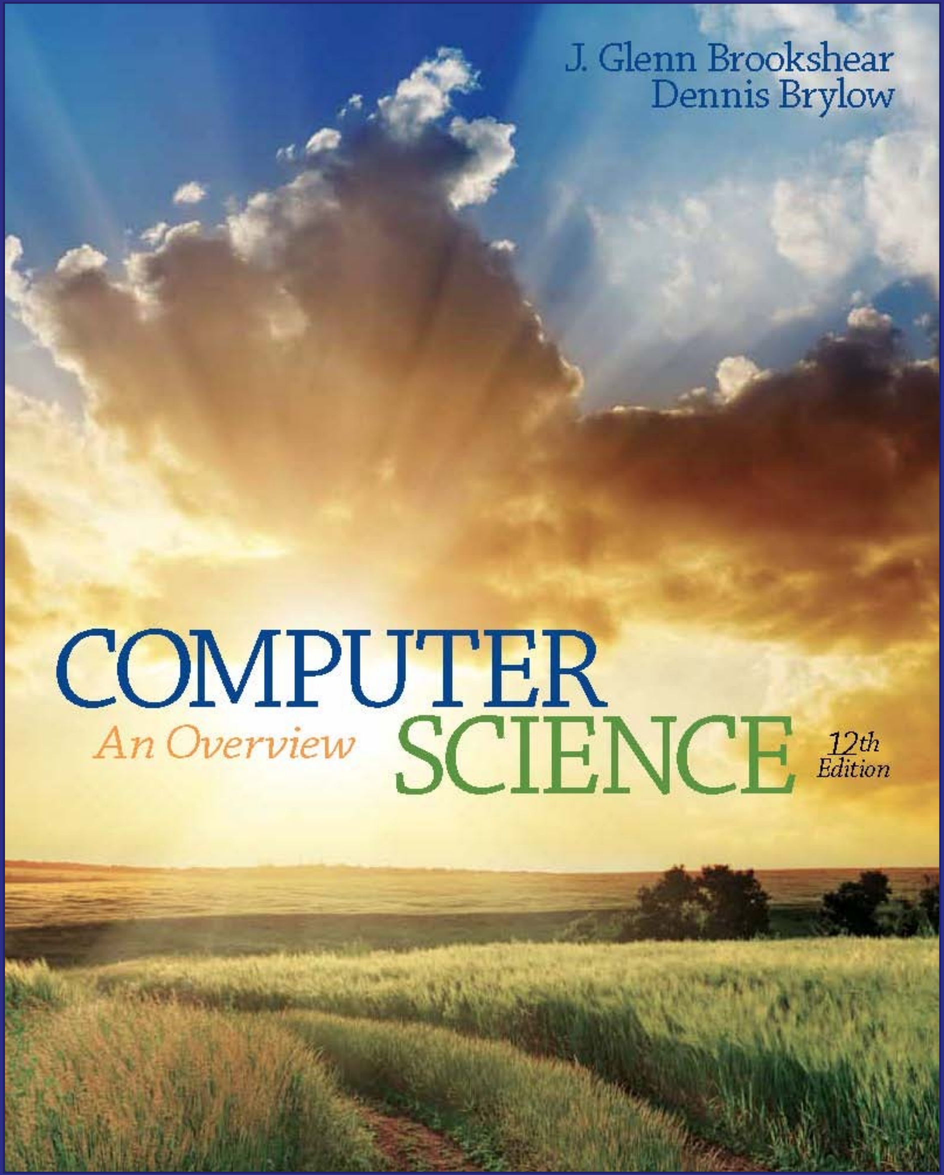
Chapter 1: Data Storage



PEARSON

Chapter 1: Data Storage

Chapter 1: Data Storage (continued)

- 1.6 Storing Integers
- 1.7 Storing Fractions
- 1.8 Data and Programming
- 1.9 Data Compression
- 1.10 Communications Errors

Bits and Bit Patterns

- **Bit:** Binary Digit (0 or 1)
- Bit Patterns are used to represent information
  - Numbers
  - Text characters
  - Images
  - Sound
  - And others

Boolean Operations

- **Boolean Operation:** An operation that manipulates one or more true/false values
- Specific operations
  - AND
  - OR
  - XOR (exclusive or)
  - NOT

Figure 1.1 **The possible input and output values of Boolean operations AND, OR, and XOR (exclusive or)**

The AND operation

$$\begin{array}{r} 0 \\ \text{AND}\quad 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ \text{AND}\quad 1 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ \text{AND}\quad 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ \text{AND}\quad 1 \\ \hline 1 \end{array}$$

The OR operation

$$\begin{array}{r} 0 \\ \text{OR}\quad 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ \text{OR}\quad 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ \text{OR}\quad 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ \text{OR}\quad 1 \\ \hline 1 \end{array}$$

The XOR operation

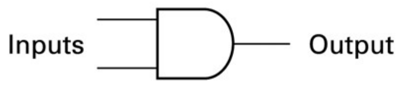
$$\begin{array}{r} 0 \\ \text{XOR}\quad 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ \text{XOR}\quad 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ \text{XOR}\quad 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ \text{XOR}\quad 1 \\ \hline 0 \end{array}$$

Gates

- **Gate:** A device that computes a Boolean operation
  - Often implemented as (small) electronic circuits
  - Provide the building blocks from which computers are constructed
  - VLSI (Very Large Scale Integration)

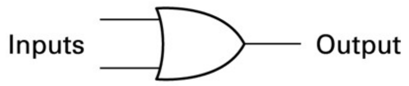
Figure 1.2 **A pictorial representation of AND, OR, XOR, and NOT gates as well as their input and output values**

AND



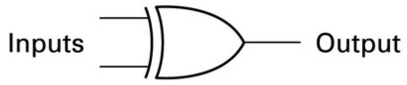
Inputs	Output
0 0	0
0 1	0
1 0	0
1 1	1

OR



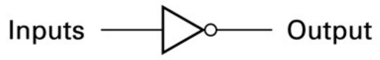
Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	1

XOR



Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	0

NOT



Inputs	Output
0	1
1	0

Flip-flops

- **Flip-flop:** A circuit built from gates that can store one bit.
  - One input line is used to set its stored value to 1
  - One input line is used to set its stored value to 0
  - While both input lines are 0, the most recently stored value is preserved

Figure 1.3 **A simple flip-flop circuit**

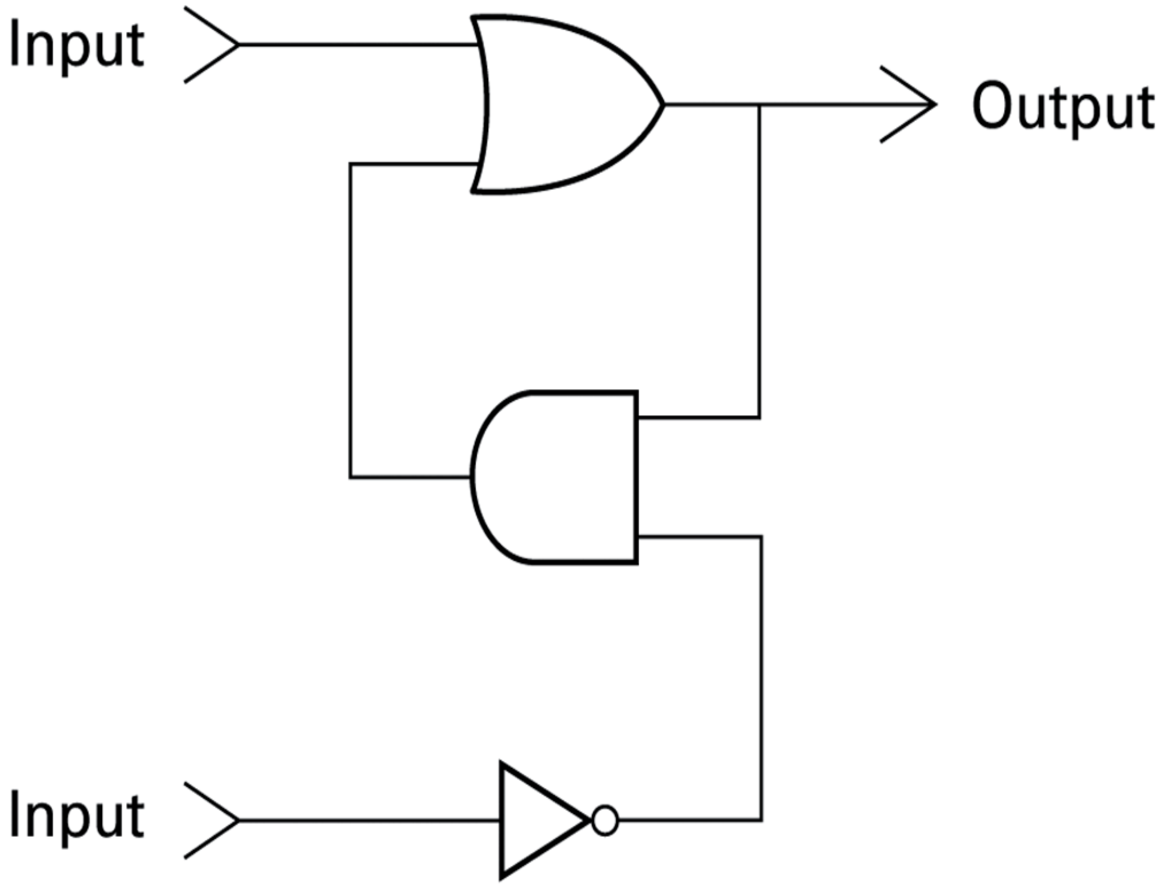


Figure 1.4 **Setting the output of a flip-flop to 1**

a. First, a 1 is placed on the upper input.

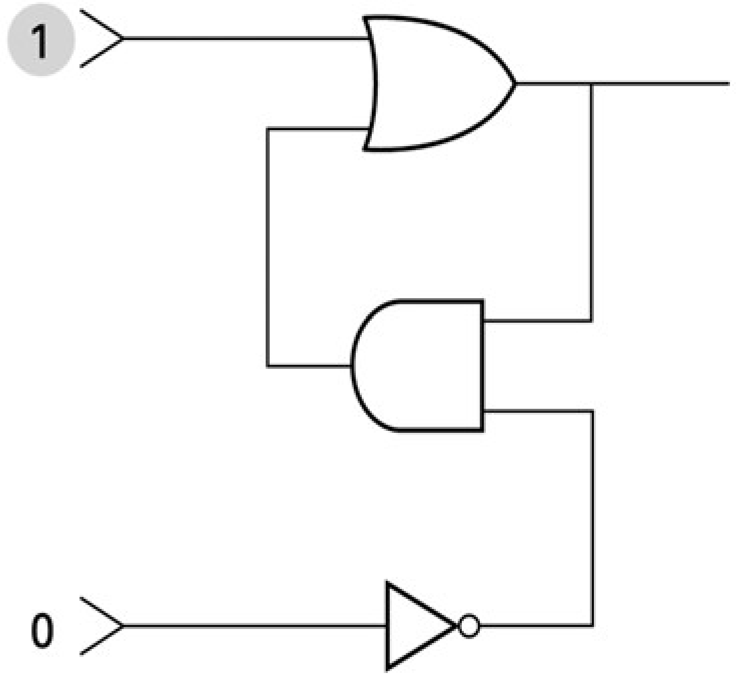


Figure 1.4 **Setting the output of a flip-flop to 1 (continued)**

b. This causes the output of the OR gate to be 1 and, in turn, the output of the AND gate to be 1.

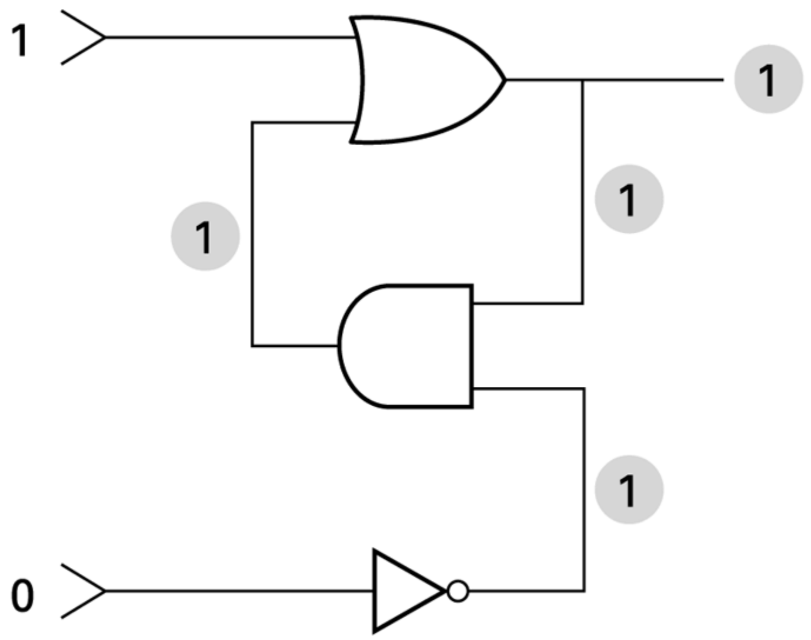


Figure 1.4 **Setting the output of a flip-flop to 1 (continued)**

c. Finally, the 1 from the AND gate keeps the OR gate from changing after the upper input returns to 0.

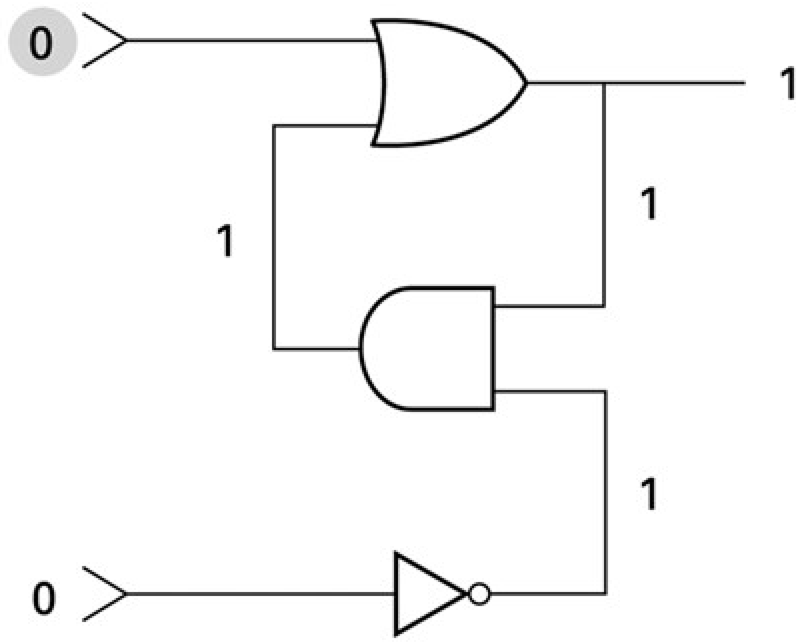
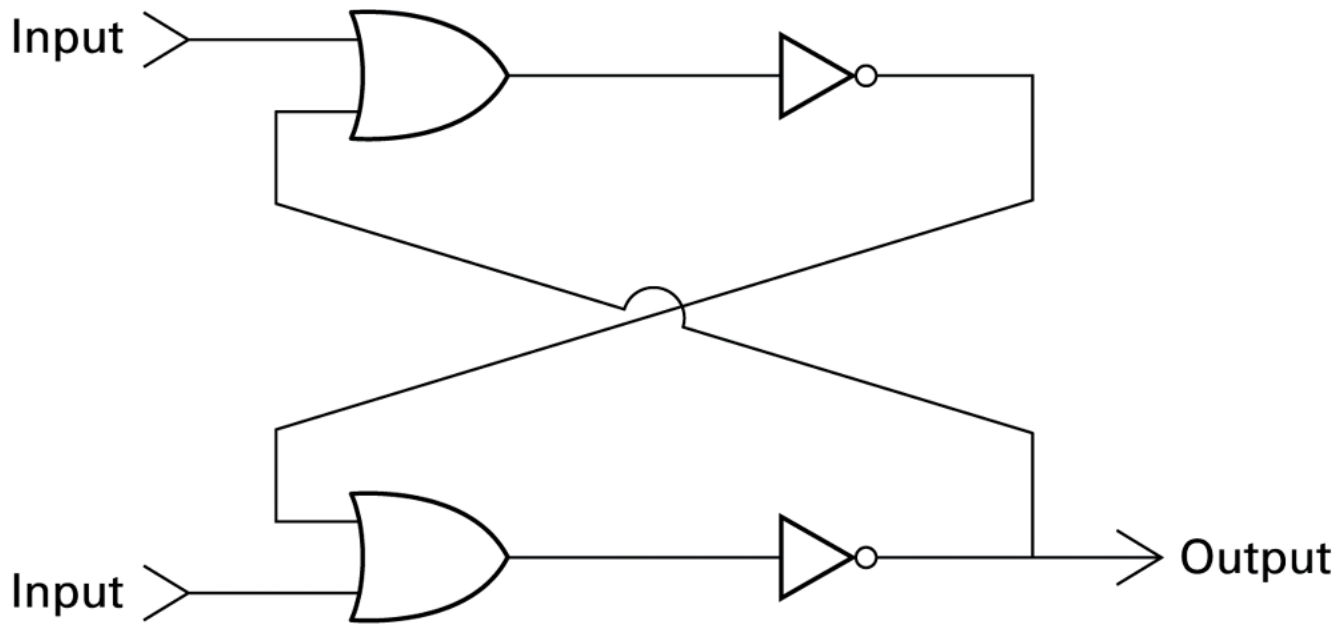


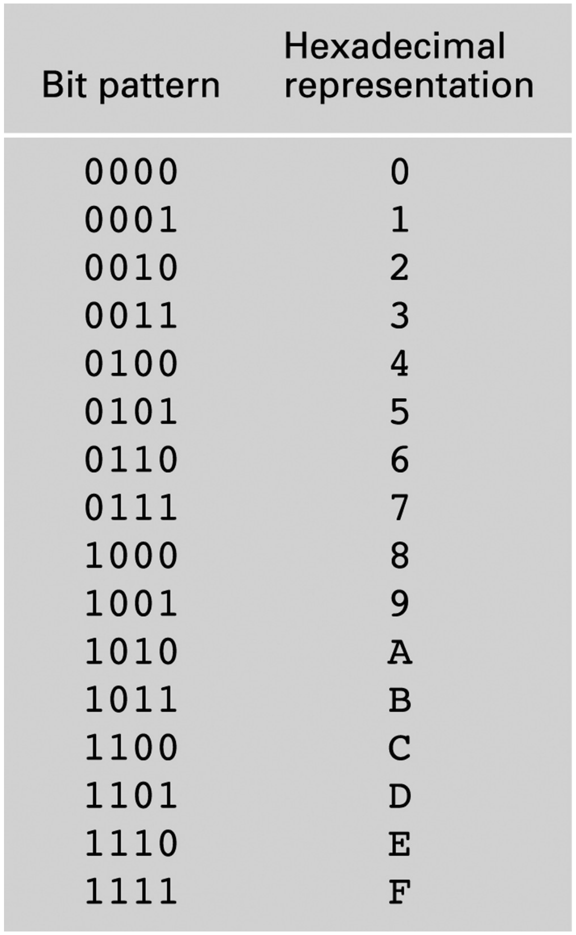
Figure 1.5 **Another way of constructing a flip-flop**



Hexadecimal Notation

- **Hexadecimal notation:** A shorthand notation for long bit patterns
  - Divides a pattern into groups of four bits each
  - Represents each group by a single symbol
- Example: 10100011 becomes A3

Figure 1.6 **The hexadecimal coding system**



Bit pattern	Hexadecimal representation
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

The Binary System

The traditional decimal system is based on powers of ten.

The Binary system is based on powers of two.

Figure 1.13 **The base ten and binary systems**

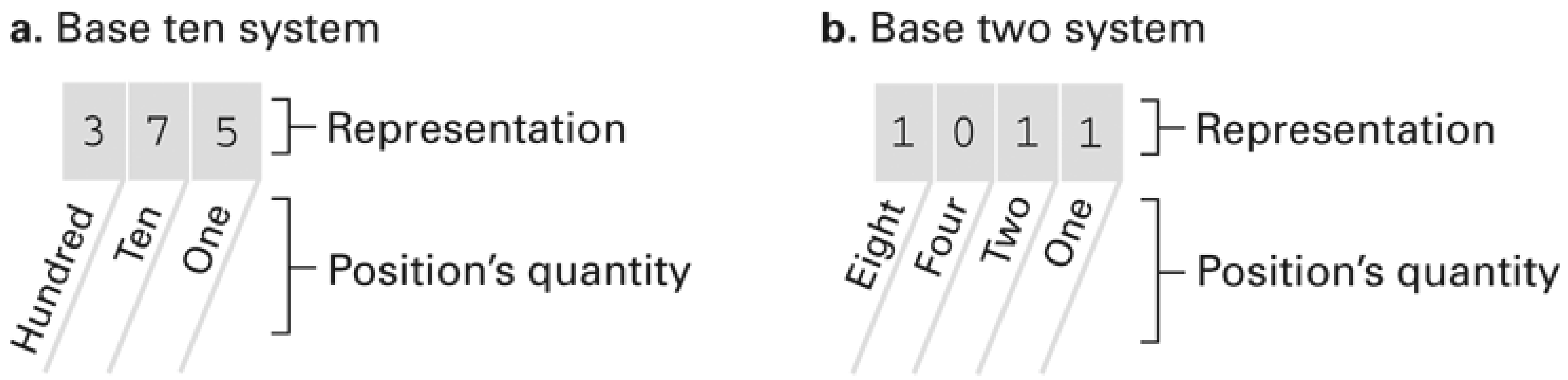


Figure 1.14 **Decoding the binary representation 100101**

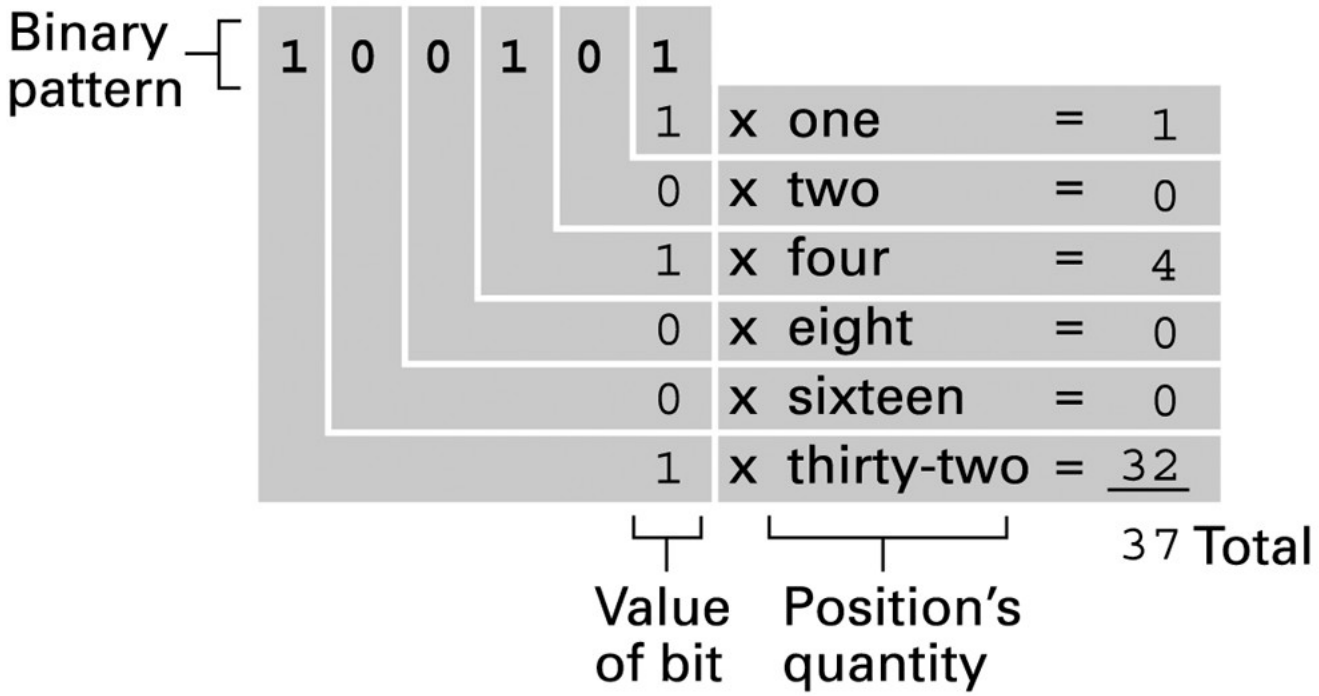
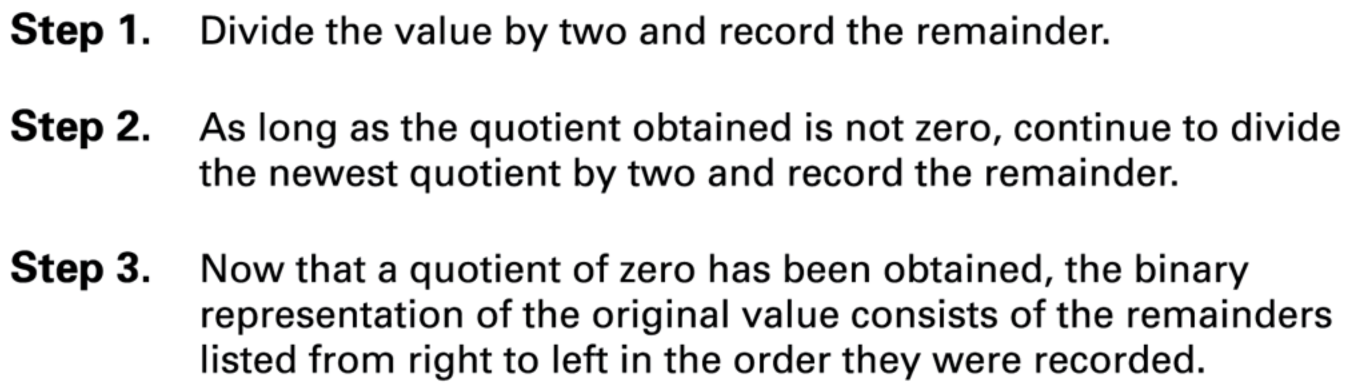


Figure 1.15 **An algorithm for finding the binary representation of a positive integer**



Step 1. Divide the value by two and record the remainder.

Step 2. As long as the quotient obtained is not zero, continue to divide the newest quotient by two and record the remainder.

Step 3. Now that a quotient of zero has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

Figure 1.16 **Applying the algorithm in Figure 1.15 to obtain the binary representation of thirteen**

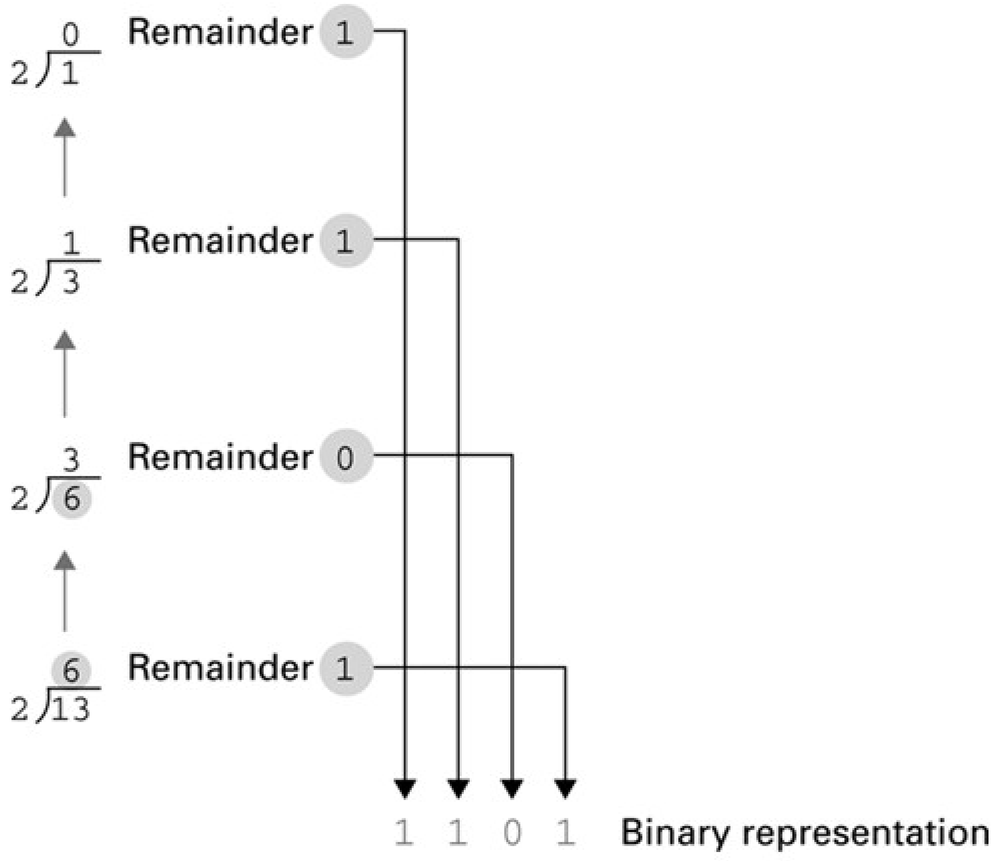
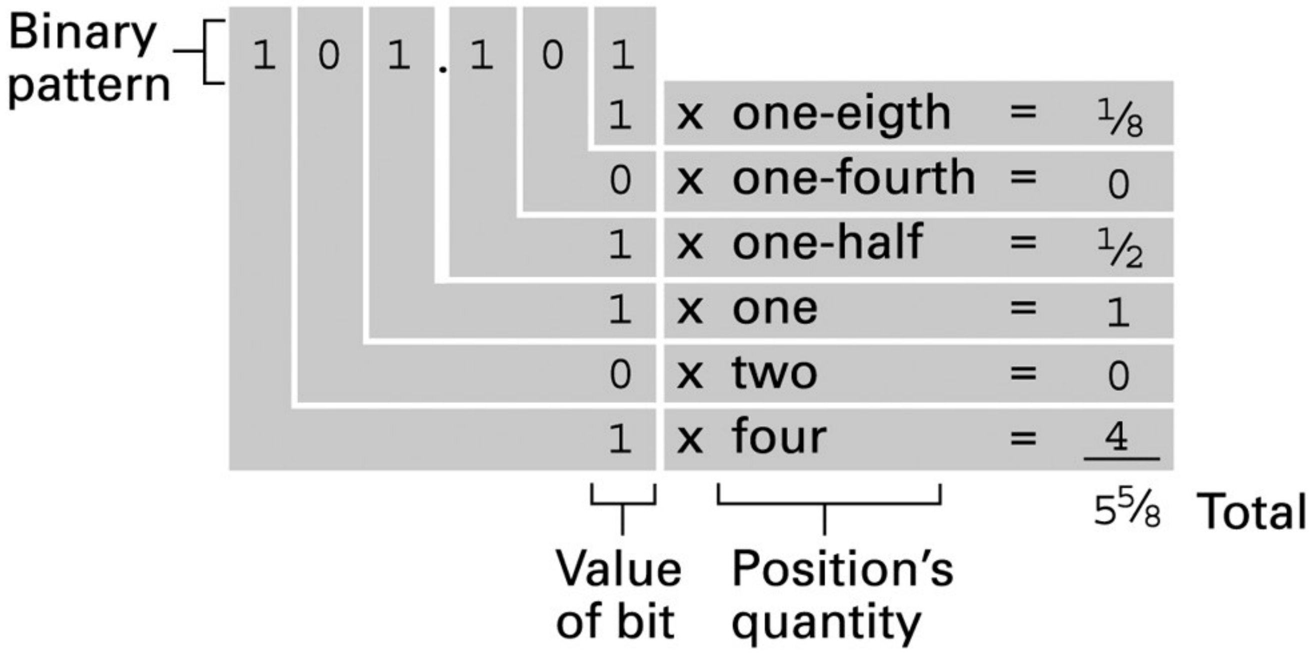


Figure 1.17 **The binary addition facts**

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

Figure 1.18 **Decoding the binary representation 101.101**

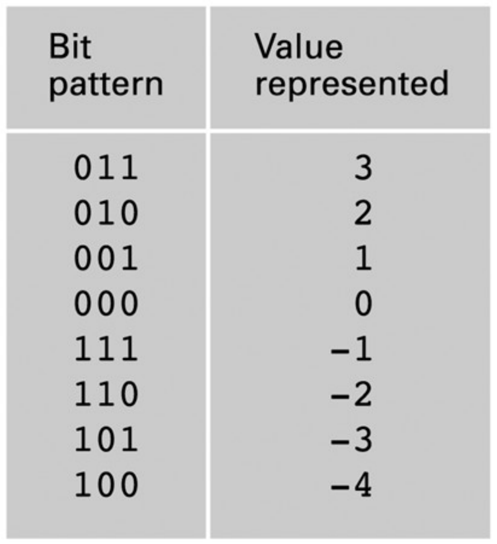


Storing Integers

- **Two's complement notation:** The most popular means of representing integer values
- **Excess notation:** Another means of representing integer values
- Both can suffer from overflow errors

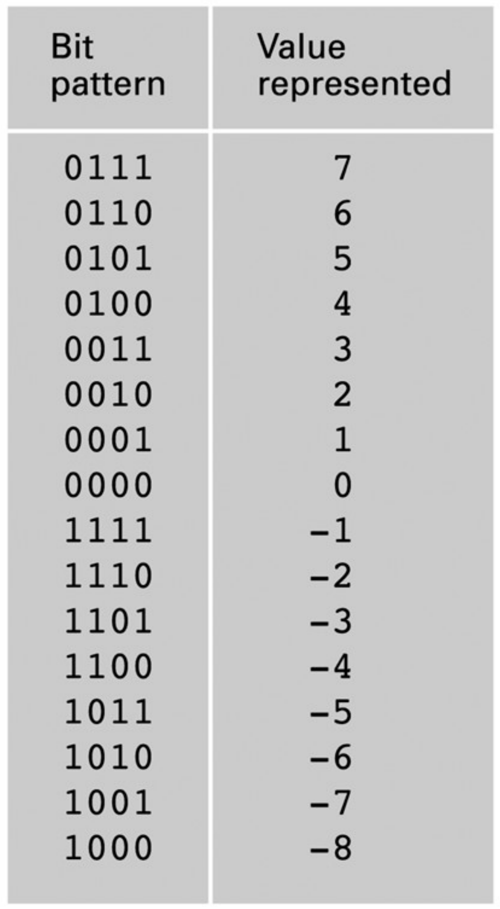
Figure 1.19 **Two's complement notation systems**

a. Using patterns of length three



Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

b. Using patterns of length four



Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Figure 1.20 **Coding the value -6 in two's complement notation using four bits**

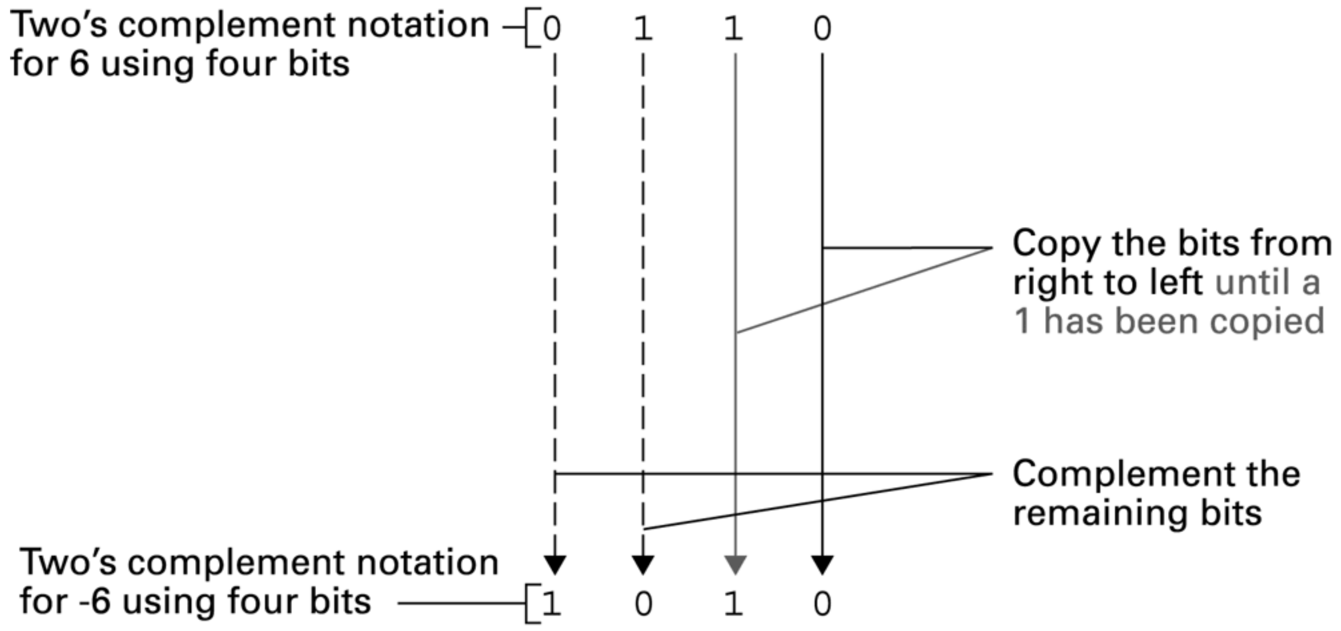
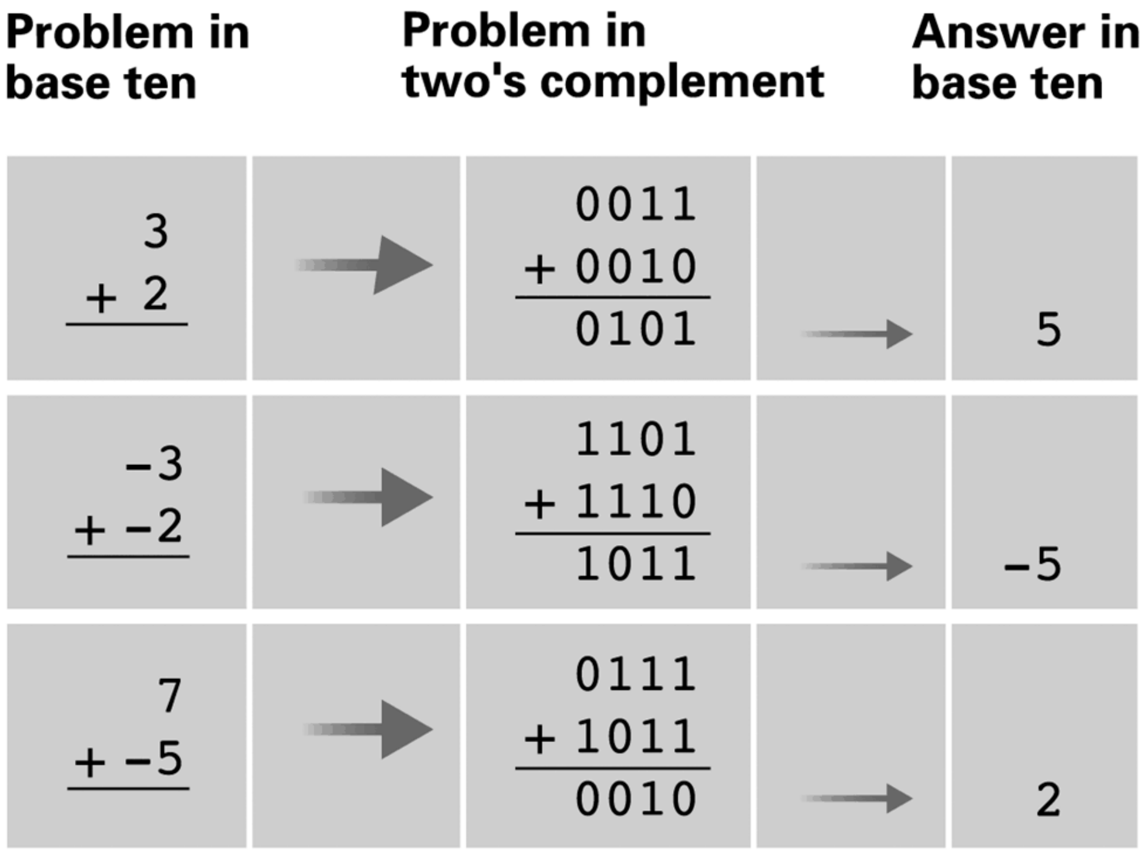

Two's complement notation for 6 using four bits
0 1 1 0
Copy the bits from right to left until a 1 has been copied
Complement the remaining bits
Two's complement notation for -6 using four bits
1 0 1 0

Figure 1.21 **Addition problems converted to two’s complement notation**

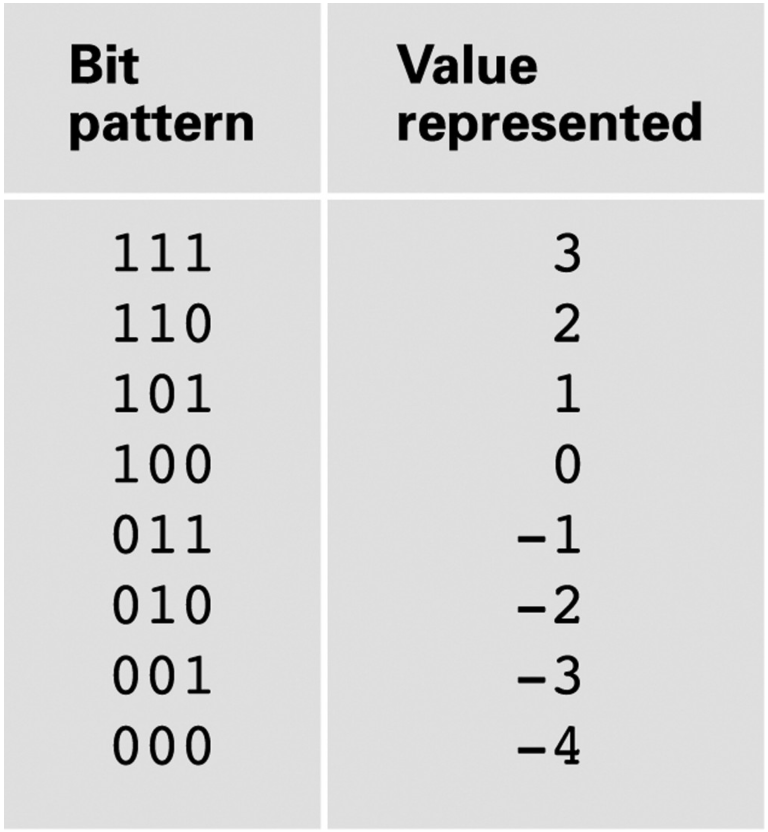


Problem in base ten	Problem in two's complement	Answer in base ten
$\begin{array}{r} 3 \\ +\ 2 \\ \hline \end{array}$	$\begin{array}{r} 0011 \\ +\ 0010 \\ \hline 0101 \end{array}$	5
$\begin{array}{r} -3 \\ +\ -2 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ +\ 1110 \\ \hline 1011 \end{array}$	−5
$\begin{array}{r} 7 \\ +\ -5 \\ \hline \end{array}$	$\begin{array}{r} 0111 \\ +\ 1011 \\ \hline 0010 \end{array}$	2

Figure 1.22 **An excess eight conversion table**

Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

Figure 1.23 **An excess notation system using bit patterns of length three**



Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

Storing Fractions

- **Floating-point Notation:** Consists of a sign bit, a mantissa field, and an exponent field.
- Related topics include
  - Normalized form
  - Truncation errors

Figure 1.24 **Floating-point notation components**

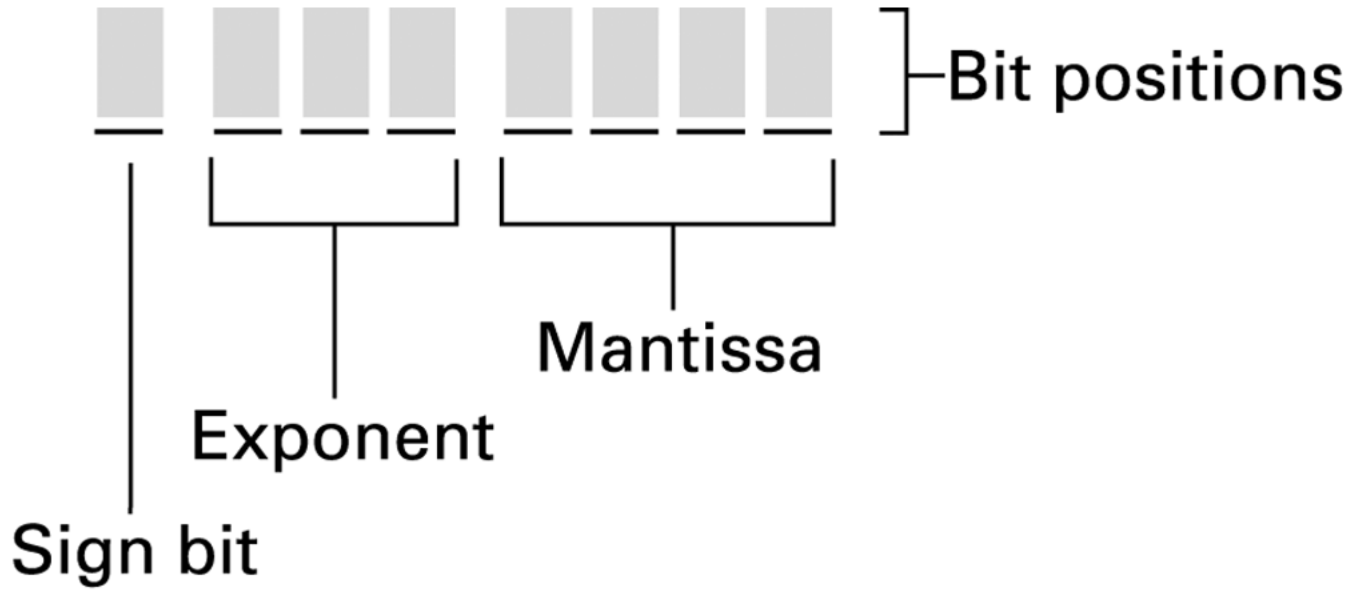
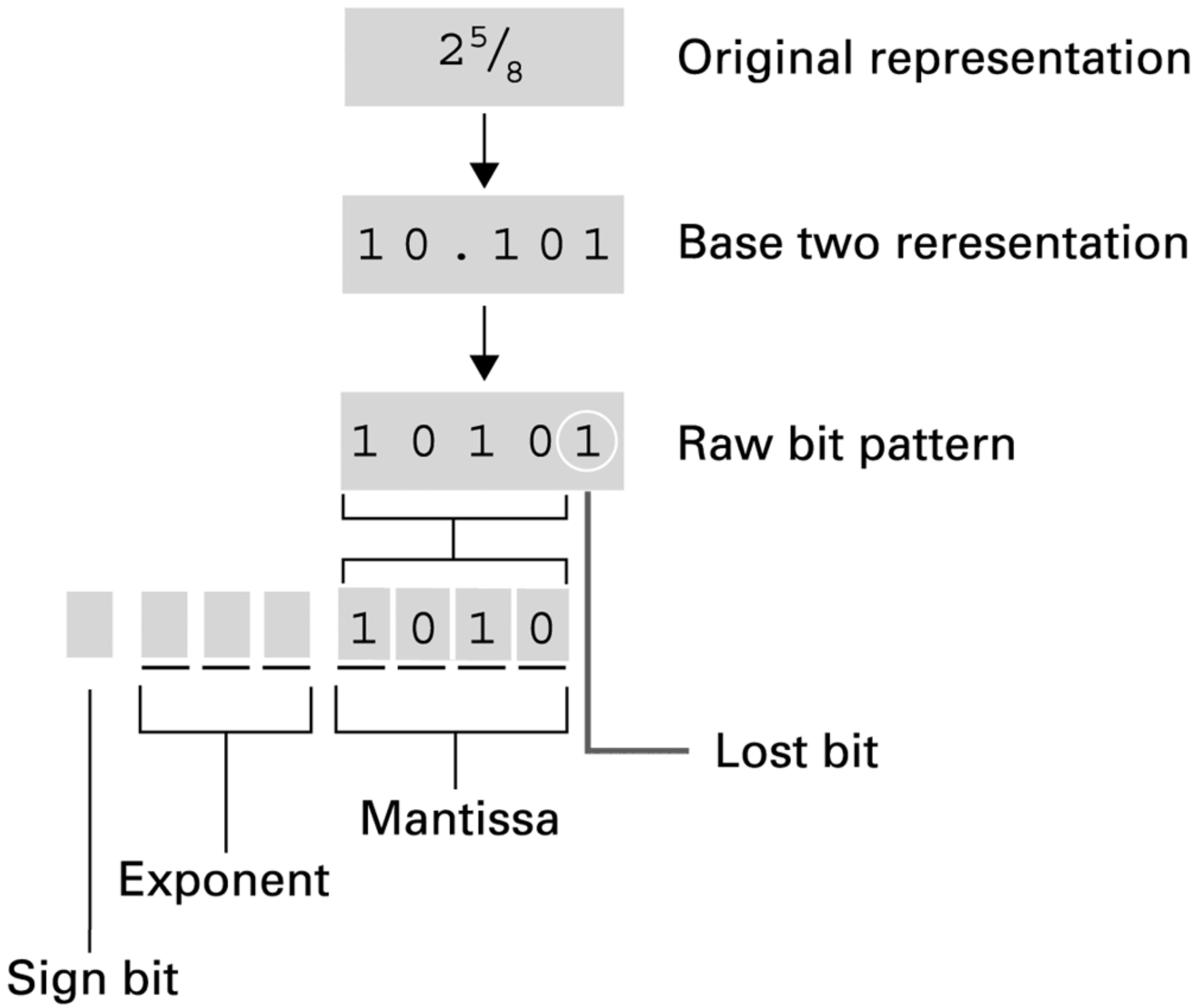


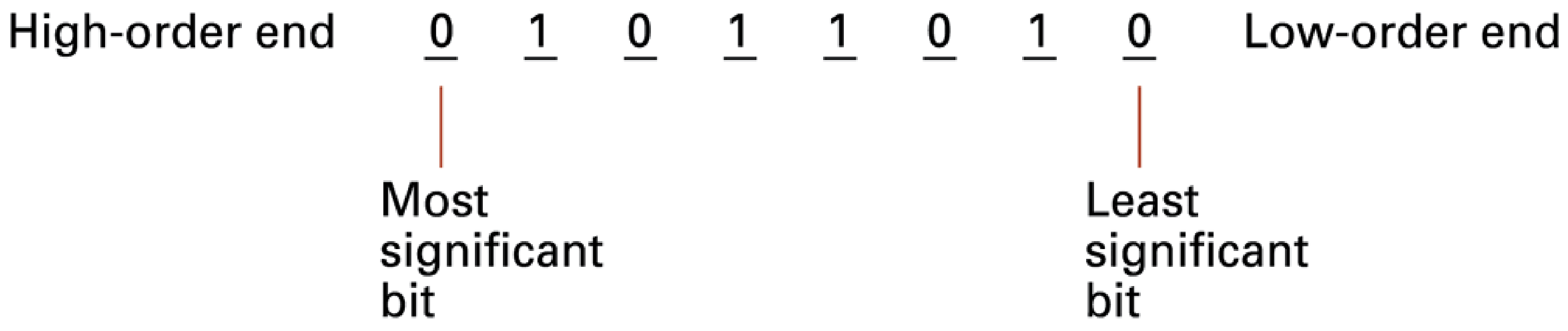
Figure 1.25 **Encoding the value 2 $\frac{5}{8}$**



Main Memory Cells

- **Cell:** A unit of main memory (typically 8 bits which is one **byte**)
  - **Most significant bit:** the bit at the left (high-order) end of the conceptual row of bits in a memory cell
  - **Least significant bit:** the bit at the right (low-order) end of the conceptual row of bits in a memory cell

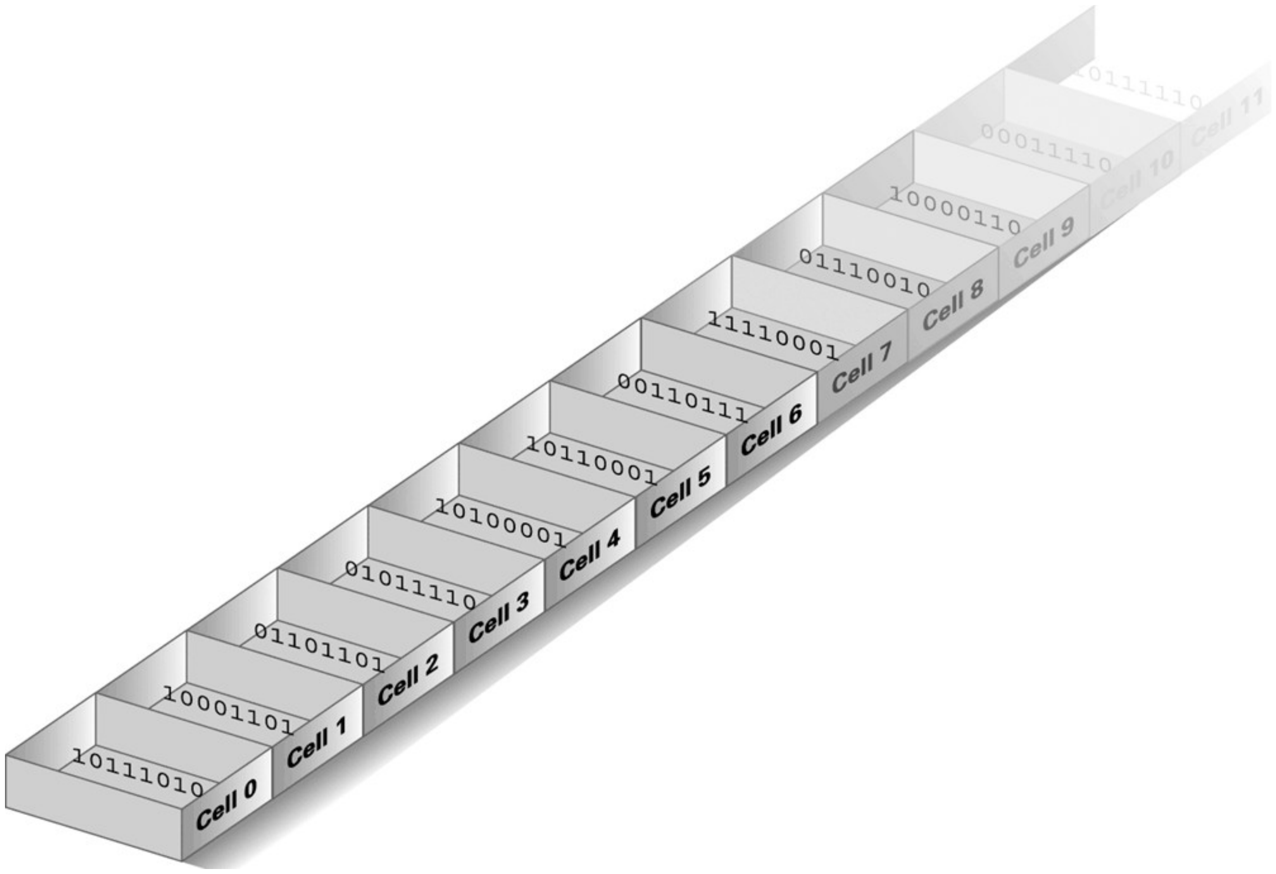
Figure 1.7 **The organization of a byte-size memory cell**



Main Memory Addresses

- **Address:** A “name” that uniquely identifies one cell in the computer’s main memory
  - The names are actually numbers.
  - These numbers are assigned consecutively starting at zero.
  - Numbering the cells in this manner associates an order with the memory cells.

Figure 1.8 **Memory cells arranged by address**



Memory Terminology

- **Random Access Memory (RAM):** Memory in which individual cells can be easily accessed in any order
- **Dynamic Memory (DRAM):** RAM composed of volatile memory

Measuring Memory Capacity

- **Kilobyte:** 2^{10} bytes = 1024 bytes
  - Example: 3 KB = 3 times1024 bytes
- **Megabyte:** 2^{20} bytes = 1,048,576 bytes
  - Example: 3 MB = 3 times 1,048,576 bytes
- **Gigabyte:** 2^{30} bytes = 1,073,741,824 bytes
  - Example: 3 GB = 3 times 1,073,741,824 bytes

Mass Storage

- Additional devices:
  - Magnetic disks
  - CDs
  - DVDs
  - Magnetic tape
  - Flash drives
  - Solid-state disks
- Advantages over main memory
  - Less volatility
  - Larger storage capacities
  - Low cost
  - In many cases can be removed

Figure 1.9 **A magnetic disk storage system**

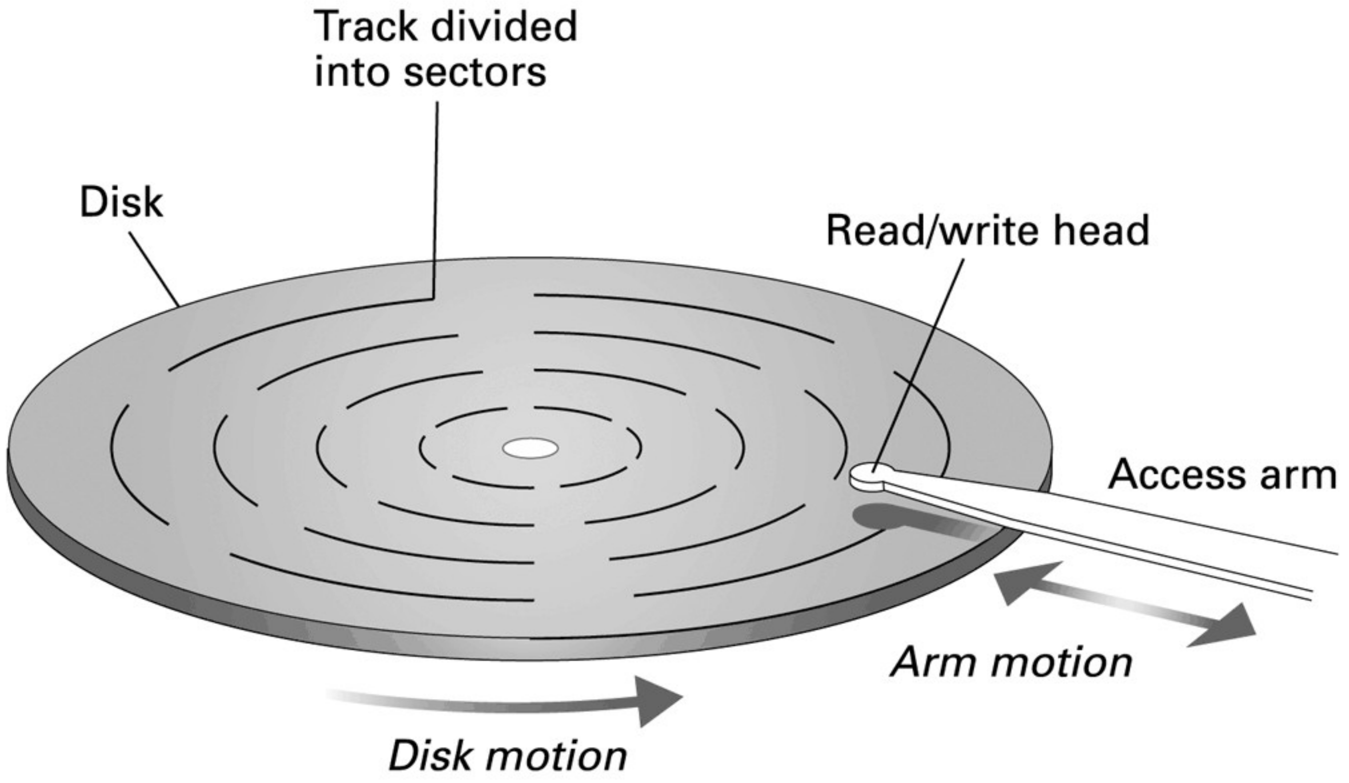
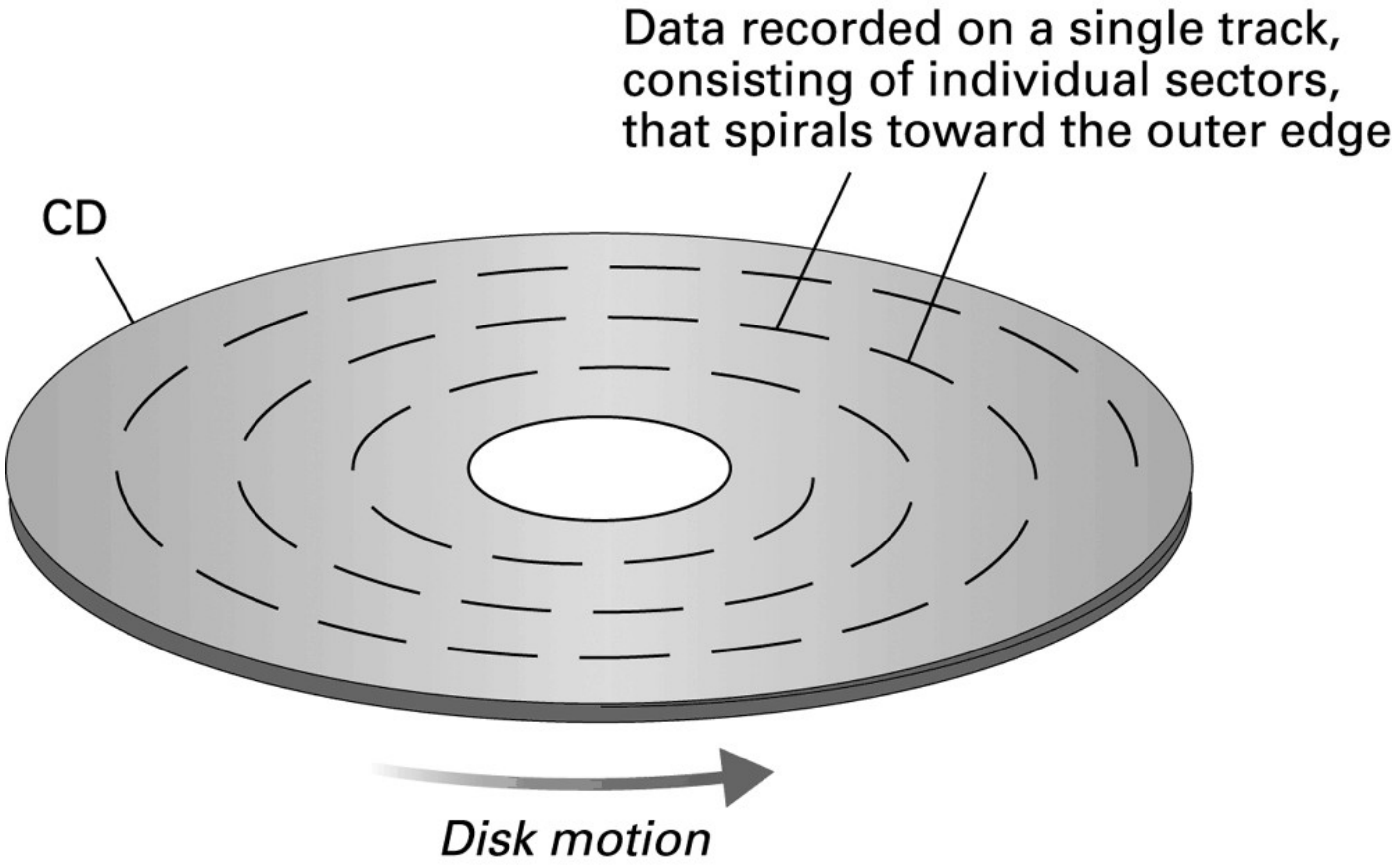


Figure 1.10 **CD storage**


Data recorded on a single track, consisting of individual sectors, that spirals toward the outer edge
CD
Disk motion

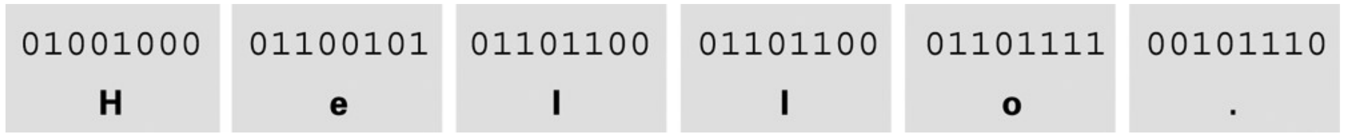
Flash Drives

- **Flash Memory** – circuits that traps electrons in tiny silicon dioxide chambers
- Repeated erasing slowly damages the media
- Mass storage of choice for:
  - Digital cameras
  - Smartphones
- **SD Cards** provide GBs of storage

Representing Text

- **Each character (letter, punctuation, etc.) is assigned a unique bit pattern.**
  - **ASCII**: Uses patterns of 7-bits to represent most symbols used in written English text
  - ISO developed a number of 8 bit extensions to ASCII, each designed to accommodate a major language group
  - **Unicode**: Uses patterns up to 21-bits to represent the symbols used in languages world wide, 16-bits for world’s commonly used languages

Figure 1.11 **The message "Hello." in ASCII or UTF-8 encoding**



01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

Representing Numeric Values

- **Binary notation**: Uses bits to represent a number in base two
- Limitations of computer representations of numeric values
  - Overflow: occurs when a value is too big to be represented
  - Truncation: occurs when a value cannot be represented accurately

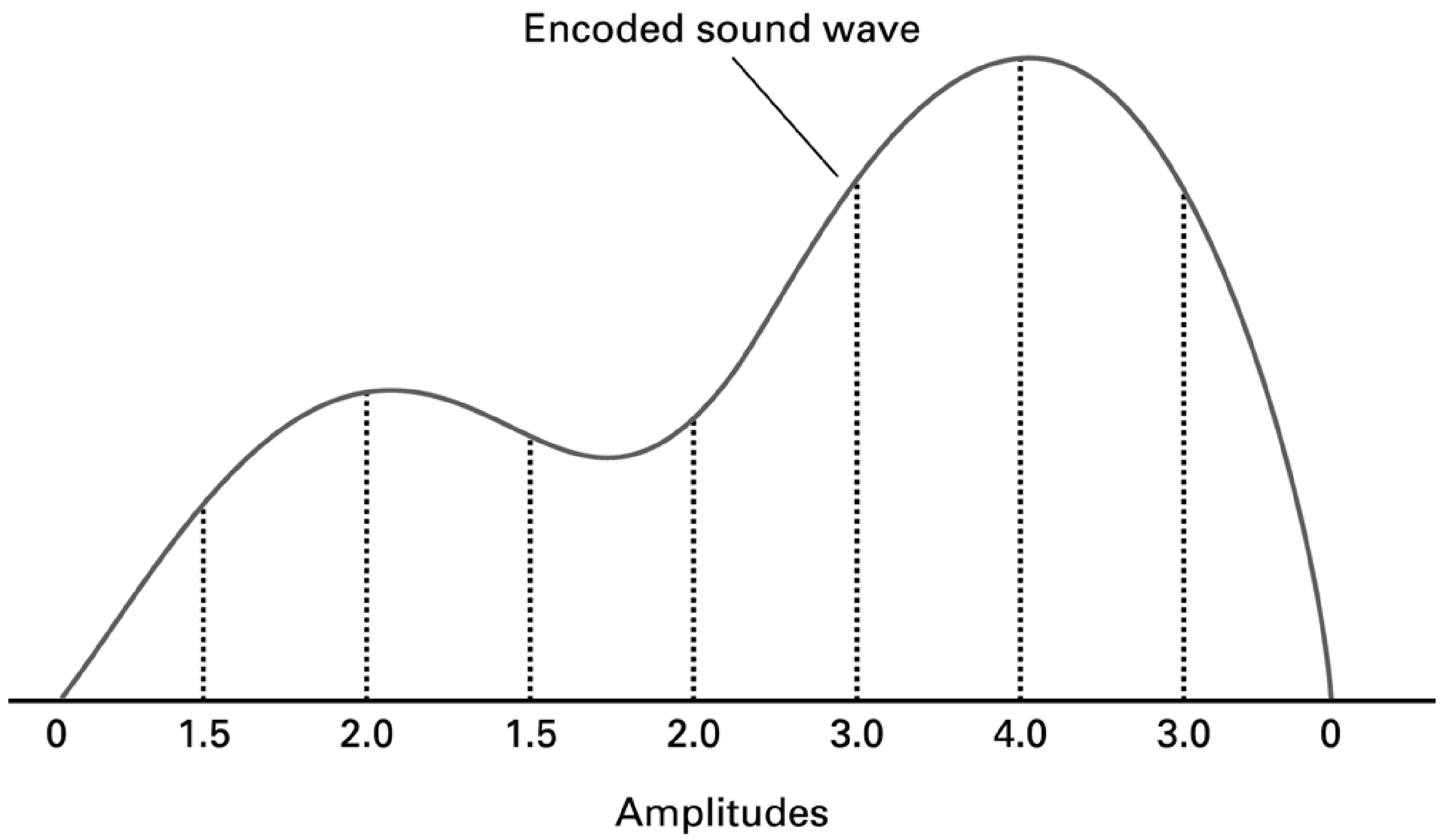
Representing Images

- Bit map techniques
  - Pixel: short for “picture element”
  - RGB
  - Luminance and chrominance
- Vector techniques
  - Scalable
  - TrueType and PostScript

Representing Sound

- Sampling techniques
  - Used for high quality recordings
  - Records actual audio
- MIDI
  - Used in music synthesizers
  - Records “musical score”

Figure 1.12 **The sound wave represented by the sequence 0, 1.5, 2.0, 1.5, 2.0, 3.0, 4.0, 3.0, 0**



Data and Programing

A ***programming language*** is a computer system created to allow humans to precisely express algorithms using a higher level of abstraction.

Getting Started with Python

- ***Python:*** a popular programming language for applications, scientific computation, and as an introductory language for students
- Freely available from **www.python.org**
- Python is an *interpreted language*
  - Typing:

```
print('Hello, World!')
```

  - Results in:

```
Hello, World!
```

Variables

- **Variables:** name values for later use
- Analogous to mathematic variables in algebra

```
s = 'Hello, World!'
print(s)

my_integer = 5
my_floating_point = 26.2
my_Boolean = True
my_string = 'characters'
my_integer = 0xFF
```

Operators and Expressions

```
print(3 + 4)          # Prints 7
print(5 – 6)      # Prints -1
print(7 * 8)      # Prints 56
print(45 / 4)         # Prints 11.25
print(2 ** 10)        # Prints 1024

s = 'hello' + 'world'
s = s * 4
print(s)
```

Currency Conversion

```
# A converter for currency exchange.

USD_to_GBP = 0.66   # Today's exchange rate
GBP_sign = '\u00A3' # Unicode value for £
dollars = 1000      # Number dollars to convert

# Conversion calculations
pounds = dollars * USD_to_GBP

# Printing the results
print('Today, $' + str(dollars))
print('converts to ' + GBP_sign + str(pounds))
```

Debugging

- *Syntax errors*

```
print(5 +)
SyntaxError: invalid syntax

pront(5)
NameError: name 'pront' is not defined
```

- *Semantic errors*
  - Incorrect expressions like

```
total_pay = 40 + extra_hours * pay_rate
```

- *Runtime errors*
  - Unintentional divide by zero

Data Compression

- Lossy versus lossless
- Run-length encoding
- Frequency-dependent encoding

  (Huffman codes)
- Relative encoding
- Dictionary encoding (Includes adaptive dictionary encoding such as LZW encoding.)

Compressing Images

- GIF: Good for cartoons
- JPEG: Good for photographs
- TIFF: Good for image archiving

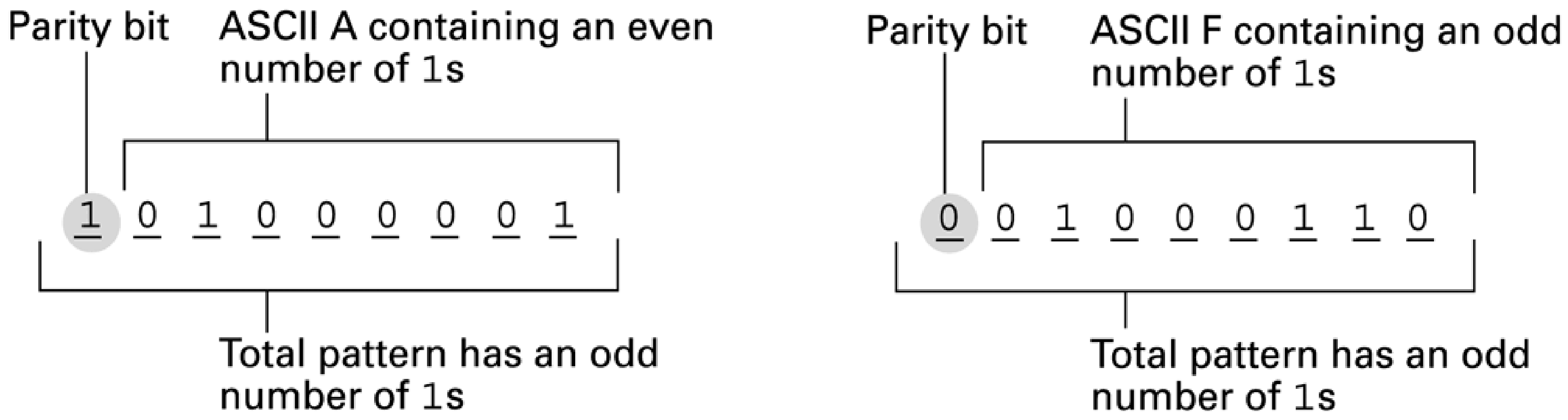
Compressing Audio and Video

- MPEG
  - High definition television broadcast
  - Video conferencing
- MP3
  - Temporal masking
  - Frequency masking

Communication Errors

- Parity bits (even versus odd)
- Checkbytes
- Error correcting codes

Figure 1.26 **The ASCII codes for the letters A and F adjusted for odd parity**

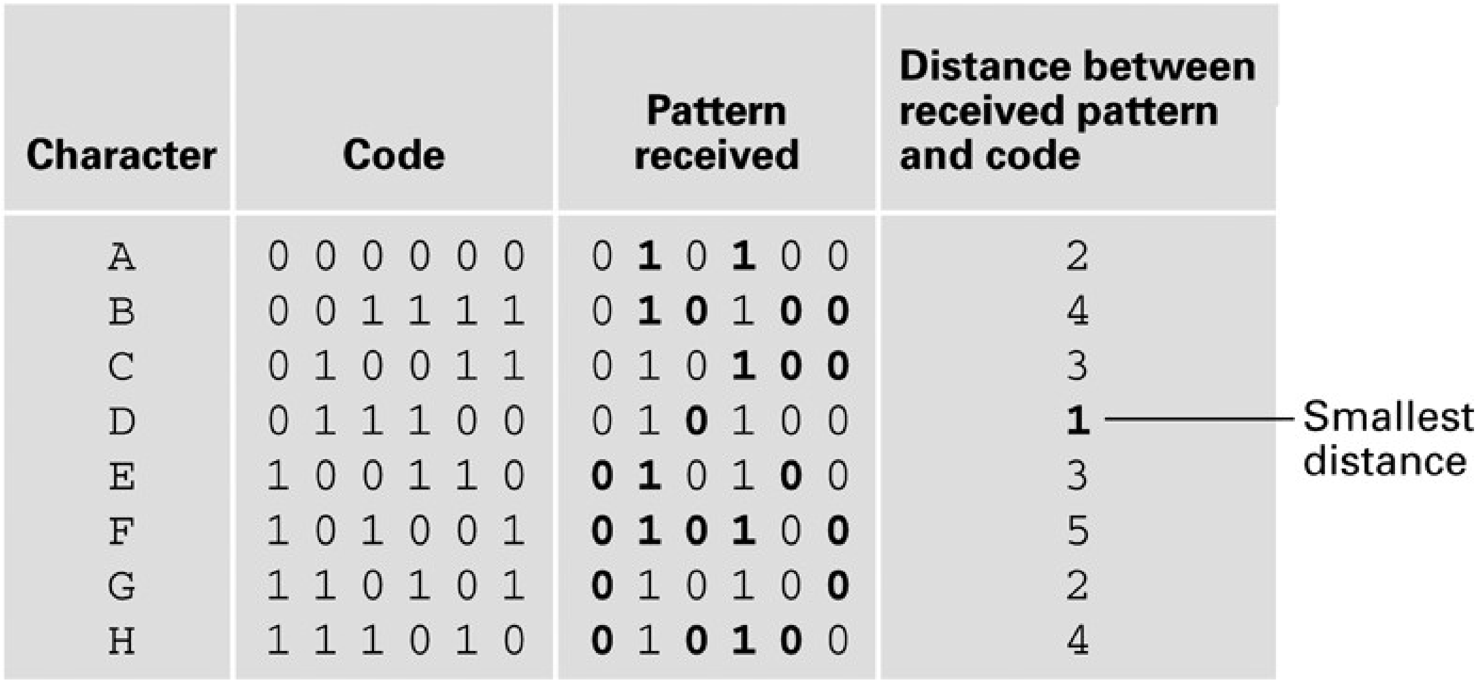


Parity bit
ASCII A containing an even number of 1s
1 0 1 0 0 0 0 0 1
Total pattern has an odd number of 1s
Parity bit
ASCII F containing an odd number of 1s
0 0 1 0 0 0 1 1 0
Total pattern has an odd number of 1s

Figure 1.27 **An error-correcting code**

Symbol	Code
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

Figure 1.28 **Decoding the pattern 010100 using the code in Figure 1.27**



Character	Code	Pattern received	Distance between received pattern and code
A	0 0 0 0 0 0	0 **1** 0 **1** 0 0	2
B	0 0 1 1 1 1	0 **1** **0** 1 **0** **0**	4
C	0 1 0 0 1 1	0 1 0 **1** **0** **0**	3
D	0 1 1 1 0 0	0 1 **0** 1 0 0	**1**
E	1 0 0 1 1 0	**0** **1** 0 1 **0** 0	3
F	1 0 1 0 0 1	**0** **1** **0** **1** 0 **0**	5
G	1 1 0 1 0 1	**0** 1 0 1 0 **0**	2
H	1 1 1 0 1 0	**0** 1 **0** **1** **0** 0	4

Smallest distance

End of Chapter