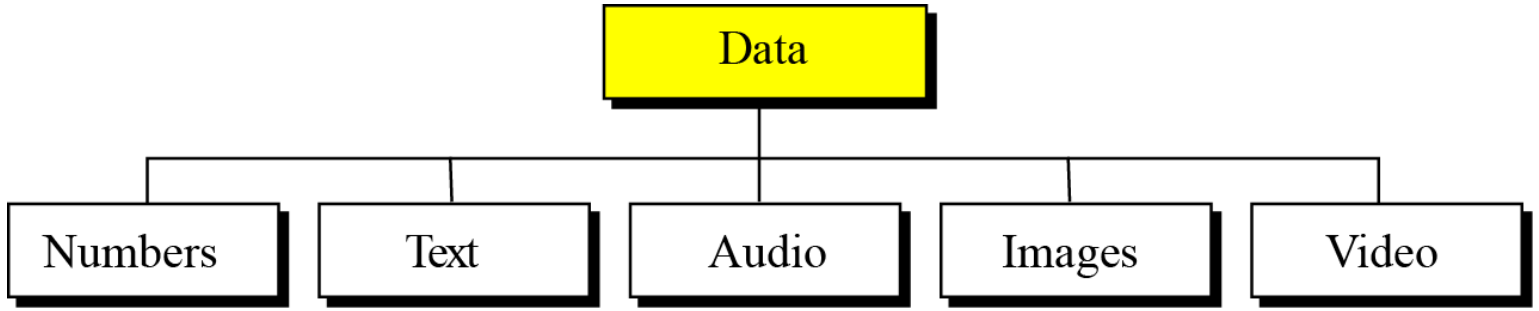
Data Storage

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

Introduction

- Data today come in different forms including numbers, text, audio, image, and video
- The computer industry uses the term '*multimedia*' to define the information that contains numbers, text, audio, images, and video.



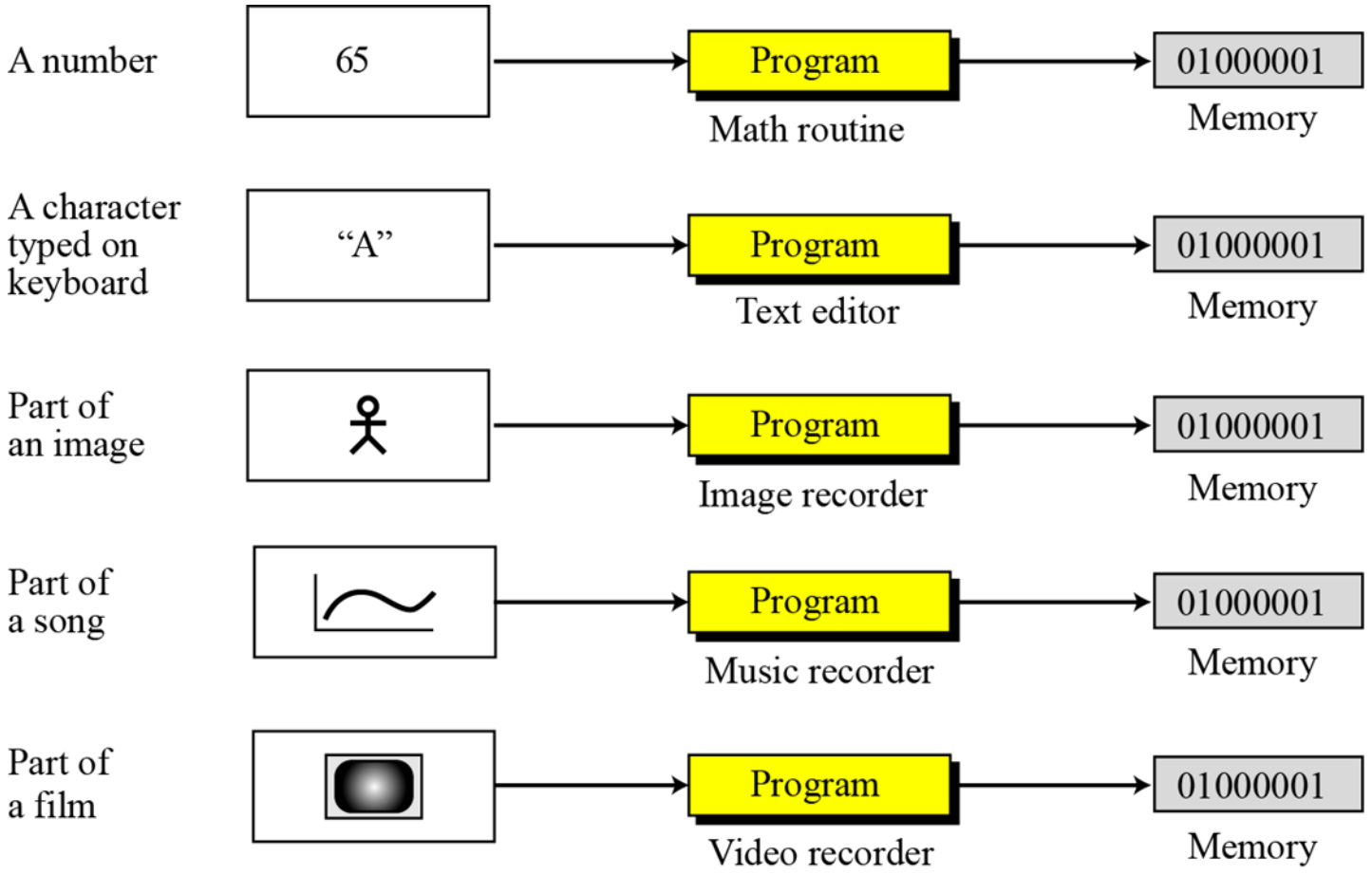
Data inside the computer

- All data types are transformed into a uniform representation when they are stored in a computer and transformed back when retrieved
  - A bit (binary digit) is the smallest unit of data that can be stored in a computer and has a value of 0 or 1
  - To represent different types of data, we use the universal representation called a *bit pattern (位元樣式)* which is a sequence of bits
  - A bit pattern with eight bits is called a *byte (位元組)*

1 0 0 0 1 0 1 0 1 1 1 1 1 1

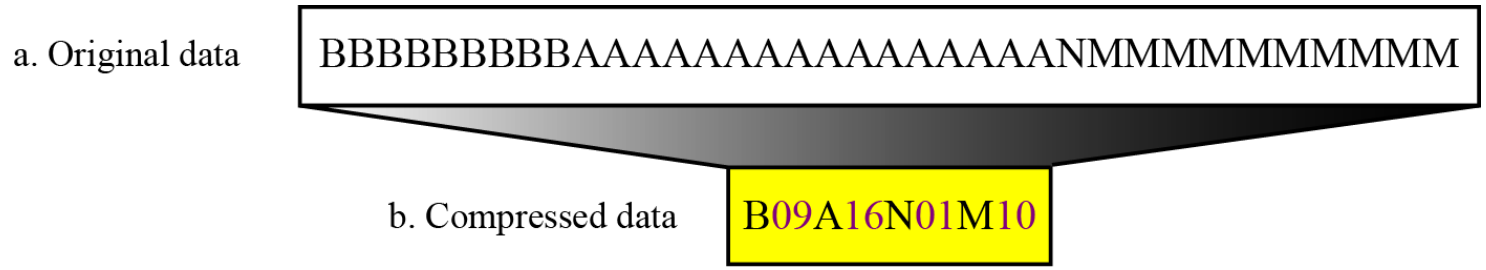
Data inside the computer

- The computer's memory stores all of them without recognizing what type of data they represent

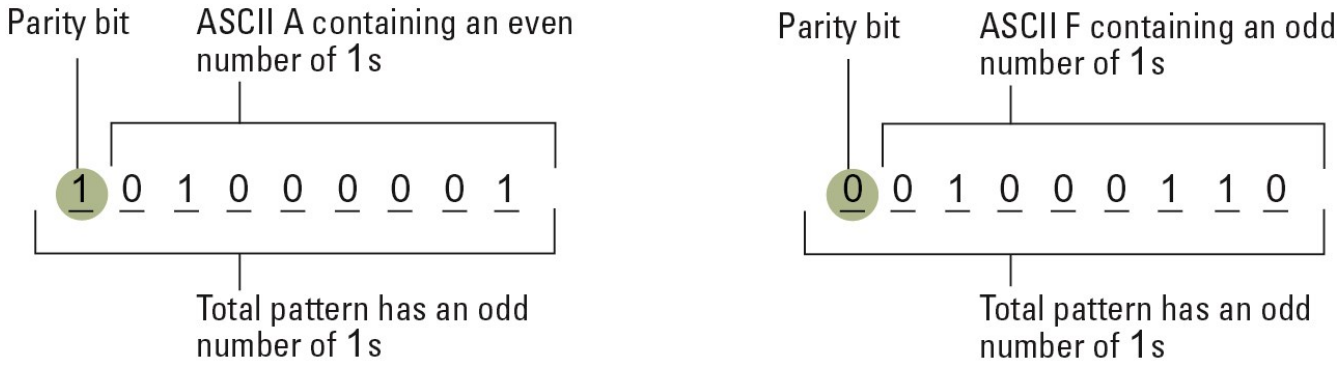


Data compression, error detection and error correction

- To occupy less memory space, data is normally compressed before being stored in the computer



- Another issue related to data is the detection and correction of errors during transmission or storage

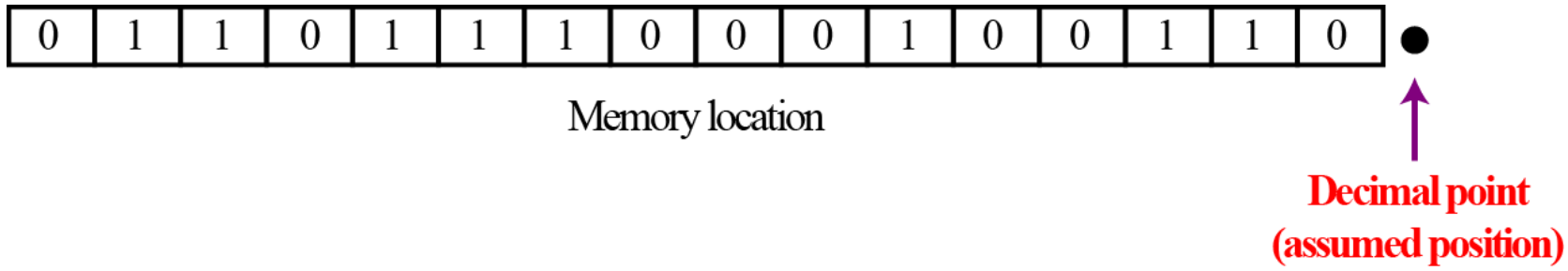


Storing Numbers

- A number is changed to the binary system before being stored in the computer memory. However, there are still two issues that need to be handled
  - How to store the sign of the number
  - How to show the decimal point
    - For the decimal point, computers use two different representations: fixed-point and floating-point. The first is used to store a number as an integer—without a fractional part, and the second is used to store a number as a real—with a fractional part

Storing integers

- An integer can be thought of as a number in which the position of the decimal point is fixed
  - The decimal point is to the right of the least significant (rightmost) bit. For this reason, fixed-point representation is used to store an integer
  - To use computer memory more efficiently unsigned (無號) and signed (有號) integers are stored inside the computer differently



Unsigned representation

- An unsigned integer is an integer that can never be negative and can take only 0 or positive values
  - Its range is between 0 and positive infinity
  - A constant called *maximum unsigned integer* which has a value $2^n - 1$ is used for most computer
- An input device stores an unsigned integer using the following steps:
  - The integer is changed to binary
  - If the number of bits is less than n, 0s are added to the left
- An output device retrieves a bit string from memory as a bit pattern and converts it to an unsigned decimal integer

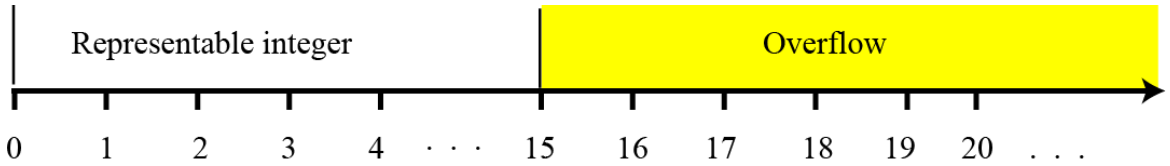
Unsigned representation

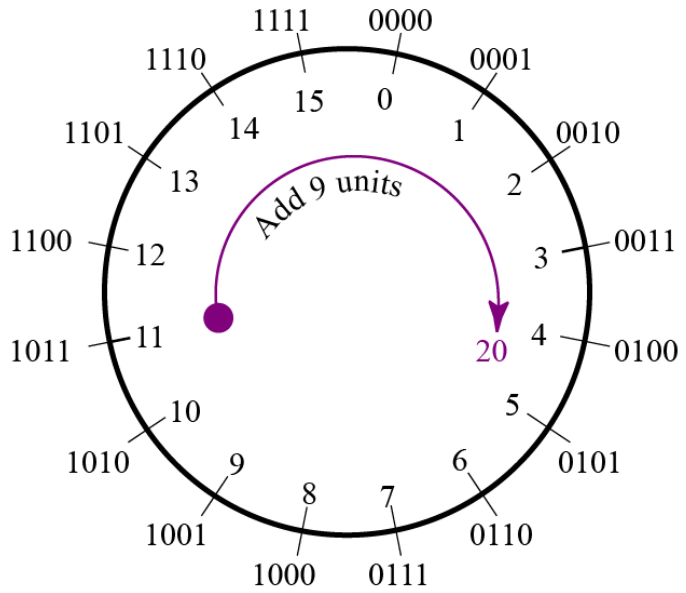
- How to store 7 in an 8-bit memory location using unsigned representation?

- What is returned from an output device when it retrieves the bit string 00101011 stored in the memory as an unsigned integer?

Overflow

- In an n-bit memory location, we can only store an unsigned integer between 0 and $2^n - 1$
  - *Overflow (溢位)* happens when we store a number that is larger than a memory location that can only hold n bits



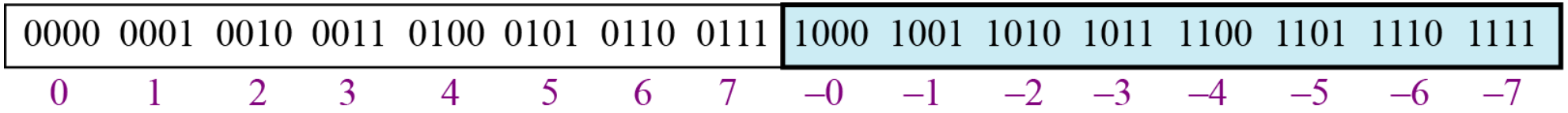


Applications of unsigned integers

- Counter
  - When we count, we do not need negative numbers. We start counting from 1 (sometimes 0) and go up
- Addressing
  - Some computer programs store the address of a memory location inside another memory location. Addresses are positive integers starting from 0 (the first memory location) and going up to an integer representing the total memory capacity
- Storing other data types
  - Other data types (text, images, audio, and video), as we will discuss shortly, are stored as bit patterns, which can be interpreted as unsigned integers

Sign-and-magnitude representation (符號大小表示法)

- In this method, the available range for unsigned integers (0 to $2^n - 1$) is divided into two equal sub-ranges
  - Sign-and-magnitude representation is often used to store part of a real number or when we quantize an analog signal, such as audio
  - The first half represents positive integers, the second half, negative integers
  - In sign-and-magnitude representation, the leftmost bit defines the sign of the integer
    - We have two 0s: positive zero (0000) and negative zero (1000)
    - The range of the numbers that can be stored in an n-bit location is $-(2^{n-1} - 1)$ to $2^{n-1} - 1$



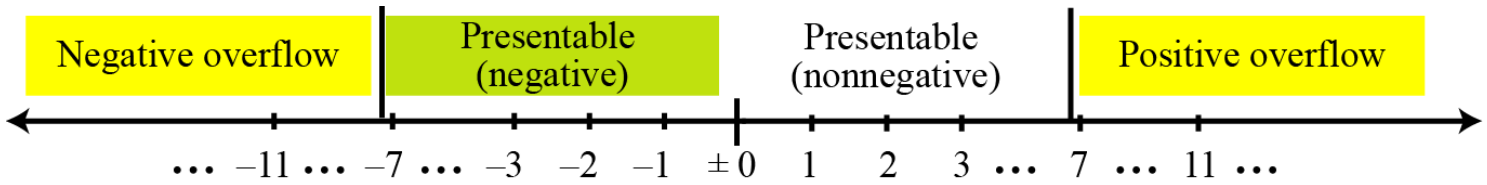
Sign-and-magnitude representation

- Store +28 and -28 in an 8-bit memory using sign-and-magnitude representation

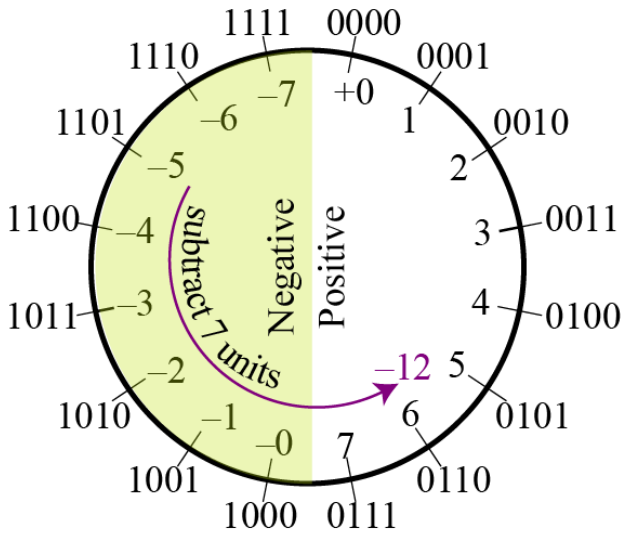
- Retrieve the integer that is stored as 01001101 and 10100001 in sign-and-magnitude representation

Overflow in sign-and-magnitude representation

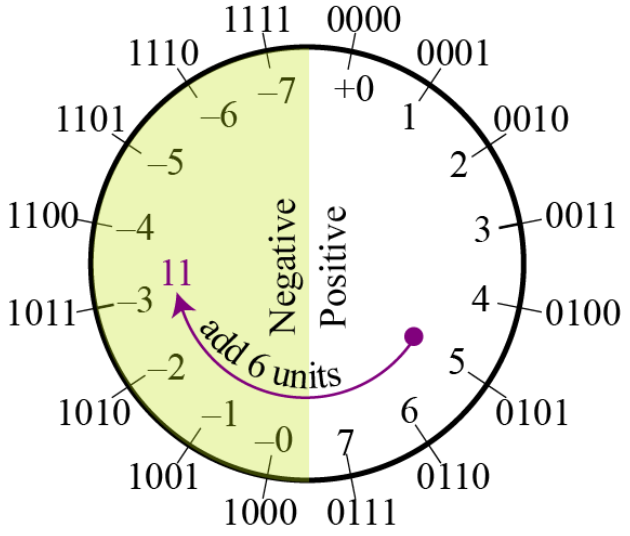
- We may have both positive and negative overflow



a. Linear characteristic of an integer in sign-and-magnitude format



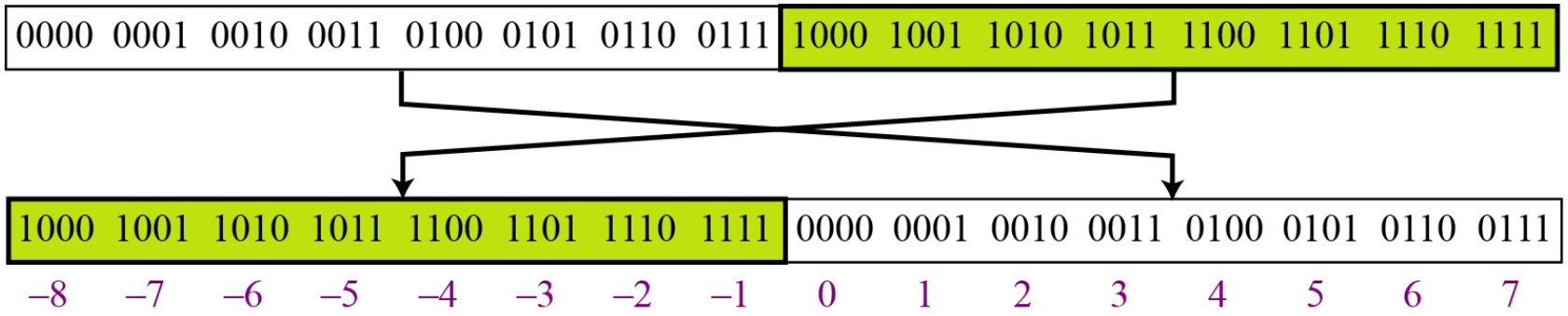
b. Negative overflow



c. Positive overflow

Two's complement representation (2補數表示法)

- To store a signed integer in an n-bit memory location
  - Two's complement representation is the standard representation for storing integers
  - The available range for an unsigned integer of (0 to $2^n - 1$) is divided into two equal subranges
  - The first subrange is used to represent nonnegative integers, and the second half to represent negative integers
  - There is only one zero in two's complement notation



One's Complementing

- Before we discuss this representation further, we need to introduce two operations
  - The first is called *one's completing* (1補數) or taking the one's complement of an integer. The operation can be applied to any integer, positive or negative
  - This operation simply reverses (flips) each bit. A 0-bit is changed to a 1-bit, a 1-bit is changed to a 0-bit
- Try to take the one's complement of the integer 00110110

Two's Complementing

- The second operation is called *two's completing* or taking the two's complement of an integer in binary
  - This operation is done in two steps: First, we copy bits from the right until a 1 is copied; then, we flip the rest of the bits
  - An alternative way to take the two's complement of an integer is to first take the one's complement and then add 1 to the result
- Try to take the two's complement of the integer 00110110

Two's complement representation

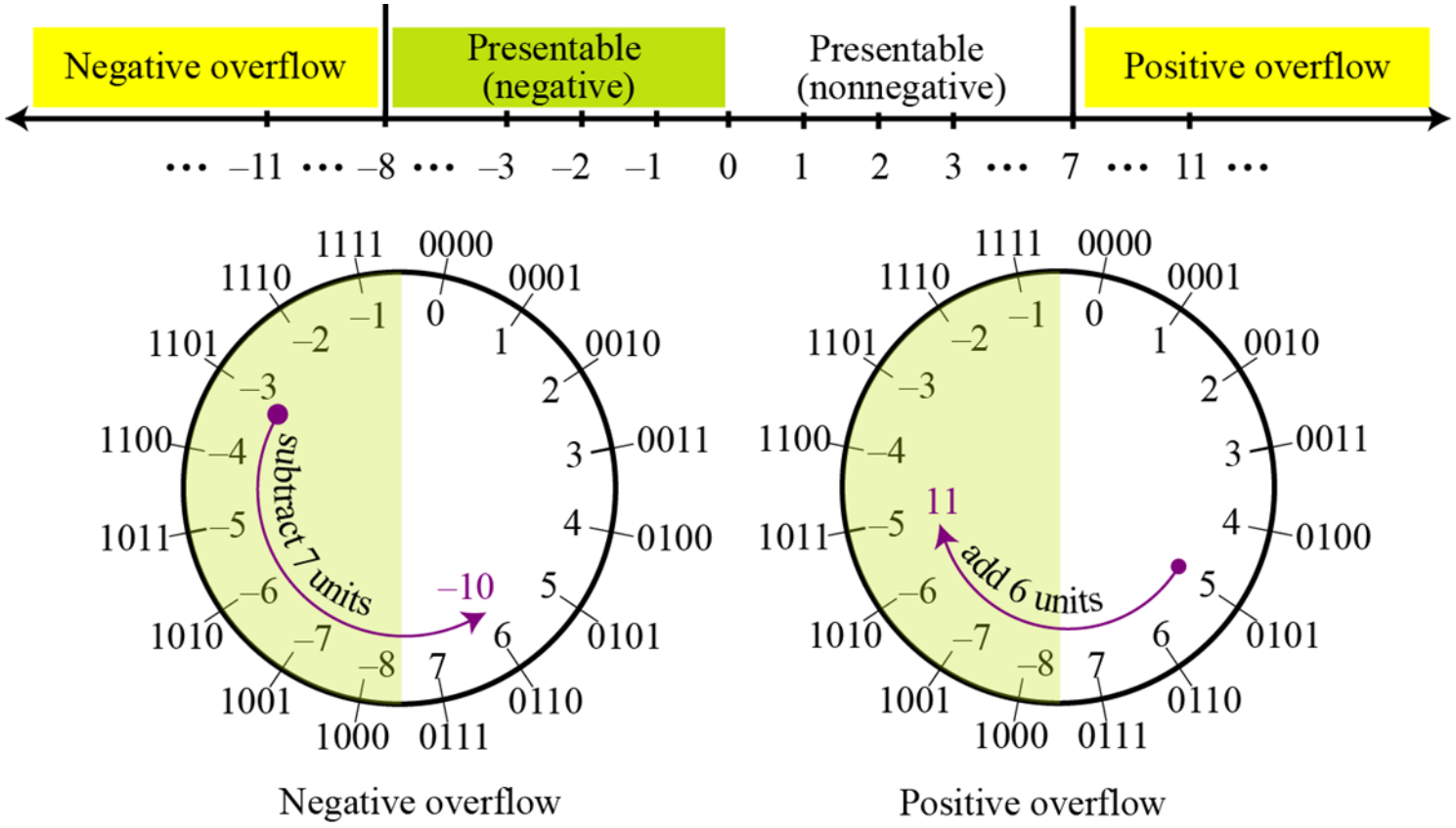
- To store an integer in two's complement representation, the computer follows the steps below:
  - The absolute value of the integer is changed to an n-bit binary
  - If the integer is positive or zero, it is stored as it is: if it is negative, the computer takes the two's complement of the integer and then stores it
- To retrieve an integer in two's complement representation, the computer follows the steps below:
  - If the leftmost bit is 1, the computer applies the two's complement operation to the integer. If the leftmost bit is 0, no operation is applied
  - The computer changes the integer to decimal

Two’s complement representation

- Store +28 and -28 in an 8-bit memory using two’s complement representation

- Retrieve the integer that is stored as 00001101 and 11100110 in memory using two’s complement format

Overflow in two's complement representation



Comparison of the three systems

Table 3.1 Summary of integer representations

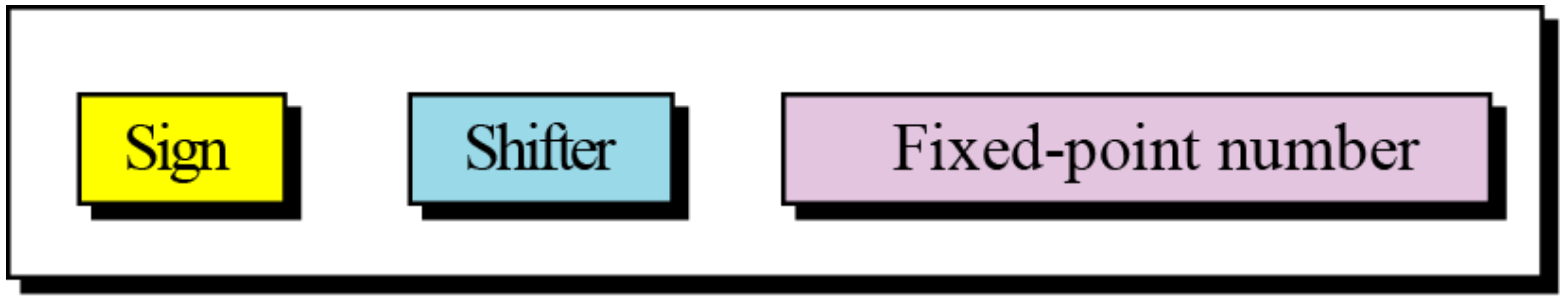
Contents of memory	*Unsigned*	*Sign-and-magnitude*	*Two's complement*
0000	0	0	+0
0001	1	1	+1
0010	2	2	+2
0011	3	3	+3
0100	4	4	+4
0101	5	5	+5
0110	6	6	+6
0111	7	7	+7
1000	8	–0	–8
1001	9	–1	–7
1010	10	–2	–6
1011	11	–3	–5
1100	12	–4	–4
1101	13	–5	–3
1110	14	–6	–2
1111	15	–7	–1

Storing reals

- A real is a number with an integral part and a fractional part
  - 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
  - In the decimal system, assume that we use a fixed-point representation with two digits at the right of the decimal point and fourteen digits at the left
    - 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
  - Assume that we use a fixed-point representation with six digits to the right of the decimal point and ten digits to the left
    - 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!

Floating-point representation (浮點數表示法)

- The solution for maintaining accuracy or precision is to use floating-point representation
  - This representation allows the *decimal point to float*
  - A floating-point representation of a number is made up of three parts
    - The first section is the sign, either positive or negative.
    - The second section shows how many places the decimal point should be shifted to the right or left
    - The third section is a fixed-point representation in which the position of the decimal is fixed



Floating-point pepresentation

Floating-point representation

- It is used in science to represent very small or very large decimal numbers which are called *scientific notation*

Actual number	→	+	7,425,000,000,000,000,000,000.00
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 and the fixed-point section has only one digit to the left of the decimal point

Floating-point representation

- Show the number -0.0000000000000232 in scientific notation

- Show the number $(10100100000000000000000000000000.00)_2$ in floating-point format

- Show the number $-(0.00000000000000000000000101)_2$ in floating-point format

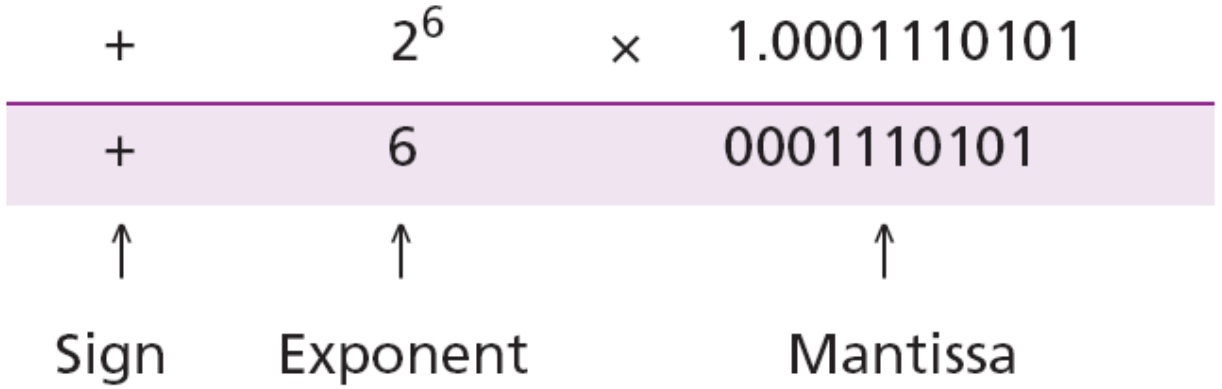
Floating-point representation – Normalization (正規化)

- 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	→	±	d.xxxxxxxxxxxxxx	Note: *d* is 1 to 9 and each *x* is 0 to 9
Binary	→	±	1.yyyyyyyyyyyyyy	Note: each *y* is 0 or 1

Floating-point representation - Sign, exponent, and mantissa

- After a binary number is normalized, only three pieces of information about the number are stored: *sign, exponent (指數), and mantissa (尾數)* (the bits to the right of the decimal point)
  - For example, +1000111.0101 becomes
  - The sign of the number can be stored using 1 bit
  - The mantissa can be stored as an unsigned integer

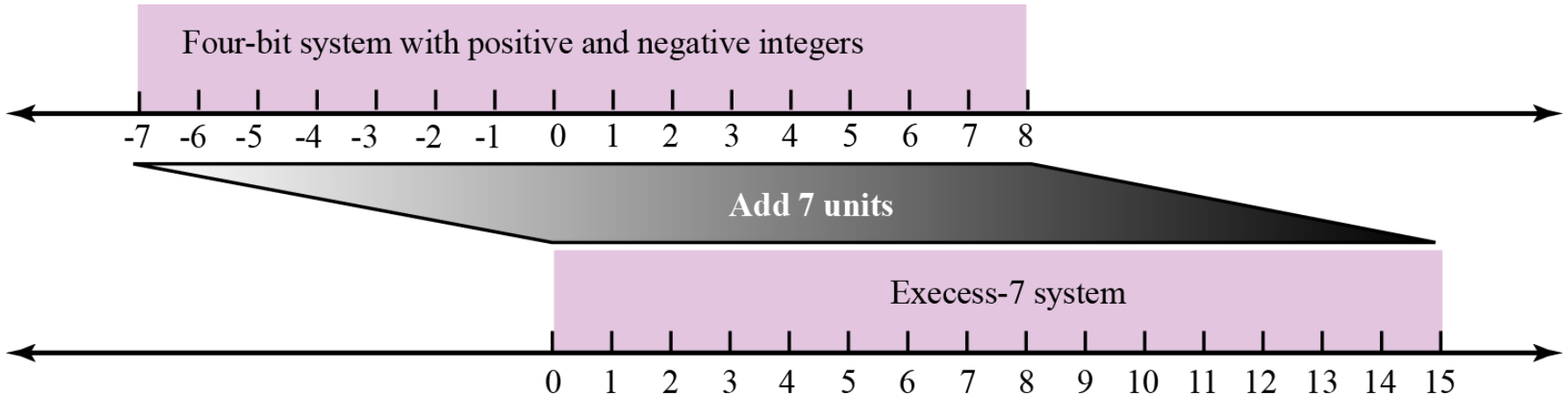


Floating-point representation - The Excess system (超碼系統)

- 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

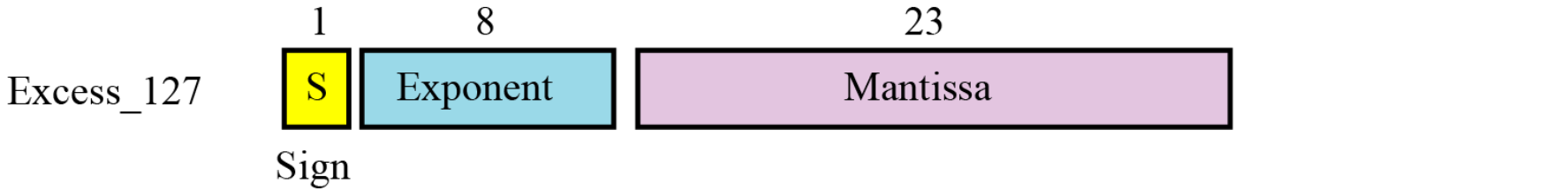
Floating-point representation - The Excess system

- 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 make all of them positive without changing the relative position of the integers with respect to each other
  - The new system is referred to as Excess-7, or biased representation with biasing value of 7
    - We don't need to be concerned about the sign when we are doing operations on the integers

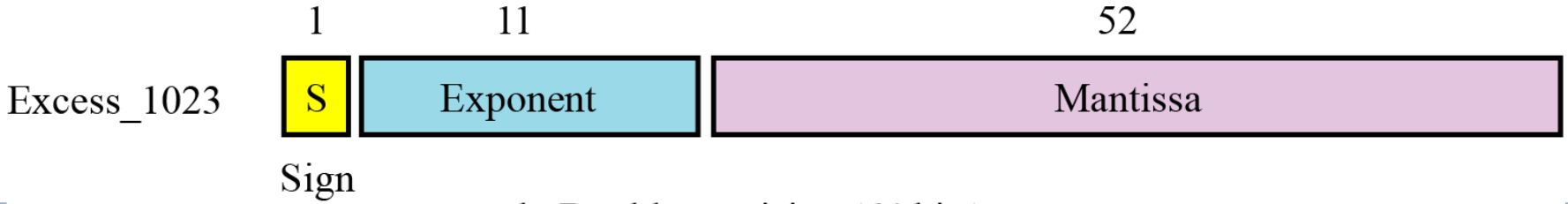


Floating-point representation - IEEE standards

- *Single-precision (單精度)* and *Double-precision (倍精度)* format uses a total of 32 and 64 bits to store a real number
  - Store the sign in S (0 or 1)
  - Change the number to binary and normalize
  - Find the values of E and M
  - Concatenate S, E, and M



a. Single precision (32 bits)



b. Double precision (64 bits)

Floating-point representation - IEEE standards

- A number stored in one of the IEEE floating-point formats can be retrieved using the following method
  - Find the value of S, E, and M
  - If $S = 0$, set the sign to positive, otherwise, set the sign to negative
  - Find the shifter $(E - 127)$
  - Denormalize the mantissa
  - Change the denormalized number to binary to find the absolute value
  - Add the sign

Table 3.2 Specifications of the two IEEE floating-point standards

Parameter	*Single Precision*	*Double Precision*
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

Floating-point representation - IEEE standards

- Show the Excess_127 representation of the decimal number 5.75

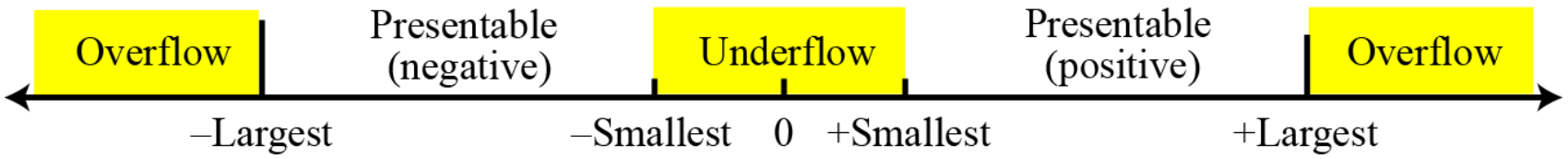
- Show the Excess_127 representation of the decimal number – 161.875

Floating-point representation - IEEE standards

- The bit pattern $(01000011111000000000000000000000)_2$ is stored in Excess_127 format. Show the value in decimal

Overflow and Underflow

- In the case of floating-point numbers, we can have both an overflow and *underflow (下限溢位)*
  - An attempt to store numbers with very small absolute values results in an underflow condition
  - An attempt to store numbers with very large absolute values results in an overflow condition



See https://zh.wikipedia.org/wiki/IEEE_754 and here for more details

Storing zero and truncation errors

- To store zero, the sign, exponent, and the mantissa are set to 0s
- When a real number is stored using floating-point representation, the value of the numbered stored may not be exactly as we expect it to be
  - For example, assume we need to store the number in memory using Excess_127 representation :

$$(111111111111111.11111111111)_2$$

  - After normalization, we have:

$$(1.11111111111111111111111111)_2$$

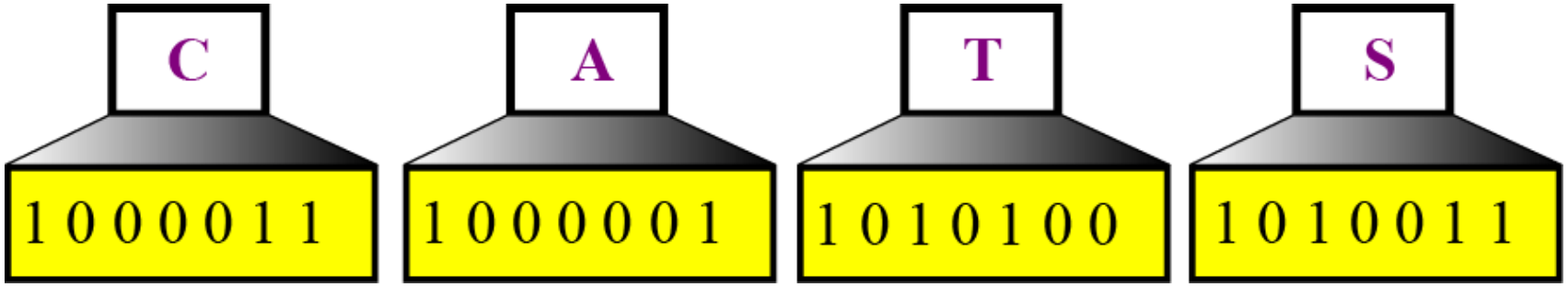
  - This means that the mantissa has 26 1s. This mantissa needs to be truncated to 23 1s

$$(1.11111111111111111111111)_2$$

  - which means the original number is changed and the difference between the original number and what is retrieved is called the *truncation error*

Storing Text

- A section of text in any language is a sequence of symbols used to represent an idea in that language
  - 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 text symbol with a bit pattern just like the number



Storing Text

- How many bits are needed in a bit pattern to represent a symbol in a language?
  - The length depends on the number of symbols and the relationship is logarithmic
  - A bit pattern of two bits can take four different forms: 00, 01, 10, and 11. Each of these forms can represent a symbol

Table 3.3 Number of symbols and bit pattern length

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

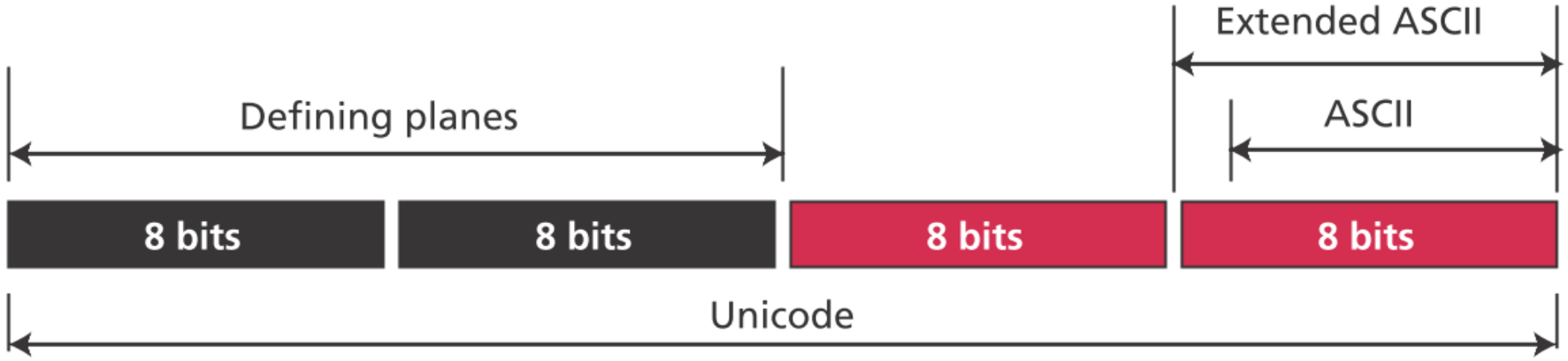
Storing Text - Codes

- Different sets of bit patterns have been designed to represent text symbols
  - Each set is called a *code*, and the process of representing symbols is called *coding*
- The American National Standards Institute (ANSI) developed a code called American Standard Code for Information Interchange (*ASCII*)
  - This code uses 7 bits for each symbol which can represent 128 different characters

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
$(11)_{16}$	DC1	$(31)_{16}$	1	$(51)_{16}$	Q	$(71)_{16}$	q
$(12)_{16}$	DC2	$(32)_{16}$	2	$(52)_{16}$	R	$(72)_{16}$	r
$(13)_{16}$	DC3	$(33)_{16}$	3	$(53)_{16}$	S	$(73)_{16}$	s
$(14)_{16}$	DC4	$(34)_{16}$	4	$(54)_{16}$	T	$(74)_{16}$	t
$(15)_{16}$	NAK	$(35)_{16}$	5	$(55)_{16}$	U	$(75)_{16}$	u
$(16)_{16}$	SYN	$(36)_{16}$	6	$(56)_{16}$	V	$(76)_{16}$	v
$(17)_{16}$	ETB	$(37)_{16}$	7	$(57)_{16}$	W	$(77)_{16}$	w
$(18)_{16}$	CAN	$(38)_{16}$	8	$(58)_{16}$	X	$(78)_{16}$	x
$(19)_{16}$	EM	$(39)_{16}$	9	$(59)_{16}$	Y	$(79)_{16}$	y
$(1A)_{16}$	SUB	$(3A)_{16}$	:	$(5A)_{16}$	Z	$(7A)_{16}$	z
$(1B)_{16}$	ESC	$(3B)_{16}$	;	$(5B)_{16}$	[	$(7B)_{16}$	{
$(1C)_{16}$	FS	$(3C)_{16}$	<	$(5C)_{16}$	\	$(7C)_{16}$	|
$(1D)_{16}$	GS	$(3D)_{16}$	=	$(5D)_{16}$	]	$(7D)_{16}$	}
$(1E)_{16}$	RS	$(3E)_{16}$	>	$(5E)_{16}$	^	$(7E)_{16}$	~
$(1F)_{16}$	US	$(3F)_{16}$	?	$(5F)_{16}$	_	$(7F)_{16}$	DEL

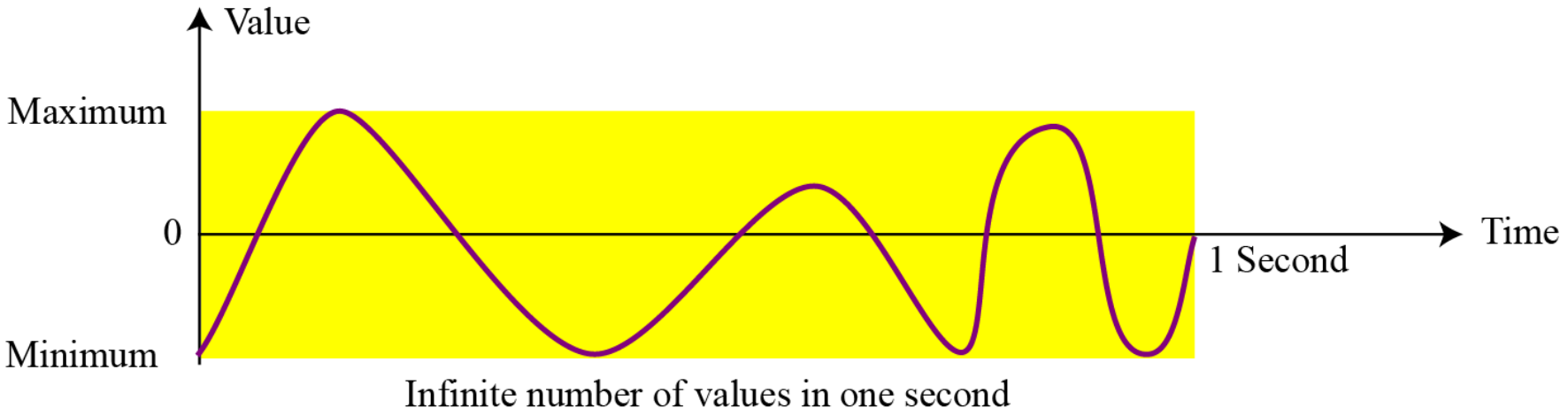
Storing Text - Codes

- A coalition of hardware and software manufacturers have designed a code called *Unicode*
  - It uses 32 bits and can therefore represent up to 2^{32} symbols
  - Different sections of the code are allocated to symbols from different languages in the world
  - It is fully compatible with ASCII and Extended ASCII



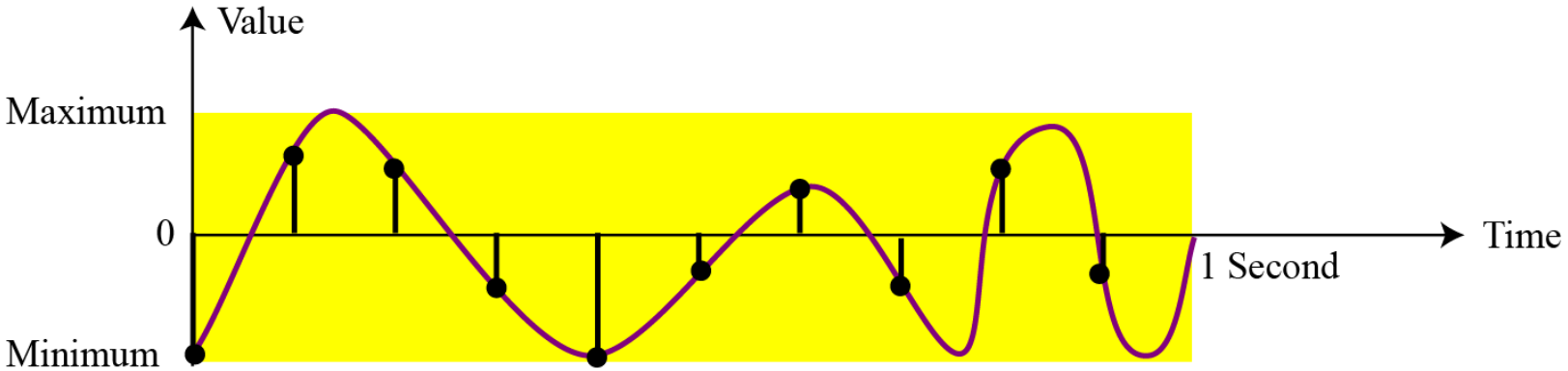
Storing Audio

- Audio is a representation of sound or music
  - The numbers or text are *digital* data that are composed of countable entities
  - Audio is an example of *analog* data which is not countable
  - We store the intensity of an audio signal over a period of time



Storing Audio - Sampling

- We can record some of the audio signals over an interval
  - *Sampling* means that we select only a finite number of points on the analog signal, measure their values, and record them
- How many samples do we need in each second?
  - It has been shown that a *sampling rate (取樣頻率)* of 40,000 samples per second is good enough to reproduce an audio signal



Ten sample values in one second

Storing Audio – Quantization and Encoding

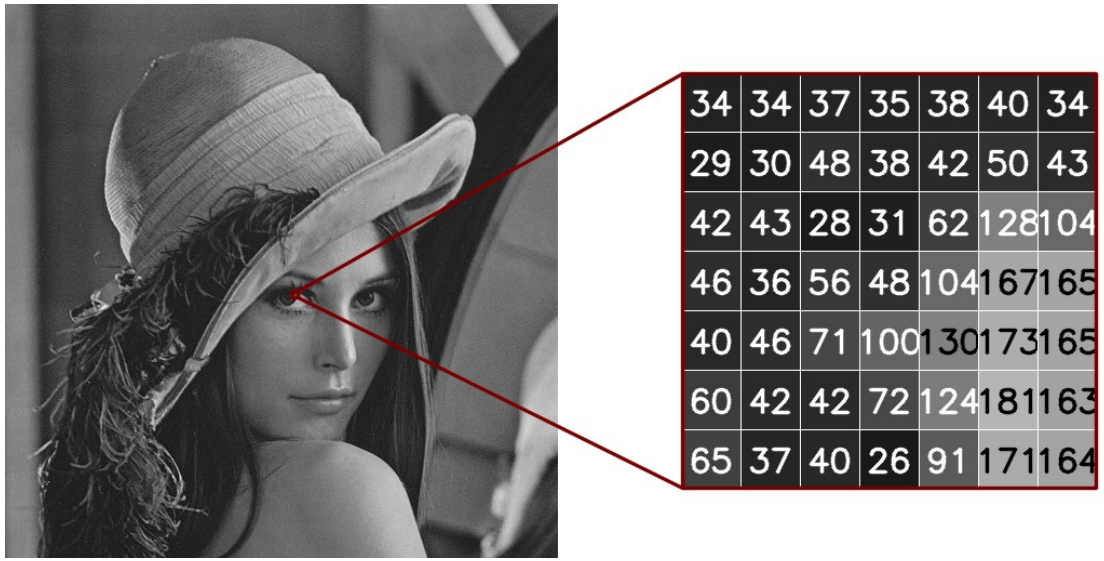
- We can store 40,000 real values for each one-second sample
  - However, it is simpler to use an integer (a bit pattern) for each sample
  - *Quantization (量化)* refers to rounding the value of a sample to the closest integer value
- The quantized sample values need to be *encoded* as bit patterns
  - Some systems use an unsigned integer to represent a sample
  - While others use signed integers and use the two's complement or the sign-and-magnitude format
- How many bits do we need for a sample?
  - The number of bits per sample is sometimes referred to as the *bit depth (位元深度)*
    - 8, 16, 24 and 32 bits per sample is normal

Storing Audio – Quantization and Encoding

- 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 quantity is called *bit rate (位元率)*
- Today the dominant standard for storing audio is MP3 (MPEG Layer 3)
  - This standard is a modification of the MPEG (Motion Picture Experts Group) compression method used for video
  - It uses 44,100 samples per second and 16 bits per sample
  - It is then compressed using a compression method that discards information that cannot be detected by the human ear

Storing Images - Raster graphics (點掃繪圖法)

- *Raster graphics* (or *bitmap graphics*) is used when we need to store an analog image such as a photograph
  - The intensity (color) of data varies in space instead of in time
  - Sampling in this case is normally called *scanning* and the samples are called *pixels (像素)*
  - In other words, the whole image is divided into small pixels where each pixel is assumed to have a single intensity value



http://kurauchi.com.br/img/post/means-medians-and-images/img2.png

Storing Images - Raster graphics

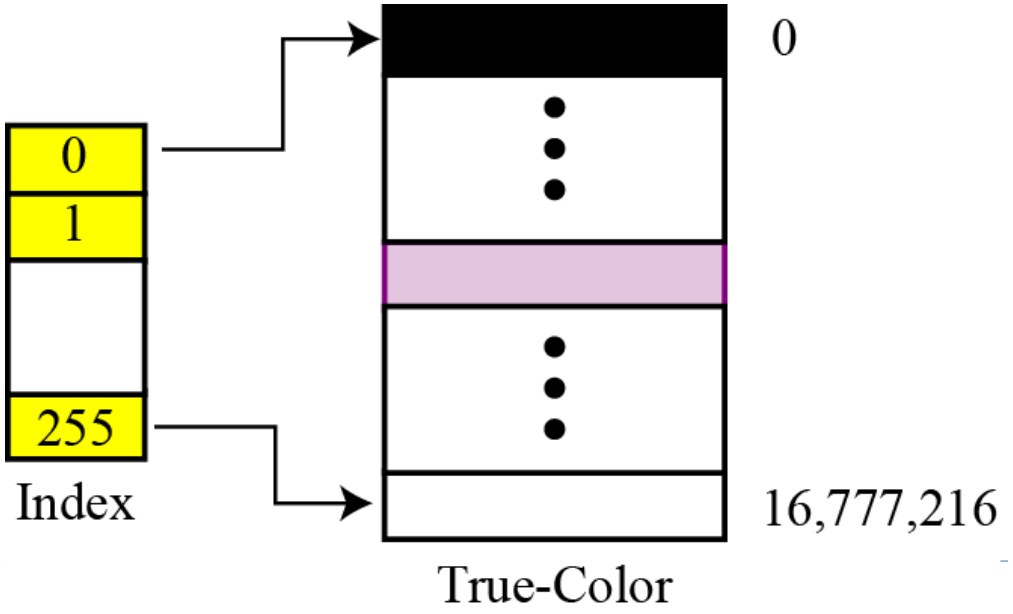
- How many pixels do we need to record for each square and how many bits do we need to represent a pixel?
  - The scanning rate in image processing is called the *resolution* and the number of bits used to represent a pixel is called the *color depth (色彩深度)*
  - *True-Color (全彩)* uses 24 bits to encode a pixel and each of the three primary colors (*RGB*) is represented by eight bits

Table 3.4 Some colors defined in True-Color

Color	*Red*	*Green*	*Blue*	*Color*	*Red*	*Green*	*Blue*
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

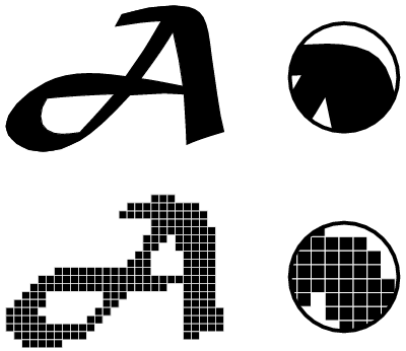
Storing Images - Raster graphics

- The True-Color scheme uses more than 16 million colors!
  - The *indexed color (索引顏色)*—or *palette color*— uses only a portion of these colors
  - 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
  - JPEG (Joint Photographic Experts Group) uses the True-Color scheme but compresses the image. GIF (Graphic Interchange Format) uses the indexed color scheme instead



Storing Images - Vector graphics (向量繪圖法)

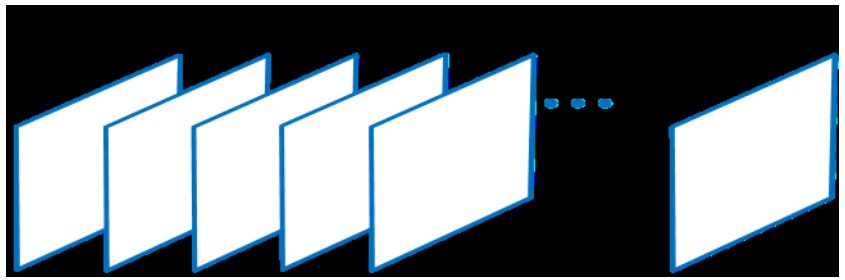
- Raster graphics has two disadvantages: the file size is big and rescaling is troublesome
- The *vector graphic* encoding does not store the bit patterns for each pixel
  - An image is decomposed into a combination of geometrical shapes such as lines, squares, or circles. Each geometrical shape is represented by a mathematical formula
  - Consider a circle of radius r:
    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

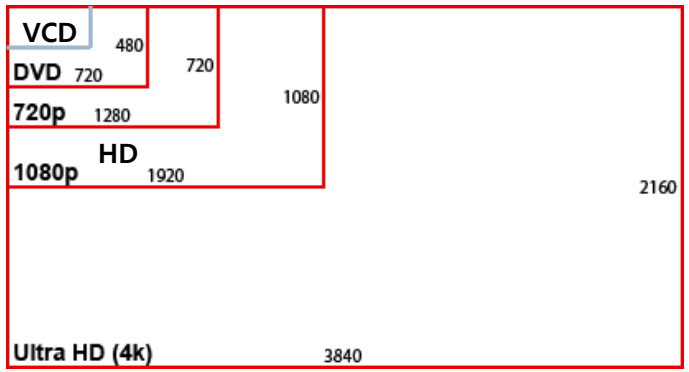


  - It is used in applications such as to create TrueType (Microsoft, Apple) and PostScript (Adobe) fonts. Computer-aided design (CAD) also uses vector graphics for engineering drawings

Storing Videos

- Video is a representation of images (called *frames (畫面)*) over time
  - A movie consists of a series of frames shown one after another to create the illusion of motion
    - The human visual system can process 10 to 12 images per second and perceive them individually, while higher rates are perceived as motion
    - Frames per second (FPS) is the frequency at which consecutive frames are captured or display
  - Today video is normally compressed using techniques such as MPEG

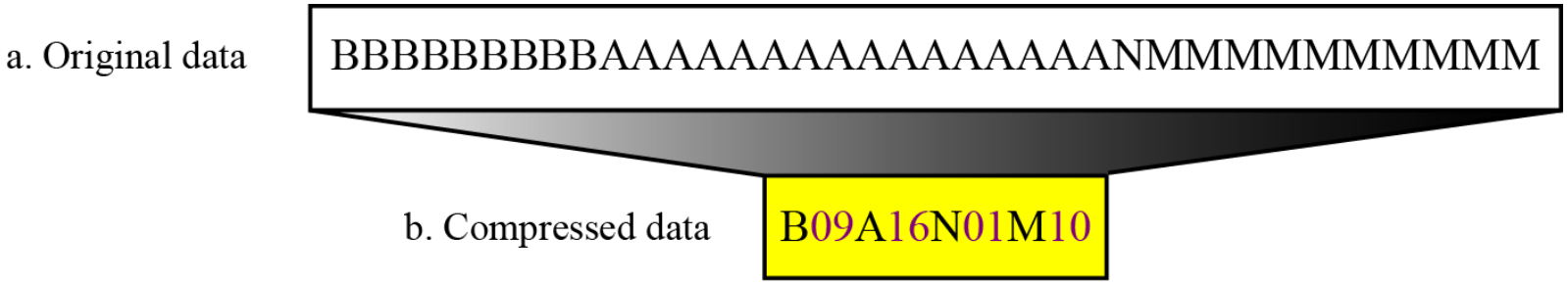




Appendix

Data compression

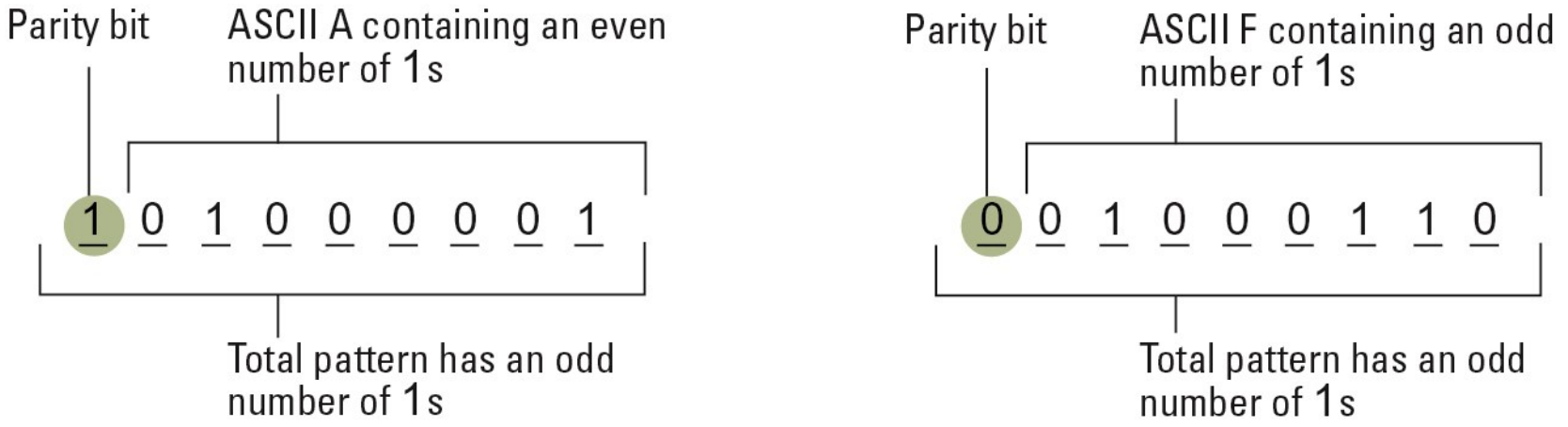
- Data compression schemes fall into two categories: *lossless* and *lossy*
  - Compressing data can reduce the amount of data to be sent or stored by partially eliminating inherent redundancy
- Lossless compression methods are used when we cannot afford to lose any data
  - For instance, *Run-length encoding* replaces consecutive repeating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences



- Lossy techniques often provide more compression than lossless ones and are therefore popular in settings in which minor errors can be tolerated

Error detection and correction

- For most applications, a system must guarantee that the data received is identical to the data transmitted
- A simple method of detecting errors is to ensure that the correct pattern in the system contains an odd number of 1s by adding the *parity bit*
  - For instance, we would obtain a 9-bit encoding system for ASCII code in which an error would be indicated by any 9-bit pattern with an even number of 1s



Error detection and correction

- *Error-correcting codes* can be designed so that errors can be not only detected but also corrected
  - For instance, the following codes based on *Hamming distance* can be used to detect up to two errors per pattern and to correct one error

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

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