

Chapter 1: Data Storage

**Computer Science: An Overview
Twelfth Edition**

**by
J. Glenn Brookshear
Dennis Brylow**

Data Storage

- 1.1 Bits and Their Storage
- 1.2 Main Memory
- 1.3 Mass Storage
- 1.4 Representing Information as Bit Patterns
- 1.5 The Binary System

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

Bits and Bit Patterns

- Computers represent data and instructions with patterns of bits.
- In most computer documentation, 8 contiguous bits are called a **byte**. A bit holds a 0 or a 1, possibly representing the on/off (or high/low) condition of a switch.

Bits and Bit Patterns (cont.)

Ques: How many patterns can be formed with a single bit?

Bits and Bit Patterns (cont.)

Ans: With one bit there are two possible patterns.

0 and 1 (or "off" and "on", or "false" and "true" or "low" and "high").

Bits and Bit Patterns (cont.)

Ques: How many patterns can be formed with two bits?

Bits and Bit Patterns (cont.)

Ans:

0 0 0

0 0 1

0 1 0

0 1 1

1 0 0

1 0 1

1 1 0

1 1 1

Bits and Bit Patterns (cont.)

Ques: Is the pattern 0 1 different from the pattern 1 0?

Bits and Bit Patterns (cont.)

Ans: Yes — the position of a bit matters.

Bits and Bit Patterns (cont.)

Ques: How many patterns can be formed with three bits?

Bits and Bit Patterns (cont.)

Ans:

0 0 0

0 0 1

0 1 0

0 1 1

1 0 0

1 0 1

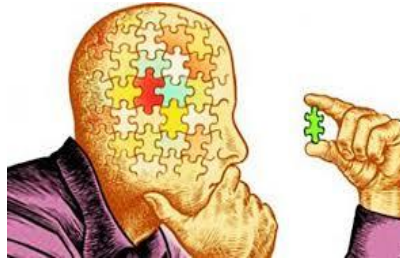
1 1 0

1 1 1

8 patterns.

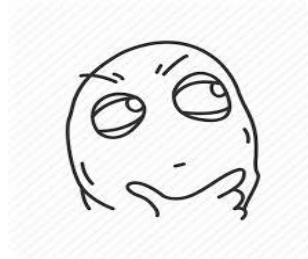
Bits and Bit Patterns (cont.)

It looks like the number of patterns that can be formed with N bits is greater than the number of bits.



What do you think?

Ans: Yes — much greater.



This simple fact is of profound importance to computer science.

Bits and Bit Patterns (cont.)

Ques: How many patterns can be formed from three bits?

Ans: 8 patterns can be formed from three bits.

Ques: How many patterns can be formed from **N** bits?

Ans: Twice the number of patterns that can be formed from (N-1) bits.

Bits and Bit Patterns (cont.)

The list of patterns for three bits has 8 lines (patterns).

To form the list of patterns for 4 bits,
make two copies of the list for 3 bits.

This gives you 16 lines.

Each line is made unique by prefixing the first half with "0"
and the second half with "1".

Bits and Bit Patterns (cont.)

- The trick can be repeated as many times as you like.
- Adding one more bit doubles the number of patterns.
- The table shows the number of patterns for 1, 2, 3 and 4 bits.

Number of Bits	Number of Patterns	Number of Patterns as power of two
1	2	2^1
2	4	2^2
3	8	2^3
4	16	2^4

Bits and Bit Patterns (cont.)

- How many patterns with 5 bits?
- Make two copies of the 4-bit patterns (16 patterns per copy).
- Make the patterns unique by prefixing "0" to the first 16 patterns and "1" to the second 16.
- You now have $16 \times 2 = 2^5$ unique patterns.
- This demonstrates the following:

Number of possible patterns of N bits = 2^N

Bits and Bit Patterns (cont.)

How many patterns can be formed with 10 bits? Use the formula:

$$2^{10} = 1024$$

This number (i.e. 1024) occurs often in computer science.

1024 bytes is called a **kilobyte**, abbreviated **K** and pronounced "Kay".

Bits and Bit Patterns (cont.)

- Ques: In the past, some computers used 16 bits to form memory addresses. Assuming no special tricks (which such limited machines often used), how many bytes maximum could be held in main storage?
- Ans: 64K bytes:
- $2^{16} = 2^{(6 + 10)} = 2^6 \times 2^{10} = 64K$

More about patterns

- Many calculations involving bit patterns use the following familiar fact of arithmetic.

- $2^{(N+M)} = 2^N \times 2^M$

- The number of patterns that can be formed with 10 or more bits are usually expressed as multiples of 1024 ($= 2^{10}$) or in "Megs" ($= 2^{20}$).

For example, how many patterns can be formed from 24 bits?

$$2^{24} = 2^4 \times 2^{20} = 16 \text{ Meg}$$

- The power of two (24) splits into a small part (2^4) and a part that has a name ($2^{20} = \text{Meg}$).
- This is a useful trick you can use to amaze your friends and impress employers.

Bits and Bit Patterns (cont.)

- Some audio cards use 12 bits to represent the sound level at an instant in time (12 bits per sample). How many signal levels are represented?
 - **$2^{12} = 2^2 \times 2^{10} = 4K$ levels**

Pattern names

- Consider the following pattern:

0010100010101010

- It is not easy to work with. It is convenient to break bit patterns into 4-bit groups (called **nibbles**):

- 0010 1000 1010 1100

- There are 16 (= 2^4) possible patterns in a nibble. Each pattern has a name, as seen in the table.

Hexadecimal Names

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

- You might be tempted to call those 4-bit patterns "binary numbers". Resist that temptation. The bit patterns in computer main memory are used for very many purposes. Representing integers is just one of them. The fundamental concept is "bit patterns". Don't confuse this concept with one of its many uses: "representing numbers".
- The above bit pattern can be written using the pattern names:
 - 0010 1000 1010 1100 = 28AC
- Bits are grouped into nibbles starting at the right. Then each nibble is named. This method of giving names to patterns is called **hexadecimal**.

- **QUES:** Name the following patterns:
 - 0001 0001
 - 0011 1001
 - 1011 1111
 - 0100 0110
 - 0000 0000

- **Ans:**
- 0001 0001 11
- 0011 1001 39
- 1011 1111 BF
- 0100 0110 46
- 0000 0000 00 (be sure to show both zeros)

Hex practice

- If there are not enough bits at the left to form a complete group of four, add zero bits *to the left*, (but be sure that it is clear by context how many bits you are describing). For example:
- $1010000010000010 = 1010\ 0000\ 1000\ 0010 = A082$ Another example:
- $10100110101111 = 10\ 1001\ 1010\ 1111 = 0010\ 1001\ 1010\ 1111 = 29AF$

Hex practice (cont.)

- Usually '0x' is placed at the front of a pattern name to show that it is a hexadecimal pattern name:

0x0010 = 0000 0000 0001 0000

0xFACE = 1111 1010 1100 1110

Hex practice (cont.)

- Adding zeros to the left of a pattern creates a new pattern. The new pattern has its own name. $0x0 = 0000$ is a different pattern than $0x00 = 0000\ 0000$. Sadly, people are not consistent about this, and depending on context, both patterns might be called "0x0".
- Keep in mind that *hexadecimal pattern names are used by humans for talking about bit patterns*. Inside the computer there are only bits and their patterns. Hexadecimal is used in books and documents (outside the computer) to describe these bit patterns.

Hex practice (cont.)

- **QUES:** Name the following patterns; include 0x in the name:
- 0000 1010 0001 0001
- 0001 0010 1001 1010
- 1111 1010 1101 1110
- 0011 0110 1100 0000
- 0000 0000 0000 0000

Hex practice (cont.)

- **Answer:**
- 0000 1010 0001 0001 0x0A11
- 0001 0010 1001 1010 0x129A
- 1111 1010 1101 1110 0xFADE
- 0011 0110 1100 0000 0x36C0
- 0000 0000 0000 0000 0x0000 (be sure to show all zeros)

Octal

- Sometimes documentation describes bit patterns in groups of three. Three-bit groups are named using the first eight pattern names of hexadecimal. This method is called **octal notation**.
- A bit pattern can be named using hexadecimal names, octal names, or many other notations.
- Example:
 - $01101010 = 01\ 101\ 010 = 152$ (octal)
 - $01101010 = 0110\ 1010 = 0x6A$ (hex)

Octal

- Octal is awkward to use with 8-bit bytes. Bytes don't evenly split into octal pattern names. But you should know about it. Historically, some computer documentation used octal pattern names. Also, in several programming languages (C and Java among them) octal notation is indicated by a leading zero:
- 0152 (octal) = 001 101 010
- 0x152 (hex) = 0001 0101 0010
- 152 (decimal) = 10011000

Octal

- When the number of bits is not a multiple of three it is conventional to add zero bits to the left, and then to name the pattern as usual. This happens frequently because computer memory is organized into 8-bit bytes, which do not divide evenly into groups of three.

Octal

- **QUES:** What is the OCTAL pattern name of the following bit patterns:
- 111 010 001
- 100 011 010
- 011011110
- 11000000

- **Ans:**
- 111 010 001 0721
- 100 011 010 0432
- 011011110 0336
- 11000000 0300

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

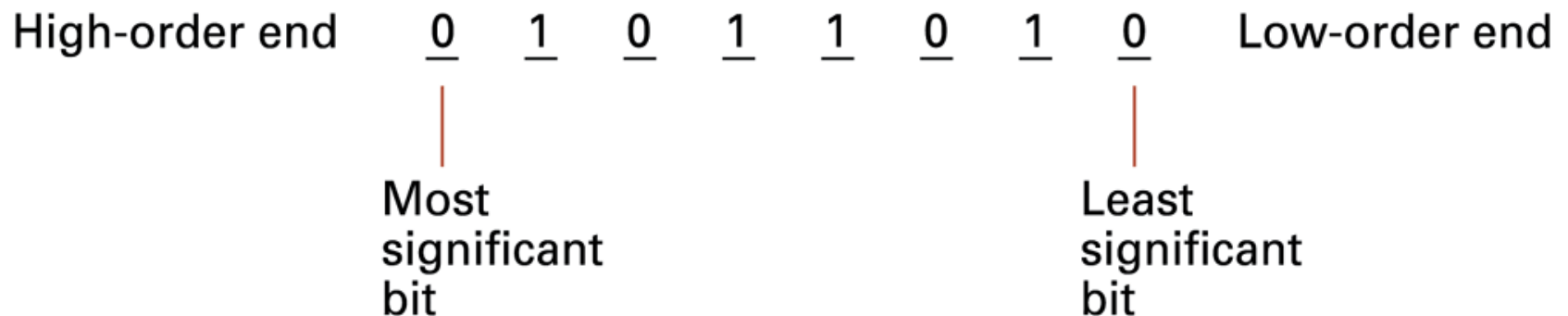
Figure 1.6 The hexadecimal coding system

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

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

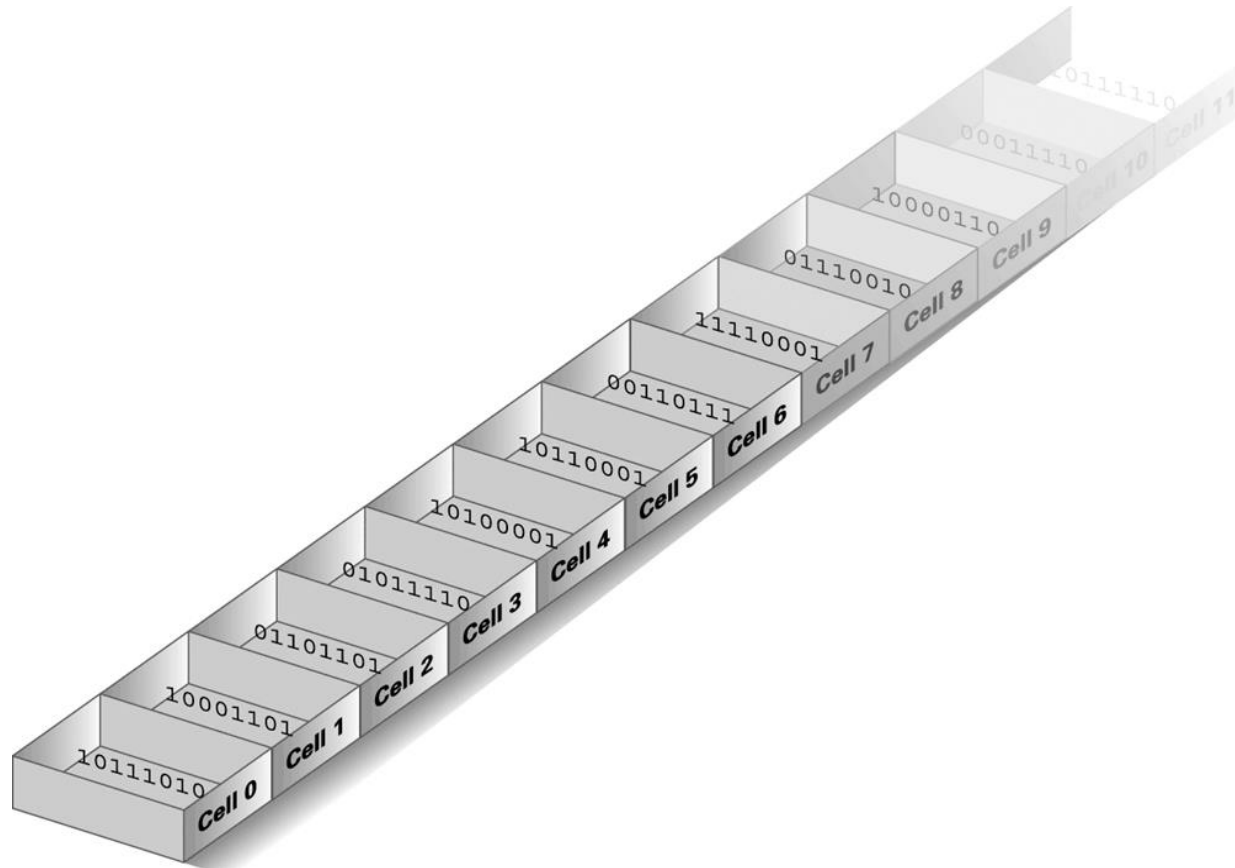
Figure 1.7 The organization of a byte-size memory cell



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
- Flip-flops is a means of storing bits
- RAM in most modern computers is analogous,
but more complex technologies provide greater miniaturization and faster response time.

- Many of these technologies store bits as tiny electric charges that dissipate quickly.
- Thus these devices require additional circuitry - refresh circuit, that repeatedly replenishes the charges many times a second. Therefore, they are called dynamic memory, or - Dynamic RAM (DRAM).
- **Dynamic Memory (DRAM):** RAM composed of volatile memory
-
- **SDRAM - Synchronous DRAM,**
 - DRAM that applies additional techniques to decrease the time needed to retrieve the contents from its memory cells.

Measuring Memory Capacity

- **Bit (Binary Digit):** A binary digit is logical 0 and 1
- **Nibble:** A group of 4 bits is called nibble.
- **Byte:** A group of 8 bits is called a **byte**. A byte is the smallest unit, which can represent a data item or a character.
- **Word:** is a group of fixed number of bits processed by a computer's CPU in one go
- (as a unit),
 - Length of a word is called **word-size or word length**. It may be as small as 8 bits or may be as long as 96 bits. A computer stores the information in the form of computer words. These days, typically **32 bits** or **64 bits**. Data bus size, instruction size, address size are usually multiples of the word size.
- **Kilobyte:** 2^{10} bytes = 1024 bytes
 - Example: 3 KB = 3 times 1024 bytes
- **Megabyte:** 2^{20} bytes = 1,048,576 bytes
 - Example: 3 MB = 3 times 1,048,576 bytes
- **Gigabyte (GB):** 2^{30} bytes = 1,073,741,824 bytes
 - Example: 3 GB = 3 times 1,073,741,824 bytes
- **TeraByte (TB):** 1 TB = 1024 GB
- **PetaByte (PB):** 1 PB = 1024 TB

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

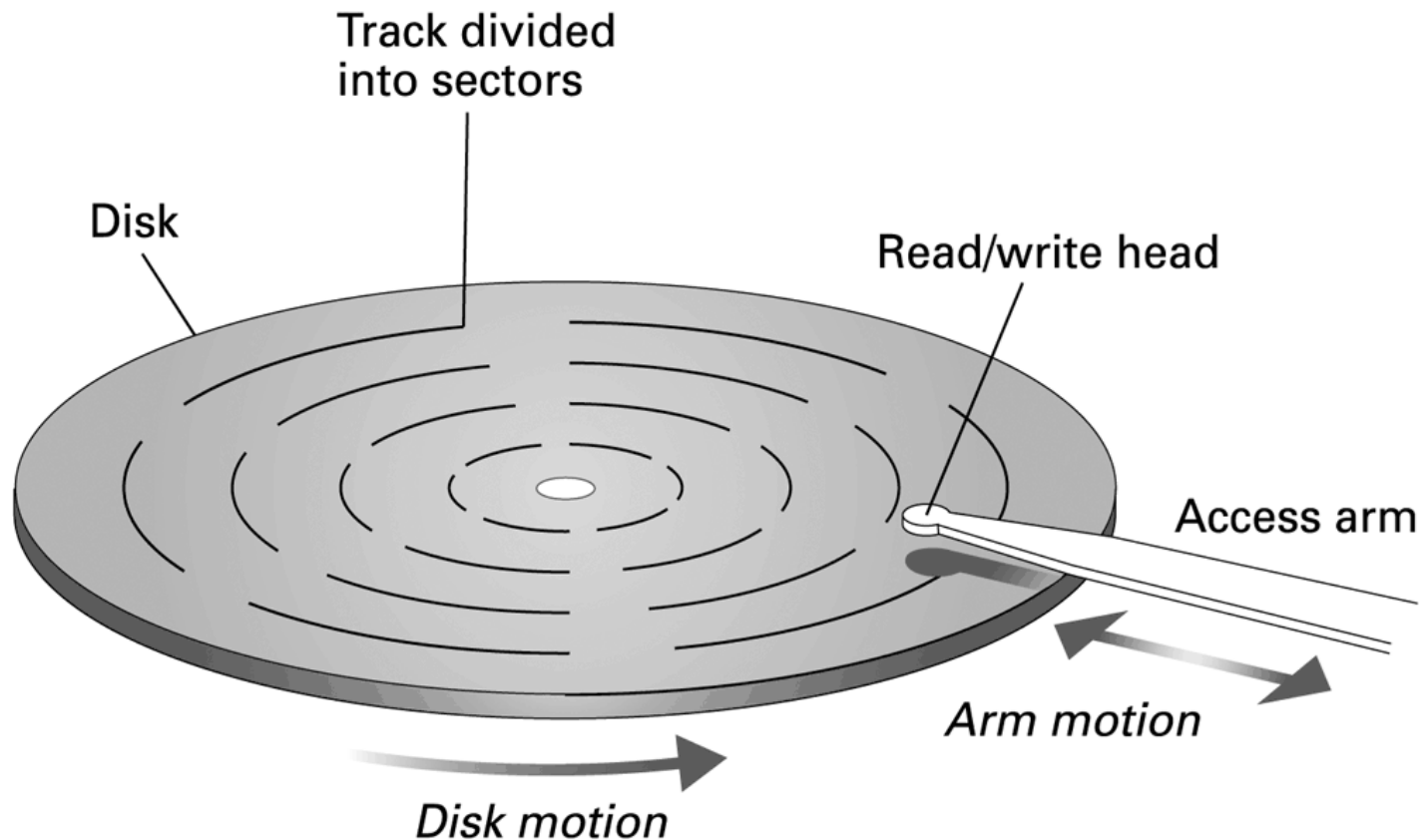
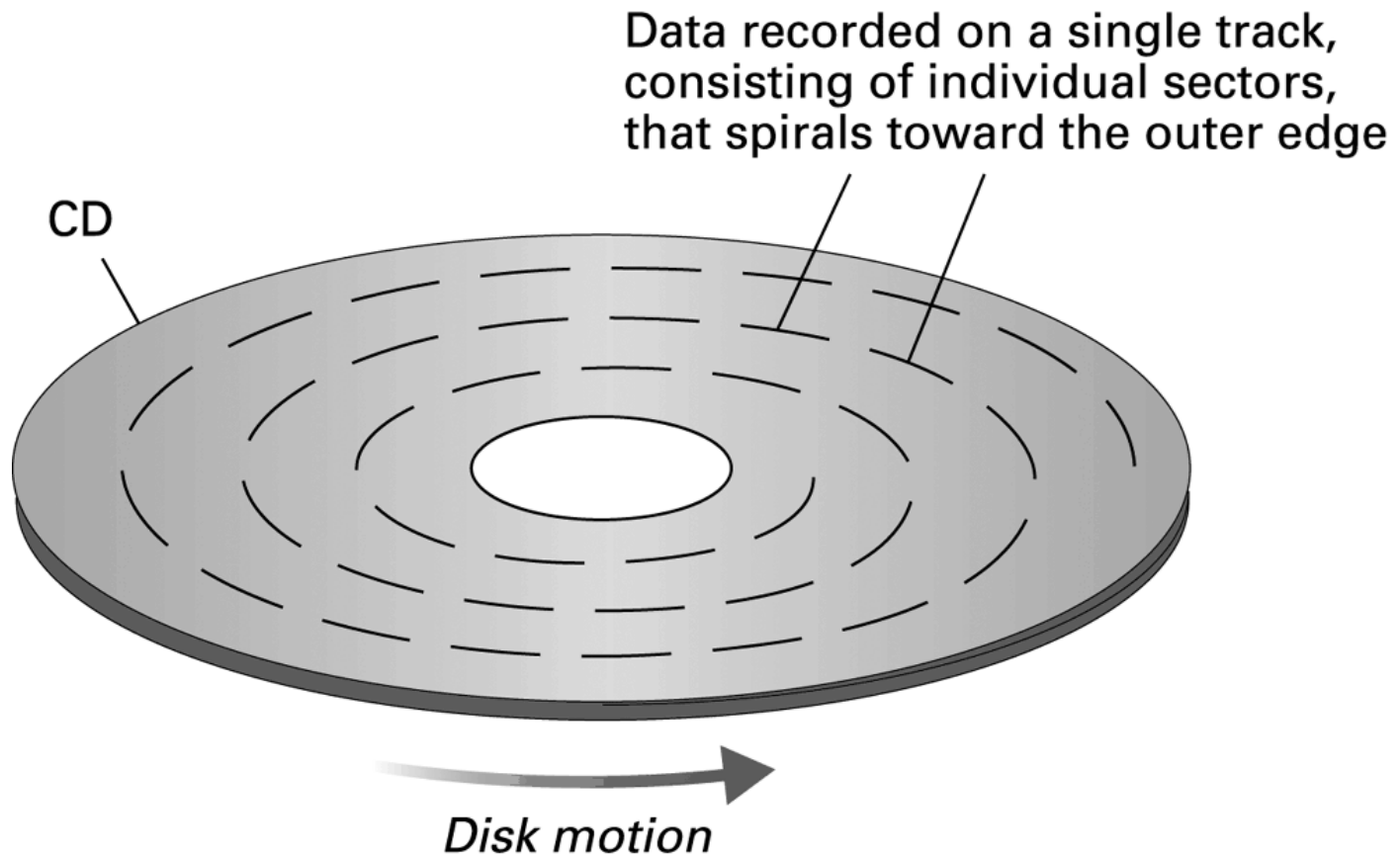


Figure 1.10 CD storage



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

Table 1.1

<i>Character</i>	<i>ASCII</i>	<i>EBCDIC</i>	<i>Character</i>	<i>ASCII</i>	<i>EBCDIC</i>
A	0100 0001	1100 0001	N	0100 1110	1101 0101
B	0100 0010	1100 0010	O	0100 1111	1101 0110
C	0100 0011	1100 0011	P	0101 0000	1101 0111
D	0100 0100	1100 0100	Q	0101 0001	1101 1000
E	0100 0101	1100 0101	R	0101 0010	1101 1001
F	0100 0110	1100 0110	S	0101 0011	1110 0010
G	0100 0111	1100 0111	T	0101 0100	1110 0011
H	0100 1000	1100 1000	U	0101 0101	1110 0100
I	0100 1001	1100 1001	V	0101 0110	1110 0101
J	0100 1010	1101 0001	W	0101 0111	1110 0110
K	0100 1011	1101 0010	X	0101 1000	1110 0111
L	0100 1100	1101 0011	Y	0101 1001	1110 1000
M	0100 1101	1101 0100	Z	0101 1010	1110 1001
0	0011 0000	1111 0000	5	0011 0101	1111 0101
1	0011 0001	1111 0001	6	0011 0110	1111 0110
2	0011 0010	1111 0010	7	0011 0111	1111 0111
3	0011 0011	1111 0011	8	0011 1000	1111 1000
4	0011 0100	1111 0100	9	0011 1001	1111 1001

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 – brightness, and chrominance – colour.

Eg. Photographic Experts Group (JPEG), Portable Network Graphics (PNGs), GIF, TIFF.

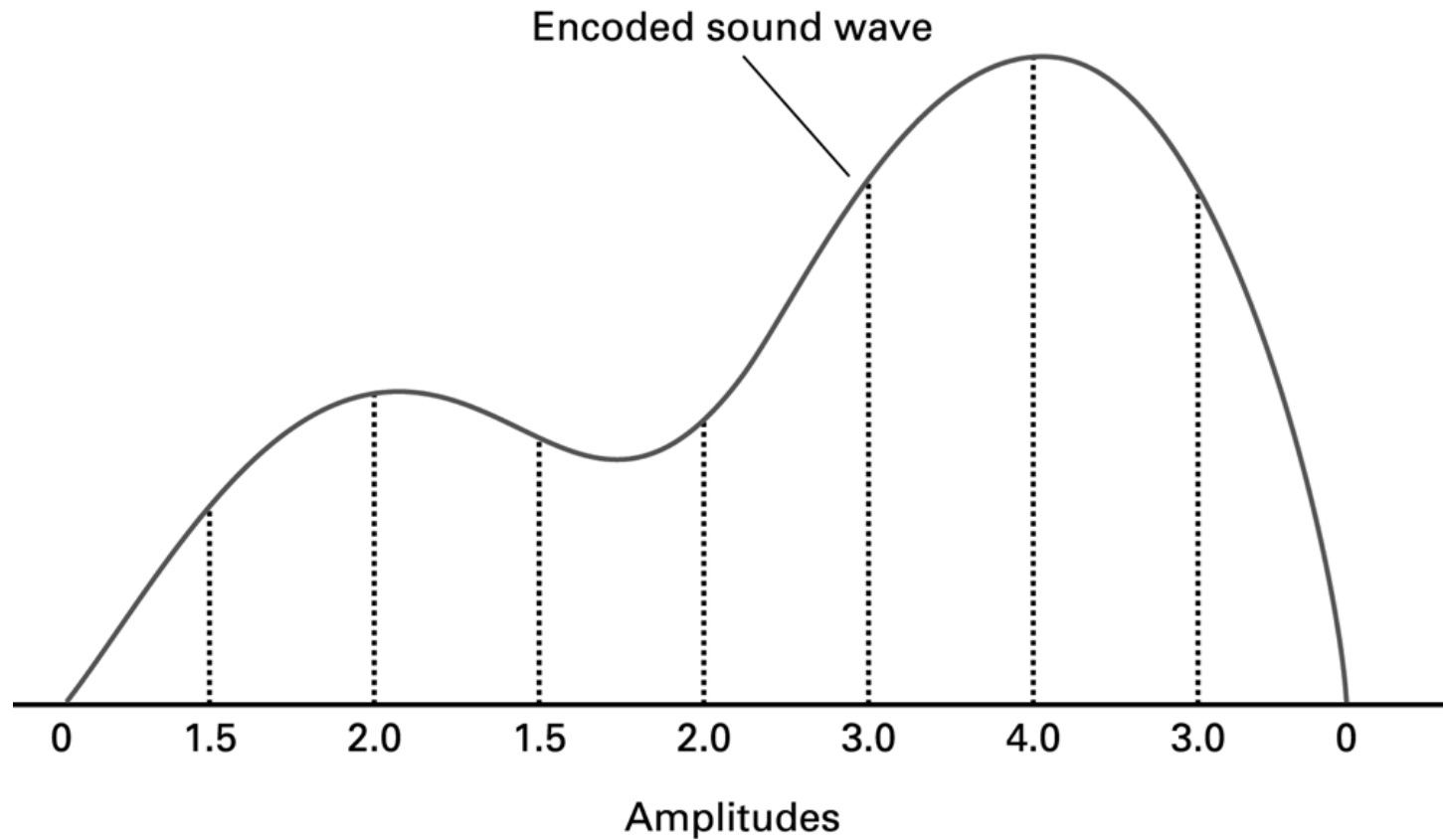
- Vector techniques

- Scalable, Flexibility
- mathematical formulas rather than individual colored blocks
- TrueType and PostScript
- Eg. Adobe Illustrator (AI), PDF, Encapsulated Postscript Vector graphics (EPS) (Adobe Illustrator), & Scalable Vector Graphics (SVG)

Representing Sound

- Sampling techniques
 - Used for high quality recordings
 - Records actual audio
- MIDI (Musical Instrument Digital Interface)
 - 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



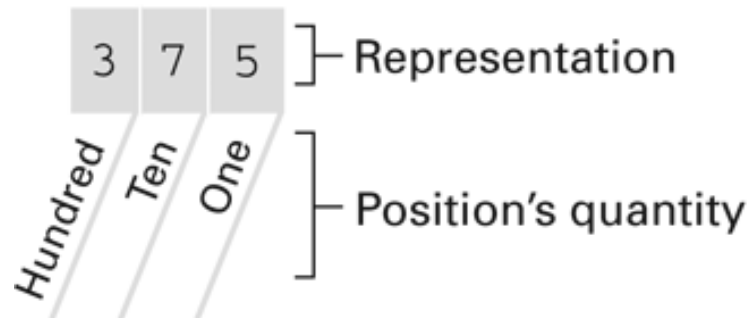
The Binary System

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

a. Base ten system



b. Base two system

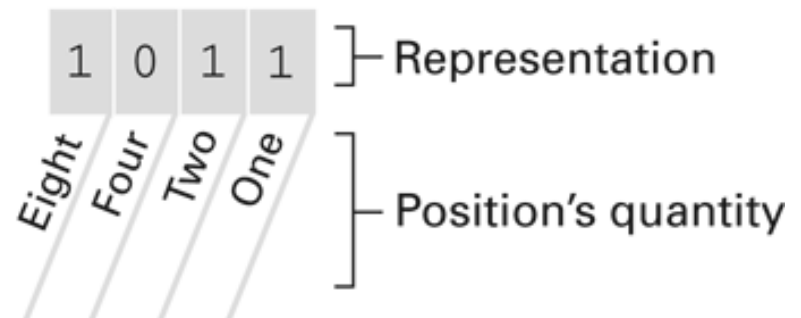


Figure 1.14 Decoding the binary representation 100101

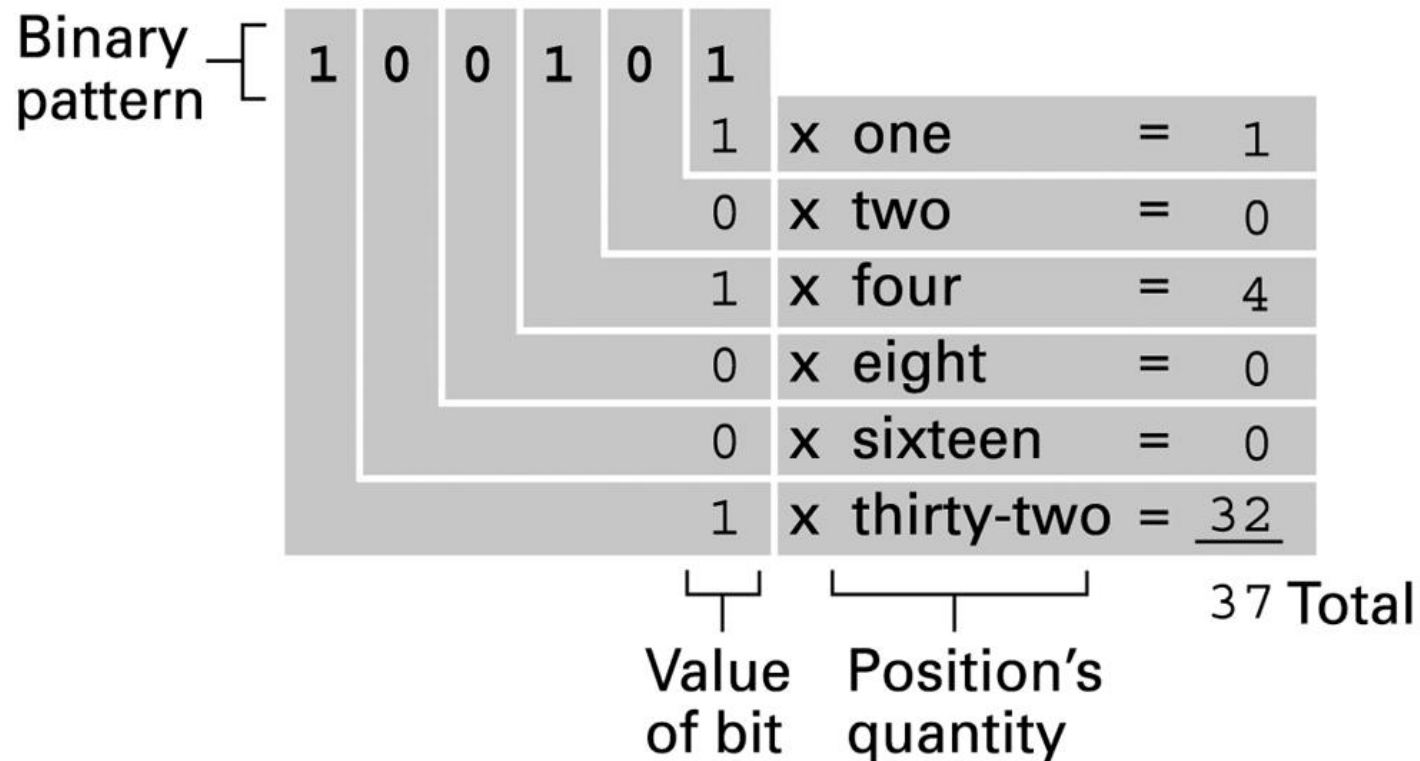


Figure 1.15 An algorithm for finding the binary representation of a positive integer

- 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

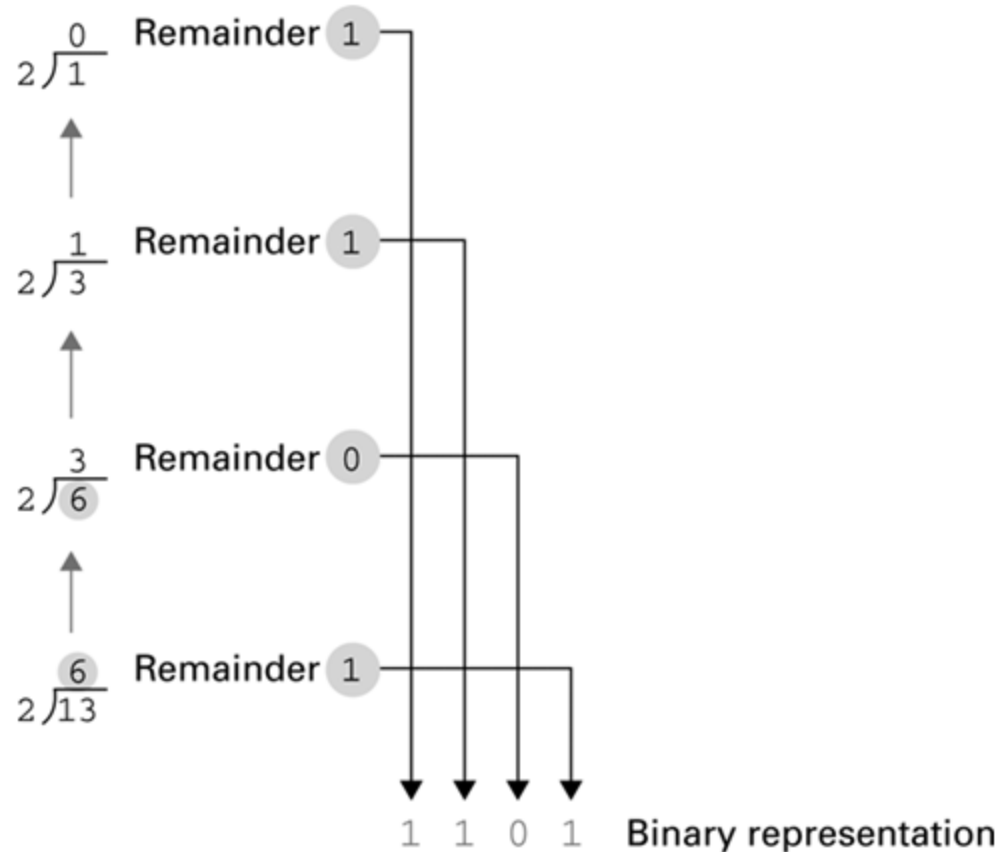


Figure 1.17 The binary addition facts

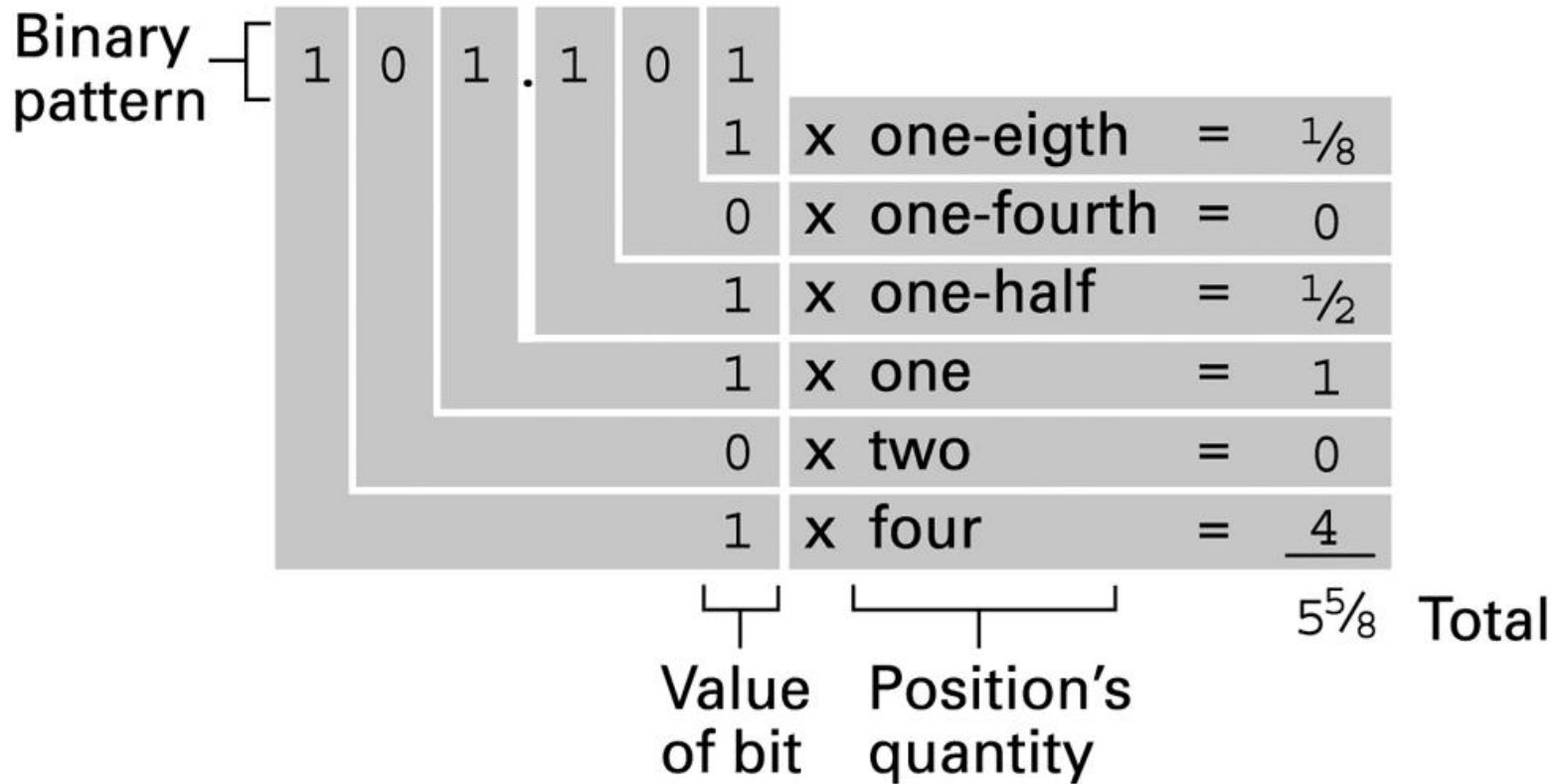
$$\begin{array}{r} 0 \\ + 0 \\ \hline 0 \end{array}$$

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

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

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

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

Storing integers

- How the computer stores both positive and negative numbers.
- Not commonly used ways
 - Sign and magnitude
 - One's complement
- Commonly used one
 - Two's complement

bin	dec
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

- 3 bit system
- Anytime you have n bits
- 2 to the power n possible values
- E.g. 3 bit system = 8 values
- I use 3 bit system to keep it simple and small
- So 4 bit system = 16 values.
- With 3 bits all of the concepts can still be explained

- Unsigned bits
- Signed bits (+ve & -ve) values
- Sign and Magnitude
 - Leading zeros for all +ves
 - Leading ones for all –ves
 - Drawbacks :
 - 2 zeroes problem, and –ve zero makes even a less intuitive sense
 - Its difficult for hardware implementation

- One's complement
 - Original value
 - To get –ve, flip bits
- Drawbacks
 - 2 zeroes problem, or –ve zero makes even a less intuitive sense
 - Its difficult for hardware implementation

Two complement

- Original value
 - To get the -ve value
- Flip bits and add 1.
- Basic things apply
- -ve1 is always all 1s regardless of the number of bits
- The smallest -ve number will always be a leading 1, followed by all zeros
- The largest +ve number will always be a leading 0 followed by all ones.
- One benefits of 2's complement is that we can implement our addition and subtraction as simple addition. Which means we can do everything with an Adder.

- Addition problems
- $2+1 = 3$; $011 + 001 = 011$
- $3-1=2$; $011 + 111 = 1010$
- Ignore the carry, because we have 3 bit system
- $2-4= -2$: $010 + 100 = 110$
- Scenario that will cause a problem
- $3 + 2 = 5$: but we don't have a 5 in the binary representation
- $011 + 010 = 101$; 2 +ve numbers resulting in –ve number is an overflow condition and we can report that as an error.
- $-4 - 3 = -7$: $100 + 101 = 1001$; 2 -ve numbers resulting in +ve number is an overflow condition and we can report that as an error.

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

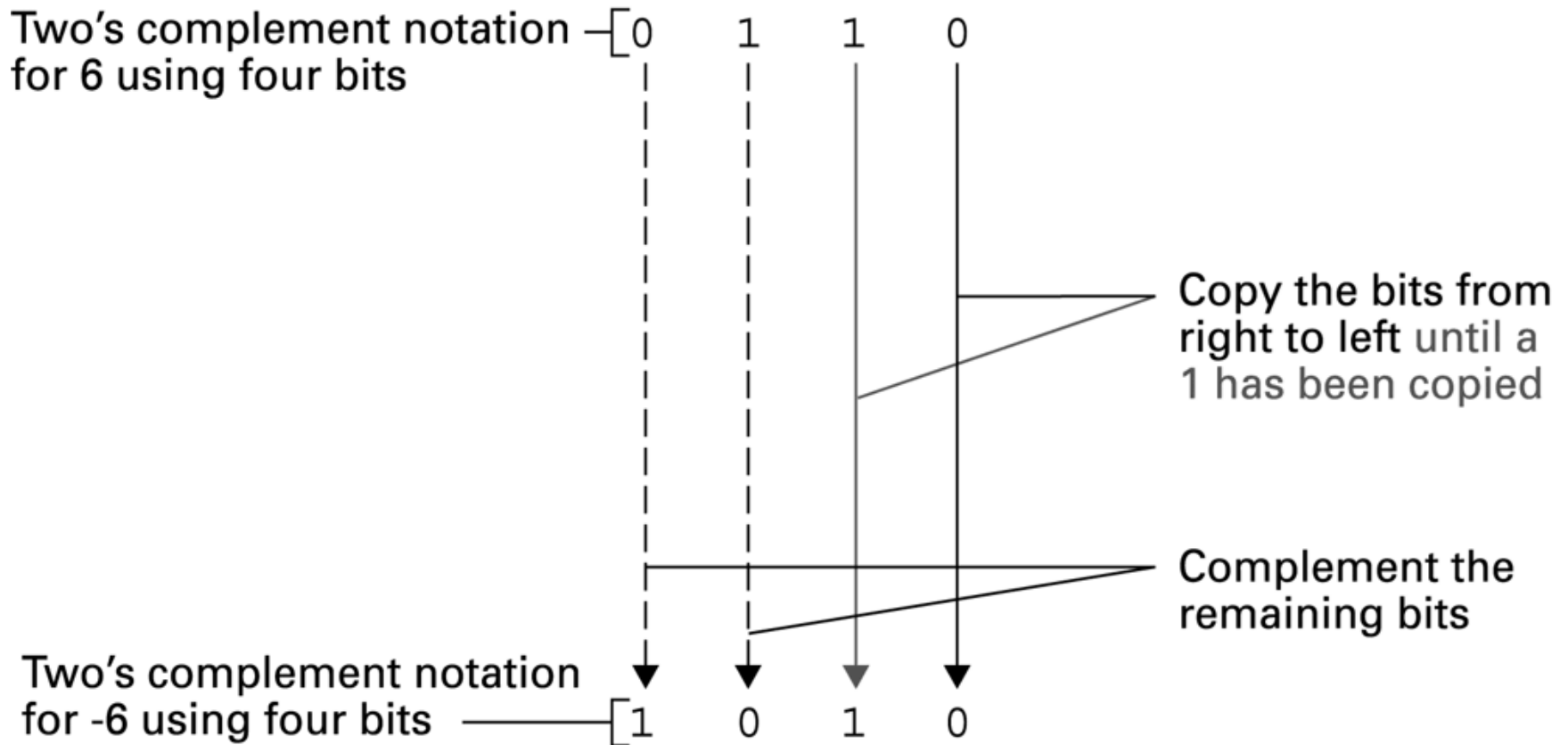








Figure 1.21 Addition problems converted to two's complement notation

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

Example:

- Example: 3 in 8-bit signed binary notation is 00000011. To flip the sign, you first flip all bits (11111100), then add 1 (11111101). So, -3 is 11111101. To flip the sign again, you first flip all bits (00000010), then add 1 (00000011), and you can see that this is the same 3.

Two's complement

- deals with the 2 zeros problem- we no longer have the 2 zeros.
- allows us to do addition and subtraction with only addition which makes it easier to implement hardware.
- enables us to detect overflow condition so that we can report errors.

Excess Notation

- A different way of expressing –ve numbers in binary
- It comes with a set of digit length
 - 3 bit excess notation
 - 6 bit excess notation
 - 8 bit excess notation

Excess Notation

- 0 = 1, followed by as many 0s as possible
- Example
 - 3 bits excess 0 = 100
 - 4 bits excess 0 = 1000
 - 5 bits excess 0 = 10000
 - 6 bits excess 0 = 100000
- NB: 0 is not a sign-less number anymore. It is neither +ve nor -ve.
- But we assigned 1 to it. Every +ve number starts with a 1.
- Hence, it is not commonly used as the 2's complement.

Building up the excess notation

Peculiarities:

- When you flip 0 you get -1. Ideally, it is when you flip +1 that we should get -1.
- There are troubles with addition, eg.
- It works with adding +ve numbers
- but adding -ve numbers gives problems. eg. $(-4) + (-3) = (-7)$

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

Negative Numbers

We can represent negative numbers in several ways.

The simplest is to simply use the leftmost digit of the number as a special value to represent the sign of the number: 0 = positive, 1 = negative.

For example, a value of **positive** 12 (decimal) would be written as **01100** in binary, but **negative** 12 (decimal) would be written as **11100**.

Notice that in this system, it is important to show the leading 0 (to indicate a positive value).

For technical reasons, the "two's complement" is more often used for representing negative numbers. In this system, a positive 12 is still 01100, but -12 would be written as 10100. Notice that there is nothing intrinsically correct about one system over another. Either 11100 or 10100 can be used to represent -12, it just depends on what system of *interpretation* is used. That is, a human programmer chooses the *meaning* of the bits.

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

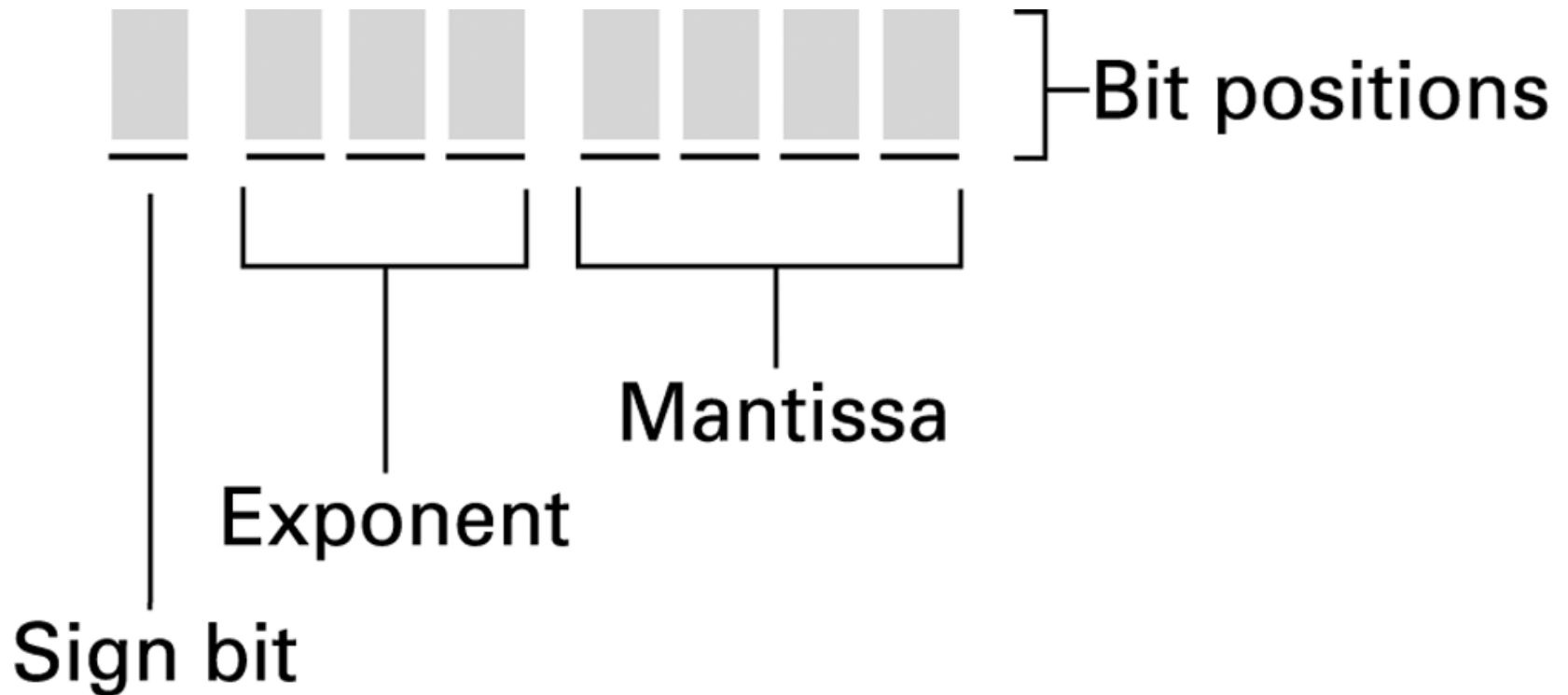
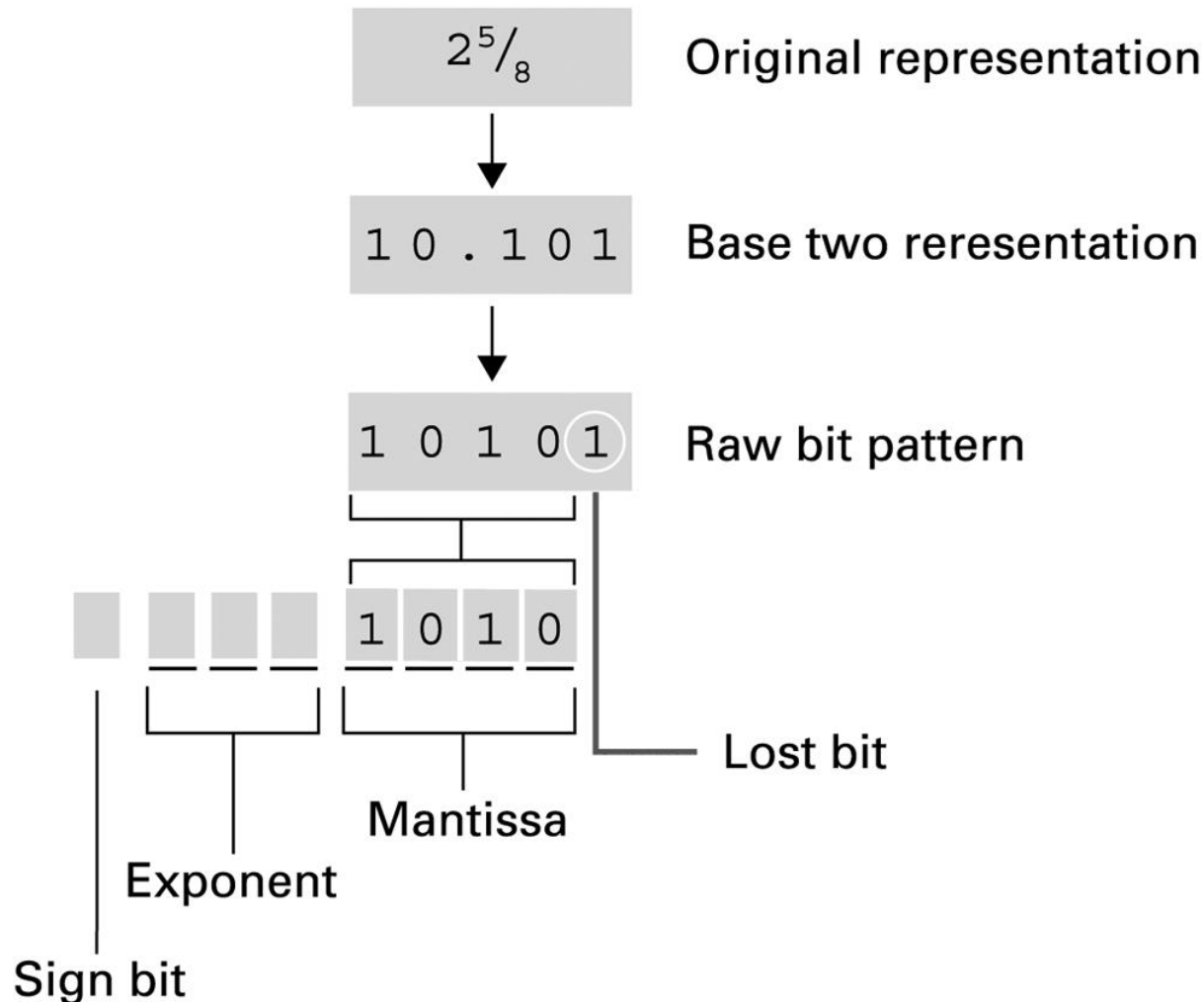


Figure 1.25 Encoding the value $2^{5/8}$



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

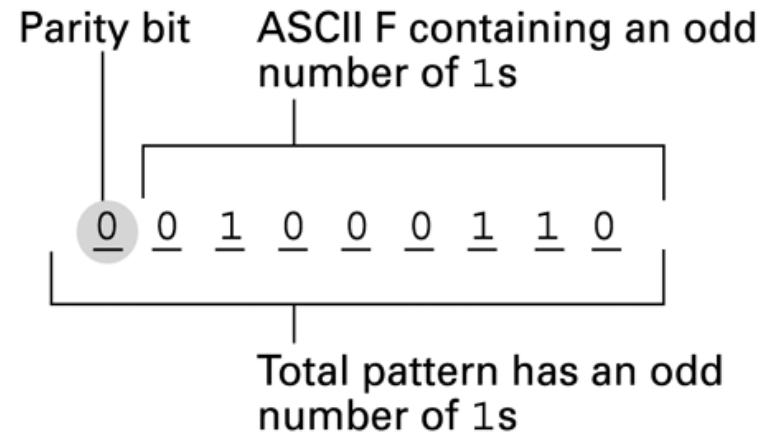
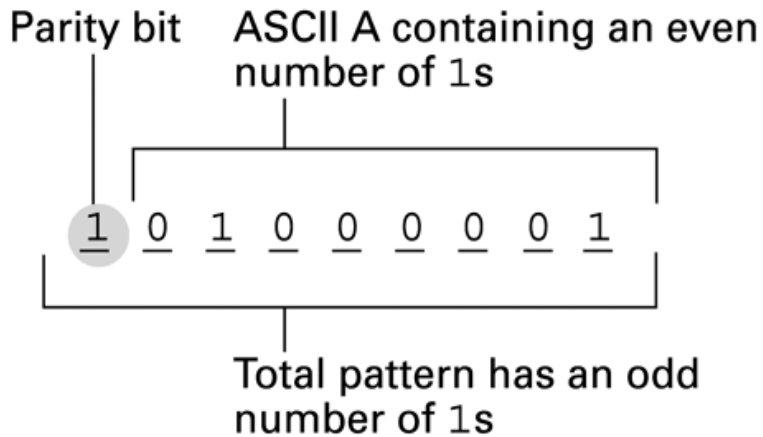


Figure 1.27 An error-correcting code

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 ——— Smallest distance
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