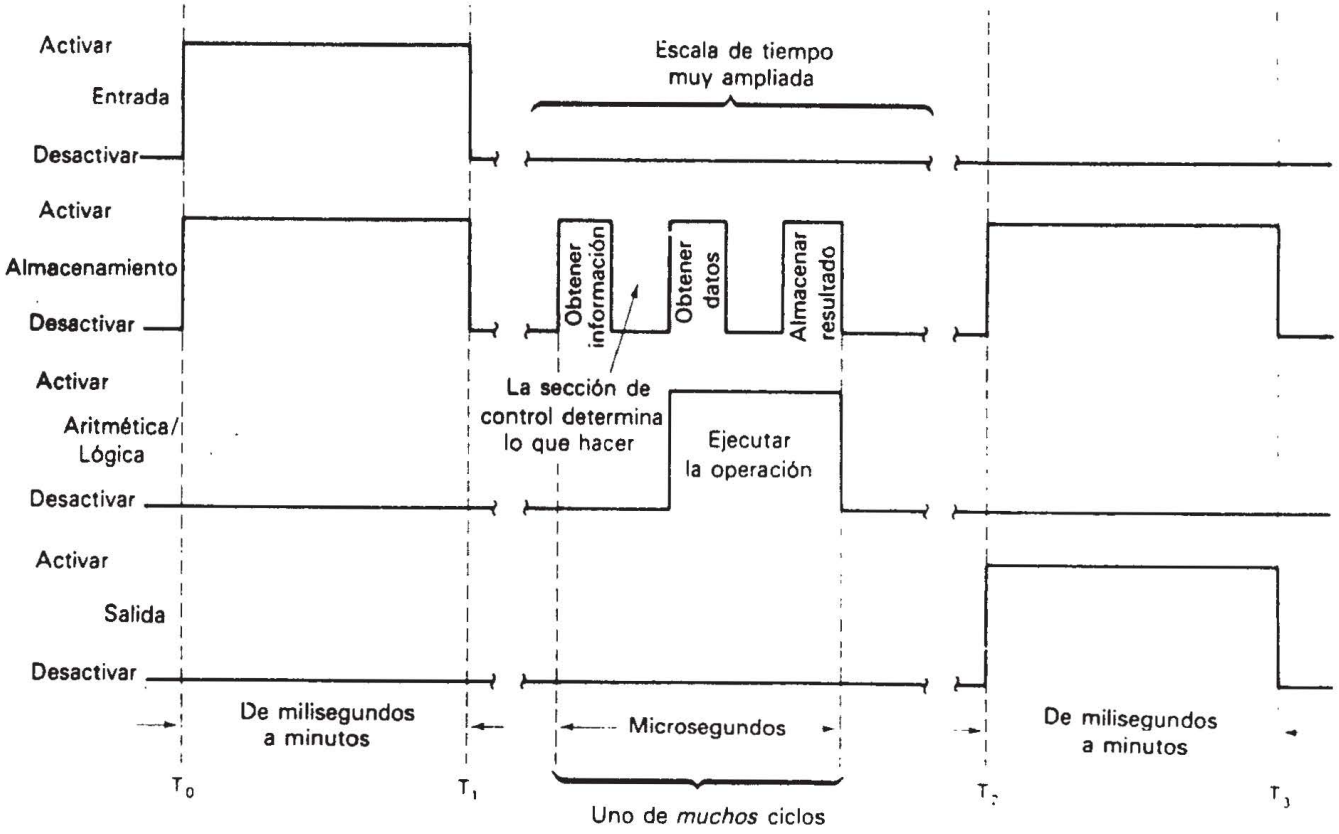


Figura 4.8

Funcionamiento del generador de sincronismo

Como ya se ha indicado, el ordenador sólo hace lo que se le ordena mediante una instrucción, cada una de las cuáles desencadena una serie de actividades en la CPU, que pueden sintetizarse en las siguientes fases:

* Envío de una dirección a la memoria de programa para buscar una instrucción
* Búsqueda de la instrucción y conducción de la misma a la CPU
* Ejecución de la instrucción buscada

En algunos μP las dos primeras fases están combinadas en un "subciclo de búsqueda", tal como muestra la **Figura 4.10.**

Todas estas acciones requieren un cierto tiempo, varios ciclos de reloj. Este tiempo se denomina *ciclo de máquina* (*machine cycle*) y se define como el ciclo de operación básico de la CPU que indica el tiempo necesario para realizar la búsqueda, decodificación y ejecución de una única instrucción. De esta manera, todas las instrucciones se ejecutan con un número fijo de ciclos de máquina que, a su vez, son múltiplos de los ciclos

de reloj. Por lo tanto, la velocidad con la cual el μP ejecuta un programa variará linealmente con la frecuencia de la señal de reloj.

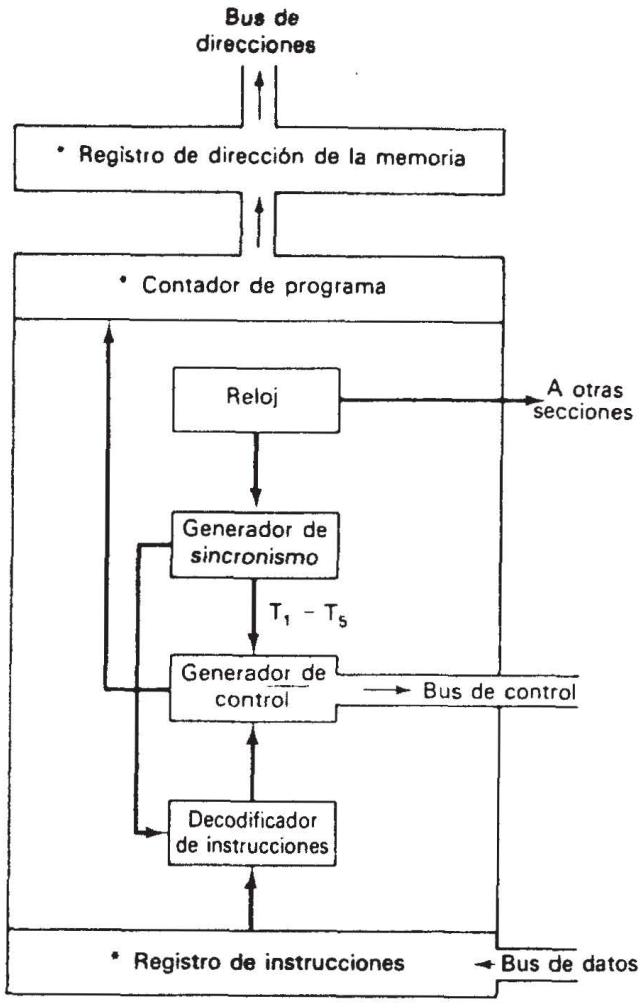


Figura 4.9

Unidad de control y tiempos

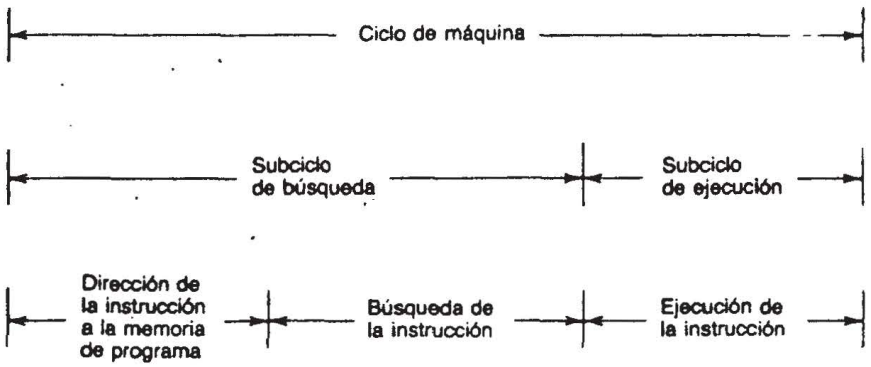


Figura 4.10

Fases de un ciclo de máquina

Sin embargo, no todos los microprocesadores utilizan un intervalo de tiempo fijo para el ciclo de máquina, sino que éste varía según cada instrucción particular. En este caso el ciclo de máquina se conoce como *ciclo de instrucción (instruction cycle)*, cada uno de los cuáles se lleva a cabo en un número variable de ciclos de reloj. Esto es lo que sucede en los modernos μP, en los que el tiempo de ejecución de un programa no sólo será función de la frecuencia de la señal de reloj y de su número de instrucciones, sino que además dependerá de las características de éstas.

4.5 LAS CONEXIONES EXTERNAS DE LA CPU: EL BUS

La CPU está conectada al resto de las unidades funcionales del ordenador, el sistema de memoria y la unidad de intercambios, por medio de grupos de líneas que transmiten señales eléctricas y constituyen el medio de comunicación entre estas unidades, llamadas *buses.* Cada línea es capaz de transmitir un pulso eléctrico, que representa un dígito binario, en cada instante, pero la misma línea puede transmitir una serie de dígitos binarios en forma de secuencia temporizada. Una línea única, como la descrita, puede usarse para la transmisión en *serie* de bits. Si hay que transmitir varios bits simultáneamente en *paralelo*, se debe disponer de una línea para cada bait.

Casi todos los μP y dispositivos afines se comunican utilizando niveles de señal que se dice que son compatibles con los *niveles TTL* (*Transistor-Transistor Logic*).

Se dice que una *señal de salida* sigue la norma TTL cuando para representar un 1 suministra al menos 400 μA de corriente sobre un rango de tensión de 2,4 a 5 V y para representar un 0 es capaz de extraer un mínimo de 16 mA de corriente sobre un rango de tensión de 0 a 0,4 V.

Se dice que una *señal de entrada* sigue la norma TTL (carga normalizada) cuando interpreta un 1 si la tensión de entrada está entre 2,0 y 5 V y no extrae más de 40 μA de corriente en este rango, e interpreta un 0 si la tensión de entrada está entre 0 y 0,8 V y no suministra más de 1,6 mA de corriente sobre este rango.

Las entradas y salidas de la mayoría de los μP y dispositivos afines

responden apropiadamente a las tensiones TTL, pero no suministran o extraen tanta corriente como las señales TTL normalizadas. Por ello estas entradas y salidas se describen como "compatibles con los niveles TTL" (de tensión), aunque no sean totalmente compatibles con TTL (en intensidad).

Los buses pueden ser *unidireccionales* (la información fluye en un único sentido) o *bidireccionales* (los bits pueden fluir en ambos sentidos). Aunque las líneas en sí mismas son siempre capaces de transmitir información en ambos sentidos, los circuitos conectados en cualquier punto de las mismas pueden ser capaces de hacer transferencias unidireccionales (entrada o salida) o bidireccionales (entrada y salida). La mayoría de los μP disponen de ambos tipos de buses.

Bajo el punto de vista funcional, el bus principal de un ordenador (bus del sistema) está compuesto por cuatro tipos de buses: alimentación, control, direcciones y datos, tal como muestra el esquema de la **Figura 4.11**, donde se ha omitido el bus de alimentación.

El *bus de alimentación* (*power bus*) hace llegar la corriente proveniente de la fuente de alimentación a los distintos componentes del ordenador. Si los circuitos son compatibles TTL, sólo se necesita una línea de +5 V y una de tierra. A veces se envía una señal de tensión más alta con una ligera ondulación (60 - 120 Hz) y cada placa rectifica y reduce la tensión convenientemente. De eta forma se consigue reducir el ruido, disminuir las caídas de tensión a lo largo de la línea y rebajar las señales parásitas.

El *bus de control* (*control bus*) es unidireccional y transporta la información que describe la operación a ejecutar. Su tamaño (número de líneas) está determinado por las señales de control requeridas por el μP utilizado y las señales que transporta corresponden a la sincronización del sistema, lectura/escritura en la memoria y lectura/escritura en la unidad de intercambios (dispositivos de E/S).

El *bus de direcciones* (*direction bus*) es unidireccional y manipula los bits que representan la situación del siguiente dato a utilizar o la posición donde deben almacenarse los datos, aunque muchos μP lo utilizan también para identificar los dispositivos de E/S que han de ser usados. Así, aunque la información circula en un solo sentido, el bus de direcciones cumple un doble cometido, definido en cada caso por la unidad de control, que proporciona una

señal para designar la realización de una u otra de las funciones descritas.

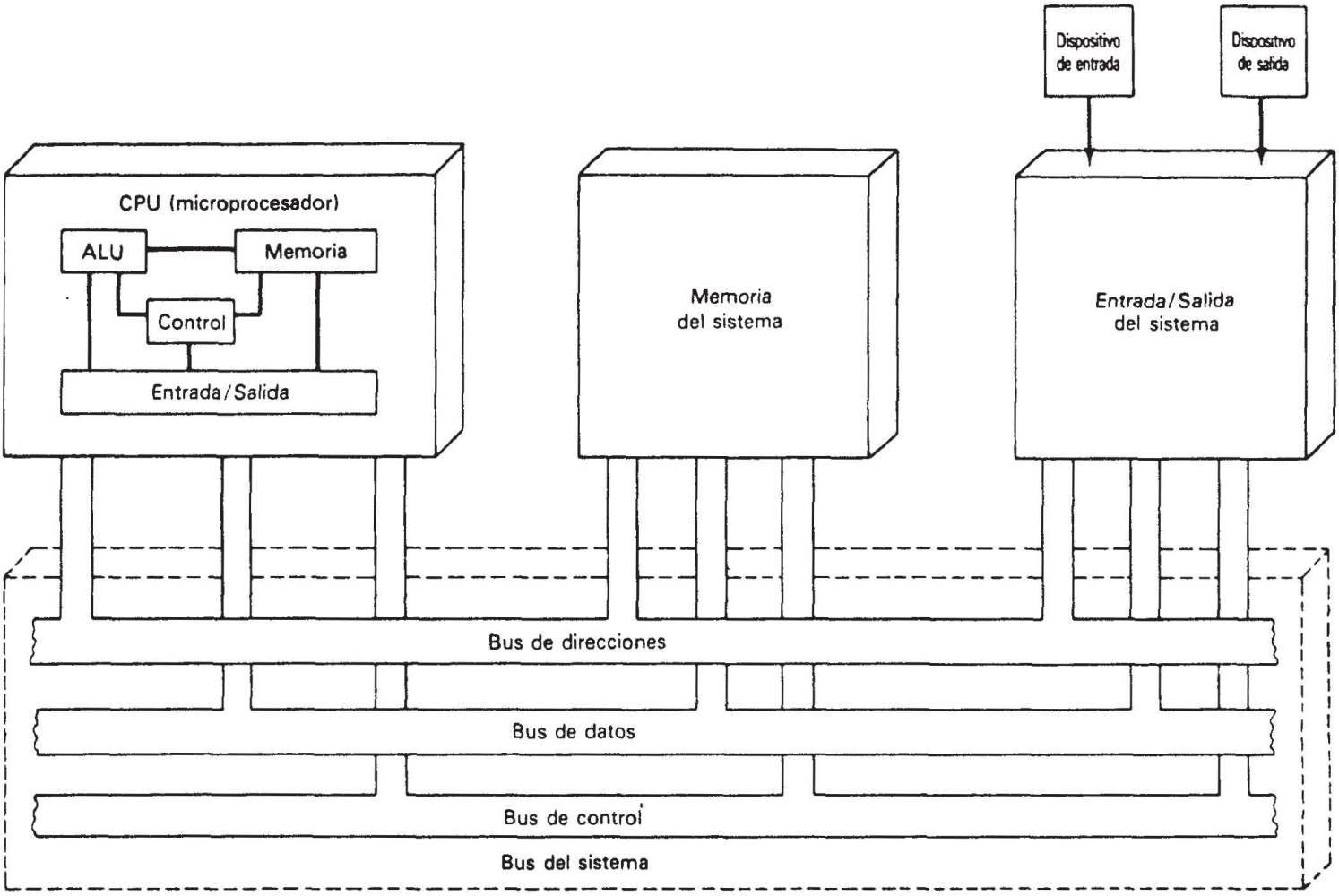


Figura 4.11
Estructura de buses de un ordenador

El *bus de datos* (*data bus*) transporta instrucciones para decodificar o información para procesar (datos). Es bidireccional, es decir, puede transmitir información del μP a otras unidades del sistema en un momento determinado, mientras que en otro momento puede llevar información de alguna unidad al μP. El sentido del flujo de datos viene determinado por la unidad de control y su número de líneas está establecido por el tamaño de la palabra que puede manipular el μP. Así, una máquina de 8 bits tiene 8 líneas de conexión en el bus de datos, mientras que una de 16 bits tiene este número de líneas.

Todos los buses llegan también a los distintos bloques funcionales de la CPU: memoria borrador, ALU y unidad de control y tiempos. Estos están conectados, directa o indirectamente a cada uno de los buses, de modo que las señales del bus, aparte de controlar todos los sistemas del ordenador, permiten el acceso a ellos de las operaciones funcionales de cada unidad de la CPU.

Cabe destacar también, independientemente de lo indicado hasta aquí, que los modernos μP utilizan los mismos terminales para las direcciones que para los datos: en las máquinas de 8 bits, el bus de datos (8 bits) comparte terminales con los 8 LSB del bus de direcciones; en las máquinas de 16 bits, el bus de datos (16 bits) comparte terminales con los 16 LSB del bus de direcciones. Ello es posible gracias al concepto de *multiplexado en el tiempo*, según el cual no todas las señales deben enviarse simultáneamente, lo que permite utilizar un mismo conjunto de líneas para enviar conjuntos distintos de señales en instantes de tiempo diferentes.

Así, durante un acceso a memoria, la dirección se envía en el primer ciclo de reloj y los datos en ciclos de reloj posteriores. Para que los distintos dispositivos puedan reconocer estos dos tipos de señal existe una señal específica que indica cuándo la información enviada es una dirección. Gracias a ella, parte de los circuitos externos de la CPU pueden trabajar con las direcciones e ignorar los datos, mientras que otra parte de los mismos puede ignorar las direcciones y guardar información sobre los datos.

Es necesario indicar finalmente que es fundamental evitar interferencias entre los diferentes sistemas del ordenador, que se podrían producir al estar todos ellos conectados al mismo bus. Este problema se resuelve utilizando puertas lógicas junto con una precisa sincronización de las operaciones que se realizan, según el esquema que se muestra a modo de ejemplo en la **Figura 4.12.**

Cada sistema del ordenador está conectado a todas las líneas del bus a través de una red de conmutación, formada por un gran número de puertas AND. Una entrada de cada puerta se conecta a una línea del bus, otra se conecta a la fuente de sincronización del μP y la tercera se conecta a la salida de la unidad de control de la CPU. La información de la línea del bus se transfiere a un sistema del ordenador solamente cuando está activa su puerta AND y en el

momento correcto determinado por la sincronización.

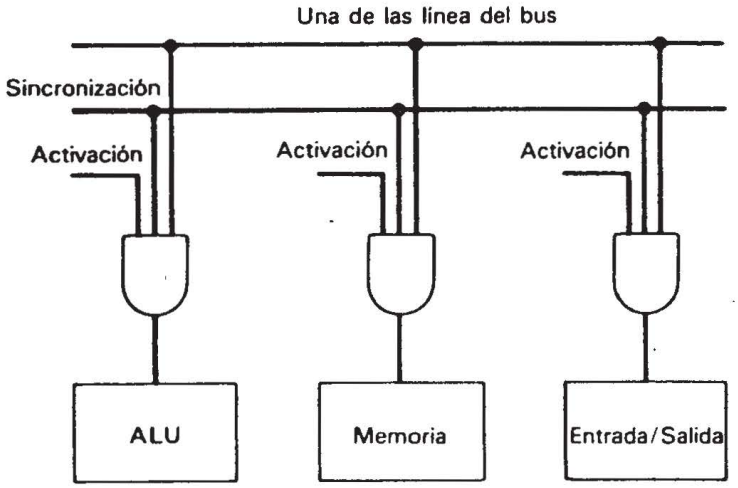


Figura 4.12
Conmutación del bus

4.6 FLUJO DE INFORMACION EN LA CPU

Anteriormente se han estudiado las distintas actividades que tienen lugar en la CPU durante un ciclo de máquina y en un ciclo de instrucción. No obstante, también es importante conocer cómo fluyen a través de la CPU las palabras de instrucción y de datos, mientras realizan sus respectivas funciones y procesos.

Una *palabra de instrucción* (*instruction word*) se procesa de la forma esquemática indicada en la **Figura 4.13**. Al inicio del ciclo de máquina el contenido del contador de programa que contiene la dirección de la próxima instrucción a buscar se transfiere a lo que genéricamente se puede denominar *registro de dirección de memoria* (**Memory Address Register**), por ejemplo, BP, SI o DI. Esta dirección se coloca en el bus de direcciones y se envía a la memoria de programa (ROM o segmento de código de RAM), donde se decodifica la dirección local y se activan los circuitos para permitir el acceso a la información almacenada en la posición seleccionada. Cuando la unidad de control y tiempos de la CPU envía la señal de lectura apropiada a la memoria, los bits de la instrucción se colocan en el bus de datos y se transfieren a una posición de la memoria borrador que se puede denominar genéricamente *registro de datos de memoria* (**Memory Data Register**), por ejemplo, AX, BX, CX

o DX. De allí la palabra de instrucción se transfiere a un registro de instrucción, donde normalmente se mantiene hasta el final del ciclo de máquina o del ciclo de instrucción.

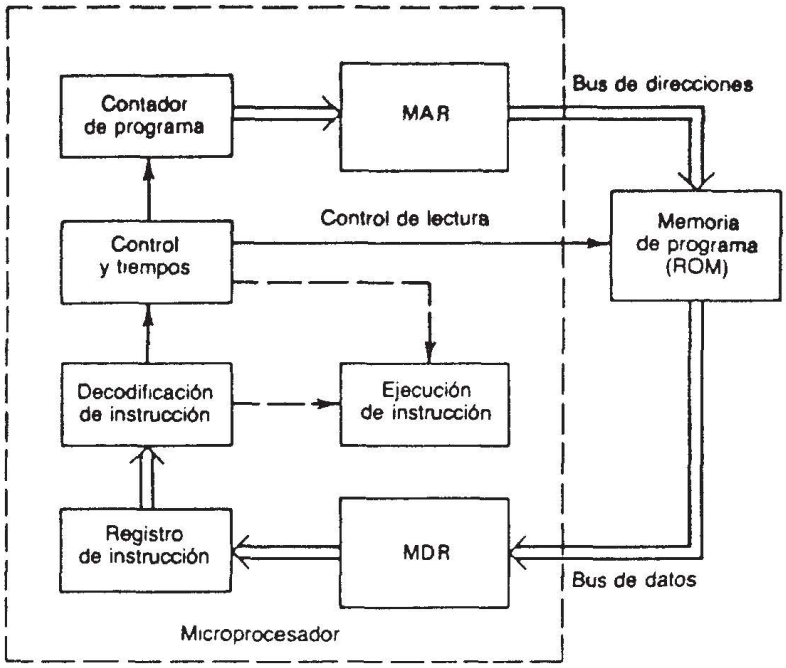


Figura 4.13
Diagrama de bloques del flujo de instrucción

La instrucción se decodifica entonces en las señales eléctricas adecuadas y, bajo el control de la unidad de control y tiempos se ejecutan las órdenes apropiadas en los distintos bloques funcionales del μP y en los diferentes sistemas del ordenador. Finalmente, el contador de programa se incrementa en uno, si la próxima instrucción a ejecutar es la que se encuentra en la siguiente dirección.

De acuerdo con el flujo de instrucción descrito, la instrucción se busca siempre en la memoria de programa. En el caso de requerirse *palabras de datos* (*data words*), la fuente de las mismas puede ser la memoria de datos o un dispositivo de E/S (**Figura 4.14**). De igual forma, una vez procesada por la CPU, la palabra de datos puede ser transferida a la memoria de datos o a un dispositivo de E/S. En cualquier caso, la dirección de la memoria o del dispositivo de E/S debe ser primero enviada desde el registro de dirección de

la CPU.

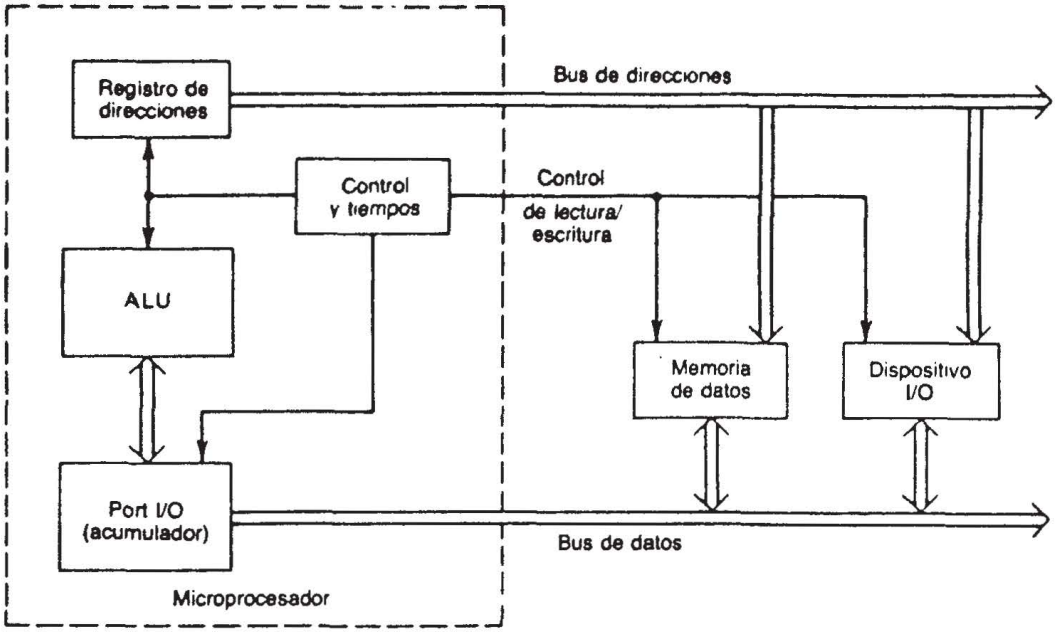


Figura 4.14

Diagrama de bloques del flujo de datos

En la mayoría de los ordenadores, el punto de entrada y salida del μP es un puerto de E/S que se ha denominado *registro acumulador* (*AX*). En muchos μP el acumulador también es el destino último de todos los datos procesados por la CPU.

Cabe indicar finalmente que todas las operaciones con palabras de datos que se especifican en una cierta instrucción tienen lugar durante la fase que se ha denominado "subciclo de ejecución" del ciclo de máquina.

BIBLIOGRAFIA

Bradley, D.J.; *"Assembly language programming for the IBM personal computer"*, Prentice Hall, Englewood Cliffs (1984).

Khambata, A.J.; *"Microprocesadores/microcomputadores. Arquitectura, software y sistemas"*, Gustavo Gili, Barcelona (1987).

Morgan, C.L. y Waite, M.; *"Introducción al microprocesador 8086/8088 (16 bit)"*, McGraw-Hill, México (1984).

Scanlon, L.J.; *"80286. Programación ensamblador en entorno MS DOS"*, Anaya Multimedia, Madrid (1988).

CAPITULO 5

LA UNIDAD DE INTERCAMBIOS (E/S)

5.1 TRANSFERENCIA DE INFORMACION

5.2 TRANSFERENCIAS E/S PROGRAMADAS

5.2.1 ESQUEMAS DE TRANSFERENCIA PROGRAMADA

5.3 TRANSFERENCIAS E/S CON INTERRUPCION

5.3.1 DEMANDA DE INTERRUPCION

5.3.2 SERVICIO DE INTERRUPCION

5.3.3 ASIGNACION DE PRIORIDADES

5.4 TRANSFERENCIA DE ACCESO DIRECTO A MEMORIA

5.4.1 INICIO DEL DMA

5.4.2 EJECUCION DEL DMA

5.5 TRANSFERENCIAS E/S PROGRAMABLES

5.5.1 PROCESADOR DE E/S INTEL 8089

5.5.2 CONTROLADORES DEL SISTEMA

5.5.2.1 CONTROLADOR PROGRAMABLE DE INTERRUPCIONES, INTEL 8259

5.5.2.2 CONTROLADOR PROGRAMABLE DE DMA, INTEL 8237

5.5.3 CONTROLADORES DE DISPOSITIVOS

5.5.3.1 CONTROLADORES DE PROPOSITO ESPECIAL

5.5.3.2 CONTROLADOR PROGRAMABLE PARALELO DE INTERFASE, INTEL 8255

5.5.3.3 CONTROLADOR PROGRAMABLE SERIE DE INTERFASE, INTEL 8251

5.1 TRANSFERENCIA DE INFORMACION

Ningún ordenador, independientemente de su tamaño, velocidad, capacidad de cálculo u otras sofisticadas características, es útil a menos que se pueda comunicar con el mundo exterior, es decir, con otros dispositivos del sistema llamados *dispositivos de E/S* o *periféricos* (*peripherials*), cuya función primaria es convertir información de una representación física (señales eléctricas digitales) a otra (caracteres o imágenes sobre papel, imágenes sobre pantalla, caracteres de teclado, señales eléctricas analógicas, etc.). Esta comunicación requiere la entrada de datos al ordenador desde los periféricos y la transferencia a los mismos desde el ordenador de datos procesados o señales de control. El bloque funcional que realiza esta tarea se denomina *unidad de intercambios* o, más corrientemente, *unidad de E/S* (*Entrada/Salida*) y está formada por unidades básicas llamadas *puertos de E/S* (*Input/Output ports*), que contienen la lógica de control y los registros direccionables utilizados para conectar el ordenador a los dispositivos periféricos.

Cabe destacar aquí que en gran parte de la bibliografía se identifican los puertos de E/S con las denominadas *interfases*. Se define una *interfase* o *interfaz* (*interface*) como las líneas de control y datos conectadas a un dispositivo que, junto a unas determinadas secuencias de señales, permiten la comunicación del dispositivo con su entorno. Obsérvese que la definición de puerto de E/S implica, evidentemente, la existencia de líneas de control y datos en esta unidad, así como el control de la misma por señales de mando. De aquí deriva probablemente la confusión de ambos términos, interpretados unívocamente como "nexo de unión" entre el microprocesador y los periféricos.

Este nexo de unión permite reconciliar las diferentes características del μP y los periféricos:

* Velocidad de operación más baja en los periféricos
* Tiempos de acceso diferentes
* Distintos formatos en los datos

Ello se logra haciendo que los puertos de E/S contengan las siguientes funciones:

* Registros de datos: permiten mantener un cierto tiempo los datos durante las transferencias E/S y así compensar las diferencias de velocidad.

* Decodificador de direcciones para la selección de dispositivos: esta función permite direccionar el dispositivo deseado a partir de una dirección enviada por la CPU.

* Decodificador de instrucciones: posibilita la recepción de señales de mando a través del bus de control.

* Decodificador de estado: permite recibir del dispositivo de E/S una señal de estado que indica si éste puede o no recibir datos en un momento determinado.

* Control y tiempos: esta función permite sincronizar las comunicaciones entre la CPU y los dispositivos de E/S y coordinarlas con el resto del sistema.

Además, durante las transferencias de datos el sistema sigue siempre una secuencia de eventos llamada *protocolo* (*protocol*), es decir, se rige por una serie de reglas que determinan los formatos en los cuáles se puede intercambiar la información entre las diferentes unidades del sistema.

Las transferencias de datos se distinguen por la extensión en que la CPU se encuentra involucrada en las operaciones de E/S. Si las operaciones de E/S están completamente controladas por la CPU, es decir, ésta ejecuta programas que inician, dirigen y finalizan las operaciones de E/S, se habla de una *transferencia E/S programada* (*programmed I/O*). En este caso es el μP, y no el periférico, el que inicia la transferencia, por lo que la velocidad del dispositivo de E/S debe ser compatible con la de la CPU; sin embargo, muy pocos periféricos son tan rápidos.

Un sistema mucho más eficiente y rápido es que el periférico interrumpa el programa principal y solicite servicio cuando está listo para una transferencia. Esta solicitud se denomina "interrupción" y la transferencia de información realizada bajo estas circunstancias se denomina *transferencia E/S con interrupción* (*I/O interrupts*). En un sistema de este tipo, aunque la transferencia se inicia con la solicitud del dispositivo externo, la CPU ejecuta la transferencia real bajo control del programa.

Por otro lado, un puerto de E/S puede ser provisto de la capacidad de transferir un bloque de información directamente hacia o desde la memoria principal sin intervención de la CPU. Ello requiere que el puerto sea capaz de generar direcciones de memoria y de transferir los datos desde y hacia el

bus del sistema. Aunque la CPU es la responsable de iniciar cada transferencia de un bloque, el puerto de E/S puede seguir con la transferencia sin ninguna actuación posterior de la CPU. A este tipo de transferencia de información se le denomina *transferencia de acceso directo a memoria* (*Direct Memory Access*), *DMA*.

Finalmente, existe también la posibilidad de que la CPU ceda completamente el control de la operación de transferencia al puerto de E/S, es decir, éste puede tener acceso directo a la memoria y puede interrumpir a la CPU, además de poder ejecutar programas por sí mismo. A este tipo de transferencias, realizadas por un μP adicional programable, se le denomina *transferencia E/S programable* (*programmable I/O*).

A continuación se analizarán los distintos tipos de transferencias indicados, tanto bajo el punto de vista funcional y lógico de las interfases relacionadas con ellas, como desde el punto de vista físico, de los circuitos integrados que permiten llevarlas a cabo y que generalmente se conocen como *controladores* (*controllers*), aunque se usa principalmente este témino para hacer referencia a las interfases de E/S programables.

5.2 TRANSFERENCIAS E/S PROGRAMADAS

Las transferencias E/S programadas son aquéllas que se inician y controlan enteramente con el μP, es decir, se ejecutan normalmente mediante una instrucción, una serie de etapas de programa o una subrutina (parte del programa que se puede llamar desde cualquier punto del programa principal tantas veces como se desee). La transferencia se realiza generalmente entre un registro de la CPU, normalmente el acumulador, y un registro de memoria, llamado *tampón* (*buffer*), contenido en el puerto de E/S.

La transferencia de información entre el dispositivo de E/S y la memoria principal requiere la ejecución de varias instrucciones por parte de la CPU, incluyendo una instrucción de entrada para mover una palabra del dispositivo de E/S a la CPU y una instrucción de almacenamiento, para pasar la palabra desde la CPU a la memoria principal. También pueden necesitarse algunas instrucciones adicionales para direccionar el cálculo y para contar el número de palabras de datos.

En los sistemas con E/S programada, los dispositivos de E/S, la memoria

principal y la CPU normalmente se comunican a través del bus del sistema, por lo que su bus de direcciones, que se utiliza para direccionar las posiciones de memoria, se puede utilizar también para seleccionar los dispositivos de E/S. Como se ha indicado anteriormente, la unión entre el bus del sistema y el dispositivo de E/S se denomina *puerto de E/S*, e incluye un registro de memoria tampón, direccionable, lo que lo diferencia muy poco de una posición de memoria, bajo el punto de vista de la CPU. Es por ello por lo que el puerto de E/S puede ser visto como tal o como una posición de memoria.

Cuando el puerto de E/S es visto como una posición de memoria (*memory-mapped I/O*) una instrucción que implique la transferencia de datos hacia una dirección se convierte automáticamente en una intrucción de E/S. Esto hace que no se necesiten instrucciones específicas de E/S. La Figura 5.1 muestra la estructura fundamental de un ordenador con este tipo de direccionamiento de E/S. Las líneas de control de lectura (READ) y de escritura (WRITE), que son activadas por la CPU cuando decodifica la instrucción de direccionamiento de memoria, se utiliza para iniciar tanto un ciclo de acceso a memoria como una transferencia de E/S.

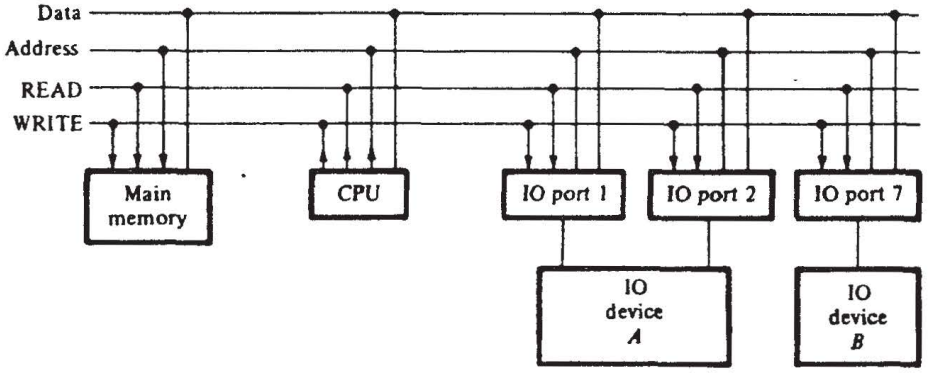


Figura 5.1
Puertos de E/S vistos como posiciones de memoria

Cuando el puerto de E/S es visto como tal (*I/O-mapped I/O*), los espacios de memoria y de direcciones de E/S se mantienen separados, tal como muestra la Figura 5.2. Una instrucción de referencia a memoria activa las líneas de control de ésta, pero no afecta a los dipositivos de E/S. Para

activar las líneas de control de lectura y escritura de éstos, se necesitan instrucciones específicas, que producen la transferencia entre el puerto de E/S direccionado y la CPU. Obsérvese que en estas condiciones un puerto de E/S y una posición de memoria pueden tener la misma dirección. Este esquema de direccionamiento es que que se usa en los modernos μP Intel 80i86.

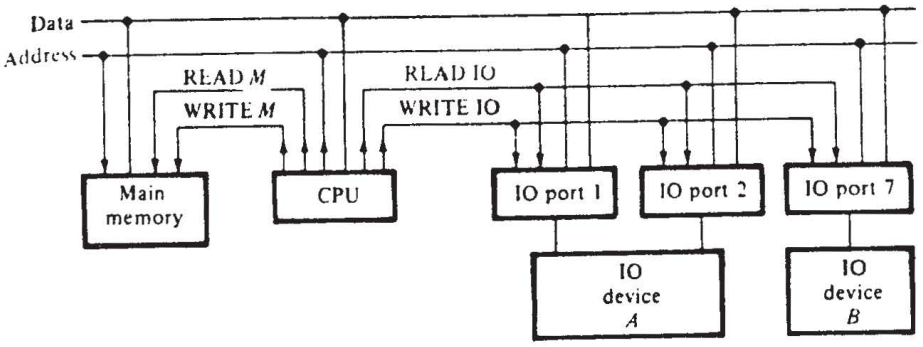


Figura 5.2
Puertos de E/S vistos como tales

5.2.1 ESQUEMAS DE TRANSFERENCIA PROGRAMADA

Las E/S programadas se llevan a cabo en casi todos los ordenadores modernos. La transferencia se ejecuta mediante una instrucción de transferencia que el programador inserta en el punto adecuado del programa para iniciar la acción del circuito integrado de la interfase de E/S. Esta instrucción realiza las tareas siguientes, siguiendo el protocolo establecido en el diseño del sistema:

* Direcciona el puerto de E/S
* Recibe los datos del puerto de E/S
* Envía los datos de la CPU al periférico
* Envía órdenes al periférico
* Verifica o no el estado del periférico

Obsérvese en el último punto que el estado del periférico puede ser o no verificado, lo que da lugar a dos esquemas de E/S programada: las incondicionales (o síncronas) y las condicionales (o asíncronas).

La *transferencia incondicional* (*unconditional transfer*) es la más simple y directa, ya que tiene lugar cuando lo ordena la CPU, la cuál no

verifica si el estado del periférico es "listo" (*ready*) u "ocupado" (*busy*). Es decir, el periférico ha de estar listo para enviar una palabra a la CPU cuando ésta inicia la transferencia, por lo que ambas unidades han de estar sincronizadas. Este tipo de transferencia tiene dos inconvenientes que limitan su uso en las aplicaciones de los ordenadores:

* El periférico siempre ha de estar listo, lo que significa que este tipo de transferencia sólo puede hacerse con periféricos cuyos tiempos de funcionamiento son fijos y conocidos, ya que éstos deben ser capaces de cumplir la transferencia requerida dentro del tiempo de instrucción del μP.

* Como los periféricos son mucho más lentos que el μP, la restricción anterior obliga al programador a prever un tiempo entre transferencias, para asegurar que el dispositivo de E/S haya cumplido la transferencia anterior.

A pesar de estas limitaciones, las transferencias E/S incondicionales se usan en muchas aplicaciones.

La *transferencia condicional* (*conditional transfer*) se utiliza con mucha más frecuencia en los ordenadores, ya que no obliga a que la CPU y el periférico estén sincronizados, es decir, es asíncrona. En este esquema primero se consulta el estado del periférico para saber si está listo para la transferencia, y ésta sólo tendrá lugar si se cumple esta condición. Si el periférico responde que está ocupado, el programa entra en un bucle y va repitiendo la consulta hasta que el periférico está listo. Así, se ejecuta la transferencia mediante dos instrucciones:

* Adquisición del estado del periférico direccionado informando a la CPU del mismo; ésta toma la decisión de repetir la consulta haciendo un bucle o seguir a la siguiente instrucción, en función del estado del periférico.

* Cuando el periférico indica a la CPU que está listo para la transferencia, el programa vanza a la siguiente instrucción, que ejecuta la transferencia, sea ésta de entrada o de salida.

El inconveniente principal de las transferencias E/S condicionales es que las repetidas consultas del estado del periférico consumen tiempo de la CPU. La principal ventaja es que permiten sincronizar el μP con los periféricos más lentos, cuyos tiempos pueden ser fijos o no.

5.3 TRANSFERENCIAS E/S CON INTERRUPCION

Las transferencias programadas estudiadas anteriormente tienen algunos inconvenientes, ya citados, que las hacen inaplicables en ciertas operaciones de los μP.

La desventaja principal de la transferencia incondicional es que la velocidad del dispositivo de E/S debe ser compatible con la de la CPU. De otra manera, si el periférico no está listo, cuando la CPU inicia una transferencia, se produce un defecto que obliga a suspender la ejecución del programa.

La transferencia condicional también tiene un inconveniente derivado de la verificación por parte de la CPU del estado del periférico antes de que tenga lugar la transferencia real de datos. Si el periférico no está listo, la CPU entra en un bucle; durante el proceso de consulta y bucle, el μP no realiza ningún trabajo positivo.

Por estos motivos cabe la alternativa de dotar al propio puerto de E/S de la capacidad de solicitar el servicio de la CPU para que atienda al dispositivo de E/S cuando éste está listo para la transferencia. Este tipo de demanda se denomina *interrupción* (*interrupt*), definiéndose ésta como el proceso de alteración del flujo normal de ejecución de un programa para responder a señales generadas por un puerto de E/S.

La capacidad de interrupción libera a la CPU de la tarea de comprobar periódicamente el estado de los dispositivos de E/S, ya que es el periférico el que inicia la transferencia. Sin embargo, la transferencia en sí es ejecutada por el ordenador bajo el control del programa. Sus usos principales son la solicitud a la CPU para iniciar una nueva operación de E/S, la indicación del final de dicha operación y la indicación de que se ha producido algún error en el sistema.

El método básico de interrupción a la CPU es por la activación de una línea de control que conecta el puerto de E/S, fuente de la interrupción, a la CPU. La señal de interrupción se almacena en un registro que es consultado periódicamente por la CPU, normalmente al final de cada ciclo de instrucción, lo que constituye el procedimiento de consulta de una *demanda de interrupción*.

El reconocimiento de la demanda de interrupción hace que la CPU tenga que ejecutar un programa específico de *servicio de interrupción*, que

normalmente es diferente en función de la fuente de la interrupción. Ello obliga a la CPU a averiguar la dirección de memoria donde se encuentra el programa a utilizar.

Por otra parte, la presencia de dos o más demandas de interrupción simultáneas requiere la *asignación de prioridades* a las interrupciones para optimizar la velocidad de transferencia de información al resto del sistema.

5.3.1 DEMANDA DE INTERRUPCION

Cuando se utiliza el sistema de interrupciones, no se deben insertar instrucciones de transferencia de E/S en un programa, como se hace en las transferencias programadas, ya que el programador no puede prever en qué punto de la ejecución del programa uno de los puertos de E/S lo interrumpirá y solicitará servicio. El μP se debe programar de tal forma, que pueda dar servicio a cualquier dispositivo que lo solicite en cualquier momento.

Ahora bien, antes de servir la interrupción, en la CPU suceden una serie de eventos, tal como se muestra en el esquema de la Figura 5.3.

Como las instrucciones para servir a los periféricos no están incluidas en el programa principal, la CPU, después de completar cada instrucción, debe verificar la existencia de una interrupción antes de seguir con la siguiente instrucción. Esta operación la realizan los circuitos de la CPU bajo la supervisión de la unidad de control y tiempos.

De esta forma, cuando se completa la ejecución de cada instrucción, la CPU consulta la línea de control que la conecta con el puerto de E/S que provoca la interrupción, para ver si ha recibido una petición de interrupción. Esta línea de control se suele denominar línea de *demanda de interrupción* (INT*errupt* REQ*uest*). En ausencia de tal señal, el contador de programa se incrementa y se busca y ejecuta la siguiente instrucción. Este proceso se repite indefinidamente y el programa principal continúa su ejecución normal hasta que recibe una señal INT REQ.

En algunas aplicaciones, ciertas partes del programa suelen ser críticas y el servicio de interrupciones durante la ejecución de este grupo de instrucciones no se debe permitir. Tales grupos críticos de instrucciones se pueden proteger de las interrupciones mediante una función de prohibición. Una instrucción especial, situada en la cabeza del grupo crítico, prohibe la

señal INT REQ y el programa sigue su curso. Otra instrucción especial, situada inmediatamente después de la última instrucción del grupo crítico, restituye los circuitos de INT REQ a su estado normal.

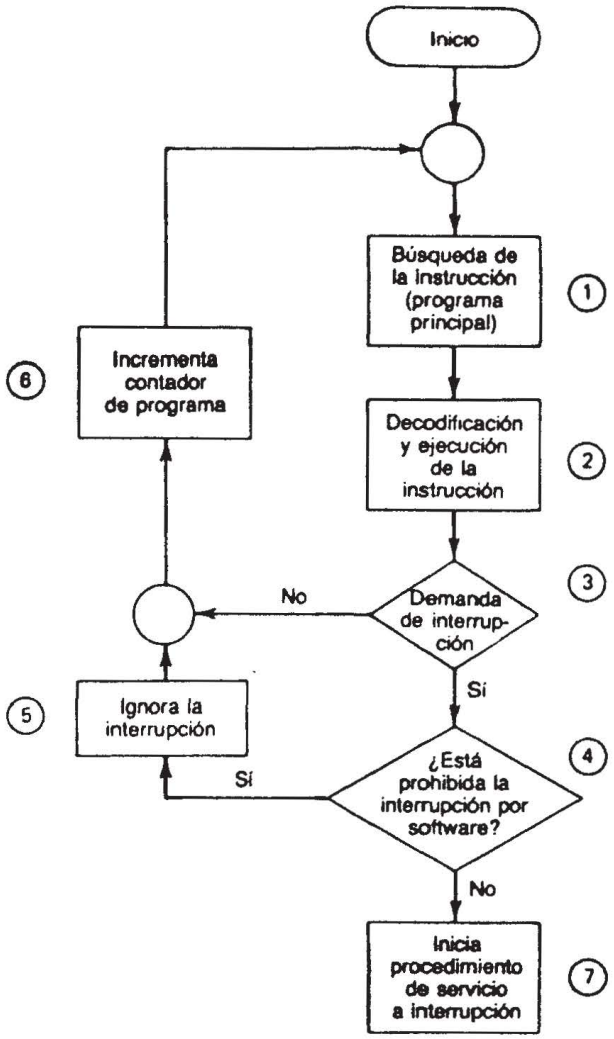


Figura 5.3

Consulta de una petición de interrupción

Después de que la CPU ha reconocido la demanda de interrupción se debe iniciar el servicio de la misma.

5.3.2 SERVICIO DE INTERRUPCION

El servicio a una señal de demanda de interrupción requiere la llamada previa a una subrutina que debe manejar adecuadamente una serie de eventos,

tal como muestra la **Figura 5.4**. Esta secuencia la dispara generalmente una señal generada por la CPU, que se denomina *reconocimiento de interrupación* (INT*errupt* ACK*nowledge*).

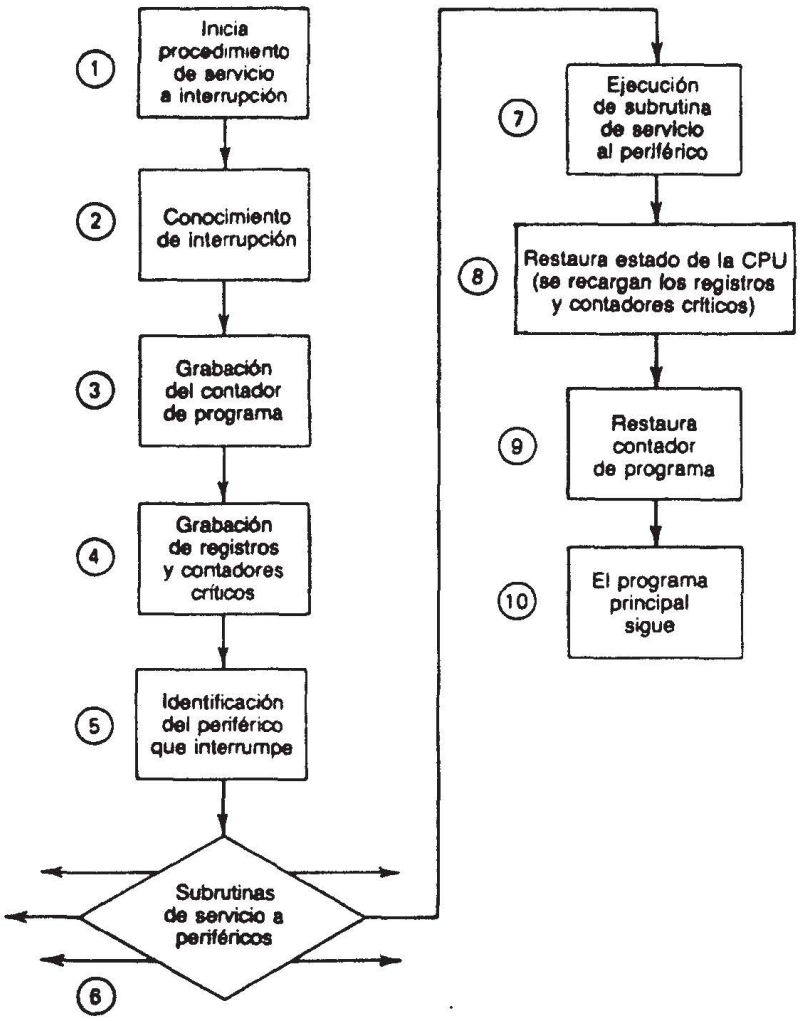


Figura 5.4

Procedimiento de servicio de interrupción

Para hacer saltar el programa principal a la subrutina de manejo de interrupción se pueden utilizar tres técnicas, que se resumen en los siguientes puntos:

* Utilización de una instrucción de bifurcación: la señal INT ACK activa un registro en el puerto de E/S que contiene una instrucción de ramificación y la correspondiente dirección de inicio de la subrutina de

manejo de interrupción, y almacena este contenido en los registros del μP, que ejecuta la instrucción.

* Utilización de un puntero de dirección: un registro específico o puntero de la CPU contiene la dirección inicial de la subrutina de manejo de interrupción; cuando la CPU recibe una señal INT REQ, la lógica del μP llamará automáticamente a ese puntero y lo colocará en el registro contador de programa.

* Utilización de una dirección dada exteriormente: el mismo puerto de E/S que interrumpe, suministra a la CPU la dirección de inicio de su propia subrutina de servicio que, por tanto, debe contener asimismo las instrucciones generales de manejo de interrupción.

Después de completar la subrutina de manejo de interrupción, el control debe volver al programa principal de forma ordenada, por lo que es necesario realizar inicialmente una serie de operaciones:

* Para que el programa conozca la dirección de retorno, hay que salvar el contenido del contador de programa.

* Como la señal de demande de interrupción es impredecible en el tiempo, hay que salvar el contenido de los registros y contadores críticos:

- Acumulador
- Registros de direcciones
- Registros índices
- Registros de estado

Generalmente se suele utilizar el segmento de pila para almacenar el contenido de estos registros, depués de lo cual es necesario identificar el puerto que general la interrupción para poder llamar a la subrutina de manejo adecuada. La identificación del puerto implica una asignación de prioridades a las interrupciones, lo que se estudiará detenidamente más adelante.

La última instrucción de servicio al puerto de E/S que ha demandado la interrupción devuelve el control a la subrutina de manejo de interrupción. Los registros y contadores críticos se recargan primero con su contenido anterior a la interrupción y luego se recarga el contador de programa con la dirección actualizada de retorno, que reinicia la operación de interrupción.

5.3.3 ASIGNACION DE PRIORIDADES

Antes de llamar y ejecutar la subrutina de servicio, el puerto de E/S que envía la señal de demanda de interrupción debe ser identificado, ya que esta señal va a la CPU por una única línea. Asimismo, la existencia de esta única línea implica la posibilidad de que vaios puertos soliciten una interrupción simultáneamente, lo que obliga a diseñar un sistema de asignación de prioridades para discriminar entre estas demandas competitivas.

La identificación y asignación de prioridades a los periféricos demandantes de interrupción se puede llevar a cabo por distintos métodos.

* La *interrupción vectorizada* (*vectored interrupt*) es la técnica más directa de identificación de un puerto de E/S, ya que éste hace saltar el control desde el programa principal a su propia subrutina de servicio, identificándose a sí mismo a la CPU mediante una dirección única, llamada *vector*, que es la dirección de inicio de su subrutina. Este sistema sirve las interrupciones estrictamente en el orden en que se reciben.

* La *interrupción indagada por programa* (*polling*) utiliza una técnica de búsqueda que verifica el estado de cada puerto de E/S para identificar el que le interrumpe. Si más de un puerto envía una señal de demanda de interrupción simultáneamente, será servido primero el que tenga una mayor prioridad. Esta se establece previamente por su posición relativa en el programa de búsqueda.

* La *interrupción indagada físicamente* (*daisy chaining*) combina ambas técnicas anteriores. La búsqueda se hace mediante puertas AND utilizando la señal de reconocimiento de interrupción; cuando se halla la unidad que interrumpe, ésta se identifica a sí misma enviando su propia dirección o vector a la CPU.

* La *interrupción con prioridad multinivel* (*multi-level priority*) permite interrumpir el servicio a un puerto de E/S cuando lo solicita una interrupción de mayor prioridad asignada. Utilizando las técnicas anteriores, durante el proceso de una interrupción, las demandas de otros puertos de E/S son ignoradas. Sin embargo, en este caso, al aparecer una demanda de interrupción cuando se está sirviendo otra anterior, la lógica del sistema determina las prioridades relativas para decidir si se debe o no saltar a la subrutina de servicio del segundo demandante.

Cabe destacar finalmente que en todos los ordenadores que utilizan interrupciones existe una clase especial de las mismas llamadas *interrupciones no enmascarables* (*non-masking interrupts*), que tienen la mayor prioridad y no pueden ser desactivadas durante ninguna etapa de la gestión de interrupciones. Estas interrupciones se generan en casos de emergencia, como fallos de alimentación eléctrica o errores de memoria.

5.4 TRANSFERENCIA DE ACCESO DIRECTO A MEMORIA

En muchas aplicaciones es preciso transferir grandes cantidades de datos entre la memoria principal y varios dispositivos de E/S, generalmente dispositivos de memoria auxiliar, como discos o cintas. Utilizando los métodos de transferencia programada o con interrupción, se perdería una gran cantidad de tiempo para estas operaciones, ya que la CPU se pasaría la mayor parte de su tiempo gestionando transferencias.

El *acceso directo a memoria* (*Direct Memory Access*) resulta en estos casos una alternativa muy eficaz para transferir grandes bloques de datos (grupos de palabras que están almacenados en posiciones de memoria sucesivas) entre la memoria principal del ordenador y la memoria auxiliar, evitando su paso a través de la CPU. Para ello el puerto de E/S que ejecuta este tipo de transferencia toma el control durante uno o más ciclos de máquina, lo que libera completamente a la CPU, que se puede dedicar a otras tareas y permite al sistema realizar dos tareas simultáneamente.

Como este sistema es físicamente complejo, el puerto de E/S requiere circuitos adicionales para ejecutar las transferencias sin intervención de la CPU, por lo que éstos se suelen englobar bajo el concepto de *controlador de DMA* (*DMA controller*), denominándose el conjunto puerto-controlador, *interfase DMA* (*DMA interface*).

La configuración funcional de este tipo de transferencia de información se muestra en la **Figura 5.5**. Obsérvese que, tanto la CPU como el DMA pueden acceder a la memoria en cualquier momento, obligando al otro a desconectarse del bus del sistema. Mientras el DMA está comunicando con la memoira, la CPU queda libre para realizar otras tareas que no requieran acceso a la memoria o al bus.

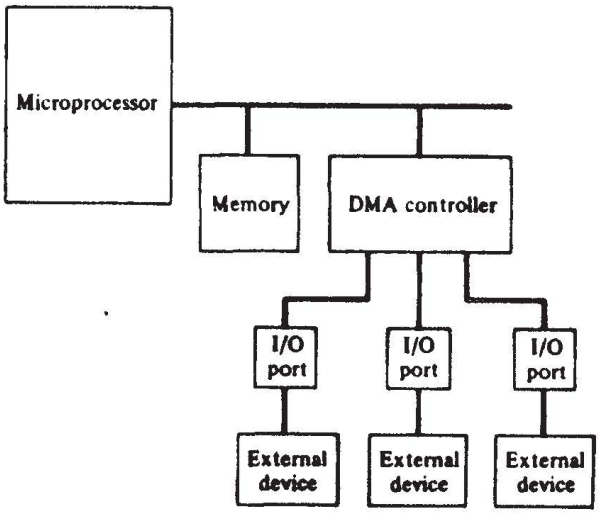


Figura 5.5

Transferencia por acceso directo a memoria

Para poder tomar el control de una transferencia de información, la interfase DMA debe contener cuatro registros fundamentales:

* Registro de direcciones: contiene la dirección de la siguiente posición de memoria a acceder, bien para una operación de lectura o de escritura.

* Registro contador: contiene el número de posiciones contiguas de memoria que quedan por llenarse durante una secuencia de escritura, o de las cuáles han de leerse datos para el balance de una secuencia de lectura.

* Registro de control: identifica la dirección del flujo de datos y se utiliza para empezar, parar o controlar las operaciones de DMA.

* Registro de estado: se utiliza para identificar el estado de cada operación de DMA que pueda estar efectuándose.

5.4.1 INICIACION DEL DMA

Existen dos métodos para iniciar una transferencia de DMA: indicación del estado de memoria y robo de ciclo.

La *indicación del estado de memoria* (*transparent DMA*) es un método simple de DMA en que la transferencia tiene lugar sin informar a la CPU, basado en que durante muchos ciclos de máquina ésta no accede a la memoria, sino que realiza procesos de palabra en los registros y contadores internos.

En tales casos la CPU informa al resto del sistema mediante una señal especial, que indica que el bus de acceso a la memoria está libre, por lo que durante este tiempo puede tener lugar una transferencia de DMA. La ventaja principal de este método es que no retrasa las operaciones de la CPU, pero su principal inconveniente es que sí retrasa las operaciones de DMA. Así, las transferencias de DMA pueden tener lugar sólo cuando la CPU no accede a memoria, por lo que la velocidad de transferencia de datos tiende a ser irregular y aleatoria.

El *robo de ciclo* (*cycle stealing*) es un método que obliga a la CPU a abandonar el control de sus operaciones durante un tiempo, en favor del DMA. El dispositivo de E/S inicia la transferencia de DMA enviando una señal de *demanda de DMA* (DMA REQ*uest*) a la CPU, cuando está listo para realizar una transferencia. La CPU responde inmediatamente con una señal de *reconocimiento de DMA* (DMA ACK*nowledge*), completa la instrucción en curso y suspende sus funciones durante el próximo ciclo de máquina, por lo que se retrasan las tareas programadas determinadas por el programa principal. Durante el siguiente ciclo, la CPU queda inactiva y cede el control a la interfase de DMA; cuando ésta ha finalizado la transferencia, lo indica a la CPU, que continúa la operación del programa principal hasta que se realice otra transferencia de DMA.

La CPU consulta la entrada de la demanda de DMA en la ejecución de cada instrucción del programa principal durante la fase del ciclo de instrucción. Por tanto, la transferencia de DMA tiene lugar "robando" un ciclo o intervalo de tiempo del programa principal, tal como se muestra gráficamente en la Figura 5.6 (a). Obsérvese que las transferencias de DMA quedan intercaladas con los ciclos de las operacioanes programadas en la CPU, es decir, operan en tiempo compartido con las operaciones del programa principal.

Ahora bien, la transferencia de una sola palabra de datos no justifica el uso de DMA. Para que éste sea rentable, deben transferirse un gran número de palabras o bloques de palabras, lo que significa que, después de recibir la señal de demanda de DMA, la CPU quedará inactiva durante más de un ciclo. Es decir, la situación real se parece más a la representada en la Figura 5.6 (b).

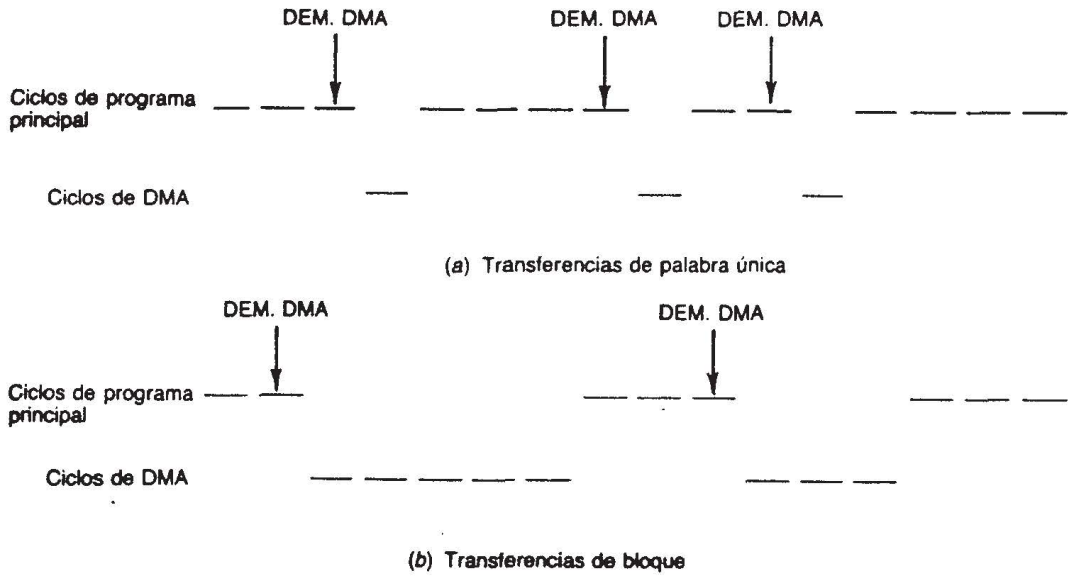


Figura 5.6
Transferencias de DMA con robo de ciclo

Obsérvese que durante el intervalo de tiempo en que tiene lugar la transferencia, los contadores y registros de la CPU no resultan afectados, por lo que su estado no debe ser salvado en la memoria ni restaurado, y la CPU puede seguir con su funcionamiento normal después de la operación de DMA. Además, como la interfase de DMA no devuelve el control a la CPU hasta que no haya completado la transferencia, la velocidad de ésta es regular, continua y elevada, lo que hace de este método el más adecuado para la transferencia rápida y masiva de datos entre la memoria principal y un dispositivo de memoria auxiliar. Al estudiar la ejecución de la transferencia de DMA, se considerará sólo este método.

5.4.2 EJECUCION DEL DMA

Como se ha indicado anteriormente, no está justificado el DMA para transferencias de palabra única, por lo que éstas afectan a bloques de datos que están almacenados, o deben ser almacenados, en una serie de posiciones de memoria consecutivas. Una transferencia de bloque requiere que el dispositivo de E/S (memoria auxiliar) suministre primeramente la dirección de inicio del bloque de datos; además, el sistema debe seguir la pista de las direcciones siguientes de cada transferencia de palabra. Esto se consigue normalmente

cargando en el registro de direcciones de la interfase de DMA la dirección de la primera palabra del bloque e incrementando este registro después de haberse transferido cada palabra, tal como muestra el diagrama de flujo de la **Figura 5.7. Este proceso sigue hasta que se ha transferido el bloque de datos** completo, cuya magnitud se carga en el registro contador de la interfase de DMA, y se reinicia la ejecución del programa principal.

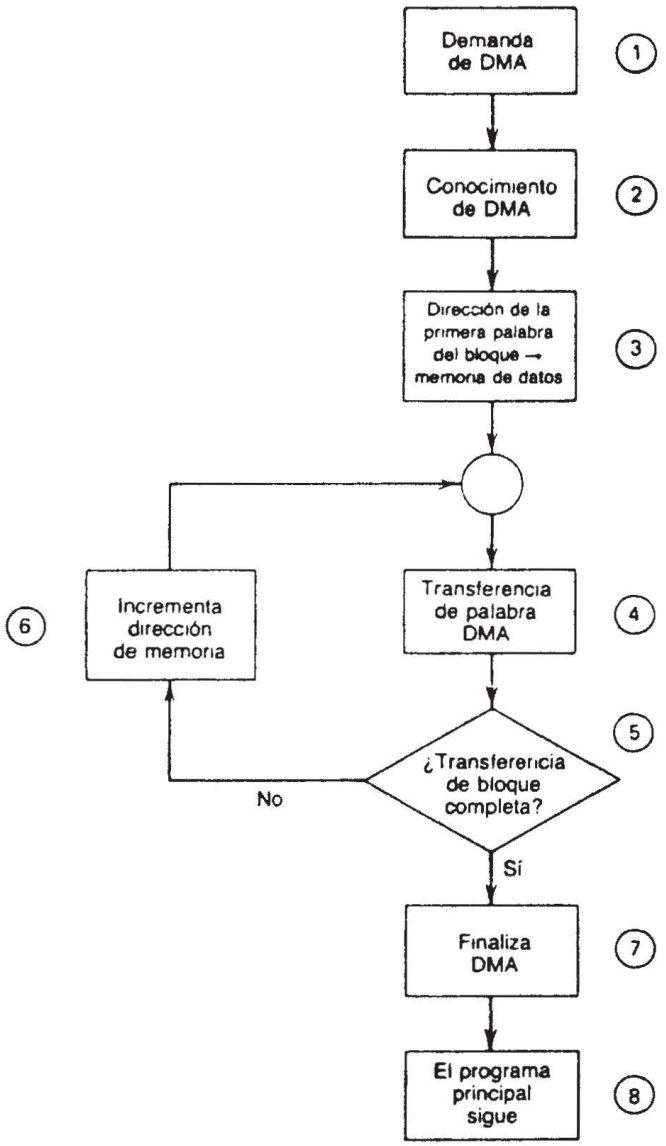


Figura 5.7
Transferencia de DMA de un bloque de datos

Ahora bien, en muchas aplicaciones, una transferencia de DMA puede ser

de varios bloques de datos diferentes. Estos bloques pueden o no estar colocados en posiciones adyacentes de la memoria de datos, y cada bloque puede o no constar del mismo número de palabras. En cualquier caso, las transferencias de DMA de cada uno de los bloques se pueden realizar de forma continua.

Es evidente, pues, que además de la dirección inicial de memoria de cada bloque, el dispositivo de E/S (memoria auxiliar) debe informar a la interfase de DMA del número de bloques a transferir, a cada demanda de DMA. Esto asegurará que el programa principal prosiga solamente cuando se hayan transferido todos los bloques de datos y se lleva a cabo cargando los valores correspondientes en los registros de direcciones y contador de la interfase de DMA.

En la **Figura 5.8** se muestra el diagrama de flujo de una transferencia de DMA de varios bloques de datos, conteniendo cada uno un número diferente de palabras y no colocados necesariamente de forma contigua en la memoria de datos.

Obsérvese finalmente que pueden incluirse funciones adicionales en un sistema de DMA. Así, es posible disponer de varios canales de DMA en un mismo controlador de DMA y se puede utilizar también un sistema de DMA con prioridad multinivel, similar al indicado para la transferencia de E/S con interrupción. En un sistema de este tipo, un dispositivo de E/S con mayor prioridad de DMA puede interrumpir la operación de otro que está realizando la transferencia con un nivel de prioridad inferior.

5.5 TRANSFERENCIAS DE E/S PROGRAMABLES

Los métodos de transferencia de E/S expuestos hasta aquí son fijos, no programables. Las funciones primarias que realizan estos sistemas son fijas e inalterables, mientras que las transferencias de E/S en sí tienen lugar bajo control del programa. En los ordenadores, estos sistemas ocupan uno o más circuitos integrados, cuyas características y funcionamiento se fijan durante la fabricación de los mismos, y el usuario los debe emplear de la forma exacta especificada por el fabricante.

Sin embargo, los ordenadores modernos ejecutan complejos y largos programas que pueden necesitar distintos dispositivos de E/S, lo que exige considerar simultáneamente la velocidad de transmisión, la forma en que se

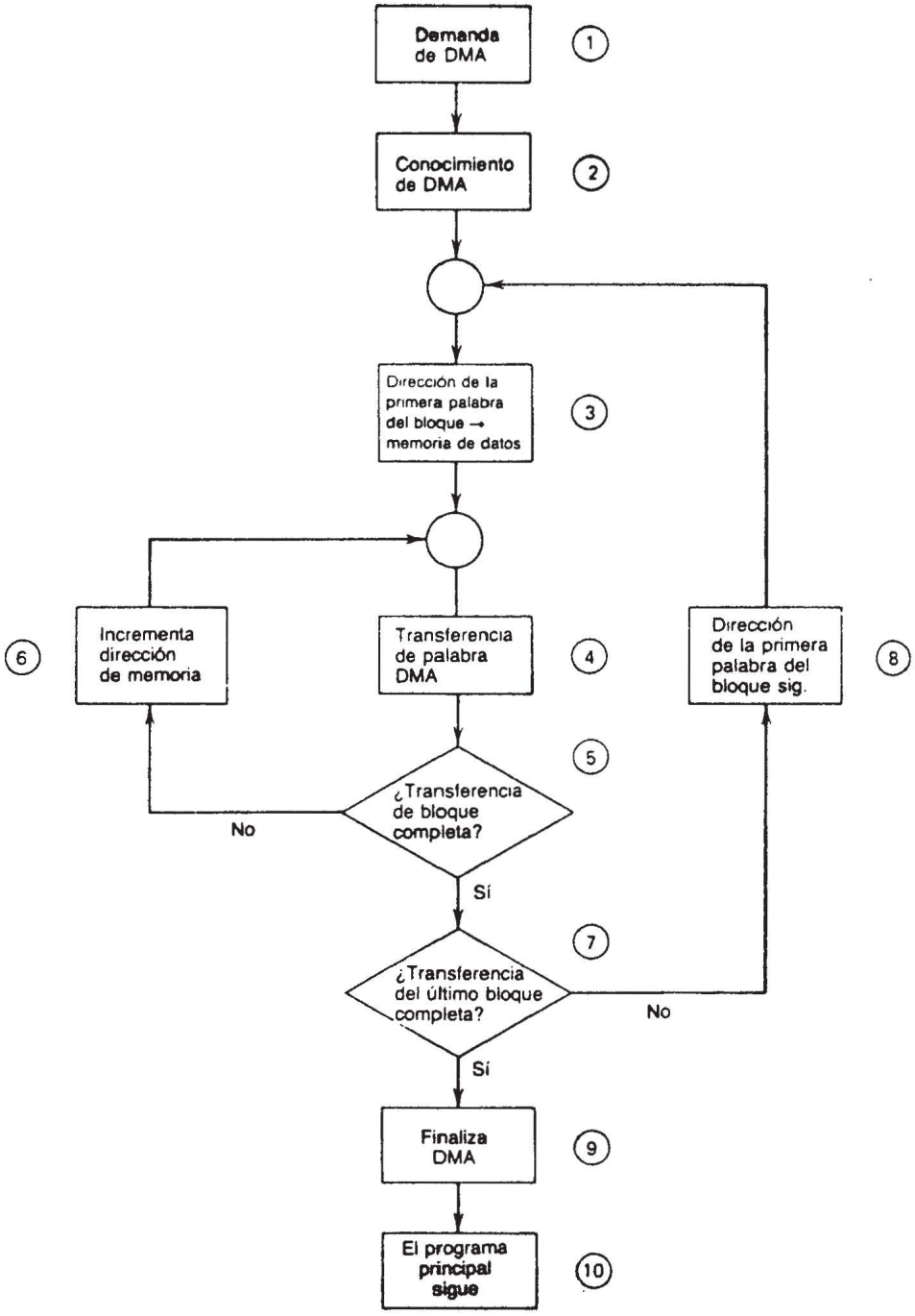


Figura 5.8

Transferencia de DMA de varios bloques de datos

agrupa la información, el grado de control necesario, la cantidad de traducción o conversión de los datos necesaria y el sistema de detección y

corrección de errores utilizado, ya que cada dispositivo de E/S tiene requerimientos distintos respecto a cada uno de estos puntos. Es por ello que se ha buscado una mayor flexibilidad en las transferencias de E/S que la que pueden proporcionar las indicadas anteriormente, lo que ha llevado a una amplia utilización de los puertos de E/S o, en general, interfases, programables, cuyas características funcionales se pueden modificar o alterar desde el ordendador mediante ciertas instrucciones. Es decir, se utiliza un procesador adicional que se encarga exclusivamente de las operaciones de E/S (*Input/Output Processor*), bajo la dirección de la CPU, tal como se muestra en el esquema de la **Figura 5.9**. De hecho, incluso se utilizan, además de un procesador general de E/S, otros procesadores encargados de cada dispositivo periférico, con objeto de optimizar al máximo las operaciones de E/S.

En esta línea se han desarrollado diversos procesadores de E/S generales, el más popular de los cuáles es el Intel 8089, e innumerables procesadores o controladores de dispositivos de E/S, siendo asimismo los más populares los de la gama Intel, debido a su aplicación masiva en los modernos ordenadores personales. Por todo ello se estudiarán a continuación estos procesadores como ejemplos característicos de puertos de E/S programables, aunque sólo se considerarán sus características principales.

5.5.1 PROCESADOR DE E/S, INTEL 8089

El *procesador de E/S* (*Input/Output Processor*), IOP 8089, es un procesador por sí mismo, pero con un juego de instrucciones de propósito especial que le permiten actuar como procesador periférico, es decir, todos los dispositivos de E/S se comunican con el 8089 que, a su vez, comunica con el μP 80*i*86, lo que aumenta sustancialmente el rendimiento de todo el sistema. El IOP 8089 es el encargado de esperar las señales de los dispositivos de E/S, quedando liberada de esta tarea la CPU, que podrá dedicarse a realizar los cálculos generales y de gestión de todo el sistema. Es capaz de gestionar las E/S mejor que la CPu al poder transferir datos entre la memoria y los periféricos con varios tipos de conversión y controlar la E/S en menos ciclos que los que necesitaría el 80*i*86. Este envía órdenes al IOP 8089 a través de las líneas de señal que unen ambos procesadores y mediante mensajes en zonas de memoria compartida. Como respuesta a estas

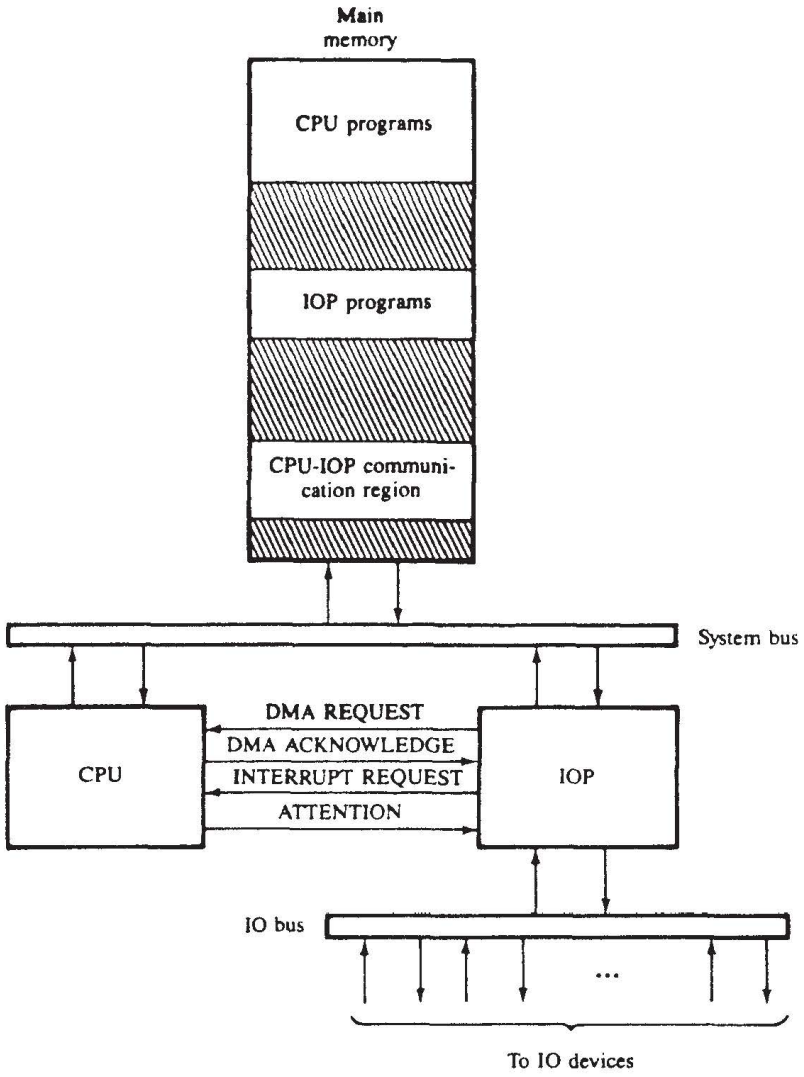


Figura 5.9
Sistema de transferencia con IOP

órdenes, el IOP 8089 realiza las tareas de E/S correspondientes por medio de su grupo de instrucciones, tareas que incluyen:

* Inicialización y supervisión de los controladores de dispositivos
* Transferencias inteligentes, tipo DMA, entre los dispositivos de E/S y la memoria

Este procesador tiene las mismas necesidades de sincronización y alimentación que el 80*i*86 y puede conectarse a éste de dos maneras: en modo local y en modo remoto.

La conexión en *modo local* (*local mode*) implica que la CPU, el IOP, la memoria principal y los controladores de dispositivos se hallan conectados al bus del sistema, tal como muestra la **Figura 5.10**. La ventaja de este modo reside en su bajo coste, al requerir un número reducido de circuitos integrados, pero tiene como desventaja la carga que añade al bus del sistema.

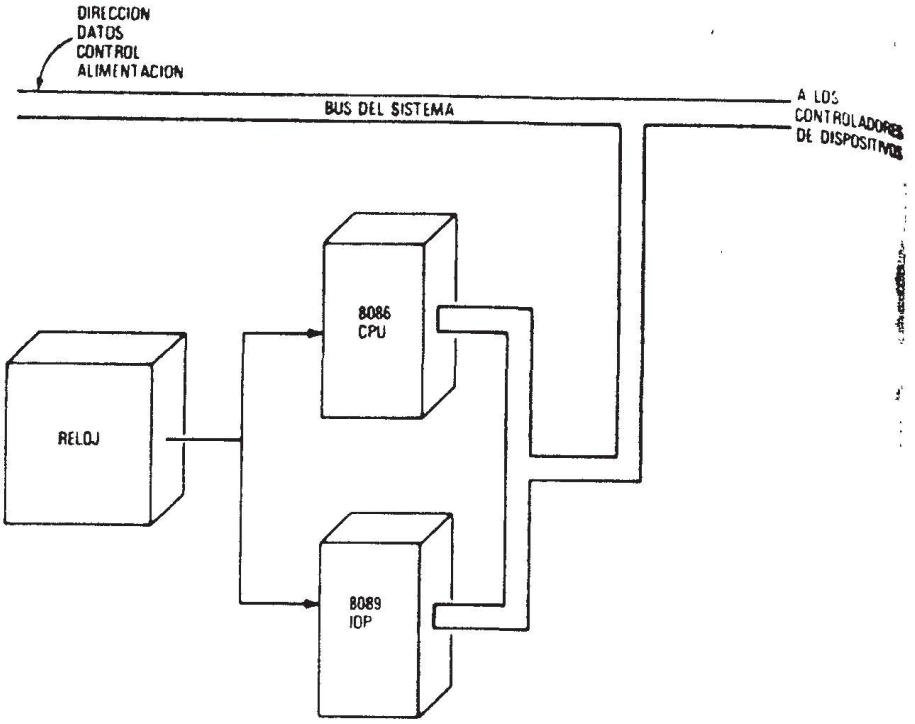


Figura 5.10
Modo local para el IOP 8089

La conexión en *modo remoto* (*remote mode*) implica la formación de un pequeño grupo entre el IOP y su propio bus local, al que se conectan los controladores de E/S; este grupo se conecta al bus del sistema, como se muestra en la **Figura 5.11**. En un sistema grande, podría haber varios grupos de este tipo, de manera que la actividad de E/S se distribuiría entre varios procesadores y varios buses distintos.

El IOP 8089 es capaz de procesar dos trabajos o *tareas* (*tasks*) al mismo tiempo, cada uno de los cuáles podría constituirse como la interfase de un dispositivo de E/S distinto. Para poder ejecutar dos tareas concurrentes, este procesador está dividido en dos secciones o *canales* (*channels*), uno para

cada tarea. Cada canal tiene su propio juego de registros y su propio programa, al que se denomina *bloque de tarea* (*task block*), que reside en la memoria del sistema o en el espacio de E/S.

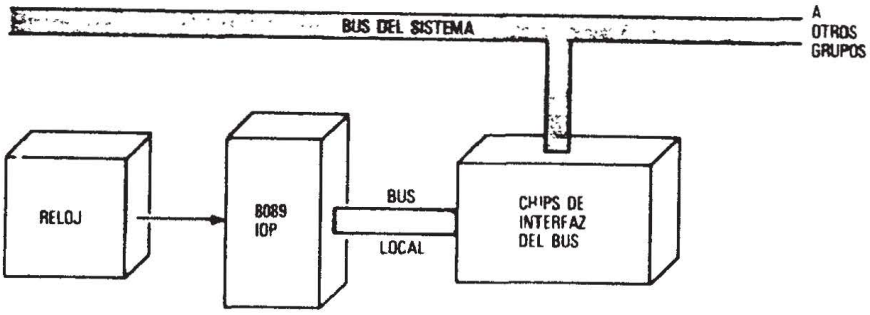


Figura 5.11
Modo remoto para el IOP 8089

Realmente, en un momento dado, un solo canal está realizando tareas. La *unidad común de control* (*Common Control Unit*) del 8089 se encarga de coordinar las tareas de los dos canales, dando control al canal de prioridad más alta, o si ambos tienen la misma prioridad, solapando ambas tareas, de manera que se realicen alternativamente instrucciones de ambos canales.

El IOP 8089 está especialmente diseñado para realizar transferencias a gran velocidad, es decir, del tipo de DMA, pudiendo ser la fuente y el destino un controlador de dispositivos, una memoria auxiliar o la memoria principal. Así, antes de que el IOP comienza a trabajar, la CPU carga en él a velocidad normal un bloque de parámetros y luego el IOP inicializa los circuitos controladores con vistas a una transferencia. tras ésto, una instrucción especial lleva a cada canal al modo de DMA.

La estructura interna simplificada del IOP 8089 puede observarse en la Figura 5.12.

5.5.2 CONTROLADORES DEL SISTEMA

Los controladores del sistema son interfases de E/S programables que se encargan de controlar automáticamente funciones especiales del sistema como la coordinación de varias interrupciones o la transferencia rápida de bloques

de datos. Se estudiarán aquí el controlador programable de interrupciones, Intel 8259, que controla las interrupciones, y el controlador programable de DMA, Intel 8237, que realiza la función de transferencia rápida.

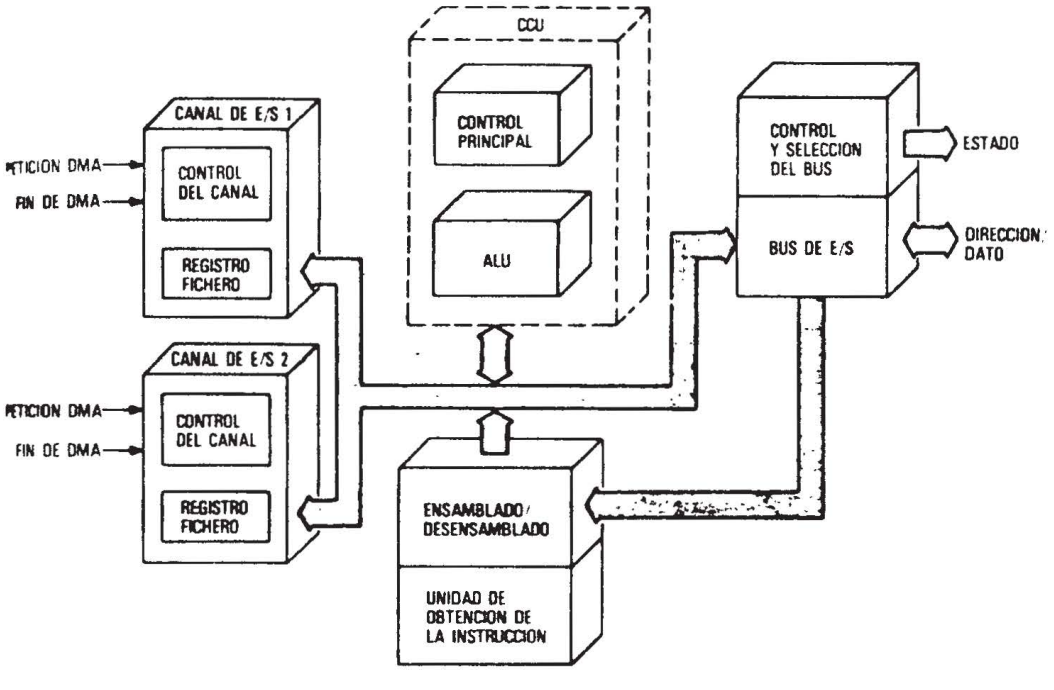


Figura 5.12
Estructura interna del IOP 8089

5.5.2.1 CONTROLADOR PROGRAMABLE DE INTERRUPCIONES, INTEL 8259

El *controlador programable de interrupciones* (*Programmable Interrupt Controller*), PIC 8259, se usa en la gestión directa de las interrupciones de hasta 8 dispositivos distintos y de hasta 64 dispositivos si se conectan 8 circuitos 8259 juntos. Cada dispositivo de E/S tiene una línea de interrupción que se conecta a una de las 8 líneas de interrupciones del PIC, que puede programarse para que ignore o controle cualquier combinación de estas líneas. El esquema del PIC 8259 se muestra en la **Figura 5.13**.

La selección de interrupciones se determina a través de la llamada *máscara de interrupciones* (*interrupt mask*), un byte que envía la CPU a este controlador a través del puerto de E/S correspondiente, cada uno de cuyos bits corresponde a uno de los 8 dispositivos. Si se quieren ignorar las interrupciones de un dispositivo, basta con poner a 1 el bit de la máscara de

interrupciones asociado a dicho dispositivo. Si al PIC le llegan señales de dos o más dispositivos no enmascarados (un 0 en el bit correspondiente) a la vez, el controlador determina la prioridad, definible por el usuario. El dispositivo al que no se atiende, espera turno en un "área de recepción"; cuando es atendido, el PIC lo saca de esta área de espera y lo lleva a la de trabajo.

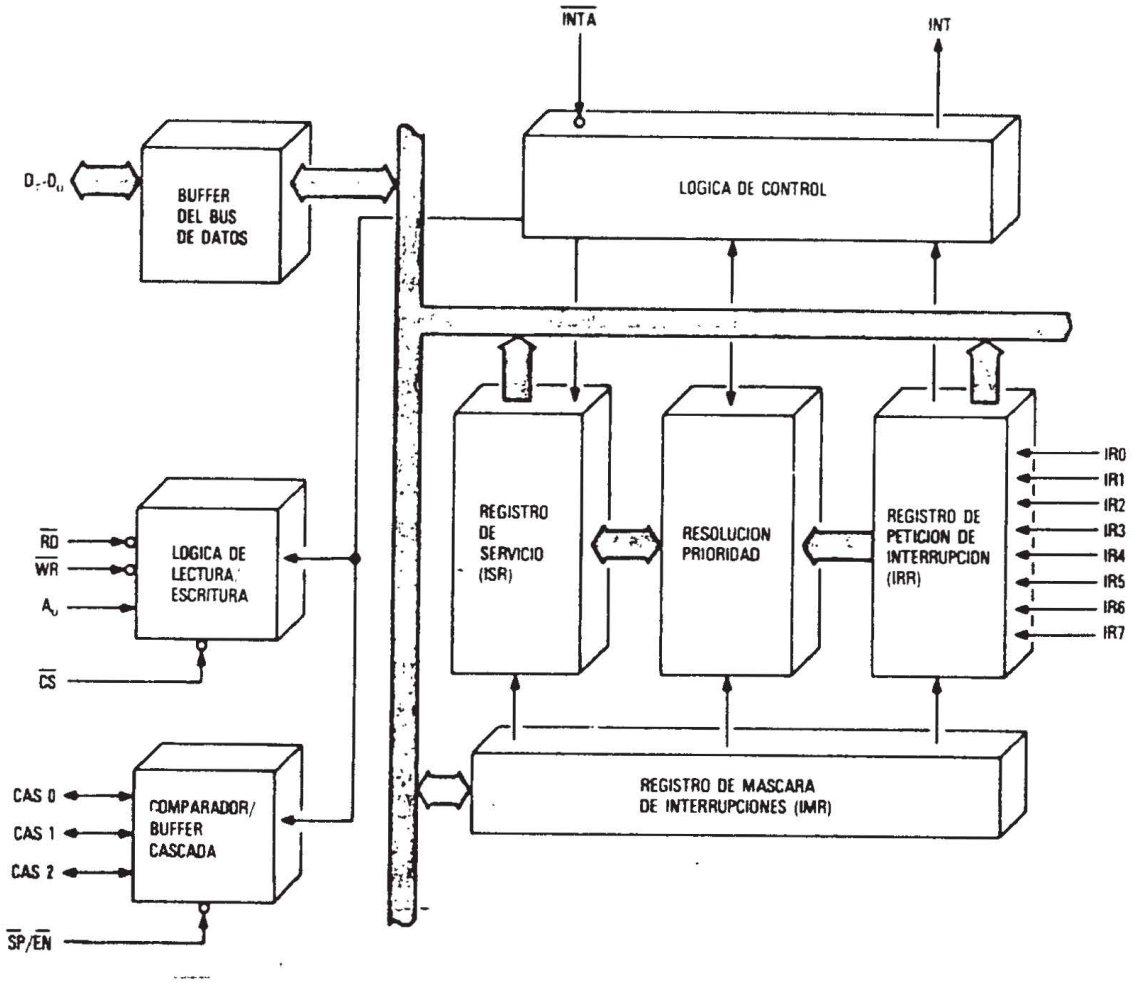


Figura 5.13
Estructura interna del PIC 8259

La CPU recibe la demanda de interrupción a través de la línea INT REQ y devuelve la de reconocimiento por la línea INT ACK. El 8259 envía 1 byte que informa a la CPU qué tipo de interrupción debe ejecutarse, en base a una tabla de interrupciones cargada en la memoria principal, y que es totalmente programable por el usuario.

5.5.2.2 CONTROLADOR PROGRAMABLE DE DMA, INTEL 8237

El controlador programable de DMA 8237 se utiliza en las transferencias por DMA. Puede dar servicio a un total de 4 dispositivos a la vez, a cada uno de los cuáles se le asigna un *canal de DMA* (*DMA channel*). Interconectando varios circuitos 8237 se pueden soportar todos los canales que se desee. Su esquema de estructura interna se muestra en la Figura 5.14.

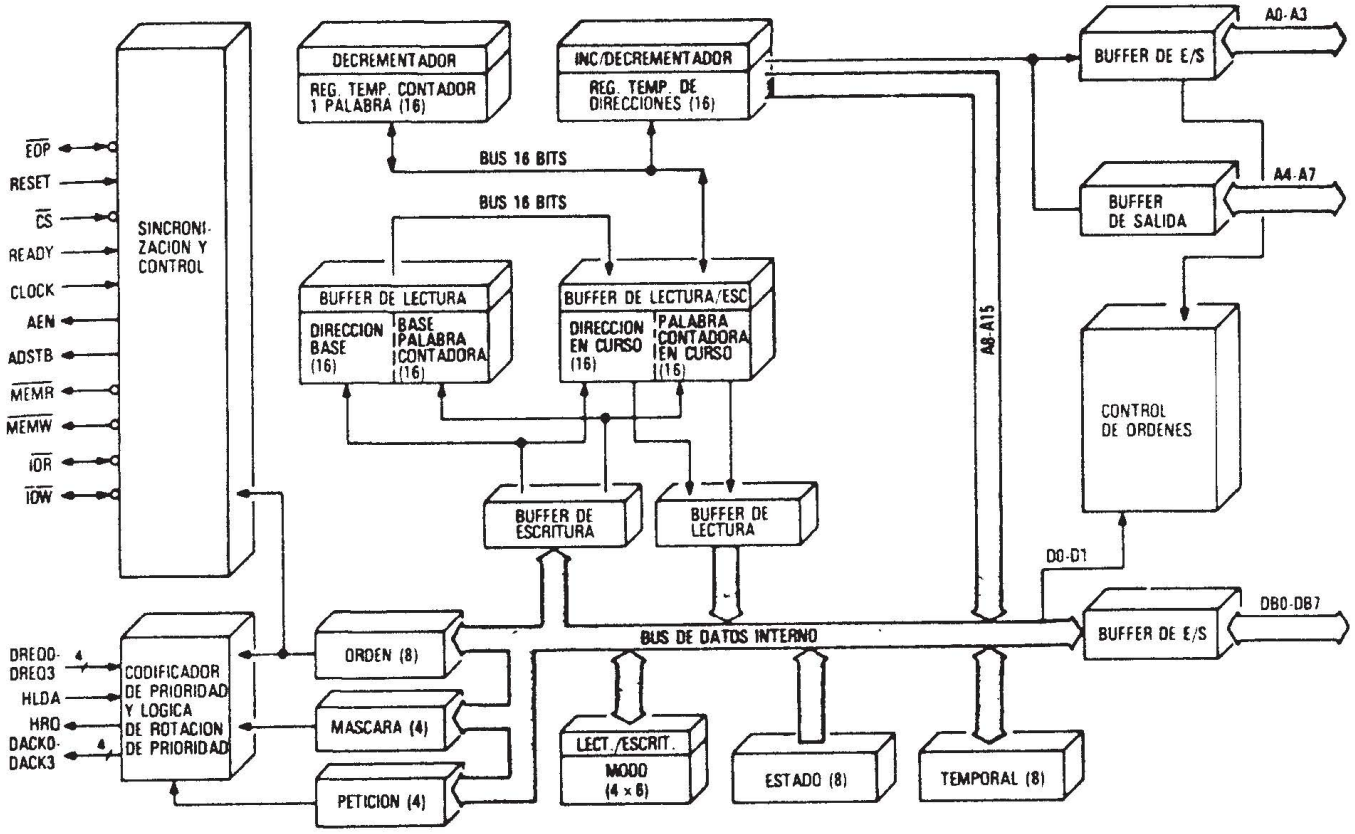


Figura 5.14

Estructura interna del controlador programable de DMA 8237

Este controlador posee registros encargados de almacenar la información sobre las direcciones fuente y destino, contadores y máscaras, órdenes y estado. Tiene disponibles muchas opciones, incluyendo temporización, esquemas de prioridades, posición, tamaño y tipo de transferencia. Estas se pueden hacer entre puertos de E/S y/o bloques de memoria.

5.5.3 CONTROLADORES DE DISPOSITIVOS

Los controladores de dispositivos hacen de interfase inteligente entre el IOP y los dispositivos periféricos externos. Según sean éstos, los controladores de dispositivos pueden ser de propósito especial o de propósito general. Los primeros son los que permiten interacciones con las principales unidades externas del ordenador: pantalla (CRT), teclado y discos. Los segundos son los que permiten la conexión del ordenador con cualquier dispositivo externo, sin especificar, destacando entre ellos la impresora (E/S paralelo) y el modem de comunicaciones (E/S serie).

5.5.3.1 CONTROLADORES DE PROPOSITO ESPECIAL

El *controlador programable de CRT* (*Programmable* CRT *Controller*), PCRTC 8275, proporciona la sincronización y el control de la visualización de texto sobre la pantalla. El texto se almacena en una sección de la memoria principal reservada a este efecto, donde la CPU coloca el código ASCII binario de los caracteres del texto; el PCRTC genera las señales de video necesarias para visualizar los símbolos correspondientes sobre la pantalla. El usuario programa en el 8275 el número de caracteres por fila (generalmente un máximo de 80) y el número de filas (generalmente un máximo de 64) en la pantalla durante la inicialización del dispositivo.

El *controlador programable de teclado* (*Programmable Keyboard Controller*), PKC 8278, proporciona el seguimiento de las acciones que se ejecutan sobre el teclado. El controlador muestrea el teclado y almacena la información en una pequeña memoria intermedia (generalmente 15 pulsaciones). Posteriormente envía una interrupción a la CPU, que traduce el significado de la pulsación mediante una tabla almacenada a tal fin en la memoria principal. De esta forma una pulsación de tecla queda preparada para ser utilizada por la CPU en cualquier programa.

El *controlador de discos flexibles* (*Floppy Disk Controller*), FDC 8272, provee los circuitos de control para hacer de interfase de hasta 4 unidades de disco, pudiendo trabajar solo o junto con un controlador de DMA. Los datos se organizan sobre los discos flexibles en círculos concéntricos llamados *pistas* (*tracks*) y cada pista se divide en cierto número de *sectores* (*sectors*). Hay una cabeza lectora/escritora que se mueve sobre el disco, que

gira a unas 300 rpm, seleccionándose las pistas por el movimiento paso a paso de la cabeza a lo largo del eje radial. Una vez seleccionada la pista, se puede transferir un sector desde el disco a la memoria principal. El FDC 8272 dispone de órdenes de movimiento de la cabeza de una unidad determinada a una cierta pista, de transferencia de sectores de datos entre la unidad seleccionado y el ordenador, búsqueda de sectores dependiendo de determinados formatos, formato de discos y definición de algunos parámetros se sincronización.

5.5.3.2 CONTROLADOR PROGRAMABLE PARALELO DE INTERFASE, INTEL 8255

El *controlador programable paralelo de interfase* (*Programmable Paralell Interface Controller*), PPIC 8255, sirve de ayuda en la conexión al ordenador de dispositivos de E/S que operan con bytes o palabras completos. La transmisión en paralelo es útil en todas aquellas aplicaciones que requieran una transmisión a gran velocidad y utilicen dispositivos no demasiado alejados del ordenador. Las palabras se envían a la velocidad que permita el programa y no están sometidas a ningún tipo de sincronización. La velocidad máxima viene limitada por aquélla con que el sistema pueda sacar los datos, por lo que se pueden obtener altas prestaciones utilizando una transferencia de DMA. El esquema de este controlador se muestra en la **Figura 5.15.**

El 8255 tiene 24 líneas de E/S, para las cuáles el usuario puede definir diferentes configuraciones, que le son transmitidas a través de un puerto E/S especial. Estas órdenes de configuración afectan al agrupamiento de los bits, la dirección de los puertos y las asignaciones de control y estado.

5.5.3.3 CONTROLADOR PROGRAMABLE SERIE DE INTERFASE, INTEL 8251

El *controlador programable serie de interfase* (*Programmable Serial Interface Controller*), PSIC 8251, sirve de ayuda en la conexión al ordenador de dispositivos que intercambian información en serie, es decir, bit a bit. La transmisión en serie se prefiere sobre la transmisión en paralelo en aquellos casos en los que la información debe transmitirse a grandes distancias. Esto se debe a varias razones:

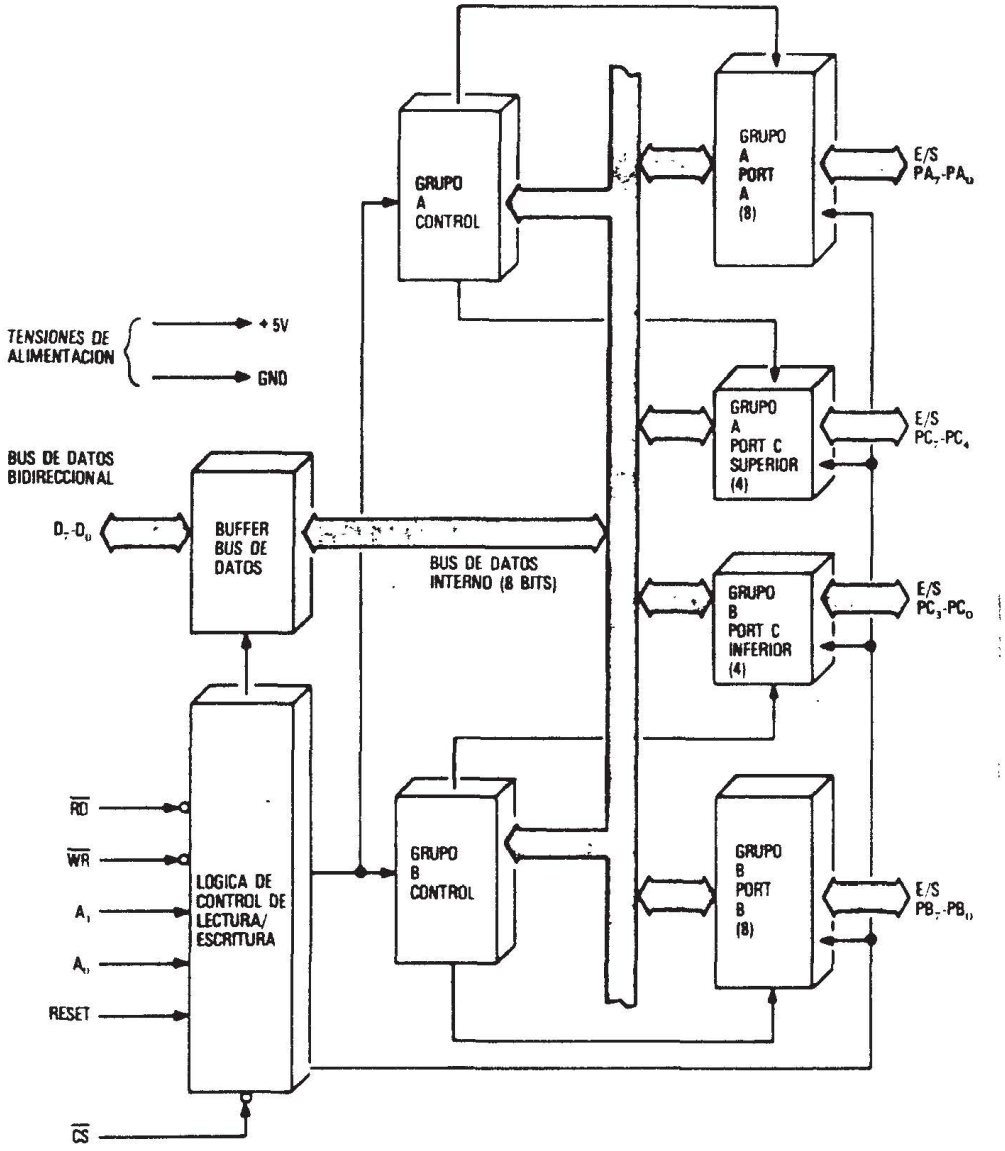


Figura 5.15

Estructura interna del PPIC 8255

* El enviar 1 bit cada vez requiere menos cables (2 ó 3, en vez de 8 ó 9)
* Las líneas en serie transmiten a velocidades menores, provocando menos errores debido al ruido
* Las transmisiones en serie son fácilmente codificables y decodificables para su envío por líneas telefónicas

Como el ordenador trabaja internamente con la información paralelo, se necesita un controlador serie de interfase para pasar de paralelo a serie. Un dispositivo de este tipo recibe el nombre de *receptor/transmisor asíncrono*

universal (*Universal Asinchronous Receiver/Transmitter*). El 8251 es en realidad un *receptor/transmisor síncrono/asíncrono universal* (*Universal Synchronous/Asynchronous Receiver/Transmitter*) y su diagrama de bloques se **muestra en la Figura 5.16.**

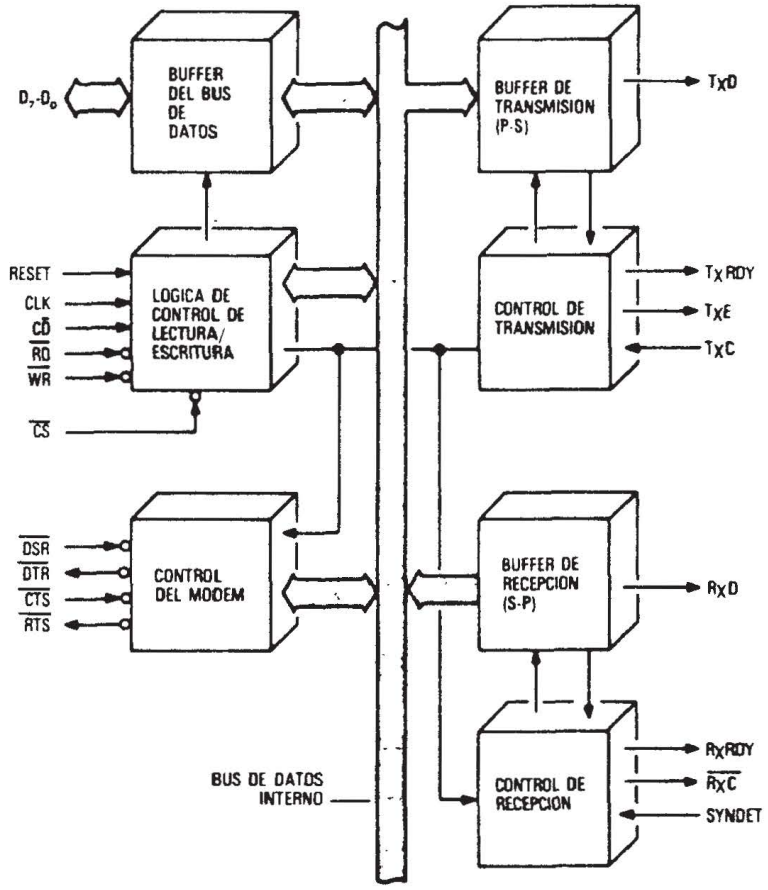


Figura 5.16

Estructura interna del PSIC 8251

El PSIC 8251 puede operar en modo síncrono o asíncrono. En ambos modos, los bits individuales se envían a una cierta velocidad, definiéndose ésta en *baudios* (*bauds*), o bits/s.

En *modo asíncrono*, cada byte se trata por separado. Cuando hay un byte preparado para la transmisión, se envían 1 ó 2 bits indicando que los bits siguientes pertenecen a un byte de información. Estos bits reciben el nombre de "bits de inicio". Cuando se han enviado todos los bits del byte normalmente, se acaba la transmisión con unos "bits de final", por lo que se

suelen necesitar 10 bits para enviar un carácter de esta manera.

En *modo síncrono*, todo un bloque de bytes se envía como una unidad. Hay un formato de inicio especial, pero no se envían bits adicionales con cada byte. Este modo es más rápido, pero necesita más programación, por lo que el trabajo adicional que produce sólo lo hace eficaz cuando hay que enviar grandes cantidades de información.

El usuario puede seleccionar uno de los modos de transmisión en un programa. Si elige transmisión asíncrona (la más utilizada en comunicaciones telefónicas), puede también definir la velocidad de transmisión y el formato de los caracteres a enviar, que incluye el número de bits de inicio, el número de bits de final y un control de paridad.

BIBLIOGRAFIA

Hayes, J.P.; *"Computer architecture and organization"*, McGraw-Hill, New York (1988).

Khambata, A.J.; *"Microprocesadores/microcomputadores. Arquitectura, software y sistemas"*, Gustavo Gili, Barcelona (1987).

Morgan, C.L. y Waite, M.; *"Introducción al microprocesador 8086/8088 (16 bit)"*, McGraw-Hill, México (1984).

Osborne, A.; *"Introducción a las microcomputadoras"*, Vol. 1, 2ª ed., Osborne/McGraw-Hill, Madrid (1986).

Stone, H.S.; *"Microcomputer interfacing"*, Addison-Wesley, Reading (1983).

CAPITULO 6

ALMACENAMIENTO MAGNETICO

6.1 PRINCIPIOS DEL ALMACENAMIENTO MASIVO

6.2 LAS CINTAS MAGNETICAS

- 6.2.1 TECNICAS DE GRABACION EN CINTA
- 6.2.2 FORMATO DE LA INFORMACION EN CINTA

6.3 LOS DISCOS MAGNETICOS

- 6.3.1 TIPOS DE DISCOS
  - 6.3.1.1 DISCOS FLEXIBLES
  - 6.3.1.2 DISCOS DUROS
- 6.3.2 TECNICAS DE GRABACION EN DISCO
- 6.3.3 FORMATO DE LA INFORMACION EN DISCO
- 6.3.4 ESTRUCTURACION LOGICA DE UN DISCO
  - 6.3.4.1 AREAS DEL SISTEMA OPERATIVO
  - 6.3.4.2 PARTICIONES DE UN DISCO DURO

6.4 DETECCION DE ERRORES

6.1 PRINCIPIOS DEL ALMACENAMIENTO MASIVO

Al aumentar la necesidad de almacenar grandes cantidades de información, resulta poco práctico y económico emplear para ello el sistema de memoria principal o memoria interna, de gran velocidad. Por este motivo se utilizan las distintas opciones de almacenamiento externo o sistema de memoria secundaria o auxiliar para guardar información de forma masiva, a pesar de que el tiempo de acceso de tales dispositivos sea considerablemente mayor, como ya se ha estudiado anteriormente. Los dispositivos de almacenamiento a gran escala pueden considerarse como sistemas independientes, aunque están conectados directamente al ordenador. El único vínculo entre ambos suele ser una configuración lógica, ya que físicamente ambos sistemas son muy diferentes.

La mayoría de los dispositivos de almacenamiento masivo tienen muchos elementos en común y pueden considerarse desde el punto de vista esquematizado en la **Figura 6.1**. El *soporte de almacenamiento* (*storage support*), que suele ser una superficie magnética, conserva la información, que es depositada en él por *cabezas de escritura* (*write heads*). La conversión de la representación eléctrica de la información del ordenador a una forma capaz de activar el soporte de almacenamiento corresponde a dichas cabezas, mientras que las *cabezas de lectura* (*read heads*) convierten la información almacenada de nuevo a la forma eléctrica. Un *accionador* (*drive*) proporciona la fuerza motriz necesaria para mover mecánicamente el soporte y un *controlador* (*controller*) ajusta el formato de los datos, los cambia de una forma a otra, detecta errores y mantiene constante la velocidad de arrastre.

Aunque existen algunos dispositivos de memoria masiva que utilizan técnicas no magnéticas (dispositivos acoplados por carga), la inmensa mayoría de los sistemas dependen de alguna forma del magnetismo, a expensas del tiempo de acceso. Asimismo, casi todos ellos emplean una técnica común de grabación/reproducción, que se introduce seguidamente.

Un material magnético que no ha sido imantado previamente se puede considerar compuesto por millones de imanes diminutos distribuidos aleatoriamente por todo el material, tal como muestra la **Figura 6.2 (a)**, lo que hace que los campos magnéticos se anulen entre sí. Por el contrario, si se aplica al material un campo magnético externo, los pequeños imanes se

alinean con el campo y el material se "magnetiza". La dirección de la magnetización [**Figura 6.2 (b) y (c)**] depende de la del campo externo, que está físicamente diseñado para que sólo pueda tener un sentido u otro, es decir, tiene naturaleza binaria.

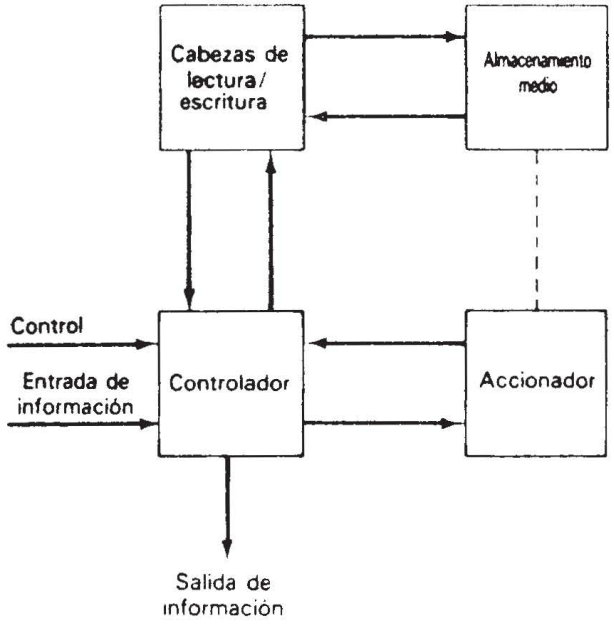


Figura 6.1
Dispositivo general de memoria masiva

La información puede grabarse (escribirse) fácilmente en los soportes magnéticos controlando el sentido del campo magnético externo. Este se genera haciendo que circule corriente por un conductor; su intensidad depende de la cantidad de corriente que circula y su dirección depende del sentido de circulación de la corriente. Además, si se arrollan varias vueltas de conductor en torno a un imán circular, el campo magnético debido al flujo de corriente en el conductor se transfiere al imán. En este caso, la intensidad del campo magnético inducido está determinada por la cantidad de corriente que circula por la bobina así construida y su dirección depende del sentido de circulación de la corriente eléctrica.

Si, por otro lado, se corta la trayectoria magnética abriendo un hueco (entrehierro) en el núcleo circular, el campo se conserva cuando desaparece la corriente que lo causó. Sin embargo, la presencia del material magnético

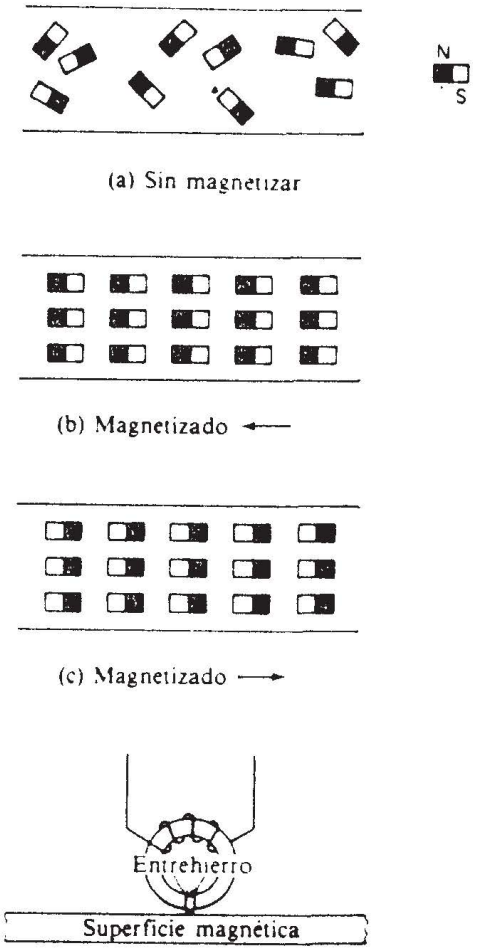


Figura 6.2
Registro magnético de información

cerca del entrehierro completa la trayectoria magnética y aquél se magnetiza igual que la partícula magnética más cercana al entrehierro. La eliminación de la corriente eléctrica hace desaparecer el campo magnético en el núcleo, pero el material magnético que hay cerca de él queda magnetizado y retiene la información [Figura 6.2 (d)]. La configuración física del núcleo, que se ha denominado "cabeza", varía mucho de unos dispositivos a otros, pero siempre tiene las propiedades que se han descrito aquí.

En este capítulo se describirán dos dispositivos de almacenamiento magnético masivo: las cintas y los discos. Aunque también se pueden utilizar otros medios, los citados son los más ampliamente utilizados y, aunque la

velocidad de transferencia de información y la densidad de almacenamiento pueden diferir considerablemente de unos sistemas a otros, los principios de funcionamiento son los mismos para todos los dispositivos de registro magnético.

6.2 LAS CINTAS MAGNETICAS

La *cinta magnética* (*magnetic tape*) es un tipo de memoria secundaria que adopta la forma de una cinta continua muy larga de material plástico, recubierta de un óxido magnético sobre el cual puede registrarse la información como una serie de puntos magnetizados. Las dimensiones generales de la cinta varían de unos sistemas a otros, pero cabe destacar dos formatos típicos:

* *Casetes* (*cassettes*): cinta de 0,15 in (3,8 mm) de ancho y unos 300 pies (91,5 m) de largo, que puede almacenar hasta 20 bytes/in (51 bytes/cm), utilizada en *grabadoras de casetes* (*cassette recorders*) de uso doméstico, y que giran a una velocidad de 1,87 in/s (4,75 cm/s).

* *Bobinas* (*reels*): cinta de 0,5 in (12,7 mm) de ancho y unos 2.400 pies (731,5 m) de largo, que permite densidades de almacenamiento de hasta 556 bytes/in (1.412 bytes/cm), utilizada en *unidades grabadoras* (*tape recorders*) de complejo diseño y que pueden girar a velocidades de hasta 1.875 in/s (4,76 m/s).

La mayoría de los dispositivos de grabación/reproducción utilizan una o varias cabezas fijas, frente a las cuáles se hace discurrir la cinta. El sistema de arrastre de la cinta va desde las técnicas más sencillas utilizadas en las grabadoras de casetes a los sistemas complejos dotados de control de arrastre y velocidad que pueden encontrarse en las costosas unidades grabadoras.

En las grabadoras de casetes la cinta pasa simplemente por guías que la mantienen en tensión contra las cabezas; se utiliza un sistema de correa o un motor de accionamiento directo para hacer girar el carrete de recogida. Lo más normal es que estas unidades no sean de velocidad controlada, por lo cual deben emplearse métodos de grabación con cierta tolerancia frente a las variaciones de velocidad. La información digital suele ser convertida en tonos de audiofrecuencia, en cuyo caso se reduce considerablemente la

velocidad de transferencia de datos.

Las complejas unidades grabadoras almacenan los datos digitales directamente, lo que hace necesario el uso de unidades de grabación más complicadas, como la mostrada en el esquema de la **Figura 6.3.**

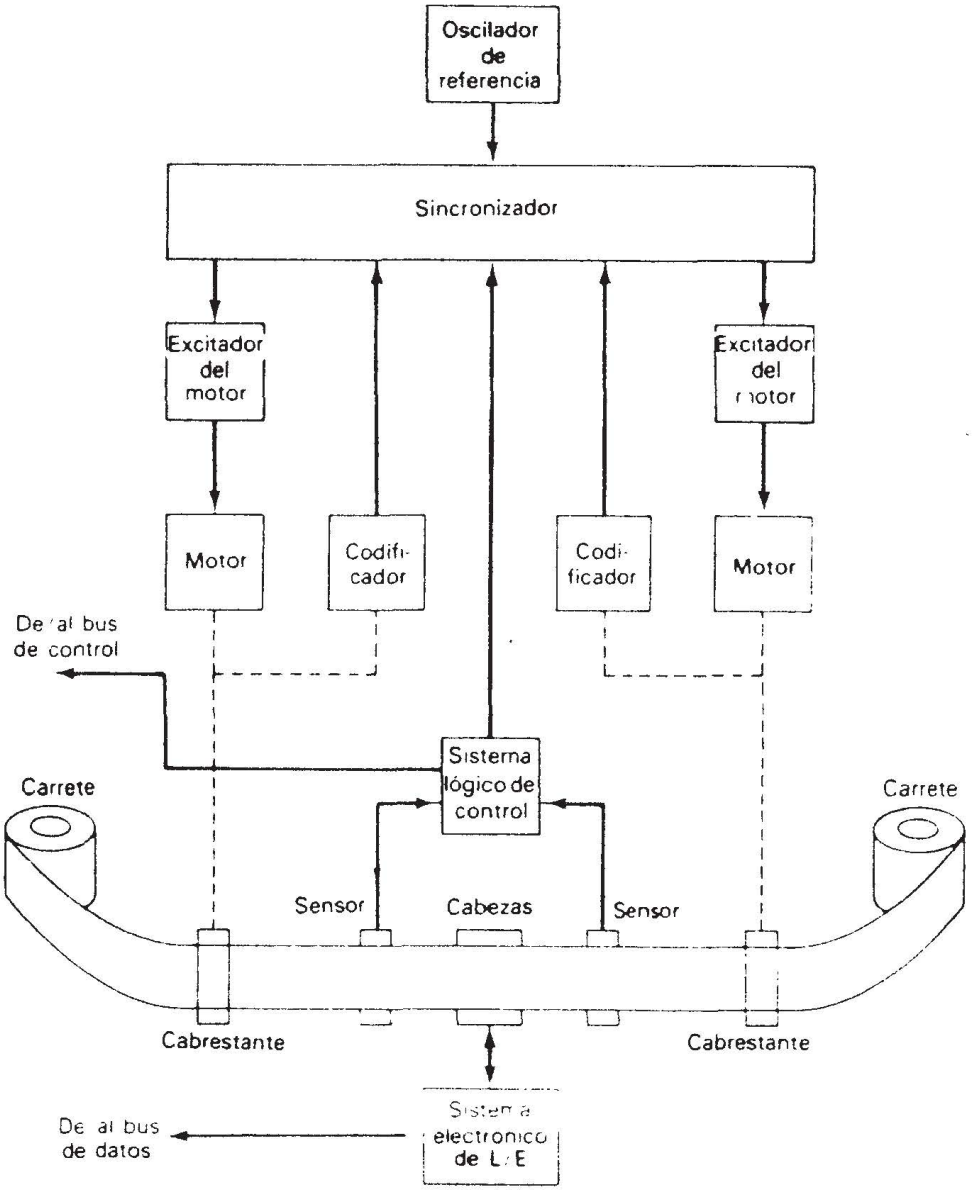


Figura 6.3

Unidad grabadora de cinta

El arrastre de la cinta utiliza cuatro motores separados: dos para accionar los carretes y otros dos para los cabestrantes, o tornos verticales,

con funcionamiento bidireccional a dos velocidades, tanto en la lectura como en la escritura. Los cabestrantes, en combinación con unos rodillos de presión, se ocupan de que la cinta tenga una tensión y velocidad constante a su paso frente a las *cabezas de lectura/escritura* (*read/write heads*) y están accionados por un servomotor que asegura el control preciso de la velocidad. De esta forma se reducen al mínimo las variaciones de la velocidad de la cinta, permitiendo una grabación y reproducción precisas de la información digital. El principio y el final de la cinta son detectados por sensores que arrancan y detienen los cabestrantes.

6.2.1 TECNICAS DE GRABACION EN CINTA

Las técnicas de grabación en cintas magnéticas varían en función de su capacidad de almacenamiento. Si se utilizan cintas de baja densidad (casetes), el problema principal es eliminar la sensibilidad de la lectura/escritura frente a las posibles variaciones de velocidad del dispositivo, principalmente si se utilizan las cintas en aparatos distintos. Cuando se usan cintas de alta densidad (bobinas), este problema no existe, ya que la densidad de los datos es suficiente para permitir una sincronización con la frecuencia del flujo de datos. La técnica utilizada para la grabación de tono de audiofrecuencia en casetes suele ser la KCS, mientras que para la grabación de señales digitales directas existen varios métodos, el más utilizado de los cuáles es el de grabación en modulación de frecuencia.

La *técnica KCS* (*Kansas City Standard*) utiliza ondas cuadradas de 4 ciclos de 1.200 Hz para representar el 0 lógico y 8 ciclos de 2.400 Hz para representar el 1 lógico, tal como muestra la **Figura 6.4**.

Obsérvese que la duración del 0 y del 1 son iguales, y que ciertas transiciones aparecen en ambas ondas. Mediante esta codificación, la velocidad de transmisión es de:

$$\frac{2.400}{8}\ \frac{(\text{ciclos/s})}{(\text{ciclos/bit})} = \frac{1.200}{4} = 300\ \frac{\text{bits}}{\text{s}} \qquad [6.1]$$

Las transiciones comunes al 0 binario y al 1 binario se usan para sincronizar un oscilador local a los datos de entrada cuando éstos son recuperados de la grabación. Como se muestra en la **Figura 6.4** (c), las

señales de reloj se producen tanto durante las transiciones a 1 como a 0, por lo que el detector de esta señal sólo ha de buscar la existencia de una transición, ignorando la polaridad de la misma. Si una transición ocurre entre dos señales de reloj, el detector supone que está codificado un 1, mientras que en caso contrario, supone que está codificado un 0.

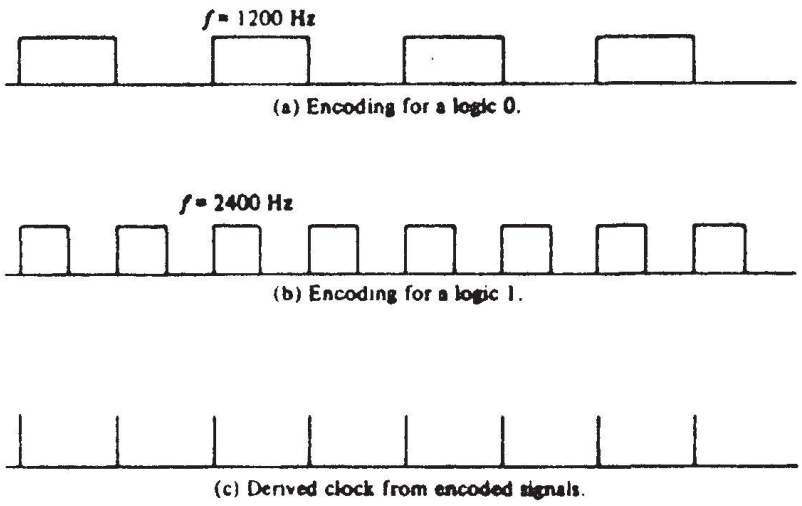


Figura 6.4
Grabación en cinta mediante la técnica KCS

La *técnica de modulación de frecuencia* (*Frecuency Modulation*) es esencialmente del mismo tipo que la anterior, pero no se graban audiofrecuencias, sino transiciones del flujo magnético sobre la cinta, es decir, señales digitales directas. La forma de codificar datos como transiciones del flujo magnético se muestra en la **Figura 6.5**, donde cada pulso corresponde a una variación del flujo magnético sobre el soporte.

El diagrama de la **Figura 6.5** muestra la grabación de un bit cada 4 μs, es decir, una velocidad de transmisión de 250 kbits/s. Al comienzo de cada intervalo de 4 μs, se graba una transición del flujo magnético, que se utiliza como señal de reloj. A la mitad del intervalo se graba el dato correspondiente a este período, bien registrando una transición magnética si se trata de un 1 lógico o bien no registrando nada si se trata de un 0 lógico. Las transiciones de reloj garantizan que al menos se produce una transición en cada intervalo entre dos bits, de tal forma que el receptor

pueda recuperar la señal de reloj a partir de los datos grabados.

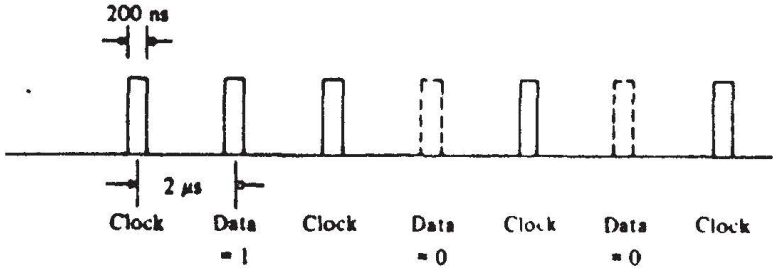


Figura 6.5
Grabación en cinta mediante la técnica FM

Obsérvese que la eficacia de densidad de grabación sólo es del 50 %, ya que sólo la mitad de los bits grabados corresponden a bits de información. Por este motivo a veces se denomina esta técnica de *simple densidad* (*single density*) o también *codificación Manchester* (*Manchester encoding*) para distinguirla de otras técnicas que permiten una mayor densidad de almacenamiento.

6.2.2 FORMATO DE LA INFORMACION EN CINTA

Recuérdese que el hecho más importante relativo a la cinta mágnética es que se trata de un medio de almacenamiento secuencial, por lo que la información ha de grabarse en ella como una secuencia de registros físicos. Cabe destacar, no obstante, que no existe un formato de grabación normalizado, por lo que los registris físicos de una cinta pueden ser todos de la misma longitud o de longitud diferente. Como tampoco hay restricciones en la longitud de los registros físicos, éstos pueden tener la longitud de un carácter o ser tan largos como permita la longitud de la cinta.

De forma general se puede indicar que una cinta se organiza de manera que cada registro o segmento contiene una sección de dirección e identificación, una sección dedicada a los datos propiamente dichos, una parte reservada al método utilizado para detectar errores y una separación entre los registros. La fiabilidad de la grabación se favorece operando con registros cortos (lo que disminuye su capacidad de almacenamiento) y grabando

registros redundantes, es decir, registrando dos veces el mismo dato, tal como se muestra en la **Figura 6.6.**

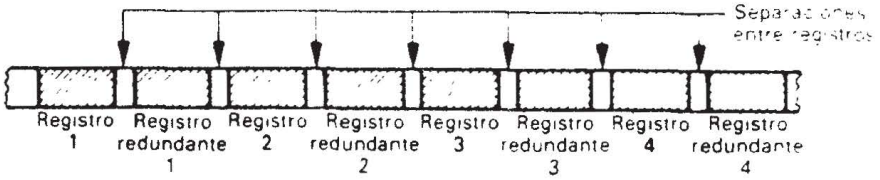


Figura 6.6
Formato genérico de una cinta

Así, cada vez que durante una operación de escritura se detecta un error, el ordenador vuelve automáticamente a escribir el bloque en cuestión; si el error se produce durante la lectura, intentará leer nuevamente el bloque. En los casos en que el error sea debido a alguna deficiencia transitoria, tal proceso puede tener éxito. Sin embargo, los errores persistentes suelen ser señal de algún defecto en la cinta o de alguna avería en la unidad de transporte.

6.3 LOS DISCOS MAGNETICOS

Los discos magnéticos constituyen el tipo de memoria de almacenamiento masivo más ampliamente utilizado en la actualidad, debido a sus altas prestaciones en cuanto a tiempo de acceso, fiabilidad de grabación y costes cada vez más reducidos. Se denomina *disco* (*disk*) al dispositivo de memoria que consta de una o varias placas circulares superpuestas, recubiertas por ambas *caras* (*sides*) con algún material magnetizable. Cada superficie dispone de un cierto número de coronas circulares concéntricas llamadas *pistas* (*tracks*), sobre las cuáles se almacena la información mediante *cabezas de lectura/escritura* (*read/write heads*).

El acceso de las cabezas de lectura/escritura a un punto cualquiera del disco se consigue mediante la combinación de dos movimientos: rotación del disco y traslación de la cabeza (**Figura 6.7**).

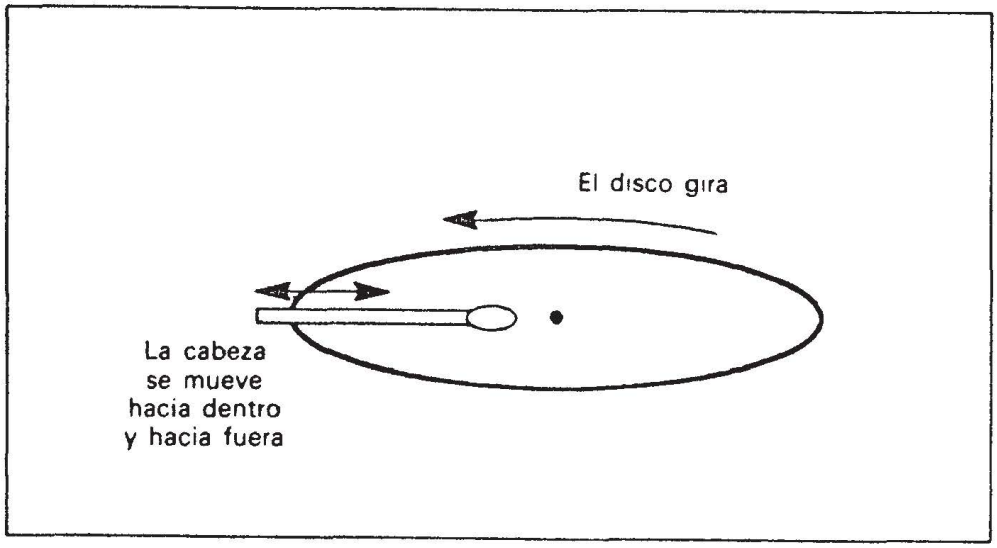


Figura 6.7
Acceso de las cabezas a un disco

La *rotación del disco* se consigue instalándolo sobre un eje que es accionado por un motor; este movimiento permite el acceso a cualquier punto de una pista, siendo su velocidad la determinante del tiempo de acceso a la información.

La *traslación de la cabeza* es el desplazamiento lineal de la misma sobre la dirección radial del disco, y permite acceder a cualquiera de sus pistas. Este movimiento se logra mediante un actuador lineal de alta precisión, que accionado mediante una corriente eléctrica, proporciona un desplazamiento muy preciso, proporcional a la corriente eléctrica aplicada.

Obsérvese que, combinado los dos movimientos indicados, se puede acceder a cualquier punto del disco muy rápidamente. Se puede decir, pues, que un disco magnético es una memoria de acceso aleatorio (RAM), ya que es posible acceder a cualquier parte de los datos grabados, directa y aleatoriamente, sin tener que pasar previamente por otras zonas de almacenamiento de información, como sucede en el caso de las cintas magnéticas.

Las cabezas no suelen estar en contacto físico con la superficie de grabación del disco en todo momento, con objeto de reducir al mínimo el desgaste de uno y otras, por lo que se suele utilizar un cojín de presión para llevar a cabo el contacto durante la operación de lectura/escritura, retrayéndose las cabezas durante los períodos de inactividad.

Por otro lado, también existen unidades de *disco de cabezas fijas* (*fixed-head disk*), lo que implica la necesidad de una cabeza por pista. Sin embargo, en los modernos ordenadores se usa cada vez menos esta tecnología, que permite un acceso más rápido a la información, ya que se están realizando grandes avances en el diseño de las unidades de *disco de cabezas móviles* (*movable-head disk*), con tiempos cada vez menores y a más bajo coste.

6.3.1 TIPOS DE DISCOS

Según el material de que están construidos, los discos pueden ser flexibles o duros. Debido a que ambos tipos presentan características físicas y lógicas altamente diferenciadas, es conveniente su estudio por separado.

6.3.1.1 DISCOS FLEXIBLES

Los *discos flexibles* (*floppy disks, diskettes*), están fabricados en una base de material plástico cuyas caras se hallan recubiertas de la película magnética que permite la grabación de información. Van alojados en fundas cuadradas especiales que protegen la superficie de registro, en las que hay unas aberturas para su montaje sobre el eje accionador (que gira a una velocidad de 360 rpm) y para el acceso de las cabezas de lectura/escritura a su superficie. En un lado poseen también una ranura de protección de escritura, que permite protegerlos contra borrados accidentales de la información.

El formato externo de un disco flexible puede ser muy variado. Hasta hace pocos años se utilizaba aún un disco de 8 pulgadas, pero se ha ido imponiendo fuertemente el formato de 5 1/4 pulgadas y, más recientemente, el de 3 1/2 pulgadas.

El *disco flexible de 5 1/4"* (**Figura 6.8**) tiene dos orificios circulares: uno es el central, dotado de un anillo de material de refuerzo para su perfecto ajuste sobre la unidad tractora; el otro está en el contorno del centro y proporciona un punto de referencia que define el comienzo de una pista. En la superficie interior de la funda o camisa hay un revestimiento de fieltro que permite al disco deslizarse suavemente y limpiarlo al mismo tiempo. La larga ranura oval es la abertura de acceso de las cabezas y en un lado se encuentra la ranura de protección de escritura, que si está tapada,

impide la escritura sobre el disco.

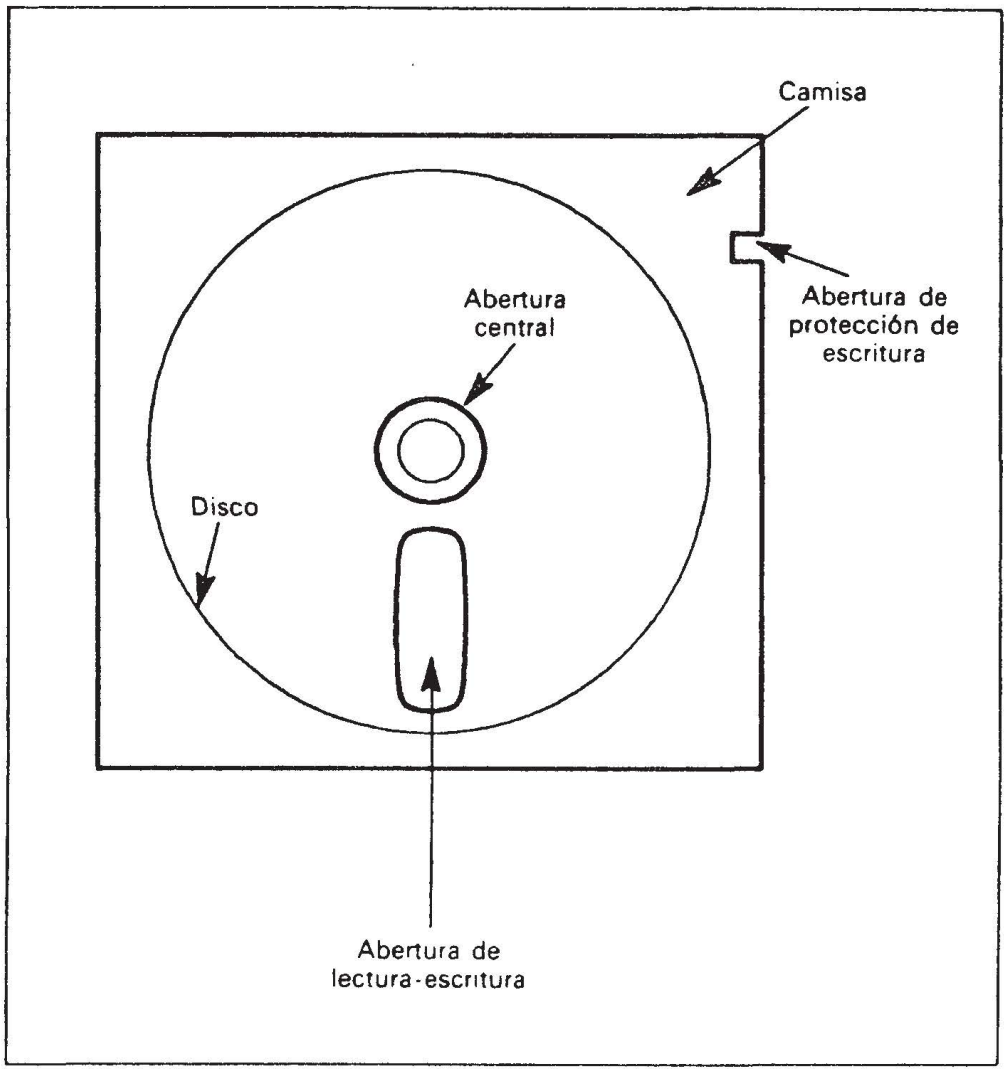


Figura 6.8
Disco flexible de 5 1/4"

Las pistas sólo cubren una pequeña zona de la superficie de grabación, aproximadamente unos 2 cm; no obstante, en este reducido espacio se suelen almacenar 40 u 80 pistas sobre cada cara, lo que da lugar a la existencia de discos de 48 *pistas/pulgada* (*Tracks Per Inch*) y de 96 tpi. El formato de 40 pistas/cara se denomina de *doble densidad* (*double density*), mientras que el de 80 pistas/cara se denomina de *cuádruple densidad* (*quadruple density*), generalmente llamados en el ámbito comercial de *alta densidad* (*high density*), aunque este término es confuso, ya que, como se verá más adelante, se debe aplicar a la capacidad de almacenamiento de la información.

Dentro de cada pista la información puede almacenarse físicamente con mayor o menor densidad, por lo que suele denominarse *densidad de registro* (*recording density*) o, más correctamente, *capacidad* (*capacity*), a la cantidad de información que puede ser almacenada en una pista. Los discos flexibles de 5 1/4" tienen una capacidad de almacenamiento normal de 6,25 kbytes/pista, aunque en la actualidad se fabrican discos de *alta capacidad* (*high capacity*), pudiendo contener 10,4 kbytes/pista, y que poseen una capa magnética especial en su superficie.

Generalmente se utilizan discos de 5 1/4" de doble cara, doble densidad y capacidad normal, que permiten almacenar 500 kbytes (40 *x* 6,25 *x* 2) de información, o discos de doble cara, cuádruple densidad y alta capacidad (llamados de "alta densidad"), que permiten almacenar 1.664 kbytes (80 *x* 10,4 *x* 2), pero sólo pueden ser manipulados en unidades lectoras especiales. Obsérvese que los datos de capacidad indicados son todos valores "brutos". Como se verá más adelante, al asignarle un formato a la información, la capacidad real de almacenamiento es considerablemente inferior a la indicada.

El *disco flexible de 3 1/2"*, o *microdisco* (**Figura 6.9**), es mucho más pequeño, manejable y seguro que el de 5 1/4". Aunque está encerrado en una caja de protección rígida, interiormente es del mismo material, plástico flexible, con una pieza metálica central. Exteriormente, la abertura de lectura/escritura está cubierta por un protector metálico deslizante movido por un muelle y la protección de escritura está formada por una lengüeta de plástico deslizante, en vez de por un corte. Su pequeño tamaño, caja de protección y mayor capacidad de almacenamiento lo ha convertido en el disco elegido para los ordenadores de nuevo diseño, pero las grandes inversiones que han rodeado a los discos de 5 1/4" están retrasando el uso masivo de este nuevo formato.

En este tipo de disco se ha adoptado un único formato de densidad, la densidad cuádruple de 80 pistas/cara, dispuestas en 135 tpi. No obstante, también se almacena la información con capacidad normal (6,25 kbytes/pista) y con alta capacidad. En este último caso se ha logrado una mayor capacidad que para los discos de 5 1/4", alcanzándose un valor de 12,5 kbytes/pista.

En estas condiciones, un microdisco de 3 1/2" de capacidad normal puede almacenar hasta 1.000 kbytes (80 *x* 6,25 *x* 2), mientras que uno de alta

capacidad permite almacenar hasta 2.000 kbytes (80 *x* 12,5 *x* 2), todos ellos valores en bruto.

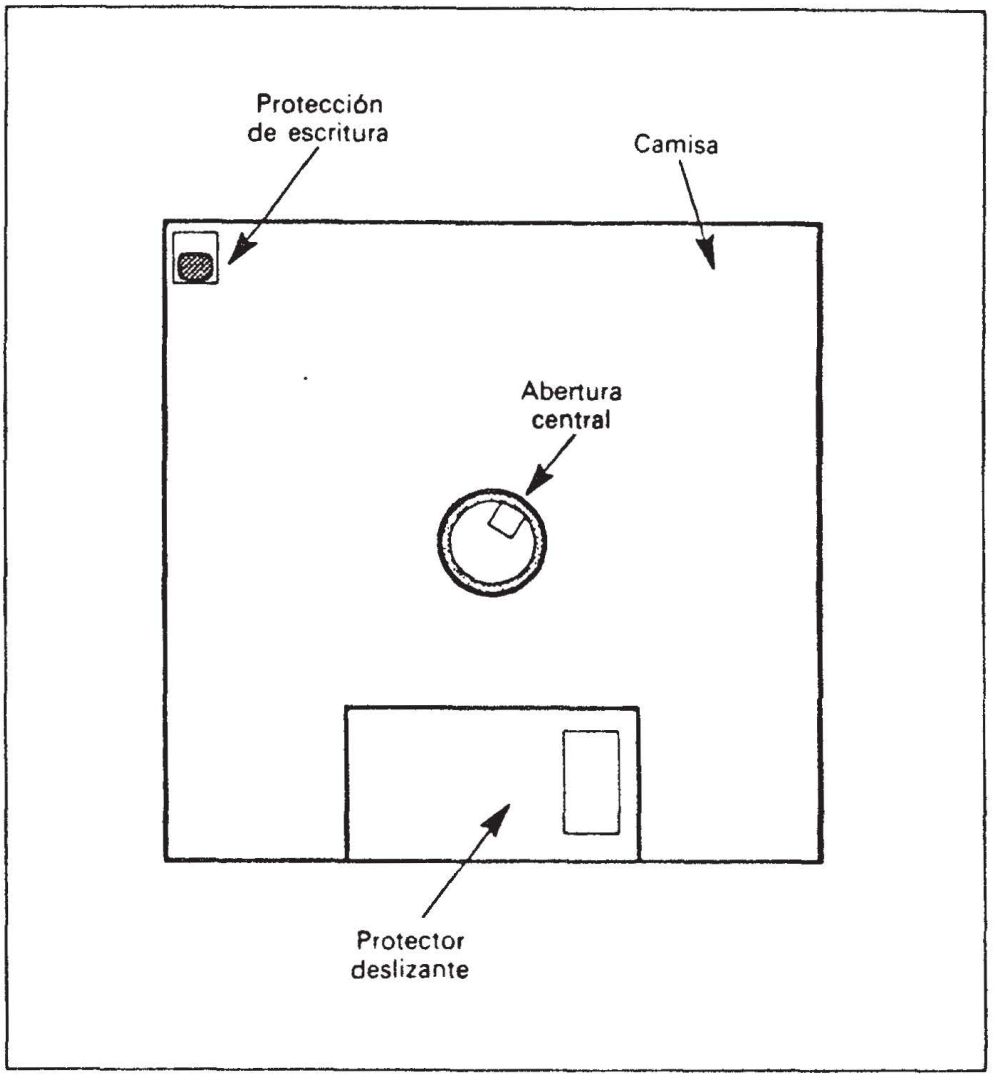


Figura 6.9
Microdisco de 3 1/2"

6.3.1.2 DISCOS DUROS

Los *discos duros* (*hard disks*) o *discos fijos* (*fixed disks*) toman su nombre del hecho de que la lámina base es rígida, fabricada con una aleación de aluminio y recubierta asimismo por una capa de material magnético. Normalmente están formados por una serie de estas láminas circulares apiladas, o platos, que giran alrededor de un eje común frente a las cabezas de lectura/escritura, soportadas por un brazo. Cada brazo forma parte de una

estructura de tipo peine de tal forma que aquél soporta al menos una pareja de cabezas, una por cada cara de cada plato. Si el brazo es fijo y contiene numerosos pares de cabezas se tiene un *disco de cabezas fijas* (*fixed-head disk*), tal como se muestra en la **Figura 6.10 (a)**; si el brazo es móvil y sólo contiene una pareja de cabezas se tiene un *disco de brazo móvil* (*movable-arm disk*), tal como se puede observar en la **Figura 6.10 (b)**. Estos últimos son los más utilizados en la actualidad.

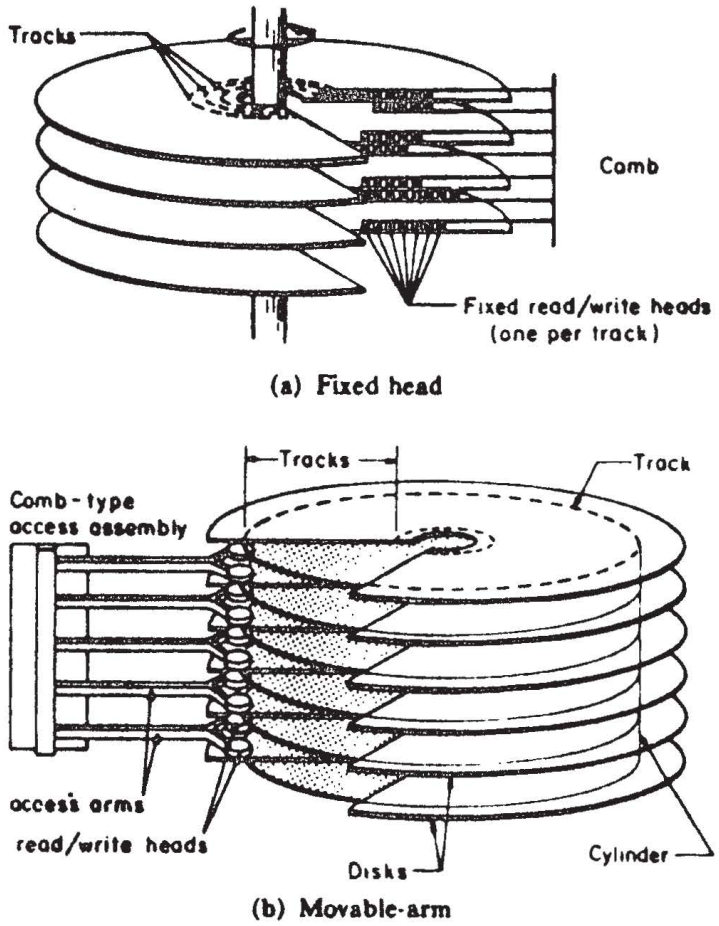


Figura 6.10
Esquema de disco duro

Al igual que sucede en los discos flexibles, la información se registra sobre las pistas de los platos. Cuando se trata de discos de brazo móvil, todas las cabezas se mueven juntas y se alinean en la misma pista de los diferentes platos. Se dice entonces que el número de pistas accesibles por todas las cabezas en cualquier posición forma un *cilindro* (*cylinder*).

Obsérvese que, según esta definición, el número de pistas por cara es equivalente al número de cilindros.

Dejando aparte los discos duros de los grandes sistemas de ordenadores, cuyas dimensiones físicas pueden ser considerables, en los modernos ordenadores personales se utilizan discos duros de tamaño similar a los discos flexibles (5 1/4" y 3 1/2"). Estos necesitan estar en una atmósfera que los proteja cuidadosamente del polvo y otros contaminantes, por lo que se montan en cajas herméticamente cerradas, que se instalan de forma fija dentro del ordenador. Muchas veces se conoce estas unidades como "discos Winchester", tomando su nombre del rifle Winchester 30-30, ya que los primeros modelos de estos discos tenían una capacidad para 30 Mbytes distribuidos en 30 pistas.

Ultimamente se están intentando popularizar los denominados *discos removibles* (*cartridge disks, removable disks*), que combinan las características de los discos duros (gran capacidad de almacenamiento) con las de los flexibles (facilidad de cambio). Así, estos discos se pueden extraer del ordenador y almacenar grandes cantidades de información, pero su alto coste aún no los ha hecho muy comerciales.

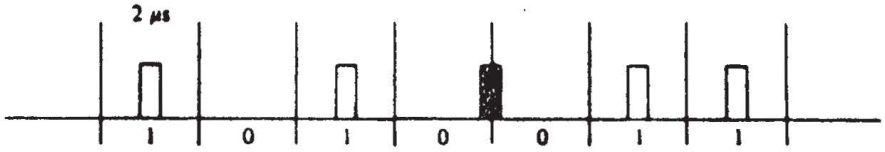
Existen muchas variedades de discos duros del tipo Winchester, distinguéndose principalmente por su velocidad, que oscila ente las 1.000 y 3.600 rpm (tiempos de acceso medios comprendidos entre unos 65 y 15 ms), y por su capacidad, que actualmente varía entre 10 Mbytes y varios cientos de Mbytes. No obstante, se utiliza mucho por su favorable relación capacidad/coste, el disco duro de 20 Mbytes y 65 ms de tiempo de acceso, por lo que a continuación se indican sus características más destacables.

El disco duro de 20 Mbytes de almacenamiento neto está formado por 2 platos, es decir, posee 4 caras, cada una de las cuales soporta 614 pistas, cuya capacidad bruta es de unos 11,8 kbytes/pista. Esto permite almacenar un total de unos 28,3 Mbytes de información bruta. Cuando se le da el formato apropiado al disco para que pueda almacenar la información adecuadamente, esta cantidad se reduce a un valor neto de 21,3 Mbytes. El valor que se le da al disco suele ser nominal, variando de 10 en 10 Mbytes, por lo que al disco duro analizado le corresponde un valor nominal de 20 Mbytes.

6.3.2 TECNICAS DE GRABACION EN DISCO

Como se ha indicado anteriormente, los discos son dispositivos de almacenamiento magnético de alta densidad. Esto significa que la densidad de la información es suficiente para permitir una sincronización con la frecuencia de datos, como sucedía en el almacenamiento digital en cintas magnéticas. Sin embargo, la técnica FM descrita para éstas sólo permite una eficacia del 50 % en la grabación de los datos, ya que sólo la mitad de los bits grabados corresponden a bits de información; el resto son bits de sincronismo.

Avanzando en este mismo concepto, puede diseñarse una técnica que **permita recuperar la señal del reloj, indispensable para la sincronización,** sin desperdiciar la mitad de los bits disponibles. Esta es la técnica más utilizada actualmente para la grabación en discos magnéticos y se denomina *técnica MFM* (**Modified-FM**) o *codificación de doble densidad* (*double-density encoding*), mostrándose su esquema de codificación en la **Figura 6.11.**



ENCODING STRUCTURE.

A 1 bit is encoded as a pulse in the middle of a bit window.

A 0 bit is encoded as the absence of a pulse in the middle of a bit window.

Insert a clock pulse at the transition of two windows if no pulse occurs in the middle of either window.

Figura 6.11

Grabación en disco mediante la técnica MFM

La idea básica es asignar una "ventana" por cada bit, cuya anchura es la mitad de la que corresponde a la codificación en simple densidad. Un 1 lógico se codifica como una transición del flujo en el centro de esa ventana y un 0 lógico se codifica como la ausencia de transición del flujo en la ventana. Sin embargo, hay que codificar también de alguna manera la señal de reloj, ya que en caso contrario no podría distinguirse la posición de los bits. Para resolver este problema se registra una transición de reloj entre

dos 0 adyacentes, es decir, se inserta un pulso correspondiente al reloj si no se produce ningún pulso en el centro de ambas ventanas.

Obsérvese que dos transmisiones consecutivas están separadas como mínimo 2 μs, que es el mismo espaciado que para la técnica de simple densidad. Sin embargo, ahora cada bit ocupa la mitad de espacio, porque sólo existe un bit de reloj cuando hay dos huecos consecutivos. Evidentemente, en este caso la unidad controladora del disco será más complicada, ya que tiene que reconocer señales de reloj que sólo aparecen ocasionalmente dentro de una serie de bits de datos.

6.3.3 FORMATO DE LA INFORMACION EN DISCO

La grabación de la información en un disco permite el acceso a un bloque específico de información sin necesidad de leer todos los bloques que le preceden, por lo que generalmente se conoce este acceso como *acceso aleatorio* (*random access*). Sin embargo, en este contexto la citada expresión es ligeramente diferente a la usada para la memoria de semiconductores, en cuyo caso significa que el acceso es a un elemento individual de información. Este hecho no se produce en los discos, por lo que algunos autores prefieren utilizar para éstos el término *acceso directo* (*direct access*).

Para acceder a la información contenida en un disco es necesario organizar sus pistas adecuadamente. Ello se logra dividiendo cada pista en unidades más pequeñas de almacenamiento denominadas *sectores* (*sectors*). Para poder reconocer estos sectores es necesario direccionarlos, lo que se puede llevar a cabo de dos formas: físicamente o lógicamente.

La *sectorización física* (*hard sectoring*) se lleva a cabo disponiendo una serie de orificios alrededor del centro del disco, espaciados a intervalos uniformes, junto con otro con espaciamiento diferente respecto a los primeros y que sirve de índice. Al girar el disco, un dispositivo óptico detecta los orificios y, por tanto, es capaz de determinar el sector que queda en cada momento bajo la cabeza de lectura/escritura. Esta técnica se muestra en la **Figura 6.12 (a)**.

La *sectorización lógica* (*soft sectoring*) utiliza un solo orificio como punto de referencia o índice físico, y la información sobre un sector se encuentra grabada en el propio sector, tal como muestra la **Figura 6.12 (b)**.

Es decir, cada sector contiene un campo de identificación, situado inmediatamente antes del campo de datos.

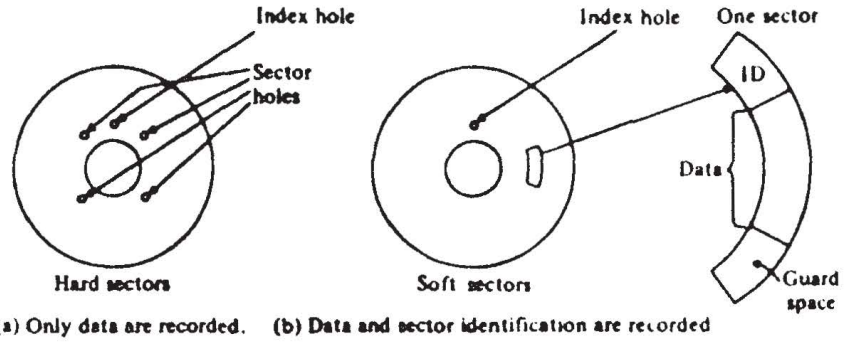


Figura 6.12
Sectorización de un disco

Para leer un bloque de datos, la cabeza se posiciona en la pista adecuada, donde comienza a leer toda la información de forma continua. Cuando la cabeza detecta un identificador con el número del sector buscado, el bloque de datos que le sigue es aceptado como la información buscada.

Para escribir un bloque de datos, la cabeza lee continuamente toda la información de una pista hasta que encuentra el número del sector buscado en un identificador. Entonces escribe el nuevo bloque de datos sobre uno ya existente, cuando éste pasa bajo la cabeza.

Durante una operación normal, los identificadores nunca son reescritos, aunque el controlador del disco tiene capacidad de grabar dichos identificadores sobre un disco en blanco, cuando éste se utiliza por primera vez. A esta operación se la denomina *formateado* (*formatting*).

El formato de sectorización lógica permite una menor capacidad del **disco que el de sectorización física, ya que el campo identificador ocupa espacio, que podría ser utilizado para los datos. Sin embargo, ofrece mucha** mayor versatilidad, ya que el usuario puede establecer el formato deseado, tanto respecto al tamaño del sector como al número de éstos. La **Figura 6.13** muestra un ejemplo de formato de un sector con un campo de datos de 128 bytes, es decir, con un 68 % de espacio utilizable.

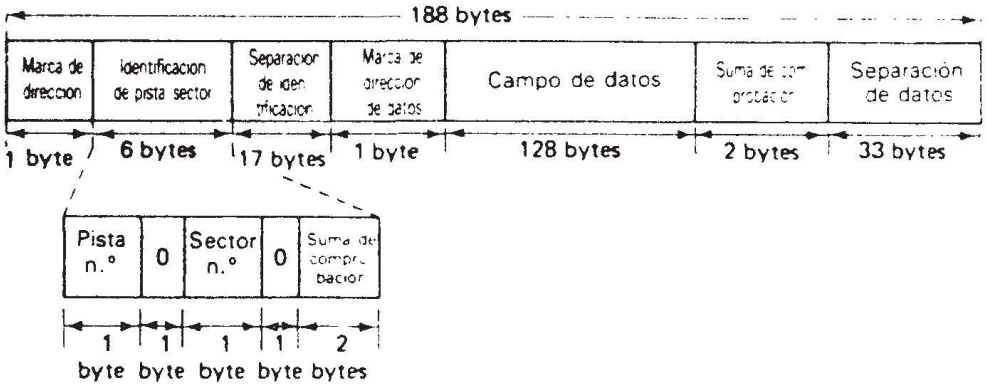


Figura 6.13
Formato de un sector lógico

Para su uso en los modernos ordenadores personales, los discos se formatean con sectores de 512 bytes (aunque puede hacerse con 128, 256, 512 6 1.024 bytes), dependiendo el número de sectores por pista de la capacidad de almacenamiento físico de la pista. Así, los discos flexibles de 5 1/4" de capacidad normal se formatean con 9 sectores/pista y los de alta capacidad, con 15 sectores/pista. Los microdiscos de 3 1/2" de capacidad normal se formatean también con 9 sectores/pista, mientras que los de alta capacidad admiten 18 sectores/pista. Los discos duros suelen formatearse a 17 sectores/pista. En todos los casos el sector contiene unos 200 bits de identificación, lo que permite un aprovechamiento del 72 % de la capacidad real del disco para almacenar información. Si se corrigen los valores de capacidad anteriormente indicados para los discos flexibles con este factor, se obtiene su capacidad real, que es la que suele indicarse comercialmente.

Obsérvese finalmente que este tipo de formateado se denomina *formateado físico* o *formateado de bajo nivel* (*low-level formatting*) para distinguirlo del *formateado lógico* o *formateado de alto nivel* (*high-level formatting*), que se estudia a continuación.

6.3.4 ESTRUCTURACION LOGICA DE UN DISCO

Un disco al que se le ha dado un formato físico (caras, pistas, sectores) no tiene demasiada utilidad si no se establece la forma de manipular estas unidades de almacenamiento de información, es decir, se

necesita un "mapa" para poder acceder de forma directa a la información. Este mapa generalmente lo crea el *sistema operativo* (*operating system*) del ordenador, entendiéndose como tal el conjunto de procedimientos que controlan los recursos, tanto físicos como lógicos, de que dispone el ordenador. Cada sistema operativo tiene su propia forma de estructurar los discos, pero como norma general contempla el disco como una serie de sectores lógicos en secuencia continua.

Debido a la gran variedad existente de sistemas operativos, no se puede realizar aquí un estudio exhaustivo de la forma en que cada uno de ellos estructura un disco. Sin embargo, la gran popularidad que ha alcanzado el DOS (*Disk Operating System*), debido a su uso en los ordenadores personales, hace que sea de interés analizar la estructuración lógica que éste hace de los discos.

Para acceder a un sector de un disco, se necesita un sistema de 4 coordenadas:

* Unidad
* Cara
* Pista de la cara
* Sector de la pista

Una vez leído el sector, para acceder a un byte específico del mismo, se necesita especificar además el desplazamiento dentro del sector. Si éste consta de 512 bytes, el desplazamiento tendrá un valor comprendido entre 000h y 1FFh.

El DOS localiza la información por un número de *sector lógico* (*logical sector*) o *sector absoluto* (*absolute sector*), cuya numeración se establece por pista (p), cara (c) y sector (s), en este orden, siendo el primer sector el 0, es decir, para un disco flexible de doble cara, doble densidad y capacidad normal, se pueden establecer 720 sectores lógicos (sl) de la forma:

```
sl 000: p = 00, c = 0, s = 1
sl 001: p = 00, c = 0, s = 2
.. ...  ......  .....  .....
sl 008: p = 00, c = 0, s = 9
sl 009: p = 00, c = 1, s = 1
.. ...  ......  .....  .....
```

sl 719: P = 39, c = 1, s = 9

Por otra parte, para controlar el espacio de datos de un disco, el DOS lo divide en unidades lógicas llamadas *unidades de ubicación* o *grupos* (*clusters*), de tal forma que un grupo sea la unidad de reserva de espacio en disco, es decir una serie de sectores adyacentes donde se puede almacenar un conjunto organizado de registros, denominado *archivo* o *fichero* (*file*). El tamaño de un grupo (w) varía de un formato a otro, pero en general es de 2 sectores (1.024 bytes) para discos flexibles de doble cara, doble densidad y capacidad normal, 1 sector (512 bytes) para discos flexibles de alta capacidad y 4 u 8 sectores (2.048 ó 4.096 bytes) para un disco duro.

Obsérvese que la reserva de espacio que produce un grupo para un fichero no es utilizable por otro, es decir, si un grupo tiene un tamaño de 1.024 bytes, cualquier fichero de tamaño inferior consumirá este espacio del disco.

Teniendo en cuenta que los grupos se numeran del 2 en adelante, por los motivos que se indicarán posteriormente, la relación entre el número del sector lógico y el número del grupo (g) para un conjunto de información situada en el área de datos del disco, será:

$$sl = (g - 2) * w + i \qquad [6.2]$$

siendo i el número del sector lógico inicial del área de datos.

6.3.4.1 AREAS DEL SISTEMA OPERATIVO

Cuando el DOS crea la estructura lógica de un disco, construye sobre él 4 áreas exclusivamente reservadas para su utilización por el sistema operativo, ubicadas en los primeros sectores lógicos del disco; el resto del espacio en disco puede ser asignado a cualquier fichero. Estas áreas especiales son:

* Sector de arranque
* Tabla de ubicación de ficheros
* Copia de la tabla de ubicación de ficheros
* Directorio raiz

El *sector de arranque* (*boot*) es el primer sector (512 bytes) de un disco y contiene un pequeño programa que arranca el proceso de carga del

sistema operativo en memoria, si sus ficheros correspondientes están en el disco. Estos ficheros tienen un nombre característico, generalmente IBMBIO.COM e IBMDOS.COM, y contienen, respectivamente, el *sistema básico de entrada/salida* (*Basic Input/Output System*) y el *sistema operativo de disco* (*Disk Operating System*). Estos dos ficheron tienen que estar indefectiblemente ubicados en los primeros sectores del área de datos del disco ya que, en caso contrario, o de no existir, el ordenador no arranca y emite un mensaje de error.

En general, el sector de arranque contiene toda la información crítica que concierne a las características del disco como medio de almacenamiento, **mostrándose su estructura en la Figura 6.11.**

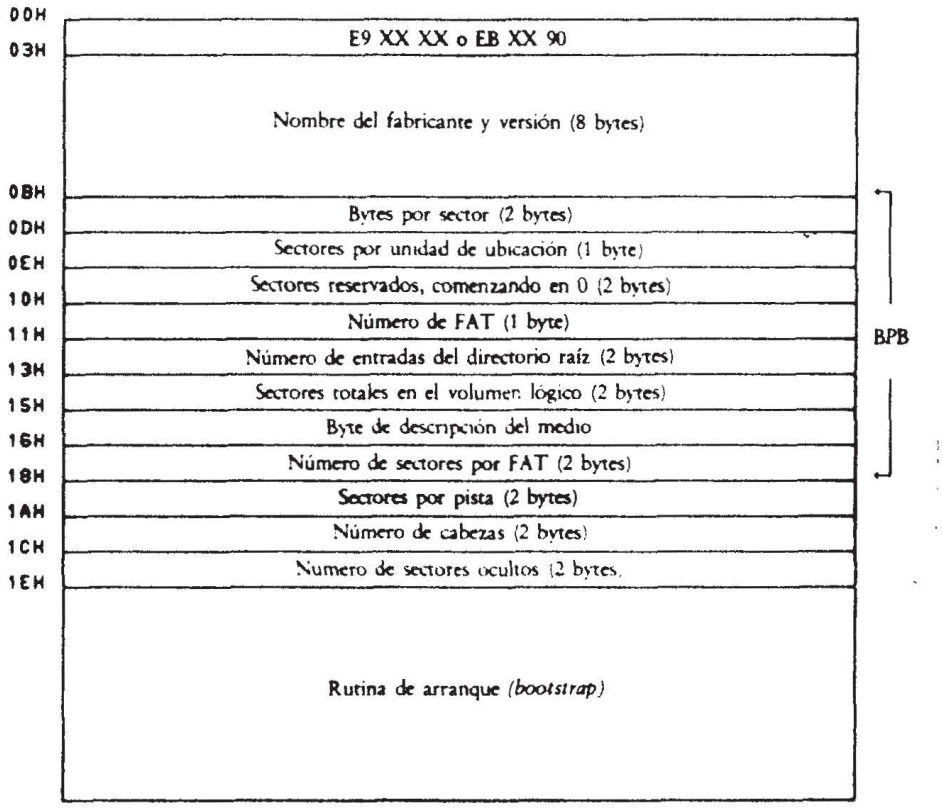


Figura 6.11

Sector de arranque de un disco bajo DOS

El primer byte de este sector es una instrucción de salto seguida de un *desplazamiento* (*offset*) dentro del sector, que indica el comienzo de la

rutina de arranque. El cuarto byte es el comienzo de un campo reservado para la identificación del fabricante del sistema operativo, que puede ocupar 8 bytes.

Entre los bytes 0Bh y 17h se encuentra el *bloque de parámetros del BIOS* (B*IOS* P*arameter* B*lock*), que describe las características físicas del disco y permite que el controlador del disco calcule las direcciones físicas adecuadas que corresponden a un número de sector lógico determinado. También contiene información para calcular la dirección y tamaño de cada una de las áreas reservadas del disco. Es imprescindible indicar en este punto que los campos asignados a 2 bytes que contienen información numérica siempre se leen en orden inverso, ya que primero se escribe el byte menos significativo y luego el bytes más significativo.

El elemento final del sector de arranque es el propio *programa de arranque del disco* (*bootstrap*), que se carga en memoria por medio del programa de arranque de la ROM, el cual se ejecuta automáticamente al conectar el ordenador.

La *tabla de ubicación de ficheros* (F*ile* A*llocation* T*able*) es una tabla que consta de una entrada por cada grupo (*cluster*) del disco. El contenido de cada entrada indica si el grupo está libre [(0)000h], defectuoso [(F)FF7h] o asignado a un fichero; en este caso, indica si es el último grupo del fichero [(F)FFFh] o cuál es el grupo siguiente del fichero [(X)XXXh]. Las entradas se agrupan en campos de 12 bits si el disco contiene menos de 4.096 (2^{12}) grupos, o de 16 bits en caso contrario, de ahí las indicaciones anteriores entre paréntesis.

Los dos primeros campos de la FAT están siempre reservados: el primer byte contiene una copia del byte descriptor del medio, que puede encontrarse también en el BPB del sector de arranque; el segundo, tercero y (cuando lo hay) cuarto byte, contienen siempre FF.

El resto de las entradas de la FAT describe la utilización de sus correspondientes grupos de disco. El número del grupo asignado a un fichero se encuentra el en directorio raiz y se utiliza como punto de entrada en la FAT. A partir de ese punto de entrada, cada entrada de la FAT contiene el número del siguiente grupo del fichero, hasta encontrar la marca del último grupo.

El número de sectores que ocupa la FAT depende del medio de almacenamiento y está indicado en el desplazamiento 16h (2 bytes) del BPB en el sector de arranque.

La *copia de la tabla de ubicación de ficheros* es una opción que permite salvar el contenido del disco si la FAT original se vuelve ilegible por accidente, ya que la copia se actualiza simultáneamente cuando se amplían los ficheros o se modifica su ubicación en el disco. Si se accede a un sector cuya FAT produce un fallo por error de lectura, se intenta obtener la información a partir de la copia, para poder leer correctamente el disco. Esta tabla ocupa el mismo número de sectores que la FAT original, y puede existir más de una copia de la misma, estando indicado el número total de tablas de ubicación de ficheros en el desplazamiento 10h (1 byte) del BPB en el sector de arranque.

El *directorio raiz* (*root*) se encuentra a continuación de las FATs y es la zona donde se almacenan los nombres y las características de los ficheros contenidos en el disco. Esta zona contiene entradas de 32 bytes en formato absolutamente rígido, y cada una de ellas describe un fichero, subdirectorio o etiqueta de volumen (si existe). El tamaño del directorio raiz queda determinado por formateado lógico y se describe en el desplazamiento 11h (2 bytes) del BPB en el sector de arranque. Como cada entrada ocupa 32bytes, pueden existir hasta 16 (512/32) entradas por sector, por lo que el número de entradas viene establecido por el número de sectores que el sistema operativo le asigna a esta área.

La estructura de una entrada se muestra en la tabla de la **Figura 6.15**, siendo de interés indicar el significado de los siguientes campos:

* Primer byte del nombre del fichero, cuyo significado muestra la **Figura 6.16**.
* Byte de atributo de fichero, cuyos valores o suma de valores están indicados en la **Figura 6.17**.
* Formato de los campos de fecha y hora, mostrados en la **Figura 6.18**.

Desp.	*Long.*	*Campo*
0	8	Nombre del fichero.
8	3	Extensión del fichero.
11	1	Atributo del fichero.
12	10	Reservado.
22	2	Hora de creación o de la última actualización.
24	2	Fecha de creación o de la última actualización.
26	2	*Cluster* inicial (bytes en orden inverso).
28	4	Tamaño del fichero (bytes en orden inverso).

Figura 6.15

Estructura de una entrada del directorio raiz

Primer byte	*Significado*
00h	No existen más entradas en el directorio (entrada no utilizada).
E5h (σ)	Entrada vacia (se usó, pero el fichero se ha borrado).
2Eh (.)	Entrada correspondiente a un subdirectorio. Si el segundo byte es también 2Eh, el campo de *cluster* inicial indica el comienzo del directorio padre, excepto cuando el padre es el raiz, en cuyo caso el campo de *cluster* inicial es 0000h.
XXh	Cualquier otro valor es el primer carácter del nombre del fichero.

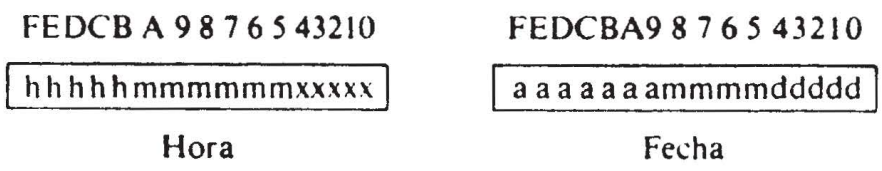
Figura 6.16

Primer byte del nombre del fichero

Bits	*Atributo*	*Significado*
0	01h	Fichero de sólo lectura.
1	02h	Fichero oculto.
2	04h	Fichero del sistema.
3	08h	Indica etiqueta del disco (en bytes 1 a 11).
4	10h	Subdirectorio.
5	20h	Fichero modificado desde el último BACKUP.
6	40h	No se usa.
7	80h	No se usa.

Figura 6.17

Byte de atributo de fichero



siendo:

hhhhh (5 bits, bits 15 a 11) = Horas (0 a 23).
mmmmmm (6 bits, bits 10 a 5) = Minutos (0 a 59).
xxxxx (5 bits, bits 4 a 0) = Incrementos de 2 segundos (0 a 29).
Hora = horas × 2048 + minutos × 32 + incrementos/2

aaaaaaa (7 bits, bits 15 a 9) = Año a sumar a 1980 (0 a 119). Indica años 1980 a 2099.
mmmm (4 bits, bits 8 a 5) = Mes (1 a 12).
ddddd (5 bits, bits 4 a 0) = Día (1 a 31).
Fecha = (año − 1980) × 512 + mes × 64 + día

Figura 6.18

Formato de los campos de fecha y hora

Finalmente, el resto del espacio de disco situado a continuación del directorio raiz se conoce con el nombre de *área de ficheros* o *área de datos* (*data area*). Los sectores de disco en esta zona se pueden contemplar como conjuntos de grupos, cada uno de ellos conteniendo uno o más sectores lógicos, dependiendo del formato del disco. Cada grupo dispone de su entrada correspondiente en la FAT y cada fichero dispone asimismo de su entrada en el directorio raiz.

Los subdirectorios son simplemente ficheros de tipo especial, por lo que pueden contener cualquier número de entradas de fichero, mientras que el directorio raiz está limitado en su tamaño a un número de sectores que le ha sido asignado por el sistema operativo. No debe olvidarse, finalmente, que si existen los programas de arranque del sistema, éstos deben estar contenidos **en los primeros grupos de la zona de datos, lo que implica ocupar los primeros lugares en la FAT y las dos primeras entradas del directorio raiz.**

6.3.4.2 PARTICIONES DE UN DISCO DURO

Como se ha indicado anteriormente, el formateado de un disco es de dos tipos: físico y lógico. El primero implica la creación de los sectores en el disco, así como sus marcas de dirección; el segundo implica la conversión a

las normas de un sistema operativo. Cuando se formatea un disco flexible con cualquier sistema operativo, se llevan a cabo simultáneamente los dos niveles de formateado; sin embargo, no ocurre lo mismo con el disco duro, ya que éste, por su gran capacidad, debe ser capaz de albergar simultáneamente diferentes sistemas operativos.

Así pues, cuando se lleva a cabo el formateado de un disco duro, lo que se hace es dividir éste en áreas o *particiones* (*partitions*), que puedan ser reconocidas por sistemas operativos diferentes. Dentro de los límites de cada partición, el disco puede ser formateado a un nivel superior, lógico, para poder adaptarlo a un sistema operativo determinado. Para ello, el primer sector de un disco duro está separado por un registro maestro especial, que contiene una *tabla de partición* (*partition table*) que describe la disposición del disco. Esta tabla muestra cuáles son las dimensiones del disco, cuántas particiones hay y cuál es la posición de cada una de ellas.

Además, el primer sector de un disco duro fraccionado contiene un *programa de arranque maestro* (*master boot*) junto con la tabla que describe las particiones. Este programa examina la tabla, determina qué partición está activa y luego ejecuta el programa de arranque de esa partición, que si corresponde al DOS, será el programa ya descrito anteriormente. En este momento el sistema operativo toma el control del ordenador y el disco duro se comporta igual que un disco flexible.

6.4 DETECCION DE ERRORES

Todo movimiento de información es una causa potencial de errores y si se requiere una transferencia de datos fiable debe contarse con algún método de detección de errores. En la práctica se utilizan tres métodos de detección de errores, aunque el último es el más ampliamente difundido:

* Detección de paridad
* Suma de comprobación
* Código de redundancia cíclica

La *detección de paridad* (*parity check*) es el método más sencillo de asegurar la precisión en las transferencias de datos. Para ello se añade al final de cada byte un *bit de paridad* (*parity bit*), determinándose arbitrariamente que todos los bytes de datos han de contener un número par

(paridad par) o impar (paridad impar) de 1. Antes de la escritura de un byte de datos, se determina el número de 1 del mismo y se pone el bit de paridad según la norma elegida. Al leer el dato se verifica el bit de paridad: si coincide con la del dato, la lectura ha sido correcta, en caso contrario ha habido un error. El método no es muy fiable, ya que sólo es eficaz para detectar un número impar de 0 que hayan cambiado a 1. Además requiere un bit adicional por cada byte, lo que representa un 11 % de ocupación de espacio.

La detección por *suma de comprobación* (*checksum*) consiste en una suma binaria sin acarreo de todos los datos del bloque que se está comprobando, byte a byte, almacenándose en un byte de suma, al final del bloque de datos. Durante una operación de lectura se realiza la suma de comprobación de los datos y se compara con la que está almacenada como parte del bloque de información; si no coinciden es señal de error. Como los grandes bloques de datos generan sumas relativamente largas, requieren gran cantidad de tiempo de ejecución, por lo que este sistema sólo se utiliza en algunos casos específicos.

La generación del *código de redundancia cíclica* (*Cyclic Redundancy Code*) es algo más compleja, pero constituye el método más fiable de detección de errores. Consiste en considerar un bloque de datos como un polinomio binario de grado N y utilizarlo como divisor de un polinomio normalizado ($X^{16} + X^{15} + X^2 + 1$ = 1 1000 0000 0000 0101) para obtener un cociente y un resto, éste de grado inferior al dividendo y único para cualquier bloque dado. El resto obtenido, un número de 16 bits, se denomina "código de redundancia cíclica" y permite detectar un error de un solo bit en bloques de datos de hasta 32.768 (2^{15}) bits de longitud. El método CRC es el sistema más habitual de detección de errores en disco debido a su extraordinaria fiabilidad.

BIBLIOGRAFIA

Baer, J.L.; *"Computer systems architecture"*, Computer Science Press, New York (1980).

Boyce, J.C.; *"Microordenadores"*, Alhambra, Madrid (1986).

Duncan, R.; *"MS-DOS avanzado"*, Anaya Multimedia, Madrid (1988).

Norton, P.; *"El IBM PC a fondo. Técnicas y programación avanzada"*, Anaya Multimedia, Madrid (1987).

Osborne, A. y Bunnel, D.; *"Introducción a las microcomputadoras"*, Vol. 0, 3ª ed., Osborne/McGraw-Hill, Madrid (1986).

Rodríguez-Roselló, M.A.; *"8088-8086/8087. Programación ensamblador en entorno MS-DOS"*, Anaya Multimedia, Madrid (1988).

Stone, H.S.; *"Microcomputer interfacing"*, Addison-Wesley, Reading (1983).

CAPITULO 7

CONVERSIONES ANALOGICAS

7.1 LOS CONVERSORES EN EL ORDENADOR

7.2 CONVERSORES DIGITAL/ANALOGICO

7.2.1 FORMATOS DE SALIDA Y ENTRADA

7.2.2 DISEÑO DE REDES DE RESISTENCIAS

7.2.3 PARAMETROS DE OPERACION DE LOS CONVERSORES D/A

7.3 CONVERSORES ANALOGICO/DIGITAL

7.3.1 PARAMETROS DE OPERACION DE LOS CONVERSORES A/D

7.1 LOS CONVERSORES EN EL ORDENADOR

Hasta ahora se ha considerado que todas las señales de entrada y salida del ordenador tienen forma binaria, ya sea en formato paralelo o serie, lo que sucede en la mayoría de las aplicaciones de proceso de datos. Sin embargo, los ordenadores se utilizan cada vez más en aplicaciones de control de procesos, cuyas señales, tanto de entrada como se salida, son analógicas o continuas.

Como es sabido, las señales digitales manejadas por los ordenadores tienen estados discretos de tensión; las señales analógicas son tensiones que varían de forma continua. Evidentemente, si se desea utilizar un ordenador en aplicaciones de control de procesos, es necesario disponer de circuitos que conviertan un tipo de señal al otro.

Así, un sistema que cambia una señal digital, representada por niveles de tensión discretos, en una señal continua, se denomina *decodificador* o *conversor digital/analógico* (*Digital/Analogic Converter*). Los circuitos que permiten llevar a cabo la transformación de una señal de tensión continua en una señal compuesta por estados discretos de tensión forman parte de un sistema llamado *codificador* o *conversor analógico/digital* (*Analogic/Digital Converter*).

7.2 CONVERSORES DIGITAL/ANALOGICO

Los conversores D/A son elementos que reciben una información digital en forma de una palabra de N bits y la transforman en analógica. Cada una de las 2^N configuraciones binarias representan otros tantos niveles discretos de tensión, que se convierten en una tensión analógica mediante una red de resistencias, que entra a un amplificador operacional para aislar la red de resistencias del resto del sistema, tal como muestra la **Figura 7.1.**

La entrada al conversor D/A es una señal binaria de N dígitos binarios en paralelo, V_{N-1}, V_{N-2}, ..., V_o, donde cada V_j es una tensión cuyo nivel es el adecuado para representar un 1 lógico o un 0 lógico. Como las tensiones V_j no tienen que darse con precisión (compatibilidad TTL), se utilizan para activar unos interruptores, cuyos valores binarios (abiertos o cerrados), S_o, S_1, ..., S_{N-1}, sí permiten generar una tensión analógica precisa a la salida de la red de resistencias. Es decir, normalmente se define una tensión de

referencia para el valor lógico 0, V(0) = 0, y otra para el valor lógico 1, V(1) = V_R, diseñándose la red de resistencias de tal forma que la resistencia total, R_T, sea una función de una resistencia fija, R, y de la posición de los interruptores S_j:

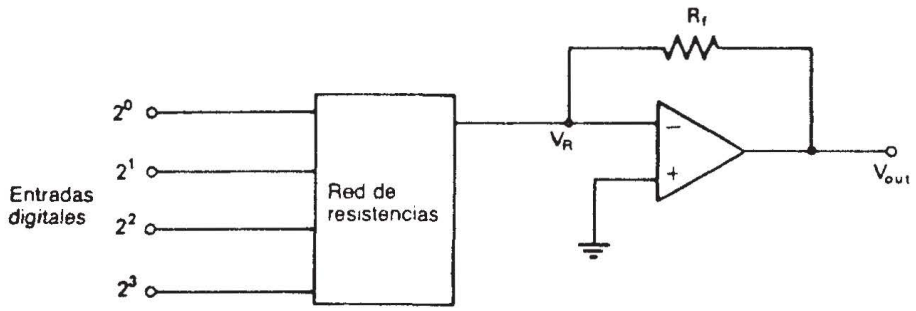


Figura 7.1
Diagrama conceptual de un conversor D/A

$$R_T = \frac{S_{N-1}\ 2^{N-1} + S_{N-2}\ 2^{N-2} + \ldots + S_o\ 2^o}{R} \qquad [7.1]$$

De esta forma, y teniendo en cuenta que el análisis del circuito del amplificador operacional permite obtener la tensión de salida del mismo de la forma:

$$V_o = \frac{R_f}{R_T} V_R \qquad [7.2]$$

considerando que en la Figura 7.1 la entrada es por el terminal negativo y el amplificador invierte el signo, se obtendrá:

$$V_o = \frac{R_f V_R}{R} \left(S_{N-1}\ 2^{N-1} + S_{N-2}\ 2^{N-2} + \ldots + S_o\ 2^o \right) \qquad [7.3]$$

expresión que indica que la tensión de salida es directamente proporcional al valor numérico del dígito binario de la entrada.

7.2.1 FORMATOS DE SALIDA Y ENTRADA

El diseño indicado sólo permite obtener tensiones analógicas positivas,

al haber tomado V(0) = 0. A este tipo de salida de un conversor D/A se le suele denominar *formato binario unipolar*. Así, si se supone como ejemplo que $R_f = R$ y que $V_R = 1$ V, para un conversor D/A de 3 bits, el valor mínimo que podrá sacar corresponderá a la configuración binaria 000, es decir:

$$(V_o)_{mín} = 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 0 \text{ V} \qquad [7.4]$$

mientras que el valor máximo corresponderá a la configuración binaria 111, es decir:

$$(V_o)_{máx} = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 15 \text{ V} \qquad [7.5]$$

Para poder obtener tensiones negativas, basta cambiar de tensiones de referencia para los niveles lógicos 1 y 0. Así, si se utiliza para el nivel lógico 0 una tensión negativa, se podrán obtener tensiones de salida del conversor D/A negativas. Siguiendo con el ejemplo anterior, si ahora se definen $V(0) = -V_R$ y $V(1) = V_R$, se tendrá que la salida de tensión mínima puede ser:

$$(V_o)_{mín} = -(1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = -7 \text{ V} \qquad [7.6]$$

ya que todos los bits de entrada puestos a 0 significa que cada uno de ellos contribuye al sistema con -1 V, al ser $V(0) = -V_R$.

Por su parte, el valor máximo será:

$$(V_o)_{máx} = +(1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) = +7 \text{ V} \qquad [7.7]$$

ya que todos los bits de entrada puestos a 1 implica que sólo contribuye a la salida una tensión positiva.

A este tipo de salida de un conversor D/A se le suele denominar *formato binario bipolar*.

La tabla mostrada en la Figura 7.2 permite apreciar el cambio de la tensión de referencia, observándose además que el formato bipolar, al ampliar al doble la escala analógica, reduce la precisión a la mitad, ya que la escala digital permanece igual.

También se puede desplazar el margen de salida de un conversor D/A inyectando una *corriente de desplazamiento* (*offset*) en la entrada del

amplificador, ajustada de tal forma que haga nula la tensión analógica para la entrada digital que se desee. Continuando con el ejemplo anterior, si se desea transformar el formato unipolar en bipolar, se puede ajustar a cero la tensión analógica correspondiente a la entrada digital 100 (o a otra cualquiera). En este caso, como la tensión analógica unipolar correspondiente a esa entrada digital es +4 V, al ajustar a cero, la salida bipolar, V_o', tendrá la expresión:

$$V_o' = V_o - 4 \qquad [7.8]$$

con lo que los valores extremos que podrá alcanzar la tensión analógica serán de +3 V y -4 V.

Entrada digital	Salida unipolar V(1) = 1; V(0) = 0	Salida bipolar V(1) = 1; V(0) = -1
111	4 + 2 + 1 = 7	4 + 2 + 1 = +7
110	4 + 2 + 0 = 6	4 + 2 - 1 = +5
101	4 + 0 + 1 = 5	4 - 2 + 1 = +3
100	4 + 0 + 0 = 4	4 - 2 - 1 = +1
011	0 + 2 + 1 = 3	-4 + 2 + 1 = -1
010	0 + 2 + 0 = 2	-4 + 2 - 1 = -3
001	0 + 0 + 1 = 1	-4 - 2 + 1 = -5
000	0 + 0 + 0 = 0	-4 - 2 - 1 = -7

Figura 7.2
Formatos de salida por cambio de la tensión de referencia

Obsérvese, sin embargo, que hasta aquí se ha operado con dígitos binarios puros a la entrada, lo que no permite obtener tensiones analógicas a partir de números negativos. Como es sabido, para representar números negativos en binario se utiliza la notación de complemento a dos. Si se desean obtener tensiones analógicas a partir de entradas digitales expresadas en complemento a dos, manteniendo su signo, habrá que aplicar al conversor D/A la entrada digital con el MSB complementado y ajustar luego el desplazamiento, de modo que la configuración 100 (000 en complemento a dos, pero el MSB se ha complementado) corresponde a 0 V (valor decimal de 000). El

procedimiento de conversión se deduce inmediatamente del análisis de la tabla mostrada en la Figura 7.3.

Decimal	Binario C 2	Entrada D/A C MSB	Salida unipolar	Salida bipolar 100 = 0 V
+3	011	111	4 + 2 + 1 = 7	7 - 4 = +3
+2	010	110	4 + 2 + 0 = 6	6 - 4 = +2
+1	001	101	4 + 0 + 1 = 5	5 - 4 = +1
0	000	100	4 + 0 + 0 = 4	4 - 4 = 0
-1	111	011	0 + 2 + 1 = 3	3 - 4 = -1
-2	110	010	0 + 2 + 0 = 2	2 - 4 = -2
-3	101	001	0 + 0 + 1 = 1	1 - 4 = -3
-4	100	000	0 + 0 + 0 = 0	0 - 4 = -4

Figura 7.3

Formato de salida bipolar a partir de entrada en complemento a dos

7.2.2 DISEÑO DE REDES DE RESISTENCIAS

Existen dos métodos para configurar la red de resistencias necesaria para realizar la conversión de niveles discretos de tensión en tensiones continuas, utilizadas en los conversores D/A:

* Red de resistencias ponderadas
* Red de escalera de resistencias

La red de *resistencias ponderadas* consiste en conectar en paralelo varias resistencias, cuyos valores están acordes con su posición relativa en las entradas digitales. La tensión de salida se obtiene en un nudo común y el otro extremo de cada resistencia se puede conectar a una tensión de referencia o a tierra, mediante un conmutador, tal como muestra la Figura 7.4.

Obsérvese que el valor más elevado de resistencia representa al LSB. El valor de cada resistencia se reduce a la mitad al aumentar la significación del bit de entrada, de forma que el MSB de la entrada digital tiene la menor resistencia de la red. Por tanto, la corriente que pasa por cada rama dobla a la anterior, de igual forma que se dobla el valor digital de cada entrada.

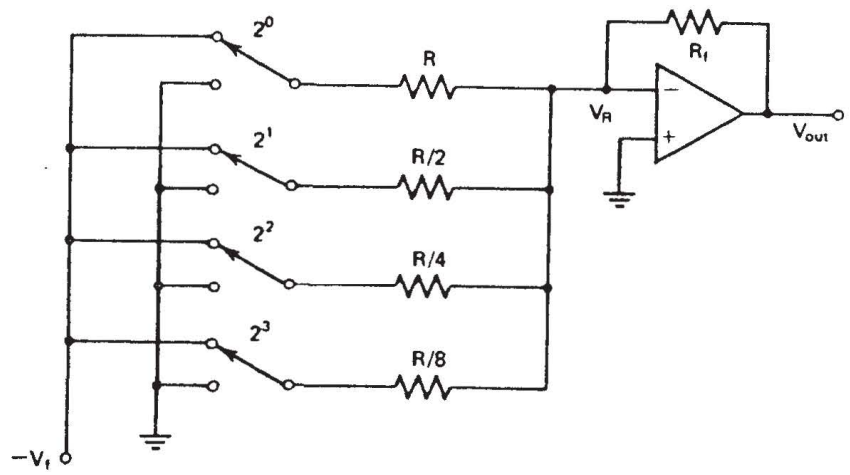


Figura 7.4

Conversor D/A con red de resistencias ponderadas

El conversor D/A de resistencias ponderadas es relativamente simple, pero tiene los siguientes inconvenientes:

* Para lograr una buena estabilidad y precisión del conversor D/A se requiere un alto grado de precisión en las resistencias, lo que gneralmente es difícil de conseguir.

* Como todas las resistencias de la red tienen valores diferentes, es muy difícil que sus parámetros (coeficiente térmico, envejecimiento, etc.) varíen de forma parecida, lo que agrava los problemas de estabilidad y precisión del conversor.

* Al ser el valor de la resistencia de cada entrada digital la mitad del precedente, a medida que aumenta el número de entradas digitales se multiplica el número de valores distintos de resistencias, lo que ocasiona problemas de integración de componentes.

La *red de escalera de resistencias* permite remediar muchos de los inconvenientes asociados a la red de resistencias ponderadas, ya que sólo utiliza dos valores de resistencia (R y 2R), por lo que esta configuración se denomina a menudo "conversor D/A de escalera R-2R". El funcionamiento de esta red se basa en el principio del divisor de tensión, tal como muestra la Figura 7.5.

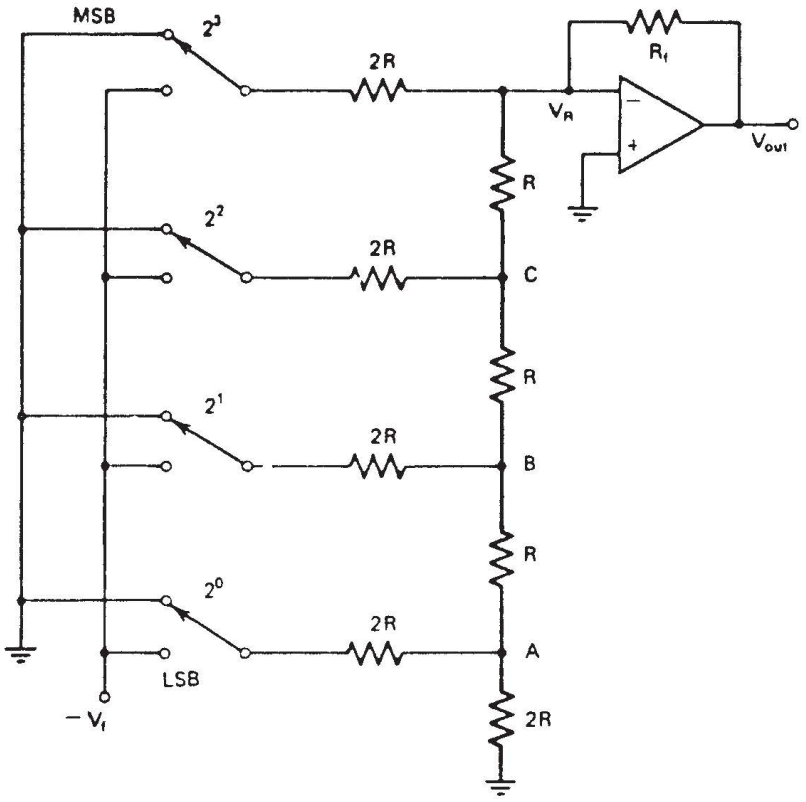


Figura 7.5

Conversor D/A con red de escalera de resistencias

La salida analógica de este circuito debe tener una resistencia de 2R respecto a tierra. Si las entradas digitales son todas 0, los cuatro conmutadores están a tierra. Con todas las resistencias 2R conectadas a tierra, los nudos A, B y C están a tierra y V_R está a 0 V. Cuando una entrada digital es distinta de 0, el conmutador correspondiente está en la línea de tensión de referencia, V_f. La corriente circula por esta rama y se divide en el nudo correspondiente. Debido a la configuración del circuito, la corriente se divide por 2 en cada nudo de éste, por lo que la corriente en el nudo V_R queda determinada por el número de nudos por los que circula.

En comparación con la red de resistencias ponderadas, la red de escalera de resistencias tiene las siguientes ventajas:

* Como todas las resistencias usadas en la escalera tienen sólo dos valores, la precisión del conversor D/A no depende de los valores absolutos de las resistencias, sino de la diferencia de valor entre ellas.

* Con solo dos valores de resistencia, se reduce considerablemente el

problema de variación de la resistencia con la temperatura; ahora sólo es necesaria una buena relación entre dos resistencias.

Sin embargo, estas ventajas tienen un precio, y es que esta configuración utiliza el doble número de resistencias que la red de resistencias ponderadas, lo que ocasiona aún más problemas en la integración de los componentes.

7.2.3 PARAMETROS DE OPERACION DE LOS CONVERSORES D/A

Los parámetros que se utilizan para describir la calidad de las prestaciones de un conversor D/A son los siguientes:

* Resolución
* Linealidad
* Precisión
* Tiempo de respuesta

La *resolución* especifica el número de bits que el conversor puede acomodar y, por tanto, el número correspondiente de tensiones de salida. Así, se puede dar la resolución como el número de bits que admite el conversor a la entrada, o como el cambio más pequeño que el conversor puede apreciar (en porcentaje) del margen de salida a plena escala. Es decir, la resolución porcentual se puede expresar por la siguiente ecuación:

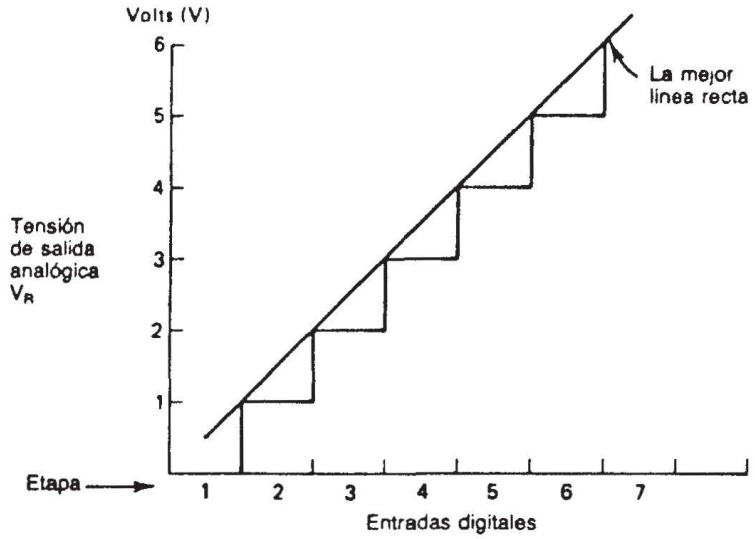
$$R_{PC} = \frac{100}{2^n} \quad (\%) \qquad [7.9]$$

Así por ejemplo, un conversor D/A que permite una entrada de 10 bits tiene una resolución de 10 bits o una resolución pocentual de:

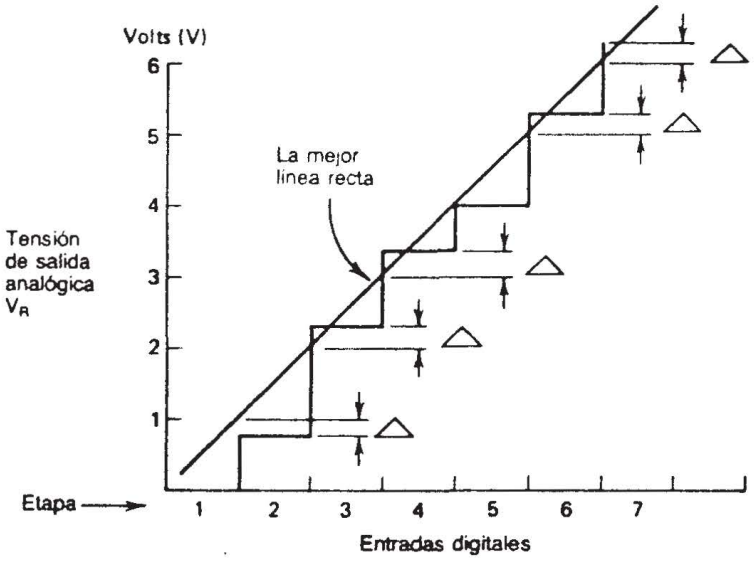
$$R_{PC} = \frac{100}{2^{10}} = \frac{100}{1.024} = 0,098 \simeq 0,1\ \% \qquad [7.10]$$

Resoluciones típicas son 8, 10, 12 y 14 bits, esta última con una resolución que alcanza hasta el 0,006 % ó 60 ppm (partes por millón).

La *linealidad* es la desviación máxima a la salida del conversor de la denominada "línea recta", es decir, la recta que une los escalones de salida en el estado ideal, tal como muestra la **Figura 7.6 (a)**. Como en la realidad raramente se puede obtener tal recta, como se muestra en la **Figura 7.6 (b)**, la diferencia entre la mejor línea recta y los niveles de tensión reales en cada escalón, Δ, es el parámetro a minimizar.



(*a*) Salida de un DAC con una linealidad perfecta



(*b*) Salida de un DAC con una linealidad típica

Figura 7.6

Linealidad de salida de un conversor D/A

Generalmente se especifica una linealidad tal, que la desviación máxima, $\Delta_{máx}$, sea igual o menor que la mitad de la magnitud del escalón teórico perfecto, V_{PF}:

$$\Delta_{máx} \leq \frac{V_{PF}}{2} \qquad [7.11]$$

obteniéndose V_{PF} dividiendo la salida total del conversor D/A por el número

de escalones posibles (2^n, siendo n el número de bits de entrada). A veces se utiliza también como base la magnitud del escalón correspondiente a la variación de entrada equivalente a un cambio en el LBS, especificándose la linealidad como "menor que ± 1/2 LSB".

La *precisión* es una medida de la desviación de la salida real de un conversor D/A respecto a la teórica, calculada para una cierta combinación de entradas digitales. A veces se confunde con la linealidad o con la resolución, lo cual no es correcto, ya que la precisión tiene un factor de escala. Una especificación típica de un conversor D/A de calidad media podría ser de una precisión del 0,2 % de plena escala.

El *tiempo de respuesta* es el tiempo requerido para que la salida se estabilice dentro de ciertos límites después de un cambio en la combinación de entradas digitales. Los límites aceptables son generalmente los de máxima linealidad, tal como se muestra en la Figura 7.7.

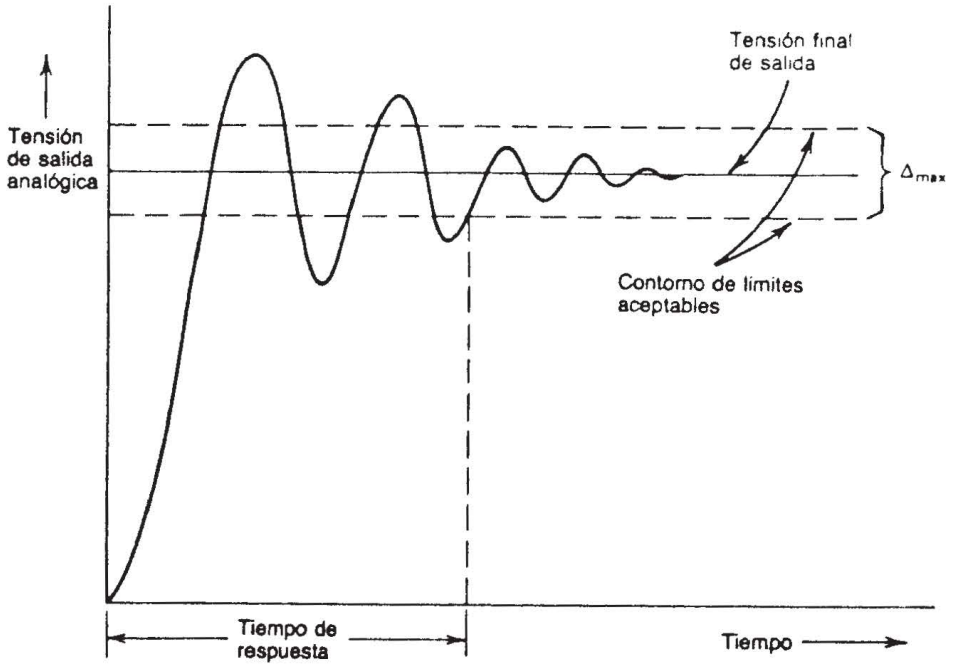


Figura 7.7

Tiempo de respuesta de un conversor D/A

7.3 CONVERSORES ANALOGICO/DIGITAL

Los conversores A/D son elementos que transforman un nivel de tensión

en una información digital con un código determinado. Como el número de bits que se obtienen del conversor es finito, el código de salida deberá ser siempre el correspondiente al valor más cercano que pueda expresarse mediante los n bits, por lo que el proceso de conversión implica hacer una aproximación.

Como la conversión A/D es, en esencia, un proceso bastante más complejo que la conversión D/A, para llevarla a cabo se han propuesto fórmulas muy diversas, aunque aquí se describirán sólo las dos técnicas más comunes: la rampa de tensión y las aproximaciones sucesivas. No obstante, todos los métodos utilizan como elemento principal del diseño un amplificador operacional denominado "comparador".

El *comparador* es simplemente un amplificador operacional diferencial, cuya salida es función de la diferencia de tensión entre las dos señales aplicadas a sus entradas. Una de ellas es una señal de referencia y la otra es la señal de entrada analógica a digitalizar. Tal como se muestra en la Figura 7.8, la señal de tensión de salida cambia bruscamente de 0 a 1 cuando la entrada analógica excede de la señal de referencia.

El *método de la rampa de tensión* se esquematiza en la Figura 7.9. La tensión analógica se aplica a una de las entradas del comparador; la tensión de referencia, que se aplica a la otra entrada, es una rampa de tensión que crece linealmente con el tiempo, tal como muestra la Figura 7.9 (a). Mientras tanto, los pulsos del reloj entran al contador a través de la puerta AND A1 y lo van incrementando.

Cuando la tensión en la rampa alcanza o sobrepasa ligeramente la tensión de entrada analógica, la salida del comparador cambia de 0 a 1, lo que desactiva el biestable y, por tanto, la puerta AND A1, evitando que los pulsos de reloj sigan incrementando el contador. La salida a 1 del comparador bloquea también la puerta AND A2, que desconecta el generador de rampa de tensión, volviendo a 0 su salida, en preparación para una próxima conversión. El contador contendrá el equivalente digital de la señal de entrada analógica.

El *método de aproximaciones sucesivas* es otra técnica ampliamente utilizada para la conversión A/D. En este caso se generan una serie de señales digitales que se convierten a sus equivalentes analógicas, que se

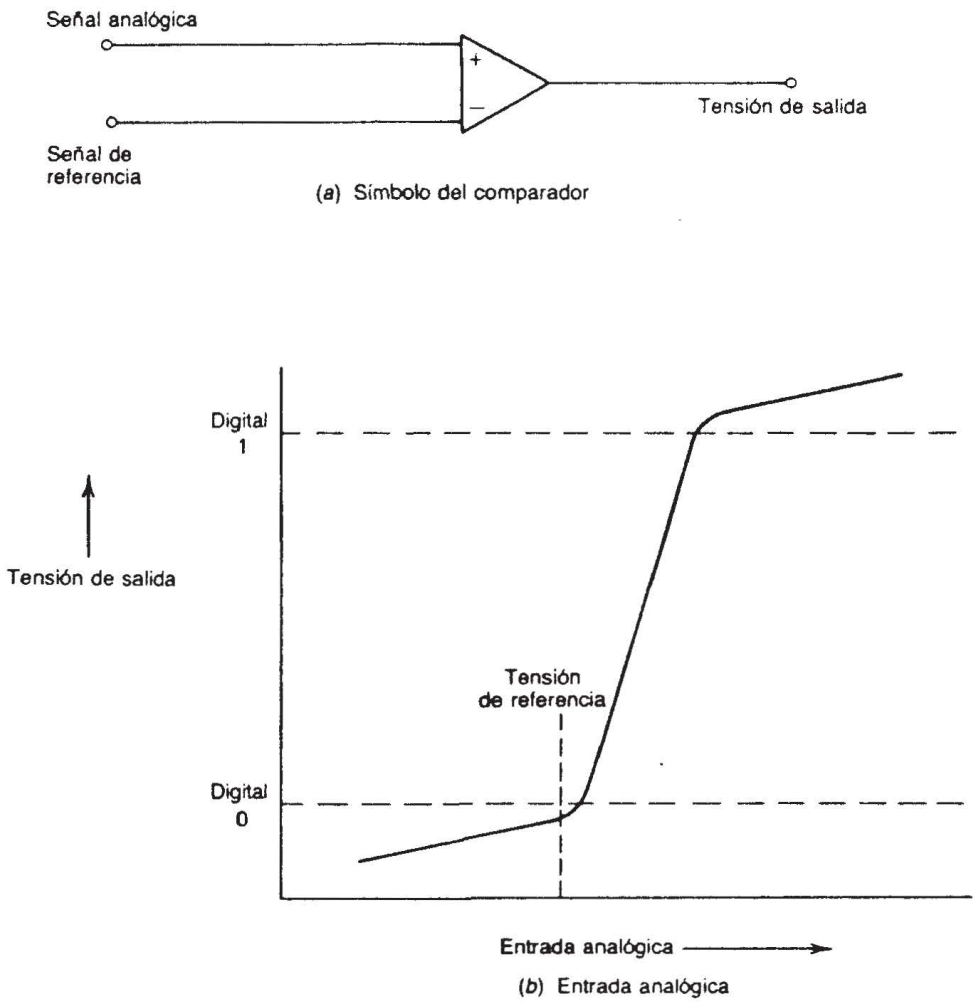


Figura 7.8
Características de un comparador

aplican como señal de referencia a una entrada de un comparador, a cuya otra entrada se conecta la señal analógica a digitalizar. Este proceso se repite varias veces con diferentes señales de referencia, a las que se denomina ***señales de ensayo***, **que se utilizan para aproximarse sucesivamente a la señal analógica de entrada.**

El método de aproximaciones sucesivas más simple es muy similar al método de la rampa de tensión, excepto que, en lugar de utilizar como entrada una rampa, se usa una tensión que varía de forma escalonada y que sale de un conversor D/A, tal como muestra la **Figura 7.10.**

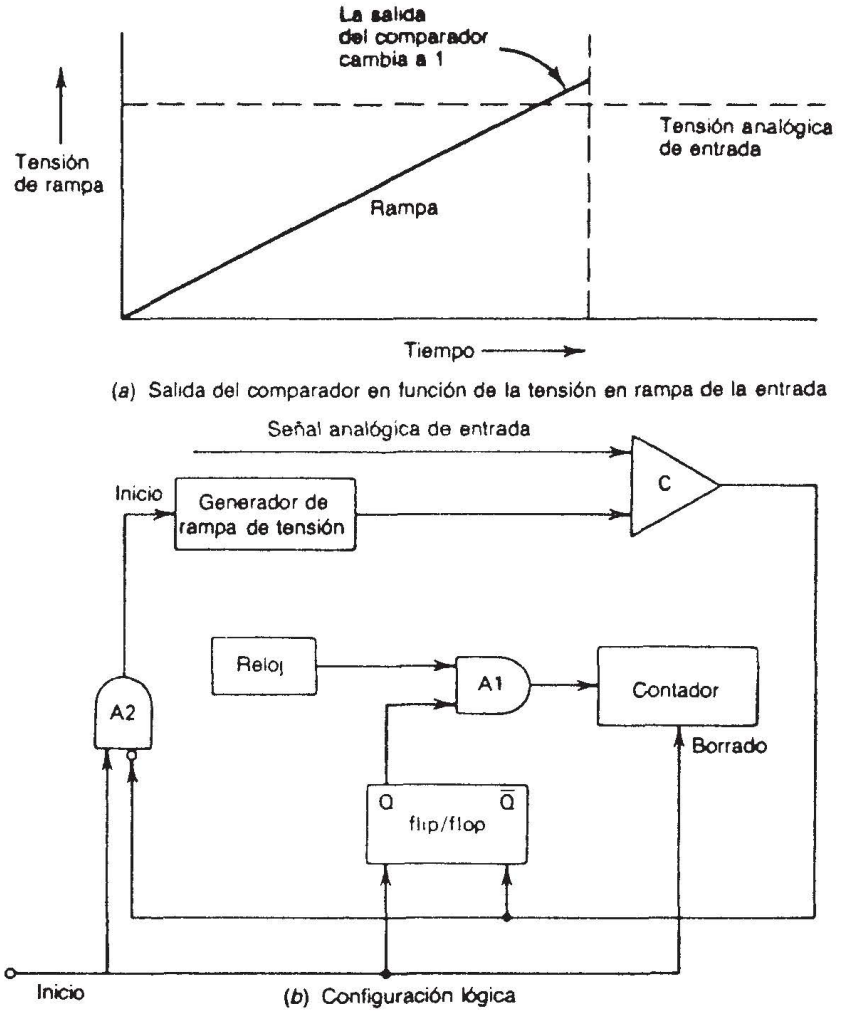


Figura 7.9

Conversión A/D por rampa de tensión

La señal analógica de entrada a digitalizar se aplica a una entrada del comparador. Mientras tanto, los pulsos del reloj pasan a través de la puerta AND A1 y empiezan a incrementar el contador de 4 bits, cuya salida se aplica al conversor D/A, que a su vez se conecta a la otra entrada del comparador.

Cuando la salida del conversor D/A excede de la señal analógica de entrada, el comparador se pone a 1 y desactiva el biestable, lo que bloquea la puerta AND A1 y, por tanto, detiene el contador. Su contenido es el equivalente digital de la señal de entrada analógica.

Generalmente se utiliza un método de aproximaciones sucesivas secuencial, más sofisticado y, por tanto, más preciso, cuyo esquema lógico es el mostrado en la **Figura 7.11**.

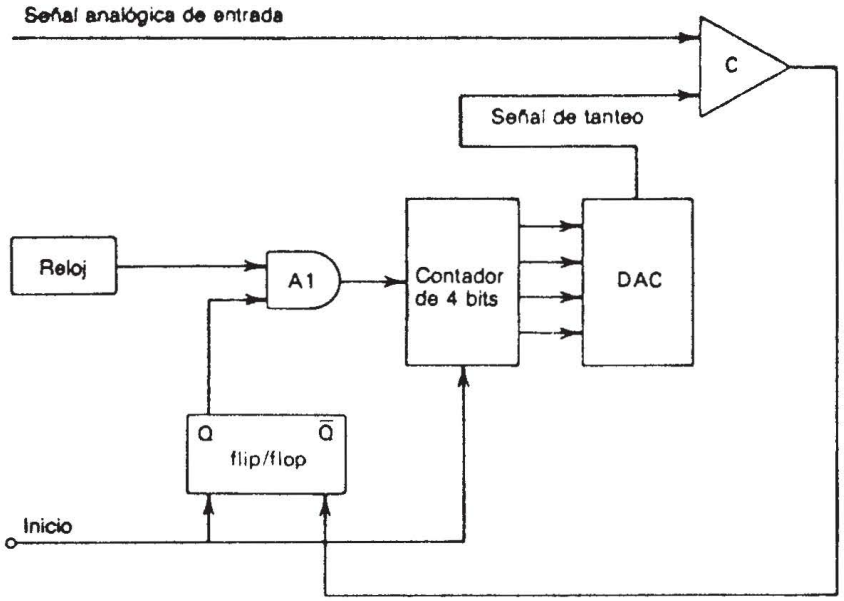


Figura 7.10

Conversión A/D simple por aproximaciones sucesivas

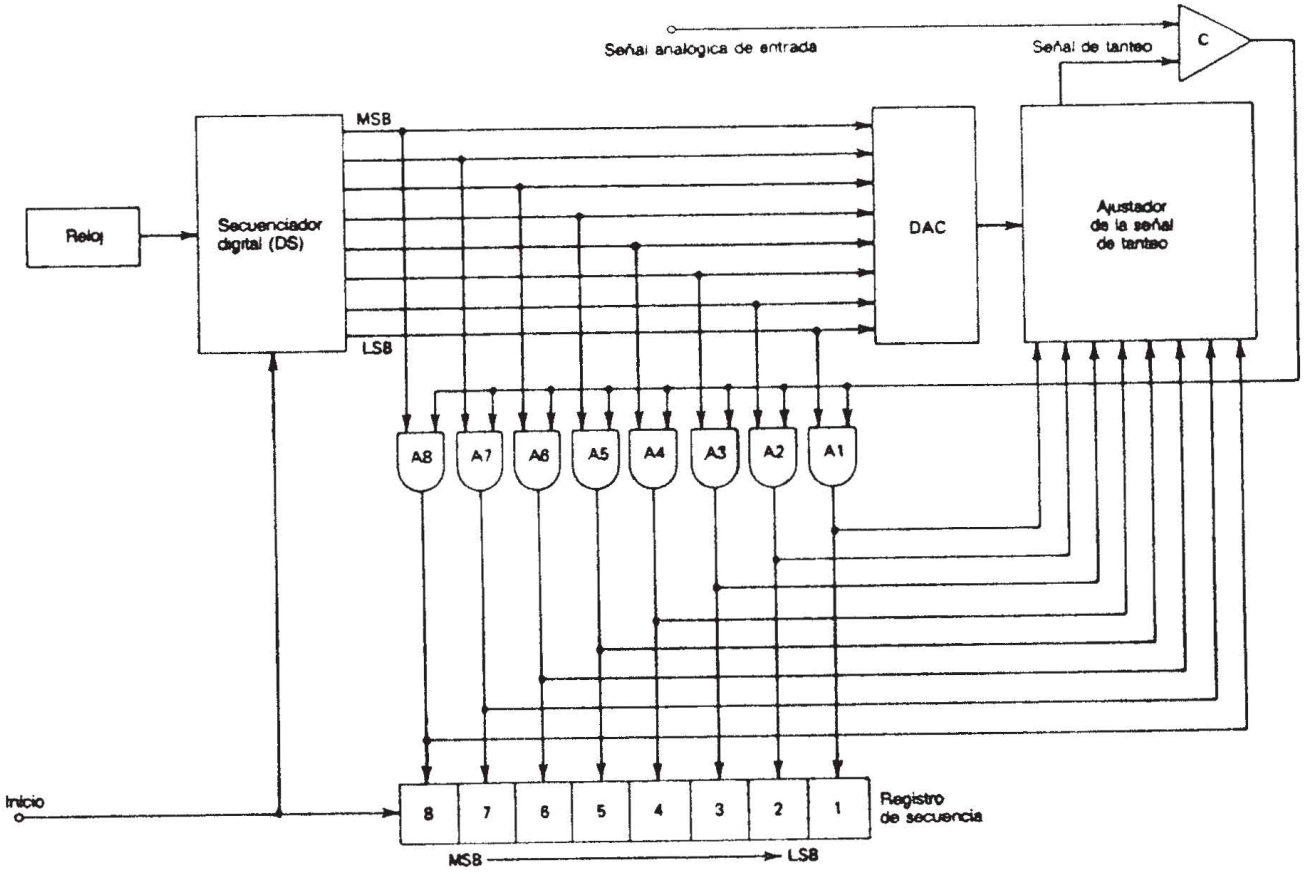


Figura 7.11

Conversión A/D secuencial por aproximaciones sucesivas

En este método, primero se determina si la señal de entrada analógica se encuentra en la mitad superior o inferior del rango completo de entrada. Para ello se aplica primero al conversor D/A (con un secuenciador digital) el MSB, lo que hace que el ajustador de la señal de ensayo envíe una señal que es la mitad del rango completo de escala. Si la señal de entrada analógica se encuentra en la mitad alta, el comparador envía un 1 y la mitad baja se ignora durante el resto de la operación. Si, por el contrario, está en la mitad baja, el comparador envía un 0 y la mitad alta es ignorada. La salida del comparador se almacena en un registro especial de secuencia y se aplica al conversor D/A otra señal digital, cuyo valor será ahora la mitad de la mitad determinada en el ensayo anterior. La nueva salida resultante del comparador se almacena en la siguiente posición del registro de secuencia, repitiéndose seguidamente el proceso con señales de ensayo que van de mitad en mitad y que corresponden al punto medio del rango determinado en el ensayo previo.

Después de las 8 etapas del proceso (supuesto que el conversor A/D sea de 8 bits), el registro de secuencia contendrá la aproximación digital de la señal analógica de entrada.

7.3.1 PARAMETROS DE OPERACION EN LOS CONVERSORES A/D

Los parámetros de un conversor A/D que normalmente son suministrados por el fabricante son los siguientes:

* Resolución
* Rango de entrada
* Formato de salida
* Tiempo de conversión
* Impedancia de entrada
* Precisión

La *resolución* es el número de bits de conversor, generalmente, 8, 10 ó 12.

El *rango de entrada* es el máximo margen permisible de la tensión analógica de entrada. Valores típicos son 10 V, ±5 V y ±10 V.

El *formato de salida* es la forma en que pueden estar codificadas las

salidas digitales de un conversor A/D. Se pueden encontrar salidas en binario puro, complemento a uno, complemento a dos o BCD.

El *tiempo de conversión* es el tiempo que tarda el dispositivo en convertir un valor de entrada analógica en un valor digital, y suele darse su valor máximo, que oscila entre los 50 μs, para conversores de velocidad moderada y los 50 ns, para conversores de alta velocidad.

La *impedancia de entrada* al conversor es un parámetro importante que asegura su compatibilidad con la fuente de la señal analógica. Es deseable disponer de una elevada impedancia de entrada para evitar cargar la fuente, por lo que se suelen especificar valores entre 1 kΩ y 1 MΩ.

La *precisión* de un conversor A/D incluye el error de cuantificación, el ruido del sistema digital y las desviaciones con respecto a la linealidad. La precisión determina la resolución del conversor y se manejan valores típicos de $\pm$0,02 % de lectura en todo el rango de entrada, aunque se pueden adquirir conversores A/D de mayor precisión.

BIBLIOGRAFIA

Khambata, A.J.; *"Microprocesadores/microcomputadores. Arquitectura, software y sistemas"*, Gustavo Gili, Barcelona (1987).

Macías, J.J.; *"Control informático de un digestor anaerobio: diseño y programación"*, Trabajo de Licenciatura, Dpto. de Química Técnica, Universidad de La Laguna, La Laguna, Tenerife (1986).

Moreno, L.; *"Apuntes de Electrotecnia y Automática"*, Licenciatura en Química (Industrial), 4^{o} Curso, Universidad de La Laguna, La Laguna, Tenerife (1988).

CAPITULO 8

COMUNICACIONES EXTERNAS

8.1 LAS COMUNICACIONES EN UN ORDENADOR

8.2 TRANSMISIONES EN SERIE

8.3 IDENTIFICACION DE LA INFORMACION: PROTOCOLOS

8.3.1 PROTOCOLOS ASINCRONOS

8.3.2 PROTOCOLOS SINCRONOS

8.4 LA INTERFASE RS-232-C

8.1 LAS COMUNICACIONES EN UN ORDENADOR

La dificultad de transferir información entre los distintos componentes físicos de un ordenador depende fuertemente de la distancia física que los separa. Como se ha visto hasta ahora, la transferencia de información interna de un ordenador se lleva a cabo mediante grupos de conductores eléctricos, llamados *buses*, que permiten una transmisión de los datos en paralelo. Sin embargo, si se desea hacer una transmisión al exterior del sistema, salvando distancias comprendidas entre varios metros y miles de kilómetros, es necesario utilizar otros medios físicos, como cables eléctricos, fibras ópticas, radioenlaces, etc., a través de los cuáles es necesario transferir los datos en serie.

Como ya se ha indicado anteriormente, las comunicaciones en serie son más baratas, fiables y simples de controlar que las comunicaciones en paralelo, por lo que son las que se utilizan en las *redes de enlace* (*networks*), formadas por diferentes ordenadores o componentes de un sistema, separados por grandes distancias.

8.2 TRANSMISIONES EN SERIE

El diagrama de bloques de una transmisión en serie se muestra en la Figura 8.1, analizándose a continuación la misión de cada uno de sus componentes.

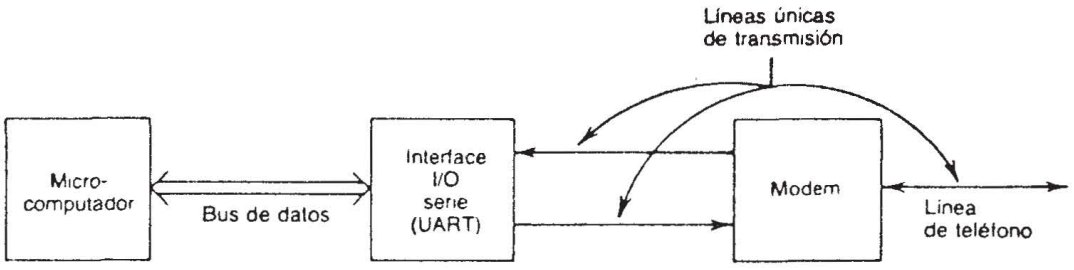
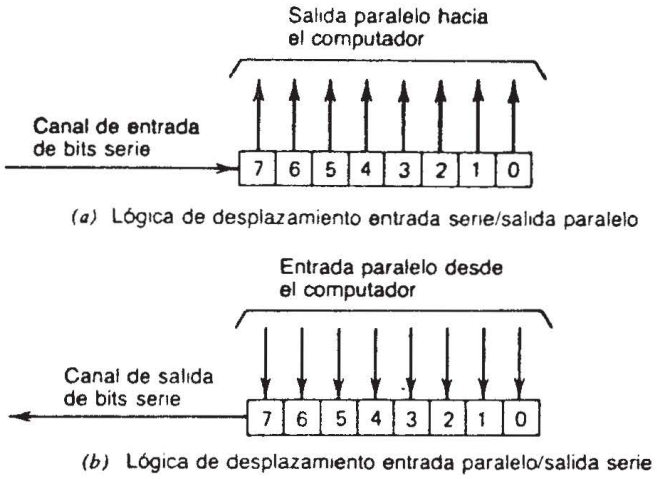


Figura 8.1
Diagrama de bloques de una transmisión en serie

La *interfase E/S serie* es el elemento que permite convertir los datos en paralelo a datos en serie, y viceversa. Se trata de un puerto de E/S programable, ya estudiado con anterioridad, que además de convertir los

datos, establece los niveles de tensión adecuados y sincroniza y espacia las señales.

La conversión serie/paralelo se puede realizar fácilmente mediante un registro de desplazamiento con entrada serie y salida paralelo, tal como se muestra en la **Figura 6.2 (a)**. Igualmente, para hacer la operación inversa, sólo se necesita un registro de desplazamiento con entrada paralelo y salida serie, como se puede observar en la **Figura 8.2 (b)**.



(a) Lógica de desplazamiento entrada serie/salida paralelo

(b) Lógica de desplazamiento entrada paralelo/salida serie

Figura 8.2

Conversiones serie/paralelo

Ahora bien, los niveles de tensión que vienen de un periférico no siempre son compatibles con los niveles requeridos por los circuitos del ordenador, por lo que deben ser normalizados. Ello se logra mediante la utilización de un biestable, que puede diseñarse de tal forma que la tensión de salida tenga un nivel totalmente compatible con el requerido por los circuitos del ordenador.

Después de esta normalización, las señales siguen siendo únicamente niveles (no señales binarias). Se pueden obtener dígitos binarios fácilmente, superponiendo los pulsos de reloj a los niveles de la señal. Esto se realiza haciendo entrar los niveles de señal en una de las dos entradas de una puerta AND y los pulsos de reloj por la otra, como se puede observar en la Figura 8.3. Este circuito no sólo produce señales binarias, que el ordenador puede utilizar, sino que además las sincroniza con el reloj del sistema.

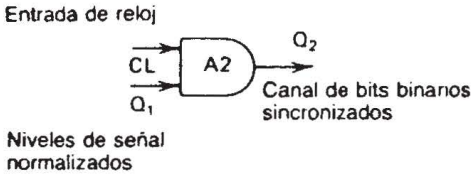


Figura 8.3
Conversión nivel/bit y sincronización

El *modem* (MO*dulador* - DEM*odulador*) es un dispositivo que transforma las señales digitales que utiliza el ordenador internamente en señales analógicas, utilizadas en la transmisión externa a largas distancias de la información, y viceversa. Si las distancias de transmisión son cortas, la información se puede transmitir mediante pulsos digitales contenidos en una *corriente continua* (*Direct Current*). Sin embargo, a medida que se van transmitiendo los pulsos, éstos se van distorsionando debido a la capacidad de la línea, interferencias externas (ruido) y otros fenómenos. El grado de distorsión aumenta con la longitud de la línea y la frecuencia de los pulsos, hasta que en cierto momento los pulsos se hacen irreconocibles y se producen errores en la transmisión. Por ello, para transmitir información a larga distancia es mucho más eficaz utilizar *corriente alterna* (*Alternate Current*).

Así pues, se genera una señal alterna adecuada, llamada *portadora* (*carrier*) y su frecuencia se modifica o *modula* (*modulated*) para producir dos tipos distintos de señales alternas que representan los valores binarios 0 y 1. El modem tiene esta misión y la inversa, para lo cual utiliza un conversor D/A para modular y un conversor A/D para demodular.

La Figura 8.4 ilustra una técnica común de modulación, denominada *modulación de frecuencia* (*Frecuency Modulation*). La onda portadora es una señal senoidal, que puede tener dos frecuencias, f_o, que representa el bit 0, y f_1, que representa el bit 1. Si se utiliza una línea telefónica normal para realizar la transmisión, se suele elegir f_o = 1,5 kHz y f_1 = 2,0 kHz. Las ondas de estas frecuencias pueden ser oídas como dos sonidos puros de distinto tono.

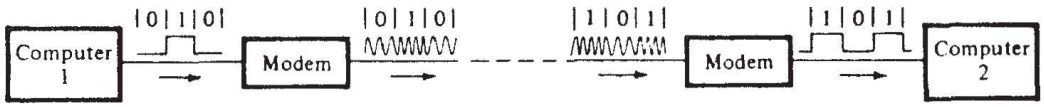


Figura 8.4
Modulación de frecuencia de señales digitales

La *línea telefónica* es el elemento del sistema de comunicaciones que permite la conexión física entre varios ordenadores o diferentes subsistemas del mismo ordenador, físicamente separados. La velocidad con que se transmite la información a través de la línea se suele expresar en *baudios* (*bauds*), o bits/s y suele variar entre 110 y 9.600. Para comunicaciones lentas se suele operar a 300 baudios, aunque actualmente se utilizan líneas que permiten operar a 1.200 ó 2.400 baudios. Las líneas especiales de transmisión de datos permiten incluso la transmisión a 9.600 baudios.

Por otro lado, la conexión física entre dos sistemas en comunicación externa puede hacerse de tres formas:

* *Unidireccional monosentido* (*simplex*): permite la transmisión en un solo sentido.
* *Unidireccional bisentido* (*half-duplex*): permite la transmisión en ambos sentidos, pero no simultáneamente.
* *Bidireccional* (*full-duplex*): permite la transmisión simultánea en ambos sentidos.

8.3 IDENTIFICACION DE LA INFORMACION: PROTOCOLOS

Cuando los dígitos binarios se transfieren en paralelo, éstos se envían simultáneamente en forma de palabras o bytes completos. Cada bit a transferir pasa por un conductor independiente y no existe duda sobre el principio y el fin del byte.

Sin embargo, cuando se transmiten datos en serie, la transferencia se realiza mediante un conductor único, bit a bit. Así, cuando entra una serie continua de palabras o bytes, la lógica de recepción no puede detectar cuándo termina una palabra y comienza otra. Además puede haber intervalos de tiempo en los que se transmiten datos no legítimos, durante los cuáles suele haber

un nivel bajo continuo de señal, que el equipo de recepción puede interpretar como 0 digitales.

Obviamente, no es suficiente sincronizar los niveles 1 y 0 con los pulsos del sistema de reloj, como se ha indicado anteriormente. Debe disponerse además de un método adecuado para reconocer cada palabra o byte, es decir, un sistema de reglas que determinen los formatos en los cuáles se puede intercambiar la información entre los diferentes sistemas. Tales reglas o conjunto de parámetros se denomina *protocolo de comunicaciones* (*communication protocol*) y permite definir la forma de transmitir la información, incluyendo su temporización, control, formato y representación.

Los protocolos para comunicaciones en serie generalmente se clasifican en dos categorías:

* *Protocolos asíncronos*: los datos sucesivos aparecen en el tren de datos en instantes arbitrarios, sin control temporal específico que gobierne los retrasos relativos entre los datos.

* *Protocolos síncronos*: cada dato correlativo en un tren continuo de datos está controlado por una señal de reloj y aparece en un instante de tiempo determinado.

A continuación se analizarán las características principales de cada uno de estos protocolos, haciendo hincapié en los más utilizados en la actualidad.

8.3.1 PROTOCOLOS ASINCRONOS

En las transmisiones asíncronas, se utiliza un protocolo general, en el que los 1 digitales están representados por señales de nivel alto y los 0 digitales por señales de nivel bajo. Si la línea de transmisión está libre, se supone que está a nivel alto. Bajo estas consideraciones, la transmisión de cada carácter se realiza según el esquema mostrado en la **Figura 8.5**.

Cada carácter comienza con un bit de inicio, 0, seguido por los 8 bits de datos del carácter, un bit de paridad (1 ó 0, dependiendo del modo de paridad seleccionado) y 2 bits de parada, 11. Cuando la línea sube a 1 debido a los bits de final, se queda a un nivel alto durante un número arbitrario de bits 1. Por tanto, el principio del próximo carácter queda indicado por la bajada entre los bits 1 y el bit 0 de inicio.

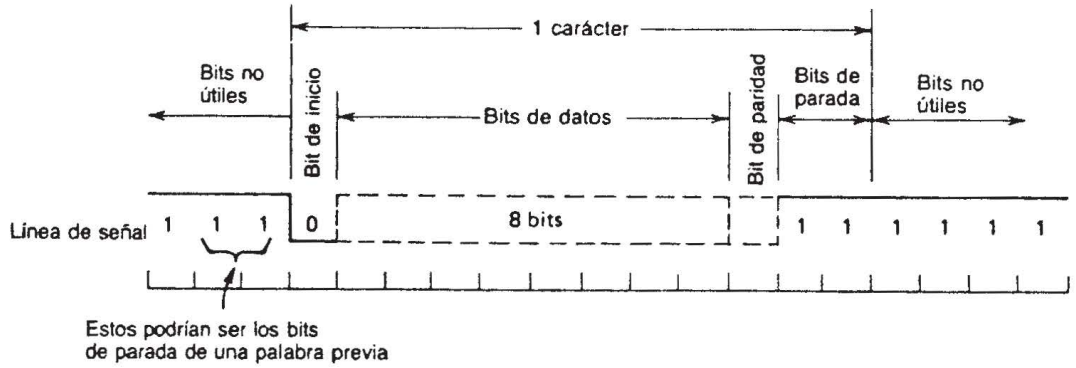


Figura 8.5

Transmisión en serie asíncrona

En las transmisiones en serie asíncronas no hay un sistema de reloj común que sincronice los trenes de datos de entrada y salida, por lo que cada sistema transmisor o receptor tiene su propio reloj. Estos dispositivos de reloj están calibrados, pero no sincronizados. Precisamente se utilizan los bits de inicio y final para sincronizar el reloj local para cada carácter. Como un carácter puede comenzar en cualquier momento, el receptor tiene que detectar este hecho con la suficiente precisión como para muestrear correctamente los 11 siguientes bits. Una vez detectado el bit de inicio, utiliza éste como una base de tiempos para muestrear los siguientes bits.

La mayoría de los receptores utilizan un reloj rápido que opera a 16 veces la velocidad de transmisión. Como ésta es en bits/s, el receptor puede determinar el comienzo de un carácter en el intervalo de 1/16 el período del bit, realizándose el muestreo en esta base de tiempos de la siguiente forma:

* Cuando aparece un bit de inicio, se inicializa un contador.

* El contador se incremente para cada señal del reloj local.

* Cuando el contador alcanza el valor 8, se ha llegado a la mitad del tiempo del bit de inicio, momento en el que éste se muestrea y se pone de nuevo a cero el contador. El arranque del contador en el centro del bit de inicio elimina la posibilidad de falsos inicios debidos a interferencias y asegura que se reconoce el próximo carácter.

* Posteriormente, cada vez que el contador llegue a 16, se muestrea la

entrada y se pone a 0 el contador, hasta que se ha muestreado el bit de paridad.

* Si el bit de paridad es correcto, se muestrean los dos bits de parada, se acepta el carácter, se almacena en una memoria intermedia (*buffer*) y se comienza de nuevo el proceso.

8.3.2 PROTOCOLOS SINCRONOS

Los enlaces síncronos utilizan un reloj para controlar la temporización carácter a carácter dentro de un bloque de caracteres. Las principales ventajas de este protocolo frente al asíncrono son las siguientes:

* El reloj sincronizado elimina la necesidad de bits de inicio y parada, por lo que se incrementa el espacio disponible para los datos.

* Sus parámetros están implícitamente orientados hacia el manejo de bloques, no de caracteres individuales, lo que permite incorporar más fácilmente la detección de errores en la comunicación.

* Las comunicaciones pueden setablecerse a mayor velocidad, por encima de los 9.600 baudios (límite de las comunicaciones asíncronas), con un límite de 500.000 baudios.

La diferencia principal de la transmisión síncrona respecto a la asíncrona estriba en el tratamiento de las señales de reloj para mantener la sincronización carácter a carácter. En este caso el receptor emplea la frecuencia de los datos que recibe para sincronizarse, en vez de establecer una frecuencia local de sincronización. Esto se consigue insertando unas palabras de control en el tren de datos, cuya distinta disposición ha determinado el establecimiento y uso de dos protocolos síncronos ligeramente diferentes:

* Protocolo BISYNC

* Protocolo HDLC

El *protocolo BISYNC* (BI*nary* SYN*chronous* C*ommunications*) se basa en uno o dos caracteres de sincronización (caracteres ASCII de control) para identificar tanto la sincronización de un bloque como la de un carácter, tal como puede observarse en la Figura 8.6.

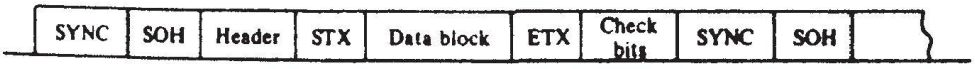


Figura 8.6
Estructura de un bloque BISYNC

Cada bloque de datos viene precedido por la palabra SYNC (ASCII 22), seguida de la palabra SOH (ASCII 1), *Start Of Header* (inicio de cabecera), que acompaña a una cabecera que puede contener una dirección fuente y destino y un mensaje numérico secuencial. En caso de un error en la recepción de un bloque de datos, el receptor transmite un mensaje de retorno al emisor con esta dirección y el número de la secuencia erróneo, lo que permite al emisor corregir el error.

El bloque de datos comienza con STX (ASCII 2), *Start of TeXt* (inicio de texto), y termina con ETX (ASCII 13), *End of TeXt* (final de texto) o con ETB (ASCII 23), *End of Text Block* (final de bloque de texto), seguido inmediatamente por una verificación del bloque enviado.

BISYNC utiliza otro carácter de control, DLE (ASCII 16 = DEL: borrar), *Data Link Escape* (escape de dato enlazador), que permite enviar un símbolo de control como dato, es decir, cuando un símbolo de control (SYNC, SOH, STX, ETX, ETB) ha de ser leído como dato, tiene que ir precedido por DLE. Esta capacidad para tratar palabras de control como tales o como datos se denomina frecuentemente *transparencia de los datos* (*data transparency*) y se utiliza con gran asiduidad.

El *protocolo HDLC* (*High-level Data-Link Control*) o control de enlace de datos de alto nivel, utiliza también una configuración especial para mantener la sincronización de los caracteres. Sin embargo, esta configuración es única, y nunca puede darse en cualquier otro lugar que no sea el comienzo de un bloque (que también es el comienzo de un carácter); se denomina *indicador* (*flag*) y es la combinación binaria 0111 1110.

Un bloque HDLC tiene la estructura mostrada en la Figura 8.7, cuyo formato es muy similar al de BISYNC en términos de la información que se transmite en un bloque, pero los campos específicos están codificados de diferente forma.

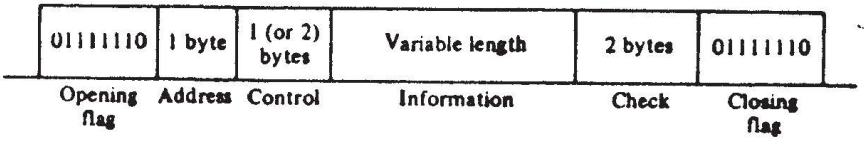


Figura 8.7
Estructura de un bloque HDLC

HDLC evita el uso de caracteres especiales para delimitar los campos, lo que simplifica la transmisión de los datos que puedan contener estos caracteres. El bloque comienza y termina con el indicador de sincronización; inmediatamente le sigue la información de direcciones y control, análoga a la utilizada por BISYNC. Como un bloque termina en el indicador, no hace falta tampoco usar un símbolo especial para identificar el final del bloque.

Un bloque de datos tampoco tiene que tener una longitud determinada, ya que cuando el receptor detecta un indicador de sincronización, los 16 bits inmediatamente anteriores son utilizados como un *verificador de redundancia cíclica* (*Cyclic Redundancy Check*) para determinar si se ha producido algún error en la transmisión previa. El siguiente bloque de datos puede utilizar el indicador de sincronización final como su indicador de inicio, pasando directamente a la información de direcciones y control.

8.4 LA INTERFASE RS-232-C

El dispositivo físico más utilizado en las comunicaciones externas de ordenadores es la interfase RS-232-C (*Recommended Standard 232-C*), construido según unas normas respecto a niveles de tensión en las líneas, velocidad de transmisión y conector a emplear, que facilita enormemente la interconexión de los aparatos que disponen de ella. Aunque se diseñó inicialmente para las conexiones por línea telefónica, vía modem, actualmente se utiliza para conectar directamente muchos elementos externos a un ordenador, incluso dos ordenadores entre sí. La norma recomienda que la longitud del cable de conexión directa entre dos dispositivos sea inferior a los 100 pies (unos 30,5 m) y describe 21 señales y un conector de 25 patas. Como las señales fundamentales de comunicación son 9, existe una versión de conector de 9

patas, cada vez más utilizada en los últimos años.

En su adaptación a través de un modem, la norma define las conexiones que se muestran en el esquema de la Figura 8.8, donde pueden observarse dos señales de tierra: una es la tierra del chasis, conectada a la malla del sistema (0) y la otra es la tierra de la señal (7), que establece el punto de referencia común para todas las señales.

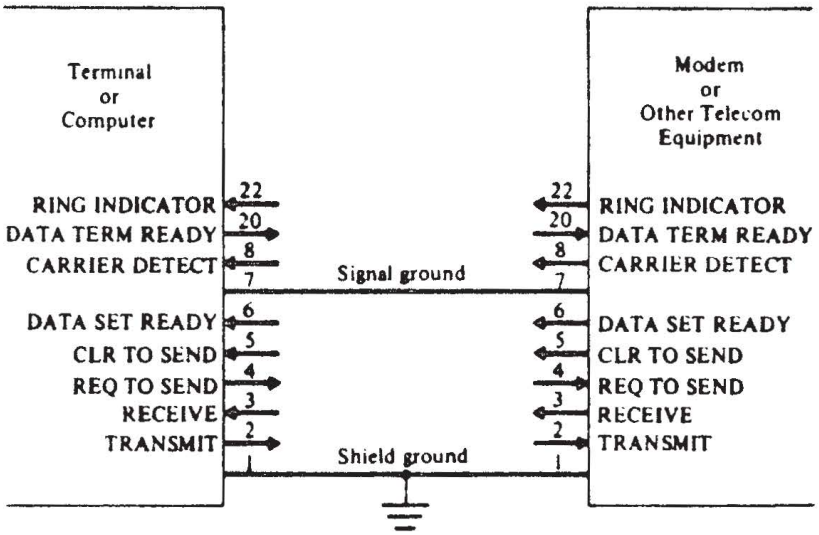


Figura 8.8

Interfase RS-232-C conectada a modem

La transmisión de datos se realiza a través de las líneas TRANSMIT (2) y RECEIVE (3). La primera va del ordenador al modem y la segunda tiene el sentido opuesto.

Las líneas REQ TO SEND (4) y CLR TO SEND (5) reflejan el protocolo de comunicación telefónica del modem en modo semi-duplex: ambas líneas son capaces de transmitir en ambos sentidos, pero no simultáneamente. Así, el ordenador manda una señal RTS cuando quiere transmitir un carácter, pero éste debe ser almacenado hasta que el modem cambia del modo recepción al modo transmisión. Cuando ésta es posible, el modem envía la señal CTS al ordenador, y la transmisión comienza. Estas dos líneas se mantienen a una tensión constante cuando la conexión es full-duplex, es decir, cuando se puede recibir y transmitir simultáneamente mediante líneas independientes.

Existen dos señales de "listo" en esta norma. DATA SET READY (6) indica

que el modem está operativo, es decir, conectado a la línea de comunicaciones; DATA TERM READY (20) indica que el ordenador es el que está operativo. Generalmente se conectan estas señales a la fuente de alimentación, con lo que se activan cuando se pone en funcionamiento el sistema. Asimismo ambas señales se envían a través de la línea externa para informar al otro dispositivo de las condiciones de operatividad.

Otras dos líneas están relacionadas con funciones telefónicas. RING INDICATOR (22) está activada durante el tiempo que se produce la señal de llamada en la línea telefónica. Cuando un puerto de E/S detecta la señal de llamada, genera una interrupción que permite iniciar al ordenador la conexión con el dispositivo que efectúa la llamada. CARRIER DETECT (8) es una señal que indica que la transmisión telefónica se está llevando a cabo. Si se pierde la comunicación, se desactiva esta señal, lo que genera una interrupción en el puerto de E/S que le indica al ordenador este hecho.

Hasta ahora se ha estudiado la conexión de un ordenador a un modem mediante la interfase RS-323-C. Sin embargo, cuando se desea conectar dos ordenadores directamente, evidentemente no se pueden contectar ambas interfases línea a línea, por lo que generalmente se entrecruzan los cables en las líneas para hacer compatibles las señales, tal como se muestra en la Figura 8.9.

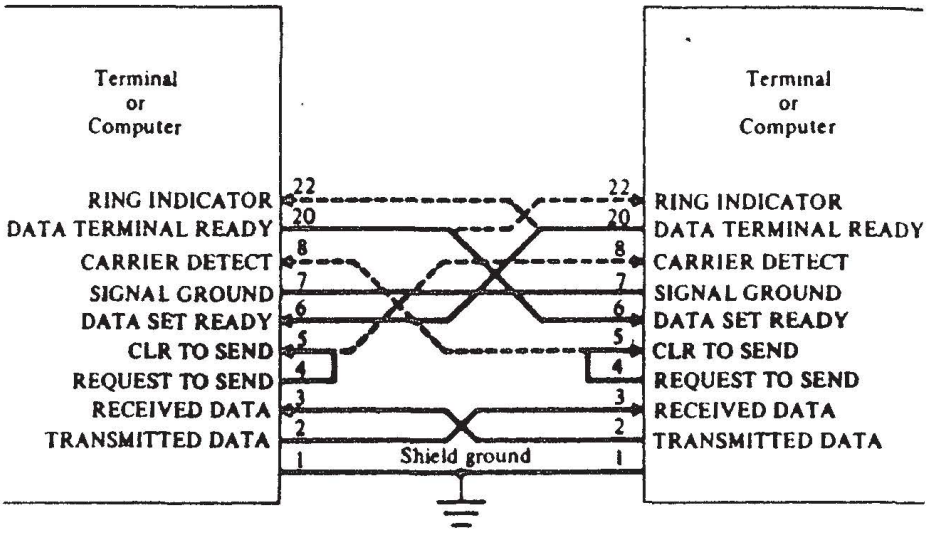


Figura 8.9

Interfase RS-232-C ordenador/ordenador

Obsérvese que las líneas TRANSMIT y RECEIVE están cruzadas, para que ambos dispositivos puedan transmitir y recibir adecuadamente.

Como en este caso se opera implícitamente en full-duplex, las líneas REQ TO SEND y CLR TO SEND ya no tienen sentido, por lo que se puentean en cada uno de los extremos. La línea discontinua muestra el uso de la señal CLR TO SEND como CARRIER DETECT en el otro extremo, ya que la presencia de REQ TO SEND es funcionalmente similar a la detección de una línea portadora en un canal de comunicación.

El último grupo de señales, DATA SET READY y DATA TERM READY están cruzadas, de tal forma que cada extremo del enlace pueda detectar la presencia de una condición de "listo" en el otro extremo. Obsérvese asimismo la línea discontinua que lleva la señal DATA TERM READY a la entrada RING INDICATOR.

Las líneas discontinuas indicadas en la **Figura 8.9** significan conexiones que se hacen a veces en la práctica, pero son mucho menos frecuentes que las que se hacen entre las señales de acoplamiento o realimentación, representadas por líneas continuas.

BIBLIOGRAFIA

Hayes, J.P.; *"Computer architecture and organization"*, McGraw-Hill, New York (1988).

Khambata, A.J.; *"Microprocesadores/microcomputadores. Arquitectura, software y sistemas"*, Gustavo Gili, Barcelona (1987).

Osborne, A.; *"Introducción a las microcomputadoras"*, Vol. 1, 2ª ed., Osborne/McGraw-Hill, Madrid (1986).

Stone, H.S.; *"Microcomputer interfacing"*, Addison-Wesley, Reading (1983).

CAPITULO 9

FUNDAMENTOS DE PROGRAMACION

9.1 EL ORDENADOR BAJO EL PUNTO DE VISTA DE LA PROGRAMACION

9.2 CONCEPTOS BASICOS DE PROGRAMACION

- 9.2.1 PLANTEAMIENTO DEL PROBLEMA
- 9.2.2 DISEÑO DEL PROGRAMA
- 9.2.3 CODIFICACION DEL PROGRAMA
- 9.2.4 VERIFICACION Y PRUEBA
- 9.2.5 DESARROLLO DE LA DOCUMENTACION

9.3 LENGUAJES DE PROGRAMACION

- 9.3.1 LENGUAJE ENSAMBLADOR
- 9.3.2 LENGUAJES DE ALTO NIVEL

9.4 PROGRAMACION DE SISTEMAS

9.5 PROGRAMAS DE APLICACION

9.1 EL ORDENADOR BAJO EL PUNTO DE VISTA DE LA PROGRAMACION

Hasta ahora se ha estudiado el ordenador desde el punto de vista físico, ya que son sus componentes físicos los que realmente ejecutan una tarea. Sin embargo, para que ésta se lleve a cabo correctamente, es necesario suministrarle al ordenador ciertas órdenes, es decir, es imprescindible definir un conjunto de actividades asociadas al desarrollo y operación de un ordenador, que son independientes de sus componentes físicos. Este conjunto de actividades se conoce normalmente por *programación* (*software*), y constituye una parte inmaterial, pero imprescindible, del ordenador, por lo que a continuación se acomete su estudio de forma genérica.

9.2 CONCEPTOS BASICOS DE PROGRAMACION

El desarrollo de la programación de un ordenador requiere un cierto número de procedimientos, actividades y tareas, que pueden esquematizarse en una secuencia de cinco etapas principales que, a su vez, requieren ciertos refinamientos antes de pasar a la siguiente. El esquema general de este desarrollo se muestra en la Figura 9.1, y las etapas que se analizarán seguidamente son las siguientes:

* Planteamiento del problema
* Diseño del programa
* Codificación del programa
* Verificación y prueba
* Desarrollo de la documentación

9.2.1 PLANTEAMIENTO DEL PROBLEMA

Antes de comenzar la programación, se debe entender claramente el problema a resolver y las distintas tareas a realizar. Esta es quizá la etapa más importante del desarrollo de la programación y el punto de inicio de todas las actividades siguientes.

Ha de definirse el problema a resolver, relacionar las distintas tareas a realizar y describir las especificaciones. Estas deben ser precisas, claras y concisas, debiendo contener todas las entradas y salidas, así como sus limitaciones y condicionantes. Es necesario también incluir procedimientos de manipulación de errores, con objeto de asegurar la fiabilidad del proceso.

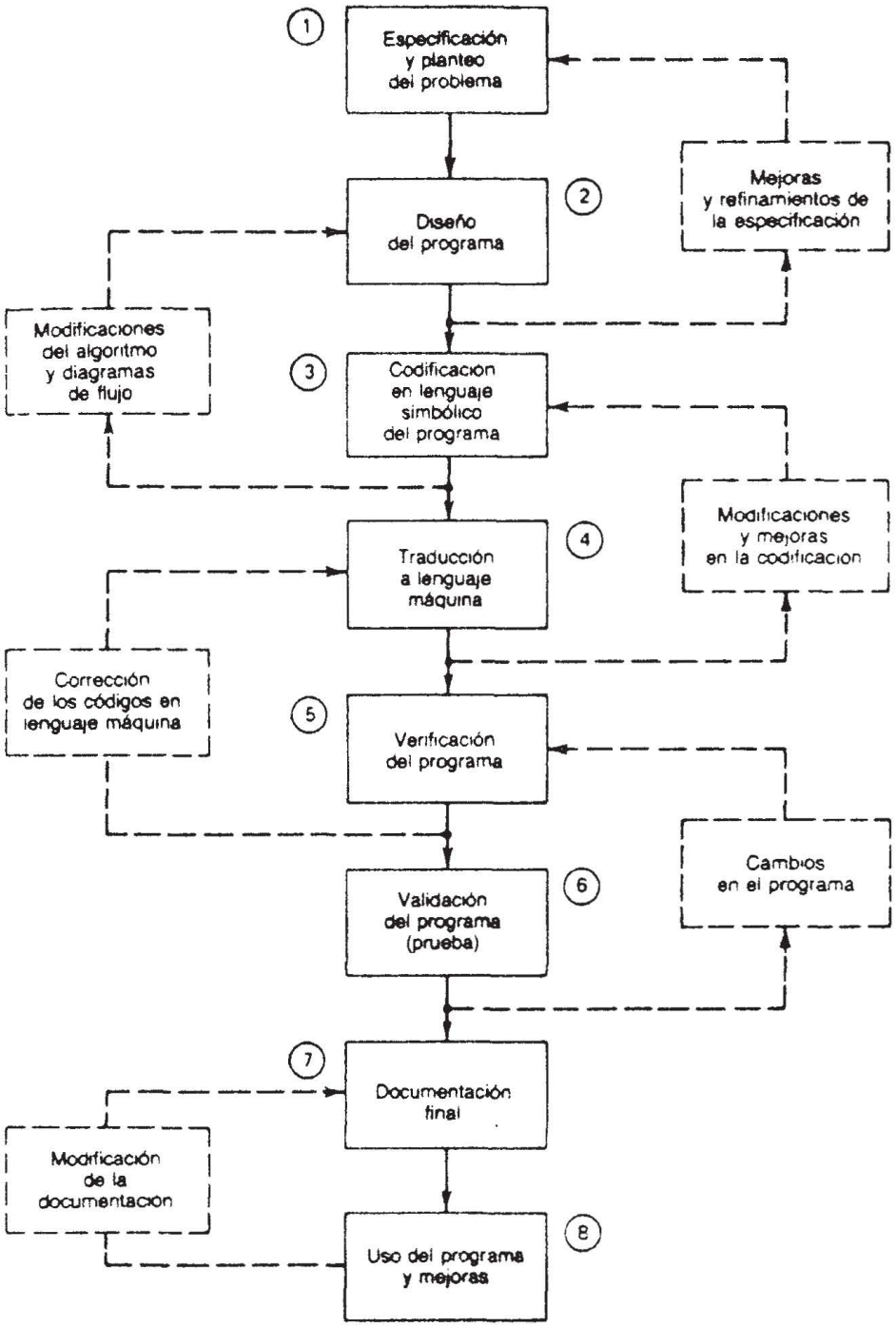


Figura 9.1

Etapas del desarrollo de una programación

9.2.2 DISEÑO DEL PROGRAMA

En esta etapa se acomete la resolución del problema planteado, estableciendo un *algoritmo* (*algorithm*) o expresión detallada, paso a paso y a

prueba de errores, de un procedimiento que tiene en cuenta todas las posibles situaciones, incluyendo, por supuesto, las posibles excepciones. El algoritmo es, pues, un procedimiento perfectamente definido que conduce a una solución en un número finito de pasos.

En este punto se suele utilizar la técnica de los *diagramas de flujo* (*flow diagrams*). Un diagrama de flujo es una técnica de análisis de sistemas que permite representar gráficamente los procedimientos o etapas principales del proceso. Esta herramienta es de gran utilidad en programación, para pasar de la definición del problema al diseño del programa de una forma lógica y sistemática.

Los símbolos utilizados en un diagrama de flujo pueden representar procedimientos o etapas del proceso (Figura 9.2) o el equipo físico implicado en un sistema determinado (Figura 9.3). En un diagrama de flujo no se especifica cómo se lleva a cabo un proceso, sino qué y cuándo se hace y quién lo hace, dentro de la operación de un ordenador. Normalmente se dibuja durante el desarrollo del programa y constituye una parte importante de la fase de documentación del proceso de programación.

Por otra parte, los diagramas de flujo pueden desarrollarse a tres niveles diferentes dentro del proceso de programación:

* Nivel de concepto
* Nivel de algoritmo
* Nivel de instrucción

El diagrama de flujo a *nivel de concepto* utiliza simplemente cajas rectangulares unidas por flechas o líneas de flujo. Sólo se representan las principales etapas implicadas en la solución del problema y no se indican detalles ni puntos de decisión.

El diagrama de flujo a *nivel de algoritmo* representa sólo la lógica implicada en la resolución del problema, con ayuda de los símbolos de las Figuras 9.2 y 9.3. Como se puede llegar a la misma solución de un problema utilizando diferentes secuencias de procedimientos, son posibles diferentes algoritmos para obtener una misma solución.

El diagrama de flujo a *nivel de instrucción* se construye teniendo en cuenta el microprocesador, por lo que previamente se debe conocer el juego de instrucciones que va a utilizar el ordenador. Esta información, más detallada

que la obtenida de los niveles anteriores, permite suministrar al programador un conjunto detallado de procedimientos, a partir de los cuáles se puede codificar el programa. Esto hace que el diagrama de flujo a nivel de instrucción sea diferente para cada microprocesador, aunque resuelva el mismo problema.

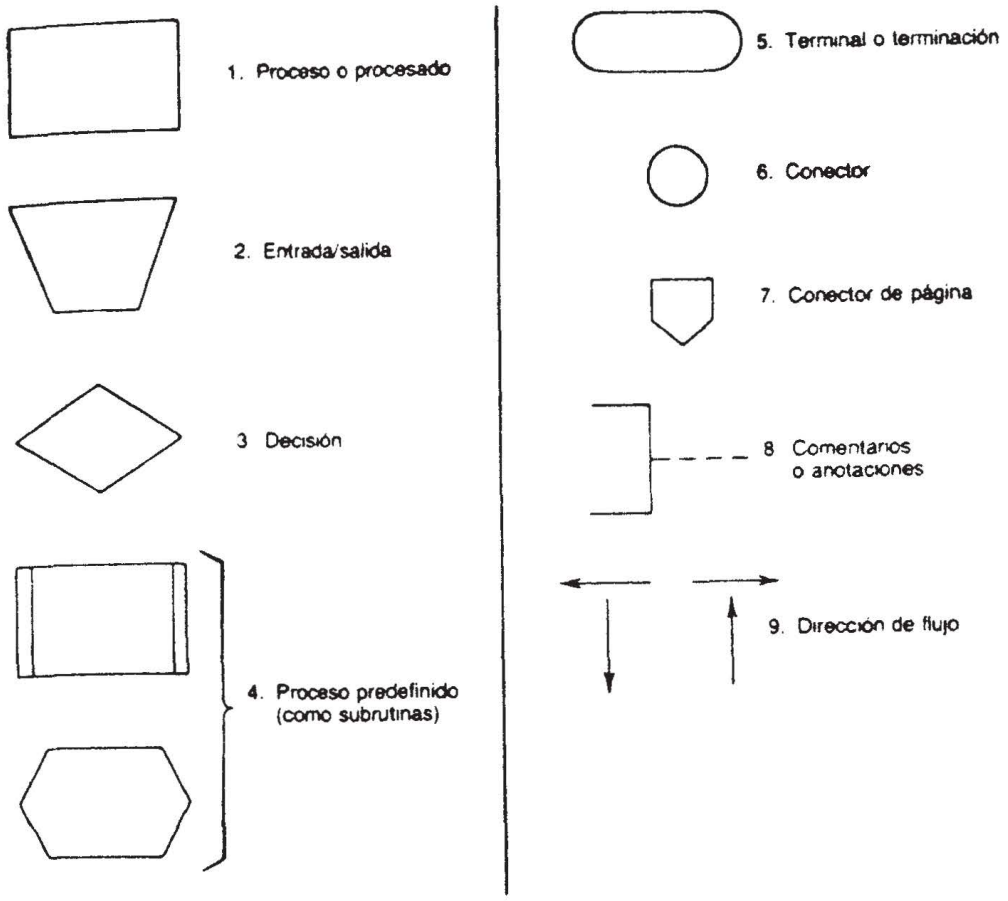


Figura 9.2

Símbolos de diagrama de flujo de programa

9.2.3 CODIFICACION DEL PROGRAMA

La codificación es simplemente el proceso de convertir las acciones indicadas en el programa, en instrucciones que den lugar a operaciones del ordenador. La programación de un ordenador puede desarrollarse utilizando un cierto lenguaje de programación que, como se verá más adelante, puede ser desde el más simple lenguaje de máquina consistente en 0 y 1, a cualquier lenguaje de alto nivel. Independientemente del lenguaje usado, todos ellos

tienen unos grupos básicos de instrucciones que realizan ciertos procedimientos, comentándose seguidamente los más importantes.

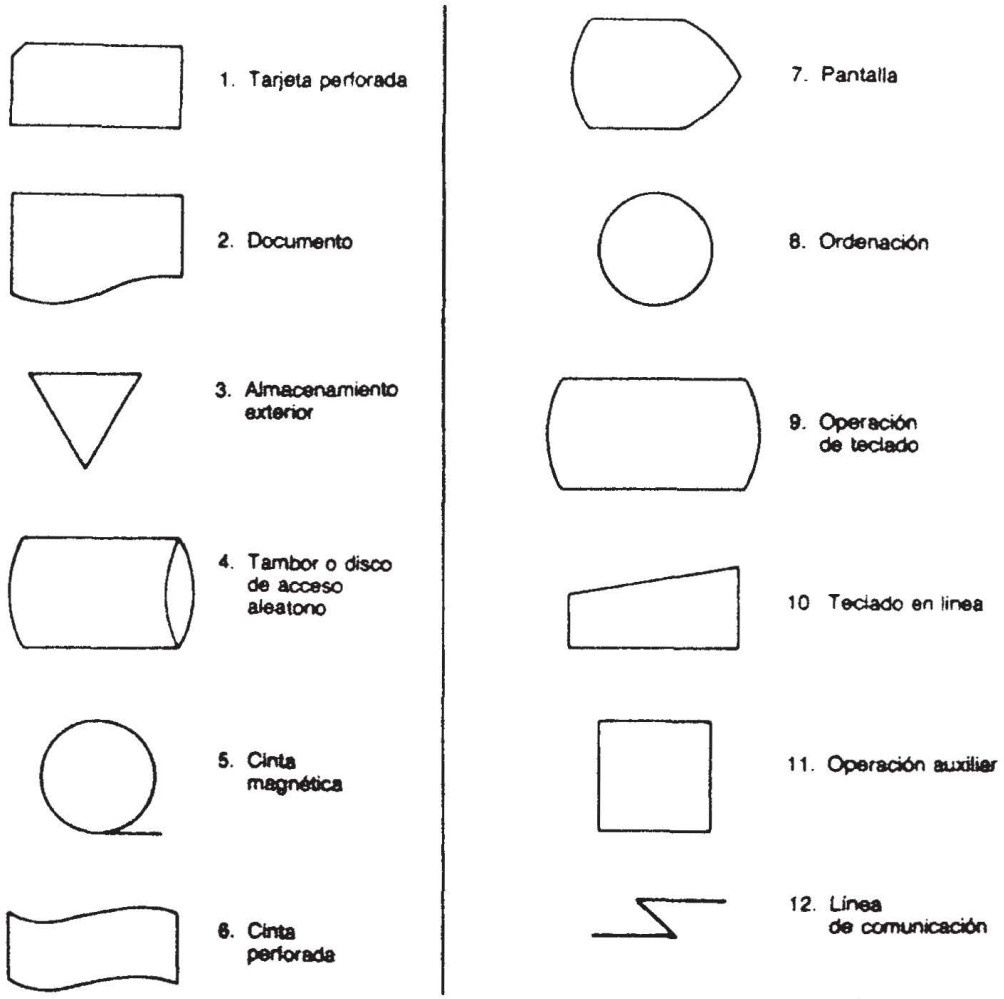


Figura 9.3

Símbolos de diagramas de flujo de sistema

* ***Instrucciones descriptivas y de transferencia de datos***: Se utilizan estas instrucciones para el movimiento de datos dentro del ordenador bajo control del programa; asignan las fuentes y los destinos de los movimientos de datos, tales como los registros y contadores de la CPU, la memoria y los distintos dispositivos de E/S.

* *Instrucciones de operación y manipulación de datos*: Estas instrucciones realizan ciertas funciones específicas, como cálculos u operaciones lógicas con los datos que se transfieren en el ordenador, considerándose dentro de este grupo la mayoría de las intrucciones del

programa.

* *Instrucciones condicionales o de decisión*: Se trata de instrucciones que hacen posible realizar ciertas funciones específicas sólo cuando se dan unas condiciones preestablecidas; la decisión la toma el ordenador al procesar los datos de que dispone y aplicar las condiciones.

* *Instrucciones de transferencia de control*: Este grupo de instrucciones interrumpe la instrucción que se está ejecutando, transfiriendo el control del programa a otra instrucción situada en algún otro lugar del mismo; a veces el control se devuelve posteriormente a la instrucción que fué interrumpida, pero este retorno del control no es necesariamente automático.

* *Instrucciones iterativas o de bucle*: El propósito de estas instrucciones es realizar ciclos de un cierto conjunto de instrucciones de forma repetitiva; el número de iteraciones puede estar predeterminado e indicado en el programa o bien depender de algún criterio relacionado con los datos que se procesan, en cuyo caso ha de tomarse la decisión de realizar o no el bucle siguiente.

* *Instrucciones varias*: Existen instrucciones que ocasionalmente no encajan en ninguna de las cinco categorías precedentes, pero que se requieren en alguna aplicación; tales instrucciones se agrupan en este apartado.

Finalmente, si la codificación del programa está escrita en un lenguaje de alto nivel, es necesario traducirlo al lenguaje máquina, que es el único que entiende la CPU. Aunque este proceso puede llevarse a cabo manualmente, siempre se hace con ayuda de otro programa "traductor", cuyas características dependen del lenguaje utilizado y que serán estudiadas más adelante.

9.2.4 VERIFICACION Y PRUEBA

Todos los programas, aunque se hayan realizado muy cuidadosamente, contienen algunos errores que han de localizarse y corregirse. Aunque esta fase del desarrollo de la programación requiere mucho tiempo, actualmente se cuenta con muchas herramientas, tanto físicas (analizadores lógicos) como lógicas (depuradores), que permiten simplificar considerablemente el proceso de verificación.

Una vez localizados y corregidos los errores de programación y traducción, el programa ha de ser probado, para confirmar si cumple los

requerimientos especificados en el planteamiento del problema. Este proceso requiere ejecutar el programa con varias entradas, comprobando las salidas y comparándolas con las esperadas.

9.2.5 DESARROLLO DE LA DOCUMENTACION

En todas las etapas del desarrollo de la programación se genera una cierta documentación. Después de haber verificado y probado el programa, se prepara una documentación final para el usuario, que también puede ser usada por los diseñadores, si posteriormente se requiere una modificación o extensión del programa original. Esta documentación incluye normalmente listados de programas, especificaciones, requerimientos, diagramas de flujo y mapas de memoria.

9.3 LENGUAJES DE PROGRAMACION

Se define un *lenguaje de programación* (*programming language*) como el sistema de representación y comunicación de datos e información entre el ser humano y el ordenador, que contiene un conjunto de caracteres bien definidos y unas reglas para combinarlos en forma de expresiones e instrucciones. Como es sabido, un ordenador ejecuta sus diversas funciones merced a una secuencia de instrucciones que, en la forma en que realmente opera con ellas la CPU, constan de una serie de números binarios. Este lenguaje, en el que cada instrucción se especifica en forma binaria, lista para cargarse en la memoria de programa y ejecutarse, recibe el nombre de *lenguaje de máquina* (*machine language*) y en su sentido más estricto se refiere a las instrucciones escritas en *código de máquina* (*machine code*), sistema de codificación adoptado en el diseño físico de un ordenador para representar su repertorio de instrucciones, y que permite conectar una parte del microprocesador con otra.

Para la programación en lenguaje de máquina se requiere conocer las partes del ordenador que ejecutan funciones de manipulación de datos (registros) y los detalles del repertorio de instrucciones. Asimismo es necesario conservar una relación precisa de todas las posiciones de almacenamiento y asegurar que todos los saltos de programa van al lugar apropiado. Además, hay que introducir el programa en el ordenador en código

binario, como muestra el ejemplo de la Figura 9.4.

Programa como una matriz binaria	Versión hexadecimal del programa
10011100	9C
00001010	0A
00110000	30
01000000	40
10011100	9C
00001010	0A
00110001	31
10000000	80
01100000	60

Figura 9.4
Programa en lenguaje de máquina

Los inconvenientes que presenta la programación en lenguaje de máquina son innumerables. En primer lugar, la probabilidad de cometer errores trabajando con dígitos binarios es muy alta; las adiciones, correcciones y alteraciones se ven sometidas a los mismos problemas y son muy lentas, particularmente si se trata de un programa largo. Además, los programas escritos en lenguaje de máquina son específicos de cada microprocesador, por lo que si se desea utilizarlos en otra máquina será necesario proceder a su reescritura.

Un programa escrito en lenguaje de máquina se conoce como *programa objeto* (*object program*), y es apto para que el ordenador lo comprenda directamente, por lo que tiene la forma precisa para su procesamiento.

El primer paso para el desarrollo de los lenguajes de ordenador tuvo lugar cuando se advirtió que la mayor parte de las comprobaciones detalladas de las direcciones, posiciones de memoria, bifurcaciones, etc., se podían realizar mediante un programa de ordenador. Ello hizo posible elevar el nivel del lenguaje y permitir así escribir un programa en un lenguaje más o menos inteligible por el programador, denominado *programa fuente* (*source program*). Se diseñaron así diferentes programas de traducción de los programas fuente a programas objeto, ejecutables por la máquina, de diferente nivel de complejidad, estableciéndose una jerarquía de los lenguajes de programación como la mostrada en la Figura 9.5, y en la que el lenguaje de máquina ocupa el nivel más bajo.

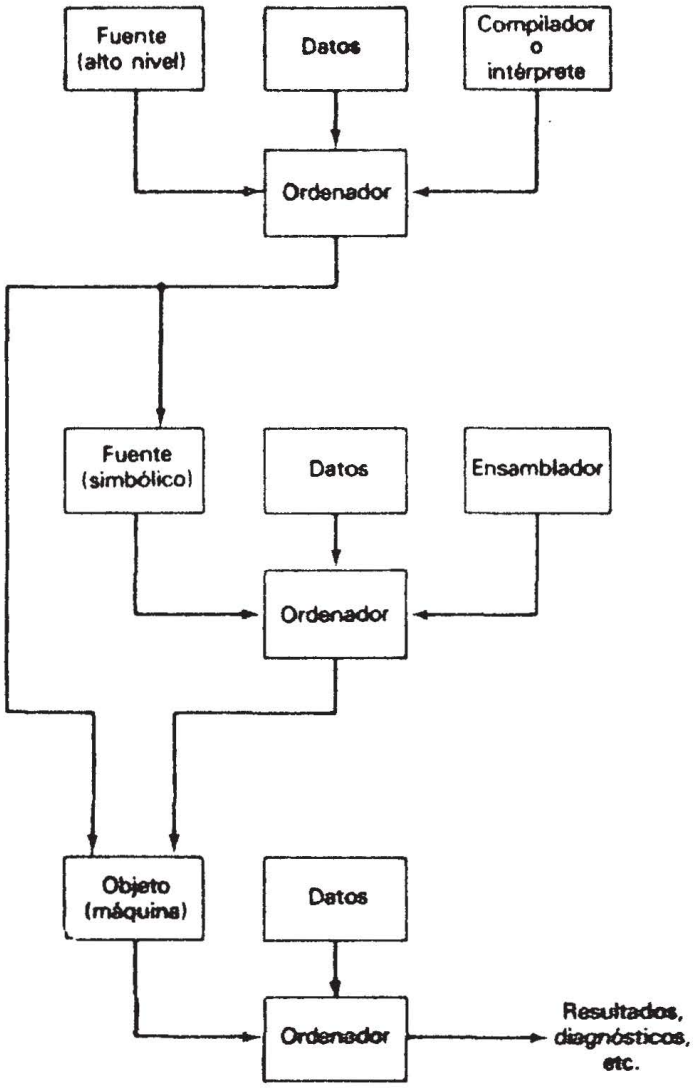


Figura 9.5

Niveles de los lenguajes de programación

Obsérvese que a cada tipo de lenguaje le corresponde un programa traductor: ensamblador, compilador o intérprete, programas que pertenecen a la denominada "programación de sistemas", que se estudiará más adelante. En este apartado se considerarán sólo los aspectos relativos al propio lenguaje que, según su nivel de complejidad, se clasificarán de la siguiente forma:

* Lenguaje ensamblador
* Lenguajes de alto nivel

9.3.1 LENGUAJE ENSAMBLADOR

El lenguaje ensamblador es un lenguaje simbólico en el que cada instrucción del repertorio del ordenador tiene asociado un símbolo o *mnemónico* (*mnemonic*), lo que permite escribir el programa completo utilizando estos mnemónicos. El conjunto de reglas que gobiernan la estructura de las instrucciones, o *sintaxis del lenguaje* (*language sintax*) viene en función del formato de la instrucción (**Figura 9.6**), que está dividida en cuatro campos:

* Campo de etiqueta
* Campo de código de operación
* Campo de argumento
* Campo de comentarios

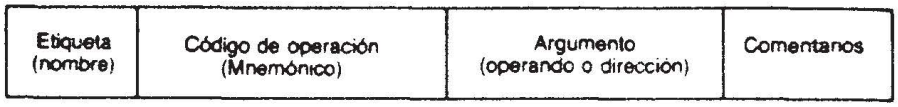


Figura 9.6
Formato de la instrucción en lenguaje ensamblador

El *campo de etiqueta* puede contener un nombre que puede ser utilizado para direccionar una instrucción. La etiqueta puede contener letras, números o algunos otros caracteres.

El *campo de código de operación* contiene el código mnemónico de la instrucción; es el más importante de la instrucción y nunca puede estar vacío. Su amplitud depende de la estructuración de caracteres de los mnemónicos de un conjunto de instrucciones, pero puede oscilar entre 3 y 10 caracteres.

El *campo de argumento* contiene el operando requerido para la ejecución de la instrucción o la dirección donde se encuentra almacenado el operando. En este caso, la dirección puede ser la dirección de memoria real o un código que identifique un cierto registro del ordenador.

El *campo de comentarios* puede contener comentarios o notas explicativas referentes a la instrucción. Estos comentarios no son ejecutados por el programa traductor, pero son muy útiles para seguir las etapas de un

programa.

Además de las instrucciones, generalmente se utilizan funciones adicionales que permiten informar al programa traductor de ciertas especificaciones que éste no podría deducir por sí mismo a partir del programa fuente. Por ello se denominan normalmente a estas funciones *pseudocódigos* o *directivas* (*directives*), y su misión engloba los siguientes aspectos:

* Generar tablas de código máquina y de memoria
* Definir símbolos y caracteres
* Asignar áreas específicas de memoria a los programas y subrutinas
* Asignar espacio de memoria para almacenar variables
* Establecer el formato del listado del programa

Finalmente, muchos ensambladores modernos tienen una función denominada *macro* (*macro*), que permite asignar un nombre a una secuencia de instrucciones previamente definidas. Este nombre define efectivamente el macro y, por tanto, es posible colocar en cualquier punto del programa la secuencia de instrucciones, utilizando este nombre como instrucción.

9.3.2 LENGUAJES DE ALTO NIVEL

Tanto el lenguaje de máquina como el lenguaje ensamblador tienden a ser específicos de un modelo concreto de microprocesador u ordenador y, por ello, es limitado, si no imposible, su uso en máquinas producidas por otro fabricante. En la actualidad se utilizan numerosos lenguajes que están orientados más a la resolución de problemas que al control de la máquina, y los programas escritos en estos lenguajes son transferibles a otras máquinas. Además, los lenguajes orientados a la resolución de problemas utilizan instrucciones que recuerdan mucho más al lenguaje humano (inglés).

Los lenguajes de alto nivel se distinguen de los de niveles inferiores en que sus instrucciones adoptan la forma de *sentencias* (*statements*) bastante complejas, que al ser traducidas al lenguaje de máquina generan varias instrucciones o bien subrutinas enteras de código máquina. Por este motivo, el programa objeto producido a partir de un lenguaje de alto nivel es mucho más largo que el producido a partir de un lenguaje de bajo nivel, ya que el programador tiene mucho menos control sobre la organización en la memoria del

programa objeto.

Por lo general, los programas escritos en lenguajes de alto nivel pueden ser leídos y comprendidos, al menos en parte, por quiénes no hayan aprendido el lenguaje. Si el lenguaje es de tipo general, el programa parece estar en un inglés sumamente artificial, pero aún reconocible; si se trata de un lenguaje matemático, el parecido será con la notación matemática.

Los lenguajes de alto nivel están diseñados para evitar la complicación y la propensión a cometer errores del lenguaje ensamblador, permitiendo al ordenador que haga tanto trabajo como le sea posible para generar las instrucciones detalladas de lenguaje de máquina. Para ello se parte de dos premisas fundamentales:

* Hacer un sumario de muchas instrucciones de lenguaje de máquina en una sola sentencia de programa

* Eliminar los detalles relativos a la forma de operar del ordenador, pero que no tienen nada que ver con el trabajo que se desea realizar

Sin embargo, el hecho de que tales lenguajes posean esta gran generalidad, que permite que sean comprendidos por los humanos, no significa que sean automáticamente comprendidos por todos los ordenadores. Todavía sigue siendo necesario diseñar un programa traductor adecuado a cada máquina en la que tenga que funcionar el programa.

No es posible aquí describir con detalle la totalidad de los diversos tipos de lenguajes y las reglas sintácticas de cada uno de ellos, por lo que se limitará el estudio a la mención de los más importantes en base a su difusión y ámbito de aplicación.

* *ADA* (en honor de ADA Augusta Lovelace, hija de Lord Byron) es un lenguaje desarrollado por el Departamento de Defensa de los EE.UU. específicamente destinado al control de sistemas en tiempo real, y en particular, para aquellos sistemas en que los ordenadores se utilizan con fines de control. Es de reciente aparición y trata de sintetizar todas las capacidades necesarias para múltiples aplicaciones.

* *ALGOL* (ALGO*rithmic* L*anguage*) es un lenguaje adaptado a problemas, concebido para uso matemático y científico, en el cuál el programa fuente proporciona un método para definir algoritmos mediante una serie de sentencias y declaraciones que conservan semejanza general con las fórmulas

algebráicas y con las frases en inglés. Inicialmente fué denominado IAL (International Algebaric Language), desarrollándose en Europa al mismo tiempo que en EE.UU. se desarrollaba el FORTRAN.

* *BASIC* (B*eginner's* A*ll-purpose* S*ymbolic* L*anguage*) es un lenguaje concebido inicialmente (finales de los años 60) para desarrollar programas en modo conversacional, es decir, de forma interactiva con el ordenador, instrucción a instrucción. Los medios proporcionados por este lenguaje son comparables a los de FORTRAN, enriquecidos con medios adicionales para la manipulación de transferencias de datos y manipulación de matrices. En razón de su sencillez y elevada potencia, es muy utilizado en ordenadores personales, aunque existen actualmente varios "dialectos" que impiden frecuentemente la ejecución de un mismo programa en diferentes ordenadores.

* *C* es un lenguaje de programación altamente estructurado, especialmente diseñado para llevar a cabo la programación de sistemas, es decir, aquellos programas que ayudan al usuario del ordenador a generar programas de aplicación, verificarlos, probarlos, modificarlos y ejecutarlos.

* *COBOL* (CO*mmon* B*usiness* O*riented* L*anguage*) es un lenguaje orientado a la gestión empresarial e inicialmente patrocinado por el Departamento de Defensa de EE.UU. Desarrollado en los años 50, alcanzó gran difusión y popularidad, siendo muy empleado en los grandes sistemas de gestión comercial. Es fácil de aprender y comprender, aunque las restricciones impuestas sobre instrucciones y métodos de realizar operaciones son muy rígidas.

* *FORTH* es un lenguaje inicialmente diseñado para el uso de astrónomos, pero que dispone de ciertas características que propiciaron su empleo en la confección de programación de sistemas. Una de sus características fundamentales es que los procedimientos contruidos por el usuario pueden formar parte efectiva del propio lenguaje, como si fueran ampliaciones del mismo, lo que proporciona un lenguaje ampliable a partir de una estructura básica, cuya ejecución es muy eficiente.

* *FORTRAN* (FOR*mula* TRAN*slation*) es un lenguaje adaptado a problemas, para uso matemático y científico, en el cual el programa se redacta mediante una combinación de fórmulas algebráicas y sentencias en inglés, de estructura normalizada, pero legible.

* *LISP* (LIS*t* P*rocessor*) es un lenguaje concebido para la manipulación de datos no numéricos, que se utiliza especialmente en aplicaciones de inteligencia artificial, es decir, la utilización del ordenador con el fin de que realice operaciones análogas a las facultades humanas de aprendizaje y adopción de decisiones.

* *LOGO* es un lenguaje de programación ideado a mediados de los años 60 con el intento de aproximar el ordenador a los niños y desarrollar el concepto de "enseñanza asistida por ordenador". Tiene incorporado un tratamiento de gráficos para lo cual, mediante órdenes del lenguaje, simula el movimiento de una tortuga, que en sus desplazamientos va dibujando líneas sobre la pantalla. Aunque es un lenguaje bastante potente, sus aplicaciones principales se encuentran en el campo educativo.

* *PASCAL* es un lenguaje desarrollado a finales de los años 60, inspirado en ALGOL, con fines didácticos. Por ser un lenguaje altamente estructurado, provisto de características algorítmicas diseñadas para lograr una rápida ejecución del programa resultante, se puede utilizar en gestión o cálculo científico. Esta potente estructura ha hecho que actualmente sea muy utilizado en los ordenadores personales.

* *PL/1* (P*rogramming* L*anguage* 1) es un lenguaje desarrollado para combinar los rasgos propios de los lenguajes de gestión (como COBOL) y de los lenguajes científicos (como ALGOL). Desarrollado originariamente por la empresa IBM, tiene bastante implantación en el sector de los grandes ordenadores.

* *PROLOG* (PRO*gramming* LOG*ic*) es un lenguaje de gran utilidad en el campo de la inteligencia artificial. Permite crear programas capaces de sacar conclusiones, utilizando cláusulas similares a las utilizadas en Lógica.

9.4 PROGRAMACION DE SISTEMAS

La *programación de sistemas* (*system software*) es el conjunto de programas que ayudan al usuario a generar programas de aplicación (programas destinados a efectuar tareas de usuario, con frecuencia escritos por él mismo), verificarlos, probarlos, modificarlos y ejecutarlos. Estos programas están escritos generalmente por el fabricante del ordenador utilizando lenguaje ensamblador o de alto nivel, y pueden ser utilizados por distintos

usuarios para diferentes programas de aplicación. La programación de sistemas se puede esquematizar en diferentes tipos de programas, cuyos objetivos se estudian con cierto detalle a continuación:

* Editores de textos
* Ensambladores
* Compiladores
* Intérpretes
* Cargadores
* Depuradores
* Enlazadores y cargadores de bibliotecas
* Sistemas operativos

9.4.1 EDITORES DE TEXTOS

El *editor de textos* (*text editor*) o simplemente *editor*, es un programa que le facilita a un operador humano la tarea de preparación de textos o datos a introducir en un ordenador, o a recoger de él, así como la realización de correcciones o modificaciones en las instrucciones de un programa, adiciones o supresiones de las mismas o cambios de secuencias. El editor hace posible retener el programa o los archivos de datos en la memoria del ordenador y modificarlos, especificando los cambios a realizar, para lo cual se reserva una zona de la memoria.

Las líneas de instrucciones o texto se introducen en el ordenador mediante un dispositivo de E/S (generalmente, el teclado) y se almacenan en la memoria por medio del editor. Para ayudar a identificar cada instrucción o línea de datos, el editor suele asignar un número a cada línea y corrige automáticamente los números de líneas si se añaden o retiran líneas de datos. El editor puede disponer además de recursos para efectuar una revisión de la sintaxis de las sentencias del lenguaje de programación que se esté utilizando.

Finalmente, el programa editado se puede listar en una impresora o grabar en un soporte magnético.

9.4.2 ENSAMBLADORES

Se denomina *ensamblador* (*assembler*) a un programa traductor que

convierte un programa fuente escrito en lenguaje ensamblador en el código de lenguaje de máquina correspondiente, es decir, crea un *programa objeto* (*object program*) o *módulo objeto* (*object module*), almacenable en un soporte magnético y cargable directamente en la memoria del ordenador para su ejecución. Los objetivos que permite alcanzar un programa de este tipo se pueden resumir en los siguientes puntos:

* Permite al programador escribir un programa en lenguaje ensamblador, utilizando los mnemónicos y símbolos suministrados por el fabricante de un ordenador en particular.

* Es un método para que el propio ordenador traduzca las instrucciones escritas en lenguaje ensamblador (fuente) en instrucciones útiles para el ordenador, en código binario (objeto).

* Asigna una dirección absoluta de memoria a cada instrucción y a cada palabra de datos que, en el programa fuente, estén identificadas por una etiqueta.

* Permite disponer de un método de verificación que determine las posible incongruencias en el programa fuente.

El ensamblador convierte el programa fuente en otro en elenguaje de máquina, traduciendo el primero paso a paso, es decir, para cada instrucción de programa fuente suministra una línea de instrucción en lenguaje de máquina. Durante este proceso de traducción, el ensamblador asigna a cada etiqueta una dirección de memoria, absoluta o relativa, y genera una lista de instrucciones en lenguaje de máquina, o programa objeto.

9.4.3 COMPILADORES

Los *compiladores* (*compilers*) son programas que traducen programas fuente escritos en lenguajes de alto nivel, a programas en lenguaje de máquina. Al contrario que los ensambladores, los compiladores generan varias instrucciones en lenguaje de máquina por cada instrucción en lenguaje fuente. De esta forma, el programa escrito en un lenguaje de alto nivel se puede utilizar en distintas máquinas, ya que el compilador permite formular programas fuente sin un conocimiento preciso de la arquitectura del ordenador. Su principal inconveniente es que son programas complejos y largos, pero las disponibilidades actuales de memoria hacen que sean las

herramientas más utilizadas para generar programas ejecutables a partir de programas fuente de alto nivel.

9.4.4 INTERPRETES

Los ensambladores y los compiladores traducen el programa fuente en un programa objeto ejecutable. En general, el programa fuente se produce previamente y, a partir de él, los ensambladores o los compiladores producen el programa objeto, que se carga en la memoria de programa y se puede ejecutar. Existe, sin embargo, otra herramienta de programación, el *intérprete* (*interpreter*), que difiere de los anteriores en algunos puntos muy significativos. El principal es que el intérprete traduce y ejecuta inmediatamente cada instrucción, lo que significa que es una especie de "lenguaje interactivo" o "traducción simultánea", porque permite al usuario introducir el programa instrucción a instrucción, y ejecutarlo inmediatamente, lo que resulta de extremada utilidad durante el desarrollo del programa. El tiempo de ejecución es muy largo comparado con los ensambladores y compiladores, pero el tiempo de desarrollo del programa es mucho menor.

En efecto, la interpretación es inherentemente un proceso lento e ineficaz, pero flexible. Es lento porque la traducción se hace al mismo tiempo que se está ejecutando la tarea del programa; es ineficaz porque la traducción se hace cada vez que se repite un paso del programa (bucles); pero es especialmente flexible, porque un programa interpretado se puede ajustar, cambiar o revisar en el propio momento de su ejecución.

Obsérvese finalmente que, cuando se ejecuta un programa interpretado, lo que se ejecuta realmente es el intérprete, mientras que el programa que introduce el usuario constituye un conjunto de "datos" para el intérprete, datos que describen los pasos que se desea que realice el ordenador. Esto significa que, si el segmento de datos que tiene asignado el intérprete no es suficientemente grande, es posible que el programa de usuario no tenga cabida en él y no pueda ejecutarse. Por el contrario, un programa compilado o ensamblado puede ser tan grande como admita la memoria del sistema.

9.4.5 CARGADORES

Un programa desarrollado como módulo objeto debe ser almacenado en la memoria del ordenador para ser ejecutado. Esto se lleva a cabo mediante un pequeño programa llamado *cargador* (*loader*), que está formado por un conjunto de instrucciones que permiten a un dispositivo de entrada asignar la dirección de inicio de la memoria y asegurar que el ordenador leerá el programa y lo almacenará byte a byte. Según sus características, los cargadores pueden ser de tres tipos principales:

* Cargadores absolutos
* Cargadores reubicables
* Cargadores de lanzamiento

Los *cargadores abnsolutos* (*absolute loaders*) son programas que cargan cada instrucción del programa objeto en una posición fija y establecida, por lo que cada instrucción tiene una dirección absoluta. El cargador lee simplemente la línea de código objeto que contiene la dirección de inicio de las instrucciones y datos y carga las palabras sucesivas en posiciones de memoria consecutivas. Este programa generalmente está incluido como parte del sistema operativo, y en el popular DOS, es el que gestiona los programas ejecutables denominados con la extensión .COM. El programa que se va a ejecutar tiene que estar diseñado expresamente para que sea gestionado por un cargador absoluto.

Los *cargadores reubicables* (*relocatable loaders*) son programas que, aparte de tener las características del cargador absoluto, permiten al usuario seleccionar y especificar las posiciones de memoria en las que se deben almacenar las palabras de instrucciones y las de datos; de esta forma, el mismo programa se puede cargar en distintas zonas de la memoria sin necesidad de reensamblarlo o recompilarlo. Aunque en este caso se diseña el programa conteniendo instrucciones asignadas a posiciones de memoria consecutivas, las direcciones no son absolutas, sino relativas, es decir, no representan las verdaderas posiciones de memoria en las que está almacenado el programa. La dirección de inicio verdadera se inserta al principio del cargador reubicable, y esta dirección se añade a cada una de las posiciones de memoria asignadas en el programa, obteniéndose así las direcciones reales de las instrucciones y los datos del programa. Normalmente este cargador

también está incluido en el sistema operativo; en el DOS permite gestionar los programas ejecutables denominados con la extensión .EXE.

Los *cargadores de lanzamiento o de arranque* (*boostrap*) son programas cuya misión es controlar la carga de otros. Generalmente los cargadores de lanzamiento son los que permiten arrancar el ordenador y cargar el sistema operativo y los sistemas de transferencias de E/S, es decir, el ordenador se pone en marcha por sí mismo. Evidentemente, es necesario suministrar previamente al ordenador el cargador de lanzamiento, por lo que se suele disponer de un conjunto de instrucciones en la ROM, que tienen como misión situar en la memoria el cargador de arranque, que generalmente está almacenado en un soporte magnético (disco flexible o disco duro). Así, al conectar el ordenador, se ejecutan las instrucciones contenidas en la ROM, se carga el cargador de lanzamiento desde el soporte magnético, y aquél a su vez carga el sistema operativo y el sistema de transferencias de E/S, dejando el ordenador listo para ejecutar los programas de aplicación.

9.4.6 DEPURADORES

Los programas escritos por el ser humano tienen tendencia al error, por lo que antes de que un programa se pueda poner en servicio productivo se debe depurar. Recibe el nombre de *depuración* (*debugging*) la técnica de detección, diagnóstico y corrección de los errores que pueden darse en los programas. Los *depuradores* (*debuggers*) son, pues, programas que se utilizan como herramientas en la depuración de otros.

El programa depurador se carga en la memoria junto con el programa objeto a depurar. Si se producen errores durante la ejecución, se activa el depurador para identificarlos, localizarlos y corregirlos. Generalmente, el depurador incluye varias funciones para ayudar al programador a eliminar los defectos de un programa, las más comunes de las cuáles son las siguientes:

* Mostrar el contenido de un registro: permite inspeccionar el contenido de los registros clave de la CPU, durante la ejecución de un programa objeto.

* Mostrar y reemplazar el contenido de la memoria: permite acceder al contenido de una cierta posición de memoria, que puede ser modificada o reemplazada.

* Volcar la memoria: permite imprimir el contenido de un grupo especificado de posiciones de memoria.

* Almacenar el contenido de la memoria: permite un volcado de memoria a un dispositivo de almacenamiento no volátil.

* Establecer puntos de pausa: permite colocar en el programa puntos de interrupción de su ejecución, para poder examinar el contenido de algunos registros de la CPU, posiciones de memoria o puertos de E/S.

* Desensamblar: permite realizar el proceso contrario al ensamblado, es decir, leer el contenido de la memoria de programa, en lenguaje de máquina, y traducirlo a los mnemónicos del lenguaje ensamblador.

9.4.7 ENLAZADORES Y CARGADORES DE BIBLIOTECAS

Cuando se diseña un programa de cierta envergadura, generalmente se construye éste haciendo uso de subrutinas. Como se sabe, las subrutinas son fragmentos del programa que realizan una parte concreta de la tarea global, transformándola en una unidad individual como parte de un diseño general de un programa más largo, lo que suele simplificar considerablemente la creación de un programa. En estas condiciones, las subrutinas pueden ser tratadas como programas completos, compilándolas o ensamblándolas individualmente, lo que acorta y simplifica el código fuente para un programa principal y facilita la utilización por cualquier otro programa de dichas subrutinas. De la misma forma, muchos compiladores incluyen ya ciertas subrutinas normalizadas, que pueden ser usadas en cualquier programa de aplicación, principalmente funciones matemáticas (raíces cuadradas, exponenciales, logaritmos, funciones trigonométricas, etc.).

Todo ello obliga a hacer las correspondientes conexiones entre las diferentes partes de un programa para que éste funcione globalmente. El compilador o el ensamblador generan sus instrucciones en lenguaje de máquina en forma de código objeto, pero en esta forma aún no está realmente preparado para funcionar como tal, ya que las citadas conexiones sólo están en forma de referencias.

Los programas *enlazadores o montadores* (*linkers*) tienen como misión reunir todas las partes del código objeto, hacer las conexiones entre ellas suministrando las direcciones reales de las subrutinas deseadas y poner el

conjunto de una forma que finalmente sea ejecutable en el ordenador. Incluso cuando un programa no tiene conexiones de subrutinas, los ensambladores y compiladores están diseñados de tal forma que traducen sus programas al formato de código objeto que necesita ser tratado con un enlazador, como forma normalizada de generar un programa ejecutable.

Ahora bien, es muy frecuente que el diseño de diversas subrutinas para un determinado programa pueda ser de utilidad en otro programa, lo que puede llevar a tener un soporte de subrutinas muy amplio, cuya manipulación individual puede resultar engorrosa, por lo que suelen almacenarse éstas en un fichero denominado *biblioteca de módulos objeto* (*object modules library*), que puede contener el código objeto de cualquier número de subrutinas de programa. Es decir, después de escribir una subrutina, se compila o ensambla por separado y se introduce en una biblioteca de módulos objeto. Esta labor se lleva a cabo con un programa denominado *cargador de biblioteca* (*library loader*), que permite reunir gran cantidad de módulos objeto de subrutinas en un paquete ordenado o biblioteca, que normalmente están grabadas en un soporte magnético, que permite almacenarlas. En general, un cargador de biblioteca puede realizar las siguientes funciones:

* Crear una biblioteca
* Cargar una subrutina en la biblioteca
* Buscar dentro de la biblioteca la subrutina deseada
* Transferir una subrutina de una biblioteca a otra
* Añadir o suprimir subrutinas de la biblioteca
* Generar un listado de las subrutinas almacenadas

Cuando las subrutinas que se desea utilizar en un programa principal se encuentran almacenadas en una determinada biblioteca, evidentemente habrá que enlazar el módulo objeto del programa principal con la biblioteca donde está contenida la subrutina deseada.

9.4.8 SISTEMAS OPERATIVOS

Un *sistema operativo* (*Operating System*) es un programa de control que supervisa todos los programas que se ejecutan en el ordenador: maneja las interrupciones, fija los dispositivos de E/S y gestiona la carga de los programas en memoria. Como controla el sistema completo, todos los demás

programas del sistema y los programas de aplicación han de interaccionar con él.

El sistema operativo lleva a cabo, fundamentalmente, tres tareas:

* *Manejar dispositivos*: realiza las operaciones necesarias para que todas las partes del ordenador funcionen correctamente.

* *Controlar programas*: carga los programas desde un soporte magnético, prepara el entorno para su correcta ejecución y gestiona la memoria que necesita el programa.

* *Procesar órdenes*: acepta órdenes procedentes del usuario y las ejecuta convenientemente.

Para llevar a cabo estas tareas, un sistema operativo suele estar estructurado en una jerarquía de tres niveles de servicios, en función de su integración con la máquina o con el usuario (Figura 9.7). El nivel central, que gestiona el sistema operativo, se denomina *núcleo* (*kernel*); a partir de éste se considera una interfase con la máquina denominada *sistema básico de E/S* (*Basic Input/Output System*) y una interfase con el usuario, denominada *procesador de órdenes* (*shell*). La importancia de estos tres niveles de la estructura del sistema operativo exige su estuido por separado, haciéndose referencia al sistema operativo más popular en los ordenadores personales, el DOS.

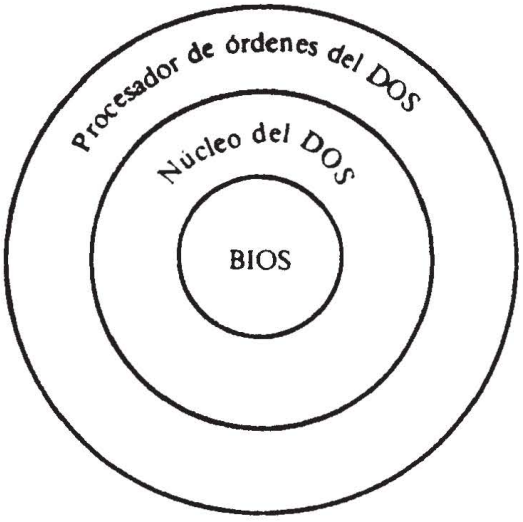


Figura 9.7

Niveles funcionales de un sistema operativo

9.4.8.1 EL NUCLEO DEL SISTEMA OPERATIVO

El núcleo del OS es un conjunto de procedimientos que se encarga de gestionar la ejecución de los programas, mediante un conjunto de servicios independientes del soporte físico, que reciben el nombre de "funciones del sistema". Los programas pueden acceder a las funciones del sistema cargando registros con los parámetros específicos de la función y transfiriendo a continuación el control al OS por medio de una llamada denominada *interrupción programada* (*software interrupt*). Estas funciones incluyen las tareas de:

* Gestión de ficheros y registros
* Gestión de memoria
* Gestión lógica de dispositivos de E/S de caracteres (acceso byte a byte)
* Acceso al reloj de tiempo real

En el DOS, el acceso de estas funciones se realiza a través de la interrupción lógica 21H, que permite a un programa inspeccionar directorios en disco, crear o borrar ficheros, leer o escribir registros dentro de ficheros, actualizar o leer el reloj en tiempo real y, en general, llevar a cabo un gran número de funciones (unas 100) de manera independiente de la máquina. Las funciones DOS disponibles a través de *Int 21H* (posición de la memoria que contiene el segmento y el desplazamiento del programa manipulador de esta interrupción) están normalizadas y pueden aplicarse a cualquier ordenador que utilice este sistema operativo. Los programas que realizan todas sus E/S por medio de estas funciones, se ejecutarán correctamente en cualquier máquina que soporte DOS, ya que el núcleo del sistema operativo es el que se encarga de relacionar los datos lógicos con los datos físicos. El núcleo del DOS está contenido en un fichero con nombre IBMDOS.COM ó MSDOS.SYS, según el fabricante del sistema operativo.

9.4.8.2 EL SISTEMA BASICO DE E/S

El sistema básico de E/S, BIOS, es un conjunto de subrutinas normalizadas que permite la transferencia de datos entre la CPU y cada dispositivo de E/S bajo el control de la subrutina adecuada para cada

dispositivo de E/S particular. El BIOS es específico de cada sistema físico particular y lo suministra el propio fabricante del sistema. Contiene las subrutinas de control de las unidades físicas que están residentes por defecto dentro de los dispositivos siguientes:

* Teclado y su representación (CONsole)
* Impresora (PRiNter)
* Dispositivo auxiliar (AUXiliary)
* Hora y fecha (CLOCK)
* Dispositivo para el disco de arranque

El núcleo del OS comunica con estos programas de control por medio de sus funciones de gestión de E/S; las subrutinas de control traducen las peticiones realizadas por el OS y las transforman en las órdenes necesarias para que funcionen los distintos controladores físicos de dispositivos: verifican que el dispositivo de E/S está conectado, comprueban su estado y ejecutan la transferencia. Cuando se ha completado la transferencia, el OS devuelve el control al programa de aplicación.

En algunos sistemas que utilizan el DOS, la mayor parte de las subrutinas manipuladoras de transferencias de E/S residen en la ROM, por lo que se conocen generalmente por ROM-BIOS; de esta forma, pueden ser utilizadas por aplicaciones individuales, diagnósticos, o el propio programa de arranque del sistema. Sin embargo, otra parte de estas subrutinas se encuentra en soporte magnético, en un fichero denominado IBMBIO.COM ó IO.SYS, que ha de ser cargado al arrancar el ordenador para construir un área de memoria que se encargue de las operaciones de E/S, tal como muestra la Figura 9.8.

En muchos casos, cuando se desea añadir un dispositivo físico adicional al ordenador, se necesita un programa denominado *controlador de dispositivo* (*device driver*) para poder traducir las instrucciones del sistema operativo a un formato comprensible para el dispositivo físico. Estos controladores de dispositivos denominados *instalables* (*installables*) para distinguirlos de los que utiliza el sistema por defecto, o *residentes* (*resident*), han de ser definidos en un fichero de configuración del DOS, llamado CONFIG.SYS, cuyo contenido es leído y almacenado en la memoria al arrancar el ordenador.

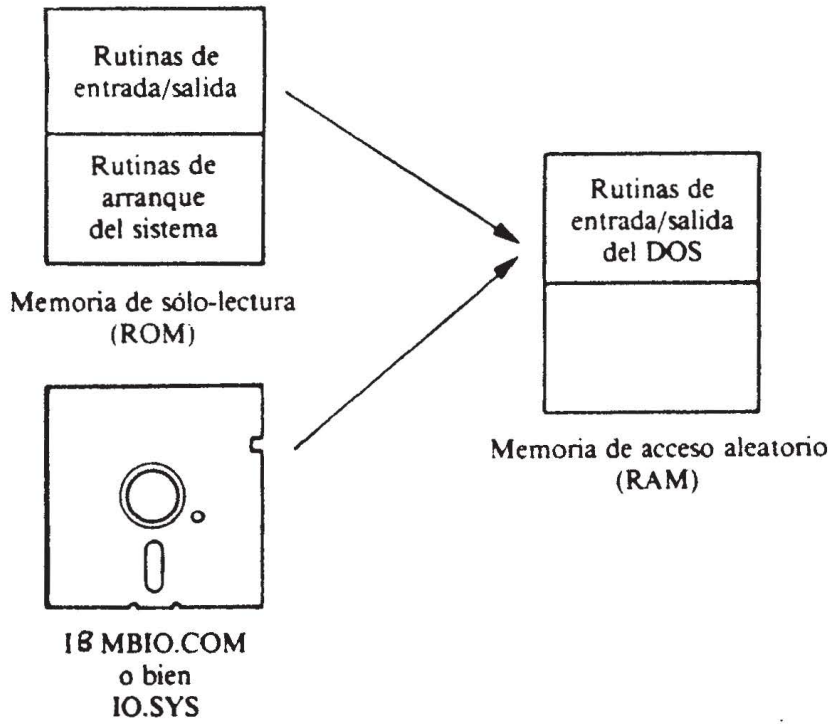


Figura 9.8

Construcción del área BIOS

9.4.8.3 EL PROCESADOR DE ORDENES

El procesador de órdenes es el programa del OS que actúa como interfase entre éste y el usuario; se responsabiliza del análisis gramatical y gestión de las órdenes del usuario, incluyendo la carga y ejecución de otros programas desde disco u otro dispositivo de almacenamiento.

En el caso del DOS, el procesador por defecto que se suministra con este sistema operativo se encuentra en un fichero llamado COMMAND.COM. Aunque los mensajes y respuestas que éste suministra constituyen la mayor parte de las informaciones emitidas y recibidas por el usuario desde y hacia el OS, es importante destacar que el fichero COMMAND.COM no es el sistema operativo en sí mismo, sino simplemente un programa especial que se ejecuta bajo el control de aquél. De hecho, COMMAND.COM puede ser sustituido por otro procesador de órdenes construido por el usuario, pasándose el control a éste con una orden contenida en el fichero de configuración del sistema, CONFIG.SYS.

El procesador por defecto del DOS, COMMAND.COM, está dividido en tres

partes:

* *Parte residente*: Se carga en la memoria y permanece en ella durante todo el tiempo de funcionamiento del sistema y desempeña todas las funciones esenciales a las que el DOS tiene que ser capaz de dar respuesta inmediata, tales como la carga de la parte transitoria del procesador de órdenes, la conclusión de la ejecución de un programa o la terminación de programas por lotes; además lleva a cabo la gestión de errores críticos del OS.

* *Módulo transitorio*: Se carga inicialmente en la memoria, per el espacio que ocupa puede ser posteriormente utilizado para otros fines por los programas de aplicación. Este módulo prepara el *símbolo para órdenes* (*prompt*) que ha seleccionado el usuario, lee órdenes desde el teclado o desde un *fichero por lotes* (*batch file*) y se encarga de que sean ejecutadas. Cuando finaliza un programa de aplicación, la parte residente de COMMAND.COM realiza una *suma de verificación* (*checksum*) del módulo transitorio para comprobar si ha sido destruido, y a continuación coloca en la posición correspondiente de la memoria una copia nueva si es necesario, leida de un medio de almacenamiento masivo, tal como muestra la **Figura 9.9.**

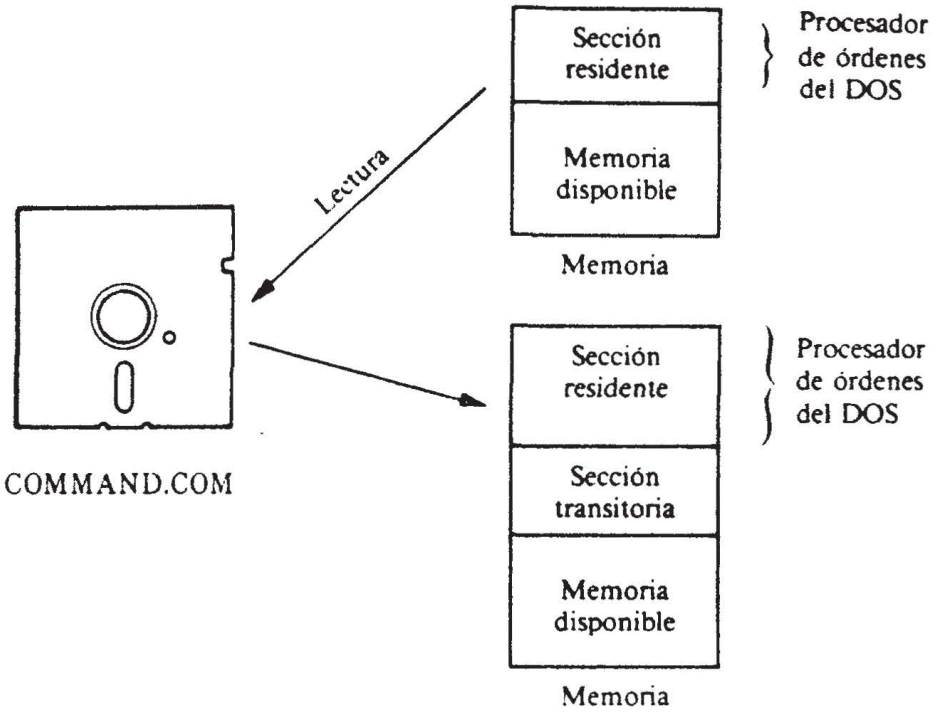


Figura 9.9

Carga del módulo transitorio del procesador de órdenes

* *Sección de arranque*: Se carga en la memoria únicamente cuando se arranca el sistema, con objeto de ejecutar el procedimiento denominado AUTOEXEC.BAT (lista de órdenes del usuario que se ejecuta en la inicialización del sistema), cuando existe. Al terminar este procedimiento, la sección de arranque del procesador de órdenes concluye y el espacio que ocupaba en la memoria se libera.

Las órdenes de usuario aceptadas por COMMAND.COM se pueden clasificar en tres categorías:

* Ordenes internas
* Ordenes externas
* Ficheros por lotes

Las *órdenes internas* (*internal commands*) u órdenes intrínsecas, son aquéllas contenidas en el propio código de COMMAND.COM. Cuando se invoca una de ellas el DOS se limita a localizar la orden en la memoria y a ejecutarla. Las subrutinas de ejecución de las órdenes internas están incluidas en el módulo transitorio de COMMAND.COM.

Las *órdenes externas* (*external commands*), órdenes extrínsecas o programas transitorios, son los nombres de los programas almacenados en los ficheros de un disco con extensiones .COM y .EXE. Antes de ejecutar estos programas, deben ser cargados desde el disco en la *zona de programas transitorios* (*Transient Program Area*) de la memoria y al finalizar, son eliminados de la misma. Por tanto, deben ser cargados de nuevo desde el disco cada vez que se vuelven a llamar.

Los *ficheros por lotes* (*batch files*) son ficheros de texto que contienen una lista de otras órdenes intrínsecas, extrínsecas (.COM ó .EXE) o por lotes (.BAT). Estos ficheros se procesan mediante un intérprete especial, incluido en el módulo transitorio de COMMAND.COM; el intérprete lee el fichero por lotes, a razón de una línea cada vez, y ejecuta las órdenes especificadas secuencialmente.

9.4.8.4 ARRANQUE DEL SISTEMA OPERATIVO

Como resumen del estudio de las características de un sistema operativo y, más concretamente, del DOS, se muestra a continuación de forma

esquemática el proceso de arranque de un ordenador personal que opera bajo DOS. La **Figura 9.10** permite observar la misión del programa de lanzamiento o autoarranque (bootstrap), ubicado en el primer sector del disco, en la carga del BIOS y del DOS. Por otra parte, la **Figura 9.11** muestra un diagrama de flujo de los pasos implicados desde el encendido del ordenador hasta que éste queda listo para que el usuario pueda suministrarle órdenes desde el teclado. Como se puede observar, este diagrama de flujo incluye la mayor parte de los conceptos estudiados acerca del sistema operativo.

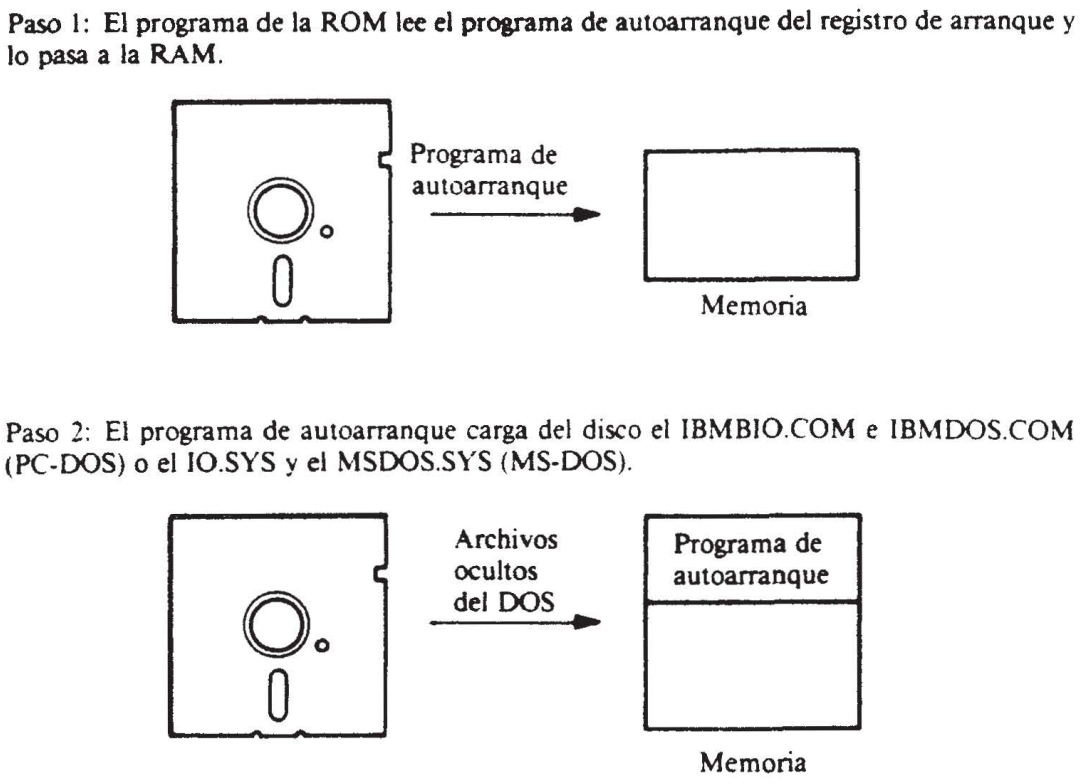


Figura 9.10

Ejecución de un programa de autoarranque

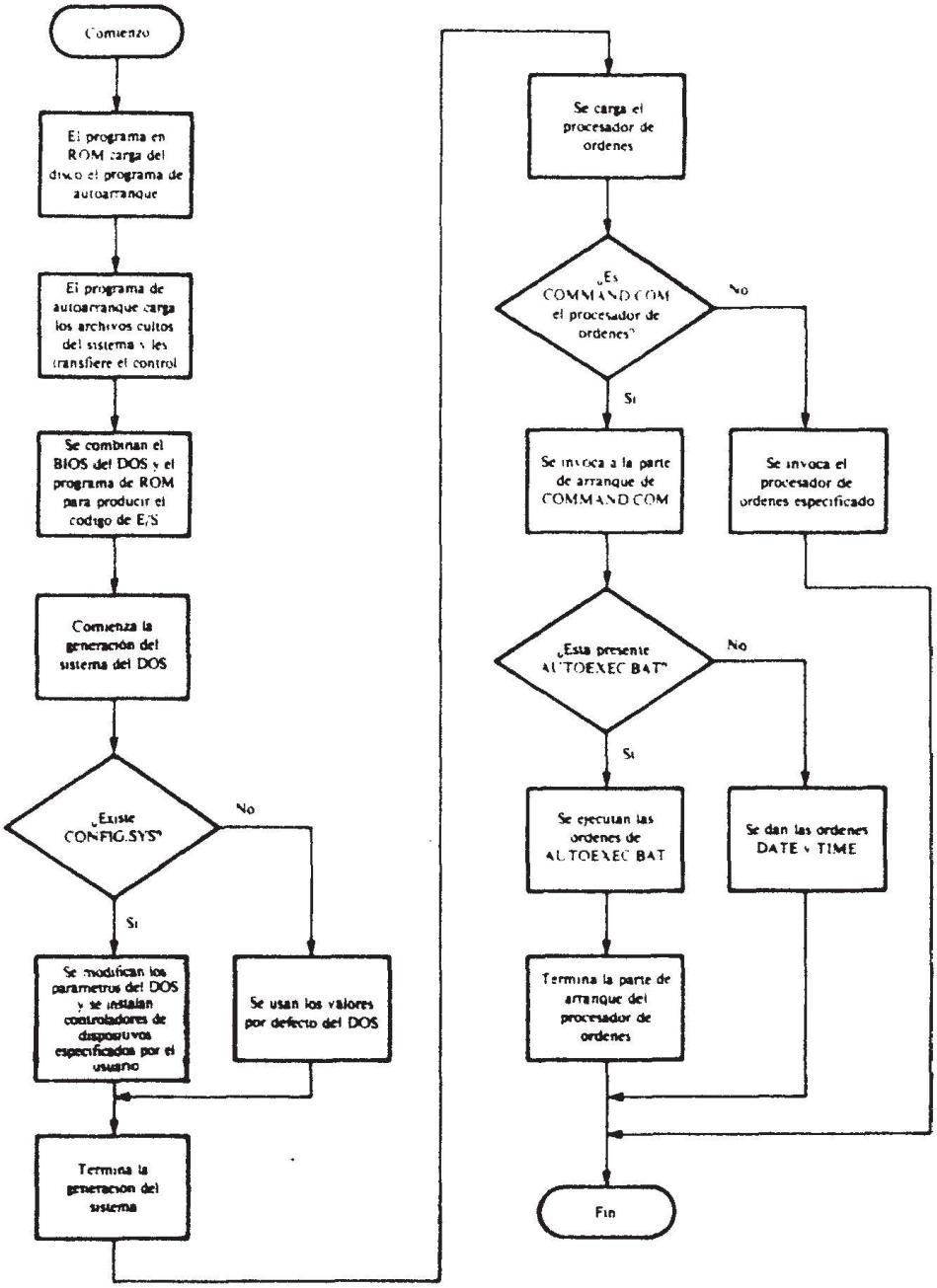


Figura 9.11

Arranque de un ordenador bajo DOS

9.5 PROGRAMAS DE APLICACION

Los *programas de aplicación* (*application programs*) son aquéllos destinados a efectuar tareas de usuario, por lo que a veces están escritos por él mismo. Se distinguen en este punto de los programas de sistemas, que componen el soporte lógico de propósito general, utilizados para dirigir y

administrar el funcionamiento global del ordenador.

Los programas de aplicación pueden escribirse en lenguaje ensamblador o en cualquiera de los lenguajes de alto nivel. No obstante, debido a la creciente compatibilidad de los ordenadores, cada vez es menos frecuente que el usuario escriba tales programas, limitándose a adquirirlos a empresas especializadas en su desarrollo. Todo ello ha llevado a que en el mercado exista un gran número de programas de aplicación, cuya utilización responde, en la mayoría de los casos, a la inclusión en uno de los grandes grupos siguientes:

* Utilidades
* Procesadores de textos
* Bases de datos
* Hojas de cálculo
* Paquetes integrados
* Paquetes matemáticos
* Juegos

Las *utilidades* (*utilities*) con programas de apoyo al sistema operativo que permiten al usuario conocer y modificar el estado de diversos parámetros de la máquina, así como emplear del modo más eficaz los recursos del sistema, tanto bajo el punto de vista físico como en el aspecto de programación.

Los *procesadores de textos* (*word processors*) son aplicaciones que permiten al usuario la gestión rápida de todo tipo de textos, ofreciendo complejas funciones de manipulación, entre ellas, la inserción y supresión de texto con justificación automática y realineación de márgenes, corrección de faltas de ortografía, paginado automático, etc. Estos programas se manejan mediante órdenes que el usuario introduce a través del teclado, que es supervisado por el programa para captar dichas órdenes.

Las *bases de datos* (*databases*) son ficheros estructurados o indexados cuyo contenido puede ser manipulado de forma muy eficaz por medio de un programa "gestor de base de datos", ya que la información se almacena sin redundancias, lo que permite ahorrar espacio de almacenamiento y acceder a ella de forma rápida y directa. El programa gestor está diseñado de tal forma, que el usuario puede crear, leer, actualizar, borrar o manipular registros de la base de datos con ayuda de órdenes de gran flexibilidad y

eficacia.

Las *hojas de cálculo* (*spreadsheets*) son programas que permiten efectuar cálculos con datos que afectan a muchas variables, con la ventaja de que, al introducir un cambio en cualquiera de los datos, automáticamente se actualizan los valores de todas las variables afectadas. La información es presentada al usuario en forma tabular y éste puede manipular filas y columnas, desplazándolas por la pantalla tanto vertical como horizontalmente.

Los *paquetes integrados* (*integrated packages*) son conjuntos de aplicaciones que permiten tener a disposición del usuario, como una sola aplicación, las herramientas más usuales de gestión comercial mediante ordenador: procesador de textos, base de datos y hoja de cálculo.

Los *paquetes matemáticos* (*math packages*) son programas especialmente desarrollados para usuarios que necesitan herramientas matemáticas de alto nivel, como cálculo matricial, cálculo diferencial e integral, cálculo estadístico, etc. Generalmente estos programas posee potentes interfases de representación gráfica, lo que permite obtener el máximo rendimiento de los mismos.

Los *juegos* (*games*) son programas basados fundamentalmente en la teoría de la simulación y su complejidad varía en márgenes muy amplios. Generalmente disponen de potentes interfases gráficas, de sonido y de control a través de dispositivos especiales de E/S como el *ratón* (*mouse*) o los *mandos de juegos* (*joysticks*).

BIBLIOGRAFIA

Boyce, J.C.; *"Microordenadores"*, Alhambra, Madrid (1986).

Chandor, A.: *"Diccionario de informática"*, Alianza Editorial, Madrid (1989).

Duncan, R.; *"MS-DOS avanzado"*, Anaya Multimedia, Madrid (1988).

Jamsa, K.; *"DOS: Manual de referencia"*, Osborne/McGraw-Hill, Madrid (1988).

Khambata, A.J.; *"Microprocesadores/microcomputadores. Arquitectura, software y sistemas"*, Gustavo Gili, Barcelona (1987).

Norton, P.; *"El IBM PC a fondo. Técnicas de programación avanzada"*, Anaya Multimedia, Madrid (1987).

Osborne, A.; *"Introducción a las microcomputadoras"*, Vol. 1, 2ª ed., Osborne/McGraw-Hill, Madrid (1986).

Osborne, A. y Bunnell, D.; *"Introducción a las microcomputadoras"*, Vol. 0, 3ª ed., Osborne/McGraw-Hill, Madrid (1986).