



IT111 - Introduction to Information Technology

Bachelor of Science in Information Technology

LECTURE 4 DATA STORAGE (CONTINUE)

Prof. Lamiaa Fattouh Ibrahim

Everyone Smart University

Faculty of INFORMATICS

Spring 2023



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

After studying this lecture, you should be able to:

- ☐ Describe how reals are stored in a computer.
- ☐ Describe how text is stored in a computer using one of the various encoding systems.
- ☐ Describe how audio is stored in a computer using sampling, quantization, and encoding.
- ☐ Describe how images are stored in a computer using raster and vector graphics schemes.
- ☐ Describe how video is stored in a computer as a representation of images changing in time.

Integers

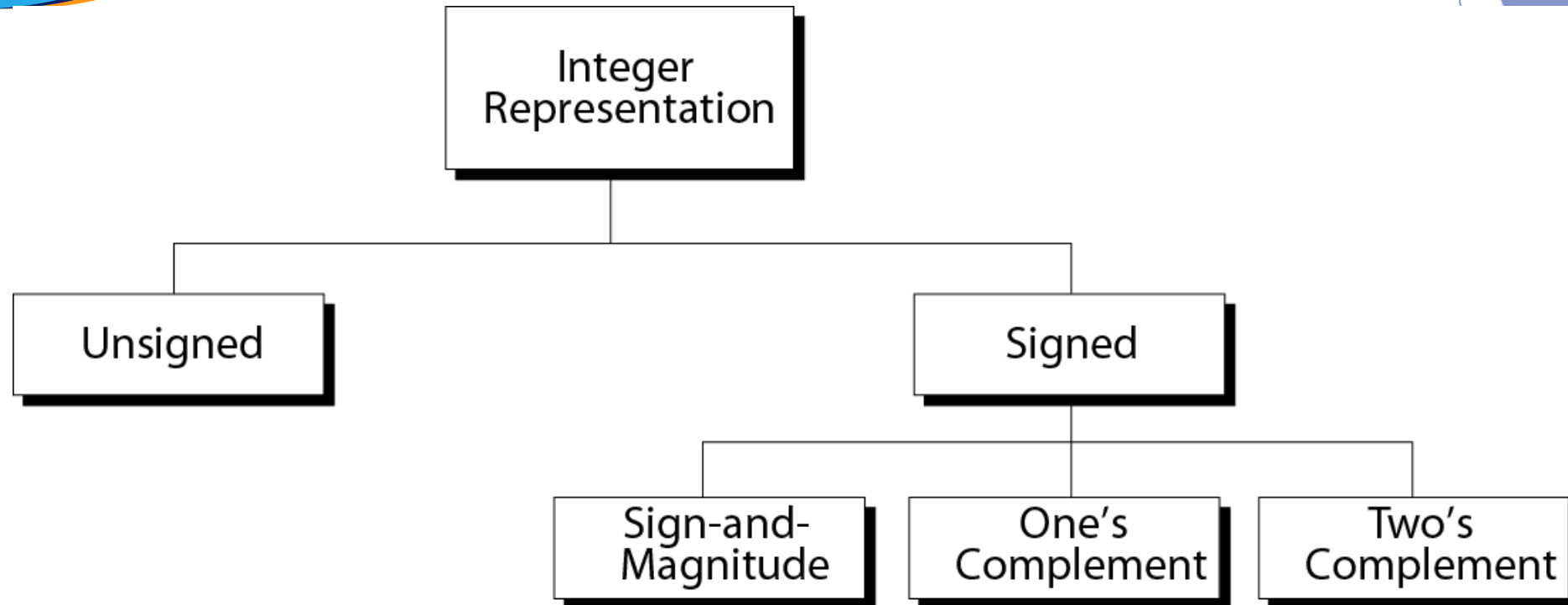


Table 3.2 Example of storing unsigned integers in two different computers

<i>Decimal</i>	<i>8-bit allocation</i>	<i>16-bit allocation</i>
-----	-----	-----
7	00000111	0000000000000000111
234	11101010	00000000011101010
258	overflow	00000000100000010
24,760	overflow	01100000010111000
1,245,678	overflow	overflow

Storing reals

A real is a number with an integral part and a fractional part. For example, 23.7 is a real number—the integral part is 23 and the fractional part is $7/10$. Although a fixed-point representation can be used to represent a real number, the result may not be accurate, or it may not have the required precision. The next two examples explain why.



Real numbers with very large integral parts or very small fractional parts should not be stored in fixed-point representation.

Example 3.16

In the decimal system, assume that we use a fixed-point representation with two digits at the right of the decimal point and six digits at the left of the decimal point, for a total of sixteen digits. The precision of a real number in this system is lost if we try to represent a decimal number such as 1.00234: the system stores the number as 1.00.



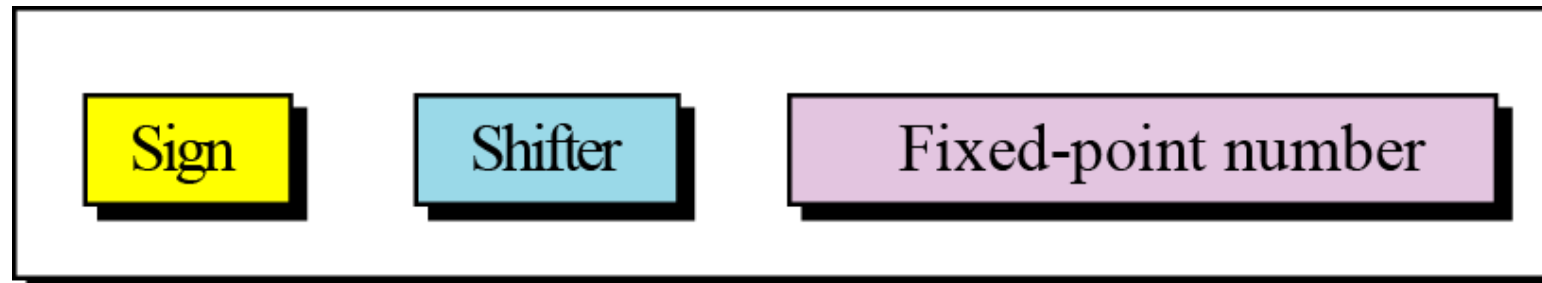
Example 3.17

In the decimal system, assume that we use a fixed-point representation with six digits to the right of the decimal point and ten digits for the left of the decimal point, for a total of sixteen digits. The accuracy of a real number in this system is lost if we try to represent a decimal number such as 236154302345.00. The system stores the number as 6154302345.00: the integral part is much smaller than it should be.

Floating-point representation

The solution for maintaining accuracy or precision is to use **floating-point representation**.

Figure 3.10 The three parts of a real number in floating-point



Floating-point representation

A floating-point representation of a number is made up of three parts: a sign, a shifter, and a fixed-point number.

Example 3.18

The following shows the decimal number

7,452,000,000,000,000,000.00

in scientific notation (floating-point representation).

Actual number → + 7,425,000,000,000,000,000.00

Scientific notation → + 7.425×10^{21}

The three sections are the **sign** (+), the **shifter** (21), and the **fixed-point part** (7.425). Note that the shifter is the exponent.

Sign

Shifter

Fixed-point number

Example 3.19

Show the number -0.00000000000000232
in scientific notation (floating-point representation).

Solution

We use the same approach as in the previous example—we move the decimal point after the digit 2, as shown below:

Actual number	→	–	0.00000000000000232
Scientific notation	→	–	2.32×10^{-14}

The three sections are the **sign** (–), the **shifter** (-14), and the **fixed-point part** (2.32). Note that the shifter is the exponent.

Example 3.20

Show the number

$(10100100000000000000000000000000.00)_2$

in floating-point representation.

Solution

We use the same idea, keeping only one digit to the left of the decimal point.

Actual number $\rightarrow + (10100100000000000000000000000000.00)_2$

Scientific notation $\rightarrow + 1.01001 \times 2^{32}$

Example 3.21

Show the number

$$-(0.\underbrace{00000000000000000000}_{\text{26 zeros}}101)_2$$

in floating-point representation.

Solution

We use the same idea, keeping only one digit to the left of the decimal point.

Actual number	→	–	$(0.000000000000000000000000101)_2$
Scientific notation	→	–	1.01×2^{-24}

Normalization

To make the fixed part of the representation uniform, both the scientific method (for the decimal system) and the floating-point method (for the binary system) use only one non-zero digit on the left of the decimal point. This is called **normalization**. In the decimal system this digit can be 1 to 9, while in the binary system it can only be 1. In the following, d is a non-zero digit, x is a digit, and y is either 0 or 1.

Decimal $\rightarrow \pm d.xxxxxxxxxxxxxx$

Note: d is 1 to 9 and each x is 0 to 9

Binary $\rightarrow \pm 1.yyyyyyyyyyyyyy$

Note: each y is 0 or 1

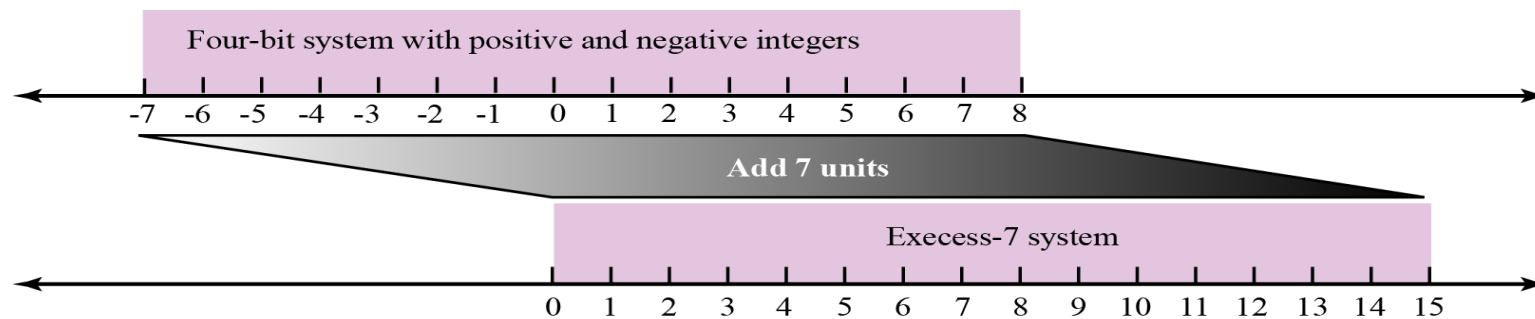
Excess System

The exponent, the power that shows how many bits the decimal point should be moved to the left or right, is a signed number. Although this could have been stored using two's complement representation, a new representation, called the Excess system, is used instead. In the Excess system, both positive and negative integers are stored as unsigned integers. To represent a positive or negative integer, a positive integer (called a bias) is added to each number to shift them uniformly to the non-negative side. The value of this bias is $2^{m-1} - 1$, where m is the size of the memory location to store the exponent.

Example 3.22

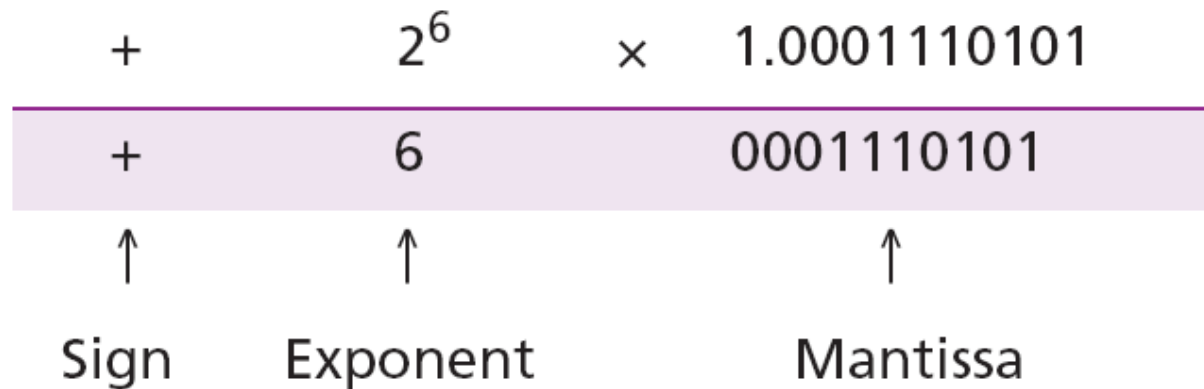
We can express sixteen integers in a number system with 4-bit allocation. By adding seven units to each integer in this range, we can uniformly translate all integers to the right and make all of them positive without changing the relative position of the integers with respect to each other, as shown in the figure. The new system is referred to as Excess-7, or biased representation with biasing value of 7.

Figure 3.11 Shifting in Excess representation



Sign, Exponent and Mantissa

After a binary number is normalized, only three pieces of information about the number are stored: sign, exponent, and mantissa (the bits to the right of the decimal point). For example, $+1000111.0101$ becomes:



Sign

The sign of the number can be stored using 1 bit (0 or 1).

Exponent

The exponent (power of 2) defines the shifting of the decimal point. Note that the power can be negative or positive. The **Excess representation** is the method used to store the exponent.

Mantissa

The **mantissa** is the binary integer to the right of the decimal point. It defines the precision of the number. The mantissa is stored in fixed-point notation. If we think of the mantissa and the sign together, we can say this combination is stored as an integer in sign-and-magnitude format. However, we need to remember that it is not an integer—it is a fractional part that is stored like an integer.



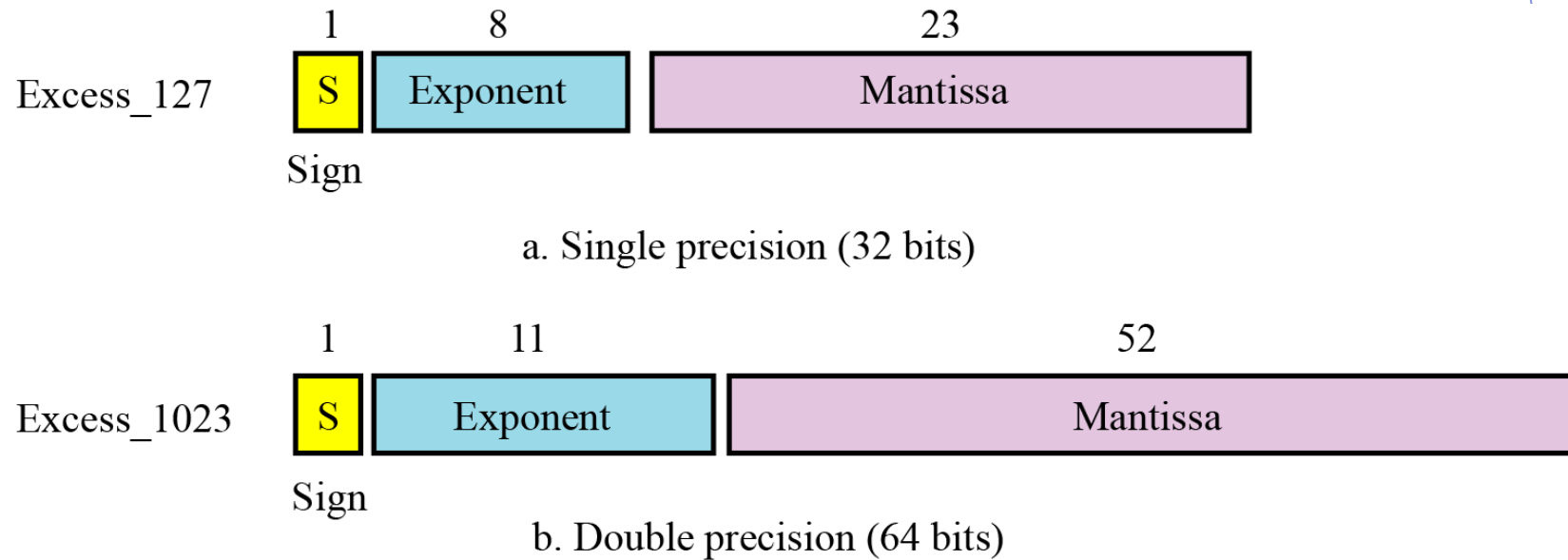
Note that the point and the bit 1 to the left of the fixed-point section are not stored—they are implicit.



The mantissa is a fractional part that, together with the sign, is treated like an integer stored in sign-and-magnitude representation.

IEEE Standard

Figure 3.12 IEEE standards for floating-point representation



IEEE Specifications

Table 3.2 Specifications of the two IEEE floating-point standards

<i>Parameter</i>	<i>Single Precision</i>	<i>Double Precision</i>
Memory location size (number of bits)	32	64
Sign size (number of bits)	1	1
Exponent size (number of bits)	8	11
Mantissa size (number of bits)	23	52
Bias (integer)	127	1023

Example 3.23

Show the Excess_127 (single precision) representation of the decimal number 5.75.

Solution

- The sign is positive, so $S = 0$.
- Decimal to binary transformation: $5.75 = (101.11)_2$.
- Normalization: $(101.11)_2 = (1.1011)_2 \times 2^2$.
- $E = 2 + 127 = 129 = (10000001)_2$, $M = 1011$. We need to add nineteen zeros at the right of M to make it 23 bits.
- The presentation is shown below:

0	10000001	101100000000000000000000
S	E	M

The number is stored in the computer as

01000000110110000000000000000000

Example 3.24

Show the Excess_127 (single precision) representation of the decimal number -161.875 .

Solution

- The sign is negative, so $S = 1$.
- Decimal to binary transformation: $161.875 = (10100001.111)_2$.
- Normalization: $(10100001.111)_2 = (1.0100001111)_2 \times 2^7$.
- $E = 7 + 127 = 134 = (10000110)_2$ and $M = (0100001111)_2$.
- Representation:

1	10000110	010000111100000000000000
S	E	M

The number is stored in the computer as

11000011010000111100000000000000

Example 3.25

Show the Excess_127 (single precision) representation of the decimal number -0.0234375 .

Solution

- $S = 1$ (the number is negative).
- Decimal to binary transformation: $0.0234375 = (0.0000011)_2$.
- Normalization: $(0.0000011)_2 = (1.1)_2 \times 2^{-6}$.
- $E = -6 + 127 = 121 = (01111001)_2$ and $M = (1)_2$.
- Representation:

1	01111001	100000000000000000000000
S	E	M

The number is stored in the computer as

10111100110000000000000000000000

Example 3.26

The bit pattern $(1100101000000000111000100001111)_2$ is stored in Excess_127 format. Show the value in decimal.

Solution

- a. The first bit represents S, the next eight bits, E and the remaining 23 bits, M.

S	E	M
1	10010100	00000000111000100001111

- b. The sign is negative.
 c. The shifter = $E - 127 = 148 - 127 = 21$.
 d. This gives us $(1.00000000111000100001111)_2 \times 2^{21}$.
 e. The binary number is $(1000000001110001000011.11)_2$.
 f. The absolute value is 2,104,378.75.
 g. The number is **$-2,104,378.75$** .

3-3 STORING TEXT

A section of text in any language is a sequence of symbols used to represent an idea in that language. For example, the English language uses 26 symbols (A, B, C,..., Z) to represent uppercase letters, 26 symbols (a, b, c, ..., z) to represent lowercase letters, nine symbols (0, 1, 2, ..., 9) to represent numeric characters and symbols (., ?, :, ; , ..., !) to represent punctuation. Other symbols such as blank, newline, and tab are used for text alignment and readability.

We can represent each symbol with a bit pattern. In other words, text such as “CATS”, which is made up from four symbols, can be represented as four n-bit patterns, each pattern defining a single symbol (Figure 3.14).

Figure 3.14 Representing symbols using bit patterns

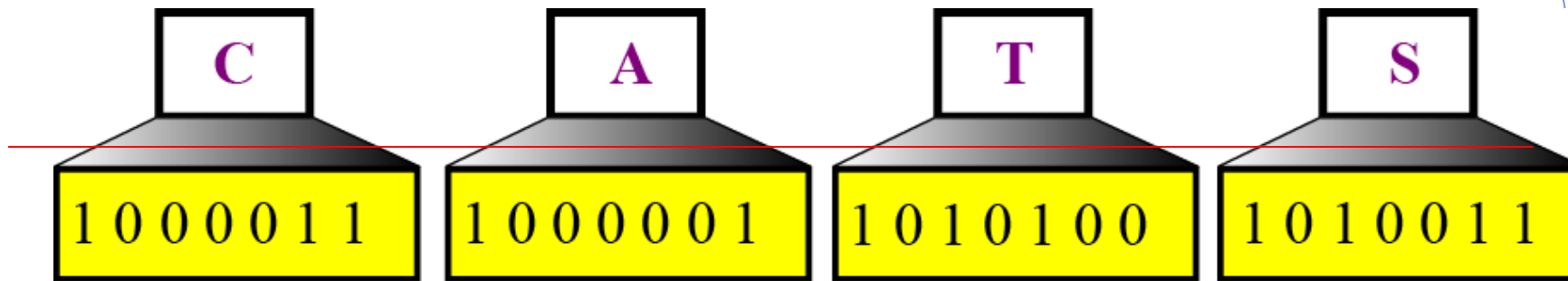


Table 3.3 Number of symbols and bit pattern length

<i>Number of symbols</i>	<i>Bit pattern length</i>	<i>Number of symbols</i>	<i>Bit pattern length</i>
2	1	128	7
4	2	256	8
8	3	65,536	16
16	4	4,294,967,296	32

Number of symbols and bit pattern length

Number of Symbols

2

4

8

16

...

128

256

...

65,536

Bit Pattern Length

1

2

3

4

...

7

8

...

16

Codes

ASCII: The **American National Standards Institute (ANSI)** developed a code called **American Standard Code for Information Interchange (ASCII)**. This code uses seven bits for each symbol. This means that $2^7 = 128$ different symbols can be defined in this code. The full bit patterns for ASCII code are included in Appendix A. Today ASCII is part of Unicode.

Unicode: uses 32 bits and can therefore represent up to $2^{32} = 4\,294\,967\,296$ symbols. Different sections of the code are allocated to symbols from different languages in the world. Some parts of the code are used for graphical and special symbols. A brief set of Unicode symbols is listed in Appendix A. ASCII is part of Unicode today.

❑ Other Codes

ASCII CODE

Table A.1 ASCII

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
(00) ₁₆	Null	(20) ₁₆	Space	(40) ₁₆	@	(60) ₁₆	`
(01) ₁₆	SOH	(21) ₁₆	!	(41) ₁₆	A	(61) ₁₆	a
(02) ₁₆	STX	(22) ₁₆	"	(42) ₁₆	B	(62) ₁₆	b
(03) ₁₆	ETX	(23) ₁₆	#	(43) ₁₆	C	(63) ₁₆	c
(04) ₁₆	EOT	(24) ₁₆	\$	(44) ₁₆	D	(64) ₁₆	d
(05) ₁₆	ENQ	(25) ₁₆	%	(45) ₁₆	E	(65) ₁₆	e
(06) ₁₆	ACK	(26) ₁₆	&	(46) ₁₆	F	(66) ₁₆	f
(07) ₁₆	BEL	(27) ₁₆	'	(47) ₁₆	G	(67) ₁₆	g
(08) ₁₆	BS	(28) ₁₆	(	(48) ₁₆	H	(68) ₁₆	h
(09) ₁₆	HT	(29) ₁₆	)	(49) ₁₆	I	(69) ₁₆	i
(0A) ₁₆	LF	(2A) ₁₆	*	(4A) ₁₆	J	(6A) ₁₆	j
(0B) ₁₆	VT	(2B) ₁₆	+	(4B) ₁₆	K	(6B) ₁₆	k
(0C) ₁₆	FF	(2C) ₁₆	,	(4C) ₁₆	L	(6C) ₁₆	l
(0D) ₁₆	CR	(2D) ₁₆	-	(4D) ₁₆	M	(6D) ₁₆	m
(0E) ₁₆	SO	(2E) ₁₆	.	(4E) ₁₆	N	(6E) ₁₆	n
(0F) ₁₆	SI	(2F) ₁₆	/	(4F) ₁₆	O	(6F) ₁₆	o
(10) ₁₆	DLE	(30) ₁₆	0	(50) ₁₆	P	(70) ₁₆	p

Figure 2-5

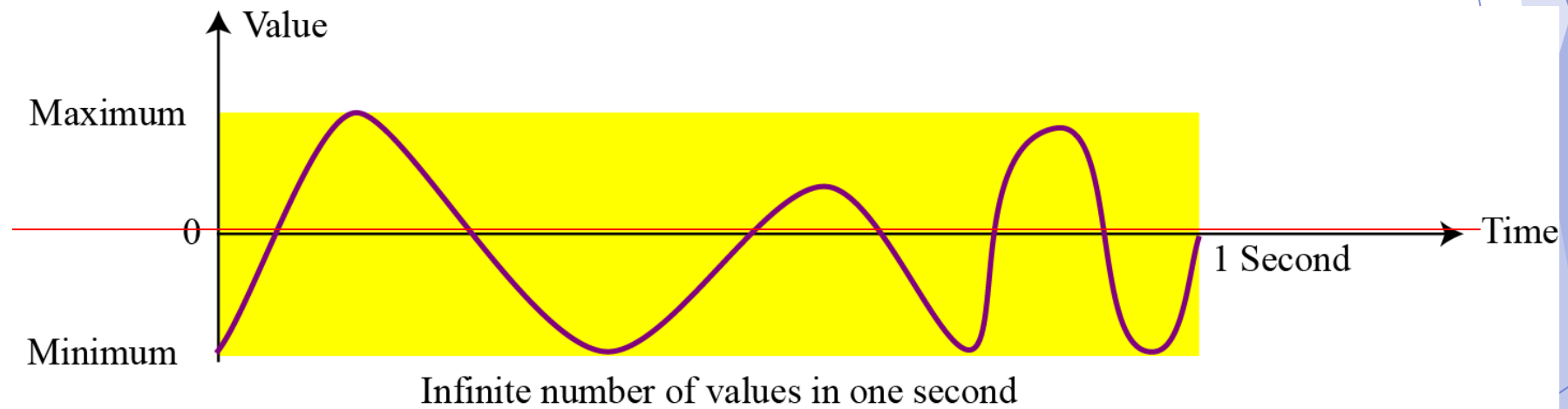
Representation of the word “BYTE” in ASCII code

B	Y	T	E
1000010	1011001	1010100	1000101

3-4 STORING AUDIO

Audio is a representation of sound or music. Audio, by nature, is different than the numbers or text we have discussed so far. Text is composed of **countable entities** (characters): we can count the number of characters in text. Text is an example of **digital data**. In contrast, audio is not countable. Audio is an example of **analog data**. Even if we are able to measure all its values in a period of time, we cannot store these in the computer's memory, as we would need infinite number of memory locations. Figure 3.15 shows the nature of an analog signal, such as audio, that varies with time.

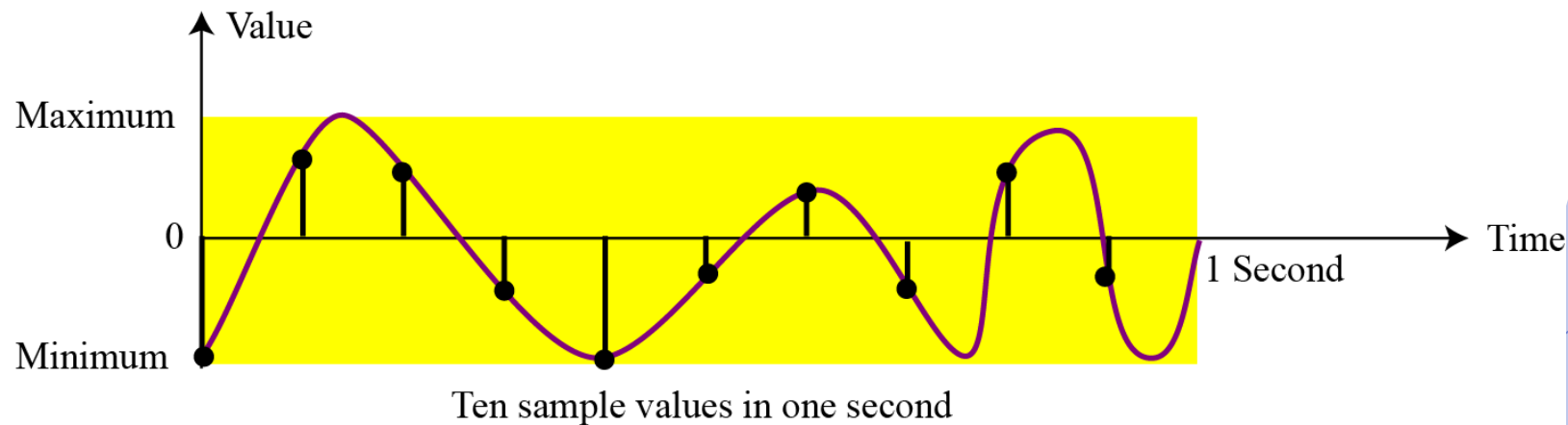
Figure 3.15 An audio signal



3.4.1 Sampling

If we cannot record all the values of an audio signal over an interval, we can record some of them. Sampling means that we select only a finite number of points on the analog signal, measure their values, and record them.

Figure 3.16 Sampling an audio signal



3.4.2 Quantization

The value measured for each sample is a real number. This means that we can store 40,000 real values for each one second sample. However, it is simpler to use an unsigned integer (a bit pattern) for each sample. Quantization refers to a process that rounds the value of a sample to the closest integer value. For example, if the real value is 17.2, it can be rounded down to 17: if the value is 17.7, it can be rounded up to 18.

3.4.2 Encoding

The quantized sample values need to be encoded as bit patterns. Some systems assign positive and negative values to samples, some just shift the curve to the positive part and assign only positive values.

If we call the **bit depth** or number of bits per sample B , the number of samples per second, S , we need to store $S \times B$ bits for each second of audio. This product is sometimes refer to as **bit rate**, R . For example, if we use 40,000 samples per second and 16 bits per each sample, the bit rate is

$$R = 40,000 \times 16 = 640,000 \text{ bits per second}$$

3.4.4 Standards for sound encoding

Today the dominant standard for storing audio is **MP3** (short for **MPEG Layer 3**). This standard is a modification of the **MPEG (Motion Picture Experts Group)** compression method used for video. It uses 44100 samples per second and 16 bits per sample. The result is a signal with a bit rate of 705,600 bits per second, which is compressed using a compression method that discards information that cannot be detected by the human ear. This is called lossy compression, as opposed to lossless compression: **see Chapter 15**.



3-5 STORING IMAGES

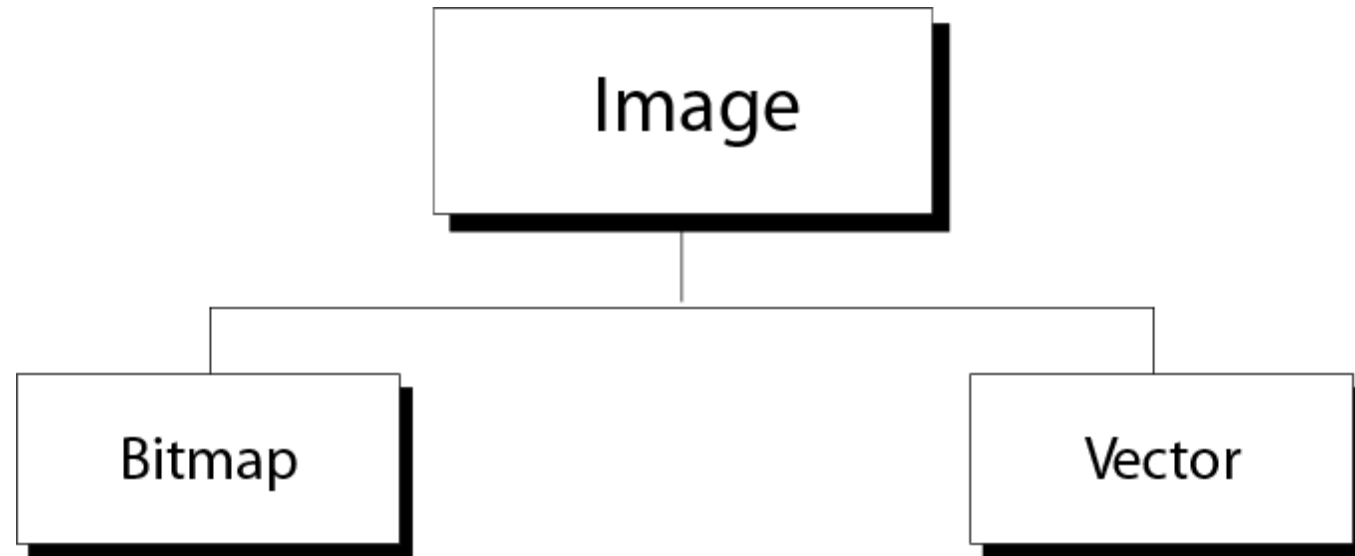
Images are stored in computers using two different techniques: **raster graphics** and **vector graphics**.

3.5.1 Raster graphics

Raster graphics (or **bitmap graphics**) is used when we need to store an analog image such as a photograph. A photograph consists of analog data, similarly to audio information: the difference is that the intensity (color) of data varies in space instead of in time. This means that data must be sampled. However, sampling in this case is normally called **scanning**. The samples are called **pixels** (picture elements).



Image representation methods



Resolution

Just like audio sampling, in image scanning we need to decide how many pixels we need to record for each square or linear inch. The scanning rate in image processing is called **resolution**. If the resolution is sufficiently high, the human eye cannot recognize the discontinuity in reproduced images.

Color depth

The number of bits used to represent a pixel, its **color depth**, depends on how the pixel's color is handled by different encoding techniques. The perception of color is how our eyes respond to a beam of light. Our eyes have different types of photoreceptor cells: some respond to the three primary colors red, green, and blue (often called **RGB**), while others merely respond to the intensity of light.

True-Color

One of the techniques used to encode a pixel is called True-Color, which uses **24** bits to encode a pixel. In this technique, each of the three primary colors (RGB) are represented by eight bits. Since an 8-bit pattern can represent a number between 0 to 255 in this technique, each color is represented by three decimal numbers between 0 to 255. Table 3.4 shows the three values for some of the colors in this technique.

Table 3.4 Some colors defined in True-Color

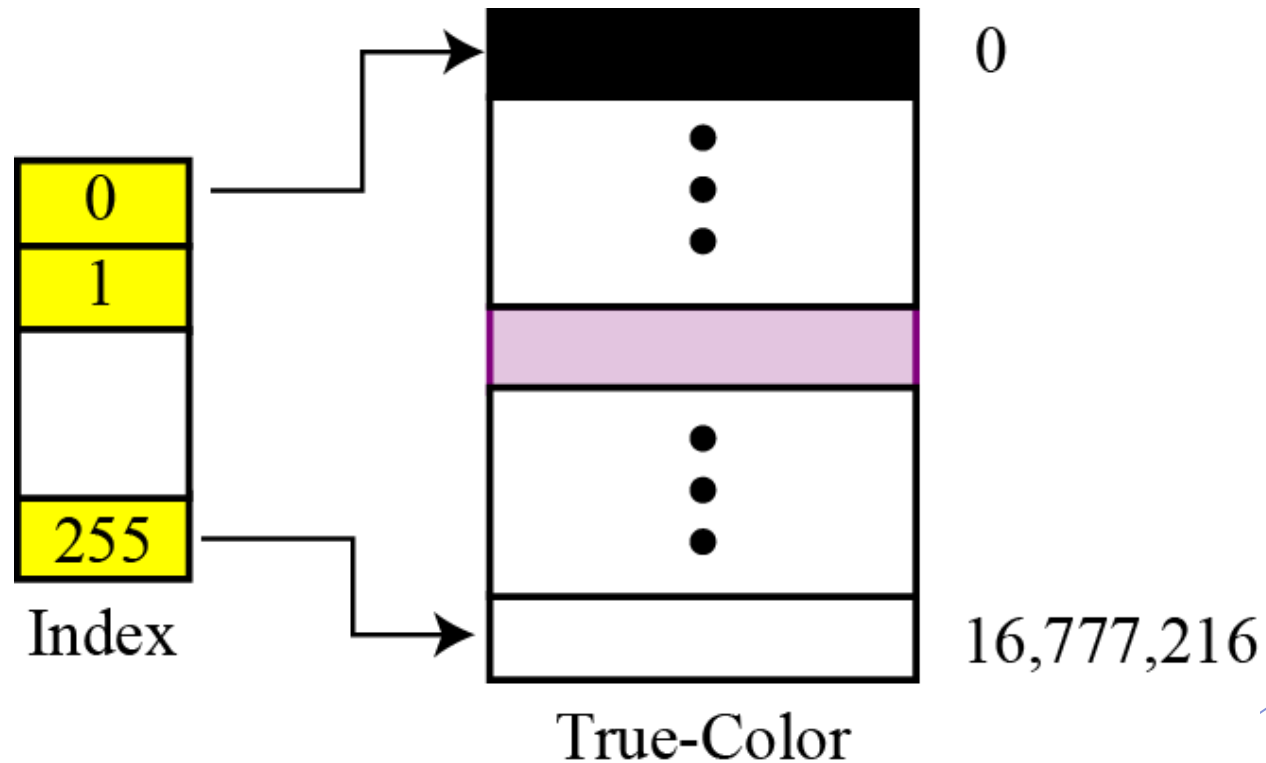
<i>Color</i>	<i>Red</i>	<i>Green</i>	<i>Blue</i>	<i>Color</i>	<i>Red</i>	<i>Green</i>	<i>Blue</i>
Black	0	0	0	Yellow	255	255	0
Red	255	0	0	Cyan	0	255	255
Green	0	255	0	Magenta	255	0	255
Blue	0	0	255	White	255	255	255

indexed color—or palette color

The True-Color scheme uses more than 16 million colors. Many applications do not need such a large range of colors. The **indexed color**—or **palette color**—scheme uses only a portion of these colors. In this scheme each application selects a few (normally 256) colors from the large set of colors and indexes them, assigning a number between 0 and 255 to each selected color. This is similar to the way in which an artist might have a great many colors in their studio, but at each moment use only a few on their palette.

Indexed color

Figure 3.17 Relationship of the indexed color to the True-Color



For example, a high-quality digital camera uses almost three million pixels for a 3×5 inch photo. The following shows the number of bits that need to be stored using each scheme:

True-Color:	3,000,000	×	24	=	72,000,000
Indexed-Color:	3,000,000	×	8	=	24,000,000

Standards for image encoding

Several de facto standards for image encoding are in use. **JPEG (Joint Photographic Experts Group)** uses the True-Color scheme, but compresses the image to reduce the number of bits (see Chapter 15). **GIF (Graphic Interchange Format)**, on the other hand, uses the indexed color scheme.

3.5.2 Vector graphics

Raster graphics has two disadvantages: the file size is big and rescaling is troublesome. To enlarge a raster graphics image means enlarging the pixels., so the image looks ragged when it is enlarged. The **vector graphic** image encoding method, however, does not store the bit patterns for each pixel. An image is decomposed into a combination of geometrical shapes such as lines, squares, or circles. For example, consider a circle of radius r . The main pieces of information a program needs to draw this circle are:

1. The radius r and equation of a circle.
2. The location of the center point of the circle.
3. The stroke line style and color.
4. The fill style and color.



3-6 STORING VIDEO

Video is a representation of images (called **frames**) over time. A movie consists of a series of frames shown one after another. In other words, video is the representation of information that changes in space and in time. So, if we know how to store an image inside a computer, we also know how to store video: each image or frame is transformed into a set of bit patterns and stored. The combination of the images then represents the video.



See Chapter 15 for video compression.

Thank You