

Matemáticas para programadores

Sistemas de numeración
y Aritmética Binaria

William Barden, Jr.



Matemáticas para programadores

**Sistemas de numeración
y aritmética binaria**

William Barden, Jr.



ANAYA MULTIMEDIA

INFORMATICA PERSONAL-PROFESIONAL

Título de la obra original:
MICROCOMPUTER MATH

Traducción: Fernando García
Diseño de colección: Antonio Lax
Diseño de cubierta: Narcís Fernández

Reservados todos los derechos. Ni la totalidad ni parte de este libro puede reproducirse o transmitirse por ningún procedimiento electrónico o mecánico, incluyendo fotocopia, grabación magnética o cualquier almacenamiento de información y sistema de recuperación, sin permiso escrito de Ediciones Anaya Multimedia, S. A.

Copyright © 1982 by Howard W. Sams & Co., Inc.
Indianapolis, IN 46268

© EDICIONES ANAYA MULTIMEDIA, S. A., 1986
Villafranca, 22. 28028 Madrid
Depósito legal: M. 2579-1986
ISBN: 84-7614-070-3
Printed in Spain
Imprime: Anzos, S. A. Fuenlabrada (Madrid)

Indice

| | |
|---|-----------|
| Introducción | 7 |
| 1. El sistema binario: donde empieza todo | 11 |
| Big Ed aprende binario. Más sobre bits, bytes y binario. Paso de binario a decimal. Paso de decimal a binario. Rellenar a ceros hasta ocho o dieciséis bits. Ejercicios. | |
| 2. Octal, hexadecimal y otras bases numéricas | 25 |
| El chile está bien en Casiopea. Hexadecimal. Octal. Trabajando con otras bases numéricas. Convenios estándar. Ejercicios. | |
| 3. Números con signo y notación en complemento a dos | 37 |
| Big Ed y el bínaco. Sumar y restar números binarios. Representación en complemento a dos. Extensión del signo. Suma y resta en complemento a dos. Ejercicios. | |
| 4. Acarreos, errores de desbordamiento e indicadores | 49 |
| Este restaurante tiene una capacidad de +127 personas. ¡Evitando errores de desbordamiento! Errores de desbordamiento. Acarreo. Otros indicadores. Indicadores en los microordenadores. Ejercicios. | |
| 5. Operaciones lógicas y desplazamientos | 57 |
| El enigma británico. Operaciones lógicas. Operaciones de desplazamiento. Ejercicios. | |

| | |
|---|------------|
| 6. Multiplicación y división | 73 |
| Zelda aprende cómo desplazar por sí misma. Algoritmos de multiplicación. Algoritmos de división. Ejercicios. | |
| 7. Múltiple precisión..... | 87 |
| ¿Tienen algo que ver las series de Fibonacci con la televisión? Suma y resta empleando múltiple precisión. Multiplicación en múltiple precisión. Ejercicios. | |
| 8. Fracciones y factores de escala | 97 |
| Big Ed pesa los números. Fracciones en sistema binario. Operando con fracciones en sistema binario. Ejercicios. | |
| 9. Transformaciones ASCII | 109 |
| Big Ed y el inventor. Códigos ASCII. Paso de ASCII a enteros binarios. Paso de ASCII a fracciones binarias. Paso de enteros binarios a ASCII. Paso de fracciones binarias a ASCII. Ejercicios. | |
| 10. Números en punto flotante | 121 |
| ... y tres mil platos combinados para la nave nodriza... Notación científica en punto flotante. Uso de potencias de dos en lugar de potencias de diez. Números en punto flotante de doble precisión. Cálculos en los que se emplean números binarios en punto flotante. Ejercicios. | |
| Apéndices: | |
| A) Respuestas a los ejercicios..... | 135 |
| B) Conversiones binario, octal, decimal y hexadecimal | 139 |
| C) Tabla de conversión de números en complemento a dos | 147 |
| Glosario..... | 149 |
| Indice alfabético | 157 |

Introducción

No pasa mucho tiempo, después de adquirir un microordenador, sin que el usuario tropiece fatalmente con referencias tales como "números binarios", "valor hexadecimal", "efectuar la operación lógica 'Y' con dos números para obtener el resultado" o "desplazar el resultado multiplicando por dos". Algunas veces, estas referencias suponen que el lector conoce el sistema binario y la forma de operar con él; otras, uno tiene la impresión de que el escritor del manual de referencia realmente tampoco sabe demasiado sobre las operaciones a realizar.

El objetivo de *Matemáticas para programadores* es poner fin a algunos de los misterios que rodean las operaciones matemáticas especiales que se emplean en BASIC y en lenguaje ensamblador. Tales operaciones, como sistema binario, octal o hexadecimal, operaciones complemento a dos, suma y resta de números binarios, indicadores en microordenadores, operaciones lógicas y desplazamientos, algoritmos de multiplicación y división, operaciones en múltiple precisión, fracciones, factores de escala y operaciones en punto flotante, se explican detalladamente a lo largo del libro, junto con ejemplos prácticos y ejercicios de autoevaluación.

Si uno puede sumar, restar, multiplicar y dividir con números decimales, entonces podrá ejecutar las mismas operaciones en binario o en cualquier otra base numérica, tal como la hexadecimal. Este libro le enseñará cómo.

También será un excelente compañero en cualquier curso de lenguaje ensamblador o BASIC Avanzado.

Matemáticas para programadores consta de diez capítulos. La mayoría de ellos se basan en el material contenido en los que le preceden. Cada capítulo finaliza con ejercicios de autoevaluación. Es provechoso realizar los ejercicios porque ayudan a fijar la materia en su mente, pero no nos enfadaremos con usted si utiliza el libro sólo como referencia.

Leyendo, se observarán algunas palabras en *cursiva*. La mayoría de ellas son términos informáticos que se definen en el glosario. Utilizándolo, también los neófitos pueden entender y sacar provecho de este libro.

El libro está estructurado como sigue:

El capítulo 1 trata el sistema binario desde la base e incluye las conversiones entre números binarios y decimales, mientras el capítulo 2 describe los números octales y hexadecimales, y las transformaciones entre estas bases y los números decimales. Los números hexadecimales se utilizan en BASIC y en lenguaje ensamblador.

Los números con signo y en complemento a dos se incluyen en el capítulo 3. Los complementos a dos es una notación usada en números negativos.

El capítulo 4 trata de los acarreos, errores de desbordamiento e indicadores. Estos términos se usan principalmente en lenguaje máquina y en programas en lenguaje ensamblador, pero pueden ser también importantes en programas especiales de BASIC.

Las operaciones lógicas, como las "Y" (AND), "O" (OR) y "NO" (NOT) del BASIC, se describen en el capítulo 5 junto con los tipos de desplazamientos posibles en lenguaje máquina. Después, el capítulo 6 habla de los algoritmos de multiplicación y división, incluyendo operaciones con y sin signo.

El capítulo 7 describe operaciones en múltiple precisión. Esta puede utilizarse en BASIC y en lenguaje ensamblador para implementar la "precisión ilimitada" con cualquier número de dígitos.

El capítulo 8 incluye fracciones binarias y factores de escala. Esta materia es necesaria para entender el formato interno de los números en punto flotante en BASIC.

Seguidamente, los códigos y las conversiones ASCII, en cuanto se refieren a cantidades numéricas, se describen en el capítulo 9.

Finalmente, el capítulo 10 proporciona una explicación de la representación de los números en punto flotante en la forma en que éstos se utilizan en muchos intérpretes en el BASIC de Microsoft.

Después, en la última sección del libro, el apéndice A contiene las respuestas a las cuestiones de autoevaluación; el apéndice B contiene una lista de números binarios, octales, decimales y hexadecimales del 0 al 1023.

La lista puede utilizarse para pasar de un tipo de sistema a otro. Por último, el apéndice C contiene una lista de los números en complemento a dos del -1 al -128, una referencia que no se encuentra habitualmente en otros textos; a continuación, se incluye un glosario de términos.

WILLIAM BARDEN, JR.

El sistema binario: donde empieza todo

En el sistema binario, todos los números se representan por una condición “encendido/apagado”. Veamos un ejemplo rápido de binario en términos fácilmente comprensibles.

Big Ed aprende binario

“Big Ed” Hackenbyte es propietario de “Big Ed’s”, un restaurante que sirve comidas rápidas y cenas lentas en el área cercana a San José (California). En esta zona, conocida como Valle del Silicio, hay docenas de compañías que fabrican microprocesadores.

Ed tiene ocho personas a su servicio: Zelda, Olive, Trudy, Thelma, Fern, Fran, Selma y Sidney. Debido a la despersonalización existente, tiene asignados números para la nómina. Los números asignados son:

| | <u>Número</u> | | <u>Número</u> |
|------------|---------------|-------------|---------------|
| Zelda..... | 0 | Fern..... | 4 |
| Olive..... | 1 | Fran..... | 5 |
| Trudy..... | 2 | Selma..... | 6 |
| Thelma.... | 3 | Sidney..... | 7 |

Cuando Big Ed rellenó por primera vez el panel de llamadas, tenía ocho luces, una para cada persona del servicio, como puede verse en la figura 1.1. Un día, sin embargo, Bob Borrow, ingeniero de diseño de una compañía de microprocesadores conocida como Inlog, llamó a Ed.

"Ed, podrías ser mucho más eficiente con tu panel de llamadas, ¿sabes? Puedo mostrarte cómo hemos diseñado el tablón con uno de nuestros microprocesadores."

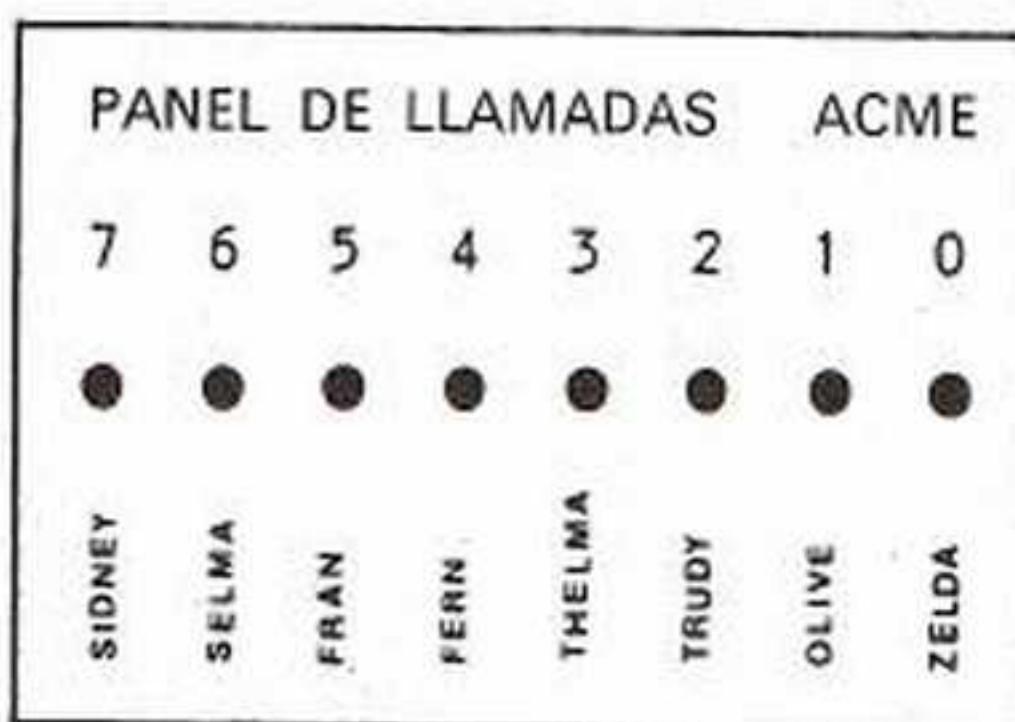


Figura 1.1. Panel de llamadas de Big Ed

Ed, interesado en la nueva tecnología, siguió su consejo. El nuevo diseño del tablón de anuncios se muestra en la figura 1.2. Tiene tres luces, controladas desde la cocina. Cuando se llama a alguien del servicio, suena un timbre; ¿cómo es posible llamar a alguna de las ocho personas del servicio por medio de combinaciones luminosas de las tres luces?

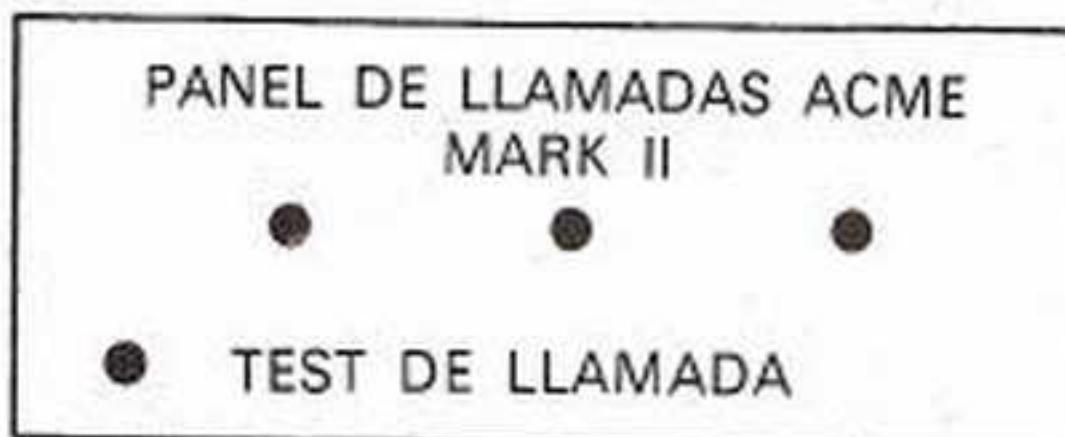


Figura 1.2. Panel de llamadas en binario

"¿Ves, Ed? Este panel es muy eficaz. Emplea cinco luces menos que tu primer panel. Hay ocho combinaciones diferentes de luces. En realidad les llamamos *permutaciones*, pues hay un orden definido en la disposición de las luces. He preparado una tabla de las permutaciones de las luces y la persona del servicio llamada." Dio a Ed la tabla mostrada en la figura 1.3.

Sólo hay ocho permutaciones diferentes de luces, Ed, ni más ni menos. Estas luces están ordenadas en forma *binaria*. Utilizamos el sistema binario en nuestros ordenadores por dos razones: primero, se ahorra espacio. Redu-

cimos el número de luces de ocho a tres. Segundo, los ordenadores baratos sólo pueden representar normalmente un estado encendido/apagado, igual que las luces están encendidas o apagadas." Hizo una pausa para dar un bocado a su "Big Edburger".

| | | |
|-------|--------|---|
| 0 0 0 | ZELDA | 0 |
| 0 0 • | OLIVE | 1 |
| 0 • 0 | TRUDY | 2 |
| 0 • • | THELMA | 3 |
| • 0 0 | FERN | 4 |
| • 0 • | FRAN | 5 |
| • • 0 | SELMA | 6 |
| • • • | SIDNEY | 7 |

Figura 1.3. Código para el panel

"Daré estos códigos a mis ayudantes para que los memoricen", dijo Ed.

"Cada persona del servicio sólo tiene que memorizar su código, Ed. Te daré la clave, de forma que puedas descifrar qué persona del servicio es llamada, sin necesidad de la tabla.

¿Ves? Cada luz representa una potencia de dos. La luz de la derecha representa dos elevado a cero. La siguiente, dos elevado a uno, y la de más a la izquierda es dos elevado a dos. En realidad es muy parecido al sistema decimal, donde cada dígito representa una potencia de diez." Garabateó un ejemplo en el mantel, como muestra la figura 1.4.

$$\begin{array}{r}
 3 \quad 7 \quad 5 \\
 | \quad | \quad | \\
 5 \times 10^0 = 5 \\
 7 \times 10^1 = 70 \\
 3 \times 10^2 = 300 \\
 \hline
 375
 \end{array}$$

$$\begin{array}{r}
 1 \quad 0 \quad 1 \\
 | \quad | \quad | \\
 1 \times 2^0 = 1 \\
 0 \times 2^1 = 0 \\
 1 \times 2^2 = 4 \\
 \hline
 5
 \end{array}$$

Figura 1.4. Comparación de los sistemas binario y decimal

"De la misma forma que podemos emplear las potencias de diez para números altos, podemos utilizar tantas potencias de dos como queramos. Podríamos usar treinta y dos luces, si quisieramos. Entonces, para pasar las

tres luces en binario a su equivalente en decimal, habría que sumar la potencia de dos correspondiente a cada luz encendida." Garabateó otra figura en el mantel (Fig. 1.5).



Figura 1.5. Paso de binario a decimal

"Bueno, parece bastante sencillo", admitió Ed. "De derecha a izquierda, las luces representan 1, 2 y 4. Si tuviéramos más camareros y camareras, las luces representarían 1, 2, 4, 8, 16, 32, 64, 128..." Su voz dejó de oírse, al no ser capaz de decir la siguiente potencia de dos.

"Exacto, Ed. Frecuentemente, tenemos el equivalente a ocho o dieciséis luces en nuestros microprocesadores. No usamos luces, por supuesto; utilizamos semiconductores que están apagados o encendidos." Dibujó otra figura en el mantel, que por aquel entonces estaba lleno de diagramas (Fig. 1.6).

"Llamamos *bit* a cada una de las ocho o dieciséis posiciones. 'Bit' es una contracción de dígito binario. Después de todo, eso es de lo que hemos estado hablando; los dígitos binarios forman números binarios, igual que los dígitos decimales forman números decimales. Tu panel representa un número de tres bits.

En 8 bits podemos representar cualquier número entre 0 y $1 + 2 + 4 + 8 + 16 + 32 + 64 + 128$. Sumando todos ellos, se obtiene 255, el mayor número que puede ser contenido en 8 bits."

"¿Qué ocurre con los 16 bits?", preguntó Ed.

| | | | | | | | | |
|-----------------------|----------|----------|----------|----------|----------|-------|-------|-------|
| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| POSICIONES DE 8 BITS | | | | | | | | |
| 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 |
| 2^{15} | 2^{14} | 2^{13} | 2^{12} | 2^{11} | 2^{10} | 2^9 | 2^8 | 2^7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| POSICIONES DE 16 BITS | | | | | | | | |

Figura 1.6. Representación de 8 y 16 bits

“Te lo puedes imaginar”, dijo Bob. “Tengo que volver al trabajo de diseñar microprocesadores.”

Ed confeccionó una lista con todas las potencias de dos hasta quince. Entonces, las sumó todas hasta llegar al resultado que muestra la figura 1.7, un total de 65,535 —el mayor número que pueden contener 16 bits.

| PESO | POSICIÓN DEL BIT |
|-------|------------------|
| 1 | 0 |
| 2 | 1 |
| 4 | 2 |
| 8 | 3 |
| 16 | 4 |
| 32 | 5 |
| 64 | 6 |
| 128 | 7 |
| 256 | 8 |
| 512 | 9 |
| 1024 | 10 |
| 2048 | 11 |
| 4096 | 12 |
| 8192 | 13 |
| 16384 | 14 |
| 32768 | 15 |
| 65535 | |

Figura 1.7. Valor máximo de 16 bits

“El asunto de los microprocesadores es fácil”, dijo Ed con una mueca, mientras daba a su Big Ed’s Jumboburger un “mordisco”¹ de 8 bits.

Más sobre bits, bytes y binario

La explicación de Bob Borrow del binario condensa muy bien la representación binaria de los microordenadores. La unidad básica es un bit o dígito binario. Un bit puede estar encendido o apagado. Porque es mucho más fácil escribir 0 ó 1, estos dígitos se usan en lugar de encendido/apagado cuando representamos valores binarios.

La posición del bit del número binario se refiere a la posición del dígito en el número. La posición del bit en la mayoría de los microordenadores tiene un número asociado, como se muestra en la figura 1.8. Puesto que éstas, en realidad, representan una potencia de dos, es conveniente numerarlas de acuerdo con la potencia de dos representada. El bit de más a la derecha es dos elevado a cero, y la posición del bit es, por tanto, cero. Las posiciones del bit hacia la izquierda se numeran 1 (dos elevado a uno), 2 (dos elevado a dos), 3, 4, 5, etc.

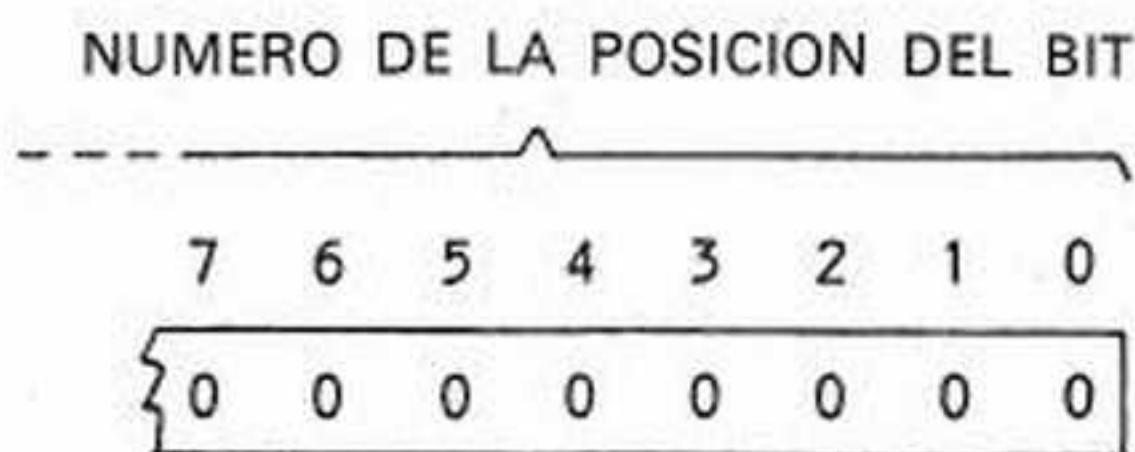


Figura 1.8. Numeración de las posiciones del bit

En todos los microordenadores actuales, un grupo de ocho bits se denomina un *byte*. De alguna manera, es obvio el origen de esta palabra si nos imaginamos a los primeros ingenieros informáticos comiendo en el equivalente de “Big Ed’s”².

La memoria de un microordenador viene a menudo indicada por el número de bytes de que consta.

Cada byte corresponde aproximadamente a un carácter, como veremos en capítulos posteriores. Las operaciones de entrada y salida de la memoria se hacen generalmente de un byte cada vez.

¹ N. del T.: Se trata de un juego de palabras entre “bite” (morder), “byte” y “bit”, que, además de dígito binario, en inglés significa mordisco.

² N. del T.: Continúa con el mismo juego de palabras relativo a byte.

Los *registros* en el interior de un microprocesador tienen también la extensión de uno o dos bytes. Estos, en realidad, no son más que posiciones de memoria de acceso rápido que se utilizan para almacenamiento temporal en el microprocesador. Normalmente hay diez bytes de registros en los microprocesadores y hasta 65,535 bytes en la memoria de un microordenador, como ilustran las figuras 1.9 y 1.10.

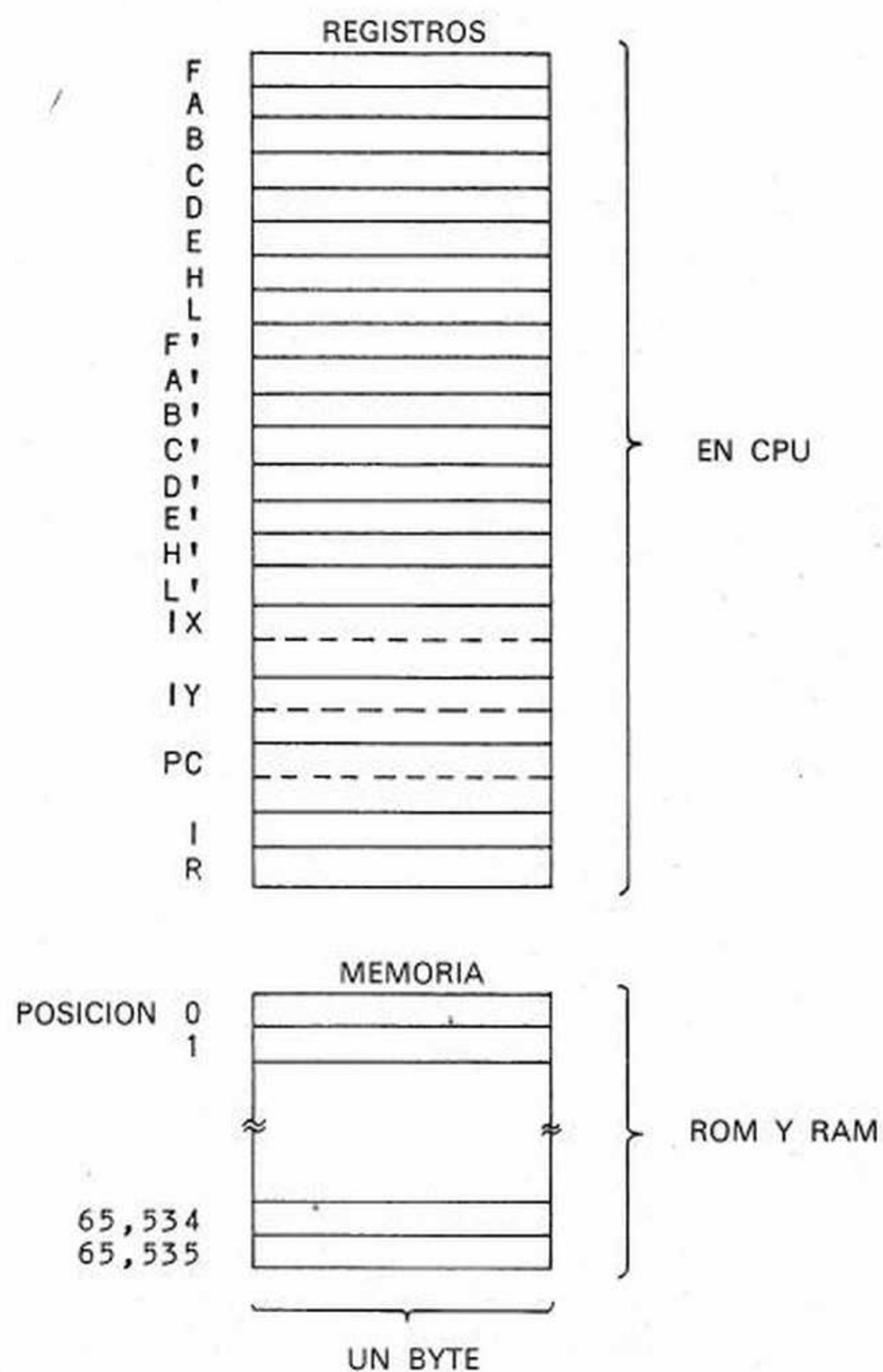


Figura 1.9. Registros y memoria del Z-80

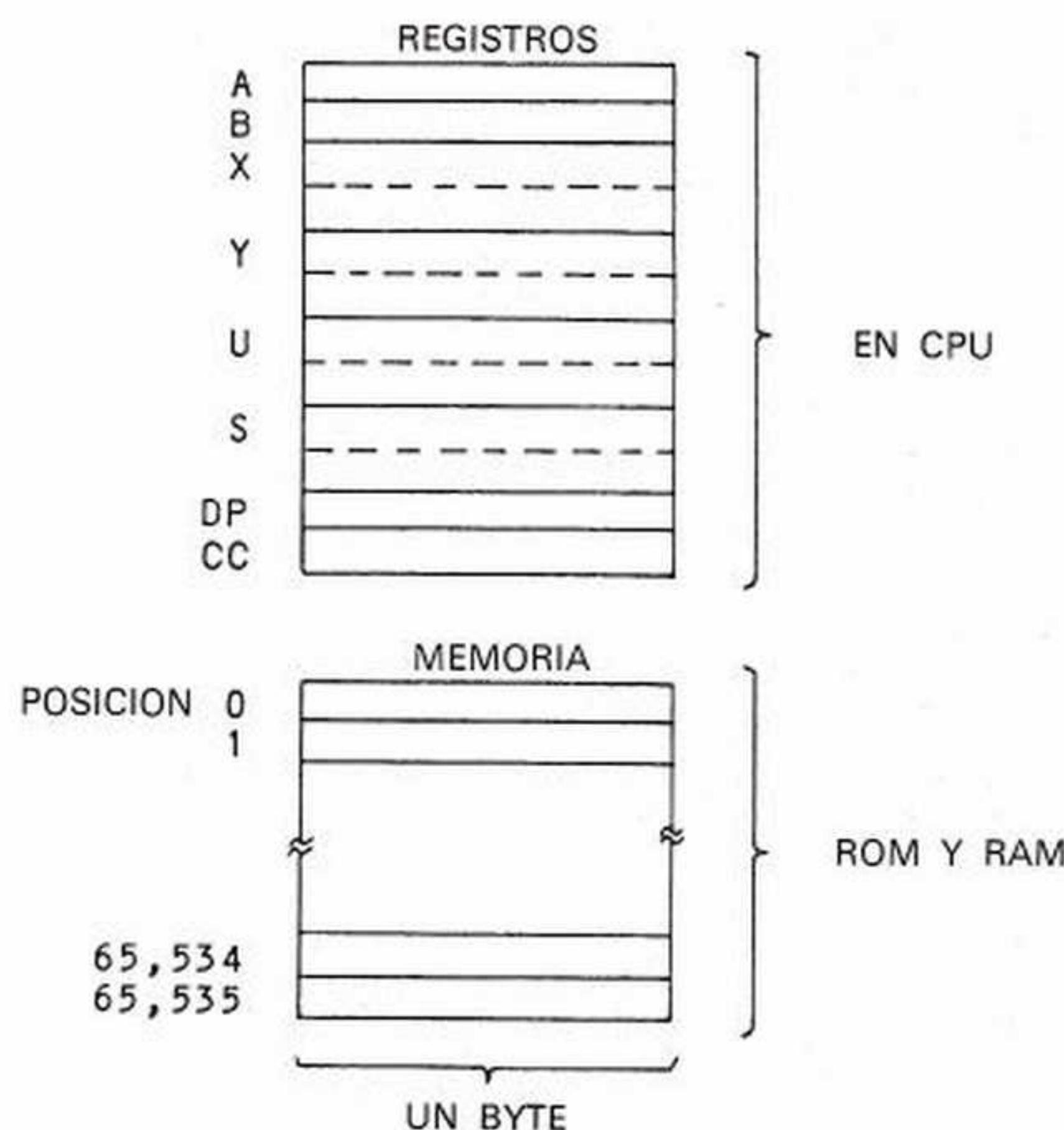


Figura 1.10. Registros y memoria del 6809E

¿Hay otras agrupaciones por encima o por debajo de los bytes? Algunos microordenadores hablan de *palabras*, que pueden ser dos o más bytes, pero, generalmente, byte es el término más utilizado para nombrar un grupo de bits. Algunas veces, las agrupaciones de cuatro *bits* se denominan *nibble*³ (los primeros ingenieros informáticos debían estar obsesionados con la comida).

El BASIC, generalmente, opera con ocho o dieciséis bits de datos a la vez —uno o dos bytes—. Un byte se utiliza para las instrucciones PEEK o POKE del BASIC, que permiten al usuario leer o escribir datos en una posición de memoria. Esto es útil cuando se usan para cambiar algunos de los parámetros del sistema que, de otra manera, serían inaccesibles desde el BASIC.

Se utilizan dos bytes para representar *variables enteras*. En este tipo de formato, una variable del BASIC puede contener valores desde -32,768 hasta +32,767. Este número se almacena como un valor entero *con signo*, como veremos en el capítulo 3.

³ N. del T.: Nibble, en inglés, significa también "bocadito".

Si está interesado en el lenguaje ensamblador de su microordenador, tendrá que operar con bytes para realizar operaciones aritméticas, como suma y resta, y de uno a cuatro bytes para representar el *lenguaje máquina* correspondiente a la instrucción del microprocesador.

Ya que nos referiremos continuamente a valores de uno y dos bytes, vamos a investigar sobre ellos más a fondo.

Paso de binario a decimal

Big Ed encontró los valores máximos que podían ser almacenados en uno o dos bytes, sumando todas las potencias de dos. Eran 255 para un byte y 65,535 para dos bytes. Cualquier número entre ellos puede ser representado empleando los bits apropiados.

Supongamos que tenemos el valor binario de 16 bits 0000101001011101. Para encontrar el número decimal de este valor binario, podríamos utilizar el método de Big Ed de sumar las potencias de dos. Esto se ha hecho en la figura 1.11, donde obtenemos como resultado 2653. ¿Hay alguna forma más sencilla de pasar de binario a decimal? Sí, hay varias.

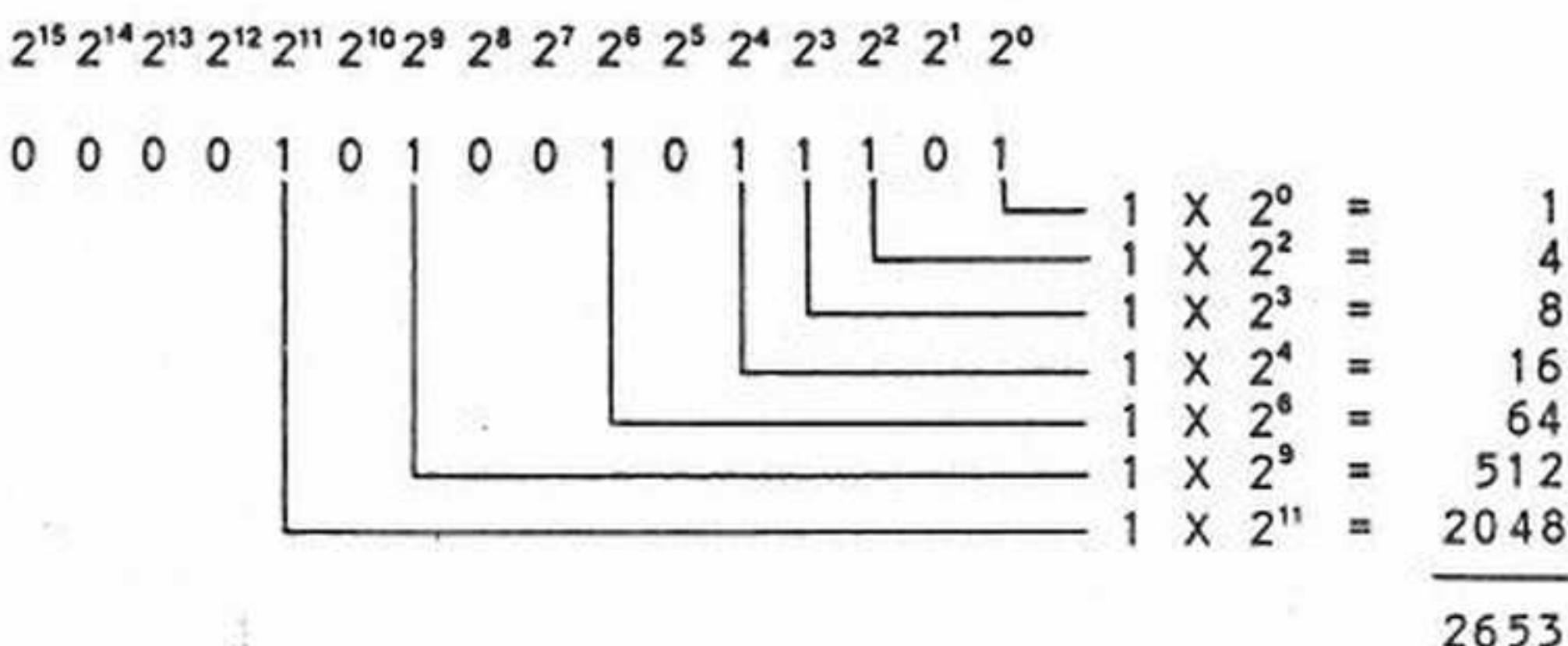


Figura 1.11. Paso de binario a decimal

El primer procedimiento consiste en usar una tabla de valores. Hemos incluido una tabla de este tipo en el apéndice C, que muestra la relación entre los sistemas binario, octal, decimal y hexadecimal para valores hasta 1023. Como para mostrar valores hasta 65,535 con 16 bits se necesita bastante espacio, tiene que existir un procedimiento más adecuado.

Un método que funciona sorprendentemente bien es el llamado “*doblar y sumar*”. Después de un poco de práctica, resulta muy fácil pasar unos diez bits de binario a decimal. En el siguiente capítulo se muestra un método que

sirve para diecisésis o más bits. Doblar y sumar es un procedimiento estrictamente mecánico, que automatiza el proceso de conversión. Es más difícil describirlo que hacerlo. Para utilizarlo, haga lo siguiente:

1. Tome el primer 1 (más a la izquierda) del número binario.
 2. Multiplique el 1 por dos y sume el siguiente bit de la derecha, sea 0 ó 1.
 3. Multiplique el resultado por dos y sume este resultado al siguiente bit.
 4. Repita este proceso hasta que el último dígito haya sido sumado al total.
- El resultado es el número decimal equivalente al número binario.

Este procedimiento se muestra en la figura 1.12 para la cantidad 2653 utilizada en el ejemplo anterior. Después de haber estado operando con números binarios durante algún tiempo, es probable que reconozca 1111 como el decimal 15 y 11111 como 31. Puede emplear también pequeños trucos, como darse cuenta que 11110 es uno menos que 31; o que 101000

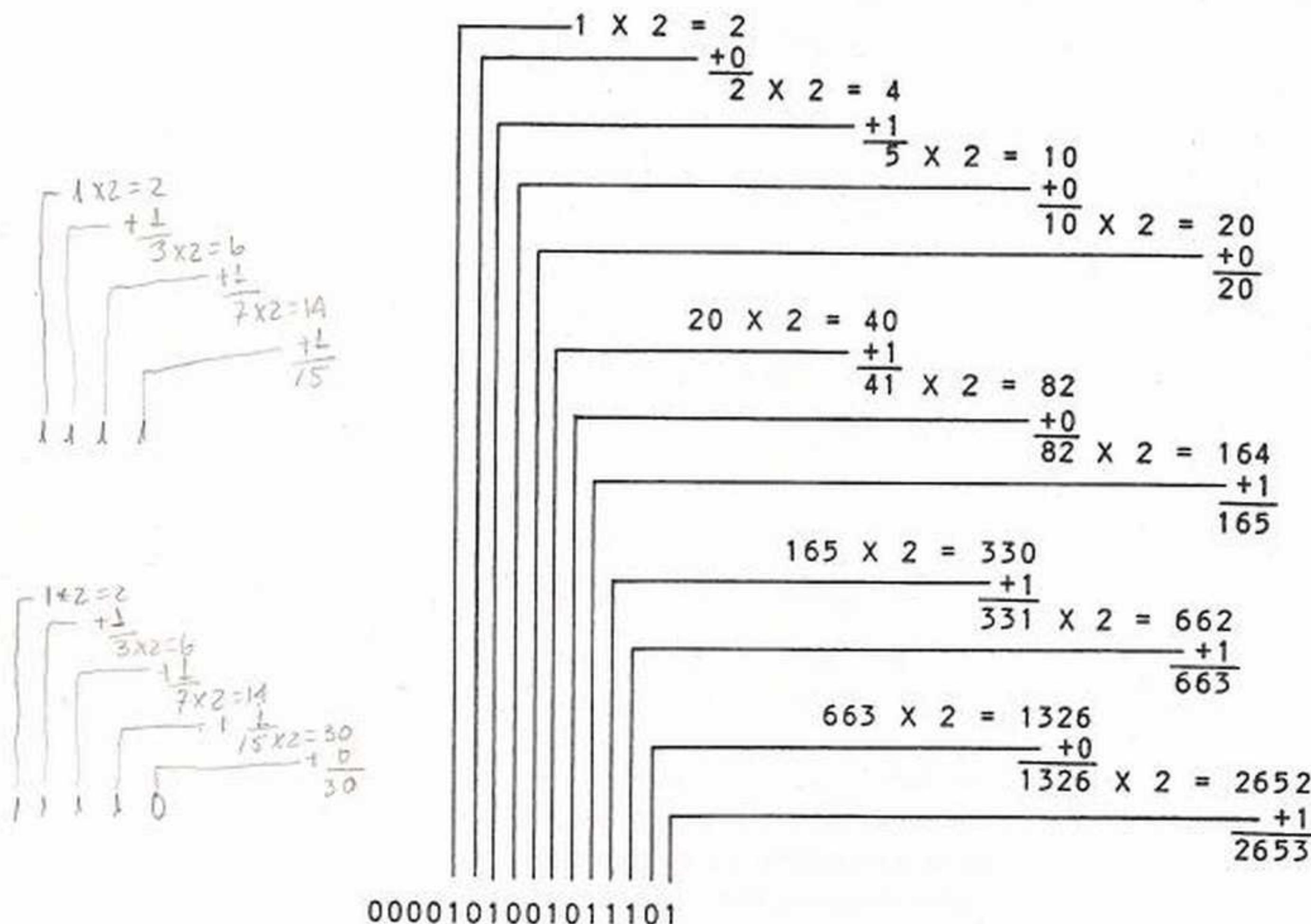


Figura 1.12. Paso de binario a decimal empleando el método doblar y sumar.

es también $1010 = 10$ (en decimal) multiplicado por cuatro para obtener un valor de 40. Por el momento, sin embargo, haga unos pocos ejemplos para acostumbrarse al procedimiento; con el tiempo lo hará automáticamente, y probablemente no vale la pena malgastar el tiempo en convertirse en experto en el sistema binario.

Paso de decimal a binario

¿Cómo se hace a la inversa? Es un proceso diferente. Supongamos que tenemos que pasar el número decimal 250 a binario. Analizaremos varios métodos.

El primer método es el de “inspección de potencias de dos”. Podría denominarse también resta sucesiva de potencias de dos, pero tal vez a Big Ed no le gustaría el nombre. En este método, todo lo que hacemos es tratar de restar una potencia de dos y poner un 1 en la posición del bit correcta, si se puede (véase Fig. 1.13). Algunas potencias de dos son: 256, 128, 64, 32, 16, 8, 4, 2, 1, comenzando por las mayores. Es obvio que 256 no se podrá restar, luego pondremos un 0 en esa posición. La potencia 128 vale, quedando 122. Ponemos un 1 en la posición 7. La potencia 64 se resta a 122, quedando 58; luego ponemos un 1 en la posición 6. Este proceso se repite hasta calcular la última posición, como muestra la figura 1.13.

El método anterior es muy aburrido. ¿Hay uno mejor? Uno más eficaz es el llamado “dividir por dos y guardar los restos”. En este método se hace lo que indica el nombre; esto se explica en la figura 1.14. La primera división es 250 entre dos, resultando 125. Resto 0. La siguiente división es 125 entre dos, dando 62 y quedando 1. El proceso se repite hasta que el “residuo” es 0. Ahora, los restos se colocan en orden inverso. El resultado es el número binario equivalente al decimal.

Rellenar a ceros hasta ocho o dieciséis bits

Este es un punto no carente de importancia. Las posiciones de la izquierda se llenan con ceros. Esto es una refinada sutileza que se emplea para “completar con ceros” el número binario hasta ocho o dieciséis bits, según el tamaño con el que se esté operando. Tiene, sin embargo, implicación en los *números con signo*, luego es mejor empezar a manejarlos en la práctica cuanto antes.

En el capítulo siguiente trataremos la notación de los números *octales* y *hexadecimales*. Mientras tanto, intente hacer algunos ejercicios de autoevaluación para practicar el paso entre números decimales y binarios.

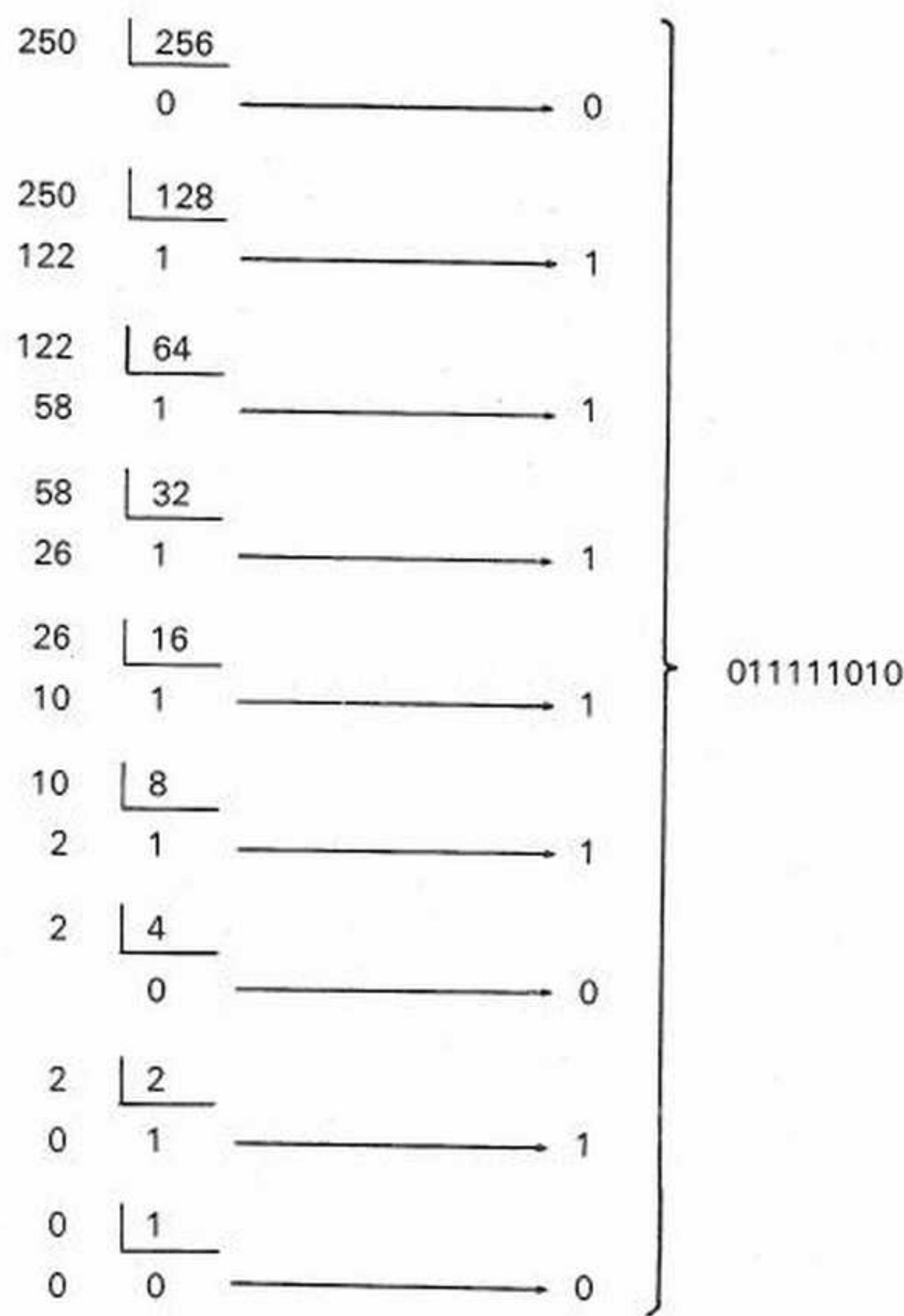


Figura 1.13. Paso de decimal a binario por inspección

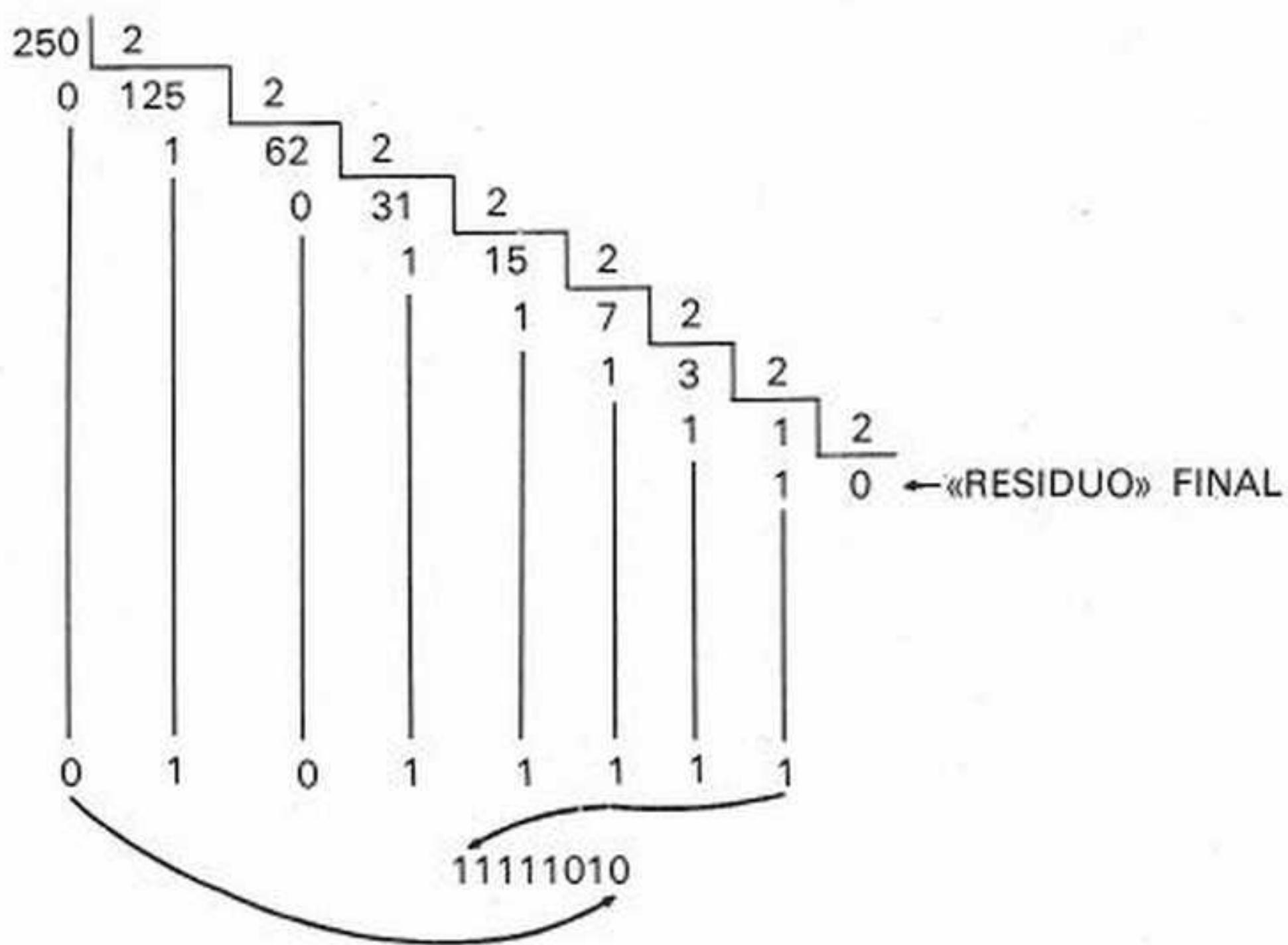


Figura 1.14. Paso de decimal a binario por el método "dividir y guardar los restos"

Ejercicios

1. Hacer una lista de los equivalentes binarios de los números decimales 20 a 32.
2. Pasar los siguientes números binarios a sus equivalentes decimales: 00110101; 00010000; 01010101; 11110000; 0011011101101001.
3. Pasar los siguientes números decimales a su forma binaria: 15, 26, 52, 105, 255, 60000.
4. "Rellenar a ceros" los siguientes números binarios hasta ocho bits: 101; 110101; 010101.
5. ¿Cuál es el mayor número decimal que puede ser almacenado en cuatro bits? ¿Y en seis bits? ¿Y en ocho? ¿Y en dieciséis? Si n es el número de bits, ¿qué regla general se puede establecer sobre el mayor número que se puede almacenar en n bits? (La respuesta "Unos números enormes", no se considera aceptable.)

Octal, hexadecimal y otras bases numéricas

Los sistemas hexadecimal y octal son variantes de los números binarios. Se utilizan normalmente en sistemas de microordenadores, especialmente el hexadecimal. Los datos pueden especificarse en notación hexadecimal, tanto en BASIC como en lenguaje ensamblador, en muchos microordenadores. Trataremos los sistemas octal y hexadecimal en este capítulo junto a otros sistemas numéricos interesantes. Hagamos otra visita a Big Ed.

El chile está bien en Casiópea

“No se ve mucha gente como tú por aquí”, dijo Big Ed, mientras ponía un cuenco del Chile Sorpresa de Big Ed frente a un cliente de piel verde con escamas.

“Sé que debería decir algo como ‘Sí, y con estos precios usted verá muchos menos’, pero le diré la verdad. Acabo de llegar de las Naciones Unidas para echar un vistazo a su industria de semiconductores”, dijo el visitante. “Es impresionante.”

“Yo mismo he trabajado en ella”, dijo Big Ed, mirando a su panel de llamadas. “¿De dónde eres, Buddy? No he podido evitar fijarme en tus manos de ocho dedos.”

"Probablemente nunca has oido hablar de ello; es una estrella pequeña... ehm... un lugar. Estas manos, por cierto, son las que me trajeron al Valle del Silicio. Vuestros últimos microprocesadores son para mi gente algo natural. ¿Te gustaría oír algo sobre ello?"

Ed asintió.

"Verás, nosotros somos los Hackers¹, y basamos todo en las potencias de dieciséis." Miró a Big Ed para ver si había cogido la idea. "Cuando nuestra civilización se desarrollaba al principio, contábamos con nuestras manos. Encontrábamos muy fácil contar vuestro valor dieciséis utilizando simplemente nuestros dedos. Posteriormente, necesitamos expresar números mayores. Hace muchos eones, uno de nosotros descubrió la *notación posicional*. Como tenemos dieciséis dedos, nuestros números utilizan una base de dieciséis, del mismo modo que los vuestros emplean una de diez. Cada dígito representa una potencia de dieciséis: 1, 16, 256, 4096, etc."

"¿Cómo funciona exactamente?", dijo Big Ed, mirando ansiosamente el mantel limpio que había puesto hacia algunos días.

"He aquí un número típico", dijo el visitante, garabateando repentinamente en el mantel. "Nuestro número A5B1 representa:

$$A \times 16^3 + 5 \times 16^2 + B \times 16^1 + 1 \times 16^0$$

"Pero, ¿qué son las Aes y las Bes?", preguntó Big Ed.

"Ah, me olvidaba. Cuando contamos con nuestros dedos decimos 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. De hecho, éstos son sólo quince dedos; nuestro último dedo representa el número después de F, nuestro 10, que es vuestro dieciséis."

"¡Ajá! Luego A representa nuestro 10, B nuestro 11, etc.", dijo Ed, dibujando una tabla como muestra la figura 2.1.

"Exactamente!", dijo el visitante. "Ahora puedes ver que el número A5B1, en nuestra base 16, es igual a vuestro número 42,417."

$$\begin{aligned} A \times 16^3 + 5 \times 16^2 + B \times 16^1 + 1 \times 16^0 &= \\ 10 \times 4096 + 5 \times 256 + 11 \times 16 + 1 \times 1 &= 42,417 \end{aligned}$$

"Bueno, resumiendo, vuestro sistema numérico decimal es realmente espantoso. Estuvimos a punto de no comerciar con vosotros después de descubrir que utilizabais números decimales. Afortunadamente, sin embargo, descubrimos que la base numérica que utilizáis con mayor frecuencia en vuestros ordenadores era la hexadecimal, que en nuestra base dieciséis."

¹ N. del T.: Un "hacker" en Estados Unidos es un maníaco de los ordenadores. Recuérdese que Big Ed se apellida Hackenbyte.

"Espera un segundo", exclamó Big Ed, "utilizamos *binarios* en nuestros ordenadores".

"Bueno, sí, por supuesto, a nivel de *hardware*, pero utilizáis el hexadecimales como una especie de abreviatura para representar valores de datos e instrucciones de memoria. Después de todo, el binario y el hexadecimales son casi idénticos." Continuó después de observar la perplejidad de Big Ed.

| DECIMAL | BASE 16 |
|---------|---------|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | A |
| 11 | B |
| 12 | C |
| 13 | D |
| 14 | E |
| 15 | F |

Figura 2.1. Representación de la base 16

"Mira, supón que tengo el número A1F5. Lo representaré alzando mis manos (Fig. 2.2). Los ocho dedos de la mano a tu derecha representan los dos dígitos hexadecimales de F5. Los ocho dedos a tu izquierda representan los dos dígitos hexadecimales de A1. Cada dígito hexadecimal se representa con cuatro dedos, que contienen cuatro bits o el dígito en binario. ¿Entendido?"

Big Ed se rascó la cabeza. "Veamos, 5 en binario es 0101, y se representa por esos cuatro números. El grupo siguiente de cuatro representa la F, que es en realidad 15, o binario 1111. El grupo siguiente... ¡Oh, claro! En lugar de escribir 1010000111110101, sólo escribís la notación abreviada A1F5."

"¡Big Ed, no sólo sirves el mejor chile de este lado de Altair; eres un matemático de los pies a la cabeza!", exclamó el visitante al tiempo que gesticulaba con su verde y escamosa mano de ocho dedos.

Big Ed miró la moneda de cobre de dieciséis lados dejada como propina y empezó a imaginar sobre el mantel...

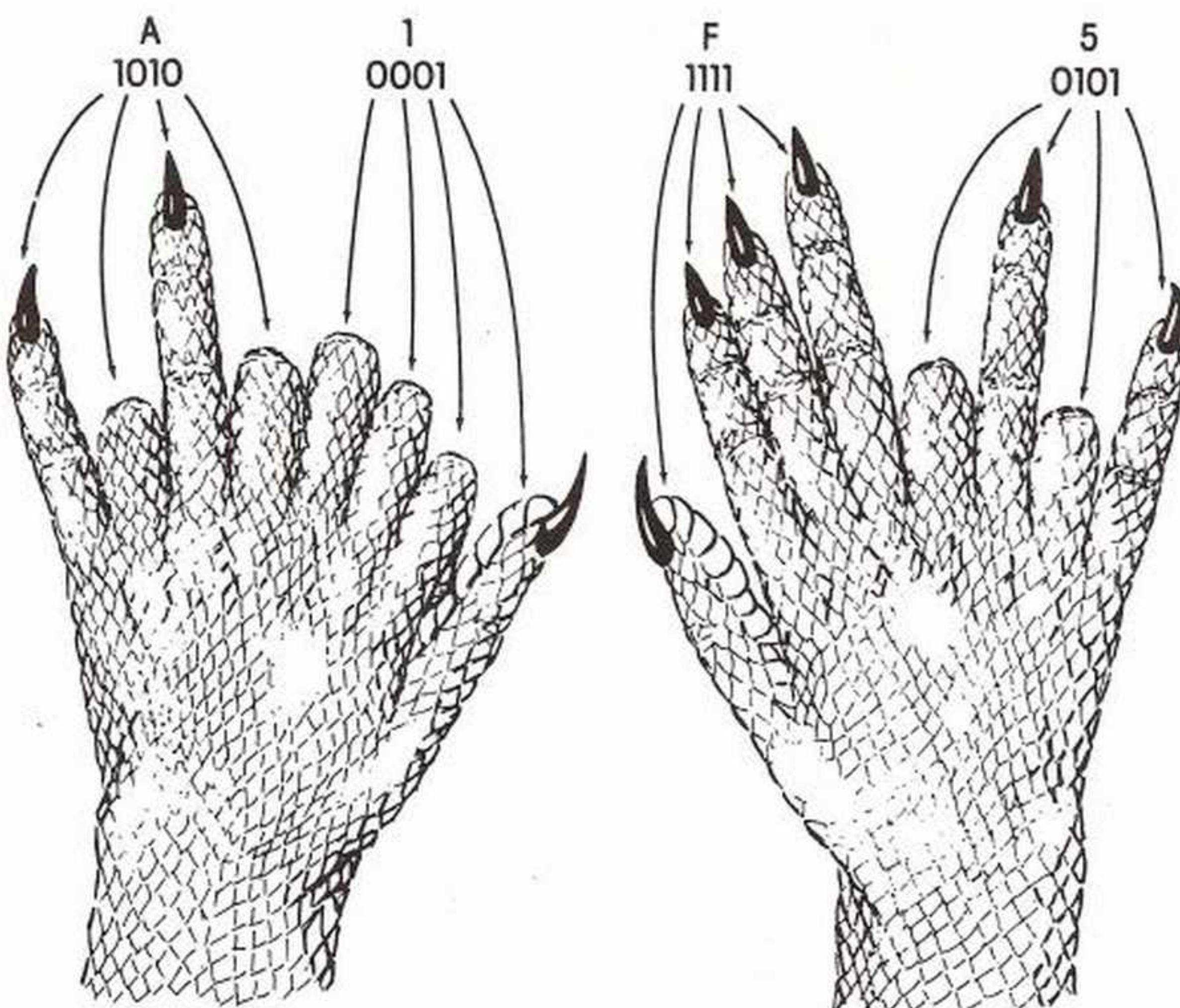


Figura 2.2. Abreviatura hexadecimal

Hexadecimal

El visitante de Ed tenía razón. El sistema hexadecimal se usa en microordenadores porque es un método adecuado para acortar largas series de unos y ceros binarios. Cada grupo de cuatro bits se puede convertir en un valor hexadecimal de 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E o F, como muestra la tabla 2.1.

Pasar de binario a hexadecimal es algo muy simple. Empezando por el bit de la derecha (bit 0), divida el número binario en grupos de cuatro. Si no tiene un múltiplo entero de cuatro (4, 8, 12, etc.), quedan algunos bits vacíos a la izquierda; en este caso, rellene a ceros sin más. Después, convierta cada grupo de cuatro bits en un dígito hexadecimal. El resultado es la representación hexadecimal del valor binario, que es la cuarta parte de largo en caracteres. La figura 2.3 muestra un ejemplo. Para pasar de hexa-

Tabla 2.1. Representación binaria, decimal y hexadecimal

| Binaria | Decimal | Hex. |
|---------|---------|------|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 4 | 4 |
| 0101 | 5 | 5 |
| 0110 | 6 | 6 |
| 0111 | 7 | 7 |
| 1000 | 8 | 8 |
| 1001 | 9 | 9 |
| 1010 | 10 | A |
| 1011 | 11 | B |
| 1100 | 12 | C |
| 1101 | 13 | D |
| 1110 | 14 | E |
| 1111 | 15 | F |

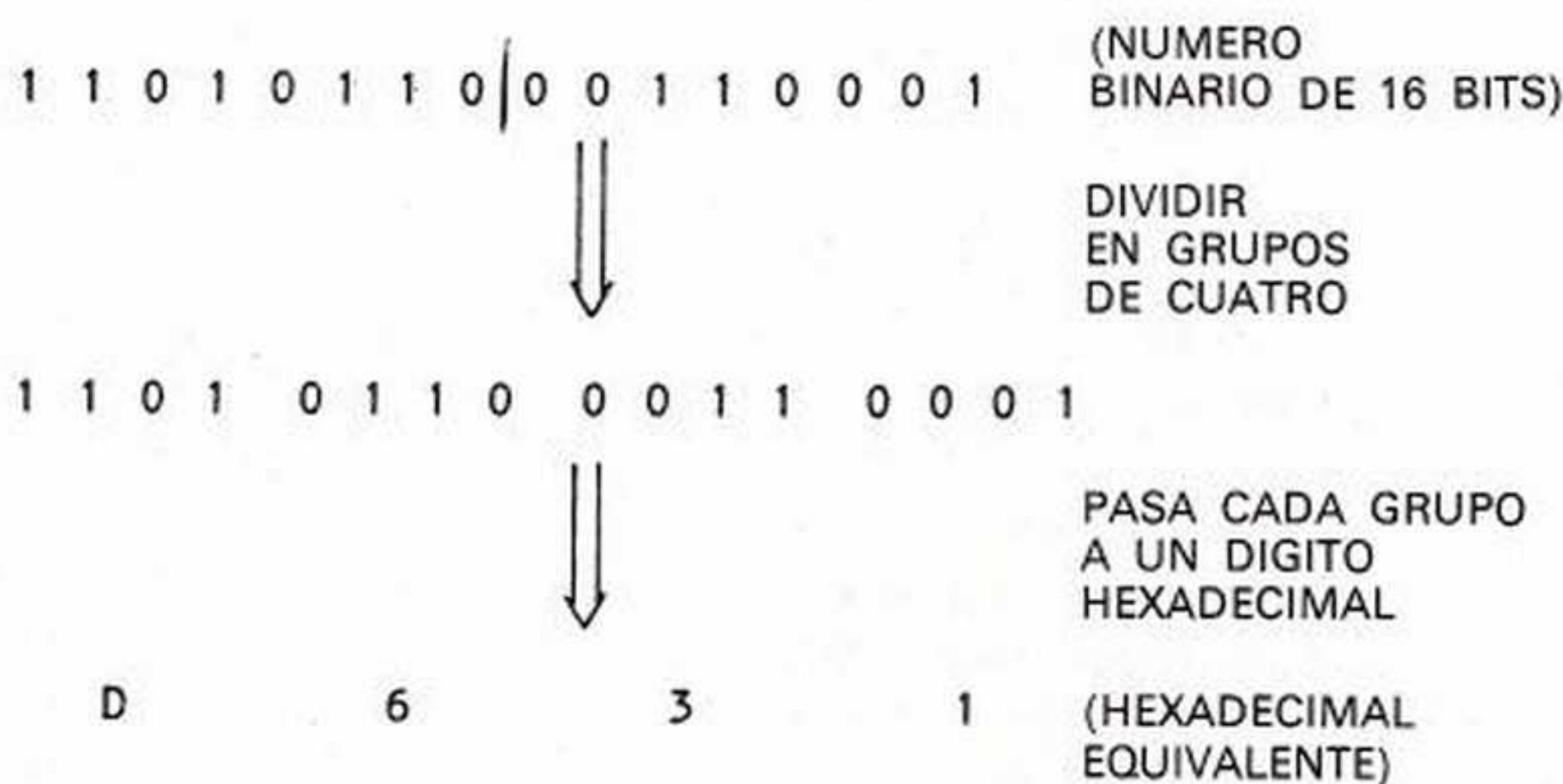


Figura 2.3. Paso de binario a hexadecimal

decimal a binario, haga a la inversa. Tome cada dígito hexadecimal y conviértalo en un grupo de cuatro bits. El ejemplo está en la figura 2.4.

La notación hexadecimal se usa para los microprocesadores Z-80, 6502, 6809 y muchos otros.

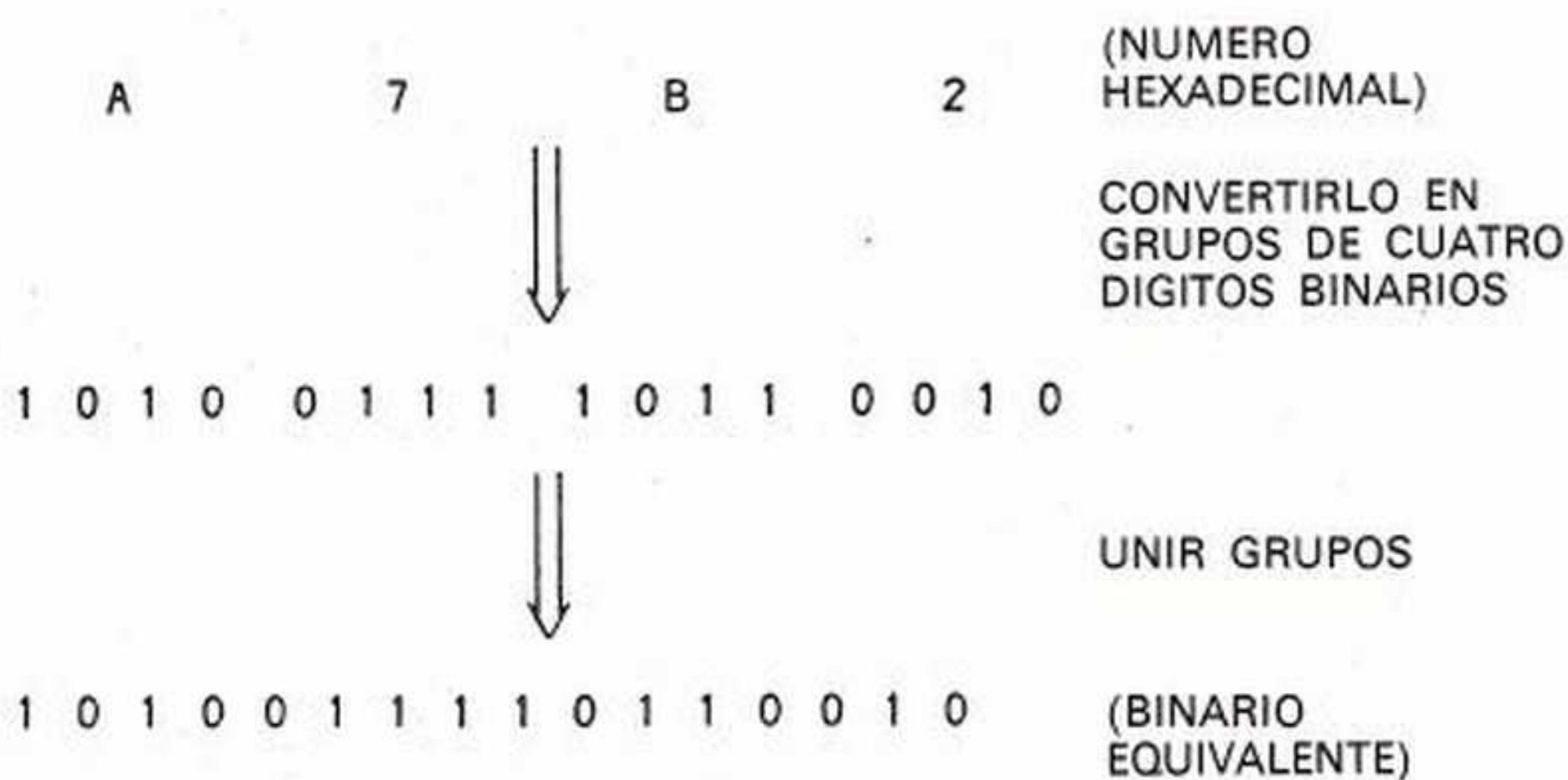


Figura 2.4. Paso de hexadecimal a binario

Paso de hexadecimal a decimal

Convertir binario en hexadecimal es fácil. ¿Lo es también entre decimal y hexadecimal? Se pueden aplicar muchos de los principios y técnicas tratados en el primer capítulo.

Para pasar de hexadecimal a decimal por el método de las potencias de dieciséis, tome cada dígito hexadecimal y multiplíquelo por la potencia adecuada de dieciséis, como hizo Big Ed. Para pasar 1F1E, por ejemplo, haríamos:

$$1 \times 4096 + 15 \times 256 + 1 \times 16 + 14 \times 1 = 7966$$

El método doblar y sumar puede también adaptarse a multiplicar por dieciséis y sumar (de hecho, este esquema sirve para cualquier base numérica). Tome el dígito hexadecimal de la izquierda y multiplíquelo por 16. Súmelo al siguiente. Multiplique el resultado por 16. Sume el siguiente dígito. Repita el proceso de multiplicación y sume hasta que el último dígito de la derecha sea operado. La figura 2.5 muestra este procedimiento.

Debería probar este procedimiento y comparar los resultados con los de doblar y sumar del sistema binario (no hace falta decirlo: más vale que el resultado sea el mismo).

Paso de decimal a hexadecimal

El método de la resta sucesiva de potencias de dieciséis no es muy práctico esta vez, ya que tendría que hacer 15 restas para obtener un dígito hexadecimal. El método análogo “dividir por dieciséis y guardar los

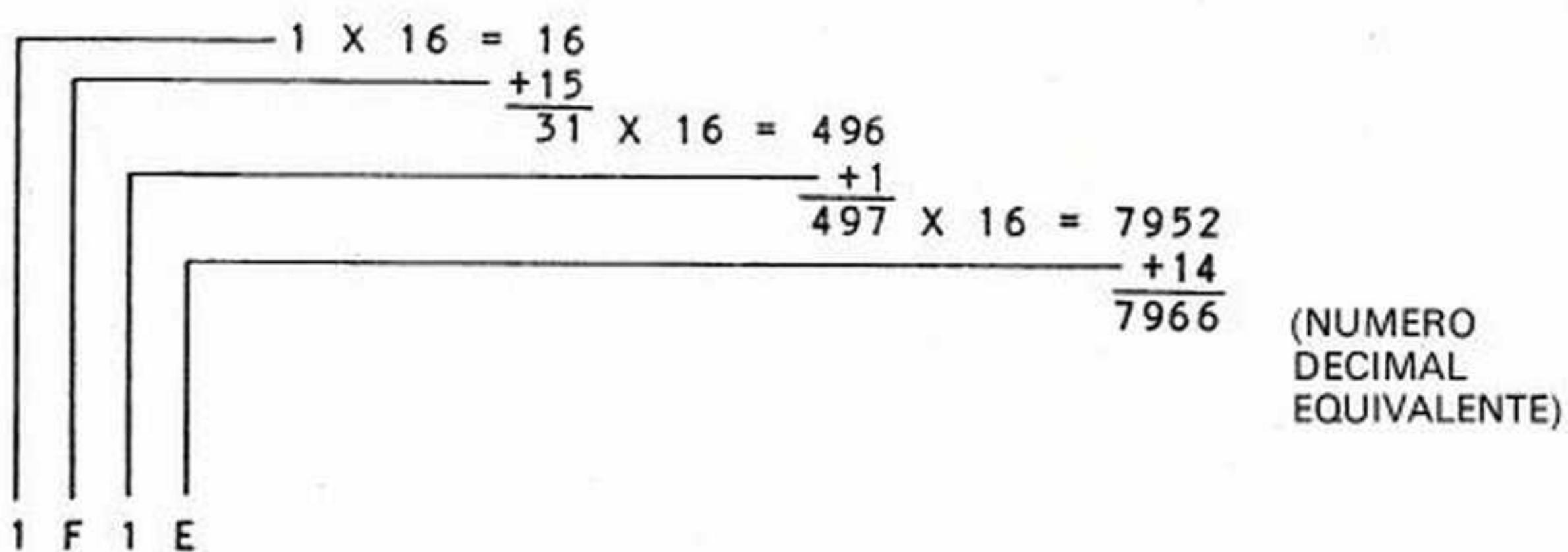


Figura 2.5. Paso de hexadecimal a decimal por el método multiplicar por dieciséis y sumar

restos”, sin embargo, es muy práctico, como muestra la figura 2.6. Tomemos, como ejemplo, el valor decimal 48,555. Dividiendo por dieciséis se obtiene un valor de 3034, con un resto de 11 (B en hexadecimal). Entonces, dividiendo 3034 entre 16 da 189, con un resto de 10 (A en hexadecimal). Dividiendo 189 por 16, resulta 11, con un resto de 13 (D hexadecimal). Finalmente, dividiendo 11 por 16 da 0, con un resto de 11. Los restos, en orden inverso, son el equivalente en hexadecimal BDAB.

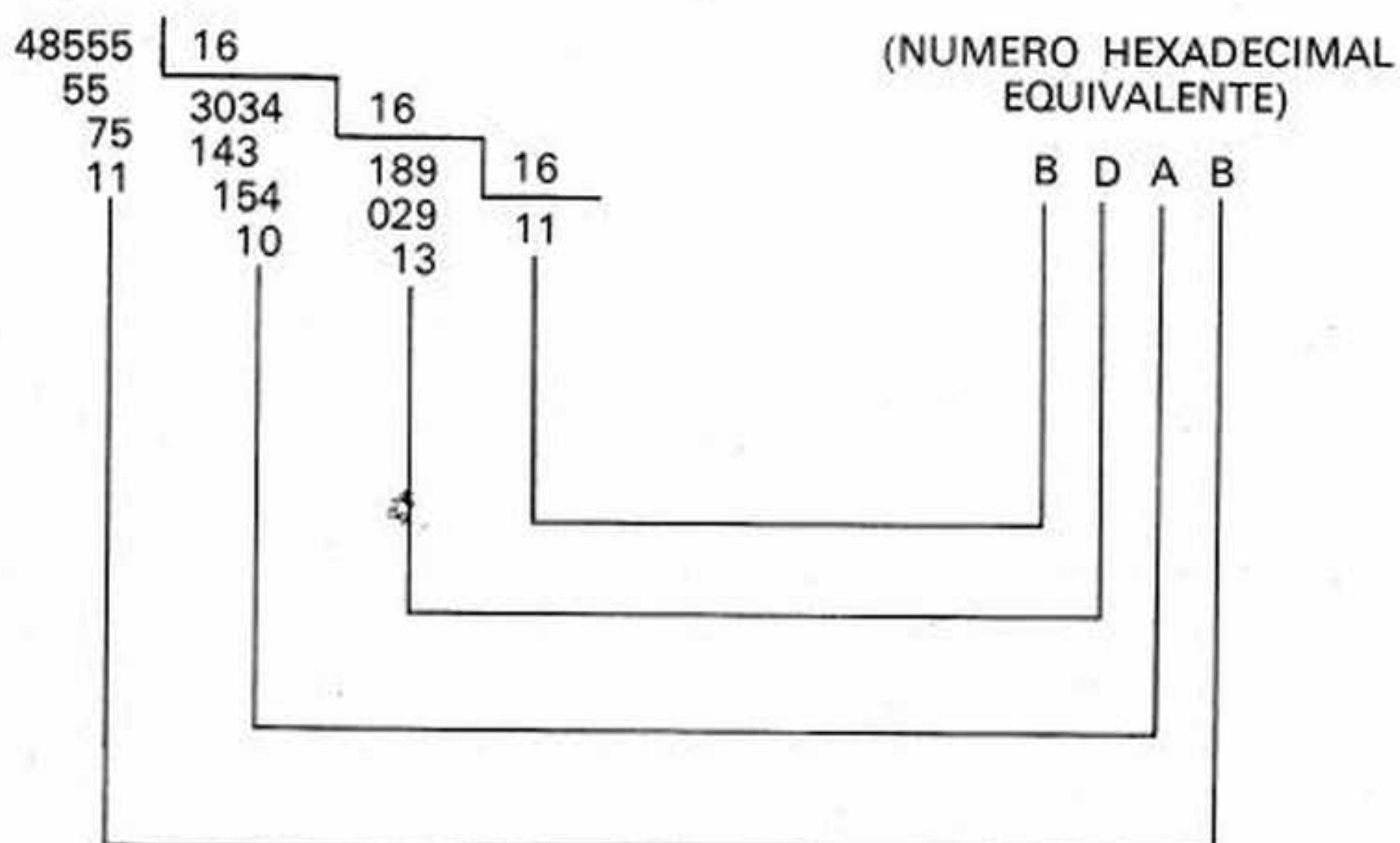


Figura 2.6. Paso de decimal a hexadecimal

Octal

El sistema hexadecimal es la base más usada en microordenadores. Sin embargo, el octal o base 8 se emplea también. Este último se utiliza preferentemente en los microprocesadores 8080. Como el Z-80 que usan muchos microordenadores, es una mejora del 8080; muchas instrucciones de éste sirven para el Z-80. Algunas de estas instrucciones utilizan campos de tres bits y su posición es tal que la representación octal es adecuada.

Los valores octales emplean potencias de 8, es decir, la posición 1 es para la potencia de cero (8^0), la posición 2 es para la primera potencia (8^1), la posición 3 es para la segunda potencia (8^2), etc. Aquí no se plantea el problema de asignar nombres a los nuevos dígitos, como sucedía en los hexadecimales A a F. Los dígitos octales son 0, 1, 2, 3, 4, 5, 6, 7. Cada dígito se puede representar por tres bits.

Paso entre binario y octal

El paso de binario a octal es similar a la conversión a hexadecimal. Para pasar de binario a octal, agrupe los bits en grupos de tres, empezando por la derecha. Si opera con números de 8 ó 16 bits, sobrarán algunos. Rellene a ceros. Después, cambie cada grupo de tres bits por un dígito octal. Un ejemplo se da en la figura 2.7. Para pasar de octal a binario, efectúe el proceso inverso.

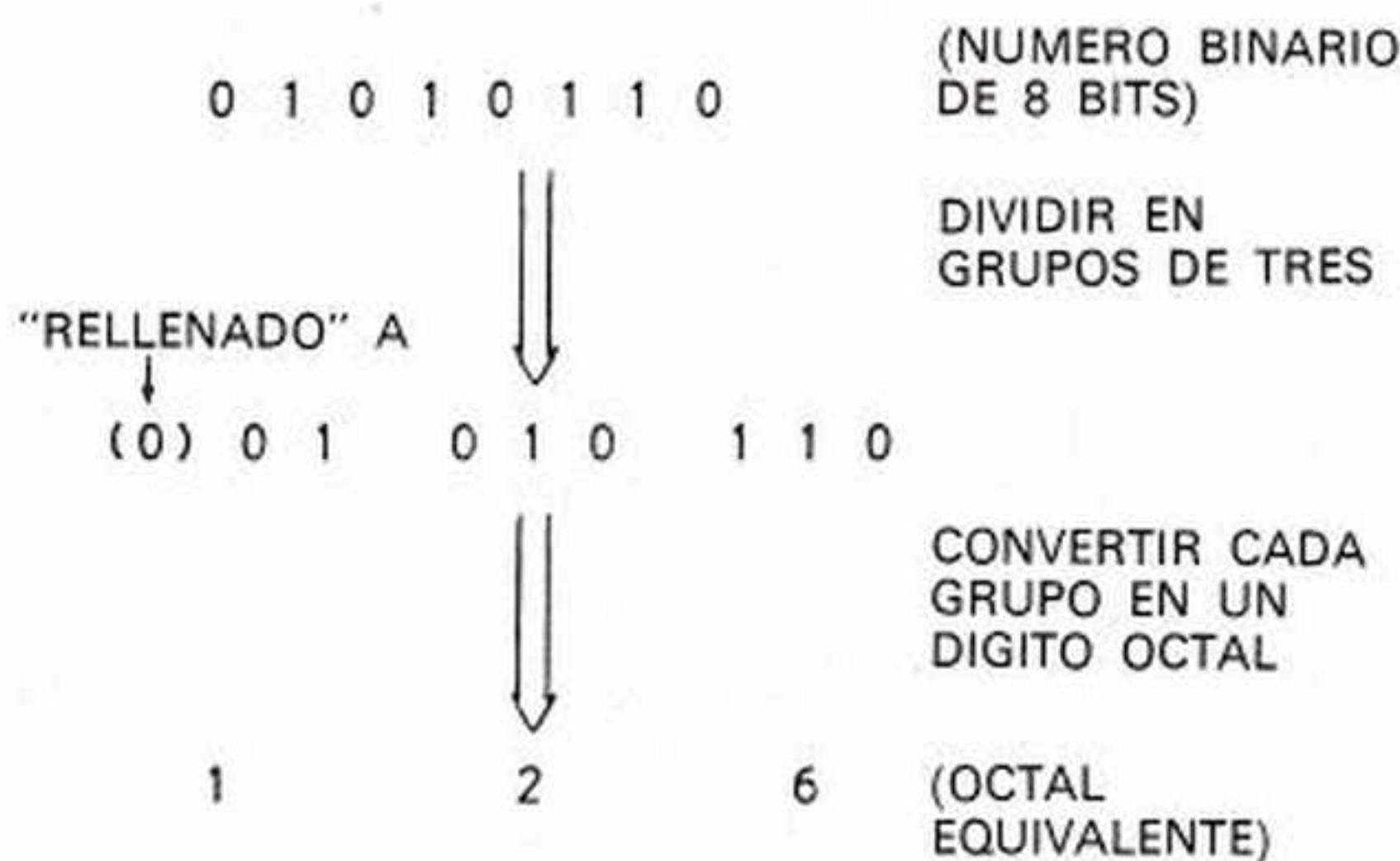


Figura 2.7. Paso de binario a octal

Paso entre octal y decimal

El paso de octal a decimal puede llevarse a cabo por el método de las potencias de ocho o por el de multiplicar por ocho y sumar. En el primer método, multiplique el dígito octal por la potencia de ocho. Para pasar el octal 360, por ejemplo, haríamos:

$$\begin{aligned}3 \times 8^2 + 6 \times 8^1 + 0 \times 8^0 &= \\3 \times 64 + 6 \times 8 + 0 \times 1 &= \\192 + 48 + 0 &= 240\end{aligned}$$

Utilizando el método multiplicar por ocho y sumar, tome el dígito octal de la izquierda y multiplíquelo por ocho. Sume el resultado al siguiente dígito. Multiplique este resultado por ocho y súmelo al siguiente. Repita este proceso hasta que el último dígito de la derecha haya sido sumado. Este procedimiento se muestra en la figura 2.8.

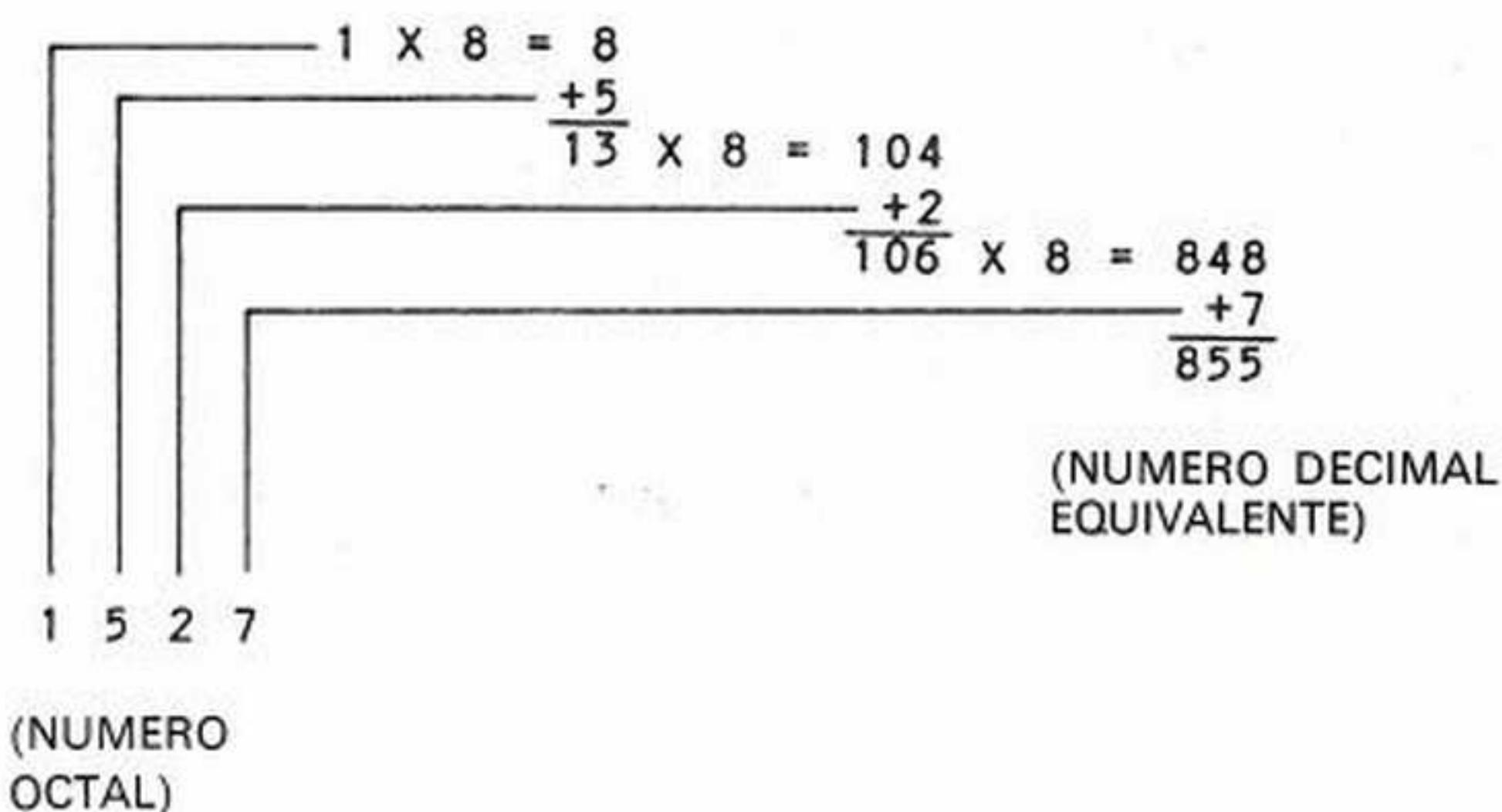


Figura 2.8. Paso octal a decimal mediante multiplicación por ocho y suma

Para pasar de decimal a octal se adopta de nuevo la técnica de "dividir y guardar los restos": divida el número octal por ocho y guarde el resto. Divida el resultado de nuevo y repita hasta que el resto sea menor que ocho. Los restos, en orden inverso, son el número octal equivalente. La figura 2.9 muestra un ejemplo.

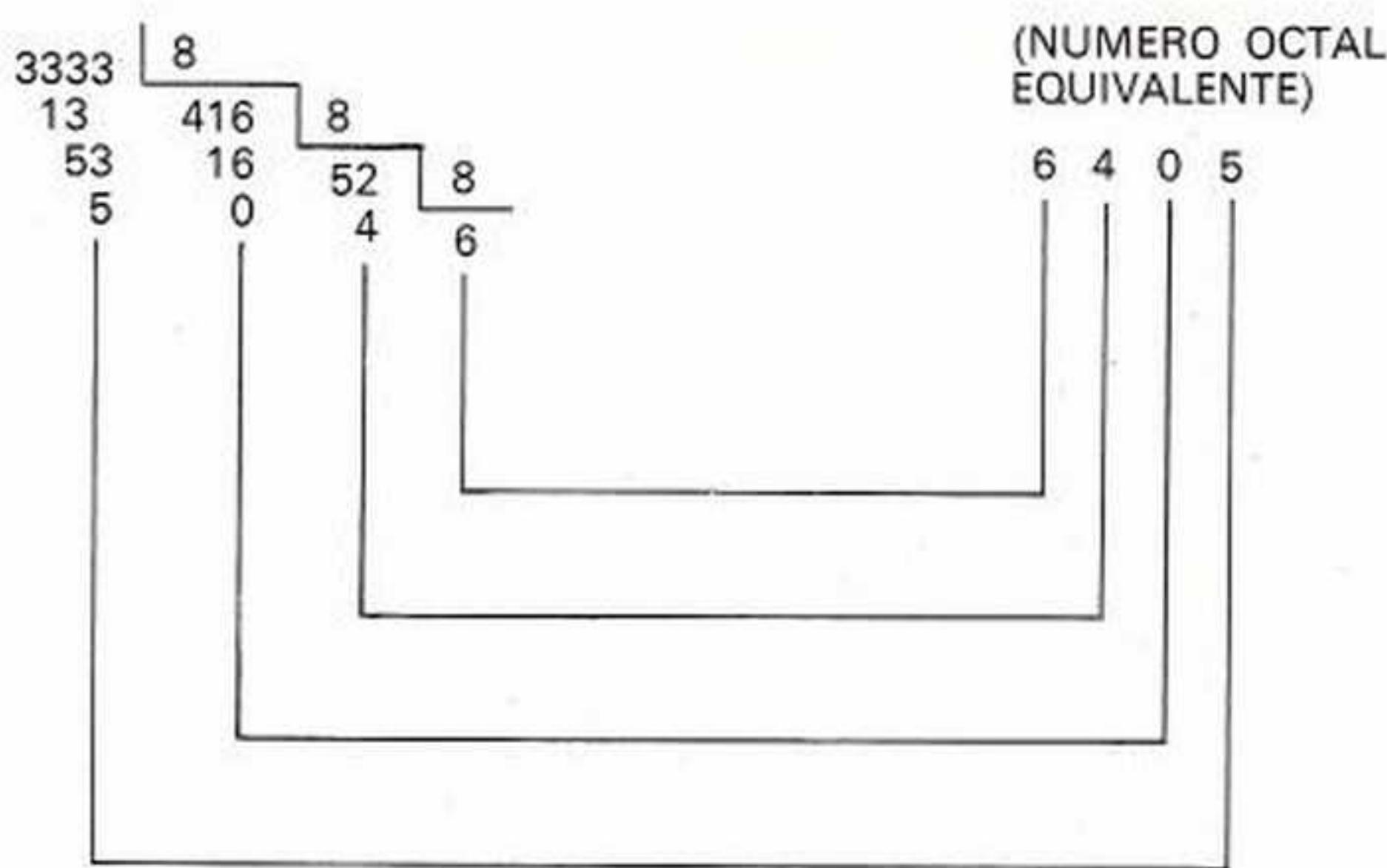


Figura 2.9. Paso de decimal a octal por el método “dividir y guardar los restos”

Trabajando con otras bases numéricas

Aunque las bases más utilizadas en microordenadores son la octal y la hexadecimal, se puede operar con cualquier base, ya sea base 3, base 5 o base 126. Un popular paquete de programas de Microsoft, muMath®, permite utilizar casi cualquier base para un extenso número de dígitos de precisión.

Dos ejemplos del uso de cualquier base pueden ser interesantes. Una vieja técnica para comprimir tres caracteres en dos bytes emplea la *base 40*. Cada uno de los caracteres alfabéticos de la A a la Z, los dígitos del 0 al 9 y cuatro caracteres especiales (punto, coma, interrogación y admiración o cualesquiera otros), forman un código que corresponde del 0 al 39 en decimal. Como el mayor número de tres dígitos en base 40 es $39 \times 40^2 + 39 \times 40^1 + 39 \times 40^0$, ó 63,999, los tres dígitos en base 40 pueden guardarse perfectamente en dieciséis bits (dos bytes). Generalmente, los tres caracteres deberían contenerse en tres bytes. El resultado de esta compresión consiste en el ahorro de un 50 por 100 del espacio de la memoria para texto utilizando sólo de la A a la Z, del 0 al 9 y cuatro caracteres especiales.

Un segundo ejemplo sería el proceso especial de tres en raya americano (*tic-tac toe*). Cada uno de los nueve elementos de un tablero de tres en raya contiene un “cuadrado”, un “círculo” o una “cruz”. Como hay tres caracteres, puede utilizarse ventajosamente una representación en base 3 (espacio = 0, círculo = 1, cruz = 2). El mayor número en base 3 para este código

sería $2 \times 3^8 + 2 \times 3^7 + 2 \times 3^6 + 2 \times 3^5 + 2 \times 3^4 + 2 \times 3^3 + 2 \times 3^2 + 2 \times 3^1 + 2 \times 3^0$, ó 19,682, que de nuevo puede guardarse perfectamente en dieciséis bits o dos bytes.

Los números en otras bases pueden pasarse a decimal, y viceversa, por el método “multiplicar por la base y sumar” y por el de “dividir por la base y guardar los restos”, de manera parecida a los números octales y hexadecimales.

Convenios estándar

En el resto de este libro emplearemos ocasionalmente el sufijo “H” para los números hexadecimales. El número “1234H”, por ejemplo, significará 1234 hexadecimal y no 1234 decimal. (También será equivalente &H1234 en algunas versiones del BASIC.) Asimismo, expresaremos las potencias de la misma forma que lo hace el BASIC. En lugar de exponentes utilizaremos una flecha hacia arriba (\uparrow) para expresar exponentiales (elevar un número a una potencia). El número 2^8 se representará $2 \uparrow 8$, 10^5 será $10 \uparrow 5$ y 10^{-7} ($0 \frac{1}{10}^7$) será $10 \uparrow -7$.

En el siguiente capítulo trataremos los *números con signo*. Mientras tanto, trabaje sobre los siguientes ejercicios en hexadecimal, octal y bases especiales.

Ejercicios

1. ¿Qué representa el número hexadecimal 9E2 en potencias de 16?
2. Haga una lista de los equivalentes hexadecimales a los decimales 0 a 20.
3. Pasar los siguientes números binarios a hexadecimal: 0101, 1010, 10101010, 01001111, 1011011000111010.
4. Pasar los siguientes números hexadecimales a binario: AE3, 999, F232.
5. Pasar los siguientes números hexadecimales a decimales: E3, 52, AAAA.
6. Pasar los siguientes números decimales a hexadecimal: 13, 15, 28, 1000.
7. La máxima dirección de memoria en un microordenador de 64K es 64,535. ¿Cuánto es en hexadecimal?
8. Pasar los siguientes números octales a decimales: 111, 333.
9. Pasar los siguientes números decimales a octal: 7, 113, 200.
10. ¿Qué puede decir del número octal 18? (Límite su respuesta a mil palabras o menos, por favor.)
11. En un sistema numérico en base 7, ¿cuál sería el equivalente decimal a 636?

Números con signo y notación en complemento a dos

Hasta ahora hemos hablado de los *números binarios sin signo*, que sólo representan valores positivos. En este capítulo aprenderemos cómo representar valores positivos y negativos en microordenadores.

Big Ed y el bínaco

“¡Hola, muchacho! ¿Quéquieres?”, preguntó Big Ed a un cliente delgado con un abrigo muy largo de colores y brocado.

“Tomaré un *chop suey*”, dijo el cliente.

“¿Qué traes ahí?”, preguntó Big Ed, cotilleando un artilugio de aspecto extraño que parecía un ábaco, con sólo unas pocas cuentas. “Parece un ábaco.”

“¡No, es un *bínaco*!”, dijo el cliente. “Iba a ser mi camino hacia la fama y la fortuna, pero, jay!, el que busca a aquellas dos como compañeras en el camino de la vida sólo encontrará desengaños. ¿Te importaría escucharme?”

Ed invitó con sus manos a que comenzara su historia, sabiendo que no tenía elección; era una pesada tarde.

“Este aparato es como un ábaco, pero funciona con números binarios.

Por eso tiene dieciséis columnas de ancho y una sola cuenta por columna (Fig. 3.1). Pasé años desarrollándolo y sólo recientemente he intentado comercializarlo a través de alguna firma de microordenadores de aquí. ¡Todas lo rechazaron!"

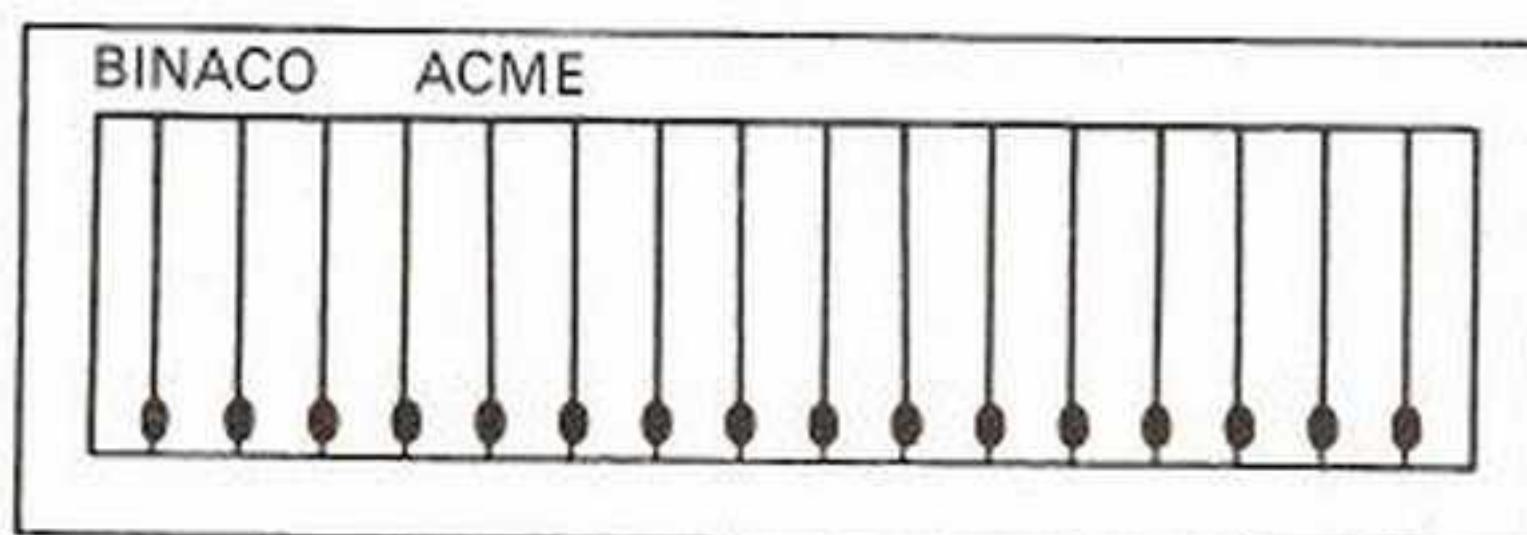


Figura 3.1. El bínaco

"¿Por qué, hombre?", dijo Ed, sirviéndole el *chop suey*.

"Puede representar cualquier número de cero a 65,535, y uno puede sumar o restar fácilmente números binarios con él. Pero no hay modo de representar números negativos."

Justo cuando pronunció estas palabras, Bob Borrow, el ingeniero de Inlog, entró. "No he podido evitar el escuchar tu historia", dijo. "Creo que puedo ser de alguna ayuda.

Verás, los ingenieros informáticos se enfrentaron con tu problema hace muchos años. Muchas veces, basta simplemente con almacenar un número absoluto. Por ejemplo, la mayoría de los microprocesadores pueden almacenar 65,536 direcciones separadas, numeradas de 0 a 65,535. No hay razón para tener números negativos en este caso. Por otro lado, los microordenadores son capaces de realizar operaciones aritméticas con *datos* y deben ser capaces de almacenar valores negativos y positivos.

Hay diferentes esquemas para representar números negativos, observa." Ed respingó cuando Bob se dirigió al mantel.

"Podría sencillamente añadir una cuenta más, la diecisiete, en el extremo izquierdo del bínaco. Si la cuenta estuviese arriba, el número representado sería positivo; si la cuenta estuviese abajo, el número sería negativo. Este esquema se denomina representación *signo/magnitud*." (Véase la figura 3.2.)

"Es cierto", dijo el forastero. "Desde luego, implicaría un nuevo diseño." Meditó, mirando el invento.

"Espera, todavía no he acabado. Tengo un proyecto para que no tengas que reformar nada. Se llama notación en complemento a dos, y te permite almacenar números de -32,768 a +32,767 sin ninguna modificación."

"¡Oh, señor! Si usted pudiera hacerlo...", dijo el forastero.

"En este proyecto, haremos que la cuenta decimosexta del final, en la posición quince, represente el *bit de signo*. Si la cuenta está arriba o 0, el signo

(COLUMNAS
AÑADIDA)

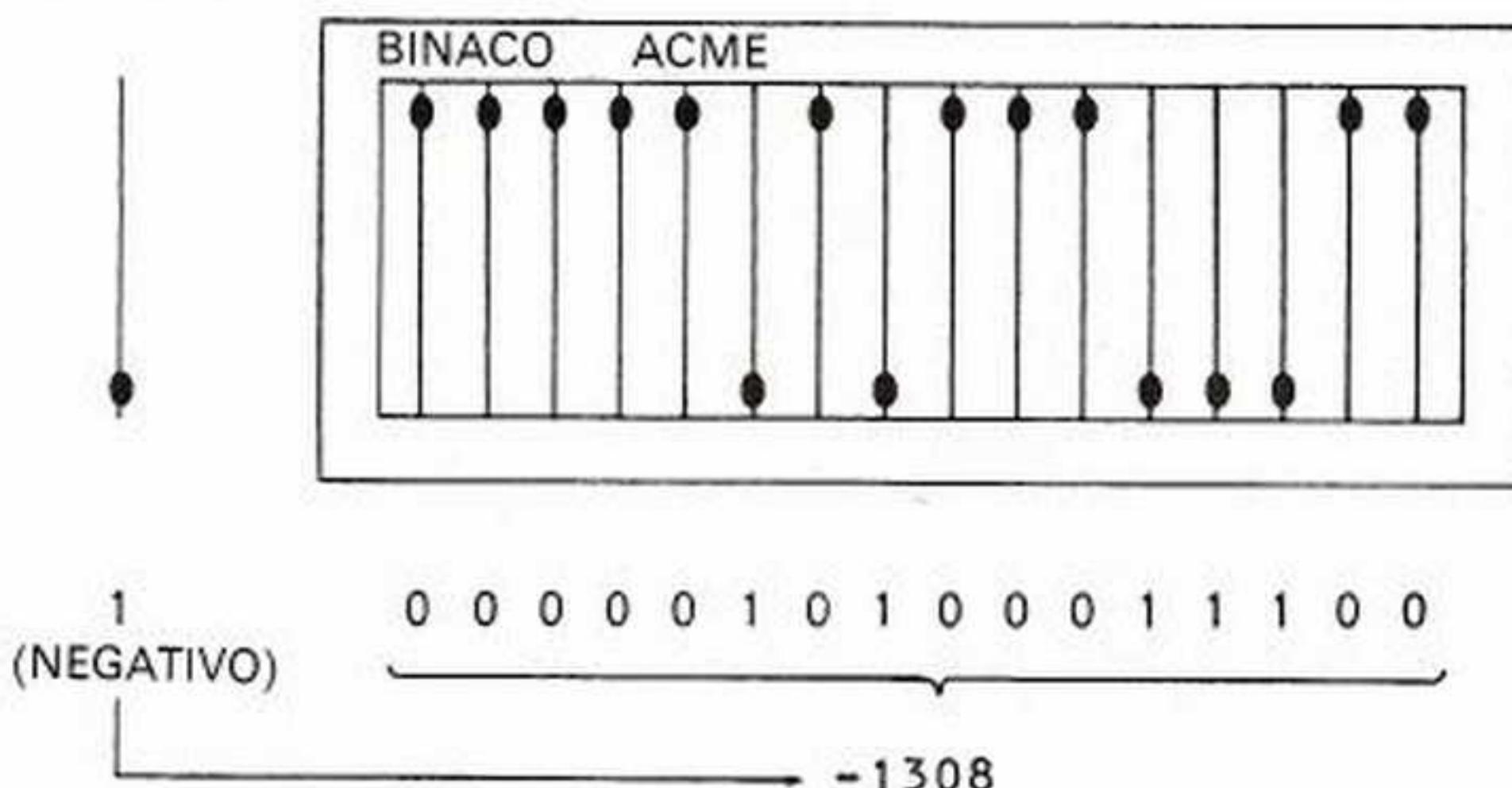


Figura 3.2. Representación signo/magnitud

será positivo y las cuentas restantes contendrán el valor del número. Desde el momento en que tenemos sólo quince cuentas, el número representado será desde 0 (000 0000 0000 0000) a 32,767 (111 1111 1111 1111)."

"¿Y qué sucede con los números negativos?", preguntó el forastero.

"Es lo que iba a decir. Si la cuenta de la posición 15 está abajo o 1, entonces el número representado es negativo. En este caso, mueva todas las cuentas que están arriba o 0 hacia abajo o 1. Mueva todas las cuentas que estén abajo o 1 hacia arriba o 0. Finalmente, sume 1."

"Gracias por su ayuda, señor", dijo el forastero con ojos de incrédulo, mientras cogía el aparato y se dirigía a la puerta.

"¡No, espere, el sistema funciona!", exclamó Bob. "Mire, permítame mostrarle. Suponga que tiene la configuración 0101 1110 1111 0001. La cuenta del signo es un 0, luego el número representa 101 1110 1111 0001 (en hexadecimal, 5EF1; o en decimal, 24,305). Ahora, suponga que la configuración es 1001 1010 0001 0101. La cuenta del signo es un 1, luego el número es negativo. Ahora invierta la posición de todas las cuentas y sume una." (Véase la figura 3.3.)

"El resultado es 0110 0101 1110 1011. Ahora pasamos el número al decimal 26,091; además, el número representado es -26,091. Este esquema es el mismo que usan los microordenadores. ¡Podrá vender fácilmente su idea del bínaco a un fabricante, si le dice que emplea este sistema de notación en complemento a dos para los números negativos!"

El forastero parecía dudar. "A ver si entiendo esto. Si la cuenta del signo es un uno, ¿invierto todas las cuentas y añado uno? Déjeme probar unos pocos ejemplos." Movió las cuentas del bínaco y calculó sobre el mantel.

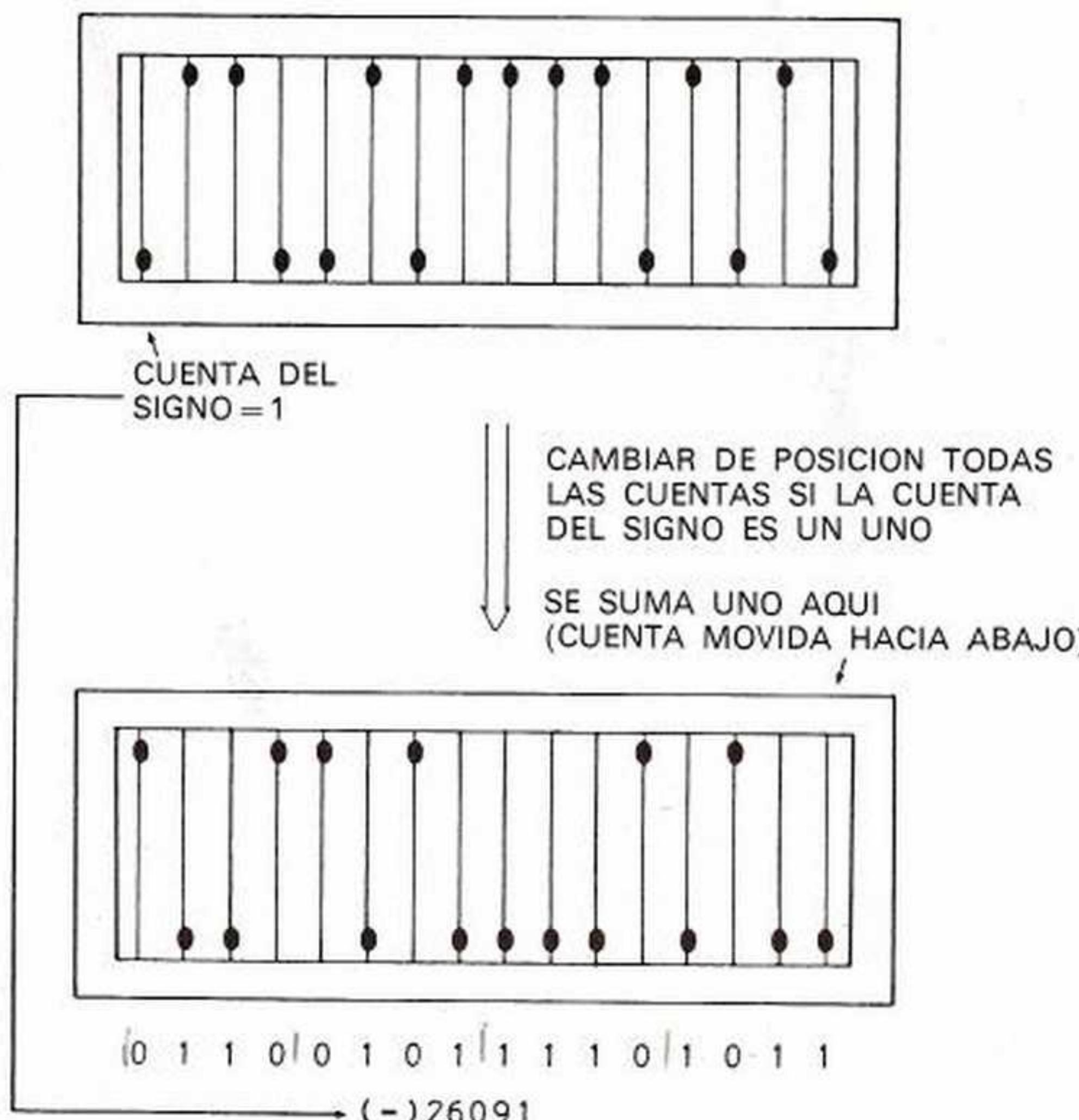


Figura 3.3. Notación en complemento a dos

Ed comenzó a desesperar. La tabla de lo que él obtuvo se muestra en la figura 3.4.

“Me parece que los números negativos de -1 a $-32,768$ podrían almacenarse por este procedimiento. Pero, ¿por qué es tan complicado?”

“Para que resulte más fácil en *hardware*, colega”, dijo Bob. “Este método implica que todos los números pueden sumarse o restarse sin necesidad de comprobar primero el signo de cada operando. Basta con seguir sumando o restando los números, y el resultado tendrá el signo correcto. Suponga que tenemos los dos números 0011 0101 0111 0100 y 1011 1111 0000 0000. Son $+13,684$ y $-16,640$, respectivamente. Los sumamos de la manera siguiente:

$$\begin{array}{r}
 0011 \quad 0101 \quad 0111 \quad 0100 \quad (+13,684) \\
 1011 \quad 1111 \quad 0000 \quad 0000 \quad (-16,640) \\
 \hline
 1111 \quad 0100 \quad 0111 \quad 0100 \quad (-2,956)
 \end{array}$$

| NUMERO EN COMPLEMENTO A DOS | REPRESENTACION DECIMAL |
|--------------------------------|---------------------------|
| 0111 1111 1111 1111 | +32,767 |
| 0101 0000 1010 0000 | +20,640 |
| 0000 0000 0000 0010 | +2 |
| 0000 0000 0000 0001 | +1 |
| 0000 0000 0000 0000 | 0 |
| 1111 1111 1111 1111 | -1 |
| 1111 1111 1111 1110 | -2 |
| 1111 1111 1111 1101 | -3 |
| 1111 0000 0000 0000 | -4,096 |
| 1000 0000 0000 0000 | -32,768 |

Figura 3.4. Funcionamiento en complemento a dos para diecisésis bits

El resultado es 1111 0100 0111 0100, ó -2,956. ¡Como debería ser!"

"Es increíble. Estudiaré esto, venderé mi bínaco y volveré forrado a casa."

"¿Dónde está tu casa?", preguntó Big Ed.

"En Brooklyn", dijo el forastero. "Adiós y gracias; algún día podréis decir que conocisteis a Michael O'Donahue."

Sumar y restar números binarios

Antes de ver el esquema del bínaco que representa números con signo (que duplica el sistema utilizado en todos los microordenadores actuales), veamos el tema de la suma y resta de números binarios en general.

Sumar números binarios es mucho más fácil que sumar números decimales. ¿Recuerda cuando tenía que aprender de memoria las tablas de sumar? Cuatro y cinco, nueve; cuatro y seis, diez, etc. En el sistema binario también hay tablas de sumar, pero son mucho más sencillas: 0 y 0 es 0, 0 y 1 es 1, 1 y 0 es 1, y 1 y 1 es 0, y nos llevamos una a la siguiente posición. Si ésta tiene 1 y 1, entonces la tabla tiene una quinta entrada de 1, y 1 y 1 da 1, llevándonos 1 a la siguiente posición. La figura 3.5 resume la tabla.

$$\begin{array}{r}
 0 \\
 +0 \\
 \hline
 0
 \end{array}
 \quad
 \begin{array}{r}
 0 \\
 +1 \\
 \hline
 1
 \end{array}
 \quad
 \begin{array}{r}
 1 \\
 +0 \\
 \hline
 1
 \end{array}
 \quad
 \begin{array}{r}
 1 \\
 +1 \\
 \hline
 1 \curvearrowleft 0
 \end{array}$$

ACARREO A LA POSICION SIGUIENTE

$$\begin{array}{r}
 1 \text{ ACARREO} \\
 1 \\
 +1 \\
 \hline
 1 \curvearrowleft 1
 \end{array}$$

ACARREO A LA POSICION SIGUIENTE

Figura 3.5. Suma de binarios

Probemos esto con dos números sin signo de ocho bits, los números con que hemos estado operando en anteriores capítulos. Supongamos que sumamos 0011 0101 y 0011 0111 (53 y 55, respectivamente).

$$\begin{array}{r}
 11 \quad 111 \quad \text{(Acarreos)} \\
 0011 \quad 0101 \quad (53) \\
 0011 \quad 0111 \quad (55) \\
 \hline
 0110 \quad 1100 \quad (108)
 \end{array}$$

Los unos encima de los operandos representan los acarreos que llevamos a la siguiente posición. El resultado es 0110 1100, ó 108, como esperábamos.

La resta es igual de fácil. 0 menos 0 es 0, 1 menos 0 es 1, 1 menos 1 es 0. Y, el más complicado, 0 menos 1 es 1, con un *acarreo negativo (borrow)* a la siguiente posición, de la misma forma que nos llevamos una en aritmética decimal. Esta tabla está resumida en la figura 3.6.

$$\begin{array}{r}
 0 \\
 -0 \\
 \hline
 0
 \end{array}
 \quad
 \begin{array}{r}
 1 \\
 -0 \\
 \hline
 1
 \end{array}
 \quad
 \begin{array}{r}
 1 \\
 -1 \\
 \hline
 0
 \end{array}
 \quad
 \begin{array}{r}
 0 \\
 -1 \\
 \hline
 -1 \curvearrowleft 1
 \end{array}$$

ACARREO NEGATIVO HACIA LA SIGUIENTE POSICION

Figura 3.6. Resta binaria

Probemos con algunos números. Restemos 0001 0001 de 0011 1010, ó 17 de 58:

$$\begin{array}{r} 1 \quad (\text{Acarreo negativo}) \\ 0011 \ 1010. \quad (58) \\ 0001 \ 0001. \quad -(17) \\ \hline 0010 \ 1001 \quad (41) \end{array}$$

El uno encima del operando representa el acarreo negativo de la siguiente posición. El resultado es 0010 1001, 41, como era de esperar.

Pongamos ahora otro ejemplo. Restemos 0011 1111 de 0010 1010, ó 63 de 42:

$$\begin{array}{r} 1111 \ 111. \quad (\text{Acarreos negativos}) \\ 0010 \ 1010. \quad (42) \\ 0011 \ 1111. \quad -(63) \\ \hline 1110 \ 1011 \quad (??) \end{array}$$

El resultado es 1110 1011, ó 235, un resultado que no nos esperábamos. Pero, espere; ¿podría ser?, ¿es posible? Si aplicamos las reglas de la representación en complemento a dos y consideramos 1110 1011 como un número negativo, entonces tenemos 0001 0100 después de cambiar todos los unos por ceros y todos los ceros por unos. Añadimos entonces uno para obtener 0001 0101. El resultado es -21, si ponemos el signo negativo, lo cual es correcto. ¡Parece como si nos viésemos forzados a usar los dichosos complemento a dos, queramos o no!

Representación en complemento a dos

Hemos cubierto bastante bien todos los aspectos de la notación en complemento a dos. Si el bit del extremo izquierdo de un valor de 8 ó 16 bits se considera el bit de signo, entonces ha de ser 0 (positivo) o 1 (negativo). La representación en complemento a dos puede, por consiguiente, obtenerse aplicando las reglas tratadas anteriormente: fijándonos en el bit de signo, cambiando todos los ceros por unos, todos los unos por ceros y añadiendo uno.

Los números se almacenan en forma de complemento a dos, como números de 8 ó 16 bits. El bit de signo ocupa siempre la posición extrema de la izquierda y es siempre (1) para un número negativo.

Los formatos para la representación en complemento a dos de 8 y 16 bits se muestran en la figura 3.7.

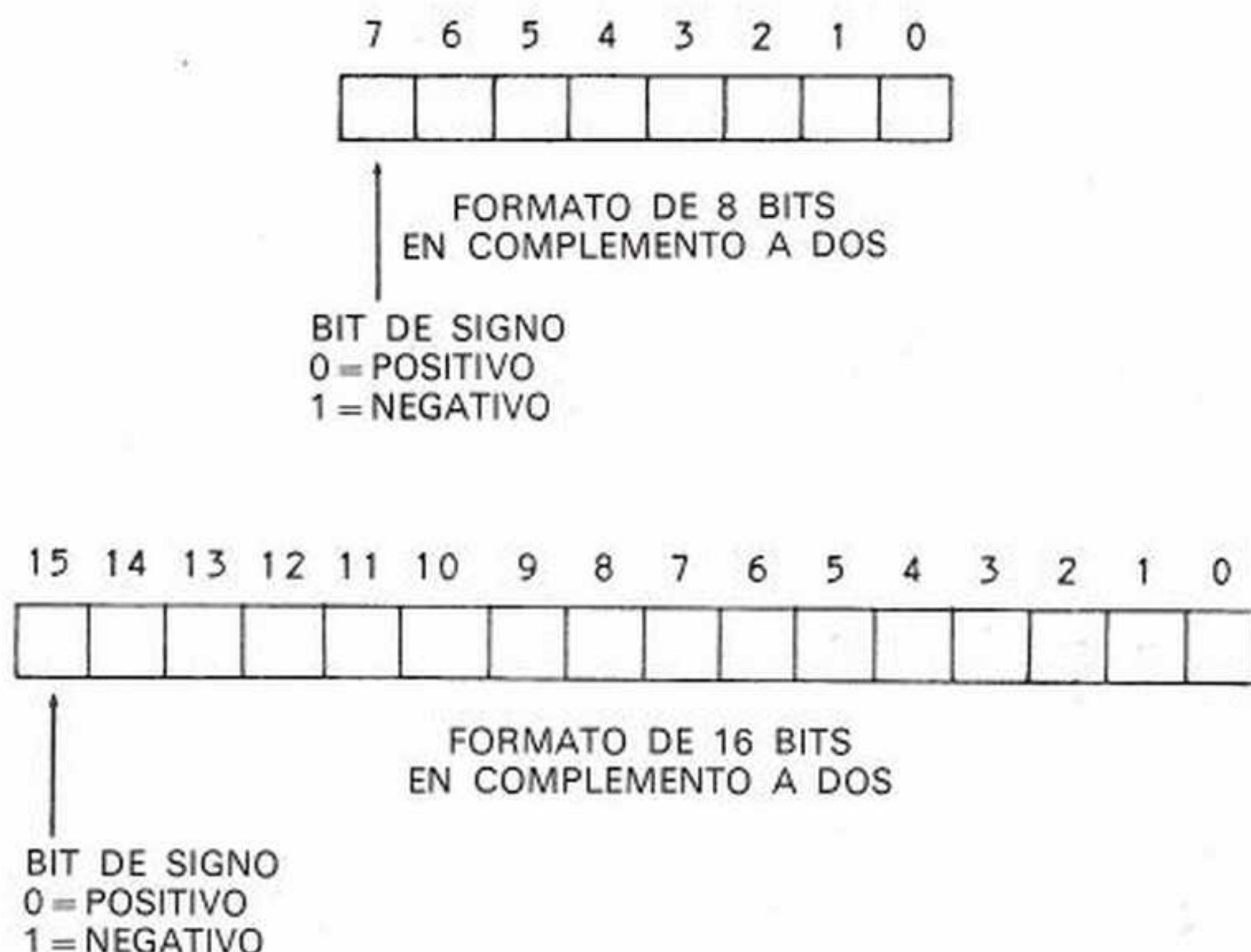


Figura 3.7. Formatos de complemento a dos

Los valores de 8 bits se utilizan para *desplazamientos* en las instrucciones del microprocesador para modificar la dirección de un operando guardado en memoria. Los valores de 16 bits se emplean como variables enteras en programas de BASIC. Por supuesto, cualquiera de ellos puede ser utilizado por el programador de BASIC o lenguaje ensamblador para representar lo que deseé.

Extensión del signo

Es posible sumar o restar números en complemento a dos, uno de 8 y otro de 16 bits. Cuando se realizan estas operaciones, el signo del menor número debe *extenderse* hacia la izquierda hasta que ambos tengan la misma longitud; es decir, todas las posiciones necesarias para que el menor número tenga el mismo tamaño que el mayor se llenan con el bit de signo. Si esto no se hace, el resultado será incorrecto. Tomemos, por ejemplo, la

suma del valor de 8 bits 1111 1111 (-1), y el valor de 16 bits 0011 1111 1111 1111 (+16,383). Si no se extiende el signo, tenemos:

$$\begin{array}{r} 0011 \quad 1111 \quad 1111 \quad 1111 \\ \quad \quad 1111 \quad 1111 \\ \hline 0100 \quad 0000 \quad 1111 \quad 1110 \end{array} \quad \begin{array}{l} (+16,383) \\ (-1) \\ (+16,638) \end{array}$$

El resultado aquí es +16,638; obviamente incorrecto. Si se extiende el signo correctamente a todas las posiciones de la izquierda, sin embargo, tenemos:

$$\begin{array}{r} 0011 \quad 1111 \quad 1111 \quad 1111 \\ 1111 \quad 1111 \quad 1111 \quad 1111 \\ \hline 0011 \quad 1111 \quad 1111 \quad 1110 \end{array} \quad \begin{array}{l} (+16,383) \\ (-1) \\ (+16,382) \end{array}$$

que es el resultado correcto.

Suma y resta en complemento a dos

Todos los microprocesadores usados en microordenadores tienen una instrucción para sumar y otra para restar dos números con signo de 8 bits. Además, algunos microprocesadores permiten la suma y la resta de dos números con signo de 16 bits.

Los dos operandos que aparecen en la suma o en la resta pueden tener cualquier configuración: dos números positivos, uno positivo y otro negativo, o dos números negativos. Al restar, hay que considerar la operación de la siguiente manera: al número que se resta (sustraendo) se le *invierte el signo* haciendo su complemento a dos: cambiando todos los unos por ceros, todos los ceros por unos, y añadiendo uno. Una vez hecho esto, la resta equivale a una suma del mismo número con el signo invertido. Pongamos un ejemplo. Supongamos que 1111 1110 (-2) se va a restar de 0111 0000 (+112). Primero se invierte el signo al sustraendo de -2. El complemento a dos de 1111 1110 es 0000 0010 (+2). Entonces, se hace la suma.

$$\begin{array}{r} 0111 \quad 0000 \quad (+112) \\ 0000 \quad 0010 \quad (+2) \\ \hline 0111 \quad 0010 \quad (+114) \end{array}$$

Hay que tener claro que el microprocesador no realiza esta operación de inversión antes de la resta. Esto es, simplemente, un medio adecuado para

ver lo que sucede en la resta, y tiene alguna relación con las operaciones de que hablaremos en el capítulo siguiente.

Antes de tratar algunas particularidades de la suma y la resta, como errores de desbordamiento y acarreos, y los indicadores de los microprocesadores ordinarios, intente hacer alguno de los siguientes ejercicios para adiestrarse en la suma y la resta en complemento a dos.

Ejercicios

1. Sume los siguientes números sin signo, dando los resultados en binario y decimal.

$$\begin{array}{r} 01 \\ 11 \\ \hline \end{array} \quad \begin{array}{r} 0111 \\ 1111 \\ \hline \end{array} \quad \begin{array}{r} 010101 \\ 101010 \\ \hline \end{array}$$

2. Reste los siguientes números sin signo, dando los resultados en binario y decimal.

$$\begin{array}{r} 10 \\ 01 \\ \hline \end{array} \quad \begin{array}{r} 0111 \\ 0101 \\ \hline \end{array} \quad \begin{array}{r} 1100 \\ 0001 \\ \hline \end{array}$$

3. Halle los complemento a dos de los siguientes números con signo, cuando sea necesario, para obtener los números decimales representados por:

$$01101111, 10101010, 10000000$$

4. ¿Cuál es la forma en complemento a dos de 8 bits de -1, -2, -3, -30, +5, y +127?
5. Extienda el signo de los siguientes números de 8 a 16 bits. Escriba los números representados antes y después.

$$01111111, 10000000, 10101010$$

6. Sume -5 a -300. (¡No, no, en binario!)
7. Reste -5 de -300 en binario.

Acarreos, errores de desbordamiento e indicadores

En este capítulo ampliaremos algunos de los conceptos introducidos en el anterior. Algunos de éstos son sencillos y otros no tanto, y todos ellos se refieren a números con signo. Analizaremos aquí algunos de ellos.

Este restaurante tiene una capacidad de +127 personas.

¡Evitando errores de desbordamiento!

Big Ed estaba sentado frente a una taza del famoso Java de Big Ed después de haberse marchado la multitud de ingenieros; en ese momento, se abrió la puerta del restaurante y entró un hombre uniformado.

“Dígame, señor, ¿en qué puedo servirle?”

“Póngame una taza de café y un suizo”, respondió el cliente.

“Veo que va de uniforme. ¿Pertenece usted a las Fuerzas Aéreas de Campo Moffett?”, preguntó Ed.

“N'a, soy transportista. Vine a esta zona en respuesta a un anuncio que solicitaba ‘especialistas en acarreos y errores de desbordamiento’ en Inlog y, aunque no sé lo que es un error de desbordamiento, puedo

acarrear lo que sea; así que pensé que daba el tipo, pero el jefe de personal se rio y me señaló la puerta."

"Creo que entiendo el problema, señor", dijo un cliente de constitución delgada que no aparentaba más de catorce años. Una tarjeta de referencia del Z-80 sobresalía del bolsillo de su camisa. "Soy un programador con mucha experiencia en errores de 'desbordamiento'."

"No sabía que trabajabais en transportes."

"Bueno, nosotros manejamos el traslado de datos de los procesadores, pero trabajamos bastante con el error de desbordamiento. ¿Le importa si le explico?" El transportista arqueó las cejas y esperó.

...después de varias tazas de café de Ed, el programador había hablado del sistema binario y de las operaciones aritméticas simples.

"Como puede ver, se puede almacenar cualquier valor de -128 a +127 en 8 bits, o cualquier valor de -32,768 a +32,767 en 16 bits. ¿Entendido?"

El transportista dijo: "Sí, salvo los números sin signo, en cuyo caso se extiende de 0 a 255 o de 0 a 65,535 ó 2^{N-1} , donde N es el tamaño del registro en bits."

"Vaya, usted aprende rápido", dijo el programador. "Ahora, continuando... Si realizamos una suma o resta cuyo resultado sea mayor de +32,767 o menor de -32,768 (o mayor de +127 o menor de -128), ¿qué ocurre?"

"Bueno, supongo que se obtiene un resultado incorrecto", dijo el transportista.

"Es cierto, se obtiene un resultado incorrecto. El resultado es demasiado grande en sentido positivo o negativo para poder ser almacenado en 8 ó 16 bits. En este caso, se produce un *error de desbordamiento* porque el resultado excede del valor que puede contenerse en 8 ó 16 bits. Sin embargo, hemos diseñado un *indicador* de error de desbordamiento como parte del microprocesador. Este indicador puede examinarse por un programador del lenguaje ensamblador para ver si se ha producido un error de desbordamiento después de una suma o de una resta. Y resulta que, cuando pasa lo mismo en operaciones sin signo, se produce un error de acarreo, que es lo que también pedían en el anuncio. No sólo tenemos un indicador de error de desbordamiento, sino un indicador de acarreo, otro de signo, otro de cero..."

"Espera un segundo. ¿Quieres decir que esa gente de personal quería un ingeniero informático especializado en aritmética de microprocesadores?", dijo el transportista con una mueca. "¡Creo que me han tomado el pelo! ¡Me doctoré en Física!"

"¿Tiene usted un doctorado en Física?", escupió Big Ed, esparciendo la mayor parte de un trago de Big Ed's Java sobre el suelo del restaurante.

"Sí, hago mudanzas y portes sólo para ganarme la vida", dijo el transportista al salir por la puerta, moviendo la cabeza con pesadumbre.

Errores de desbordamiento

Como vimos en el ejemplo anterior, el error de desbordamiento es posible cada vez que se hace una suma o resta y el resultado es demasiado grande para almacenarse en el número de bits reservado para el resultado. En la mayoría de los microordenadores, esto significaría que el resultado es mayor que el que puede almacenarse en 8 ó 16 bits.

Supongamos que operamos con un registro de 8 bits en el microprocesador. Una típica instrucción de suma sumaría los contenidos de un registro de 8 bits, llamado *acumulador*, a los contenidos de otro registro o posición de memoria. Ambos *operando*s estarían en complemento a dos con signo. ¿En qué condiciones se produciría el error de desbordamiento? Cualquier resultado mayor de +127 o menos de -128 daría lugar a error de desbordamiento. Ejemplos:

$$\begin{array}{r} 1111 \ 0000 \quad (-16) \\ + 1000 \ 1100 \quad (-116) \\ \hline 0111 \ 1100 \quad (+124 \text{ ¡no!}) \end{array} \qquad \begin{array}{r} 0111 \ 1111 \quad (+127) \\ 0100 \ 0000 \quad (+64) \\ \hline 1011 \ 1111 \quad (-80 \text{ ¡no!}) \end{array}$$

Observe que, en ambos casos, el resultado era obviamente incorrecto y el signo era el contrario al verdadero. El error de desbordamiento sólo puede ocurrir cuando la operación consiste en sumar dos números positivos, dos números negativos, restar un número negativo de otro positivo o uno positivo de otro negativo.

El resultado de la suma o resta, en la mayoría de los microprocesadores, sube (1) o baja (0) un *indicador*. Este indicador puede utilizarse en un *salto condicional*: un salto *si hay* error de desbordamiento o un salto *si no lo hay*. La comprobación se hace a nivel de lenguaje máquina y no en BASIC.

Acarreo

Otra condición importante es la de *acarreo*. Hemos visto cómo se usan los acarreos en sumas de números binarios, y cómo los *acarreos negativos* se emplean en restas. El acarreo que se produce por una suma es aquel que se produce a continuación del dígito más significativo del resultado. Un ejemplo: supongamos que sumamos los dos números siguientes:

$$\begin{array}{r} 1 \ 1111 \ 111 \quad (\text{Acarreos}) \\ 0111 \ 1111 \quad (+127) \\ 1111 \ 1111 \quad (-1) \\ \hline 0111 \ 1110 \quad (+126) \end{array}$$

Los unos encima de los operandos representan los acarreos a la siguiente posición. El acarreo de más a la izquierda, sin embargo, "cae fuera" de las posiciones ocupadas por los bits. De hecho, al no coincidir con estas posiciones, pone el indicador del acarreo a 1 en el microprocesador de que se trate. ¿Es útil este acarreo? No mucho, en este ejemplo. Habrá un acarreo siempre que el resultado no sea negativo. Cuando el resultado cambia de 0000 0000 a 1111 1111, no se produce ningún acarreo. Como este indicador puede comprobarse con un "salto condicional, si hay acarreo" o un "salto condicional, si no hay acarreo" a nivel de lenguaje máquina, el estado del acarreo es útil a veces. También se utiliza para desplazamientos, que veremos más adelante.

Otros indicadores

Los resultados de operaciones aritméticas como suma y resta establecen generalmente otros dos indicadores en el microprocesador. Uno es el indicador "cero". Se pone a (1) cuando el resultado es cero, y se pone a (0) cuando el resultado no es cero. Se pondría a (1) para una suma de +23 y -23, y se pondría a (0) para una suma de -23 y +22. Puesto que constantemente comparamos valores en programas en lenguaje máquina, el indicador cero se maneja mucho en la práctica.

El indicador "signo" también se utiliza generalmente. Este se pone a 1 o a 0 de acuerdo con el resultado de la operación; refleja el valor del bit en la posición más significativa.

El indicador de cero y el de signo pueden comprobarse en programas en lenguaje máquina por medio de saltos condicionales como "salto si es cero", "salto si no es cero", "salto si es positivo" y "salto si es negativo". Nótese que una condición de positivo incluye el caso de que el resultado sea cero. El cero es un número positivo en la notación en complemento a dos.

Indicadores en los microordenadores

Los indicadores en el microprocesador Z-80 son representativos de los de todos los microprocesadores. Se muestran en la figura 4.1. Los indicadores del microprocesador 6809 son un segundo ejemplo de indicadores de un microprocesador cualquiera, como muestra la figura 4.2.

En la sección siguiente analizaremos de forma detallada las operaciones lógicas y de desplazamiento, que también afectan a los indicadores. Antes de empezar dicho capítulo, no obstante, he aquí algunos ejercicios sobre error de desbordamiento, acarreos e indicadores.

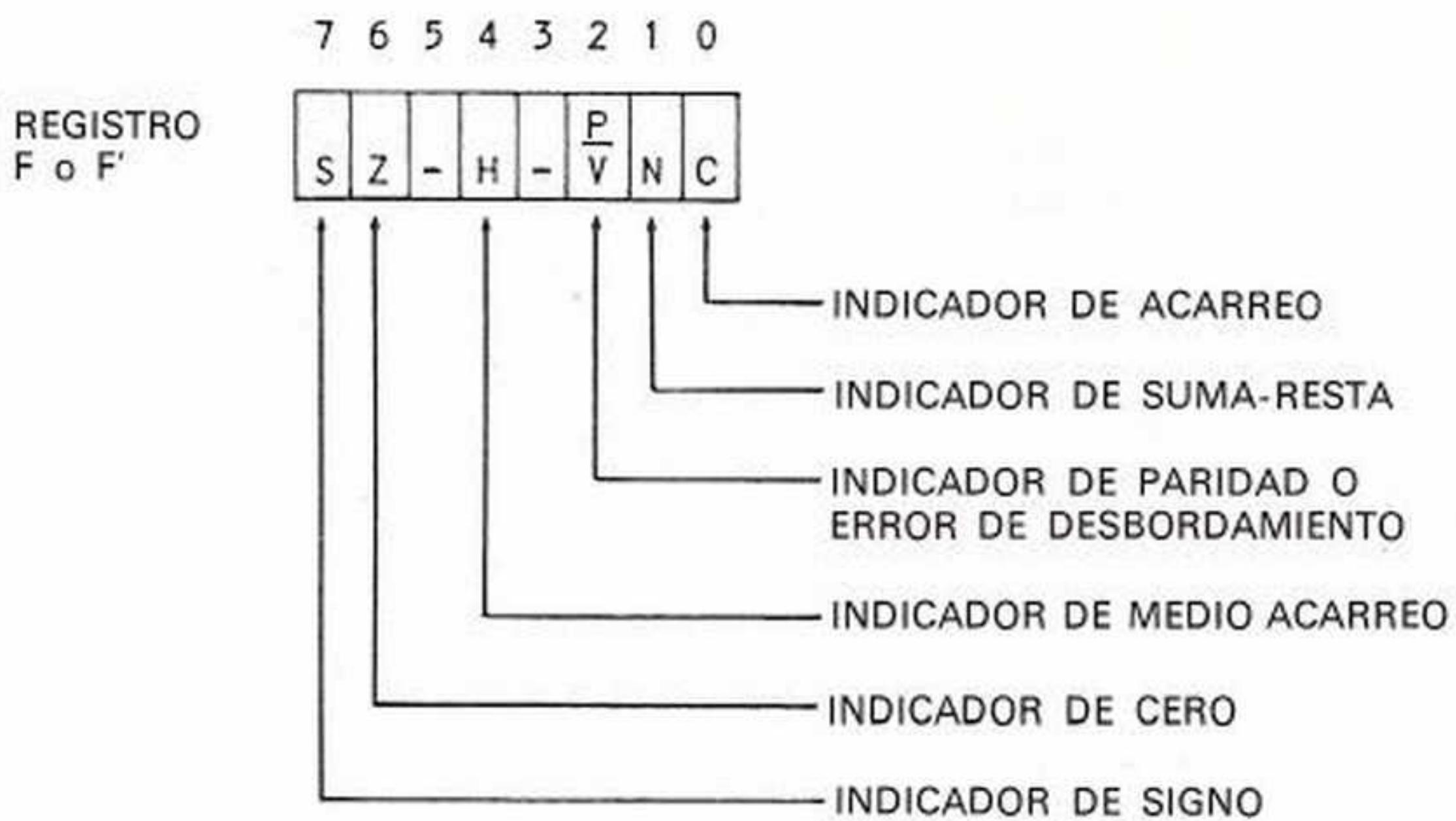


Figura 4.1. Indicadores del microprocesador Z-80



Figura 4.2. Indicadores del microprocesador 6809

Ejercicios

1. ¿Cuáles de las siguientes operaciones darán error de desbordamiento? Halle su valor.

$$\begin{array}{r} 0111111 \\ + 00000001 \\ \hline \end{array} \quad \begin{array}{r} 0111111 \\ - 00000001 \\ \hline \end{array} \quad \begin{array}{r} 10101111 \\ + 11111111 \\ \hline \end{array}$$

2. ¿Cuáles de las siguientes operaciones produce un acarreo “fuera del límite” que ponga a (1) este indicador?

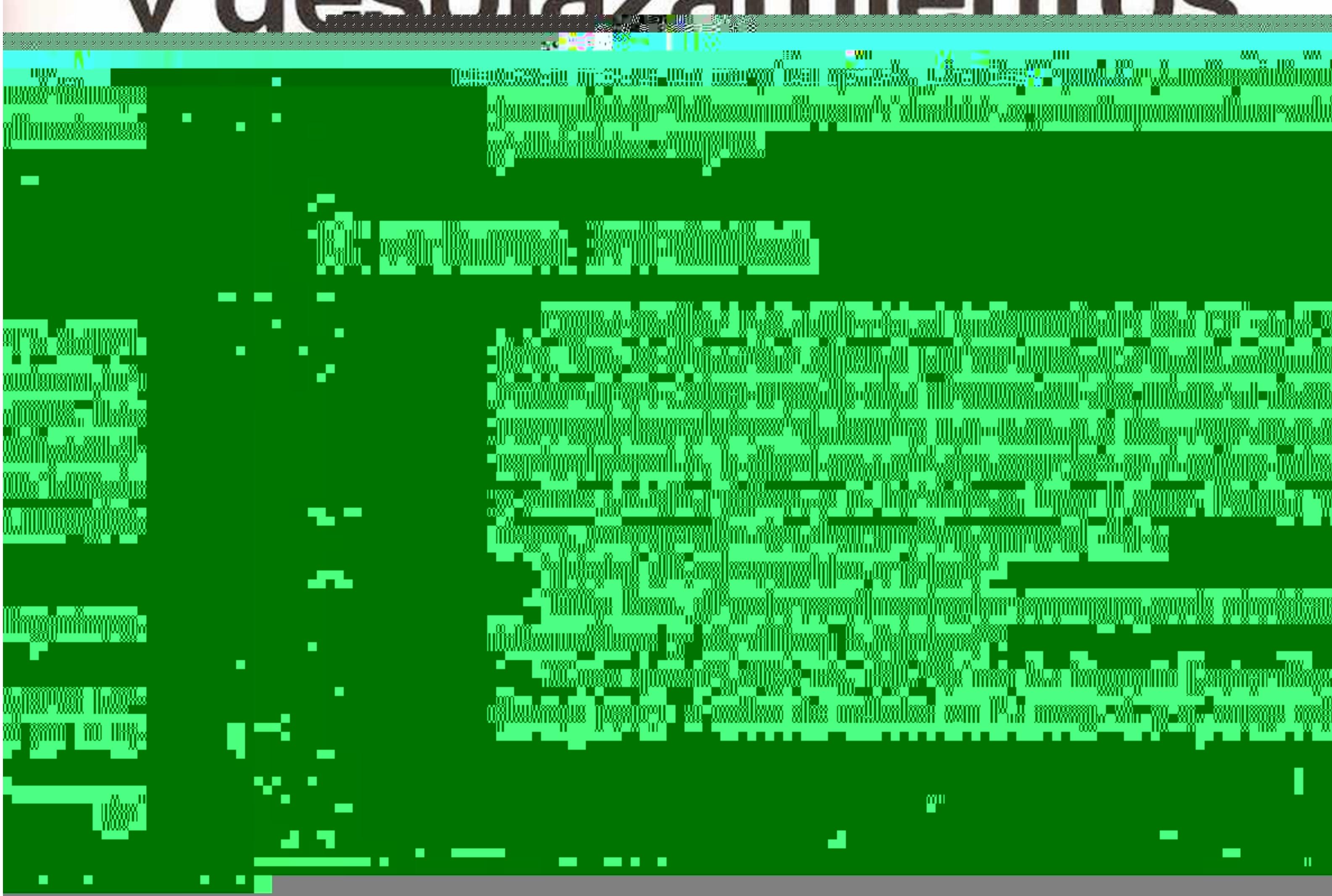
$$\begin{array}{r} 0111111 \\ + 00000001 \\ \hline \end{array} \quad \begin{array}{r} 1111111 \\ + 00000001 \\ \hline \end{array}$$

3. Supongamos que tenemos un indicador Z (cero) y otro S (signo) en el microprocesador. ¿Cuáles serán los “estados” (0,1) de los indicadores Z y S después de las operaciones siguientes?

$$\begin{array}{r} 0111111 \\ + 10000001 \\ \hline \end{array} \quad \begin{array}{r} 11011111 \\ - 10101010 \\ \hline \end{array}$$

5

Operaciones lógicas y desplazamientos



Uno de ustedes tendrá que sentarse en una silla, cerca de la mesa A. Es la silla de mi hija Carrie, pero creo que servirá. Esto acumulará..., ejem..., acomodará a todos”¹.

“Muy bien”, dijo el portavoz; y el grupo llenó las dos mesas y la silla extra (Fig. 5.1).

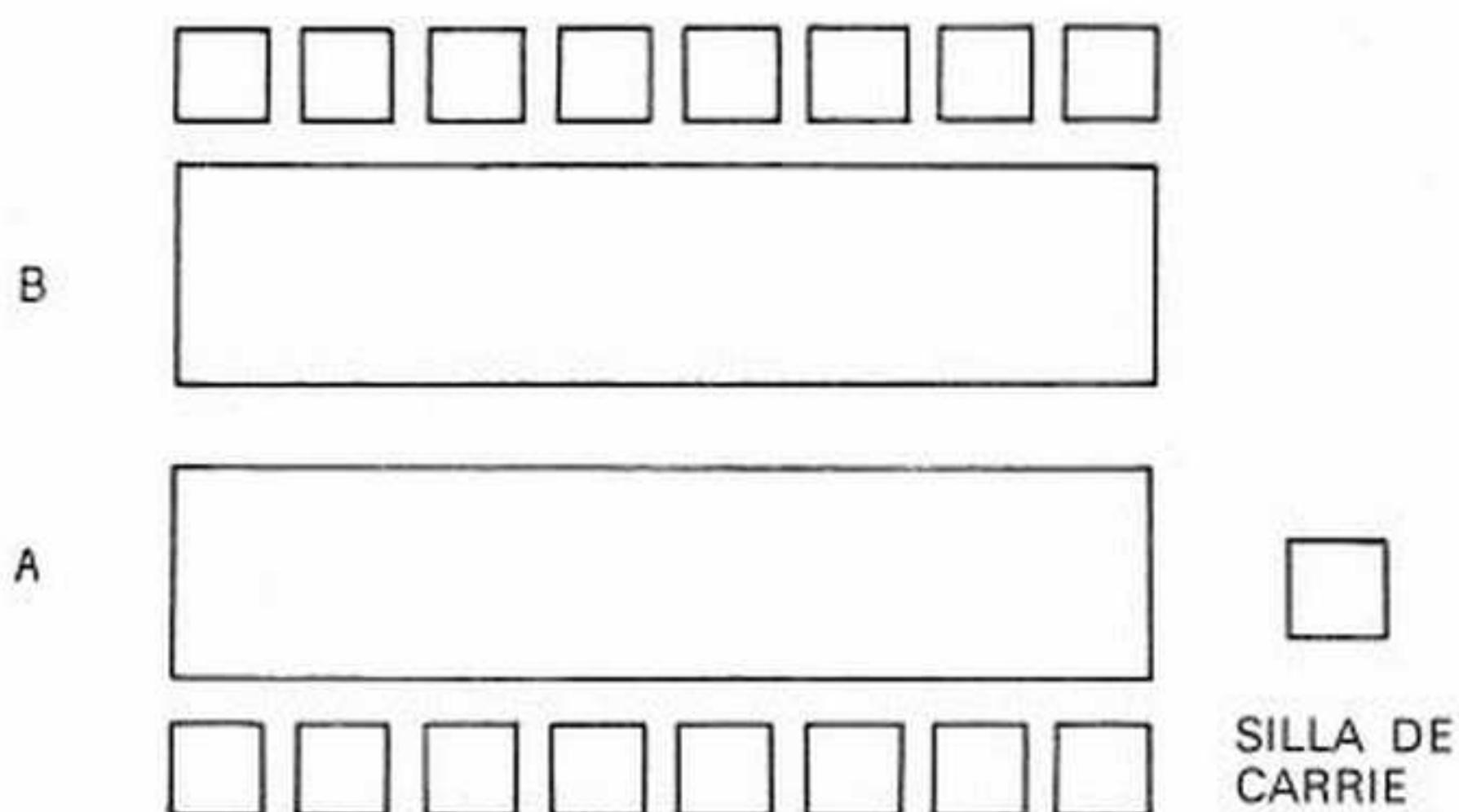


Figura 5.1. Disposición de los asientos

“Caramba, nunca pensé que hubiera tantos especialistas en informática en Gran Bretaña”, dijo Ed.

“Tantos como yanquis”, dijo el portavoz. “¿Ha oido usted hablar de Turing, el Proyecto Coloso para descifrar Enigmas, o los Tubos de Williams?”

Big Ed negó con la cabeza. “Lo siento, no tengo ni idea.”

“Está bien, yanqui. Vamos a ver: comamos primero, y luego asistamos a una conferencia especial sobre microprocesadores y sus aplicaciones al cricket. Caballeros, ¿pueden prestarme atención, por favor?

Como ustedes saben, el Ministerio nos ha concedido fondos limitados para este viaje. Por tanto, tenemos que poner ciertas restricciones al almuerzo de hoy. He echado un vistazo al menú y he llegado a las siguientes conclusiones:

Uno. Pueden ustedes tomar café (puaj) O té, pero NO ambos.

Dos. Pueden tomar sopa O ensalada O ambos.

Tres. Pueden tomar un sandwich O un plato combinado O una Sorpresa de Big Ed.

Cuatro. Si toman postre Y copa, deben pagar el suplemento adicional. ¿Alguna pregunta?”

¹ N. del T.: La nota de humor se desprende, en esta ocasión, del hecho de que *Carrie* y *carry* (acumular, acarreo) se pronuncian en inglés de manera idéntica.

Uno de los científicos de edad más avanzada habló. "Vamos a ver. Si he entendido bien, tenemos una O exclusiva del café y del té; una O inclusiva de la sopa y la ensalada; una O inclusiva del plato fuerte, y una Y del postre y de la copa con suplemento adicional. ¿Correcto?"

"Correcto, Geoffrey. ¡Bien, compañeros; al ataque!"

"Este aire acondicionado me está dejando el cuello helado", dijo uno de los jóvenes ingenieros.

"Lo que tenemos que hacer entonces es rotar alrededor de la mesa A, pasando por la silla extra, de modo que cada cual reciba por turno el aire frío", dijo el portavoz. "Cada vez que diga: '¡a rotar!', cambiaremos de silla una posición."

"¿Es realmente necesario pasar por mi silla?", preguntó el ingeniero sentado en la silla extra.

"Bueno, podemos elegir entre rotar pasando por la silla o sin pasar, pero creo que es más justo hacerlo como he dicho."

A intervalos espaciados regularmente, a lo largo de la comida, el portavoz gritaría: "¡rotad!", y el grupo de la mesa A y la silla extra se movería como muestra la figura 5.2. Finalmente, todo el grupo de la mesa A se levantó, pagó su cuenta y se fue.

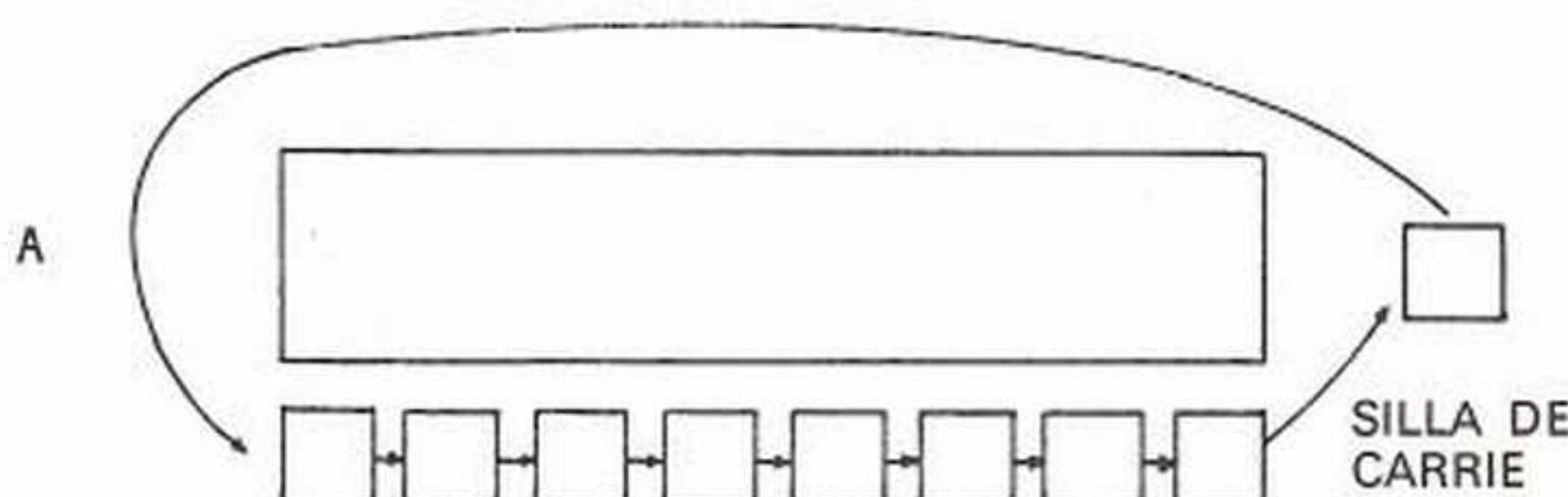


Figura 5.2. El grupo rota

"Creo que hemos acabado aquí, señor", dijo el grupo de la mesa B.

"Muy bien; entonces, desplazaos por orden hacia la izquierda por la silla vacía, para que pueda ver vuestra cuenta", dijo el portavoz.

El científico más próximo a la caja se levantó y ocupó la silla vacía. El portavoz examinó la cuenta. "¡El siguiente!", exclamó. La persona de la silla extra fue hacia la caja, y la persona a continuación ocupó su lugar en la silla extra. Este proceso continuó hasta que la mesa B se vació (véase la figura 5.3).

"Excelente comida", dijo el portavoz a Big Ed al marcharse.

"Encantado de tenerles por aquí", dijo Ed. "Espero que su seminario de cricket tenga éxito."

"Eso no nos preocupa demasiado; lo que nos preocupa son las arañas",

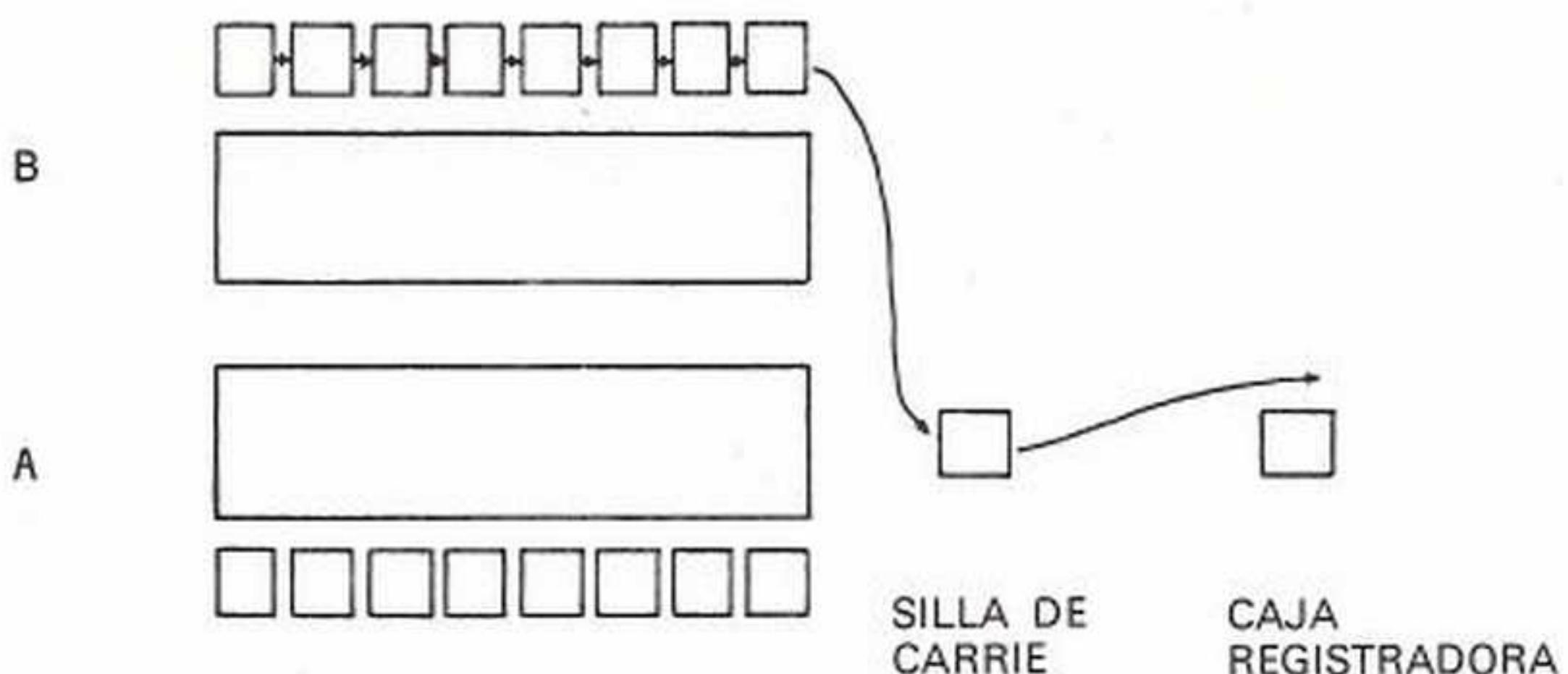


Figura 5.3. El grupo se desplaza

dijo el portavoz, abriendo el paraguas y saliendo por la puerta al soleado cielo de San José. "¡Hasta la vista!"

Operaciones lógicas*

Todos los microordenadores son capaces de realizar las operaciones lógicas de Y, O y O exclusiva a nivel de lenguaje máquina. Además, la mayoría de las versiones del BASIC permiten realizar la Y la O.

Todas las operaciones lógicas trabajan "bit a bit". No hay acarreos a otras posiciones. A nivel de lenguaje máquina, las operaciones lógicas se ejecutan en un byte de datos; el BASIC permite que las operaciones lógicas tengan lugar con operandos de dos bytes. Siempre hay dos operandos y un resultado.

Operación O

La operación O se muestra en la figura 5.4. Su *tabla de verdad* establece que habrá un 1 en el resultado si uno de los operandos, o ambos, tienen un 1.

La operación O se utiliza, a nivel de lenguaje máquina, para *poner* a 1 un bit. Una aplicación típica debería emplear los ocho bits de una posición de memoria como ocho *indicadores* para diversas condiciones, según muestra la figura 5.5. La O no es tan ampliamente utilizada en BASIC, pero

* N. del T.: Hemos decidido traducir las operaciones lógicas DND, OR, XOR y NOT por Y, O, O_{ex} y NO, respectivamente, para lograr una más fácil comprensión del texto.

| | | | | | |
|----------|----------|----------|---------|---|-----------|
| 0 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 0 | 1 | 1 | |
| <u>—</u> | <u>—</u> | <u>—</u> | | | |
| 0 | 1 | 1 | | 1 | |
| 0 | 1 | 1 | | 1 | RESULTADO |
| 0 | 00110111 | } | 8-BIT 0 | | |
| 0 | 01011011 | } | | | |
| <u>—</u> | <u>—</u> | <u>—</u> | | | |
| 01111111 | } | | | | |

Figura 5.4. Operación O

INDICADORES EN UNA POSICION DE MEMORIA

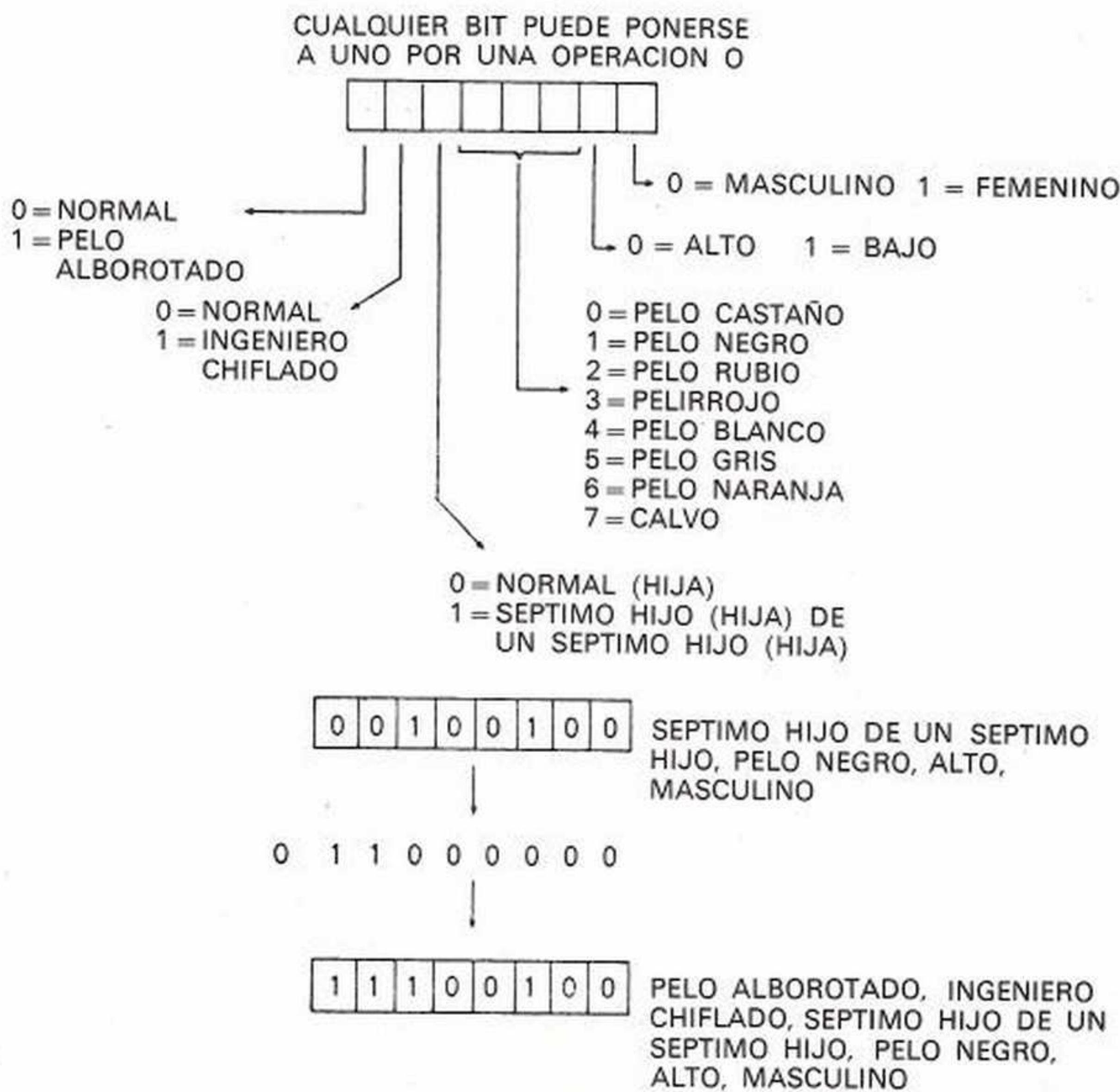


Figura 5.5. Usando la operación O

su uso se requiere ocasionalmente cuando, por ejemplo, se ponen los bits de un byte de una memoria de video para mostrar minúsculas, como ilustra la figura 5.6.



Figura 5.6. Ejemplo de O

Operación Y

La operación Y se describe gráficamente en la figura 5.7. También opera únicamente a nivel de bit, sin acarreos a otras posiciones. El resultado de Y es un 1 si ambos operandos son 1; si alguno tiene 0, el resultado es 0.

$$\begin{array}{r}
 Y \quad 0 \\
 \underline{-} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 Y \quad 0 \\
 \underline{-} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 Y \quad 1 \\
 \underline{-} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 Y \quad 1 \\
 \underline{-} \\
 1
 \end{array}
 \\
 \begin{array}{c}
 00110111 \\
 01011011 \\
 \hline
 00010011
 \end{array}
 \left. \right\} \text{8 BITS Y}$$

Figura 5.7. Operación Y

La Y se utiliza primordialmente en lenguaje máquina, para *poner* a 0 un bit o *quitar* ciertas partes de una palabra de 8 bits, como muestra la figura 5.8. En BASIC, la operación Y tiene aplicaciones más limitadas. La figura 5.9 nos ofrece un ejemplo; comprueba los múltiplos de 32 en un contador de 32 líneas por página.

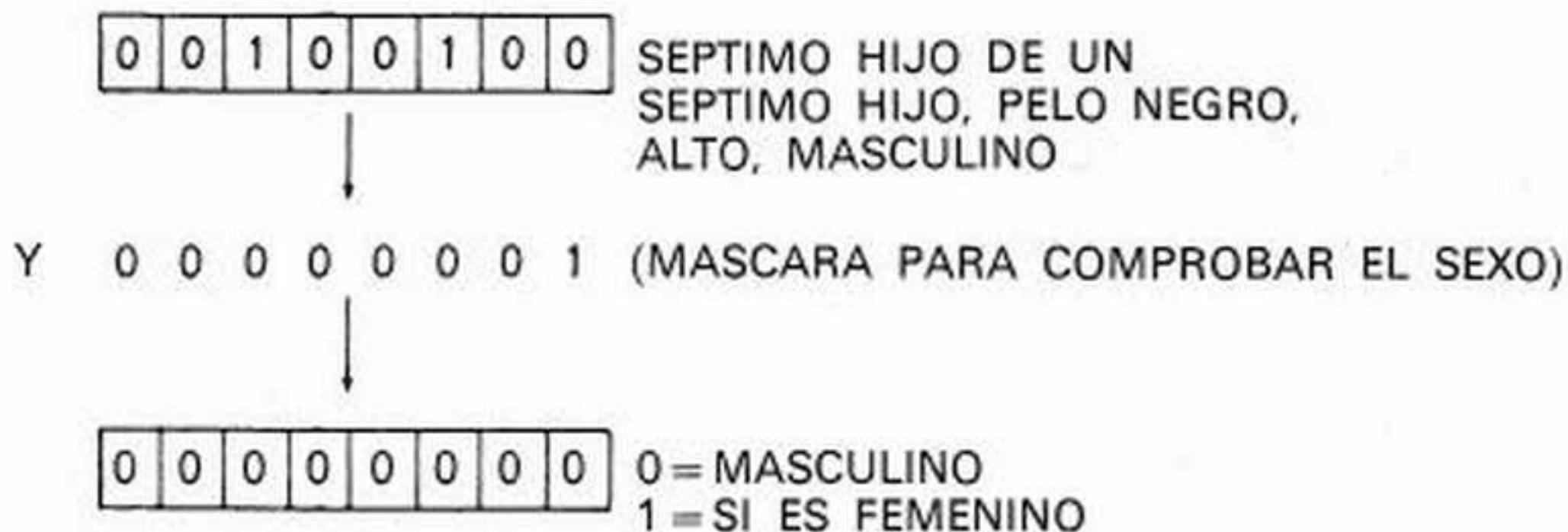


Figura 5.8. Ejemplo de Y



Figura 5.9. Otro ejemplo de operación Y

Operación O exclusiva

La figura 5.10 muestra esta operación. Sus reglas establecen que el resultado es un 1, si uno u otro bit, pero no ambos, son unos. En otras palabras, si ambos bits son unos, el resultado es 0.

| | | | | | | | | |
|----------|---|----------|---|----------------------|---|----------|---|-------------------|
| O_{ex} | 0 | O_{ex} | 0 | O_{ex} | 1 | O_{ex} | 1 | |
| | — | | — | | — | | — | |
| | 0 | | 1 | | 1 | | 0 | RESULTADO |
| | | | | | | | | |
| | | | | 00110111 01011011 | | | | BIT 8 O EXCLUSIVA |
| | | | | ————— | | | | |
| | | | | 01101100 | | | | |

Figura 5.10. Operación O exclusiva

La O exclusiva no se utiliza con frecuencia en lenguaje máquina y en BASIC. La figura 5.11 muestra un ejemplo donde el bit menos significativo

se utiliza como un conmutador (*toggle*), para indicar el número de pasadas: par o impar.

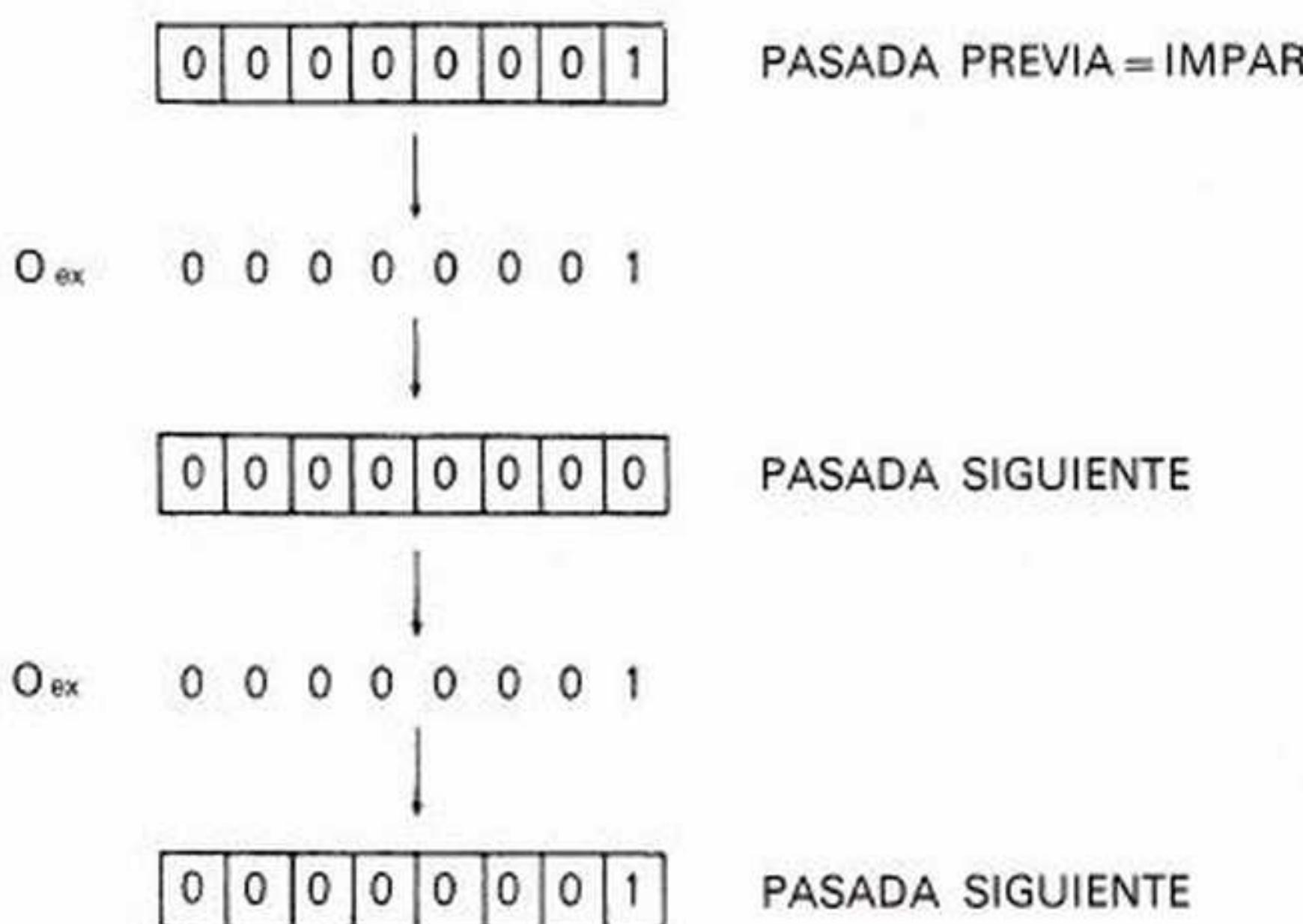


Figura 5.11. Ejemplo de O exclusiva

Otras operaciones lógicas

Hay otras operaciones lógicas que se realizan en BASIC y en lenguaje máquina. Una de éstas es la operación NO. Es similar a la operación de *invertir el signo* tratada en el capítulo 4, salvo que ésta da lugar al *complemento a uno*. El complemento a uno de un número se obtiene cambiando todos los unos por ceros y todos los ceros por unos, *sin añadir un uno*. ¿Qué efecto produce? Analicemos un ejemplo de un número con signo.

Si realizamos la operación NO sobre el número 0101 0101, obtenemos 1010 1010. El valor original era +85; el resultado es un número negativo que, una vez transformado por las reglas del complemento a dos, se convierte en $0101\ 0101 + 1 = 0101\ 0110 = -86$. Se podría decir, por tanto, que la operación NO suma uno al número y entonces invierte el signo. La versión en lenguaje máquina de la operación NO se denomina habitualmente CPL —para complemento a uno.

La operación NOT de BASIC puede emplearse para comprobar condiciones lógicas en un programa, como muestra la figura 5.12. El lenguaje máquina CPL no se utiliza con frecuencia.

```
1010 IF NOT (IMPRESORA) THEN  
    PRINT "NO IMPRESORA-COMPRE UNA-ESPERARE"  
ELSE LPRINT "RESULTADO=";A
```

Figura 5.12. Operación NO

Operaciones de desplazamiento

La reunión de los británicos en el restaurante mostraba dos tipos de desplazamiento comúnmente utilizados en microordenadores: *rotaciones* y *desplazamientos lógicos*. Son susceptibles de realizarse en lenguaje máquina, pero no en BASIC, y generalmente operan en ocho bits de datos. Están relacionados con el indicador de acarreo tratado en el último capítulo.

Rotaciones

La figura 5.2 mostraba una rotación con la mesa A en el restaurante. Observemos ahora la figura 5.13, donde los datos se rotan a derecha o izquierda, una posición cada vez. Aunque los ordenadores más complejos permitirán cualquier número de desplazamientos con una instrucción, los más simples sólo permiten un desplazamiento por cada instrucción.

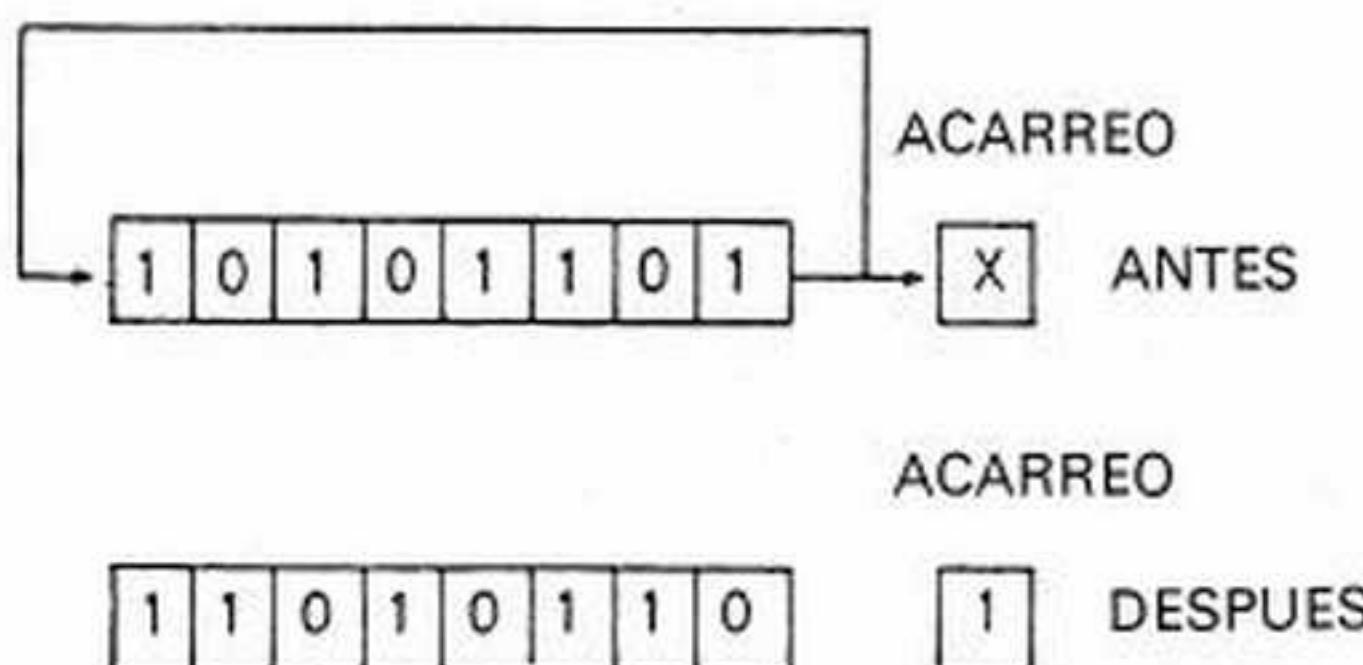
Como los datos rotan fuera de los límites del registro del microprocesador o posición de memoria, o bien vuelve al límite opuesto del registro o posición de memoria, o bien va al indicador de acarreo. Si los datos pasan a través del acarreo, se trata en realidad de una *rotación de 9 bits*. Si los datos 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 5.13.

El indicador de acarreo puede comprobarse por una instrucción de salto condicional a nivel de lenguaje máquina para comprobar efectivamente si el bit desplazado era un cero o un uno. La rotación se utiliza para comprobar un bit de una vez para operaciones como multiplicación (véase capítulo siguiente) o el alineamiento de datos en una operación Y.

Desplazamientos lógicos

El segundo tipo de desplazamiento reflejado en la anécdota del restaurante era el *lógico*. Este no es una rotación. Los datos caen fuera del límite, de la misma forma que los científicos se fueron del restaurante.

ROTACION HACIA LA DERECHA



ROTACION HACIA LA IZQUIERDA A TRAVES DEL ACARREO

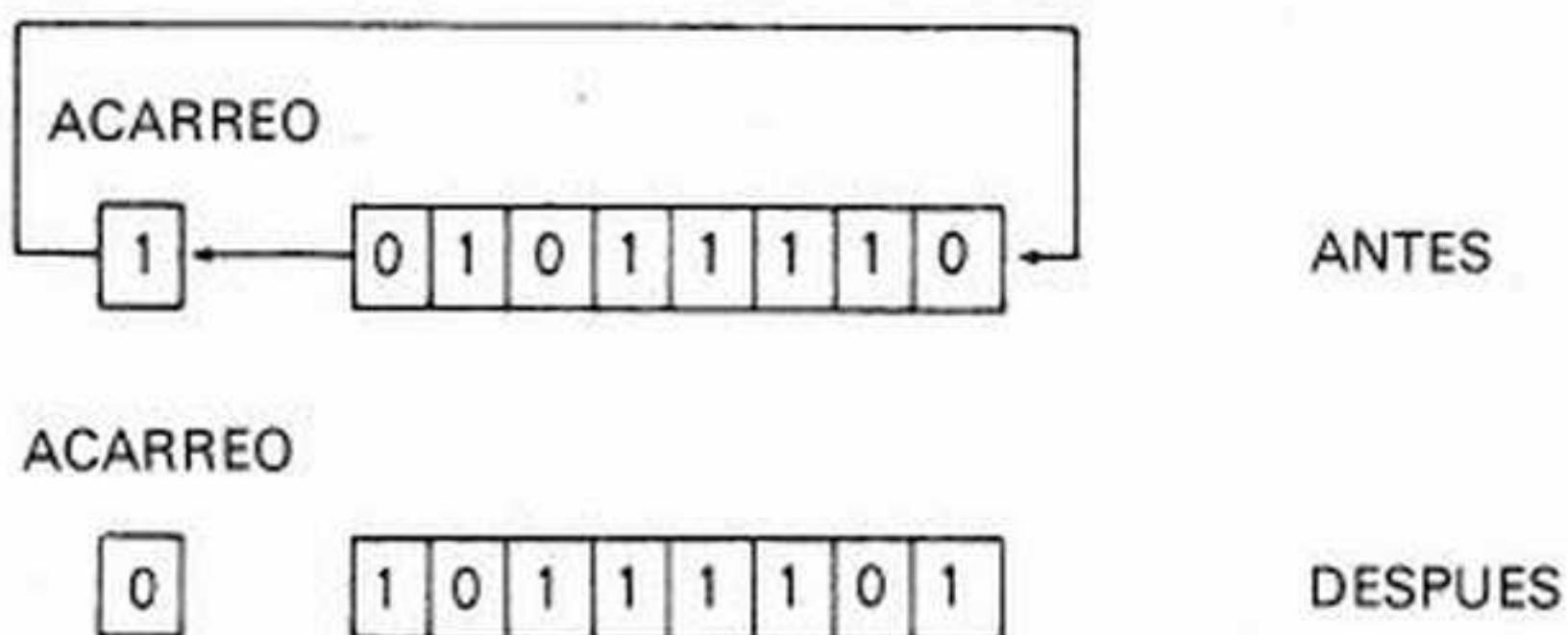


Figura 5.13. Dos rotaciones

Cuando cada bit es desplazado, sin embargo, va al acarreo de forma que éste siempre almacena el resultado del desplazamiento. El límite opuesto del registro o posición de memoria se rellena con ceros a medida que se desplaza cada bit. Aquí, de nuevo, se desplaza una posición cada vez. La operación de desplazamiento lógico se muestra en la figura 5.14.

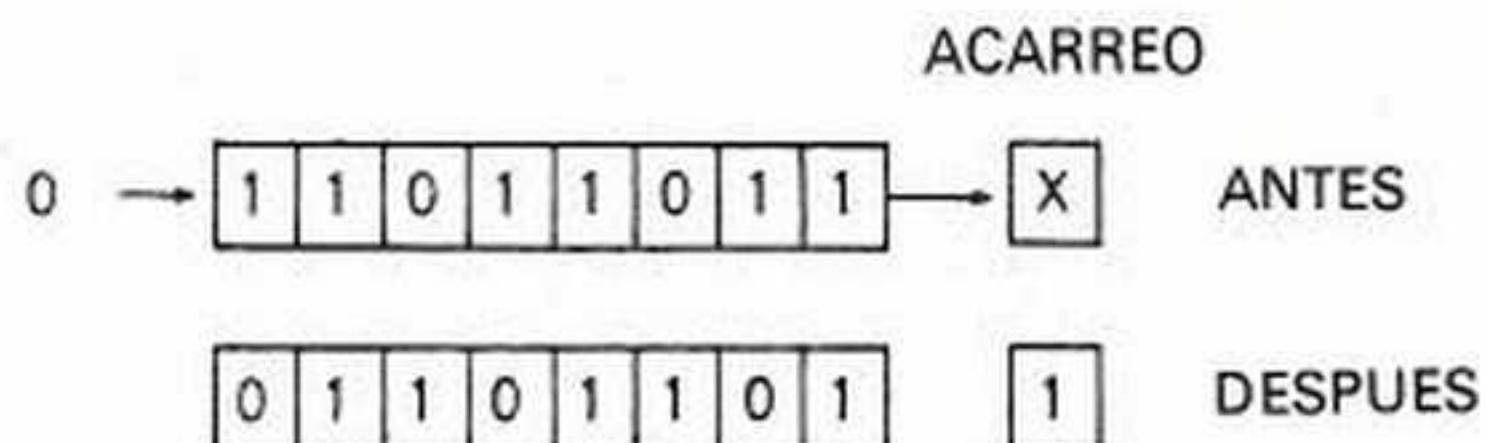
En realidad no podemos hablar mucho sobre la rotación, relacionada con lo que sucede aritméticamente con el contenido. Esto se debe a que los datos vuelven a entrar en el registro y, en sentido aritmético, los resultados no se pueden predecir.

En el caso del desplazamiento lógico hacia la derecha o hacia la izquierda, sin embargo, los resultados son plenamente previsibles. Observemos algunos ejemplos. Supongamos que tenemos el valor 0111 1111, con el acarreo conteniendo un valor cualquiera. Mostraremos el registro y el acarreo con nueve bits con el acarreo a la derecha; es decir, 0111 1111 x. Después de una operación lógica de desplazamiento a la derecha tenemos 0011 1111 1. El valor original era +127; una vez producido el desplazamiento, el valor es +63 más el acarreo. ¡Parece como si un desplazamiento lógico a la derecha dividiera por dos y pusiera el resto de 0 ó 1 en el acarreo! Esto es cierto,

y la operación lógica de desplazamiento a la derecha puede utilizarse cada vez que se quiera dividir por 2, 4, 8, 16 u otra potencia de dos.

¿Qué sucede con el desplazamiento a la izquierda? ¡Efectivamente! Una operación lógica de desplazamiento a la izquierda multiplica por dos. Por ejemplo, tomemos $x\ 0001\ 1111$, donde x es el estado del indicador del acarreo. Después del desplazamiento lógico a la izquierda, el resultado es $0\ 0011\ 1110$. El número original era $+31$, y el resultado es $+62$ con el acarreo puesto a 0 por el bit más significativo. El desplazamiento lógico a la izquierda se puede realizar cada vez que un número ha de multiplicarse por 2, 4, 8 o cualquier otra potencia de dos.

DESPLAZAMIENTO LOGICO HACIA LA DERECHA



DESPLAZAMIENTO LOGICO HACIA LA IZQUIERDA

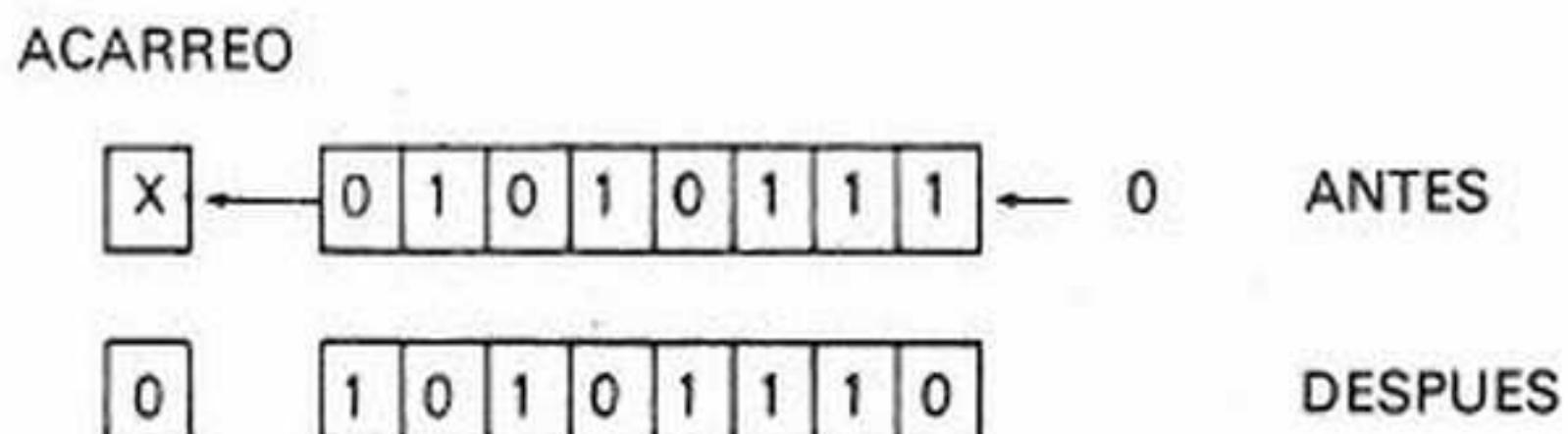


Figura 5.14. Dos desplazamientos lógicos.

Desplazamientos aritméticos

Cuando se realiza un desplazamiento lógico con números con signo, surge un problema. Consideremos el caso del número 1100 1111; es un valor de -49 . Cuando el número se desplaza hacia la derecha, el resultado es 0110 0111, representando el valor $+103$. Obviamente, el desplazamiento no dividió el número -49 por 2 para obtener un resultado de -24 .

Para resolver el problema de desplazar datos aritméticos, se suele incluir un desplazamiento aritmético en los ordenadores.

El desplazamiento aritmético *conserva* el signo según se desplaza a la derecha, de forma que el desplazamiento es (casi) aritméticamente correcto.

Si se realizase un desplazamiento aritmético en el ejemplo anterior, el resultado sería el que muestra la figura 5.15.



Figura 5.15. Desplazamiento aritmético a la derecha

¿Qué sucede con los desplazamientos a la izquierda? En algunos microordenadores, el desplazamiento aritmético a la izquierda mantiene el bit de signo y desplaza el siguiente bit más significativo al acarreo, como muestra la figura 5.16. En otros microordenadores no hay auténticos desplazamientos a la izquierda.

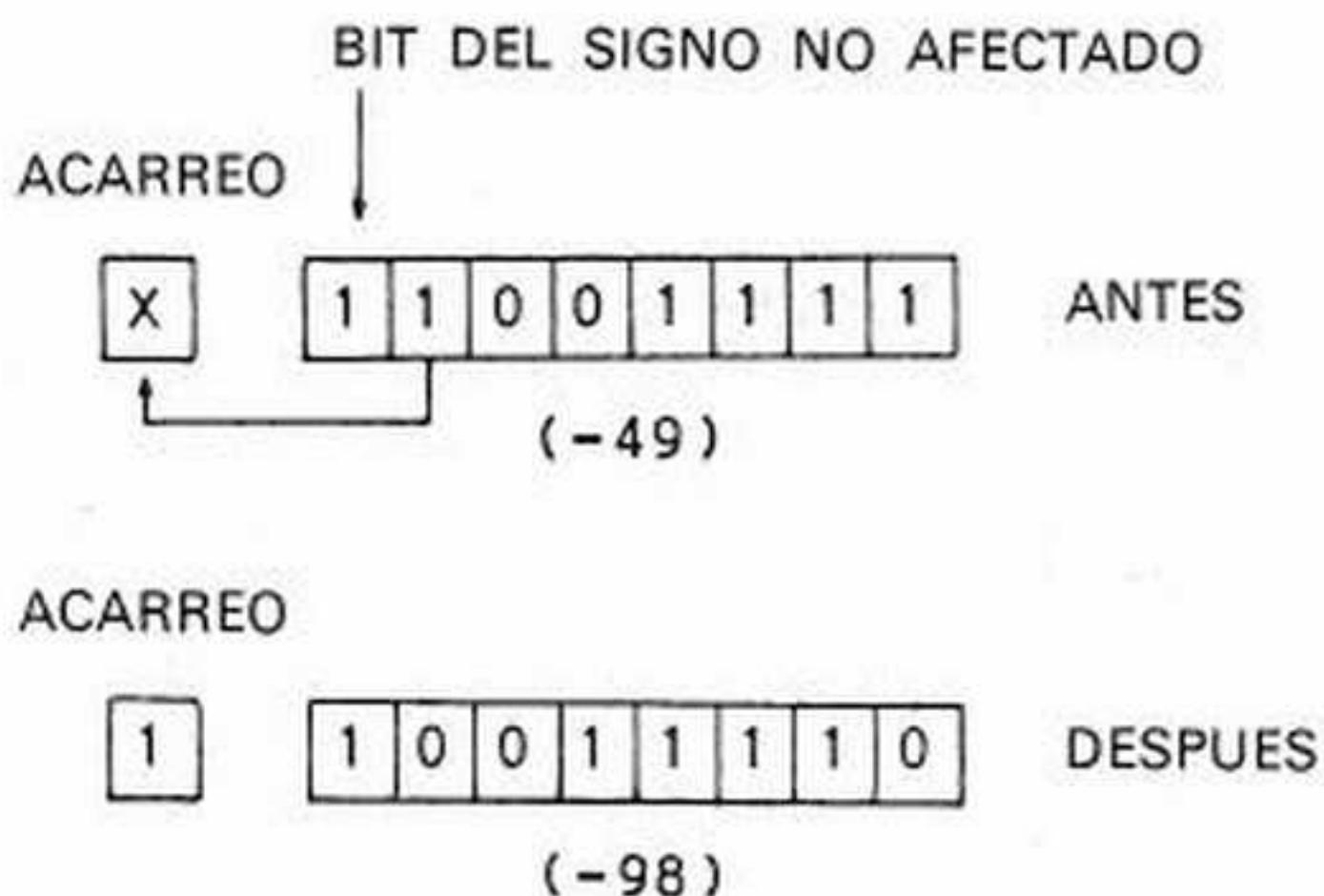


Figura 5.16. Desplazamiento aritmético a la derecha

En el próximo capítulo veremos cómo el desplazamiento puede utilizarse para llevar a cabo muchos tipos diferentes de *algoritmos* de multiplicación y división. Mientras tanto, intente contestar a las siguientes preguntas.

Ejercicios

1. Efectúe la operación O en los siguientes conjuntos de operandos binarios de 8 bits.

$$\begin{array}{r} 10101010 \\ O \underline{00001111} \end{array} \quad \begin{array}{r} 10110111 \\ O \underline{01100000} \end{array}$$

2. Realice la operación O exclusiva en los siguientes operandos binarios de 8 bits.

$$\begin{array}{r} 10101010 \\ O_{ex} \underline{00001111} \end{array} \quad \begin{array}{r} 10110111 \\ O_{ex} \underline{01100000} \end{array}$$

3. Los bits 3 y 4 de una operación de memoria tienen el código siguiente:
00 = PELO CASTAÑO, 01 = PELO NEGRO, 10 = PELO RUBIO, 11 = CALVO. Utilizando la operación Y, muestre cómo estos bits pueden ordenarse en un único resultado de 8 bits. La posición es XXXYYXXX, donde X = bit desconocido, e Y = bit de código.

4. Invierta el signo de los siguientes operandos (con signo). Escriba sus equivalentes en decimal.

00011111, 0101, 10101010

5. Realice una rotación a la izquierda sobre estos operandos:

00101111, 10000000

6. Efectúe una rotación a la derecha sobre los siguientes operandos:

00101111, 10000000

7. Efectúe una rotación a la derecha con acarreo sobre estos operandos y acarreos:

C = 1 00101111, C = 0 10000000

8. Efectúe una rotación a la izquierda con acarreo sobre estos operandos y acarreos:

C = 0 00101111, C = 1 10000000

9. Efectúe un desplazamiento lógico a la derecha de los siguientes operandos.

Escriba el acarreo después del desplazamiento, y el valor decimal de los operandos antes y después de los desplazamientos:

01111111, 01011010, 10000101, 10000000

10. Efectúe un desplazamiento lógico a la izquierda de los siguientes operandos. Escriba el acarreo después del desplazamiento, y el valor decimal de los operandos antes y después de los desplazamientos:

01111111, 01011010, 10000101, 10000000

11. Efectúe un desplazamiento lógico hacia la derecha sobre los siguientes operandos, y escriba los valores decimales antes y después del desplazamiento:

01111111, 10000101, 10000000

6

Multiplicación y división

La mayoría de los microordenadores actuales no incluyen instrucciones de multiplicación y división. En consecuencia, estas operaciones han de hacerse en rutinas de *software*; al menos, en lenguaje máquina. En este capítulo analizaremos algunas de las formas en que la multiplicación y división pueden realizarse en *software*.

Zelda aprende cómo desplazar por sí misma

“¡Hola, Don! ¿Qué tal la comida?”, preguntó Zelda, la camarera, a un ingeniero de Inlog que acababa de llegar a la caja.

“Bien, bastante bien. Bueno, la carne estaba un poco correosa...”

“Sucede siempre que no es fresca”, dijo Zelda cogiendo la cuenta. “Veamos; una taza de café... un sandwich de carne... y doce postres bajos en calorías...” Cada vez que Zelda leía una partida de la cuenta, realizaba algún tipo de operación fuera del alcance de la vista en la caja registradora. En la última partida, doce postres bajos en calorías, tardó mucho tiempo.

“Zelda, ¿qué estás haciendo?”, preguntó el ingeniero.

“Verás, Don; Big Ed quiere que nos acostumbremos a trabajar en binario, ya que este restaurante está en el centro de las industrias de microor-

denadores y todo eso. Quiere que hagamos todos los cálculos en binario para que practiquemos. No me importa cuando se trata de sumar o restar, pero la multiplicación me vuelve loca."

"¿Qué método empleas, Zelda? Quizá pueda ayudarte."

"Cada vez que multiplico, hago una suma sucesiva. Como en este caso, en que tenías doce postres. Cada uno cuesta 65 centavos, así que sumo 1100, que es doce en binario, 65 veces."

"Sí, eso realmente es correcto; de acuerdo", dijo Don, tratando de no reírse. "Ese método de suma sucesiva es válido, pero lleva mucho tiempo. Permíteme que te enseñe un método más rápido; se llama desplazamiento y suma."

Rápidamente hizo sitio en el mantel de una mesa cercana, amontonando un poco los cubiertos y demás objetos. Sacó un portaminas. "Es una pena que no tengamos papel cuadriculado, pero los cuadros del mantel servirán.

Tomemos estos 65 centavos por postre para un ejemplo de doce unidades. Antes de nada, dibujaremos dos registros. El registro del *producto parcial* tiene una amplitud de 16 bits; así (véase la figura 6.1). El registro de los multiplicandos tiene una amplitud de 8 bits. Entonces, en los ocho bits de arriba del registro del producto parcial pondremos los doce, el *multiplicador*. Lo hemos *rellenado* a ceros hasta ocho bits para obtener 0000 1100. El resto del registro será una serie de ceros. Seguidamente, pondremos el multiplicando en el registro del multiplicando.

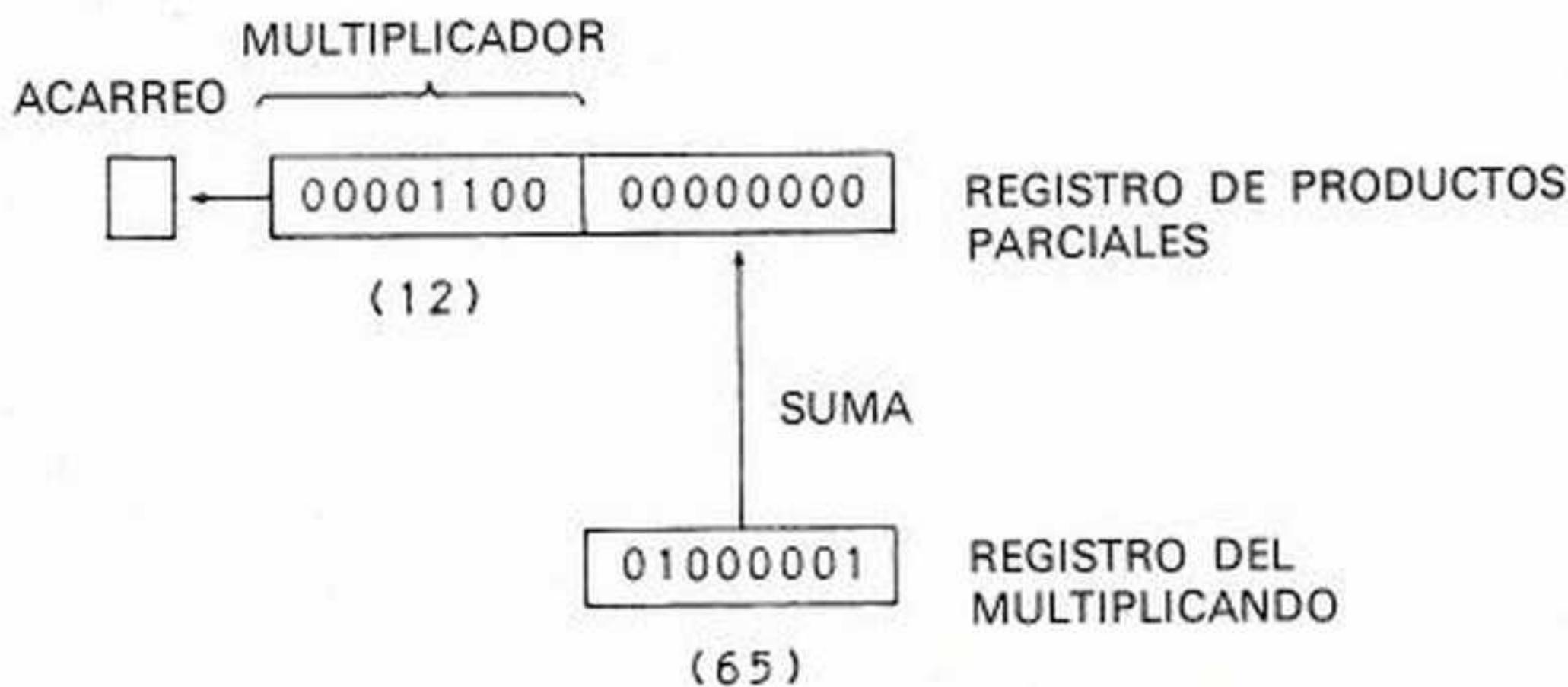


Figura 6.1. Multiplicación por desplazamiento y suma

Ya está todo preparado para hacer la multiplicación. Daremos ocho pasos para ello. Los ingenieros informáticos los denominamos *iteraciones*. Para cada iteración, desplazaremos el multiplicador una posición hacia la izquierda; el bit desplazado fuera irá al acarreo. Si el bit del acarreo es un 1,

sumaremos el multiplicando al registro del producto parcial. Si el bit del acarreo es un cero, no haremos la suma. Al final de las ocho iteraciones, estará hecho."

"Pero, ¿no se *destruirá el contenido* del multiplicador de ocho bits de arriba al sumar el registro del producto parcial?", preguntó Zelda, obviamente orgullosa de haber aprendido un poco de la jerga de los ordenadores.

"No, no se destruirá. Recuerda que los datos se desplazan a fin de hacer sitio para la posible expansión del producto parcial. Después de ocho iteraciones se habrá desplazado totalmente, y el registro del producto parcial será el producto final de la multiplicación. Mira, he dibujado todas las iteraciones para este caso" (véase la figura 6.2).

"¡Oh, sí! Muchas gracias, Don. Seguro que lo utilizaré. Ahora, déjame el resto de tu cuenta. El total es 15.63 más los impuestos. Eso es 1563 centavos dividido por cien veces seis. Veamos: 011000011011 menos 01100100 da 010110110111; esto es una vez. 010110110111 menos 01100100 da 010101010011; esto es dos veces. Restando 01100100 a..."

Algoritmos de multiplicación

Sumas sucesivas

Zelda utilizaba un método de multiplicación sencillo, llamado *suma sucesiva* (Fig. 6.3). En él, el multiplicando (número que ha de ser multiplicado) es multiplicado por el multiplicador. El proceso se realiza haciendo cero un resultado llamado producto parcial y sumando el multiplicando al producto parcial por el número de veces indicado por el multiplicador. El ejemplo anterior era 65 veces 12, que Zelda resolvió sumando 12 al producto parcial 65 veces.

Aunque este método es sencillo, es *muy largo* en la mayoría de los casos. Supongamos que trabajamos con una multiplicación "ocho por ocho". Una multiplicación de 8 por 8 bits produce un resultado de 16 bits como máximo. El multiplicador promedio es 127, si se hace una *multiplicación sin signo*. Esto quiere decir que, en la mayoría de los casos, tendrían que hacerse 127 sumas separadas del producto parcial.

Esto difiere de la técnica de Don de *desplazamiento y suma* de ocho iteraciones. En el peor de los casos, la suma sucesiva supondrá 255 sumas; en el mejor de los casos, una. Es mejor dejar la multiplicación por suma sucesiva para aquellos casos en que el multiplicador se fija en algún valor bajo y constante; por debajo de 15 o así, o para cuando es necesario hacer multiplicaciones poco frecuentes.

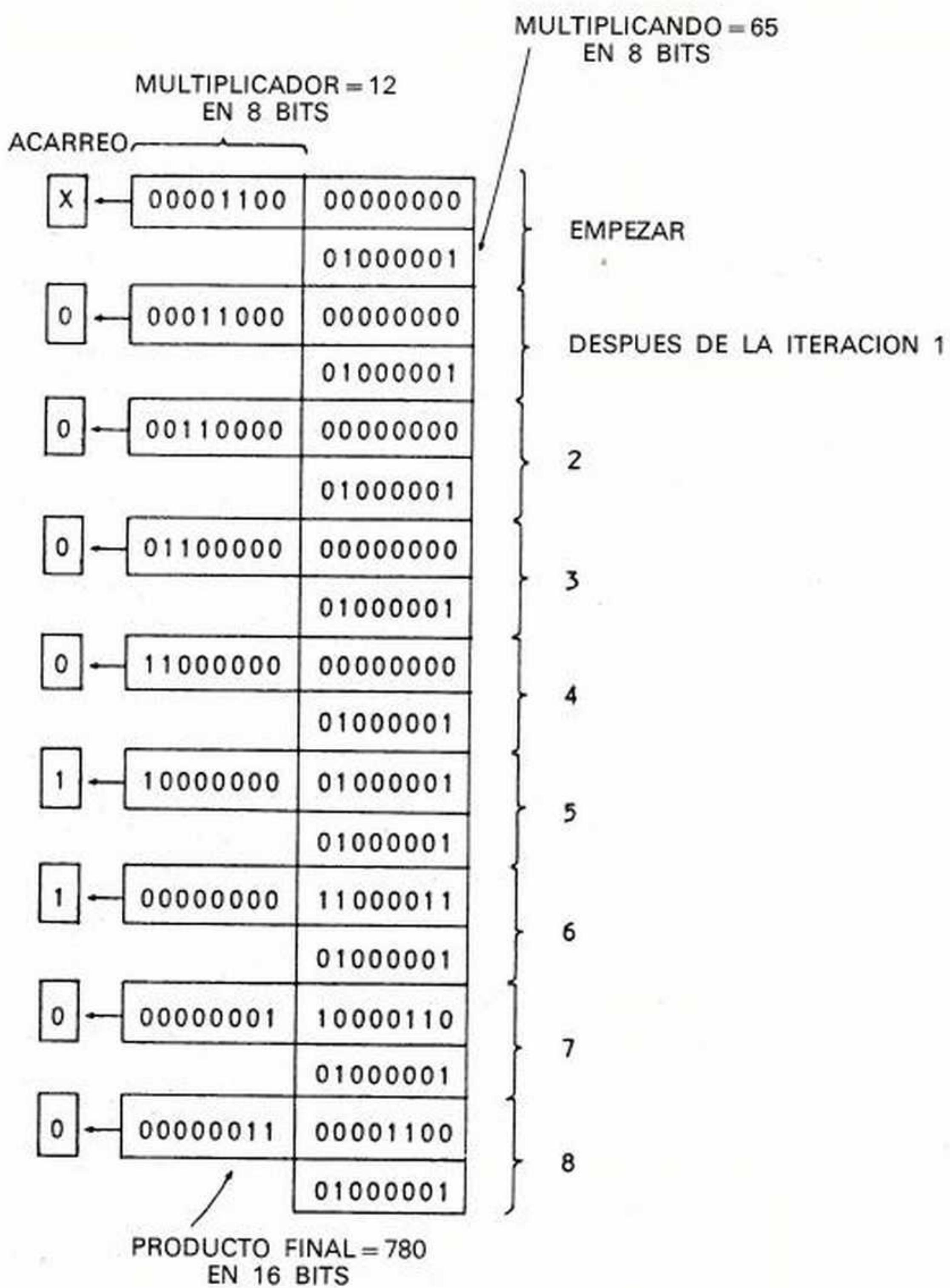


Figura 6.2. Ejemplo de multiplicación por desplazamiento y suma

| 16 BITS | | | |
|----------|----------|------|------------------------------|
| 00000000 | 00000000 | 0 | |
| 00000000 | 00001100 | +12 | ITERACION 1 |
| 00000000 | 00001100 | +12 | RESULTADO DE LA ITERACION 1 |
| 00000000 | 00001100 | +12 | ITERACION 2 |
| 00000000 | 00011000 | +24 | RESULTADO DE LA ITERACION 2 |
| | | | |
| 00000010 | 11110100 | +756 | RESULTADO DE LA ITERACION 63 |
| 00000000 | 00001100 | +12 | ITERACION 64 |
| 00000011 | 00000000 | +768 | RESULTADO DE LA ITERACION 64 |
| 00000000 | 00001100 | +12 | ITERACION 65 |
| 00000011 | 00001100 | +780 | RESULTADO DE LA ITERACION 65 |

Figura 6.3. Ejemplo de multiplicación por sumas sucesivas

Suma sucesiva de potencias de dos

El método de la suma sucesiva de potencias de dos consiste en una multiplicación que se realiza a menudo cuando el multiplicador es un valor fijo. Supongamos que tenemos que multiplicar siempre una suma por diez. Diez puede dividirse en un número de sumas; una combinación de éstas es $8 + 2$.

Como vimos en el capítulo anterior, el desplazamiento lógico hacia la izquierda multiplica un valor por dos. Para multiplicar por diez se puede hacer una combinación de desplazamientos y sumas para producir el efecto de la multiplicación. La operación se desarrolla así:

1. Desplazar el multiplicando por dos para obtener dos veces X.
2. Guardar este valor como "DOSX".
3. Desplazar el multiplicando por dos para obtener cuatro veces X.
4. Desplazar el multiplicando por dos para obtener ocho veces X.
5. Sumar "DOSX" al multiplicando para obtener diez veces X.

Este procedimiento se muestra en la figura 6.4 para un multiplicando de diez. Puede utilizarse cuando el multiplicador es fijo. Multiplicar por 35, por ejemplo, puede hacerse desplazando cinco veces el multiplicando para obtener $32X$, y sumando $2X + 1X = 32 + 2 + 1 = 35$.

MULTIPLICACION DE 11 POR 10

00001011 MULTIPLICANDO = 11

00001010 MULTIPLICADOR = 10

DESPLAZAR MULTIPLICANDO (DOSX)

00010110 MULTIPLICANDO = $11 \times 2 = 22$. GUARDAR

DESPLAZAR MULTIPLICANDO (CUATROX)

00101100 MULTIPLICANDO = $22 \times 2 = 44$

DESPLAZAR MULTIPLICANDO (OCHOX)

SUMAR 01011000 MULTIPLICANDO = $44 \times 2 = 88$

→ 00010110 RESULTADO PREVIO

01101110 PRODUCTO = 110

Figura 6.4. Ejemplo de suma sucesiva utilizando potencias de dos

Multiplicación por desplazamiento y suma

La multiplicación por desplazamiento y suma que Don enseñó a Zelda es el método más comúnmente utilizado en ordenadores cuyo *hardware* no está provisto de capacidad de multiplicación. Se asemeja a una técnica de papel y lápiz que sigue estrechamente el método de multiplicación decimal normal que se aplicaría en su lugar. Por ejemplo, la figura 6.5 muestra una multiplicación binaria de papel y lápiz que se parece a una multiplicación decimal. La única diferencia real entre el papel y lápiz y el método binario de desplazamiento y suma está en el desplazamiento.

Con papel y lápiz, el multiplicando se desplaza a la izquierda y después se suma. Con desplazamiento y suma, el multiplicando es estacionario, y se desplaza el producto parcial como muestra la figura 6.2.

Esta técnica de desplazamiento y suma puede utilizarse para cualquier tamaño de multiplicando o multiplicador. La regla para el tamaño del pro-

| | |
|-------------|--------------------|
| 01000001 | MULTIPLICANDO (65) |
| 00001100 | MULTIPLICADOR (12) |
| 0100000100 | |
| 01000001 | |
| 01100001100 | PRODUCTO (780) |

Figura 6.5. Multiplicación binaria en papel y lápiz

ducto es ésta: *El producto de una multiplicación binaria nunca puede ser mayor que el número total de bits del multiplicador y del multiplicando.* Dicho de otra forma, si el multiplicador y el multiplicando tienen ocho bits cada uno, el producto puede tener hasta dieciséis bits; si el multiplicador tiene doce bits y el multiplicando ocho, el producto puede tener hasta dieciséis bits, etcétera.

Otro dato interesante acerca de la multiplicación por desplazamiento y suma es que el desplazamiento puede ser a la inversa. Podemos operar con multiplicadores, empezando por el lado menos significativo. En este caso, podemos parar el proceso cuando el multiplicador es cero, lo que significa que sólo tenemos que realizar tantas iteraciones como bits significativos haya en el multiplicador. Esto reduce el tiempo medio de la multiplicación hasta la mitad del valor máximo del multiplicador. La figura 6.6 muestra este eficaz método.

Multiplicación con signo y sin signo

En todos los ejemplos anteriores hemos considerado los multiplicadores y multiplicandos *sin signo*. Los productos obtenidos en estos casos son números en valor absoluto, sin un dígito para el signo. Por ejemplo; al multiplicar 1111 1111 (255) por 1011 1011 (187) se obtiene un producto de 1011 1010 0100 0101 (47,685), donde el bit más significativo es 2^{15} y no un bit de signo.

¿Qué sucede con la multiplicación *con signo* donde el multiplicador y el multiplicando son números en complemento a dos con signo? En este caso, los números se convierten en su *valor absoluto*; se realiza el producto y al resultado se le coloca después el signo correspondiente. Por supuesto, más por más es más; más por menos es menos; menos por más es menos, y menos por menos es más, como en aritmética decimal.

He aquí un buen ejemplo de uno de nuestros operadores lógicos, la



Figura 6.6. Ejemplo del método “multiplicar hasta multiplicador cero”

O exclusiva. Si se toma una O exclusiva del multiplicando y del multiplicador, el bit más significativo del resultado (signo) será el signo del producto. Por ejemplo,

$$\begin{array}{r}
 1011 \quad 1010 \quad (-70) \\
 0000 \quad 1010 \quad (+10) \\
 \hline
 1011 \quad 0000 \quad (O_{ex})
 \end{array}$$

se obtiene una O exclusiva cuyo bit más significativo es un 1; por tanto, el producto será negativo.

El algoritmo para una operación con signo es éste:

1. Hacer la O exclusiva del multiplicador y del multiplicando. Guardar el resultado.
2. Hallar el valor absoluto del multiplicando, cambiando el signo de negativo a positivo si fuere preciso.
3. Hallar el valor absoluto del multiplicador cambiando el signo negativo a positivo si fuera preciso.

4. Multiplicar los dos números por el método estándar de desplazamiento y suma.
5. Si el resultado de aplicar el operador O exclusivo tiene un 1 en el bit de signo, cambiar el producto a un número negativo por el método del complemento a dos (operación de inversión del signo).

Hay un pequeño problema en el método anterior. El error de desbordamiento no era posible en el método sin signo, pero es posible en el método con signo tan sólo en un caso. Cuando tanto el multiplicador como el multiplicando son valores negativos máximos, el producto tendrá un error de desbordamiento. Por ejemplo, si en una multiplicación de "ocho por ocho" tanto el multiplicador como el multiplicando son 1000 0000 (-256), el producto será +65,536, que es demasiado grande para ser almacenado en diecisésis bits. Esta condición puede ser comprobada antes de que la multiplicación tenga lugar.

Algoritmos de división

Los algoritmos de división en *software* son algo más difíciles de implementar que los de multiplicación. Uno de los métodos de división es la resta sucesiva, el que Zelda utilizaba cuando la dejamos. El segundo es la "división con recuperación" bit a bit.

Resta sucesiva

La resta sucesiva se muestra en la figura 6.7. En este método el *cociente* se rellena a ceros inicialmente. El *divisor* se resta del *dividendo* sucesivamente hasta que éste se vuelve negativo. Cada vez que se resta el divisor, la cuenta del cociente se incrementa con uno. Cuando el dividendo se vuelve negativo, el divisor se *suma* una vez al *residuo* para recuperar el resto. (El residuo es la cantidad restante del dividendo.) La cuenta es el cociente de la división, mientras el resto está en el residuo, como muestra la figura 6.7.

Como en el caso de la multiplicación por suma sucesiva, la resta sucesiva es muy lenta en la mayoría de los casos. Si operamos con un dividendo de 16 bits y un divisor de 8, el cociente medio es 255. Un total de 255 restas es tan intolerable como en el caso de la multiplicación. La resta sucesiva para una división es buena cuando el tamaño del divisor es grande comparado con el del dividendo; por ejemplo, si el divisor fuese 50 y el dividendo fuera un máximo de 255, sólo habría que hacer cinco restas en el peor de los casos; por lo cual este método de división sería eficaz.

1563/100=?

| | | | | CUENTA (COCIENTE) |
|--------------------------------------|----------|----------|------|-----------------------|
| DIVIDENDO | 00000110 | 00011011 | 1563 | 0 |
| DIVISOR | | 01100100 | -100 | |
| | 00000101 | 10110111 | | 1 |
| | | 01100100 | -100 | |
| | 00000101 | 01010011 | | 2 |
| | | 01100100 | -100 | |
| | 00000100 | 11101111 | | 3 |
| | | 01100100 | -100 | |
| | 00000100 | 10001011 | | 4 |
| | | 01100100 | -100 | |
| | 00000100 | 00100111 | | 5 |
| | | 01100100 | -100 | |
| | 00000011 | 11000011 | | 6 |
| | | 01100100 | -100 | |
| | 00000011 | 01011111 | | 7 |
| | | 01100100 | -100 | |
| | 00000010 | 11111011 | | 8 |
| | | 01100100 | -100 | |
| | 00000010 | 10010111 | | 9 |
| | | 01100100 | -100 | |
| | 00000010 | 00110011 | | 10 |
| | | 01100100 | -100 | |
| | 00000001 | 11001111 | | 11 |
| | | 01100100 | -100 | |
| | 00000001 | 01101011 | | 12 |
| | | 01100100 | -100 | |
| | 00000001 | 00000111 | | 13 |
| | | 01100100 | -100 | |
| | 00000000 | 10100011 | | 14 |
| | | 01100100 | -100 | |
| | 00000000 | 00111111 | | 15=COCIENTE |
| | | 01100100 | -100 | |
| (DIVIDENDO SE VUELVE NEGATIVO) | 11111111 | 11011011 | | |
| | | 01100100 | +100 | (RESTO RECUPERADO) |
| | 00000000 | 00111111 | | RESTO = 63 |

Figura 6.7. Método de división por resta sucesiva

División con recuperación

La respuesta a una técnica de división relativamente rápida reside en la división bit a bit. El método de la división con recuperación recuerda la forma en que dividimos con lápiz y papel. La figura 6.8 muestra el esquema.

| | |
|------------------|---------------|
| DIVIDENDO = 3500 | DIVISOR = 96 |
| 0000110110101100 | 0110 0000 |
| <hr/> | |
| 1100000 | 100100 |
| <hr/> | |
| 0001101011 | COCIENTE = 36 |
| <hr/> | |
| 1100000 | |
| <hr/> | |
| 000101100 | |
| RESTO = 44 | |

Figura 6.8. Ejemplo de división bit a bit

Dividimos un dividendo de 16 bits entre un divisor de 8. La regla general para el tamaño del cociente, por cierto, es que debe ser igual el número de bits del dividendo, ya que el valor 1 es un divisor legítimo.

Empezamos con los dos valores absolutos (números positivos) de +96 para el divisor y +3500 para el dividendo.

Para empezar, 0110 0000 (96) se resta del primer 0 del dividendo de 0000 1101 1010 1111. Por supuesto, esta resta es imposible y el resultado es negativo. Si el resultado es negativo después de cualquier resta, "recuperamos" el residuo anterior sumando el divisor, como hacemos en este ejemplo. Continuamos con esta resta, comprobamos si es negativa o positiva y si es preciso recuperar o no para cada uno de los dieciséis bits del dividendo. Cada vez que el resultado es negativo, recuperamos sumando el divisor y poniendo un 0 en el cociente. Cuando el resultado es positivo, no recuperamos y ponemos un 1 en el cociente. Al final de las dieciséis iteraciones, el cociente refleja el valor final, y el residuo es el resto de la operación. Puede que el residuo tenga que ser recuperado por medio de una suma final, para obtener el verdadero resto.

Esta técnica de papel y lápiz puede ser casi duplicada con exactitud por el microordenador. El dividendo se introduce en un registro de 24 bits (tres registros de 8 bits). El divisor se introduce en un registro de 8 bits. Los dos registros se alinean de la forma que muestra la figura 6.9. El divisor se resta de los ocho bits de arriba del dividendo. Si es necesario se hace una recuperación sumando el divisor.

Después de que la resta y la posible recuperación se han efectuado, el dividendo se desplaza hacia la izquierda una posición. Al mismo tiempo, el bit de cociente (0 ó 1) es desplazado hacia la izquierda en el registro del dividendo desde el lado derecho. Al final de las dieciséis iteraciones se habrán desplazado dieciséis cocientes al registro del dividendo, y estarán en los dieciséis bits de abajo. Los ocho bits de arriba almacenarán un resto de 8 bits, suponiendo que se haya hecho alguna recuperación final.



Figura 6.9. Implementación de la división bit a bit

División con signo y sin signo

Los operandos de todas las divisiones anteriores no tenían signo. Como en el caso de la multiplicación, el camino más fácil para realizar una división con signo es encontrar el “sentido” (positivo o negativo) del producto final, tomar el valor absoluto de dividendo y divisor y realizar entonces una división sin signo, dando un valor negativo al cociente, si se dividía una cantidad positiva entre una negativa o una negativa entre una positiva.

Hay un pequeño problema. Cuando $-32,768$ se divide por -1 , el resultado es $+32,768$, que producirá un error de desbordamiento en el cociente. Se puede comprobar fácilmente esta condición antes de proceder a dividir.

En el capítulo siguiente trataremos las operaciones aritméticas en “múltiple precisión”, que nos permitirán trabajar con valores mayores de los que pueden almacenarse en dieciséis bits. Antes de entrar en este tema, sin embargo, los siguientes ejercicios están destinados a lectores que quieren adquirir soltura en los algoritmos y cálculos de multiplicación y división.

Ejercicios

1. ¿Cuál es el resultado de la multiplicación “ocho por ocho” sin signo de 1111 1111 por 1111 1111? ¿Cuáles son los valores decimales equivalentes?

2. ¿Cuál es el resultado de la división sin signo de 1010101011011101 entre 0000 0000? ¿Cuál es el resultado de la división sin signo de 1111 1111 1111 1111 entre 0000 0001? ¿En qué condiciones no puede permitirse esta división en un ordenador?
3. Si todos los operandos son números con signo, ¿cuál será el signo del resultado de estas operaciones?

$$10111011 \times 00111000 = ?$$

$$1011011100000000 / 10000000 = ?$$

Múltiple precisión

Muchas veces es necesario trabajar más allá de la *precisión* ofrecida por ocho o dieciséis bits. Las cantidades del mundo real, como las constantes físicas, datos de cuentas y otros valores numéricos exceden, a menudo, el rango de +32,767 a -32,768 de cantidades de 16 bits. Los valores mayores pueden expresarse utilizando varios bytes para almacenar cada operando. En este capítulo veremos cómo puede hacerse esto utilizando bytes de 8 bits.

¿Tienen algo que ver las series de Fibonacci con la televisión?

"Scusame. ¿Es éste un ristorante de Big Ed?"

Big Ed se volvió y vio a un hombre sonriente, con un gran cuaderno entrando en su restaurante. "Sí, señor. Aquí es. ¿En qué puedo servirle?"

"Me gustaría una poca de lasagne y chianti, si por favor."

"Por supuesto, señor. Por supuesto, ¿podría usted no pronunciar con acento italiano?" El escritor se pone muy nervioso. "No es muy bueno imitando acentos extranjeros..."

"Oh, lo siento. Tengo que utilizarlo en los ciclos de conferencias. Me temo que he adquirido malos hábitos."

"¿Ciclos de conferencias?"

"Sí. Uno de mis parientes lejanos fue Leonardo de Pisa, también conocido como Fibonacci, el gran matemático italiano. Me encargo de su trabajo. Por eso estoy aquí, en el Valle del Silicio. Quiero usar un ordenador para analizar las series de Fibonacci. Estaba un poco consternado al comprobar que no tienen bastante precisión para hacerlo."

"¿Qué quiere decir con eso? Pensaba que los microordenadores podían manejar cifras de cualquier tamaño."

"Bueno, pueden manejar *aproximadamente* números de cualquier tamaño en *punto flotante*, pero así no se obtiene una precisión exacta. Por ejemplo, el valor 122,234,728,956 sólo podría manejarse en BASIC como 122,235,000,000, perdiendo el resto de los dígitos. Necesito precisión exacta para mi trabajo en las series de Fibonacci."

"Realmente, es que no veo la televisión, ni me interesan los seriados americanos..."

"No, verá, lo siento... Usted verá, las series de Fibonacci funcionan así. Si usted toma los números 1 y 1 y los suma, obtiene 2. Ese es el tercer término de la serie. Ahora, tome el 1 y el 2 y súmelos para obtener 3. Este es el cuarto término. A continuación, sume 2 y 3 para obtener 5. Ese es el quinto término. Bueno, si usted continúa así, obtendrá la serie de 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, etc."

"Bueno, parece bastante sencillo. ¿Sirve para algo?"

"Tienen ciertas aplicaciones en la representación matemática y en la naturaleza, pero el principal interés de la serie parece centrarse en un grupo especial de gente que se divierte con juegos matemáticos. Hay literalmente decenas de miles de personas en el mundo que investigan las propiedades de las series de Fibonacci. Mire, sólo a mi conferencia de Inlog acudieron 134 ingenieros, programadores y científicos, más un caballero que pedía el precio del alquiler de la sala de conferencias.

Resumiendo, sin embargo, mi principal problema al trabajar con la serie es que los términos se hacen muy grandes rápidamente. El vigésimo cuarto término de la serie es 46,368, demasiado grande para almacenarlo en dieciséis bits. Tuve que acudir a la múltiple precisión para procesar los términos más grandes de la serie."

"¿Múltiple precisión? ¿Cómo funciona eso?"

"Bueno, como usted probablemente ya sabe por su relación con la gente de las fábricas de ordenadores de aquí, ocho bits almacenan valores de hasta 255, y dieciséis bits hasta 65,535; si los números son sin signo, por supuesto. Yo quería almacenar números de hasta 18,446,745,073,709,548,616; eso cubriría los números de Fibonacci hasta casi el centésimo término."

"Usted necesitaría cientos de bytes para eso", exclamó Ed.

"La verdad es que no; preste atención: cuatro bytes almacenan 4,294,967,296, seis bytes almacenan 281,474,976,710,656, y ocho bytes almacenan 18,446,745,073,709,548,616. Así que puede usted ver que si yo tuviese un programa capaz de procesar únicamente ocho bytes, tendría precisión más que suficiente. Todo lo que este programa tiene que hacer es trabajar con operandos de ocho bytes para la suma y la resta. Al final, encontré una casa de *software* especializada en programas para calcular números grandes en microordenadores."

"Sí, soy todo oídos..."

"Pero, por desgracia, han conseguido un importante contrato federal para procesar los programas del presupuesto estatal. Ellos están ahora en Washington y yo estoy aquí, buscando otra compañía asesora..."

Suma y resta empleando múltiple precisión

La múltiple precisión no se utiliza para manejar números grandes (como se verá en el capítulo 10, se emplea el punto flotante en su lugar), pero viene bien para ciertos tipos de procesos, como el problema de Fibonacci, las operaciones de alta velocidad o las de alta precisión.

En la suma y en la resta en múltiple precisión se suman o restan dos operandos de una longitud de determinado número de bytes. ¿Cuál es la apariencia de los operandos? Supongamos que hemos determinado que queremos representar números hasta un tamaño de la mitad de 18,446,745,073,709,548,616. Sabemos, por la conversación sobre Fibonacci, que esta magnitud podría contenerse en un número con signo de 8 bytes o 64 bits. El valor de +9,223,372,536,854,774,308 se representaría por:

| | | | |
|----------|----------|----------|----------|
| 01111111 | 11111111 | 11111111 | 11111111 |
| 11111111 | 11111111 | 11111111 | 11111111 |

Los bits se numerarían según nuestra representación estándar de potencias de dos, con el bit 0 al extremo de la derecha y con el 63 a la izquierda, como un *bit de signo*. El valor de -9,223,372,536,854,774,309 (observe que es uno más que en magnitud que el número positivo), se representaría:

| | | | |
|----------|----------|----------|----------|
| 10000000 | 00000000 | 00000000 | 00000000 |
| 00000000 | 00000000 | 00000000 | 00000000 |

Los demás valores estarían entre estos dos límites. Fíjese que hay sólo un bit de signo y que el número se considera como un grupo único de 64 bits, aunque esté *físicamente separado* en ocho bytes.

La suma de los dos operandos en esta precisión de ocho bytes implica tomar los bytes de cada operando de derecha (menos significativo) a izquierda (más significativo) y sumarlos todos. En caso de *acarreo* de la suma del último, éste debe sumarse al byte menos significativo.

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 bit más significativo afectará al byte siguiente y, por tanto, debe guardarse y añadirse. El indicador de acarreo se utiliza para registrar cualquier acarreo de la suma anterior y se utiliza una instrucción de "suma con acarreo" para efectuar esta suma.

Para un operando de ocho bytes, la primera suma es un simple "sumar" (no hay acarreo previo), mientras que las siete sumas siguientes son instrucciones de "suma con acarreo". La figura 7.1 muestra esta operación en un ejemplo de suma de dos operandos de 8 bits.

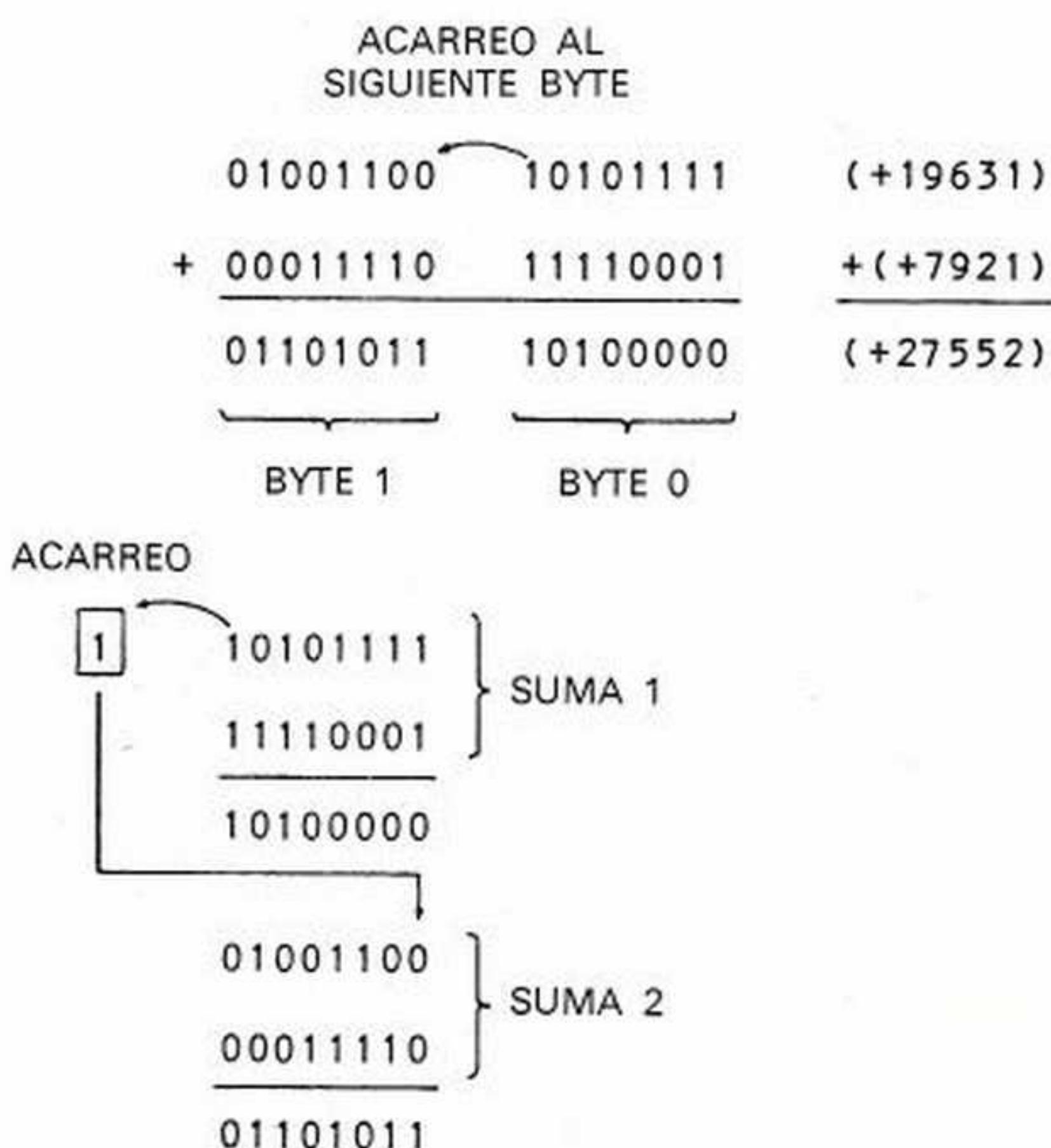


Figura 7.1. Suma de dos bytes en múltiple precisión

La resta en múltiple precisión funciona, en gran parte, de la misma manera, salvo que el acarreo se resta del siguiente byte en lugar de sumarse. La primera resta es normal, mientras que las restantes son opera-

ciones de "resta con acarreo negativo", comúnmente denominadas operaciones de "resta con acarreo". La figura 7.2 muestra la operación de resta con dos operandos de 8 bits.

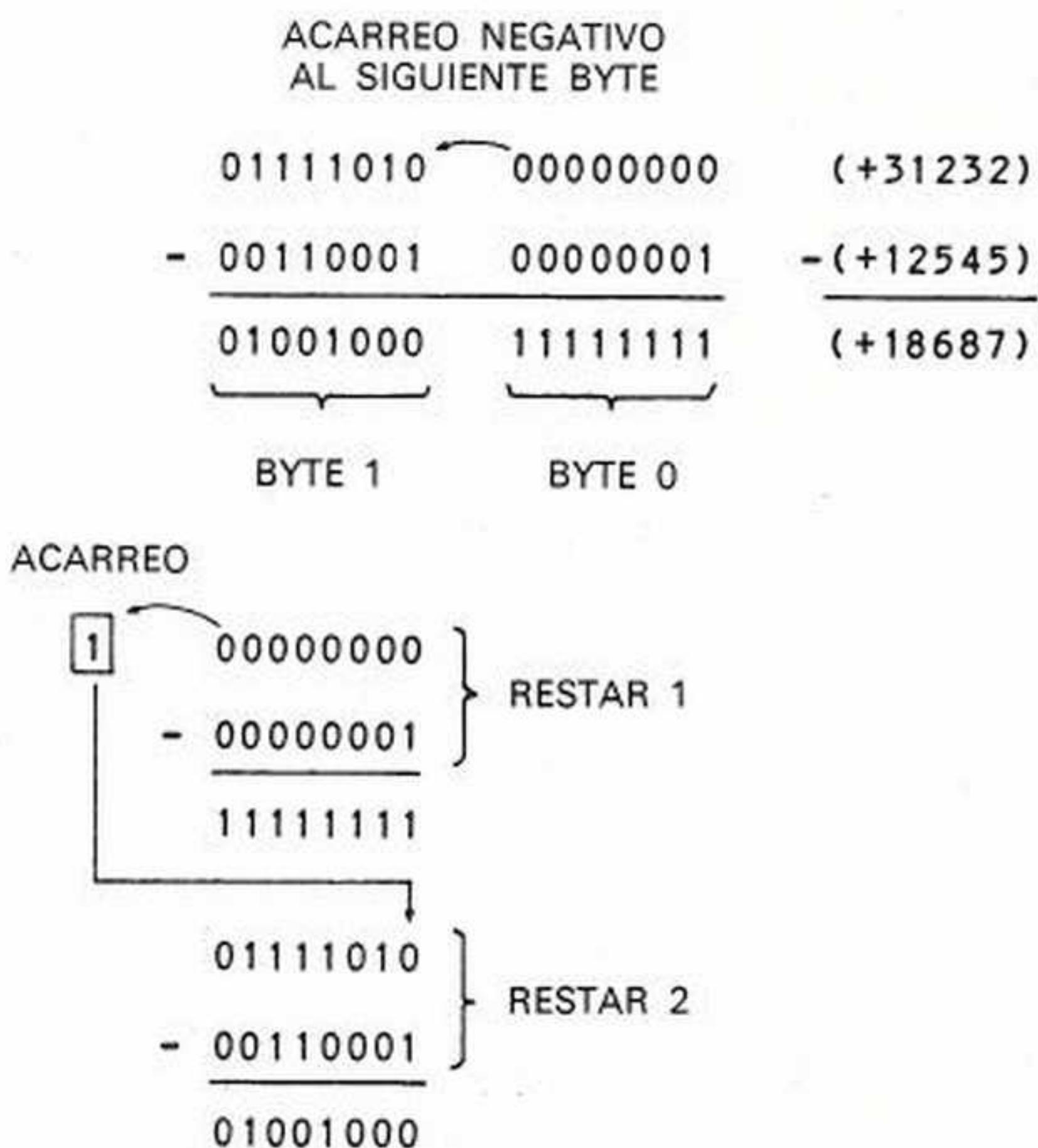


Figura 7.2. Resta de dos bytes en múltiple precisión

Las anteriores operaciones de suma y resta funcionan bien con cualquier combinación de números en complemento a dos; los operandos no tienen que ser valores absolutos.

Cuando los números en múltiple precisión se almacenan en la memoria, debería añadirse una nota de aviso. En los microordenadores con Z-80, el almacenamiento normal en memoria de 16 bits de datos incluyendo las direcciones de memoria y los valores de dos bytes, es el byte menos significativo, seguido por el más significativo, como muestra la figura 7.3.

Si se almacena un número de múltiple precisión en el formato del byte más significativo al menos significativo, no hay problema si los datos acceden byte a byte para: 1) introducir los datos en un registro, 2) sumar o restar el segundo operando y, después, 3) almacenar el byte de resultado en la memoria. Si los datos se introducen en un registro de 16 bits, sin embargo, el microprocesador dará entrada al primer byte en los ocho bits inferiores

2,122,448,880 EN CUADRUPLE PRECISION

01111110 10000001 11111111 11110000

BYTE SIGUIENTE SIGUIENTE BYTE
MAS SIGN. BYTE MAS BYTE MENOS
SIGN. SIGN. SIGN. MENOS SIGN.

REGISTRO DE ACCESO DE OCHO BITS

01111110 10000001 11111111 11110000 MEMORIA DE
BYTE MAS SIGN. SIGN. BYTE SIGN. BYTE
SIGN. MAS SIGN. MENOS SIGN. MENOS SIGN. ALMACENAMIENTO

REGISTRO DE ACCESO DE DIECISEIS BITS

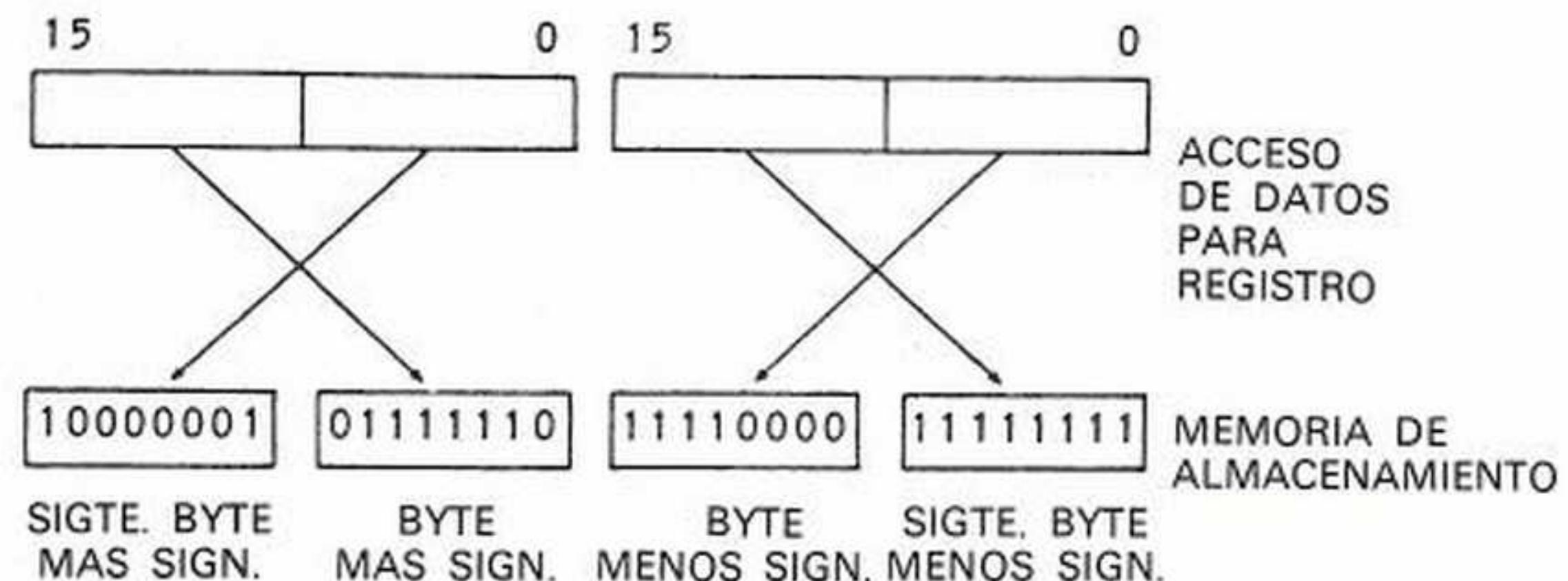


Figura 7.3. Almacenamiento en la memoria con múltiple precisión

del registro, y al segundo byte en los ocho superiores. En este caso, los datos deben disponerse en grupos de dos bytes ordenados del menos al más significativo, como muestra la figura 7.3.

Multiplicación en múltiple precisión

Es muy difícil realizar una multiplicación cuando se utilizan muchos bytes. Una de las razones de ello es que la mayoría de los microprocesadores no tienen registros lo "suficientemente amplios" para llevar a cabo tales multiplicaciones. Lo máximo que se puede hacer es multiplicar un multiplicando y un multiplicador, ambos de dos bytes, para obtener un producto de cuatro bytes. Otro método para llevar a cabo una multiplicación en múltiple precisión es el de aprovechar el *desarrollo* de $(A + B) \times (C + D)$. Este es $(A + B) \times (C + D) = A \times C + B \times C + B \times D + A \times D$.

Suponga que queremos multiplicar los dos números sin signo 778 y 1066. Ambos pueden almacenarse en 16 bits cada uno, como sabemos. Los números aparecen como muestra la figura 7.4. Observe algo interesante: cada número puede almacenarse en dos partes; los ocho bits superiores y los ocho bits inferiores. En el caso de 778, por ejemplo, los superiores son 3×256 , y los inferiores, 10. La parte superior de 1066 es igual a 4×256 , y la pequeña inferior es 42. Podríamos expresar 778×1066 como:

$$778 \times 1066 = ([3 \times 256] + 10) \times ([4 \times 256] + 42)$$

Cuyo desarrollo es:

$$([3 \times 256] \times [4 \times 256]) + (10 \times [4 \times 256]) + ([3 \times 256] \times 42) + (10 \times 42)$$

$$\begin{array}{r}
 3 * 256 + 10 = 778 \\
 \hline
 \overbrace{00000011}^3 * \overbrace{00001010}^{256} \\
 \hline
 \overbrace{00000100}^4 * \overbrace{00101010}^{256} \\
 \hline
 4 * 256 + 42 = 1066
 \end{array}$$

Figura 7.4. Desarrollo en múltiple precisión

El primer término $3 \times 256 \times 4 \times 256$ es lo mismo que 3×4 desplazado hacia la izquierda dieciséis bits. El segundo término es lo mismo que 10×4 desplazado hacia la izquierda ocho bits. El tercero es lo mismo que 42×3 desplazado a la izquierda ocho bits. El cuarto es una simple multiplicación de 10×42 . De hecho, para calcular el producto, todo lo que hay que hacer es:

1. Poner a cero un producto de 32 bits (cuatro bytes).
2. Multiplicar 3×4 y sumar el resultado a los dos primeros bytes del producto.
3. Multiplicar 10×4 y sumar el resultado a los bytes segundo y tercero del producto.
4. Multiplicar 42×3 y sumar el resultado a los bytes segundo y tercero del producto.
5. Multiplicar 10×42 y sumar el resultado a los bytes tercero y cuarto del producto.

Este procedimiento se muestra en la figura 7.5. Sirve, por supuesto, no sólo para este ejemplo, sino para cualquier multiplicador y multiplicando de 16 bits, y también para valores mayores. Divida los operandos en tantas partes como sean necesarias, realice cuatro multiplicaciones, alinee los resultados y sume para obtener el producto.

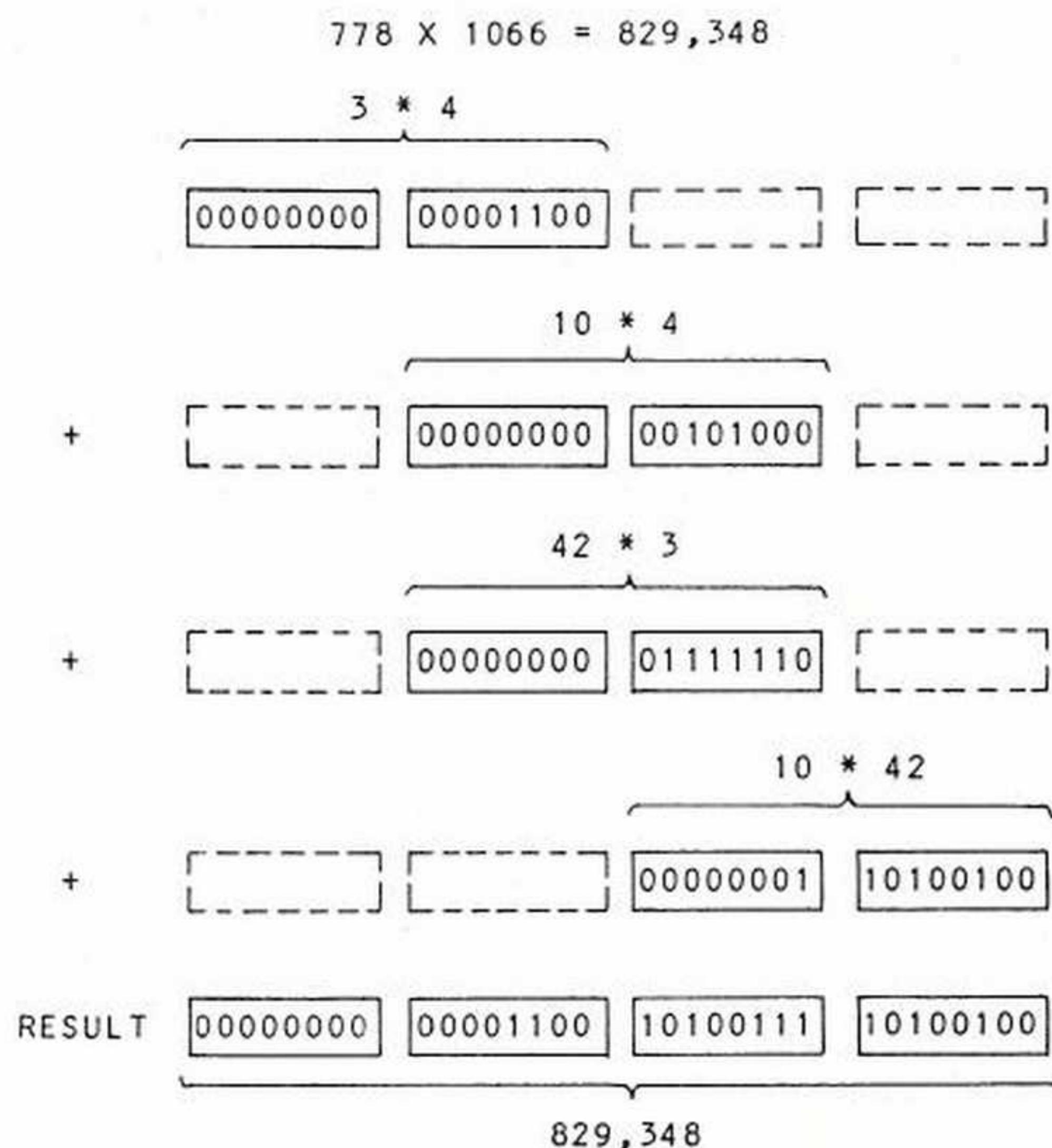


Figura 7.5. Multiplicación en múltiple precisión

Desgraciadamente, las divisiones en múltiple precisión no pueden ejecutarse tan fácilmente. De nuevo, en este caso, no hay bastantes registros en el microprocesador y no son suficientemente amplios para manejar con eficacia la división de muchos bytes. Trataremos la división de números más grandes, con alguna pérdida de precisión, por el método de puntos flotantes en el capítulo 10. Mientras tanto, intente hacer los siguientes ejercicios para fijar lo que ha aprendido en este capítulo.

Ejercicios

1. ¿Cuál es el valor que puede almacenarse en 24 bits?
2. Sume los siguientes operandos de múltiple precisión de cuatro bytes:

$$\begin{array}{r} 00010000 & 11111111 & 01011011 & 00111111 \\ + 00010001 & 00000000 & 11111111 & 00000001 \\ \hline \end{array}$$

3. Reste los siguientes operandos de cuatro bytes en múltiple precisión:

$$\begin{array}{r} 00010000 & 11111111 & 01011011 & 00000000 \\ - 00010000 & 00000000 & 10000001 & 00000001 \\ \hline \end{array}$$

4. Halle los complementos a dos del siguiente operando de cuatro bytes en múltiple precisión:

$$00111111 \quad 10101000 \quad 10111011 \quad 01111111$$

5. Efectúe un desplazamiento lógico a la derecha del siguiente operando de cuatro bytes de múltiple precisión:

$$00111111 \quad 10101011 \quad 101111011 \quad 01111111$$

Fracciones y factores de escala

Hemos hablado mucho acerca de los números binarios a lo largo de este libro, pero todos los números eran valores *enteros*. En este capítulo veremos cómo se pueden representar fracciones en notación binaria y cómo los números pueden escalarse para representar números mixtos.

Big Ed pesa los números

“¿Es usted el propietario?”, preguntó un individuo con una corbata verde, una chaqueta a cuadros rojos y blancos, pantalones color beige y zapatos negros, algo gastados.

“Sí, soy yo”, dijo Big Ed, moviendo su cabeza por la falta de pañuelo en el bolsillo de la chaqueta del desconocido. Ciertamente, no tiene muy buen gusto en el vestir, pensó para sus adentros.

“Bueno, ¡hola! Soy John Upchuck, de Ventas Acme. Tengo aquí una muestra de nuestra Balanza Binaria Mark II, especialmente diseñada para restaurantes como el suyo. Este aparatito no es una cortadora, ni una freidora, ni una picadora... o sea... Lo siento; me estoy yendo por las ramas. Lo que quiero decir es que tengo una balanza que a usted le será imprescindible para pesar con precisión sus porciones de carne. He leído su

anuncio, 'Doce onzas de auténtica carne', en su Big Edburger. Bueno, pues con este pequeño artilugio será cierto eso de que cada Big Edburger pesa exactamente doce onzas; ni más, ni menos."

"¿Cómo funciona?", preguntó Ed, intrigado por el calificativo de "binario".

"Permitame mostrárselo. ¿Ve usted?, hay ocho pesas. La primera es de ocho onzas; la siguiente, de cuatro; la tercera de dos, y esta otra es de una" (véase la Fig. 8.1).

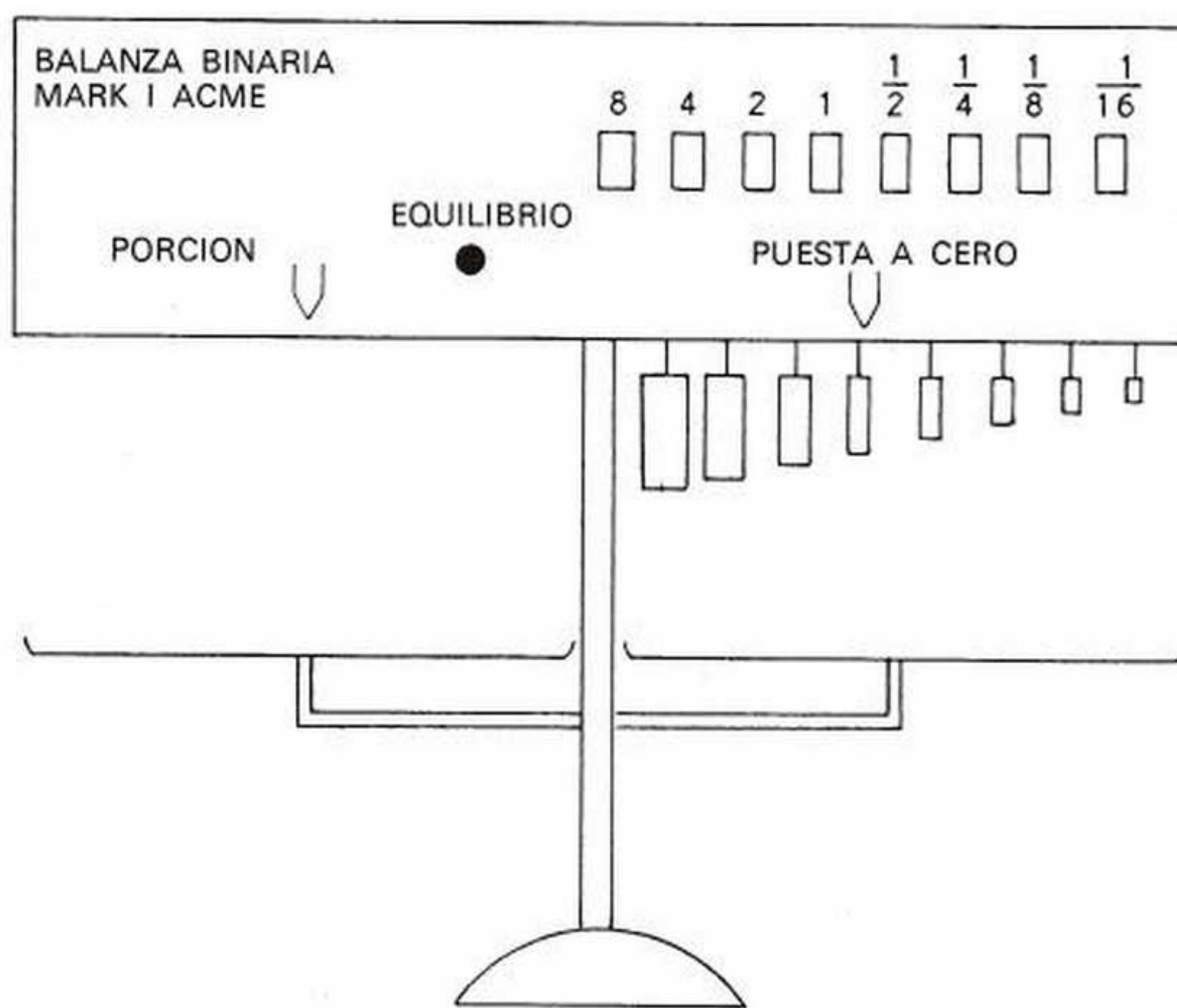


Figura 8.1. La escala binaria

"Esto me recuerda vagamente algo...", musitó Big Ed.

"Bueno; como decía, las siguientes pesas son de $\frac{1}{2}$ onza, de $\frac{1}{4}$, de $\frac{1}{8}$ y, la última, de $\frac{1}{16}$. Un total de ocho pesas, que le permiten pesar cualquier ración de carne o verdura entre $\frac{1}{16}$ y 15 con $\frac{15}{16}$ onzas.

Ahora, la operación es simple. Usted pone la carne en la bandeja de la izquierda de la balanza. Luego pulsa uno o algunos de los botones de la parte frontal; puede usted ver que están marcados: 8, 4, 2, 1, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ y $\frac{1}{16}$. Cada vez que se aprieta un botón, la pesa correspondiente se depo-

sita en la bandeja de la derecha, y una luz ilumina el botón. Pulse otra vez el mismo botón y la pesa se quitará de la bandeja, apagándose la luz. Ahora bien; suponga que usted quiere pesar doce onzas y $\frac{1}{4}$ de carne. Ponga la carne aquí y apriete a continuación los botones 8, 4 y $\frac{1}{4}$."

Así lo hizo, y las tres pesas de 8, 4 y $\frac{1}{4}$ se depositaron en la bandeja; las luces sobre los tres botones se iluminaron. El vendedor colocó carne hasta que la luz llamada de "EQUILIBRIO" se encendió en el centro del panel.

"Bueno, está muy bien", dijo Ed. "Por cierto, quería preguntarle algo: ¿cómo era el modelo Mark I?"

"El modelo Mark I era un diseño primitivo; estaba calibrado en unidades de $\frac{1}{16}$ de onza. El panel estaba marcado así..." Tomó un trozo de mantel cercano y comenzó a dibujar furiosamente (Fig. 8.2).

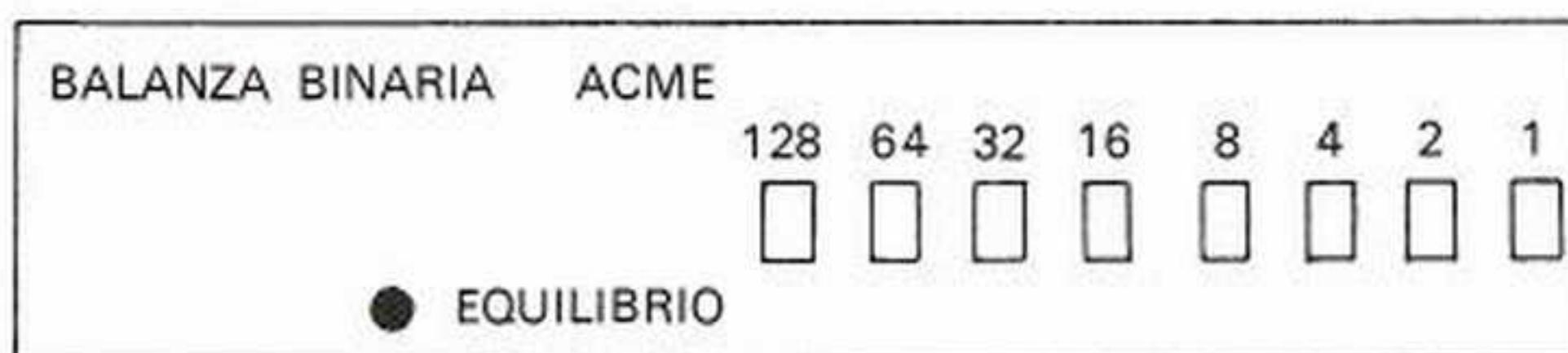


Figura 8.2. El modelo primitivo

"El panel frontal, como puede ver, está marcado 128, 64, 32, 16, 8, 4, 2, 1, representando $\frac{1}{16}$ de onza, $\frac{1}{32}$ de onza, $\frac{1}{64}$ de onza, $\frac{1}{128}$ de onza, $\frac{1}{256}$ de onza, $\frac{1}{512}$ de onza y $\frac{1}{1024}$ de onza. Sin embargo, se notó que su uso era demasiado complicado para profanos, ya que tenían que multiplicar el número de onzas requerido por 16 y, entonces, seleccionar la combinación de botones. La gente que usaba la máquina empezó a referirse a este proceso como *escalamiento*, de forma irónica. El modelo II es mucho más fácil de manejar."

"Bueno, ¿le gustaría quedarse con uno, y ya pagará más adelante?"

"No; ahora mismo, no. Muchas gracias por la demostración, de todas formas. Tenga, coja un sandwich de carne; le hará juego con su corbata verde."

Fracciones en sistema binario

Las fracciones binarias tienen un formato similar al de las decimales. Hay un punto binario en lugar del punto decimal, que separa la porción entera del número binario de la parte fraccionaria (Fig. 8.3). La posición más inmediata a la derecha del punto binario representa un peso de $\frac{1}{2}$; la siguiente posición es $\frac{1}{4}$; la siguiente es $\frac{1}{8}$; la siguiente es $\frac{1}{16}$, etc.

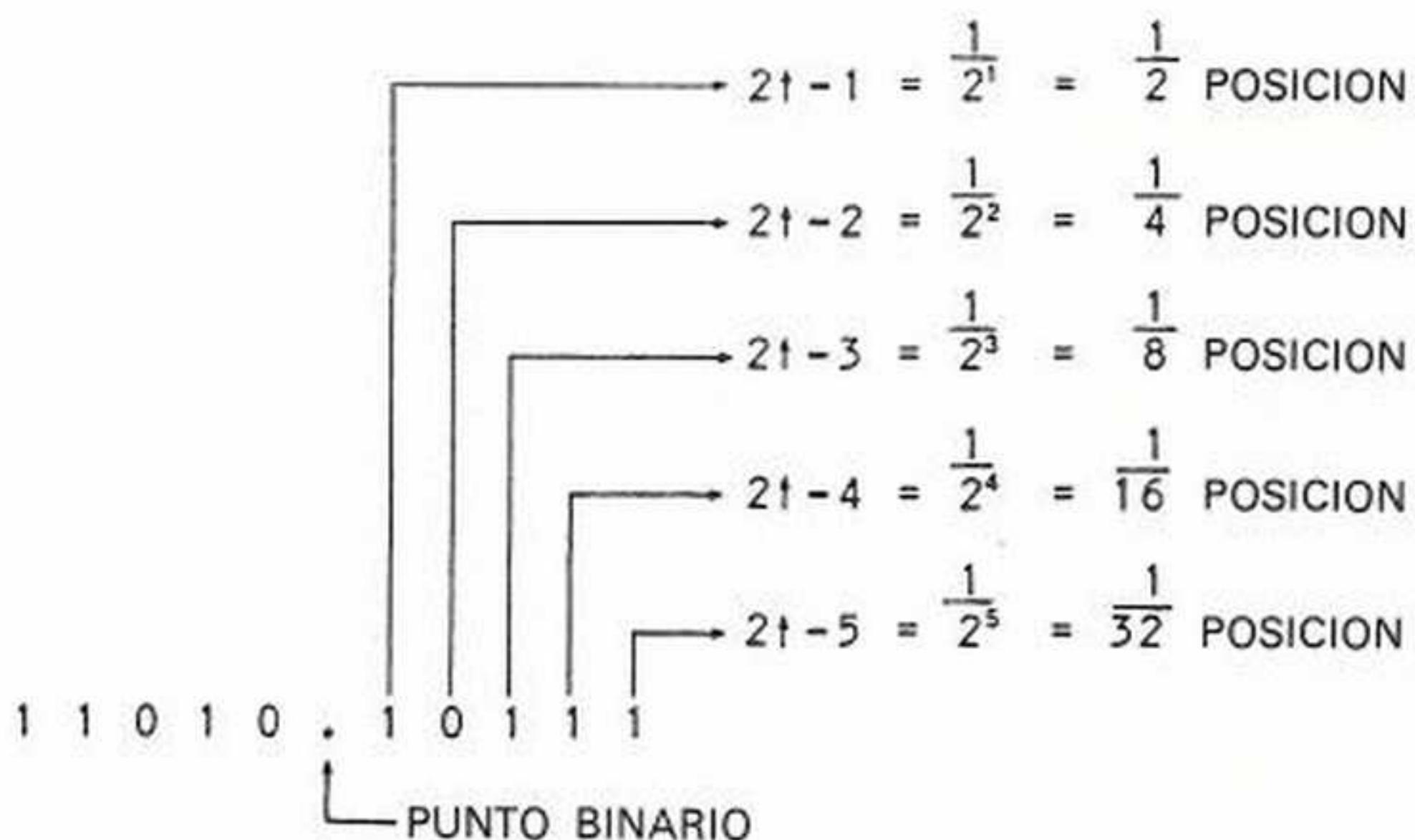


Figura 8.3. Representación binaria de números mixtos

Mientras que las posiciones de los enteros son potencias de dos, las posiciones fraccionarias son los *inversos* de las potencias de dos: $\frac{1}{2} \uparrow 1$, $\frac{1}{2} \uparrow 2$, $\frac{1}{2} \uparrow 3$, etc.

Consideremos algunos ejemplos más. El número binario mixto 0101.1010 se forma a partir del entero 0101, que es el decimal cinco, y la porción fraccionaria .1010. Esto representa $\frac{1}{2} + \frac{1}{8}$ ó $\frac{4}{8} + \frac{1}{8}$, que da $\frac{5}{8}$. El número total, pues, equivale al número decimal $5\frac{5}{8}$ ó 5.625.

El número binario mixto 0111.0101011 se forma a partir del entero 0111, ó 7 decimal, y la parte fraccionaria .0101011. La parte fraccionaria es $\frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{128}$ ó $\frac{32}{128} + \frac{8}{128} + \frac{2}{128} + \frac{1}{128} = \frac{43}{128}$. El número mixto total es, por tanto, 7.3359375.

Para pasar de un número fraccionario binario a decimal, hay que transformar la parte entera por alguno de los métodos anteriormente vistos; luego hay que transformar la parte fraccionaria como si fuera un número entero. Por ejemplo, transformar 0101011 en 43 como si fuera un entero. Primero, cuente el número de posiciones de la fracción y eleve el número 2 a esa potencia. Así, el número de posiciones es 7 y 2 elevado a la séptima potencia, o $2 \uparrow 7$, es 128. Dividiendo 43 entre 128 se obtiene $\frac{43}{128}$, el mismo valor que obtuvimos sumando las fracciones separadas. La figura 8.4 muestra este procedimiento.

Operando con fracciones en sistema binario

Hay varias formas diferentes para procesar números mixtos que contienen fracciones en los microordenadores. El BASIC, por supuesto, emplea

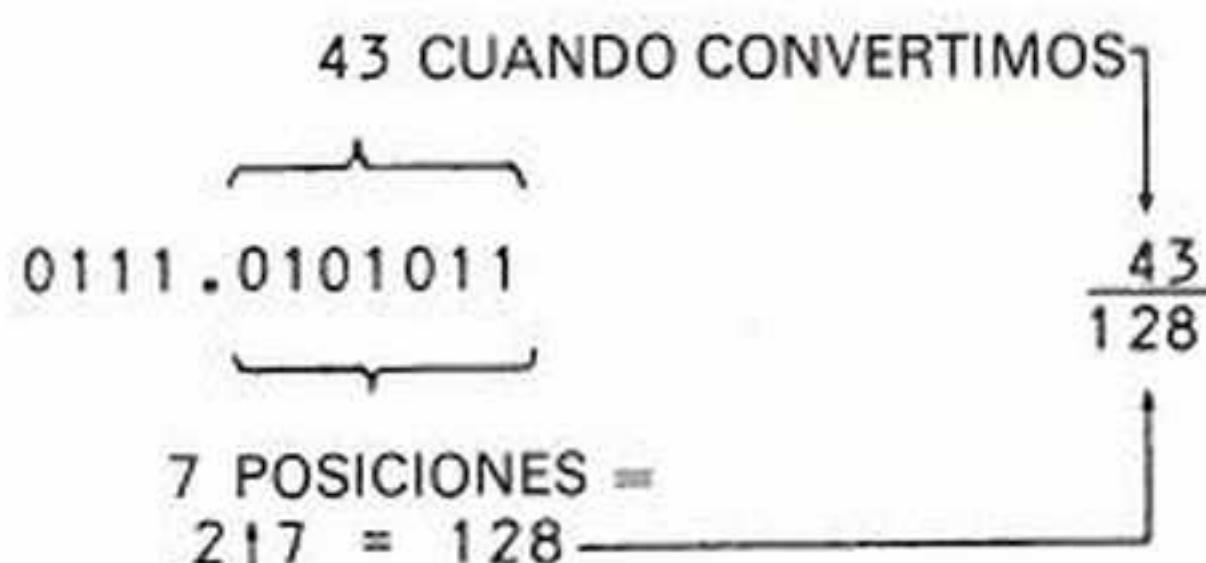


Figura 8.4. Transformación de una fracción binaria

rutinas en *punto flotante*, que, automáticamente, procesan números mixtos; pero nosotros hablamos aquí principalmente de un nivel de lenguaje máquina, o quizás de un código BASIC especializado.

Conservando una fracción separada

El primer método para manejar fracciones consiste en mantener separadas las partes fraccionaria y entera de un número mixto. Este esquema se muestra en la figura 8.5, donde dos bytes de la memoria almacenan la parte entera y un byte adicional almacena la fraccionaria. El punto binario se fija permanentemente entre las partes fraccionaria y entera del número. Ambas partes se procesan separadamente.

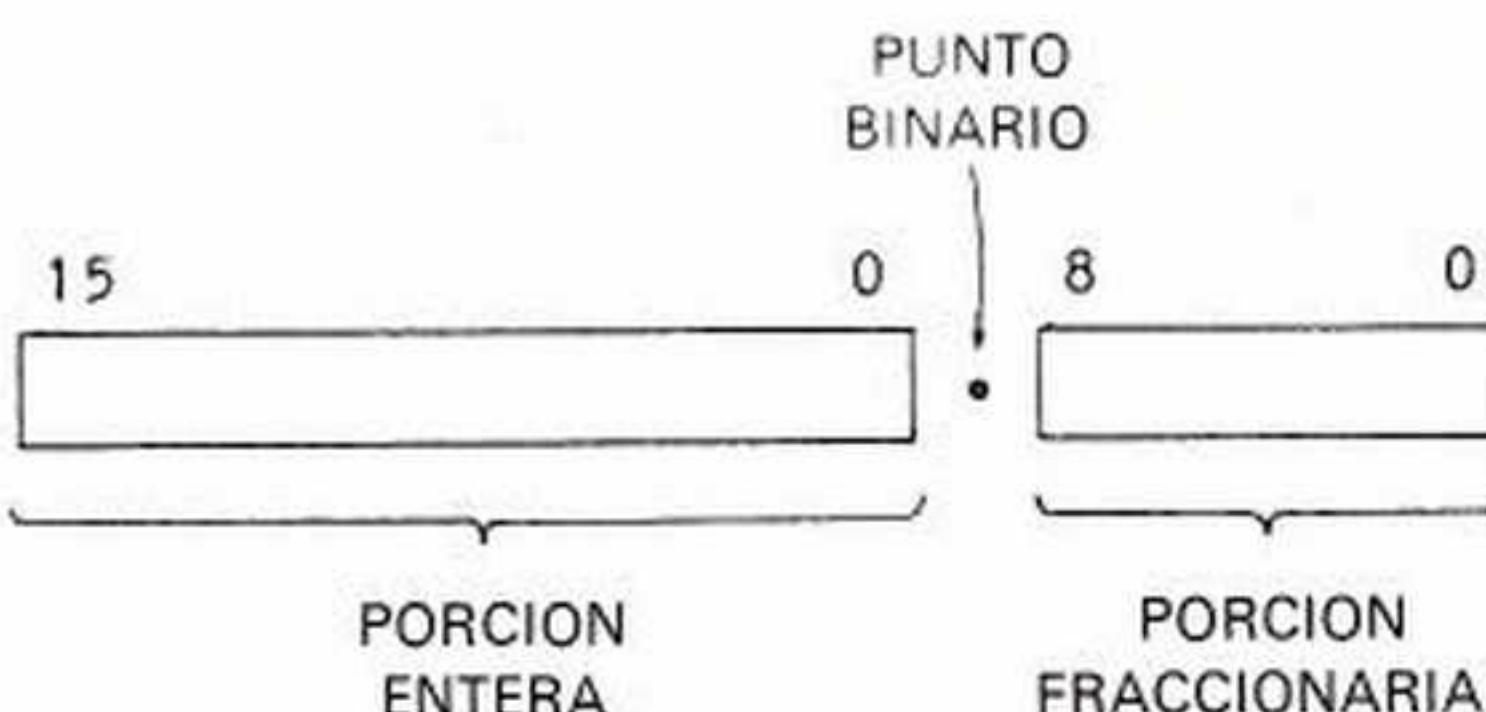


Figura 8.5. Manejo de números binarios mixtos

El máximo número positivo que puede almacenarse en este esquema es +32,767.9960..., representado por 01111111 11111111 en la parte entera y 11111111 en la fraccionaria.

Los puntos de detrás del número indican que el mismo tiene dígitos adicionales. El máximo número negativo que se puede representar es -32,768, y se representa por 10000000 00000000 en la parte entera y

00000000 en la fraccionaria. Para encontrar la magnitud de un número negativo en esta forma hay que utilizar la regla del complemento a dos tratada anteriormente. Cambiar todos los ceros por unos, todos los unos por ceros y sumar uno. En este caso, hay que *sumar uno al bit menos significativo de la fracción* y sumar cualquier acarreo de la misma a la siguiente posición superior de la parte entera.

La ventaja de este método es que la parte fraccionaria del número está disponible para realizar operaciones tales como *redondear* a la centena más próxima, si se trata de un cálculo monetario. De hecho, el número puede considerarse como un valor de 24 bits dividido en dos partes cuando se ha de sumar o restar. La parte fraccionaria se procesa primero, y cualquier acarreo positivo (suma) o negativo (resta) se lleva a la parte entera. La figura 8.6 muestra un ejemplo de suma y resta utilizando este método.

SUMA

$$\begin{array}{r}
 0001000101010101 . 11000000 (+4437.75) \\
 + 0000111000010011 . 10010000 +(+3603.53...) \\
 \hline
 000111101101001 . 01010000 (+8041.31...)
 \end{array}$$

ACARREO

RESTA

$$\begin{array}{r}
 0000111101010101 . 10000000 (+3925.5) \\
 + 0010111000010011 . 10010000 -(+11795.5625) \\
 \hline
 1110000101000001 . 11110000 (-7870.0625)
 \end{array}$$

||

ACARREO
NEGATIVO

COMPLEMENTO
A DOS

↓

$$0001111010111110 . 00010000$$

Figura 8.6. Suma y resta de números mixtos

Factor de escala

Otro método emplea un punto fijo menos obvio. Este método es parecido al utilizado por la Balanza Binaria Mark I de Acme. A cada número se le aplica un cierto factor de escala. El punto binario, aunque no se define con tanta precisión como cuando se situaba entre dos bytes, está ahí en la mente del programador.

He aquí un ejemplo. Supongamos que utilizamos números de 16 bits. Decidimos que queremos tener cuatro bits fraccionarios. Esta decisión se basa en nuestra exigencia de *resolución*. Sabemos que cuatro bits fraccionarios nos permitirán obtener $\frac{1}{16}$ del valor actual, lo que se ajusta a nuestras necesidades.

La apariencia de nuestros números "con un factor de escala de 16" tendrá el aspecto que muestra la figura 8.7. Hay doce bits de entero, un punto binario invisible entre los bits 4 y 3, y cuatro bits de fracción. El máximo número positivo que podemos manejar es 01111111 11111111, que es +2047.9375; el máximo número negativo que podemos procesar es -2048.0000 que es 10000000 00000000.

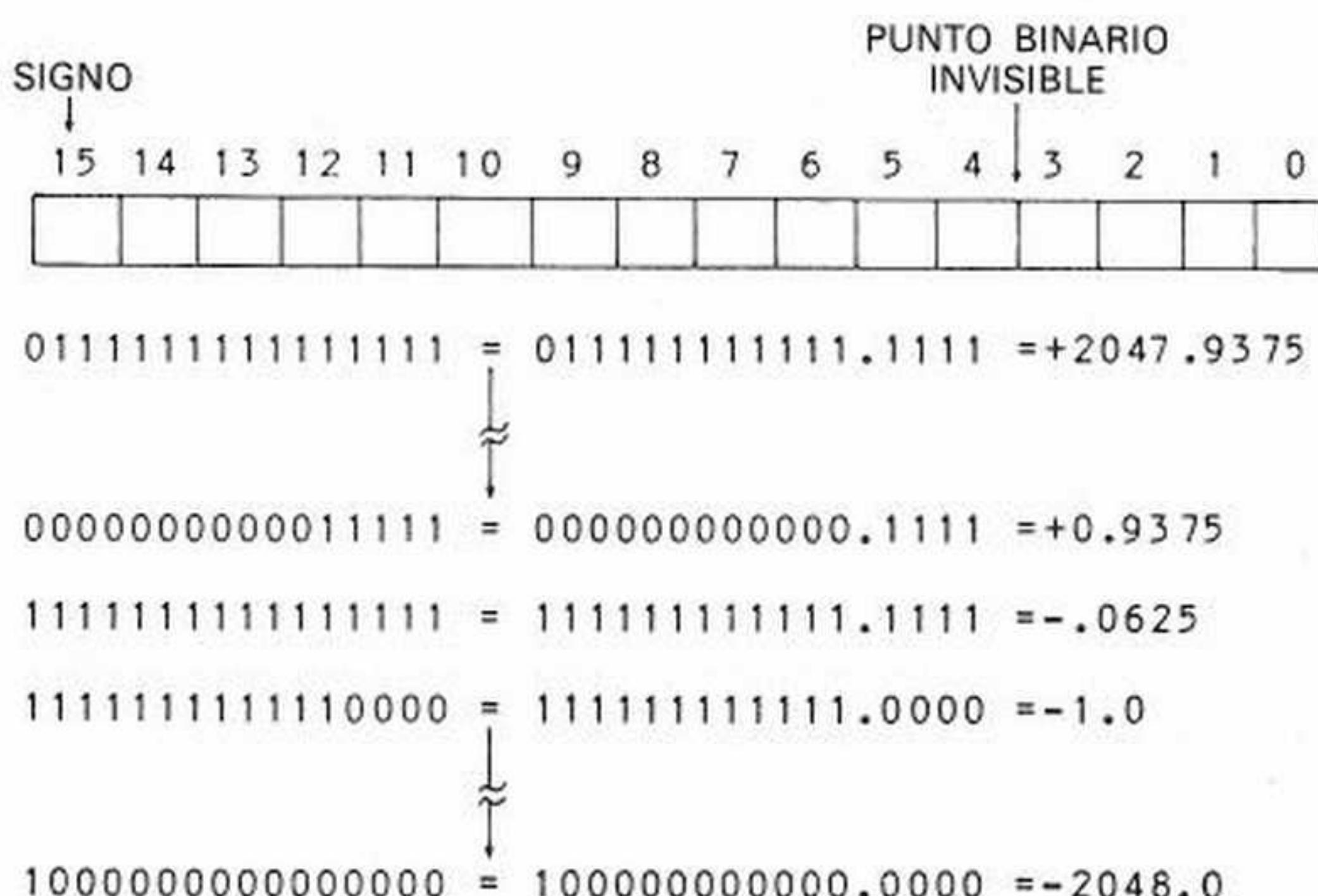


Figura 8.7. Ejemplo de escalamiento

Antes de que podamos procesar algún número del mundo exterior, debemos escalarlos hasta la representación de este punto fijo. Hacemos esto introduciendo un número válido en el rango de -2048.0000 a +2047.9375

y, después, multiplicando por 16. Introduciendo 1000.55, por ejemplo, resulta 16008.8. Truncamos el .8 (lo quitamos) e introducimos el número 16008 como un número binario de 16 bits. (El número reconvertido es $16008_{10} / 16 = 1000.5$, perdiendo, por tanto, algo de precisión, como esperábamos.) Otros números se escalan de la misma forma.

Después, seguimos adelante y sumamos, restamos, multiplicamos y dividimos esos números en cualquier operación que queramos. Aquí aparece el obstáculo. Si sumamos o restamos números modificados de esta forma, el punto binario queda fijo en su posición sin problema. *¡Sin embargo, la multiplicación y la división mueven el punto binario!* La figura 8.8 muestra algunos ejemplos. Tenemos que seguir la pista al punto en la multiplicación y división. Cada multiplicación mueve el punto del resultado dos posiciones más a la izquierda. Cada división mueve el punto dos posiciones más a la derecha.

Al final del proceso, después que la colocación del punto ha sido determinada por la multiplicación y por la división, reconvertimos los números en los valores actuales, dividiendo por 16. Como multiplicar y dividir por 16 se puede hacer fácilmente por medio de cuatro desplazamientos de un bit, la conversión y reconversión son sencillas. La figura 8.9 muestra un típico proceso para números en este formato.

Aunque tener que seguir la pista del punto binario es aburrido, la ventaja de este método es que podemos operar con las partes fraccionaria y entera de los números mixtos al mismo tiempo, sin tener que procesar la parte fraccionaria separadamente y “propagar el acarreo” a la parte entera. Utilice esta aproximación para cualquier proceso que requiera una serie fija de operaciones cuyo número sea reducido.

Conversión de datos

Si ha leído cuidadosamente este capítulo, probablemente se le planteará una pregunta: ¿cómo se pasan los datos en el formato de caracteres del mundo exterior a la forma binaria, escalada o no, y cómo se reconvierten después en una forma adecuada para mostrarse en una pantalla o en una impresora? Responderemos a esta pregunta en el próximo capítulo. Mientras tanto, hemos incluido algunos ejercicios de autoevaluación sobre fracciones y factores de escala.

NUMEROS ORIGINALES

5.25 0101.01
2.5 0010.10

APLICAR UN FACTOR DE ESCALA 4

010101 ($5.25 \times 4 = 21$)
001010 ($2.5 \times 4 = 10$)

SUMA

010101
001010
011111 = 0111.11 = 7.75

RESTA

010101
001010
001011 = 0010.11 = 2.75

MULTIPLICACION

010101
001010
0101010
101010
11010010 = 110100.10 = 52.511
= 1101.0010 = 15.125

DIVISION

001010 10 R1
010101
10 = .10 = .511
1000 = 10.00 = 2.0

Figura 8.8. Operaciones con factores de escala

INTRODUCCION Y
MULTIPLICACION +36.75 POR +20.25

INTRODUCCION

+36.75 → 00100100.110 X 16 =
0000001001001100 (+100.75 ESCALADO)

+20.25 → 10100.01 X 16 =
0000000101000100 (+20.25 ESCALADO)

MULTIPLICACION

| |
|--------------------------|
| 0000001001001100 |
| 0000000101000100 |
| 000000100100110000 |
| 0000001001001100000 |
| 000000100100110000 |
| 000000101110100000110000 |

MOVIMIENTO DEL PUNTO
PARA MULTIPLICAR

RECONVERSIÓN

0000001011101000.0011
↓
744.1875

Figura 8.9. Proceso de números escalados

Ejercicios

1. ¿Cuáles son las fracciones decimales equivalentes a:

.10111; .1; .1111; .000000000001?

2. Pasar las siguientes fracciones decimales a valores binarios:

.365; .3125; .777

3. ¿Cuáles son los números decimales equivalentes a los siguientes números binarios mixtos con signo?

00101110.1011; 10110111.1111

4. Escale estos números por 256 y escriba el resultado en la forma de números binarios con signo de 16 bits:

100; -99

5. Los siguientes números con signo se escalan por ocho. ¿Qué números decimales mixtos representan?

01011100; 1010110101101110

Transformaciones ASCII

Hasta aquí hemos hablado mucho acerca de procesar datos binarios en diversos formatos. Pero, ¿cómo entran los datos en el ordenador? Ciertamente, algunos se almacenan en el programa como *constantes*, pero otros deben introducirse desde el mundo real por medio de un teclado o terminal y, después, mostrarse en pantalla o por impresora. En este capítulo veremos cómo los datos del mundo real se transforman en su representación binaria y viceversa.

Big Ed y el inventor

Big Ed acababa de abrir el restaurante. Estaba ocupado preparándose para recibir al batallón hambriento de ingenieros, programadores, ejecutivos y científicos procedentes de las muchas industrias de microordenadores y semiconductores de los alrededores.

“¿Llego tarde?”, bufó un hombre bajito y gordo, con un traje manchado de tiza y una pajarita.

“¡Oh, hola! ¿Tarde para qué?”

“Para la comida. Quería llegar aquí a tiempo para comer con toda la gente de las compañías de microordenadores.”

"No, llega justo a tiempo. Están al llegar. ¿Espera usted a alguien?"

"No, quería hacer amistad con algunos ingenieros. Me llamo Antón Slivovitz. Soy inventor, y quería tener algún apoyo para mi nuevo invento."

"Bueno, puedo presentarle a alguno de nuestros clientes habituales. ¿Qué clase de cosas inventa usted?"

"Principalmente, cosas de alta tecnología. He inventado un sáser."

"¿Un sáser? ¿Es como un láser?"

"Bueno, algo así. Es una ampliación del sonido por emisión estimulada de radiación. Produce haces coherentes de sonido de una frecuencia determinada. Pensé que, posiblemente, podría convertirlo en algún tipo de rayo de la muerte; pero cuando lo intenté en un centro comercial abarrotado de gente, nadie resultó afectado. También he inventado un bínaco."

"¡Oh, oh... eh... hum..., ¿cuál es su último invento?"

"El que voy a intentar promocionar hoy es un código uniforme para dispositivos periféricos de microordenadores. Verá usted, si todos los fabricantes de terminales, teclados, impresoras, tableros de dibujo electrónicos y otros dispositivos (orientados a caracteres) utilizaran los mismos códigos, entonces se podría emplear fácilmente *cualquier* periférico con cualquier otro, o con cualquier sistema de microordenador. He desarrollado lo que llamo Antón Slivovitz Código de Información Interna, o ASCII, para resumir. Representa todos los caracteres alfabéticos (mayúsculas y minúsculas), dígitos numéricos y caracteres especiales, como el símbolo de la libra, el del dólar y el del tanto por ciento. Incluso está previsto para códigos especiales."

Según Antón describía su código, Bob Borrow, el ingeniero de Inlog, entraba.

"¿Dijo usted que ha desarrollado el código ASCII? ¿Puedo estrecharle la mano, señor... er...?"

"Slivovitz."

"Slivovitz. ¿Puede firmarme un autógrafo en esta tarjeta?"

"Por supuesto, yo... pero... No entiendo. Esta tarjeta dice códigos ASCII. Y *son* lo mismo que el mío!"

"Señor Slivovitz, éstos *son* los códigos ASCII!"

"¿Quiere eso decir que alguien ya los utiliza?"

"Todo el mundo, salvo los de IBM. Y usted sabe ese viejo chiste de que cuando el gran gorila duerme, ja, ja, ja..."

"¡Oh, Dios mío! Bueno, volvamos al sáser..."

Diciendo así, el abatido inventor salió rápidamente por la puerta.

Códigos ASCII

Como debería haber deducido de lo anterior, los códigos ASCII son un conjunto estándar de códigos de caracteres de 7 bits, que se utilizan en los *dispositivos periféricos* para ordenadores, incluyendo microordenadores. Los códigos ASCII se muestran en la figura 9.1. El bit más significativo para

| | | DIGITO HEXADECIMAL MAS SIGNIFICATIVO | | | | | | | |
|---|----|---|-------|---------|----|----|-------|-------|-------|
| | | 0X | 1X | 2X | 3X | 4X | 5X | 6X | 7X |
| DIGITO HEXADECIMAL MENOS SIGNIFICATIVO | X0 | / / / | / / / | ESPACIO | 0 | € | P | / / / | p |
| | X1 | / / / | / / / | I | 1 | A | Q | a | q |
| | X2 | / / / | / / / | " | 2 | B | R | b | r |
| | X3 | / / / | / / / | # | 3 | C | S | c | s |
| | X4 | / / / | / / / | \$ | 4 | D | T | d | t |
| | X5 | / / / | / / / | % | 5 | E | U | e | u |
| | X6 | / / / | / / / | & | 6 | F | V | f | v |
| | X7 | / / / | / / / | ' | 7 | G | W | g | w |
| | X8 | / / / | / / / | (| 8 | H | X | h | x |
| | X9 | / / / | / / / |) | 9 | I | Y | i | y |
| | XA | LF | / / / | * | : | J | Z | j | z |
| | XB | / / / | / / / | + | ; | K | / / / | k | / / / |
| | XC | / / / | / / / | , | < | L | / / / | l | / / / |
| | XD | CR | / / / | - | = | M | / / / | m | / / / |
| | XE | / / / | / / / | . | > | N | / / / | n | / / / |
| | XF | / / / | / / / | / | ? | O | / / / | o | / / / |

LF = SALTO DE LINEA
CR = RETORNO DE CARRO

 VARIAN

Figura 9.1. Códigos ASCII

todos los códigos no se utiliza y, normalmente, se establece como 0. Los que más nos interesan son los que representan los numerales de 0 a 9, el código para un signo más, el código para un signo menos y el código para un punto. Los códigos del 0 al 9 van de 30 a 39H, respectivamente; mientras el signo más es 2BH, el signo menos es 2DH y el de un punto es 2EH.

Cuando los datos se introducen o se sacan del microordenador, se maneja una *cadena de caracteres* de estos códigos. Por ejemplo, introduciendo una cadena de datos desde un teclado, el resultado podría ser el de los códigos mostrados en la figura 9.2, siendo almacenados en una *memoria temporal* que está dedicada al teclado. Sacar datos a una impresora podría tener lugar desde una línea de caracteres en una *memoria temporal de impresión*, como muestra la misma figura. El programa tiene que pasar los caracteres introducidos a binario antes de que el proceso se lleve a cabo.

Una vez finalizado éste, los resultados binarios se transforman de nuevo a caracteres para sacarlos por pantalla o por impresora.

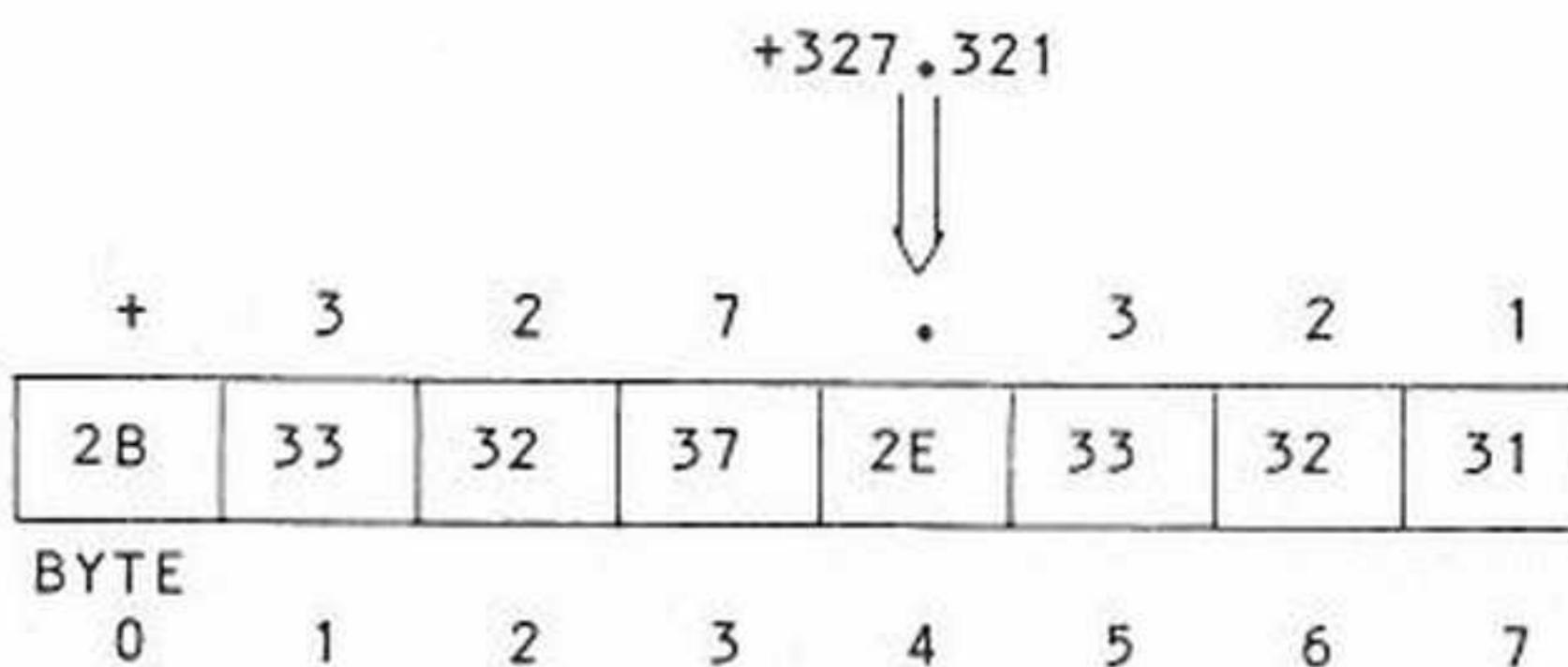


Figura 9.2. Almacenamiento ASCII en una memoria temporal

Paso de ASCII a enteros binarios

Consideremos primero un paso de código ASCII a un valor entero binario. Supongamos que hemos introducido un valor de cinco decimales desde el teclado. El programa del manejo del teclado ha puesto esos caracteres en una memoria temporal, en algún lugar dentro del ordenador.

Antes de convertir los caracteres ASCII (que representan los dígitos del 0 al 9) a un valor binario, tenemos que definir algunas condiciones del valor introducido. Primero, tenemos que definir el tamaño de los datos con que operamos. Si tenemos que operar con enteros binarios de 16 bits, por ejemplo, entonces tendremos que limitar el valor de entrada de -32,768 a +32,767. Cualquier número mayor daría lugar a una entrada inválida. Tendremos que limitar también la entrada a valores enteros, sin fracciones.

Finalmente, tendremos que aceptar únicamente los caracteres del 0 al 9 y un prefijo de “+” o “-”.

La conversión sería así:

1. Poner a cero un producto parcial.
2. Multiplicar el producto parcial por 10. En la primera pasada esto produce un cero.
3. Empezando por el carácter ASCII más significativo, cogerlo y restarle 30H.
4. Sumar el resultado al producto parcial.
5. Si éste no fuese el último carácter ASCII, volver al paso 2.
6. Si hubiese un prefijo “+”, el producto parcial contiene ya el valor binario de 16 bits. Si tiene un prefijo “-”, invertir el signo del producto de 16 bits para obtener el valor negativo.

Por supuesto, queda mucho por decir acerca del establecimiento de indicadores de la entrada en la memoria temporal, obtener el carácter, buscar un valor válido entre 30 y 39H, pasando por alto cualquier prefijo de signo, etc., pero éste es el algoritmo de la transformación general. La figura 9.3 muestra un ejemplo de esta transformación.

Este esquema de conversión puede emplearse para cualquier valor entero de entrada, si bien los valores mayores presentarán problemas a la hora de almacenar el producto parcial entero en los registros de una sola vez.

Paso de ASCII a fracciones binarias

Cuando la línea de entrada representa un número mixto con un entero, un punto decimal y una fracción, el problema de la transformación es más difícil. Tomemos el ejemplo que muestra la figura 9.4. La porción entera desde el primer carácter hasta el punto decimal se transforma según el método anterior, y se guarda el resultado.

La parte fraccionaria se transforma ahora en un valor entero (desde el primer carácter después del punto decimal hasta el último). Luego, hay que determinar cuántos bits va a ocupar la fracción. Escalar esa cantidad. Por ejemplo, si la fracción va a ocupar ocho bits, multiplique el resultado de la transformación por 256; esto puede hacerse por simple desplazamiento o, mejor aún, sumando un byte de ceros al resultado.

Después, tome el resultado escalado y efectúe una división de 10 veces el número de las posiciones decimales de la fracción. Si, como sucede en este caso, hay cuatro posiciones, divida la fracción escalada por 10,000. El cociente de la división es una fracción binaria que se ha de utilizar, y el

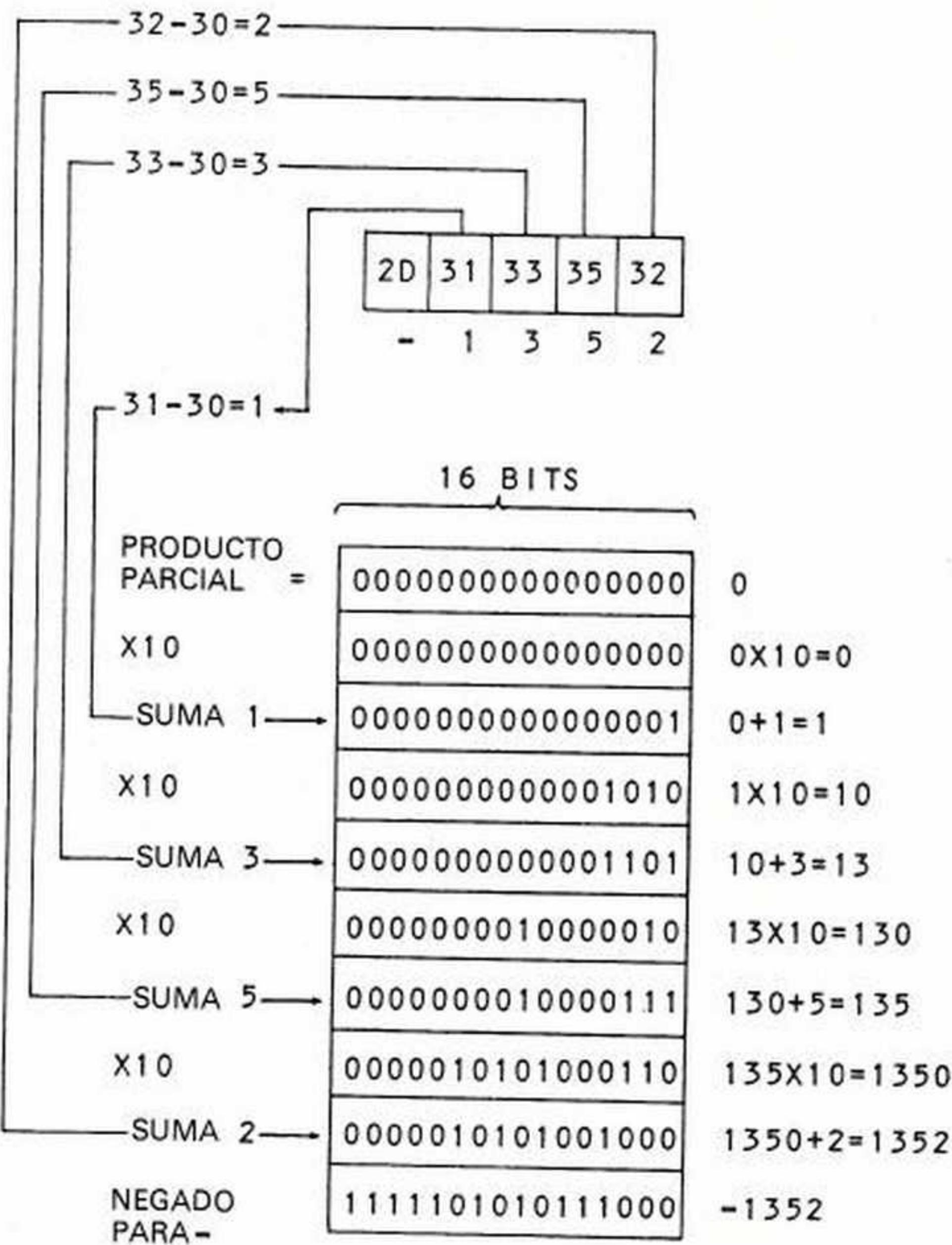


Figura 9.3. Paso de ASCII a enteros binarios

punto binario se alinea después del valor original de la transformación, como muestra la figura 9.4. Si el valor de entrada es negativo, invertir el signo de la parte entera y de la fracción.

Este esquema puede utilizarse para pasar un número mixto, ya el formato de entero/fracción o al de "punto binario implícito", como se describió en el capítulo anterior.

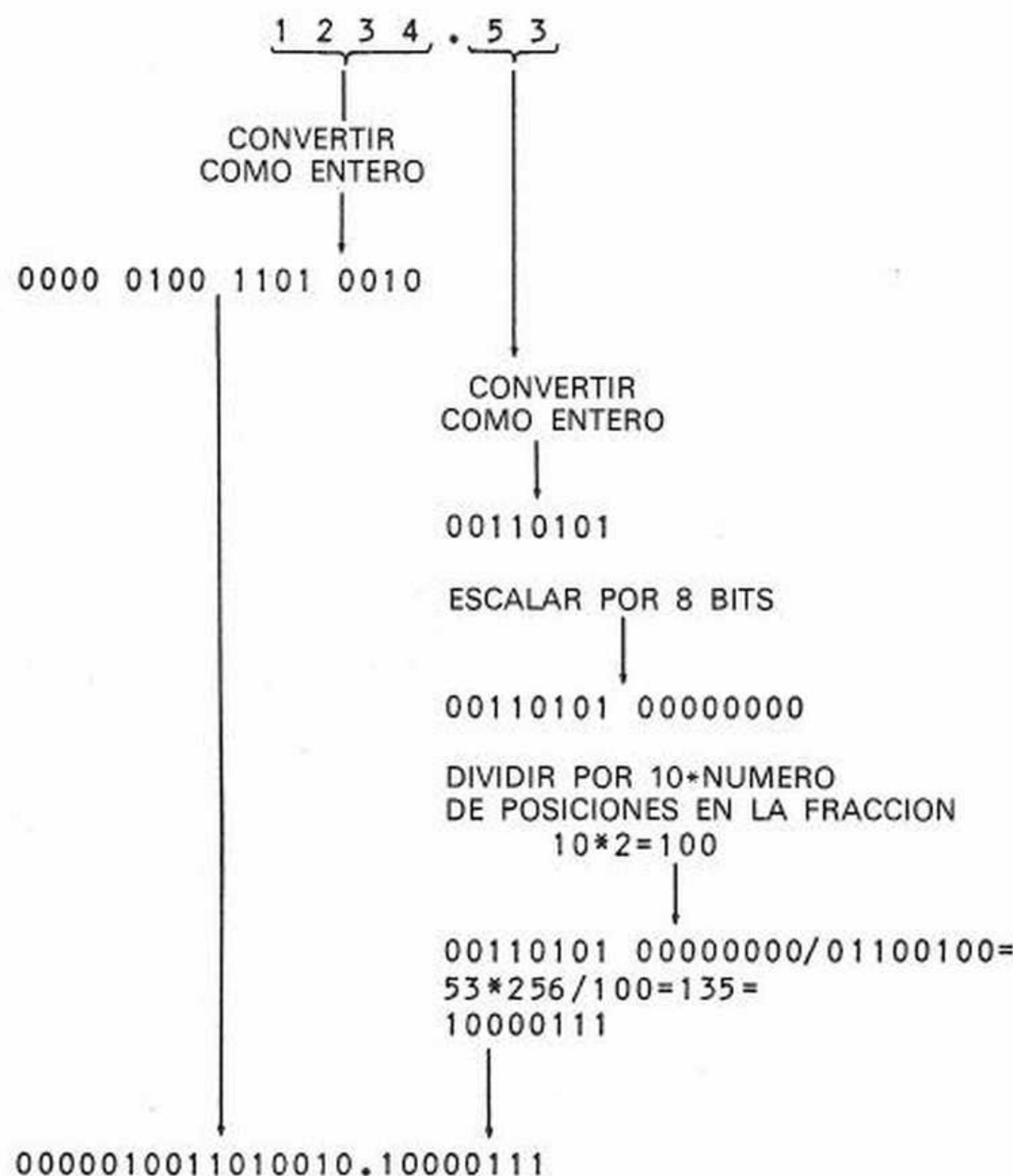


Figura 9.4. Paso de ASCII a fracción binaria

Paso de enteros binarios a ASCII

Después de haber realizado el proceso en el programa, el resultado debe pasarse a una línea ASCII de dígitos decimales, incluyendo un posible signo y un punto decimal.

Consideremos primero el paso de un valor entero de 8, 16 u otro número de bits. Una primera aproximación sería utilizar el algoritmo “dividir por diez y guardar los restos” para obtener una línea de valores. Cada valor puede variar entre cero y nueve. Como el *último* valor representa el primer carácter a imprimir, la transformación completa debe hacerse

antes de la conversión a ASCII. La figura 9.5 muestra un ejemplo de conversión.

El valor entero que ha de ser convertido está en un registro de 16 bits en el microprocesador. Primero se hace una prueba del signo. Si el signo es negativo, se *invierte* para obtener el valor absoluto. Se guarda el signo del resultado.

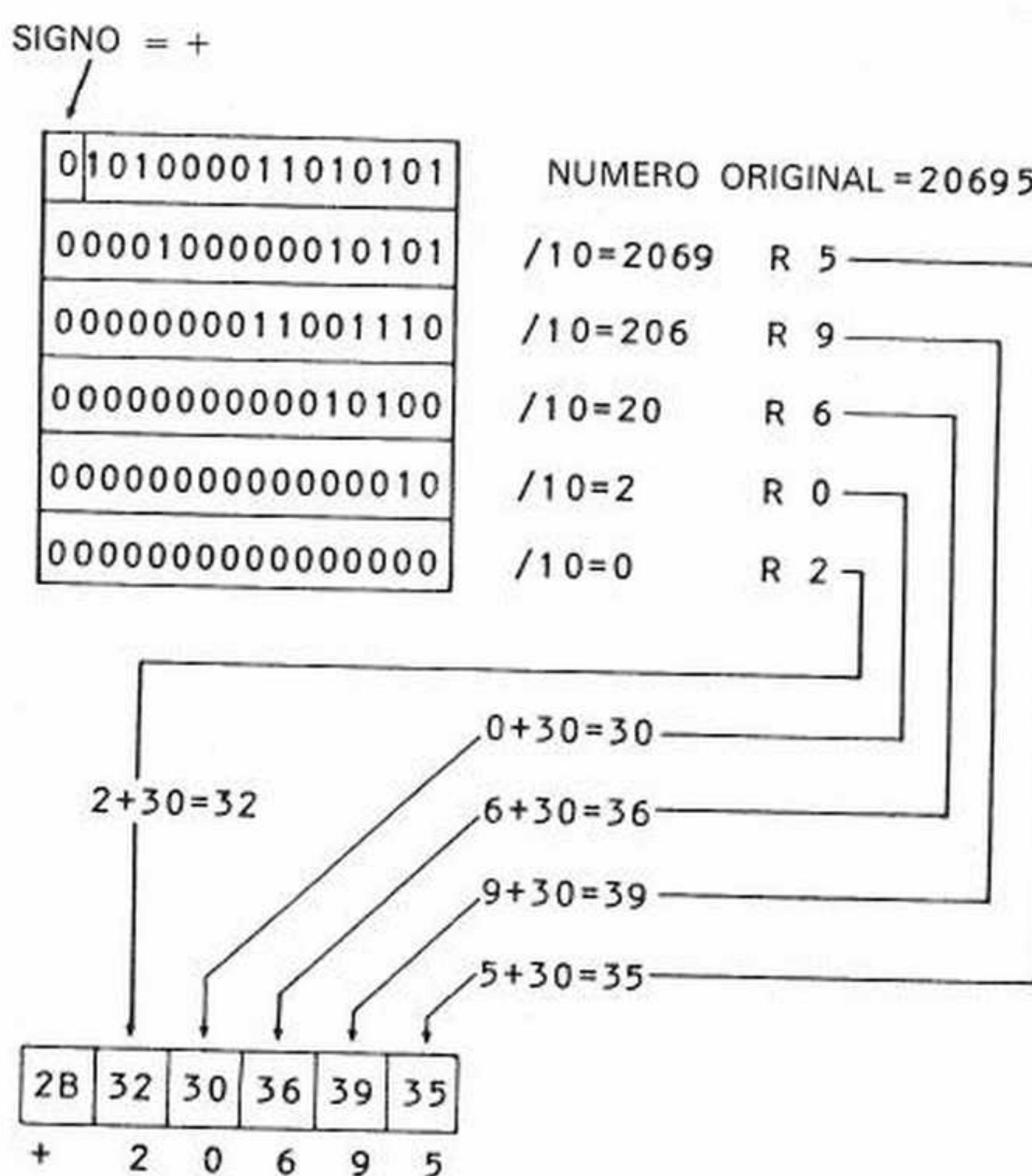


Figura 9.5. Conversión de binario entero a ASCII

El siguiente paso es una división sucesiva por diez, yendo los restos a una memoria temporal de impresión en orden inverso. Cuando el cociente es cero, la división se completa y todos los restos están en la memoria temporal.

Después, cada resto se transforma en ASCII sumándole 30H. El resultado es un dígito ASCII de 0 a 9. Después de transformar el último dígito, se almacena un signo + o - al principio de la memoria temporal de impresión, dependiendo del valor inicial. La línea de caracteres ASCII puede imprimirse llamando a una rutina de impresión.

Paso de fracciones binarias a ASCII

Antes de sacar un número, hay que comprobar su signo. Si el signo es negativo, se pone a 1 un indicador, y se *invierte* el signo del número para obtener su valor absoluto, a fin de obtener su transformación.

El número se separa entonces en una parte entera y otra fraccionaria. Puede dividirse de esta forma, si se mantienen separados el entero y la fracción. Si el número se ha escalado y tiene un punto binario fijo implícito, el desplazamiento separará la fracción y el entero en dos componentes. El entero puede transformarse ahora por el esquema "dividir por diez y guardar los restos", como muestra la figura 9.6. Los restos se ponen en orden inverso en una memoria temporal de impresión. El código ASCII para un punto decimal se almacena ahora en la memoria temporal de impresión después del último resto. Entonces se pasa cada resto a ASCII sumándole 30H.

Finalmente, hay que transformar la parte fraccionaria. Se halla el número de bits de la fracción. Este es en realidad el número de bits que se mantuvo mientras el proceso del número binario mixto tenía lugar. Alinee estos bits en un multiplicando. Luego, cada una, multiplique por una potencia de diez. En este caso, se van a emplear tres posiciones decimales, luego la parte fraccionaria se multiplica por 1000.

Luego, divida el resultado de la multiplicación por la potencia de dos igual al número de bits utilizados en la fracción. En este caso, se emplean cuatro bits, luego debería dividirse por dieciséis el resultado de la multiplicación. Esta división se puede hacer por desplazamiento. Descarte los bits desplazados fuera. El número resultante puede ahora pasarse a ASCII por el método empleado para los enteros, anteriormente descrito: "dividir por diez y guardar los restos" en orden inverso y sumando después 30H para pasar a ASCII. Sume un signo negativo en ASCII al principio de la memoria temporal de impresión y utilice una rutina de impresión para mostrar el resultado final.

Si se pregunta por qué se emplea el punto flotante, los anteriores algoritmos de multiplicación y división deberían arrojar alguna luz sobre la cuestión de por qué puede ser necesaria una forma genérica de manejar un amplio abanico de números en formato estándar. Las operaciones de punto flotante se describen en el próximo capítulo. Son aburridas, pero ¡no tanto como un programa que procese una variedad de números escalados como el que acabamos de ver!

Los siguientes ejercicios le ayudarán a entender el próximo capítulo.

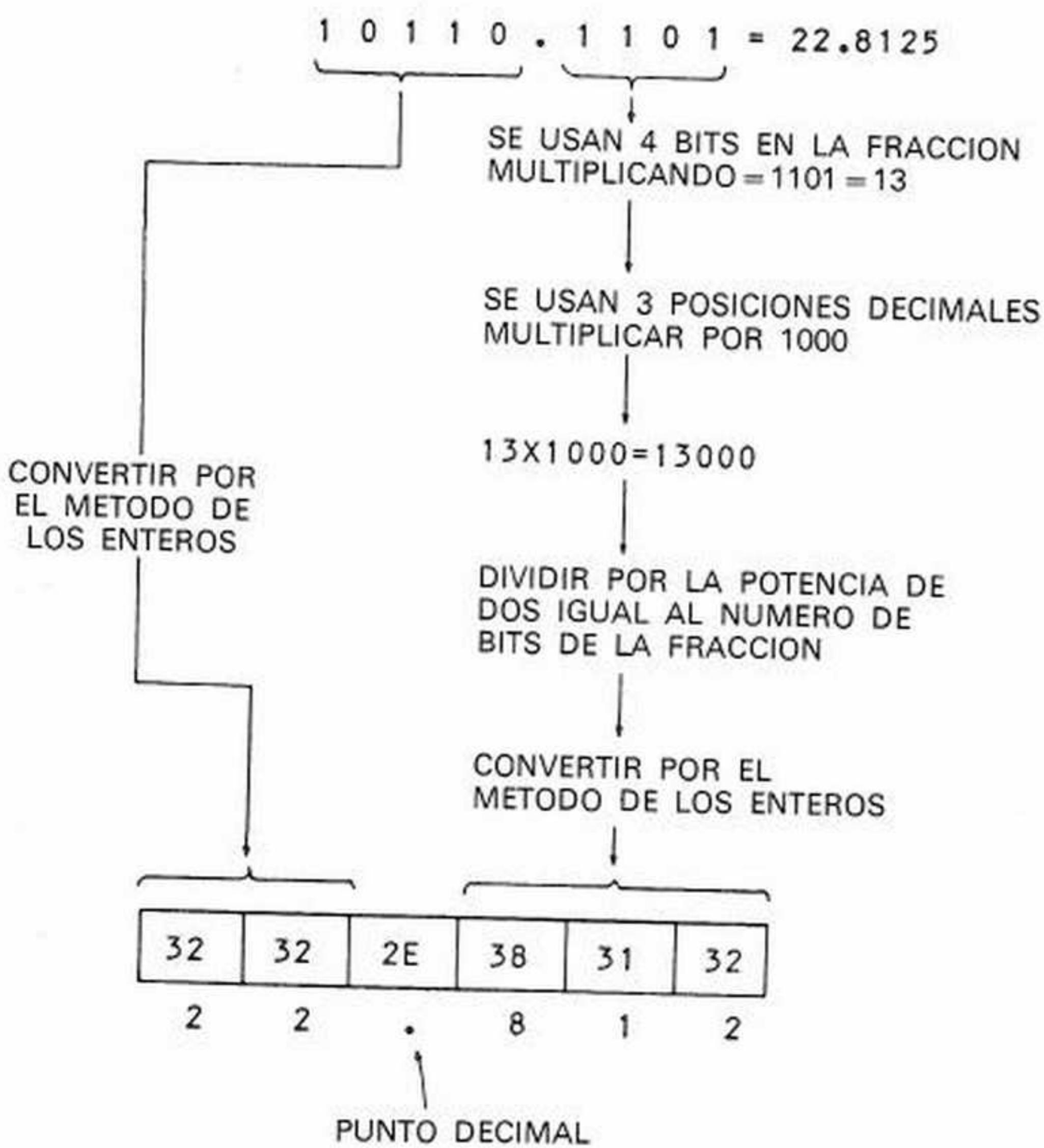


Figura 9.6. Paso de binario a fracción ASCII

Ejercicios

- Una memoria temporal almacena los siguientes valores ASCII. ¿Qué números decimales representan?
 - 2B, 30, 31, 32, 33, 39, 31
 - 2D, 30, 39, 38,
 - 2D, 30, 31, 2E, 35, 32
 - 2B, 2E, 37, 36, 38
- Pasar los siguientes números a ASCII (con signo):

+958.2; -1011.59

10

Números en punto flotante

Los números en punto flotante de los microordenadores expresan valores en *notación científica*, donde cada uno está formado por una *mantisa* y una potencia de dos. En este capítulo veremos cómo se desarrollan las operaciones en punto flotante.

... y tres mil platos combinados para la nave nodriza...

Estaba empezando la noche, y Big Ed se preparaba para cerrar el restaurante. Después de echar fuera a media docena de ingenieros informáticos, que se quedaron contemplando la clara noche californiana, Big Ed cerró la puerta del edificio.

“Bueno; buenas noches, muchachos; hasta ma... ¡¿Qué diablos es eso?!”

Todas las cabezas giraron en la dirección que Ed señalaba con el dedo. Un gran objeto con luces intermitentes de olor naranja estaba suspendido en el aire, encima de la cabeza de Big Ed. Podía oírse un débil zumbido.

“¡Es un OVNI! ¡Por fin he visto uno!”, dijo uno de los ingenieros, excitado y alegre.

Los ojos de todos quedaron fijos en el objeto que flotaba en el aire.

“¡No nos quedemos pasmados, sin hacer nada! ¡Hay que tratar de sacar una foto, o de comunicar con los ocupantes de la nave, o algo!”

“No sé... ¡Recuerden lo que pasó en ‘La guerra de los mundos’!”

“Tengo una linterna en el coche. Un momento; la traeré”, dijo uno de los ingenieros, de espíritu más aventurero que los demás.

Corrió hacia el coche y volvió con una linterna alargada, bastante potente.

“Más vale que ‘ellos’ no piensen que se trata de una especie de rayo laser o algo así, Frank”, previno uno.

“Escuchad, tenemos que intentarlo. He visto todos los programas de Carl Sagan. Sé lo que hay que hacer. ¡Enviaré un número, a ver si responden!”

Apuntó con la linterna directamente al objeto, y la encendió y apagó varias veces.

“Nueve señales; ¡ahora veremos si responde con el mismo número!”

En medio de una gran excitación general, un brillante rayo de luz dio un largo destello, uno corto, uno corto, uno largo y paró.

“¡Eso no es nueve!”, dijo uno de los ingenieros.

“Sí; es nueve en binario. ¡Están probándonos para ver si hemos progresado; si hemos superado la etapa de los equipos perforadores de tarjetas! Vamos a probar otra cosa. Probemos con la serie de Fibonacci.”

Envío un “1, 1, 2, 3, 5, 8, 13, 21, 34” y esperó. Casi inmediatamente después se recibió un “largo, largo, corto, largo, largo, largo”.

“Eso es 110111, para el siguiente término de 55 en la serie”, gritó el ingeniero de la linterna.

“¿Probamos con pi?”, sugirió uno del grupo.

“Sí, vamos a intentarlo. Tres destellos, uno, cuatro, uno..., sí, esos son los cuatro primeros dígitos...”

Todos estaban mirando expectantes, cuando el objeto respondió con una serie de destellos.

“Cópialos, Roy”, dijo el ingeniero de la linterna.

El OVNI mandó una explosión de destellos, y la secuencia se paró.

“¡Chicos, esto no tiene ningún sentido!”, dijo el ingeniero que estaba grabando la respuesta.

La figura 10.1 muestra lo que grabó:

| | | |
|----------------------------|---|---|
| • - . . - . . - = 01001001 | } | PI EN NOTACION EN PUNTO FLOTANTE MICROSOFT™ |
| • . . . - - - = 00001111 | | |
| - - . - - . - = 11011011 | | |
| - - . = 10000010 | | |

Figura 10.1. Respuesta del OVNI

“¡Espera! ¡Ya sé! ¡Esto es pi en notación en punto flotante!”

Los ingenieros se agruparon con excitación en torno al mensaje. Al cabo de los diez minutos siguientes recibieron algunas otras series.

“¿Qué dice ahora?”, preguntó uno de los ingenieros.

“No sé; es diferente del resto. ¡Eh, espera! Esto es ASCII. Se lee... eeeh... ‘PUEDEN LEER ESTO?’ Mensaje de vuelta: ‘SI.’”

El ingeniero de la linterna respondió con un ‘SI’ en ASCII.

“¿Qué dice ahora?”, preguntó uno de los ingenieros, cuando su compañero transformó el mensaje recibido en un texto.

“Parece como un pedido para llevar”, exclamó Ed, mirando por encima del hombro del ingeniero que sostenía el texto. “‘CUATRO SANDWICHES DE REUBEN, DOS DE PATATAS FRITAS, CUATRO COCACOLAS.’ No hay que preguntarse por qué pararon aquí. Al fin y al cabo, un cliente es un cliente...”

Ed volvió hacia la puerta de su restaurante, y sacó las llaves para abrirla...

Notación científica en punto flotante

El punto flotante es, en realidad, una versión binaria de la *notación científica*. La notación científica puede expresar fácilmente números muy grandes y muy pequeños, en un formato uniforme que simplifica los procesos. En la notación científica, un valor se representa por un número mixto y una potencia de diez. El número se compone de un dígito y una fracción.

Tomemos el ejemplo del número de pulgadas en un kilómetro cuadrado. Hay que admitir que esto no es algo que se haga normalmente, a menos que uno se dedique a dividir parcelas para construir colonias de hormigas, pero muestra lo fácil que es trabajar con la notación científica. Hay 39 pulgadas en un metro. En un metro cuadrado, por tanto, hay $39 \times 39 = 1521$ pulgadas cuadradas. El paso a notación científica es así: $1521 \times 10^0 = 1521$, ya que todo número elevado a 0 es uno. Normalizando la parte del número mixto, movemos el punto decimal tres dígitos, de forma que esté entre el 1 y el 5. Por cada desplazamiento a la izquierda, sumamos uno al *exponente*, o potencia de diez, y así tenemos:

$$1521 = 1521 \times 10^0 = 152.1 = 15.21 \times 10^1 = 1.521 \times 10^2$$

La última cifra es la forma normalizada, o estándar, de la notación científica. Ahora queremos hallar el número de metros cuadrados que hay en un kilómetro cuadrado. Sabemos que cada kilómetro tiene 1000 metros,

luego debe haber 1000×1000 metros cuadrados en un kilómetro cuadrado. Vamos a pasar 1000 a notación científica antes de continuar el proceso.

$$1000 = 100.0 \times 10^1 = 10.00 \times 10^2 = 1.000 \times 10^3$$

Para hallar la respuesta final, el número de pulgadas cuadradas de un kilómetro cuadrado, podemos decir:

$$\text{Número de pulgadas cuadradas}/\text{km}^2 = 1000 \times 1000 \times 1.521 \times 10^9$$

que da lugar a 1.521×10^9 . Para expresar esto como un número sin exponente, desplace el punto decimal hacia la derecha y reste uno del exponente. Añada ceros si es necesario.

$$\begin{aligned}\text{Número de pulgadas}/\text{km}^2 &= \\ 1.521 \times 10^9 &= 15.21 \times 10^8 = \\ 152.1 \times 10^7 &= 1521 \times 10^6 = \\ 15210 \times 10^5 &= 152100 \times 10^4 = \\ 1521000 \times 10^3 &= 15210000 \times 10^2 = \\ 152100000 \times 10^1 &= 1521000000 \times 10^0 = \\ &1,521,000,000\end{aligned}$$

Los exponentes pueden también utilizarse en notación científica para expresar números muy pequeños. Calculemos cuántos ángeles pueden bailar en la cabeza de un alfiler. Adoptaremos las dimensiones angélicas estándar de un ángel por $\frac{1}{11.000}$ centímetros cuadrados o 0.0000909. El tamaño de la cabeza de un alfiler corriente (según la Asociación Americana de Fabricantes de Alfileres Corrientes) es 0.000965 centímetros cuadrados. Expresando ambos números en notación científica, tenemos:

$$\begin{aligned}\text{Área de un ángel} &= 0.0000909 \\ .0000909 \times 10^0 &= 0.000909 \times 10^{-1} = \\ 0.00909 \times 10^{-2} &= 0.0909 \times 10^{-3} = \\ 0.909 \times 10^{-4} &= 9.09 \times 10^{-5}\end{aligned}$$

y

$$\begin{aligned}\text{Área de la cabeza de un alfiler} &= .000965 \\ .000965 \times 10^0 &= 0.00965 \times 10^{-1} = \\ 0.0965 \times 10^{-2} &= 0.965 \times 10^{-3} = \\ 9.65 \times 10^{-4} &\end{aligned}$$

En las anteriores conversiones, restamos uno del exponente cada vez que desplazamos el punto decimal a la derecha. La potencia negativa de diez nos permite representar potencias de diez inversas $-1/10$, $1/100$, $1/1000$, etcétera.

Para hallar el número de ángeles bailando el vals del microordenador en la cabeza de un alfiler, dividimos el tamaño de un alfiler por el de un ángel:

$$\text{Número de ángeles sobre la cabeza} = (9.65 \times 10^{14} - 4) / (9.09 \times 10^{14} - 5)$$

Se aplica aquí la regla de que, si las bases son iguales, podemos restar los exponentes para dividir:

$$\begin{aligned}\text{Número de ángeles sobre la cabeza} &= \\ 9.65/9.09 \times 10^{14}(-4 + 5) &= \\ 9.65/9.09 \times 10^{14}1 &= 1.06 \times 10^{14}1 = \\ 10^{14}6 \times 10^{14}0 &= 10.6 \text{ ángeles}\end{aligned}$$

El uso de la notación científica estándar ha simplificado mucho los cálculos, puesto que todos los valores estaban en formato estándar en base 10, y podemos sumar exponentes para multiplicar y restarlos para dividir.

Resulta que sumar y restar dos números en notación científica es más complicado, en algunos aspectos, que multiplicarlos. Consideremos los dos números $3/32$ y $6/60$. En notación científica, son:

$$3/32 = 0.09375 = 0.09375 \times 10^{14}0 = .9375 \times 10^{14}-1 = 9.375 \times 10^{14}-2$$

y

$$6/60 = 0.1 = 0.1 \times 10^{14}0 = 1.0 \times 10^{14}-1$$

Pues bien; a la hora de sumar o restar dos números en notación científica, *sus exponentes deben ser los mismos*. Uno u otro número, por tanto, debe ajustarse para que ambos exponentes sean iguales. Podemos hacer esto, bien moviendo el punto decimal de $9.375 \times 10^{14}-2$ hacia la izquierda y sumando uno para que dé $.9375 \times 10^{14}-1$, o bien moviendo el punto decimal de $1.0 \times 10^{14}-1$ a la derecha y restando uno para de dé $10.0 \times 10^{14}-2$. Una vez igualados los exponentes, podemos sumar o restar.

$$\begin{array}{r} 3/32 + 6/60 = 9.375 \times 10^{14}2 \\ + 10.0 \times 10^{14}-2 \\ \hline 19.375 \times 10^{14}-2 = \\ 1.9375 \times 10^{14}-1 = \\ 0.19375 \times 10^{14}0 = 0.19375 \end{array}$$

Ahora mismo, seguro que usted se estará preguntando: "¿Por qué no se limitaron a sumar las dichosas cantidades en notación 'normal'?" En muchos ordenadores, tiene más sentido pasar a notación científica; es más fácil seguir la pista al punto decimal, por poner un ejemplo.

Uso de potencias de dos en lugar de potencias de diez

Resulta que las reglas para sumar, restar, multiplicar y dividir números de la misma base sirven para *cualquier* base. ¡Podríamos haber utilizado simplemente la base 8, la base 11 o (dijo, triunfante) la base dos!

Hagamos la definición de un formato estándar en base dos para números. Será el mismo que se utiliza en muchos microordenadores. Para empezar, tenemos que elegir una serie de números. Si utilizamos la base dos (algunas máquinas, más grandes, utilizan la base 6), necesitaremos un lugar para colocar un exponente. Podemos reservar un número conveniente de bits para exponente.

Puesto que todo en los ordenadores parece estar estructurado en torno a los bytes, utilicemos un byte para el exponente. Esto nos dará ocho bits, permitiéndonos una serie de 2^{10} a $2^{18}-1$, o de 0 a 255, lo cual permitirá representar números que serían equivalentes a 5.79×10^{176} .

Un momento, no obstante. Necesitamos también potencias negativas de dos, ya que hay que representar valores pequeños. Convertiremos los ocho bits de un exponente en un *código exceso a 128*. Esto significa que sumaremos 128 al valor del exponente para obtener el número almacenado en su byte, y lo restaremos de cualquier resultado para hallar el verdadero exponente. El código exceso a 128 se aplica para simplificar el manejo de números en punto flotante como una simple entidad. La figura 10.2 muestra el formato del exponente y algunos valores de muestra. Tenemos ahora una serie de exponentes, desde $2^{10} - 128$ hasta $2^{18} + 127$ (3.4×10^{176} a 1.7×10^{-38}).

Veamos qué ocurre con la *mantisa*; es decir, el número que se multiplica por la potencia de dos. Más que definir un rango, la mantisa define una *precisión*. Sabemos que dos bytes, o dieciséis bits, nos dan valores desde cero hasta 65,535 y alrededor de $4\frac{1}{2}$ dígitos decimales. No parece lo bastante amplio para muchos problemas. Para continuar con múltiples de bytes, tendremos que acudir a tres bytes, o 24 bits. Ello nos dará de 0 a 16,777,215 o unos 7 dígitos decimales de precisión, que es probablemente una buena solución intermedia para resolver el conflicto entre las necesidades de almacenamiento y la precisión.

EXPONENTE EN
"EXCESO 128"

| <u>VALOR</u> | <u>DESPUES DE AJUSTAR POR RESTA DE -128</u> | <u>POTENCIA DE DOS</u> |
|--------------|---|----------------------------|
| 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 10.2. Formato y valores del exponente

Lo que ahora tenemos como un formato aproximado se muestra en la figura 10.3: tres bits de mantisa, y uno para el exponente. ¿Qué podemos decir de la *normalización*? La regla que nos guía aquí es que nos gustaría alcanzar la máxima precisión posible.

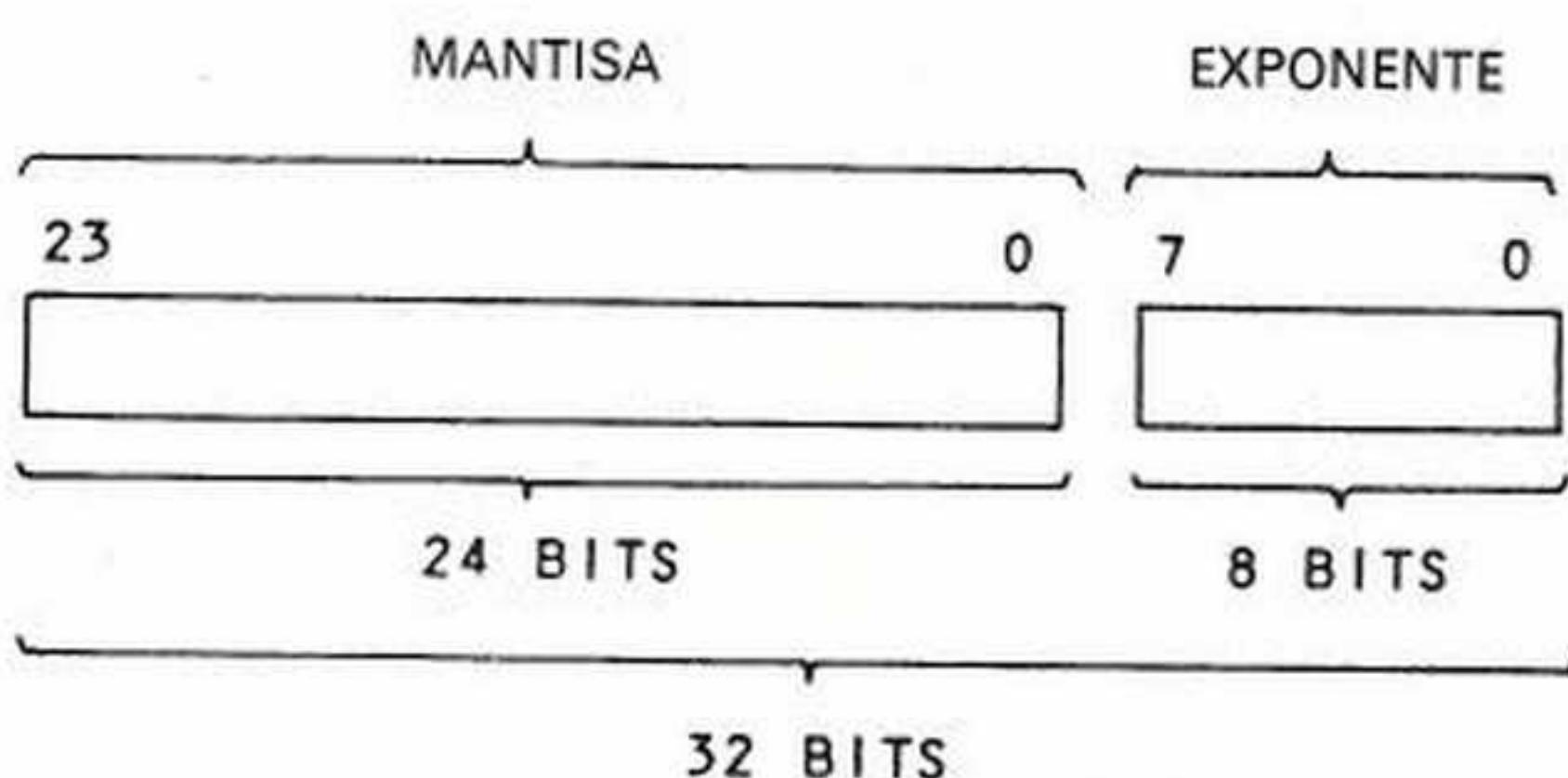


Figura 10.3. Primer esbozo de formato en punto flotante

Para ello, hay que guardar tantos bits significativos como sea posible. Puesto que tenemos un número limitado de bits para almacenar la mantisa, debemos deshacernos de los bits despreciables y conservar sólo los significativos. Esto lo hacemos normalizando la mantisa, de modo que el primer "1" se sitúa inmediatamente a la derecha de un supuesto punto binario, justamente junto al bit de la mantisa, como muestra la figura 10.4. De esta forma, aseguramos la máxima precisión, ya que no hay ceros despreciables junto al primer 1, ni parte *truncada* de la mantisa al final. El primer bit de la mantisa, por tanto, es siempre un 1.

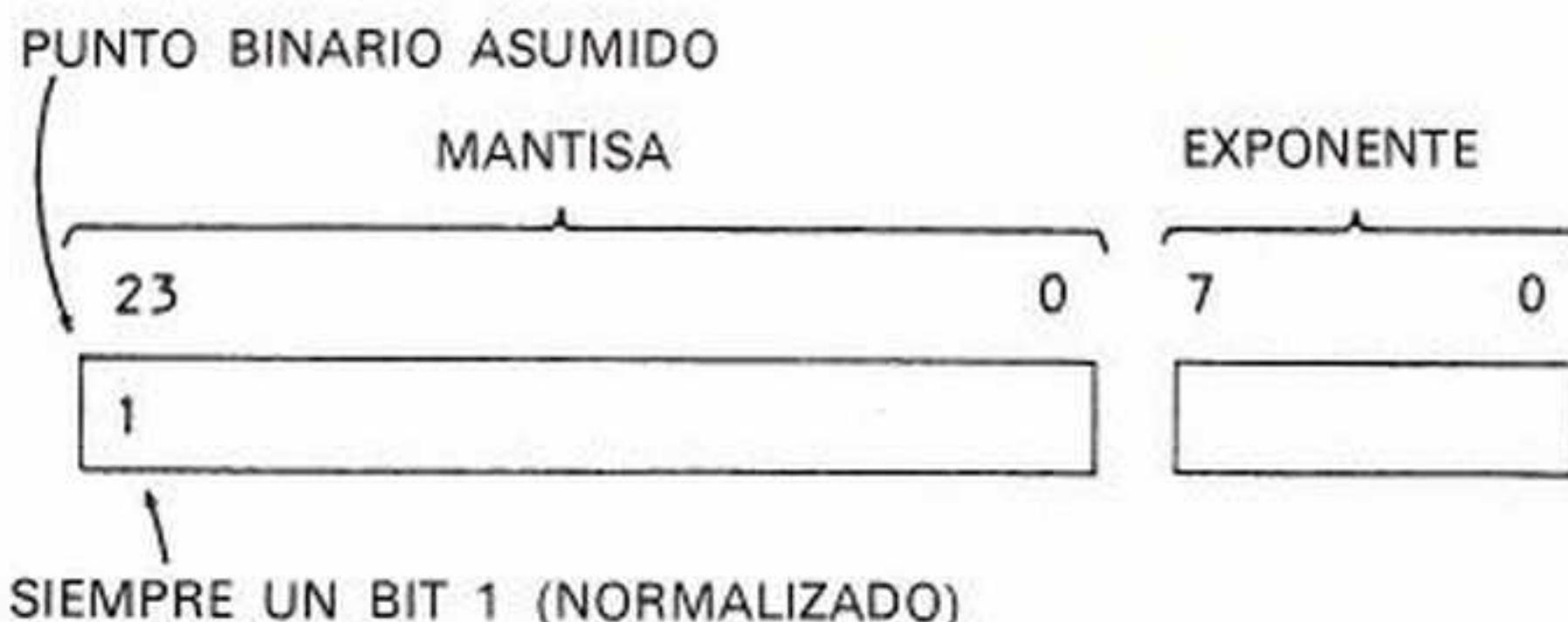


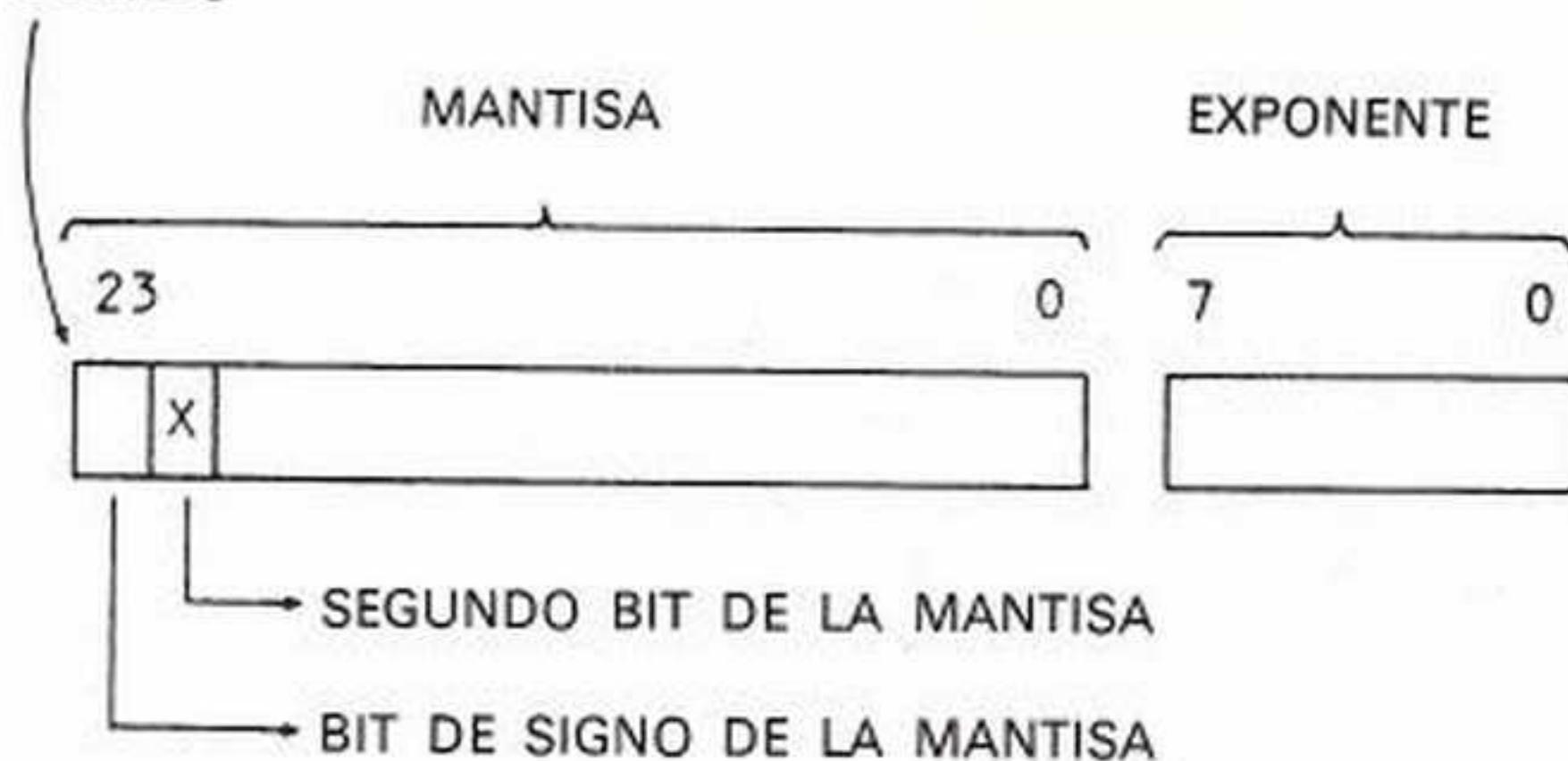
Figura 10.4. Segundo esbozo de formato en punto flotante

Veamos. Tenemos ahora un byte para el exponente y una mantisa normalizada de tres bytes. ¿Qué se nos ha olvidado? Resulta que podemos expresar números positivos, pero no números con signo. ¡El bit de signo ha desaparecido! Definitivamente, necesitamos un bit de signo, a menos que prefiramos que nuestros usuarios del BASIC se limiten exclusivamente a los números positivos...

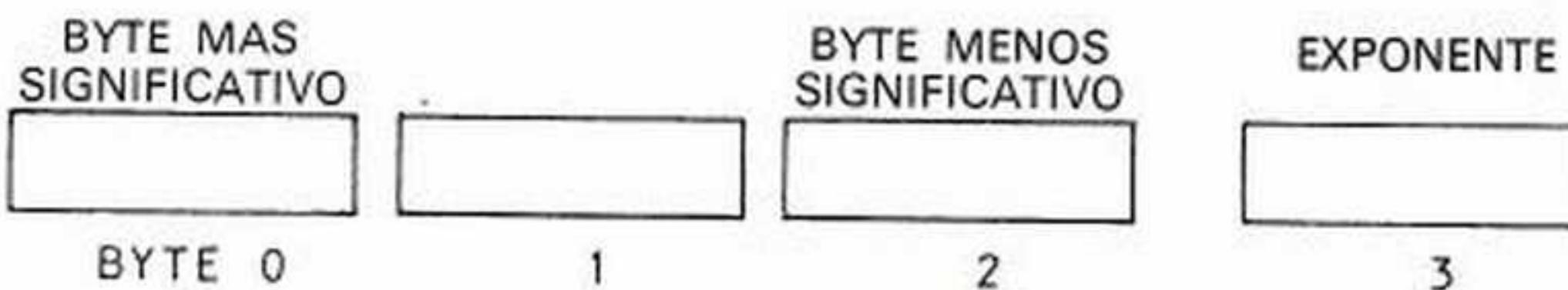
Una solución es ésta: como todo número en punto flotante va a ser forzosamente normalizado, ¿por qué no dejar que el primer dígito de la mantisa represente el bit de signo? El resto del número se almacenará en complemento a dos. Nos limitaremos a despreciar el primer 1 cuando sepamos lo que es. El formato final de un cálculo en punto flotante tiene la forma que muestra la figura 10.5.

Se observará en la figura 10.5 que hay dos métodos para almacenar números en punto flotante. Esto se debe a la distinta manera en que los microprocesadores almacenan valores de 16 bits. Si los datos se almacenan en la memoria en dos almacenes de 16 bits (microprocesador Z-80), el formato presenta primero el byte menos significativo, seguido del más significativo. Esto quiere decir que el número en punto flotante se almacena de la manera siguiente: byte menos significativo de la mantisa, el siguiente byte más significativo de la mantisa, byte más significativo de la mantisa y byte del exponente.

PUNTO BINARIO
ASUMIDO



PRIMER ESQUEMA DE MEMORIA DE ALMACENAJE



SEGUNDO ESQUEMA DE MEMORIA DE ALMACENAJE

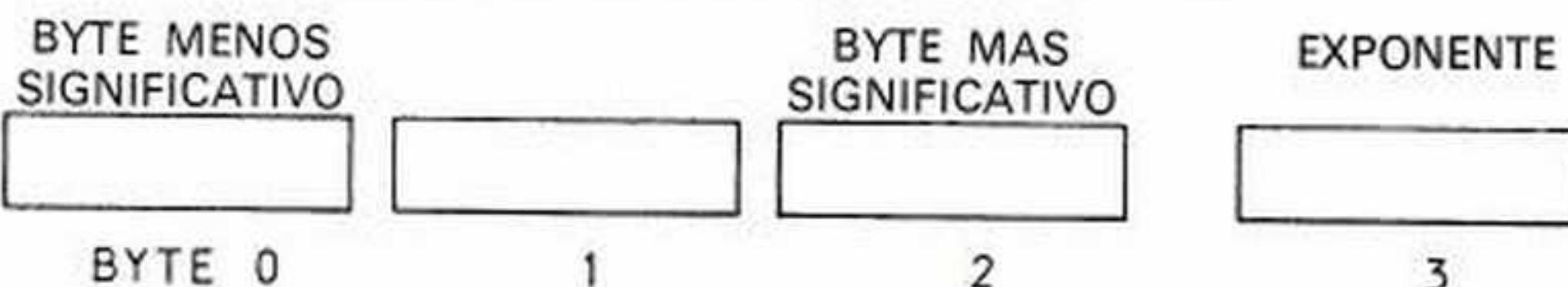
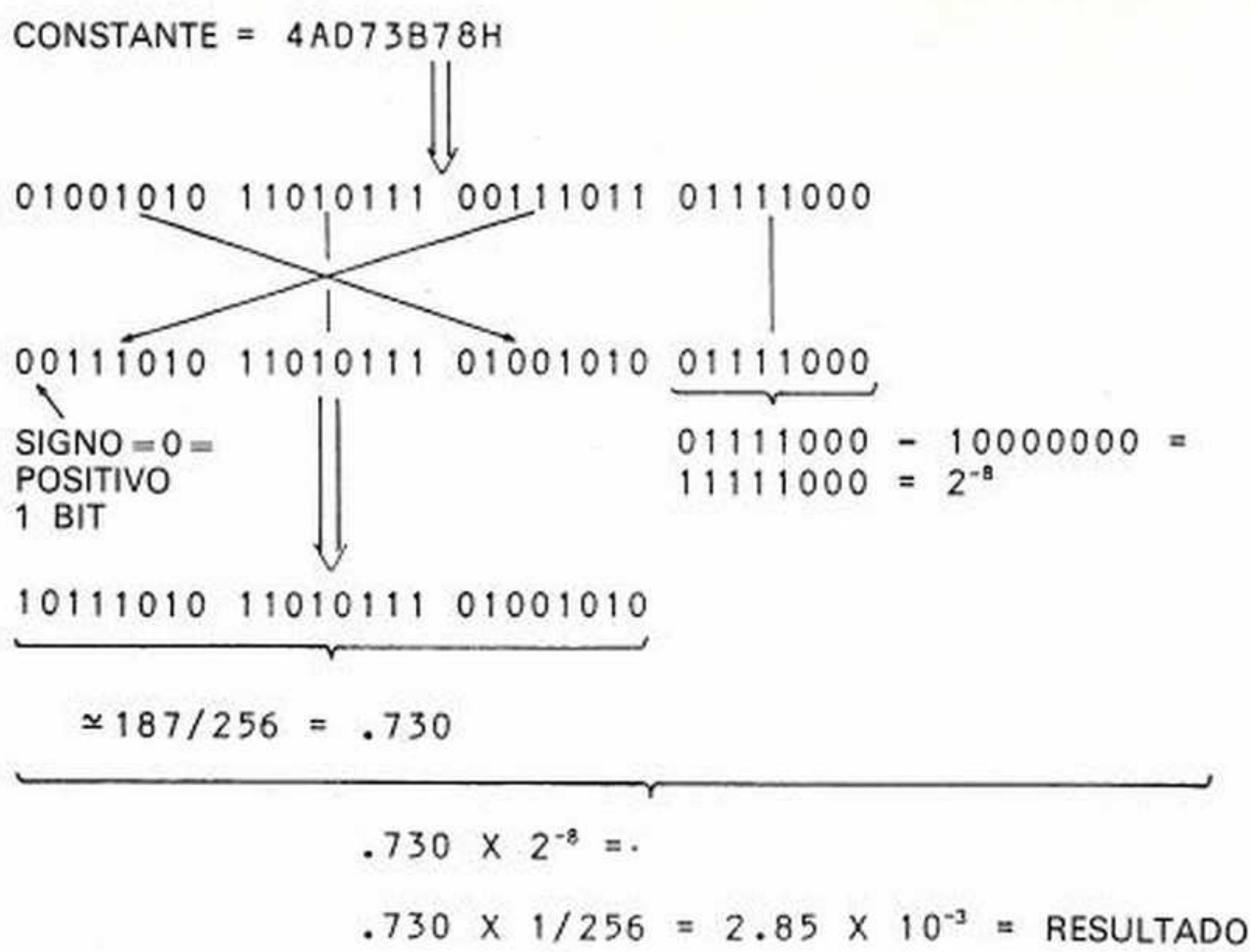


Figura 10.5. Versión final de formato en punto flotante

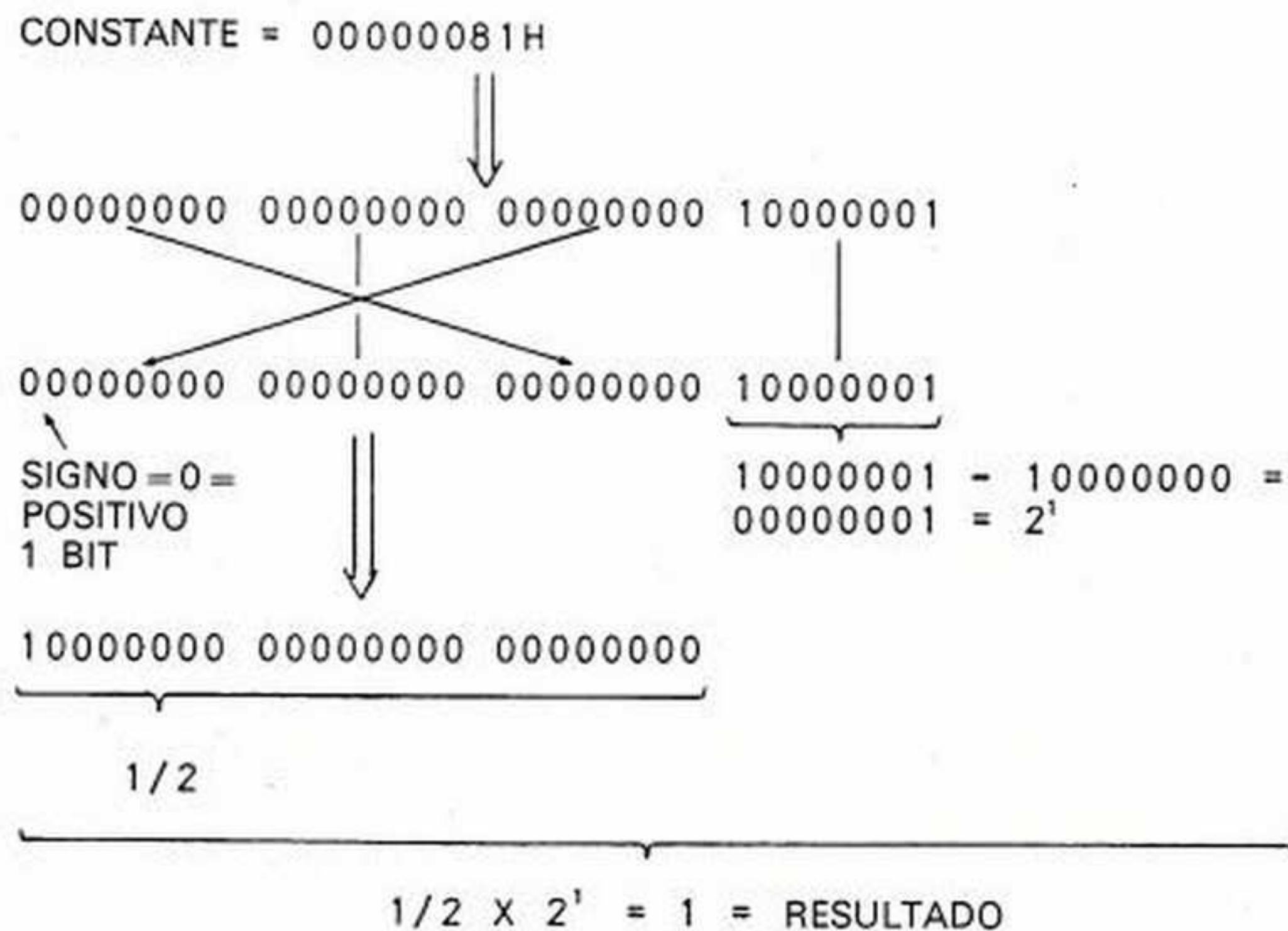
La figura 10.6 muestra el ejemplo —tres ejemplos— de constantes almacenadas en la memoria. Es conveniente estudiarlas para aprender mejor cómo se descifran formatos en punto flotante.

Números en punto flotante de doble precisión

El rango de los exponentes es, probablemente, más que adecuado para la mayoría de los procesos. Pocas cantidades son mayores que 10^{39} . El número de dígitos de precisión, sin embargo, debería incrementarse si el coste de almacenaje no es muy elevado. Se añade otro dígito decimal por cada $3\frac{1}{2}$ bits, aproximadamente (3 bits escalados por 8 y

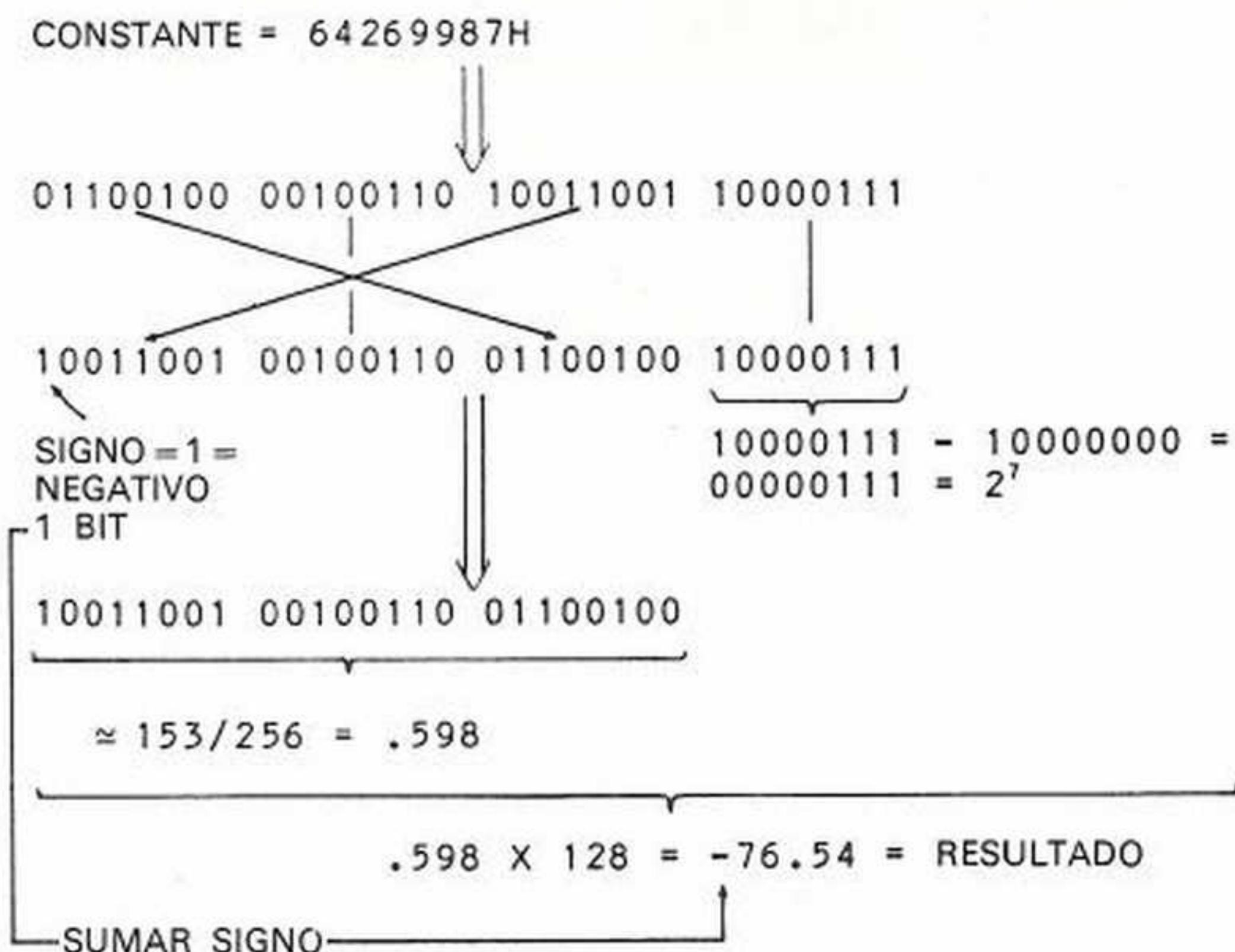


A) Primer ejemplo.



B) Segundo ejemplo.

Figura 10.6. Constantes en punto flotante de la memoria



C) Tercer ejemplo.

Fig. 10.6 (cont.). Constantes en punto flotante de la memoria.

4 escalados por 16). Por consiguiente, por cada dos bytes sumados a la mantisa, se añaden alrededor de 5 dígitos decimales.

Un formato de punto flotante en doble precisión que se halla en algunas versiones de BASIC añade cuatro bytes más a la mantisa, para obtener una precisión total de unos 17 dígitos decimales, al tiempo que mantiene el mismo rango de números. Este esquema se muestra en la figura 10.7.

Cálculos en los que se emplean números binarios en punto flotante

Las operaciones en las que se utilizan números binarios en punto flotante se asemejan a las operaciones en notación científica. Se pueden multiplicar o dividir dos números normalizados en punto flotante efectuando una operación de multiplicar enteros en la mantisa, sumando o restando los exponentes.

Las sumas o restas de dos números en punto flotante han de efectuarse igualando los exponentes. Como el formato de la mantisa sólo ad-

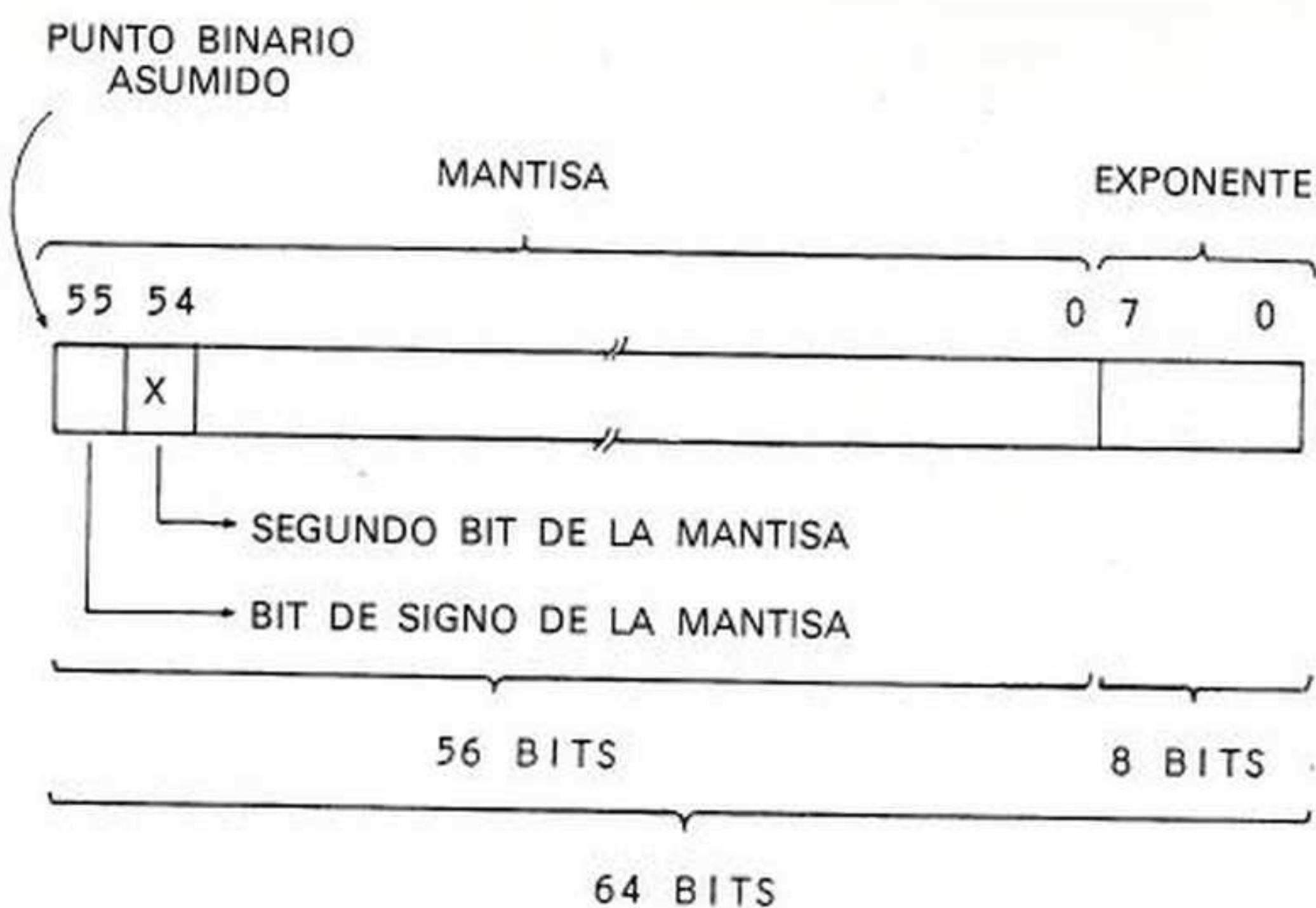


Figura 10.7. Formato de punto flotante en doble precisión

mite fracciones de valor menor de 1, el número del menor exponente se ajusta desplazando la mantisa a la derecha y sumando uno al valor del exponente por cada desplazamiento, hasta que se igualen.

Los algoritmos de operaciones en punto flotante son bastante complicados, y el código actual de operaciones en punto flotante constituye una parte considerable del intérprete BASIC. Aunque aquí no podemos entrar en detalles, esperamos que la materia de los dos capítulos anteriores haya proporcionado algún conocimiento de los principios básicos del manejo de fracciones, números mixtos y operaciones en punto flotante.

Ejercicios

- Pase los números siguientes a notación científica:

$$3.141600; \quad 93,000,000; \quad -186,000; \quad 0.0000135$$

- Efectúe las siguientes operaciones mediante notación científica:

$$3.14 \times .000152 \times 200,000 = ?$$

$$3.100,000 / .000015 = ?$$

3. Expresar estas potencias de dos como valores exponenciales de 8 bits con código exceso a 128:

$$2^{\uparrow 0}, \quad 2^{\uparrow 1}, \quad 2^{\uparrow -5}, \quad 2^{\uparrow 35}, \quad 2^{\uparrow -40}, \quad 2^{\uparrow 128}, \quad 2^{\uparrow -129}$$

4. Pasar estos números en punto flotante a valores decimales. La mantisa va desde el byte más significativo al menos significativo, de izquierda a derecha. El exponente es el byte del extremo de la derecha.

$$00010000 \quad 00000000 \quad 00000000 \quad 10000111 = ?$$

$$10010011 \quad 10000000 \quad 00000000 \quad 10000111 = ?$$

Apéndice A

Soluciones de los ejercicios

CAPITULO 1

1. 10100, 10101, 10110, 10111, 11000, 11001, 11010, 11011, 11100, 11101, 11110, 11111, 100000.
2. 53, 16, 85, 240, 14185.
3. 1111, 11010, 110100, 01101001, 11111111, 1110101001100000.
4. 00000101, 00110101, 00010101.
5. 15, 63, 255, 65535. $(2 \uparrow n) - 1$.

CAPITULO 2

1. $9 \times 16 \uparrow 2 + 14 \times 16 \uparrow 1 + 2 \times 16 \uparrow 0$.
2. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14.
3. 5, AH, AAH, 4FH, B63AH.
4. 101011100011, 100110011001, 1111001000110010.
5. 227, 82, 43690.
6. DH, FH, 1CH, 3E8H.
7. FFFFH.
8. 73, 219.
9. 7, 161, 310.
10. Es imposible.
11. 321.

CAPITULO 3

1. $1 + 3 = 4$ (100), $7 + 15 = 22$ (10110), $21 + 42 = 63$ (111111).
2. $2 - 1 = 1$ (1), $7 - 5 = 2$ (10), $12 - 1 = 11$ (1011).
3. No es necesario 111, es necesario -86 , es necesario -128 .
4. 11111111, 11111110, 11111101, 11100010, 00000101, 01111111.
5. 0000000011111111 ($+127$, $+127$), 1111111100000000 (-128 , -128), 111111110101010 (-85, -85).
6. 111111011010100 (-300) + 111111111111011 (-5) = 111111011001111 (-305).
7. 111111011010100 (-300) - 111111111111011 (-5) = 111111011011001 (-295).

CAPITULO 4

1. $+127 + 1 = +128$ (10000000, desbordamiento), $+127 - 1 = +126$ (01111110, no hay desbordamiento), $-81 + (-1) = -82$ (10101110, no hay desbordamiento).
2. Resultado = 10000000 (desbordamiento, pero no acarreo), resultado = 0000000 (acarreo, pero no desbordamiento).
3. Resultado = 00000000 ($Z = 1$, $S = 0$), resultado = 00110101 ($Z = 0$, $S = 0$).

CAPITULO 5

1. 10101111, 11110111.
2. 10100101, 11010111.
3. XXXYYXXX y 00011000 = 000YY000.
4. 11100001 (-31), 1011 (-5), 01010110 ($+86$).
5. 01011110, 00000001.
6. 10010111, 01000000.
7. C = 0 01011111, C = 1 00000000.
8. 00010111 C = 1, 11000000 C = 0.
9. 00111111 C = 1 (antes = $+127$ después = $+63$), 00101101 C = 0 (antes = $+90$ después = $+45$), 01000010 C = 1 (antes = -123 después = $+66$), 01000000 C = 0 (antes = -128 después = $+64$).
10. 11111110 C = 0 (antes = $+127$ después = -2), 10110100 C = 0 (antes = $+90$ después = -76), 00001010 C = 1 (antes = -123 después = $+10$), 00000000 C = 1 (antes = -128 después = 0).
11. 00111111 (antes = $+127$ después = $+63$), 11000010 (antes = -123 después = -62), 11000000 (antes = -128 después = -64).

CAPITULO 6

1. 1111110000000001 ($255 \times 255 = 65025$).
2. 11111111111111 , 11111111111111 ,
3. 1 = negativo, 0 = positivo haciendo O-exclusivo con los signos de los operandos.

CAPITULO 7

1. 1111111 1111111 1111111 o 16,777,215.
2. 00100010 00000000 01011010 01000000.
3. 00000000 11111110 11011001 11111111.
4. 11000000 01010111 01000100 10000001.
5. 00011111 11010101 11011101 10111111 C = 1.

CAPITULO 8

1. .71875, .5, .9375, .00012207..
2. .01011101..., .0101, .1100011...
3. 46.6875, -73.0625.
4. 0110010000000000, 1001110100000000.
5. 11.5, 2642.25.

CAPITULO 9

1. +012391, -098, -01.52, +.768.
2. 2B 39 35 38 2E 32, 2D 31 30 31 31 2E 35 39.

CAPITULO 10

1. $3.1416 \times 10^{+0}$, $9.3 \times 10^{+7}$, $-1.86 \times 10^{+5}$, 1.35×10^{-5} .
2. $3.14 \times 1.52 \times 10^{-4} \times 2.0 \times 10^{+5} = 9.5 \times 10^{+1}$
 $3.1 \times 10^{+6} / 1.5 \times 10^{-5} = 2.0 \times 10^{+11}$.
3. 10000000, 10000001, 01111011, 10100011, 01011000, impossible, impossible.
4. 72.0, -73.75.

Apéndice B

Conversiones binario, octal, decimal y hexadecimal

| Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. |
|------------|-------|------|------|------------|-------|------|------|------------|-------|------|------|
| 0000000000 | 0000 | 0 | 000 | 0000011010 | 0032 | 26 | 01A | 0000110100 | 0064 | 52 | 034 |
| 0000000001 | 0001 | 1 | 001 | 0000011011 | 0033 | 27 | 01B | 0000110101 | 0065 | 53 | 035 |
| 0000000010 | 0002 | 2 | 002 | 0000011100 | 0034 | 28 | 01C | 0000110110 | 0066 | 54 | 036 |
| 0000000011 | 0003 | 3 | 003 | 0000011101 | 0035 | 29 | 01D | 0000110111 | 0067 | 55 | 037 |
| 0000000100 | 0004 | 4 | 004 | 0000011110 | 0036 | 30 | 01E | 0000111000 | 0070 | 56 | 038 |
| 0000000101 | 0005 | 5 | 005 | 0000011111 | 0037 | 31 | 01F | 0000111001 | 0071 | 57 | 039 |
| 0000000110 | 0006 | 6 | 006 | 0000100000 | 0040 | 32 | 020 | 0000111010 | 0072 | 58 | 03A |
| 0000000111 | 0007 | 7 | 007 | 0000100001 | 0041 | 33 | 021 | 0000111011 | 0073 | 59 | 03B |
| 0000001000 | 0010 | 8 | 008 | 0000100010 | 0042 | 34 | 022 | 0000111100 | 0074 | 60 | 03C |
| 0000001001 | 0011 | 9 | 009 | 0000100011 | 0043 | 35 | 023 | 0000111101 | 0075 | 61 | 03D |
| 0000001010 | 0012 | 10 | 00A | 0000100100 | 0044 | 36 | 024 | 0000111110 | 0076 | 62 | 03E |
| 0000001011 | 0013 | 11 | 00B | 0000100101 | 0045 | 37 | 025 | 0000111111 | 0077 | 63 | 03F |
| 0000001100 | 0014 | 12 | 00C | 0000100110 | 0046 | 38 | 026 | 0001000000 | 0100 | 64 | 040 |
| 0000001101 | 0015 | 13 | 00D | 0000100111 | 0047 | 39 | 027 | 0001000001 | 0101 | 65 | 041 |
| 0000001110 | 0016 | 14 | 00E | 0000101000 | 0050 | 40 | 028 | 0001000010 | 0102 | 66 | 042 |
| 0000001111 | 0017 | 15 | 00F | 0000101001 | 0051 | 41 | 029 | 0001000011 | 0103 | 67 | 043 |
| 0000010000 | 0020 | 16 | 010 | 0000101010 | 0052 | 42 | 02A | 0001000100 | 0104 | 68 | 044 |
| 0000010001 | 0021 | 17 | 011 | 0000101011 | 0053 | 43 | 02B | 0001000101 | 0105 | 69 | 045 |
| 0000010010 | 0022 | 18 | 012 | 0000101100 | 0054 | 44 | 02C | 0001000110 | 0106 | 70 | 046 |
| 0000010011 | 0023 | 19 | 013 | 0000101101 | 0055 | 45 | 02D | 0001000111 | 0107 | 71 | 047 |
| 0000010100 | 0024 | 20 | 014 | 0000101110 | 0056 | 46 | 02E | 0001001000 | 0110 | 72 | 048 |
| 0000010101 | 0025 | 21 | 015 | 0000101111 | 0057 | 47 | 02F | 0001001001 | 0111 | 73 | 049 |
| 0000010110 | 0026 | 22 | 016 | 0000110000 | 0060 | 48 | 030 | 0001001010 | 0112 | 74 | 04A |
| 0000010111 | 0027 | 23 | 017 | 0000110001 | 0061 | 49 | 031 | 0001001011 | 0113 | 75 | 04B |
| 0000011000 | 0030 | 24 | 018 | 0000110010 | 0062 | 50 | 032 | 0001001100 | 0114 | 76 | 04C |
| 0000011001 | 0031 | 25 | 019 | 0000110011 | 0063 | 51 | 033 | 0001001101 | 0115 | 77 | 04D |

| Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. |
|------------|-------|------|------|------------|-------|------|------|------------|-------|------|------|
| 0001001110 | 0116 | 78 | 04E | 0010000110 | 0206 | 134 | 086 | 0010111110 | 0276 | 190 | 0BE |
| 0001001111 | 0117 | 79 | 04F | 0010000111 | 0207 | 135 | 087 | 0010111111 | 0277 | 191 | 0BF |
| 0001010000 | 0120 | 80 | 050 | 0010001000 | 0210 | 136 | 088 | 0011000000 | 0300 | 192 | 0C0 |
| 0001010001 | 0121 | 81 | 051 | 0010001001 | 0211 | 137 | 089 | 0011000001 | 0301 | 193 | 0C1 |
| 0001010010 | 0122 | 82 | 052 | 0010001010 | 0212 | 138 | 08A | 0011000010 | 0302 | 194 | 0C2 |
| 0001010011 | 0123 | 83 | 053 | 0010001011 | 0213 | 139 | 08B | 0011000011 | 0303 | 195 | 0C3 |
| 0001010100 | 0124 | 84 | 054 | 0010001100 | 0214 | 140 | 08C | 0011000100 | 0304 | 196 | 0C4 |
| 0001010101 | 0125 | 85 | 055 | 0010001101 | 0215 | 141 | 08D | 0011000101 | 0305 | 197 | 0C5 |
| 0001010110 | 0126 | 86 | 056 | 0010001110 | 0216 | 142 | 08E | 0011000110 | 0306 | 198 | 0C6 |
| 0001010111 | 0127 | 87 | 057 | 0010001111 | 0217 | 143 | 08F | 0011000111 | 0307 | 199 | 0C7 |
| 0001011000 | 0130 | 88 | 058 | 0010010000 | 0220 | 144 | 090 | 0011001000 | 0310 | 200 | 0C8 |
| 0001011001 | 0131 | 89 | 059 | 0010010001 | 0221 | 145 | 091 | 0011001001 | 0311 | 201 | 0C9 |
| 0001011010 | 0132 | 90 | 05A | 0010010010 | 0222 | 146 | 092 | 0011001010 | 0312 | 202 | 0CA |
| 0001011011 | 0133 | 91 | 05B | 0010010011 | 0223 | 147 | 093 | 0011001011 | 0313 | 203 | 0CB |
| 0001011100 | 0134 | 92 | 05C | 0010010100 | 0224 | 148 | 094 | 0011001100 | 0314 | 204 | 0CC |
| 0001011101 | 0135 | 93 | 05D | 0010010101 | 0225 | 149 | 095 | 0011001101 | 0315 | 205 | 0CD |
| 0001011110 | 0136 | 94 | 05E | 0010010110 | 0226 | 150 | 096 | 0011001110 | 0316 | 206 | 0CE |
| 0001011111 | 0137 | 95 | 05F | 0010010111 | 0227 | 151 | 097 | 0011001111 | 0317 | 207 | 0CF |
| 0001100000 | 0140 | 96 | 060 | 0010011000 | 0230 | 152 | 098 | 0011010000 | 0320 | 208 | 0D0 |
| 0001100001 | 0141 | 97 | 061 | 0010011001 | 0231 | 153 | 099 | 0011010001 | 0321 | 209 | 0D1 |
| 0001100010 | 0142 | 98 | 062 | 0010011010 | 0232 | 154 | 09A | 0011010010 | 0322 | 210 | 0D2 |
| 0001100011 | 0143 | 99 | 063 | 0010011011 | 0233 | 155 | 09B | 0011010011 | 0323 | 211 | 0D3 |
| 0001100100 | 0144 | 100 | 064 | 0010011100 | 0234 | 156 | 09C | 0011010100 | 0324 | 212 | 0D4 |
| 0001100101 | 0145 | 101 | 065 | 0010011101 | 0235 | 157 | 09D | 0011010101 | 0325 | 213 | 0D5 |
| 0001100110 | 0146 | 102 | 066 | 0010011110 | 0236 | 158 | 09E | 0011010110 | 0326 | 214 | 0D6 |
| 0001100111 | 0147 | 103 | 067 | 0010011111 | 0237 | 159 | 09F | 0011010111 | 0327 | 215 | 0D7 |
| 0001101000 | 0150 | 104 | 068 | 0010100000 | 0240 | 160 | 0A0 | 0011011000 | 0330 | 216 | 0D8 |
| 0001101001 | 0151 | 105 | 069 | 0010100001 | 0241 | 161 | 0A1 | 0011011001 | 0331 | 217 | 0D9 |
| 0001101010 | 0152 | 106 | 06A | 0010100010 | 0242 | 162 | 0A2 | 0011011010 | 0332 | 218 | 0DA |
| 0001101011 | 0153 | 107 | 06B | 0010100011 | 0243 | 163 | 0A3 | 0011011011 | 0333 | 219 | 0DB |
| 0001101100 | 0154 | 108 | 06C | 0010100100 | 0244 | 164 | 0A4 | 0011011100 | 0334 | 220 | 0DC |
| 0001101101 | 0155 | 109 | 06D | 0010100101 | 0245 | 165 | 0A5 | 0011011101 | 0335 | 221 | 0DD |
| 0001101110 | 0156 | 110 | 06E | 0010100110 | 0246 | 166 | 0A6 | 0011011110 | 0336 | 222 | 0DE |
| 0001101111 | 0157 | 111 | 06F | 0010100111 | 0247 | 167 | 0A7 | 0011011111 | 0337 | 223 | 0DF |
| 0001110000 | 0160 | 112 | 070 | 0010101000 | 0250 | 168 | 0A8 | 0011100000 | 0340 | 224 | 0E0 |
| 0001110001 | 0161 | 113 | 071 | 0010101001 | 0251 | 169 | 0A9 | 0011100001 | 0341 | 225 | 0E1 |
| 0001110010 | 0162 | 114 | 072 | 0010101010 | 0252 | 170 | 0AA | 0011100010 | 0342 | 226 | 0E2 |
| 0001110011 | 0163 | 115 | 073 | 0010101011 | 0253 | 171 | 0AB | 0011100011 | 0343 | 227 | 0E3 |
| 0001110100 | 0164 | 116 | 074 | 0010101100 | 0254 | 172 | 0AC | 0011100100 | 0344 | 228 | 0E4 |
| 0001110101 | 0165 | 117 | 075 | 0010101101 | 0255 | 173 | 0AD | 0011100101 | 0345 | 229 | 0E5 |
| 0001110110 | 0166 | 118 | 076 | 0010101110 | 0256 | 174 | 0AE | 0011100110 | 0346 | 230 | 0E6 |
| 0001110111 | 0167 | 119 | 077 | 0010101111 | 0257 | 175 | 0AF | 0011100111 | 0347 | 231 | 0E7 |
| 0001111000 | 0170 | 120 | 078 | 0010110000 | 0260 | 176 | 0B0 | 0011101000 | 0350 | 232 | 0E8 |
| 0001111001 | 0171 | 121 | 079 | 0010110001 | 0261 | 177 | 0B1 | 0011101001 | 0351 | 233 | 0E9 |
| 0001111010 | 0172 | 122 | 07A | 0010110010 | 0262 | 178 | 0B2 | 0011101010 | 0352 | 234 | 0EA |
| 0001111011 | 0173 | 123 | 07B | 0010110011 | 0263 | 179 | 0B3 | 0011101011 | 0353 | 235 | 0EB |
| 0001111100 | 0174 | 124 | 07C | 0010110100 | 0264 | 180 | 0B4 | 0011101100 | 0354 | 236 | 0EC |
| 0001111101 | 0175 | 125 | 07D | 0010110101 | 0265 | 181 | 0B5 | 0011101101 | 0355 | 237 | 0ED |
| 0001111110 | 0176 | 126 | 07E | 0010110110 | 0266 | 182 | 0B6 | 0011101110 | 0356 | 238 | 0EE |
| 0001111111 | 0177 | 127 | 07F | 0010110111 | 0267 | 183 | 0B7 | 0011101111 | 0357 | 239 | 0EF |
| 0010000000 | 0200 | 128 | 080 | 0010111000 | 0270 | 184 | 0B8 | 0011110000 | 0360 | 240 | 0F0 |
| 0010000001 | 0201 | 129 | 081 | 0010111001 | 0271 | 185 | 0B9 | 0011110001 | 0361 | 241 | 0F1 |
| 0010000010 | 0202 | 130 | 082 | 0010111010 | 0272 | 186 | 0BA | 0011110010 | 0362 | 242 | 0F2 |
| 0010000011 | 0203 | 131 | 083 | 0010111011 | 0273 | 187 | 0BB | 0011110011 | 0363 | 243 | 0F3 |
| 0010000100 | 0204 | 132 | 084 | 0010111100 | 0274 | 188 | 0BC | 0011110100 | 0364 | 244 | 0F4 |
| 0010000101 | 0205 | 133 | 085 | 0010111101 | 0275 | 189 | 0BD | 0011110101 | 0365 | 245 | 0F5 |

| Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. |
|-------------|-------|------|------|------------|-------|------|------|------------|-------|------|------|
| 0011110110 | 0366 | 246 | 0F6 | 0100101110 | 0456 | 302 | 12E | 0101100110 | 0546 | 358 | 166 |
| 0011110111 | 0367 | 247 | 0F7 | 0100101111 | 0457 | 303 | 12F | 0101100111 | 0547 | 359 | 167 |
| 0011111000 | 0370 | 248 | 0F8 | 0100110000 | 0460 | 304 | 130 | 0101101000 | 0550 | 360 | 168 |
| 0011111001 | 0371 | 249 | 0F9 | 0100110001 | 0461 | 305 | 131 | 0101101001 | 0551 | 361 | 169 |
| 0011111010 | 0372 | 250 | 0FA | 0100110010 | 0462 | 306 | 132 | 0101101010 | 0552 | 362 | 16A |
| 0011111011 | 0373 | 251 | 0FB | 0100110011 | 0463 | 307 | 133 | 0101101011 | 0553 | 363 | 16B |
| 0011111100 | 0374 | 252 | 0FC | 0100110100 | 0464 | 308 | 134 | 0101101100 | 0554 | 364 | 16C |
| 0011111101 | 0375 | 253 | 0FD | 0100110101 | 0465 | 309 | 135 | 0101101101 | 0555 | 365 | 16D |
| 0011111110 | 0376 | 254 | 0FE | 0100110110 | 0466 | 310 | 136 | 0101101110 | 0556 | 366 | 16E |
| 0011111111 | 0377 | 255 | 0FF | 0100110111 | 0467 | 311 | 137 | 0101101111 | 0557 | 367 | 16F |
| 01000000000 | 0400 | 256 | 100 | 0100111000 | 0470 | 312 | 138 | 0101110000 | 0560 | 368 | 170 |
| 01000000001 | 0401 | 257 | 101 | 0100111001 | 0471 | 313 | 139 | 0101110001 | 0561 | 369 | 171 |
| 01000000010 | 0402 | 258 | 102 | 0100111010 | 0472 | 314 | 13A | 0101110010 | 0562 | 370 | 172 |
| 01000000011 | 0403 | 259 | 103 | 0100111011 | 0473 | 315 | 13B | 0101110011 | 0563 | 371 | 173 |
| 01000000100 | 0404 | 260 | 104 | 0100111100 | 0474 | 316 | 13C | 0101110100 | 0564 | 372 | 174 |
| 01000000101 | 0405 | 261 | 105 | 0100111101 | 0475 | 317 | 13D | 0101110101 | 0565 | 373 | 175 |
| 01000000110 | 0406 | 262 | 106 | 0100111110 | 0476 | 318 | 13E | 0101110110 | 0566 | 374 | 176 |
| 01000000111 | 0407 | 263 | 107 | 0100111111 | 0477 | 319 | 13F | 0101110111 | 0567 | 375 | 177 |
| 0100001000 | 0410 | 264 | 108 | 0101000000 | 0500 | 320 | 140 | 0101111000 | 0570 | 376 | 178 |
| 0100001001 | 0411 | 265 | 109 | 0101000001 | 0501 | 321 | 141 | 0101111001 | 0571 | 377 | 179 |
| 0100001010 | 0412 | 266 | 10A | 0101000010 | 0502 | 322 | 142 | 0101111010 | 0572 | 378 | 17A |
| 0100001011 | 0413 | 267 | 10B | 0101000011 | 0503 | 323 | 143 | 0101111011 | 0573 | 379 | 17B |
| 0100001100 | 0414 | 268 | 10C | 0101000100 | 0504 | 324 | 144 | 0101111100 | 0574 | 380 | 17C |
| 0100001101 | 0415 | 269 | 10D | 0101000101 | 0505 | 325 | 145 | 0101111101 | 0575 | 381 | 17D |
| 0100001110 | 0416 | 270 | 10E | 0101000110 | 0506 | 326 | 146 | 0101111110 | 0576 | 382 | 17E |
| 0100001111 | 0417 | 271 | 10F | 0101000111 | 0507 | 327 | 147 | 0101111111 | 0577 | 383 | 17F |
| 0100010000 | 0420 | 272 | 110 | 0101001000 | 0510 | 328 | 148 | 0110000000 | 0600 | 384 | 180 |
| 0100010001 | 0421 | 273 | 111 | 0101001001 | 0511 | 329 | 149 | 0110000001 | 0601 | 385 | 181 |
| 0100010010 | 0422 | 274 | 112 | 0101001010 | 0512 | 330 | 14A | 0110000010 | 0602 | 386 | 182 |
| 0100010011 | 0423 | 275 | 113 | 0101001011 | 0513 | 331 | 14B | 0110000011 | 0603 | 387 | 183 |
| 0100010100 | 0424 | 276 | 114 | 0101001100 | 0514 | 332 | 14C | 0110000100 | 0604 | 388 | 184 |
| 0100010101 | 0425 | 277 | 115 | 0101001101 | 0515 | 333 | 14D | 0110000101 | 0605 | 389 | 185 |
| 0100010110 | 0426 | 278 | 116 | 0101001110 | 0516 | 334 | 14E | 0110000110 | 0606 | 390 | 186 |
| 0100010111 | 0427 | 279 | 117 | 0101001111 | 0517 | 335 | 14F | 0110000111 | 0607 | 391 | 187 |
| 0100011000 | 0430 | 280 | 118 | 0101010000 | 0520 | 336 | 150 | 0110001000 | 0610 | 392 | 188 |
| 0100011001 | 0431 | 281 | 119 | 0101010001 | 0521 | 337 | 151 | 0110001001 | 0611 | 393 | 189 |
| 0100011010 | 0432 | 282 | 11A | 0101010010 | 0522 | 338 | 152 | 0110001010 | 0612 | 394 | 18A |
| 0100011011 | 0433 | 283 | 11B | 0101010011 | 0523 | 339 | 153 | 0110001011 | 0613 | 395 | 18B |
| 0100011100 | 0434 | 284 | 11C | 0101010100 | 0524 | 340 | 154 | 0110001100 | 0614 | 396 | 18C |
| 0100011101 | 0435 | 285 | 11D | 0101010101 | 0525 | 341 | 155 | 0110001101 | 0615 | 397 | 18D |
| 0100011110 | 0436 | 286 | 11E | 0101010110 | 0526 | 342 | 156 | 0110001110 | 0616 | 398 | 18E |
| 0100011111 | 0437 | 287 | 11F | 0101010111 | 0527 | 343 | 157 | 0110001111 | 0617 | 399 | 18F |
| 0100100000 | 0440 | 288 | 120 | 0101011000 | 0530 | 344 | 158 | 0110010000 | 0620 | 400 | 190 |
| 0100100001 | 0441 | 289 | 121 | 0101011001 | 0531 | 345 | 159 | 0110010001 | 0621 | 401 | 191 |
| 0100100010 | 0442 | 290 | 122 | 0101011010 | 0532 | 346 | 15A | 0110010010 | 0622 | 402 | 192 |
| 0100100011 | 0443 | 291 | 123 | 0101011011 | 0533 | 347 | 15B | 0110010011 | 0623 | 403 | 193 |
| 0100100100 | 0444 | 292 | 124 | 0101011100 | 0534 | 348 | 15C | 0110010100 | 0624 | 404 | 194 |
| 0100100101 | 0445 | 293 | 125 | 0101011101 | 0535 | 349 | 15D | 0110010101 | 0625 | 405 | 195 |
| 0100100110 | 0446 | 294 | 126 | 0101011110 | 0536 | 350 | 15E | 0110010110 | 0626 | 406 | 196 |
| 0100100111 | 0447 | 295 | 127 | 0101011111 | 0537 | 351 | 15F | 0110010111 | 0627 | 407 | 197 |
| 0100101000 | 0450 | 296 | 128 | 0101100000 | 0540 | 352 | 160 | 0110011000 | 0630 | 408 | 198 |
| 0100101001 | 0451 | 297 | 129 | 0101100001 | 0541 | 353 | 161 | 0110011001 | 0631 | 409 | 199 |
| 0100101010 | 0452 | 298 | 12A | 0101100010 | 0542 | 354 | 162 | 0110011010 | 0632 | 410 | 19A |
| 0100101011 | 0453 | 299 | 12B | 0101100011 | 0543 | 355 | 163 | 0110011011 | 0633 | 411 | 19B |
| 0100101100 | 0454 | 300 | 12C | 0101100100 | 0544 | 356 | 164 | 0110011100 | 0634 | 412 | 19C |
| 0100101101 | 0455 | 301 | 12D | 0101100101 | 0545 | 357 | 165 | 0110011101 | 0635 | 413 | 19D |

| Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. |
|------------|-------|------|------|------------|-------|------|------|------------|-------|------|------|
| 0110011110 | 0636 | 414 | 19E | 0111010110 | 0726 | 470 | 1D6 | 1000001110 | 1016 | 526 | 20E |
| 0110011111 | 0637 | 415 | 19F | 0111010111 | 0727 | 471 | 1D7 | 1000001111 | 1017 | 527 | 20F |
| 0110100000 | 0640 | 416 | 1A0 | 0111011000 | 0730 | 472 | 1D8 | 1000010000 | 1020 | 528 | 210 |
| 0110100001 | 0641 | 417 | 1A1 | 0111011001 | 0731 | 473 | 1D9 | 1000010001 | 1021 | 529 | 211 |
| 0110100010 | 0642 | 418 | 1A2 | 0111011010 | 0732 | 474 | 1DA | 1000010010 | 1022 | 530 | 212 |
| 0110100011 | 0643 | 419 | 1A3 | 0111011011 | 0733 | 475 | 1DB | 1000010011 | 1023 | 531 | 213 |
| 0110100100 | 0644 | 420 | 1A4 | 0111011100 | 0734 | 476 | 1DC | 1000010100 | 1024 | 532 | 214 |
| 0110100101 | 0645 | 421 | 1A5 | 0111011101 | 0735 | 477 | 1DD | 1000010101 | 1025 | 533 | 215 |
| 0110100110 | 0646 | 422 | 1A6 | 0111011110 | 0736 | 478 | 1DE | 1000010110 | 1026 | 534 | 216 |
| 0110100111 | 0647 | 423 | 1A7 | 0111011111 | 0737 | 479 | 1DF | 1000010111 | 1027 | 535 | 217 |
| 0110101000 | 0650 | 424 | 1A8 | 0111100000 | 0740 | 480 | 1E0 | 1000011000 | 1030 | 536 | 218 |
| 0110101001 | 0651 | 425 | 1A9 | 0111100001 | 0741 | 481 | 1E1 | 1000011001 | 1031 | 537 | 219 |
| 0110101010 | 0652 | 426 | 1AA | 0111100010 | 0742 | 482 | 1E2 | 1000011010 | 1032 | 538 | 21A |
| 0110101011 | 0653 | 427 | 1AB | 0111100011 | 0743 | 483 | 1E3 | 1000011011 | 1033 | 539 | 21B |
| 0110101100 | 0654 | 428 | 1AC | 0111100100 | 0744 | 484 | 1E4 | 1000011100 | 1034 | 540 | 21C |
| 0110101101 | 0655 | 429 | 1AD | 0111100101 | 0745 | 485 | 1E5 | 1000011101 | 1035 | 541 | 21D |
| 0110101110 | 0656 | 430 | 1AE | 0111100110 | 0746 | 486 | 1E6 | 1000011110 | 1036 | 542 | 21E |
| 0110101111 | 0657 | 431 | 1AF | 0111100111 | 0747 | 487 | 1E7 | 1000011111 | 1037 | 543 | 21F |
| 0110110000 | 0660 | 432 | 1B0 | 0111101000 | 0750 | 488 | 1E8 | 1000100000 | 1040 | 544 | 220 |
| 0110110001 | 0661 | 433 | 1B1 | 0111101001 | 0751 | 489 | 1E9 | 1000100001 | 1041 | 545 | 221 |
| 0110110010 | 0662 | 434 | 1B2 | 0111101010 | 0752 | 490 | 1EA | 1000100010 | 1042 | 546 | 222 |
| 0110110011 | 0663 | 435 | 1B3 | 0111101011 | 0753 | 491 | 1EB | 1000100011 | 1043 | 547 | 223 |
| 0110110100 | 0664 | 436 | 1B4 | 0111101100 | 0754 | 492 | 1EC | 1000100100 | 1044 | 548 | 224 |
| 0110110101 | 0665 | 437 | 1B5 | 0111101101 | 0755 | 493 | 1ED | 1000100101 | 1045 | 549 | 225 |
| 0110110110 | 0666 | 438 | 1B6 | 0111101110 | 0756 | 494 | 1EE | 1000100110 | 1046 | 550 | 226 |
| 0110110111 | 0667 | 439 | 1B7 | 0111101111 | 0757 | 495 | 1EF | 1000100111 | 1047 | 551 | 227 |
| 0110111000 | 0670 | 440 | 1B8 | 0111110000 | 0760 | 496 | 1F0 | 1000101000 | 1050 | 552 | 228 |
| 0110111001 | 0671 | 441 | 1B9 | 0111110001 | 0761 | 497 | 1F1 | 1000101001 | 1051 | 553 | 229 |
| 0110111010 | 0672 | 442 | 1BA | 0111110010 | 0762 | 498 | 1F2 | 1000101010 | 1052 | 554 | 22A |
| 0110111011 | 0673 | 443 | 1BB | 0111110011 | 0763 | 499 | 1F3 | 1000101011 | 1053 | 555 | 22B |
| 0110111100 | 0674 | 444 | 1BC | 0111110100 | 0764 | 500 | 1F4 | 1000101100 | 1054 | 556 | 22C |
| 0110111101 | 0675 | 445 | 1BD | 0111110101 | 0765 | 501 | 1F5 | 1000101101 | 1055 | 557 | 22D |
| 0110111110 | 0676 | 446 | 1BE | 0111110110 | 0766 | 502 | 1F6 | 1000101110 | 1056 | 558 | 22E |
| 0110111111 | 0677 | 447 | 1BF | 0111110111 | 0767 | 503 | 1F7 | 1000101111 | 1057 | 559 | 22F |
| 0111000000 | 0700 | 448 | 1C0 | 0111111000 | 0770 | 504 | 1F8 | 1000110000 | 1060 | 560 | 230 |
| 0111000001 | 0701 | 449 | 1C1 | 0111111001 | 0771 | 505 | 1F9 | 1000110001 | 1061 | 561 | 231 |
| 0111000010 | 0702 | 450 | 1C2 | 0111111010 | 0772 | 506 | 1FA | 1000110010 | 1062 | 562 | 232 |
| 0111000011 | 0703 | 451 | 1C3 | 0111111011 | 0773 | 507 | 1FB | 1000110011 | 1063 | 563 | 233 |
| 0111000100 | 0704 | 452 | 1C4 | 0111111100 | 0774 | 508 | 1FC | 1000110100 | 1064 | 564 | 234 |
| 0111000101 | 0705 | 453 | 1C5 | 0111111101 | 0775 | 509 | 1FD | 1000110101 | 1065 | 565 | 235 |
| 0111000110 | 0706 | 454 | 1C6 | 0111111110 | 0776 | 510 | 1FE | 1000110110 | 1066 | 566 | 236 |
| 0111000111 | 0707 | 455 | 1C7 | 0111111111 | 0777 | 511 | 1FF | 1000110111 | 1067 | 567 | 237 |
| 0111001000 | 0710 | 456 | 1C8 | 1000000000 | 1000 | 512 | 200 | 1000111000 | 1070 | 568 | 238 |
| 0111001001 | 0711 | 457 | 1C9 | 1000000001 | 1001 | 513 | 201 | 1000111001 | 1071 | 569 | 239 |
| 0111001010 | 0712 | 458 | 1CA | 1000000010 | 1002 | 514 | 202 | 1000111010 | 1072 | 570 | 23A |
| 0111001011 | 0713 | 459 | 1CB | 1000000011 | 1003 | 515 | 203 | 1000111011 | 1073 | 571 | 23B |
| 0111001100 | 0714 | 460 | 1CC | 1000000100 | 1004 | 516 | 204 | 1000111100 | 1074 | 572 | 23C |
| 0111001101 | 0715 | 461 | 1CD | 1000000101 | 1005 | 517 | 205 | 1000111101 | 1075 | 573 | 23D |
| 0111001110 | 0716 | 462 | 1CE | 1000000110 | 1006 | 518 | 206 | 1000111110 | 1076 | 574 | 23E |
| 0111001111 | 0717 | 463 | 1CF | 1000000111 | 1007 | 519 | 207 | 1000111111 | 1077 | 575 | 23F |
| 0111010000 | 0720 | 464 | 1D0 | 1000001000 | 1010 | 520 | 208 | 1001000000 | 1100 | 576 | 240 |
| 0111010001 | 0721 | 465 | 1D1 | 1000001001 | 1011 | 521 | 209 | 1001000001 | 1101 | 577 | 241 |
| 0111010010 | 0722 | 466 | 1D2 | 1000001010 | 1012 | 522 | 20A | 1001000010 | 1102 | 578 | 242 |
| 0111010011 | 0723 | 467 | 1D3 | 1000001011 | 1013 | 523 | 20B | 1001000011 | 1103 | 579 | 243 |
| 0111010100 | 0724 | 468 | 1D4 | 1000001100 | 1014 | 524 | 20C | 1001000100 | 1104 | 580 | 244 |
| 0111010101 | 0725 | 469 | 1D6 | 1000001101 | 1015 | 525 | 20D | 1001000101 | 1105 | 581 | 245 |

| Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. |
|------------|-------|------|------|------------|-------|------|------|------------|-------|------|------|
| 1001000110 | 1106 | 582 | 246 | 1001111110 | 1176 | 638 | 27E | 1010110110 | 1266 | 694 | 2B6 |
| 1001000111 | 1107 | 583 | 247 | 1001111111 | 1177 | 639 | 27F | 1010110111 | 1267 | 695 | 2B7 |
| 1001001000 | 1110 | 584 | 248 | 1010000000 | 1200 | 640 | 280 | 1010111000 | 1270 | 696 | 2B8 |
| 1001001001 | 1111 | 585 | 249 | 1010000001 | 1201 | 641 | 281 | 1010111001 | 1271 | 697 | 2B9 |
| 1001001010 | 1112 | 586 | 24A | 1010000010 | 1202 | 642 | 282 | 1010111010 | 1272 | 698 | 2BA |
| 1001001011 | 1113 | 587 | 24B | 1010000011 | 1203 | 643 | 283 | 1010111011 | 1273 | 699 | 2BB |
| 1001001100 | 1114 | 588 | 24C | 1010000100 | 1204 | 644 | 284 | 1010111100 | 1274 | 700 | 2BC |
| 1001001101 | 1115 | 589 | 24D | 1010000101 | 1205 | 645 | 285 | 1010111101 | 1275 | 701 | 2BD |
| 1001001110 | 1116 | 590 | 24E | 1010000110 | 1206 | 646 | 286 | 1010111110 | 1276 | 702 | 2BE |
| 1001001111 | 1117 | 591 | 24F | 1010000111 | 1207 | 647 | 287 | 1010111111 | 1277 | 703 | 2BF |
| 1001010000 | 1120 | 592 | 250 | 1010001000 | 1210 | 648 | 288 | 1011000000 | 1300 | 704 | 2C0 |
| 1001010001 | 1121 | 593 | 251 | 1010001001 | 1211 | 649 | 289 | 1011000001 | 1301 | 705 | 2C1 |
| 1001010010 | 1122 | 594 | 252 | 1010001010 | 1212 | 650 | 28A | 1011000010 | 1302 | 706 | 2C2 |
| 1001010011 | 1123 | 595 | 253 | 1010001011 | 1213 | 651 | 28B | 1011000011 | 1303 | 707 | 2C3 |
| 1001010100 | 1124 | 596 | 254 | 1010001100 | 1214 | 652 | 28C | 1011000100 | 1304 | 708 | 2C4 |
| 1001010101 | 1125 | 597 | 255 | 1010001101 | 1215 | 653 | 28D | 1011000101 | 1305 | 709 | 2C5 |
| 1001010110 | 1126 | 598 | 256 | 1010001110 | 1216 | 654 | 28E | 1011000110 | 1306 | 710 | 2C6 |
| 1001010111 | 1127 | 599 | 257 | 1010001111 | 1217 | 655 | 28F | 1011000111 | 1307 | 711 | 2C7 |
| 1001011000 | 1130 | 600 | 258 | 1010010000 | 1220 | 656 | 290 | 1011001000 | 1310 | 712 | 2C8 |
| 1001011001 | 1131 | 601 | 259 | 1010010001 | 1221 | 657 | 291 | 1011001001 | 1311 | 713 | 2C9 |
| 1001011010 | 1132 | 602 | 25A | 1010010010 | 1222 | 658 | 292 | 1011001010 | 1312 | 714 | 2CA |
| 1001011011 | 1133 | 603 | 25B | 1010010011 | 1223 | 659 | 293 | 1011001011 | 1313 | 715 | 2CB |
| 1001011100 | 1134 | 604 | 25C | 1010010100 | 1224 | 660 | 294 | 1011001100 | 1314 | 716 | 2CC |
| 1001011101 | 1135 | 605 | 25D | 1010010101 | 1225 | 661 | 295 | 1011001101 | 1315 | 717 | 2CD |
| 1001011110 | 1136 | 606 | 25E | 1010010110 | 1226 | 662 | 296 | 1011001110 | 1316 | 718 | 2CE |
| 1001011111 | 1137 | 607 | 25F | 1010010111 | 1227 | 663 | 297 | 1011001111 | 1317 | 719 | 2CF |
| 1001100000 | 1140 | 608 | 260 | 1010011000 | 1230 | 664 | 298 | 1011010000 | 1320 | 720 | 2D0 |
| 1001100001 | 1141 | 609 | 261 | 1010011001 | 1231 | 665 | 299 | 1011010001 | 1321 | 721 | 2D1 |
| 1001100010 | 1142 | 610 | 262 | 1010011010 | 1232 | 666 | 29A | 1011010010 | 1322 | 722 | 2D2 |
| 1001100011 | 1143 | 611 | 263 | 1010011011 | 1233 | 667 | 29B | 1011010011 | 1323 | 723 | 2D3 |
| 1001100100 | 1144 | 612 | 264 | 1010011100 | 1234 | 668 | 29C | 1011010100 | 1324 | 724 | 2D4 |
| 1001100101 | 1145 | 613 | 265 | 1010011101 | 1235 | 669 | 29D | 1011010101 | 1325 | 725 | 2D5 |
| 1001100110 | 1146 | 614 | 266 | 1010011110 | 1236 | 670 | 29E | 1011010110 | 1326 | 726 | 2D6 |
| 1001100111 | 1147 | 615 | 267 | 1010011111 | 1237 | 671 | 29F | 1011010111 | 1327 | 727 | 2D7 |
| 1001101000 | 1150 | 616 | 268 | 1010100000 | 1240 | 672 | 2A0 | 1011011000 | 1330 | 728 | 2D8 |
| 1001101001 | 1151 | 617 | 269 | 1010100001 | 1241 | 673 | 2A1 | 1011011001 | 1331 | 729 | 2D9 |
| 1001101010 | 1152 | 618 | 26A | 1010100010 | 1242 | 674 | 2A2 | 1011011010 | 1332 | 730 | 2DA |
| 1001101011 | 1153 | 619 | 26B | 1010100011 | 1243 | 675 | 2A3 | 1011011011 | 1333 | 731 | 2DB |
| 1001101100 | 1154 | 620 | 26C | 1010100100 | 1244 | 676 | 2A4 | 1011011100 | 1334 | 732 | 2DC |
| 1001101101 | 1155 | 621 | 26D | 1010100101 | 1245 | 677 | 2A5 | 1011011101 | 1335 | 733 | 2DD |
| 1001101110 | 1156 | 622 | 26E | 1010100110 | 1246 | 678 | 2A6 | 1011011110 | 1336 | 734 | 2DE |
| 1001101111 | 1157 | 623 | 26F | 1010100111 | 1247 | 679 | 2A7 | 1011011111 | 1337 | 735 | 2DF |
| 1001110000 | 1160 | 624 | 270 | 1010101000 | 1250 | 680 | 2A8 | 1011100000 | 1340 | 736 | 2E0 |
| 1001110001 | 1161 | 625 | 271 | 1010101001 | 1251 | 681 | 2A9 | 1011100001 | 1341 | 737 | 2E1 |
| 1001110010 | 1162 | 626 | 272 | 1010101010 | 1252 | 682 | 2AA | 1011100010 | 1342 | 738 | 2E2 |
| 1001110011 | 1163 | 627 | 273 | 1010101011 | 1253 | 683 | 2AB | 1011100011 | 1343 | 739 | 2E3 |
| 1001110100 | 1164 | 628 | 274 | 1010101100 | 1254 | 684 | 2AC | 1011100100 | 1344 | 740 | 2E4 |
| 1001110101 | 1165 | 629 | 275 | 1010101101 | 1255 | 685 | 2AD | 1011100101 | 1345 | 741 | 2E5 |
| 1001110110 | 1166 | 630 | 276 | 1010101110 | 1256 | 686 | 2AE | 1011100110 | 1346 | 742 | 2E6 |
| 1001110111 | 1167 | 631 | 277 | 1010101111 | 1257 | 687 | 2AF | 1011100111 | 1347 | 743 | 2E7 |
| 1001111000 | 1170 | 632 | 278 | 1010110000 | 1260 | 688 | 2B0 | 1011101000 | 1350 | 744 | 2E8 |
| 1001111001 | 1171 | 633 | 279 | 1010110001 | 1261 | 689 | 2B1 | 1011101001 | 1351 | 745 | 2E9 |
| 1001111010 | 1172 | 634 | 27A | 1010110010 | 1262 | 690 | 2B2 | 1011101010 | 1352 | 746 | 2EA |
| 1001111011 | 1173 | 635 | 27B | 1010110011 | 1263 | 691 | 2B3 | 1011101011 | 1353 | 747 | 2EB |
| 1001111100 | 1174 | 636 | 27C | 1010110100 | 1264 | 692 | 2B4 | 1011101100 | 1354 | 748 | 2EC |
| 1001111101 | 1175 | 637 | 27D | 1010110101 | 1265 | 693 | 2B5 | 1011101101 | 1355 | 749 | 2ED |

| Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. | Binario | Octal | Dec. | Hex. |
|-------------|-------|------|------|------------|-------|------|------|------------|-------|------|------|
| 1011101110 | 1356 | 750 | 2EE | 1100100110 | 1446 | 806 | 326 | 1101011110 | 1536 | 862 | 35E |
| 1011101111 | 1357 | 751 | 2EF | 1100100111 | 1447 | 807 | 327 | 1101011111 | 1537 | 863 | 35F |
| 1011110000 | 1360 | 752 | 2F0 | 1100101000 | 1450 | 808 | 328 | 1101100000 | 1540 | 864 | 360 |
| 1011110001 | 1361 | 753 | 2F1 | 1100101001 | 1451 | 809 | 329 | 1101100001 | 1541 | 865 | 361 |
| 1011110010 | 1362 | 754 | 2F2 | 1100101010 | 1452 | 810 | 32A | 1101100010 | 1542 | 866 | 362 |
| 1011110011 | 1363 | 755 | 2F3 | 1100101011 | 1453 | 811 | 32B | 1101100011 | 1543 | 867 | 363 |
| 1011110100 | 1364 | 756 | 2F4 | 1100101100 | 1454 | 812 | 32C | 1101100100 | 1544 | 868 | 364 |
| 1011110101 | 1365 | 757 | 2F5 | 1100101101 | 1455 | 813 | 32D | 1101100101 | 1545 | 869 | 365 |
| 1011110110 | 1366 | 758 | 2F6 | 1100101110 | 1456 | 814 | 32E | 1101100110 | 1546 | 870 | 366 |
| 1011110111 | 1367 | 759 | 2F7 | 1100101111 | 1457 | 815 | 32F | 1101100111 | 1547 | 871 | 367 |
| 1011111000 | 1370 | 760 | 2F8 | 1100110000 | 1460 | 816 | 330 | 1101101000 | 1550 | 872 | 368 |
| 1011111001 | 1371 | 761 | 2F9 | 1100110001 | 1461 | 817 | 331 | 1101101001 | 1551 | 873 | 369 |
| 1011111010 | 1372 | 762 | 2FA | 1100110010 | 1462 | 818 | 332 | 1101101010 | 1552 | 874 | 36A |
| 1011111011 | 1373 | 763 | 2FB | 1100110011 | 1463 | 819 | 333 | 1101101011 | 1553 | 875 | 36B |
| 1011111100 | 1374 | 764 | 2FC | 1100110100 | 1464 | 820 | 334 | 1101101100 | 1554 | 876 | 36C |
| 1011111101 | 1375 | 765 | 2FD | 1100110101 | 1465 | 821 | 335 | 1101101101 | 1555 | 877 | 36D |
| 1011111110 | 1376 | 766 | 2FE | 1100110110 | 1466 | 822 | 336 | 1101101110 | 1556 | 878 | 36E |
| 1011111111 | 1377 | 767 | 2FF | 1100110111 | 1467 | 823 | 337 | 1101101111 | 1557 | 879 | 36F |
| 11000000000 | 1400 | 768 | 300 | 1100111000 | 1470 | 824 | 338 | 1101110000 | 1560 | 880 | 370 |
| 11000000001 | 1401 | 769 | 301 | 1100111001 | 1471 | 825 | 339 | 1101110001 | 1561 | 881 | 371 |
| 11000000010 | 1402 | 770 | 302 | 1100111010 | 1472 | 826 | 33A | 1101110010 | 1562 | 882 | 372 |
| 11000000011 | 1403 | 771 | 303 | 1100111011 | 1473 | 827 | 33B | 1101110011 | 1563 | 883 | 373 |
| 1100000100 | 1404 | 772 | 304 | 1100111100 | 1474 | 828 | 33C | 1101110100 | 1564 | 884 | 374 |
| 1100000101 | 1405 | 773 | 305 | 1100111101 | 1475 | 829 | 33D | 1101110101 | 1565 | 885 | 375 |
| 1100000110 | 1406 | 774 | 306 | 1100111110 | 1476 | 830 | 33E | 1101110110 | 1566 | 886 | 376 |
| 1100000111 | 1407 | 775 | 307 | 1100111111 | 1477 | 831 | 33F | 1101110111 | 1567 | 887 | 377 |
| 1100001000 | 1410 | 776 | 308 | 1101000000 | 1500 | 832 | 340 | 1101111000 | 1570 | 888 | 378 |
| 1100001001 | 1411 | 777 | 309 | 1101000001 | 1501 | 833 | 341 | 1101111001 | 1571 | 889 | 379 |
| 1100001010 | 1412 | 778 | 30A | 1101000010 | 1502 | 834 | 342 | 1101111010 | 1572 | 890 | 37A |
| 1100001011 | 1413 | 779 | 30B | 1101000011 | 1503 | 835 | 343 | 1101111011 | 1573 | 891 | 37B |
| 1100001100 | 1414 | 780 | 30C | 1101000100 | 1504 | 836 | 344 | 1101111100 | 1574 | 892 | 37C |
| 1100001101 | 1415 | 781 | 30D | 1101000101 | 1505 | 837 | 345 | 1101111101 | 1575 | 893 | 37D |
| 1100001110 | 1416 | 782 | 30E | 1101000110 | 1506 | 838 | 346 | 1101111110 | 1576 | 894 | 37E |
| 1100001111 | 1417 | 783 | 30F | 1101000111 | 1507 | 839 | 347 | 1101111111 | 1577 | 895 | 37F |
| 1100010000 | 1420 | 784 | 310 | 1101001000 | 1510 | 840 | 348 | 1110000000 | 1600 | 896 | 380 |
| 1100010001 | 1421 | 785 | 311 | 1101001001 | 1511 | 841 | 349 | 1110000001 | 1601 | 897 | 381 |
| 1100010010 | 1422 | 786 | 312 | 1101001010 | 1512 | 842 | 34A | 1110000010 | 1602 | 898 | 382 |
| 1100010011 | 1423 | 787 | 313 | 1101001011 | 1513 | 843 | 34B | 1110000011 | 1603 | 899 | 383 |
| 1100010100 | 1424 | 788 | 314 | 1101001100 | 1514 | 844 | 34C | 1110000100 | 1604 | 900 | 384 |
| 1100010101 | 1425 | 789 | 315 | 1101001101 | 1515 | 845 | 34D | 1110000101 | 1605 | 901 | 385 |
| 1100010110 | 1426 | 790 | 316 | 1101001110 | 1516 | 846 | 34E | 1110000110 | 1606 | 902 | 386 |
| 1100010111 | 1427 | 791 | 317 | 1101001111 | 1517 | 847 | 34F | 1110000111 | 1607 | 903 | 387 |
| 1100011000 | 1430 | 792 | 318 | 1101010000 | 1520 | 848 | 350 | 1110001000 | 1610 | 904 | 388 |
| 1100011001 | 1431 | 793 | 319 | 1101010001 | 1521 | 849 | 351 | 1110001001 | 1611 | 905 | 389 |
| 1100011010 | 1432 | 794 | 31A | 1101010010 | 1522 | 850 | 352 | 1110001010 | 1612 | 906 | 38A |
| 1100011011 | 1433 | 795 | 31B | 1101010011 | 1523 | 851 | 353 | 1110001011 | 1613 | 907 | 38B |
| 1100011100 | 1434 | 796 | 31C | 1101010100 | 1524 | 852 | 354 | 1110001100 | 1614 | 908 | 38C |
| 1100011101 | 1435 | 797 | 31D | 1101010101 | 1525 | 853 | 355 | 1110001101 | 1615 | 909 | 38D |
| 1100011110 | 1436 | 798 | 31E | 1101010110 | 1526 | 854 | 356 | 1110001110 | 1616 | 910 | 38E |
| 1100011111 | 1437 | 799 | 31F | 1101010111 | 1527 | 855 | 357 | 1110001111 | 1617 | 911 | 38F |
| 1100100000 | 1440 | 800 | 320 | 1101011000 | 1530 | 856 | 358 | 1110010000 | 1620 | 912 | 390 |
| 1100100001 | 1441 | 801 | 321 | 1101011001 | 1531 | 857 | 359 | 1110010001 | 1621 | 913 | 391 |
| 1100100010 | 1442 | 802 | 322 | 1101011010 | 1532 | 858 | 35A | 1110010010 | 1622 | 914 | 392 |
| 1100100011 | 1443 | 803 | 323 | 1101011011 | 1533 | 859 | 35B | 1110010011 | 1623 | 915 | 393 |
| 1100100100 | 1444 | 804 | 324 | 1101011100 | 1534 | 860 | 35C | 1110010100 | 1624 | 916 | 394 |
| 1100100101 | 1445 | 805 | 325 | 1101011101 | 1535 | 861 | 35D | 1110010101 | 1625 | 917 | 395 |

| Binario | Octal | Dec. Hex. | Binario | Octal | Dec. Hex. | Binario | Octal | Dec. Hex. |
|------------|-------|-----------|------------|-------|-----------|------------|-------|-----------|
| 1110010110 | 1626 | 918 396 | 1110111010 | 1672 | 954 3BA | 1111011110 | 1736 | 990 3DE |
| 1110010111 | 1627 | 919 397 | 1110111011 | 1673 | 955 3BB | 1111011111 | 1737 | 991 3DF |
| 1110011000 | 1630 | 920 398 | 1110111100 | 1674 | 956 3BC | 1111100000 | 1740 | 992 3E0 |
| 1110011001 | 1631 | 921 399 | 1110111101 | 1675 | 957 3BD | 1111100001 | 1741 | 993 3E1 |
| 1110011010 | 1632 | 922 39A | 1110111110 | 1676 | 958 3BE | 1111100010 | 1742 | 994 3E2 |
| 1110011011 | 1633 | 923 39B | 1110111111 | 1677 | 959 3BF | 1111100011 | 1743 | 995 3E3 |
| 1110011100 | 1634 | 924 39C | 1111000000 | 1700 | 960 3C0 | 1111100100 | 1744 | 996 3E4 |
| 1110011101 | 1635 | 925 39D | 1111000001 | 1701 | 961 3C1 | 1111100101 | 1745 | 997 3E5 |
| 1110011110 | 1636 | 926 39E | 1111000010 | 1702 | 962 3C2 | 1111100110 | 1746 | 998 3E6 |
| 1110011111 | 1637 | 927 39F | 1111000011 | 1703 | 963 3C3 | 1111100111 | 1747 | 999 3E7 |
| 1110100000 | 1640 | 928 3A0 | 1111000100 | 1704 | 964 3C4 | 1111101000 | 1750 | 1000 3E8 |
| 1110100001 | 1641 | 929 3A1 | 1111000101 | 1705 | 965 3C5 | 1111101001 | 1751 | 1001 3E9 |
| 1110100010 | 1642 | 930 3A2 | 1111000110 | 1706 | 966 3C6 | 1111101010 | 1752 | 1002 3EA |
| 1110100011 | 1643 | 931 3A3 | 1111000111 | 1707 | 967 3C7 | 1111101011 | 1753 | 1003 3EB |
| 1110100100 | 1644 | 932 3A4 | 1111001000 | 1710 | 968 3C8 | 1111101100 | 1754 | 1004 3EC |
| 1110100101 | 1645 | 933 3A5 | 1111001001 | 1711 | 969 3C9 | 1111101101 | 1755 | 1005 3ED |
| 1110100110 | 1646 | 934 3A6 | 1111001010 | 1712 | 970 3CA | 1111101110 | 1756 | 1006 3EE |
| 1110100111 | 1647 | 935 3A7 | 1111001011 | 1713 | 971 3CB | 1111101111 | 1757 | 1007 3EF |
| 1110101000 | 1650 | 936 3A8 | 1111001100 | 1714 | 972 3CC | 1111110000 | 1760 | 1008 3F0 |
| 1110101001 | 1651 | 937 3A9 | 1111001101 | 1715 | 973 3CD | 1111110001 | 1761 | 1009 3F1 |
| 1110101010 | 1652 | 938 3AA | 1111001110 | 1716 | 974 3CE | 1111110010 | 1762 | 1010 3F2 |
| 1110101011 | 1653 | 939 3AB | 1111001111 | 1717 | 975 3CF | 1111110011 | 1763 | 1011 3F3 |
| 1110101100 | 1654 | 940 3AC | 1111010000 | 1720 | 976 3D0 | 1111110100 | 1764 | 1012 3F4 |
| 1110101101 | 1655 | 941 3AD | 1111010001 | 1721 | 977 3D1 | 1111110101 | 1765 | 1013 3F5 |
| 1110101110 | 1656 | 942 3AE | 1111010010 | 1722 | 978 3D2 | 1111110110 | 1766 | 1014 3F6 |
| 1110101111 | 1657 | 943 3AF | 1111010011 | 1723 | 979 3D3 | 1111110111 | 1767 | 1015 3F7 |
| 1110110000 | 1660 | 944 3B0 | 1111010100 | 1724 | 980 3D4 | 1111111000 | 1770 | 1016 3F8 |
| 1110110001 | 1661 | 945 3B1 | 1111010101 | 1725 | 981 3D5 | 1111111001 | 1771 | 1017 3F9 |
| 1110110010 | 1662 | 946 3B2 | 1111010110 | 1726 | 982 3D6 | 1111111010 | 1772 | 1018 3FA |
| 1110110011 | 1663 | 947 3B3 | 1111010111 | 1727 | 983 3D7 | 1111111011 | 1773 | 1019 3FB |
| 1110110100 | 1664 | 948 3B4 | 1111011000 | 1730 | 984 3D8 | 1111111100 | 1774 | 1020 3FC |
| 1110110101 | 1665 | 949 3B5 | 1111011001 | 1731 | 985 3D9 | 1111111101 | 1775 | 1021 3FD |
| 1110110110 | 1666 | 950 3B6 | 1111011010 | 1732 | 986 3DA | 1111111110 | 1776 | 1022 3FE |
| 1110110111 | 1667 | 951 3B7 | 1111011011 | 1733 | 987 3DB | 1111111111 | 1777 | 1023 3FF |
| 1110111000 | 1670 | 952 3B8 | 1111011100 | 1734 | 988 3DC | | | |
| 1110111001 | 1671 | 953 3B9 | 1111011101 | 1735 | 989 3DD | | | |

Apéndice C

Tabla de conversión de números en complemento a dos

| Complemento a dos | Dec. |
|----------------------|------|----------------------|------|----------------------|------|----------------------|------|
| 11111111 | -1 | 11100101 | -27 | 11001011 | -53 | 10110001 | -79 |
| 11111110 | -2 | 11100100 | -28 | 11001010 | -54 | 10110000 | -80 |
| 11111101 | -3 | 11100011 | -29 | 11001001 | -55 | 10101111 | -81 |
| 11111100 | -4 | 11100010 | -30 | 11001000 | -56 | 10101110 | -82 |
| 11111011 | -5 | 11100001 | -31 | 11000111 | -57 | 10101101 | -83 |
| 11111010 | -6 | 11100000 | -32 | 11000110 | -58 | 10101100 | -84 |
| 11111001 | -7 | 11011111 | -33 | 11000101 | -59 | 10101011 | -85 |
| 11111000 | -8 | 11011110 | -34 | 11000100 | -60 | 10101010 | -86 |
| 11110111 | -9 | 11011101 | -35 | 11000011 | -61 | 10101001 | -87 |
| 11110110 | -10 | 11011100 | -36 | 11000010 | -62 | 10101000 | -88 |
| 11110101 | -11 | 11011011 | -37 | 11000001 | -63 | 10100111 | -89 |
| 11110100 | -12 | 11011010 | -38 | 11000000 | -64 | 10100110 | -90 |
| 11110011 | -13 | 11011001 | -39 | 10111111 | -65 | 10100101 | -91 |
| 11110010 | -14 | 11011000 | -40 | 10111110 | -66 | 10100100 | -92 |
| 11110001 | -15 | 11010111 | -41 | 10111101 | -67 | 10100011 | -93 |
| 11110000 | -16 | 11010110 | -42 | 10111100 | -68 | 10100010 | -94 |
| 11101111 | -17 | 11010101 | -43 | 10111011 | -69 | 10100001 | -95 |
| 11101110 | -18 | 11010100 | -44 | 10111010 | -70 | 10100000 | -96 |
| 11101101 | -19 | 11010011 | -45 | 10111001 | -71 | 10011111 | -97 |
| 11101100 | -20 | 11010010 | -46 | 10111000 | -72 | 10011110 | -98 |
| 11101011 | -21 | 11010001 | -47 | 10110111 | -73 | 10011101 | -99 |
| 11101010 | -22 | 11010000 | -48 | 10110110 | -74 | 10011100 | -100 |
| 11101001 | -23 | 11001111 | -49 | 10110101 | -75 | 10011011 | -101 |
| 11101000 | -24 | 11001110 | -50 | 10110100 | -76 | 10011010 | -102 |
| 11100111 | -25 | 11001101 | -51 | 10110011 | -77 | 10011001 | -103 |
| 11100110 | -26 | 11001100 | -52 | 10110010 | -78 | 10011000 | -104 |

| Complemento a dos | Dec. |
|----------------------|------|----------------------|------|----------------------|------|----------------------|------|
| 10010111 | -105 | 10010001 | -111 | 10001011 | -117 | 10000101 | -123 |
| 10010110 | -106 | 10010000 | -112 | 10001010 | -118 | 10000100 | -124 |
| 10010101 | -107 | 10001111 | -113 | 10001001 | -119 | 10000011 | -125 |
| 10010100 | -108 | 10001110 | -114 | 10001000 | -120 | 10000010 | -126 |
| 10010011 | -109 | 10001101 | -115 | 10000111 | -121 | 10000001 | -127 |
| 10010010 | -110 | 10001100 | -116 | 10000110 | -122 | 10000000 | -128 |

6

Glosario

ACARREO (*carry*). Suma de un dígito 1 a la posición superior siguiente del bit o a indicador de acarreo.

ACARREO NEGATIVO (*borrow*). Un bit 1 que se resta del dígito binario superior siguiente.

ACUMULADOR (*accumulator*). Registro principal de un microprocesador utilizado para operaciones aritméticas, lógicas, desplazamientos y otras. El microprocesador Z-80 posee uno (registro A), mientras que el microprocesador 6809 tiene dos (registros A y B).

ALGORITMO (*algorithm*). Descripción paso a paso de un proceso para ejecutar una tarea.

ASCII (*ASCII*). Código estándar para la representación de caracteres en ordenadores y en sus equipos periféricos. Su significado es “American Standard Code for Information Interchange”.

BASE (*base*). Punto de partida para la representación de un número en forma escrita, donde los números se expresan como múltiplos de potencias del valor de la base.

BINARIO (*binary*). Representación de números en “base dos”, donde todo número se expresa por combinación de los dígitos binarios 0 y 1.

BIT (*bit*). Contracción de “dígito binario” (*binary digit*).

BIT DE SIGNO (*sign bit*). Bit más a la izquierda (posición 15 ó 7) de un número en complemento a dos. Si es un cero, el signo del número es positivo. Si es un uno, el signo del número es negativo.

BIT MAS SIGNIFICATIVO (*most significant bit*). El bit de más a la izquierda en un valor binario, representa el orden superior de potencias de dos. En notación en complemento a dos este bit es el signo.

BIT MENOS SIGNIFICATIVO (*least significant bit*). El bit más a la derecha de un valor binario, representado por 2^{10} .

BITS SIGNIFICATIVOS (*significant bits*). Número de bits en un valor binario, después de quitar los ceros a la izquierda.

BYTE (*byte*). Colección de 8 bits. Cada posición de memoria en la mayoría de los microordenadores tiene el tamaño de un byte.

BYTE MAS SIGNIFICATIVO (*most significant byte*). El byte de orden superior. En el número A13EF122H de múltiple precisión, los dígitos hexadecimales A y 1 forman el byte más significativo.

CLOBBER (*clobber*). Destruir el contenido de una memoria o de un registro.

COCIENTE (*quotient*). Resultado de una división.

CODIGO EXCESO A 128 (*excess code 128*). Método estándar para poner el exponente en el BASIC de Microsoft™. El valor 128 se añade a la potencia actual de dos y luego se almacena como un exponente en una representación de punto flotante.

COLOSSUS (*Colossus*). Ordenador británico utilizado durante la Segunda Guerra Mundial para descifrar los códigos germanos "Enigma".

COMPLEMENTO A DOS (*two's complement*). Forma estándar de representar números positivos y negativos en microordenadores.

DATOS (*data*). Término genérico que designa números, operandos, instrucciones del programa, indicadores o cualquier representación de información utilizando unos o ceros binarios.

DESPLAZAMIENTO (*displacement*). Valor con signo utilizado en lenguaje máquina, que se emplea para definir una dirección de memoria.

DESPLAZAMIENTO ARITMETICO (*arithmetic shift*). Tipo de desplazamiento en el que un operando es movido a derecha o a izquierda usando el bit de signo (desplazamiento a la derecha) o manteniéndolo (desplazamiento a la izquierda).

DESPLAZAMIENTO LOGICO (*logical shift*). Tipo de desplazamiento en el que un operando se desplaza a derecha o izquierda, con un cero ocupando la posición vacante del bit.

DESPLAZAMIENTO Y SUMA (*shift and add*). Método en el que la multiplicación se ejecuta por desplazamiento y suma del multiplicando.

DIGITO BINARIO (*binary digit*). Los dos dígitos (0 y 1) utilizados en notación binaria. A menudo abreviados por "bit".

DISPOSITIVOS PERIFERICOS (*peripheral devices*). Término genérico para definir el equipo conectado a un ordenador, tal como teclados, unidades de disco, magnetófonos de cassette, impresoras, tableros de dibujo electrónicos, sintetizadores de voz, etc.

DIVIDENDO (*dividend*). Número por el que se divide al divisor. En A/B , A es el dividendo.

DIVISION CON RECUPERACION (*restoring division*). División en la que el divisor se recupera si la operación no efectúa ninguna iteración. Técnica común de división en microordenadores.

DIVISOR (*divisor*). Número que va bajo el dividendo en una división. En A/B , B es el divisor.

DOBLAR Y SUMAR (*double-dabble*). Método para convertir de binario a representación decimal por duplicación del bit más a la izquierda, sumando el próximo bit y continuando hasta que se use el bit de más a la derecha.

ENIGMA (*enigma*). Máquina de claves alemana (Segunda Guerra Mundial).

EQUIPO PERFORADOR DE TARJETAS (*punched-card equipment*). Dispositivos periféricos que permiten perforar o leer las tarjetas de papel empleadas para almacenar caracteres o datos binarios.

ERROR DE DESBORDAMIENTO (*overflow error*). Condición que se da cuando el resultado de una suma, resta u otra operación aritmética es demasiado grande para poderlo meter en el número de bits que se le asignaron.

EXPONENTE (*exponent*). En este libro, normalmente, la potencia de dos de un número binario en punto flotante.

EXTENSION DEL SIGNO (*sign extension*). Extender el bit de signo de un número en complemento a dos a la izquierda por duplicación.

FACTOR DE ESCALA (*scaling*). Cantidad fija por la que hay que multiplicar un número de forma que pueda procesarse como un valor entero.

H (H). Sufijo para números hexadecimales.

HEXADECIMAL (*hexadecimal*). Representación de números en "base dieciséis" utilizando los dígitos hexadecimales 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.

INDICADOR DE ACARREO (*carry flag*). Un bit en el microprocesador utilizado para almacenar el acarreo resultante de una instrucción en lenguaje máquina.

INDICADOR DE CERO (*zero flag*). Un bit del microprocesador utilizado para almacenar el resultado cero/no cero de una instrucción en lenguaje máquina.

INDICADOR DE ERROR DE DESBORDAMIENTO (*overflow flag*). Bit en el microprocesador utilizado para almacenar una condición de error de desbordamiento producido en las operaciones en lenguaje máquina.

INDICADOR DE SIGNO (*sign flag*). Bit en el microprocesador que se usa para almacenar el signo del resultado de una operación en lenguaje máquina.

INVERSION DE SIGNO. Véase NEGACION.

ITERACION (*iteration*). Una pasada a través de un conjunto dado de instrucciones.

LENQUAJE ENSAMBLADOR (*assembly language*). Lenguaje simbólico de ordenadores que se traduce por medio de un programa ensamblador al lenguaje máquina (códigos numéricos que son equivalentes a las instrucciones del microprocesador).

LENQUAJE MAQUINA (*machine language*). Conjunto ordenado de códigos numéricos compuesto por instrucciones del microporcesador. Estos valores son producidos por un programa ensamblador desde el código de lenguaje ensamblador.

MAGNITUD Y SIGNO (*sign magnitude*). Forma no estándar de representar números positivos y negativos en microordenadores.

MANTISA (*mantissa*). Porción fraccionaria de un número en punto flotante.

MEMORIA TEMPORAL (*buffer*). Porción de memoria destinada a almacenar caracteres (u otros datos) cuando son leídos, o utilizada para almacenar caracteres (u otros datos) para salida.

MEMORIA TEMPORAL DE IMPRESION (*print buffer*). Lugar de la memoria dedicado a almacenar las líneas de caracteres que se van a imprimir.

MINUENDO (*minuend*). Número al que se le resta el sustraendo. En 5-3, 5 es el minuendo.

MULTIPLICACION DE OCHO POR OCHO (*eight-by-eight multiply*). Multiplicación de ocho bits por ocho bits para generar un resultado de dieciséis bits.

MULTIPLICACION POR DIECISEIS Y SUMA (*hexa-dabble*). Conversión de hexadecimal a decimal multiplicando cada dígito hexadecimal por 16 y sumando el siguiente dígito hasta llegar al último dígito (el de más a la derecha).

MULTIPLICADOR (*multiplier*). Número que se multiplica por el multiplicando. El número “de abajo”.

MULTIPLICANDO (*multiplicand*). Número multiplicado por el multiplicador. El número de “arriba”.

MULTIPLICAR POR OCHO Y SUMAR (*octal-dabble*). Transformación de un número octal en decimal multiplicando por ocho y sumando el siguiente dígito octal, continuando así hasta que el último dígito se transforme (el de más a la derecha).

NEGACION (*negation*). Cambiar un valor positivo por otro negativo o viceversa. En complemento a dos significa cambiar todos los unos por ceros, todos los ceros por unos, y sumar un uno.

NO (OPERACION) (*not*). Operación lógica que invierte el número o realiza su complemento a uno.

NORMALIZACION (*normalization*). Convertir un dato a un formato estándar para procesarlo. En formato de punto flotante, convertir un número de modo que un bit significativo (o dígito hexadecimal) esté en el primer bit (o cuarto bit) de la fracción.

NOTACION CIENTIFICA (*scientific notation*). Forma estándar de representar cualquier tipo de número con una mantisa y una potencia de diez.

NOTACION POSICIONAL (*positional notation*). Representación de un número donde cada dígito representa una potencia superior de la base.

NUMERO CON SIGNO (*signed numbers*). Números que pueden ser positivos o negativos.

NUMERO EN PUNTO FLOTANTE (*floating-point number*). Forma estándar de representar un número de cualquier tamaño en microordenadores. Los números en punto flotante constan de una parte fraccionaria (mantisa) y de una potencia de dos (exponente) en una forma similar a la notación científica.

NUMERO ESCALADO (*scaling up*). Se refiere a un número que se ha multiplicado por un factor de escala para procesarlo.

NUMERO MIXTO (*mixed number*). Número que consta de un entero y una fracción, como, por ejemplo, 4.35 o (en binario) 1010.1011.

NUMEROS DE MULTIPLE PRECISION (*multiple-precision numbers*). Números de varios bytes, que permiten aumentar la precisión.

NUMEROS SIN SIGNO (*unsigned numbers*). Números que sólo pueden ser positivos. Números en valor absoluto.

O (OPERACION) (*or*). Véase O INCLUSIVA.

- O EXCLUSIVA (OPERACION) (*exclusive-or*). Operación lógica bit a bit que produce un 1 en el resultado sólo si uno u otro (pero no ambos) de los bits operados es un 1.
- O INCLUSIVA (OPERACION) (*inclusive-or*). Operación lógica bit a bit en la que se obtiene un 1 como resultado si uno u otro de los bits que operan, o ambos, es un 1.
- OCTAL (*octal*). Representación de números en "base ocho", utilizando los dígitos octales: 0, 1, 2, 3, 4, 5, 6 y 7.
- OPERANDOS (*operands*). Valores numéricos empleados en sumas, restas y otras operaciones.

- PALABRA (*word*). Colección de diecisésis dígitos binarios. Dos bytes.
- PERMUTACION (*permutation*). Colocación de cosas en un orden definido. Dos dígitos binarios tienen cuatro permutaciones: 00, 01, 10 y 11.
- POSICION DEL BIT (*bit position*). Posición de un dígito binario dentro de un byte o de un grupo más largo de dígitos binarios. Las posiciones de los bits en la mayoría de los microordenadores están numeradas de derecha a izquierda, de O a N, donde este número corresponde a la potencia de 2 representada.
- PRECISION (*precision*). Número de dígitos significativos que puede tener una variable o el formato de un número.
- PRODUCTO (*product*). Resultado de una multiplicación.
- PRODUCTO PARCIAL (*partial product*). Resultado intermedio de una multiplicación. Al final, el producto parcial pasa a ser el producto final.
- PROPAGACION (*propagation*). Sistema por el que el acarreo (positivo o negativo) viaja a la posición del bit más alto siguiente.
- PUNTO BINARIO (*binary point*). El punto, análogo al punto decimal, que separa porciones enteras y fraccionarias de un número puesto en sistema binario.

- REDONDEO (*rounding*). Proceso de quitar los bits a la derecha de una posición y añadir un 0 o un 1 a la siguiente superior, basada en el valor de la derecha. Redondear la fracción binaria 1011.1011 a dos bits fraccionarios, por ejemplo, da como resultado 1011.11.
- REGISTRO (*register*). Posición de memoria de acceso rápido en el microprocesador de un microordenador. Se utiliza para almacenar resultados parciales y, también, para operaciones en lenguaje máquina.
- REGISTRO DE MULTIPLICANDOS (*multiplicand register*). Registro utilizado para almacenar el multiplicando en una operación en lenguaje máquina.

REGISTRO DE PRODUCTOS PARCIALES (*partial-product register*).

Registro utilizado para almacenar los resultados parciales de una multiplicación en lenguaje máquina.

RELLENO A CEROS (*padding*). Rellenar posiciones a la izquierda de una cifra con ceros para formar un total de 8 ó 16 bits.

RESIDUO (*residue*). Cantidad que queda como dividendo cuando una división no está terminada.

RESTA CON ACARREO (*subtract with carry*). Instrucción en lenguaje máquina en la que un operando se resta de otro, junto con un posible acarreo negativo de un byte anterior.

RESTO (*remainder*). Cantidad que queda como dividendo después de terminada una división.

ROTAR (*rotate*). Tipo de desplazamiento en el que el dato que sale por la derecha o por la izquierda entra por el lado opuesto.

SALTO CONDICIONAL (*conditional jump*). Instrucción en lenguaje máquina que realiza un salto a otra instrucción si un indicador o indicadores determinados están a nivel alto o a nivel bajo.

SERIES DE FIBONACCI (*Fibonacci series*). La secuencia de números 1, 1, 2, 3, 5, 8, 13, 21, 34, ..., donde cada término se obtiene por la adición de los dos términos que le preceden.

SUSTRAENDO (*subtrahend*). El número que se resta del minuendo. En $5 - 3 = 2$, 3 es el sustraendo.

SUMA CON ACARREO (*add with carry*). Instrucción en lenguaje máquina en la que un operando se suma a otro, con un posible acarreo de la suma anterior de orden inferior.

SUMAS SUCESIVAS (*successive addition*). Método de multiplicación en el que el multiplicando se suma un número de veces igual al multiplicador para encontrar el resultado.

TABLA DE VERDAD (*truth table*). Tabla que define los resultados de varias variables diferentes y que contiene todos los posibles estados de éstas.

TRUNCAR (*truncation*). Proceso de quitar bits de la derecha de la posición de un bit. Truncar la fracción binaria de 1011.1011 a un número con una fracción de dos bits, por ejemplo, resulta 1011.10.

TUBO DE WILLIAMS (*Williams tube*). Primitivo tipo de memoria basado en el almacenaje de los datos en la superficie de un tubo de rayos catódicos. Diseñado por F. C. Williams, Universidad de Manchester.

TURING (*Turing*). Científico y matemático británico, pionero en la ciencia de los ordenadores.

VARIABLE ENTERA (*integer variable*). Tipo de variable en BASIC. Puede tomar valores desde -32,768 hasta +32,767, en una notación en complemento a dos de dos bytes.

Y (OPERACION) (*and*). Operación lógica bit a bit que produce un 1 en el bit de resultado sólo cuando los dos operandos son unos.

Índice alfabético

- Acarreo, 42, 46, 51-52, 66-67, 94, 102.
Acarreo negativo (*borrow*), 42-43, 51, 149.
Acarreos, 8, 49, 60, 62, 149.
Acumulador, 51, 149.
Algoritmo, 149.
AND, 8.
Aritmética decimal, 42, 79.

Base 126, 34.
Base 3, 34.
Base 40, 34.
Base 5, 34.
Base 8, 32.
BASIC, 7-8, 18, 25, 35, 44, 51, 60, 62-65, 88, 100-101, 128, 132.
Binario, 14, 16, 27-29, 149.
Bit, 14, 21, 28, 43, 57, 62, 66-67, 90, 149.
“Bit a bit”, 60, 81.
Bit de acarreo, 75.

Bit de signo, 38, 43, 68, 89, 150.
Bit más significativo, 68, 80, 90, 111, 150.
Bit menos significativo, 63, 102, 150.
Bits, 15-16, 19, 27, 29, 32, 35, 42, 44-45, 50-52, 62-63, 65, 74-75, 79, 81, 83, 87-88, 104, 111.
Bits de datos, 18.
Bits significativos, 150.
Buffer, 152.
Byte del exponente, 128.
Byte más significativo, 128, 150.
Byte menos significativo, 90-91, 128.
Bytes, 16-19, 34-35, 60, 62, 87, 89-92, 103, 150.

Campo, 57.
Carácter ASCII, 113.
Clobber, 150.
Cociente, 150.

- Códigos ASCII, 8, 110-111.
Código exceso a 128, 150.
Complemento a dos, 7-9, 41, 43, 46, 51, 91, 147, 150.
Complemento a uno, 64.
Condiciones lógicas, 64.
Conmutador (*toggle*), 64.
Constantes, 129.
Convenios estándar, 35.
Conversiones, 139-145.
CPU, 17-18.
- Datos binarios, 109, 150.
Decimal, 7, 14, 29-31, 34-35.
Desplazamiento, 44, 57, 66, 79, 150.
Desplazamiento aritmético, 67-68, 150.
Desplazamiento lógico, 65-66, 77, 150.
Dispositivos periféricos, 151.
Direcciones de memoria, 91.
División, 73.
División bit a bit, 82.
División con "recuperación", 81-82, 151.
División con signo, 84.
División sin signo, 84.
Divisiones en múltiple precisión, 95.
- Errores de desbordamiento, 8, 46, 49-52, 81, 151.
Exponente, 123, 151.
Extensión del signo, 44, 151.
- Factor de escala, 103, 151.
Forma escalada, 104.
Formatos, 44, 91, 109.
Formatos en punto flotante, 129.
- H, 151.
Hardware, 27, 40, 78.
Hexadecimal, 25, 27-29, 31-32, 34, 151.
- Indicador "cero", 52, 152.
Indicador de acarreo, 52, 65, 90, 151.
Indicador de error, 50, 152.
Indicador "signo", 52, 152.
Indicadores, 46, 52-53, 60.
Instrucción PEEK, 18.
Instrucción POKE, 18.
Intérprete BASIC, 132.
Iteraciones, 74-75, 152.
- Lenguaje ensamblador, 7-8, 19, 25, 44, 50, 152.
Lenguaje máquina, 8, 19, 51-52, 60, 62-65, 73, 101, 152.
Lenguaje máquina CPL, 64.
- Magnitud y signo, 152.
Mantisa, 121, 128, 152.
Memoria, 16, 18, 27, 34, 44, 91.
Memoria de video, 62.
Método "desplazamiento y suma", 74.
Método "dividir por dos y guardar los restos", 21.
Método "doblar y sumar", 19.
Método de "inspección de potencias de dos", 21.
Microordenadores, 7, 16-19, 25, 28, 32, 37-39, 41, 45, 51, 57, 73, 83, 88-89.
Microprocesador, 17, 19, 44, 50, 52, 65.
Microprocesador 6809, 52.
Microprocesador Z-80, 52.
Microprocesadores, 11, 14-16, 26, 38, 45, 58.
Microprocesadores 6502, 29.
Microprocesadores 6809, 29.
Microprocesadores 8080, 32.
Microprocesadores Z-80, 29.
Múltiple precisión, 87-88, 91.
Multiplicación, 73.
Multiplicación con signo, 79.
Multiplicación en múltiple precisión, 92.

Multiplicación por desplazamiento y suma, 78.
Multiplicación sin signo, 79.
MuMath, 34.

Negación, 153.
Nibble, 18.
Normalización, 127, 153.
Normalizar, 123.
Notación científica, 121, 153.
Notación científica en punto flotante, 123.
Notación en complemento a dos, 37-40.
Notación en punto flotante, 122.
Notación posicional, 26, 153.
Número binario, 16.
Números binarios, 7, 14, 20, 25, 41, 57, 97.
Números binarios sin signo, 37.
Números con signo, 37, 41, 49, 67, 153.
Números decimales, 8, 14, 26, 41.
Números en punto flotante, 121, 153.
Números en punto flotante de doble precisión, 129.
Números escalados, 153.
Números hexadecimales, 8, 21, 35.
Números mixtos, 100-101, 153.
Números negativos, 38-39.
Números octales, 8, 21, 35.

Octal, 7, 25, 32, 34, 154.
Operación NO, 64, 153.
Operación NOT, 8, 64, 153.
Operación O exclusiva, 59-60, 63, 80, 154.
Operación O, 60-61, 153.
Operación O inclusiva, 59, 154.
Operación Y, 62-63, 65.
Operación de invertir el signo, 64.
Operación lógica, 7.
Operación lógica de desplazamiento, 67.

Operaciones aritméticas, 19, 38, 50, 52.
Operaciones de desplazamiento, 52, 65.
Operaciones lógicas, 8, 52, 57, 60.
Operadores lógicos, 79.
OR, 8.
Ordenadores, 65, 67, 75, 78, 88.

Palabras, 18, 154.
Paso de ASCII a enteros binarios, 112.
Paso de ASCII a fracciones binarias, 113.
Paso de binario a decimal, 19.
Paso de binario a hexadecimal, 29.
Paso de decimal a binario, 21.
Paso de decimal a hexadecimal, 30.
Paso de enteros binarios a ASCII, 115.
Paso de fracciones binarias a ASCII, 117.
Paso de hexadecimal a binario, 30.
Paso entre binario y octal, 32.
Paso entre octal y decimal, 33.
Permutación, 154.
Posiciones de memoria, 17-18, 51, 65.
Potencias de diez, 126.
Potencias de dos, 126.
Punto flotante, 88-89, 101.

RAM, 17-18.
Rango de números, 131.
Redondeo, 154.
Registro de los multiplicandos, 74, 154.
Registro de producto parcial, 74-75, 154.
Registros, 17, 50-51, 65-66, 74, 83, 92, 154.
Representación “signo/magnitud”, 38.
Residuos, 155.
Resta sucesiva, 81.

- ROM, 17-18.
Rotaciones, 65-66.
Rutinas, 73, 101.
- Salto condicional, 51-52, 65, 155.
Semiconductores, 14, 25, 109.
Series de Fibonacci, 155.
Sistema binario, 7-8, 11-12, 19, 50.
Sistema decimal, 13, 19.
Sistema hexadecimal, 19.
Sistema octal, 19.
Software, 73, 81, 89.
Suma sucesiva de potencias de dos, 77.
Suma y resta de números binarios, 41.
- Suma y resta en complemento a dos, 45.
Sumas sucesivas, 75, 155.
- Tabla de verdad, 60, 155.
Transformaciones ASCII, 109.
Truncar, 128, 155.
- Valor hexadecimal, 7.
Valores absolutos, 79, 91.
Valores enteros, 97.
Variables enteras, 18, 44, 156.
- Y, 59-60, 156.
- Z-80, 32, 50, 91, 128.

Matemáticas para programadores

Cuando se programa en BASIC suele ocurrir que, para ganar efectividad, hay que trabajar en binario o hexadecimal y realizar operaciones lógicas, desplazamientos y otras manipulaciones de bits; cuando se trabaja en ensamblador es necesario operar con números en punto flotante o con signo y realizar operaciones matemáticas que van más allá de lo que permite el microprocesador.

El objetivo de **MATEMATICAS PARA PROGRAMADORES** es desvelar y aclarar eficazmente los algoritmos de las operaciones matemáticas involucradas en la programación BASIC o máquina.

A lo largo del libro encontrará:

- Definición de bases y algoritmos de transformación entre ellas.
- Sistemas de notación de números con signo.
- Sumas y restas en complemento a uno y a dos.
- Algoritmos de multiplicación y división en lenguaje máquina.
- Estructura de los valores de precisión múltiple y punto flotante.
- Códigos y conversiones ASCII.

MATEMATICAS PARA PROGRAMADORES posee, además, numerosísimos ejemplos y ejercicios de evaluación que facilitan el uso del libro, bien como referencia, o como manual de aprendizaje.

MATEMATICAS PARA PROGRAMADORES es un texto imprescindible en la biblioteca de cualquier programador y será tu mejor compañero cuando programes en ensamblador o en BASIC Avanzado.