
Chapter 1 Data Representation

99.10.11

References

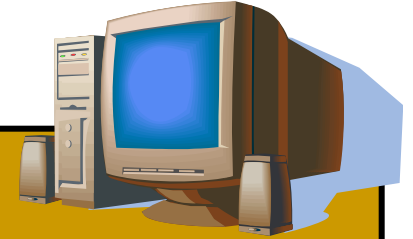
- <http://www.cs.nthu.edu.tw/~king/courses/cs4100.html> (清大金仲達老師)
- *Computer Science: An Overview (10th ed)* by J. Glenn Brookshear




Outline

- 1. The Binary System: Integers, Fractions (Sec 1.5~ 1.7)
- 2. Representing Information as Bit Patterns (Sec 1.4)
- 3. Binary operations and logic gates (Sec 1.1)
- 4. Data Storage (Sec 1.2~1.3)
- 5. Data Processing (Sec 1.8~1.9)

0/1

Computers use 0 and 1 to represent and store all kinds of data



What we see/hear		Inside computers
Text	a,b,c	01100001,01100010,01100011
Number	1,2,3	00000001,00000010,00000011
Sound		01001100010101000110100...
Image		10001001010100000100111...
Video		00110000001001101011001...



Discrete/digital and binary!

Definition

- **Bit:** a binary digit (0 or 1)
- **Byte:** 8 bits
Basic storage unit in computer system
- **Kilobyte:** 2^{10} bytes = 1024 bytes $\approx 10^3$ bytes
 - Example: 3 **KB** $\approx 3 \times 10^3$ bytes
- **Megabyte:** 2^{20} bytes $\approx 10^6$ bytes
 - Example: 3 **MB** $\approx 3 \times 10^6$ bytes
- **Gigabyte:** 2^{30} bytes $\approx 10^9$ bytes
 - Example: 3 **GB** $\approx 3 \times 10^9$ bytes
- **Terabyte:** 2^{40} bytes $\approx 10^{12}$ bytes
 - Example: 3 **TB** $\approx 3 \times 10^{12}$ bytes

Hexadecimal Notation

- Internally, computers store and process 0 and 1 (bits)
- But, it is hard for humans to deal with writing, reading and remembering bits
- Hexadecimal notations help humans to communicate
- Represents each 4 bits by a single symbol
 - Example: A3 denotes 1010 0011

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

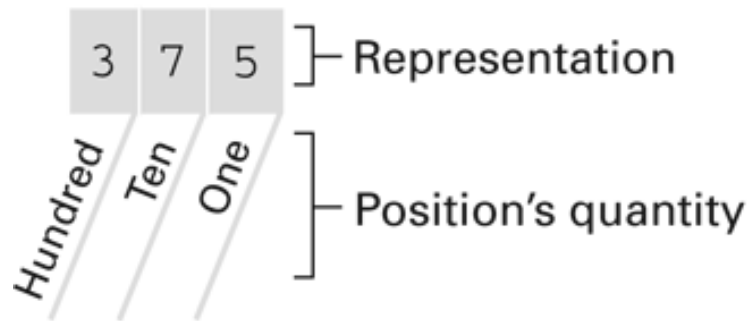
Outline

- 1. The Binary System: Integers, Fractions (Sec 1.5~ 1.7)
 - 1.1 Binary Numeral System
 - 1.2 2's Complement Notation
 - 1.3 Fractions
 - 1.4 Float Point
- 2. Representing Information as Bit Patterns (Sec 1.4)
- 3. Binary operations and logic gates (Sec 1.1)
- 4. Data Storage (Sec 1.2~1.3)
- 5. Data Processing (Sec 1.8~1.9)

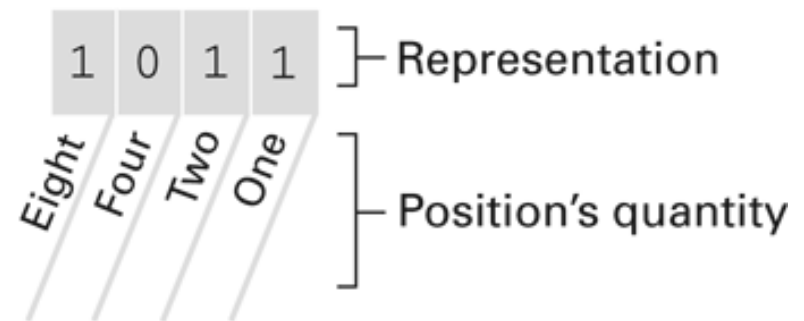
1.1 Binary Numeral System

- Uses bits to represent a number in **base-2**

a. Base ten system



b. Base two system

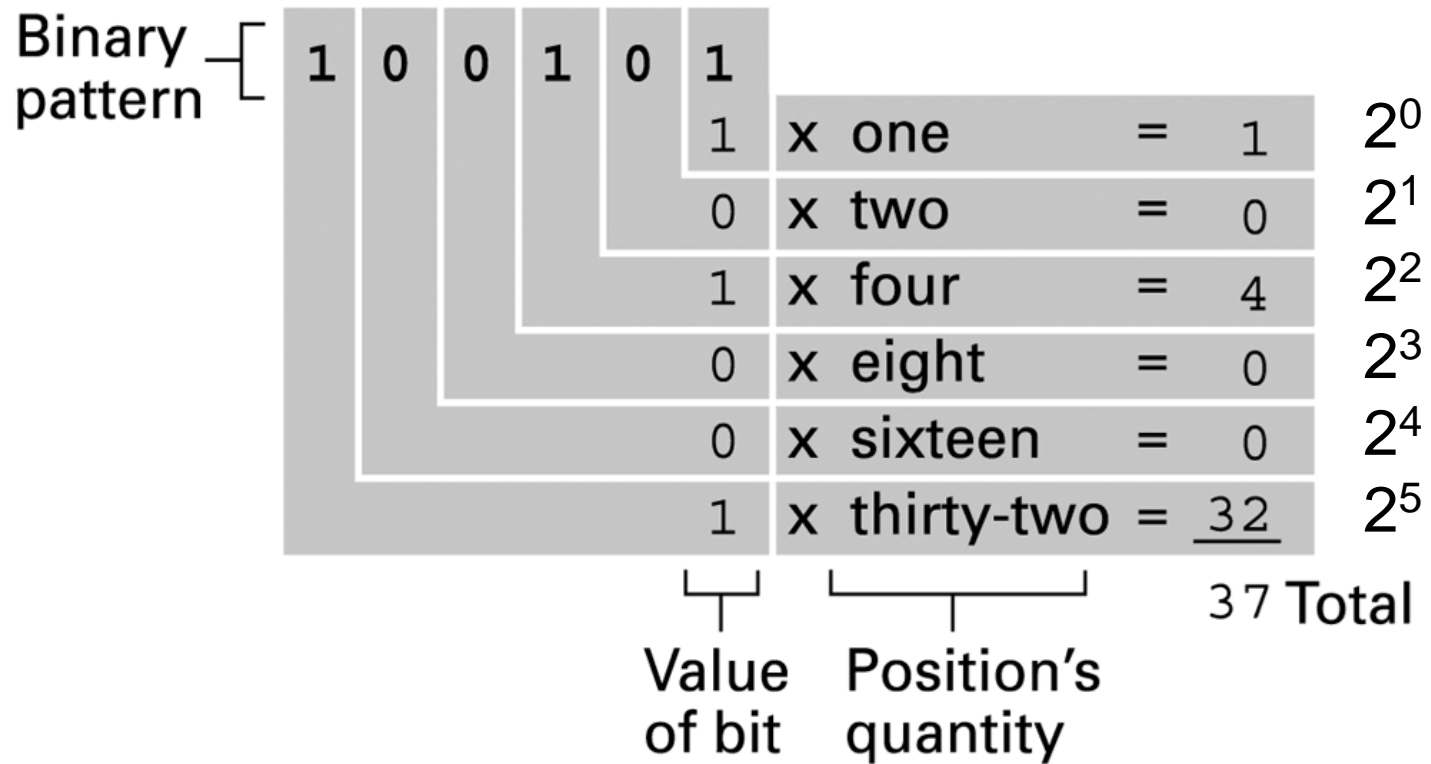


(Fig. 1.15)

- We put a subscript **b** to a number for binary, and a subscript **d** for decimal
 - 10_d is number ten, and 10_b is number two

Binary to Decimal

- What is the decimal number of 100101_b ?



(Fig. 1.16)

Decimal to Binary

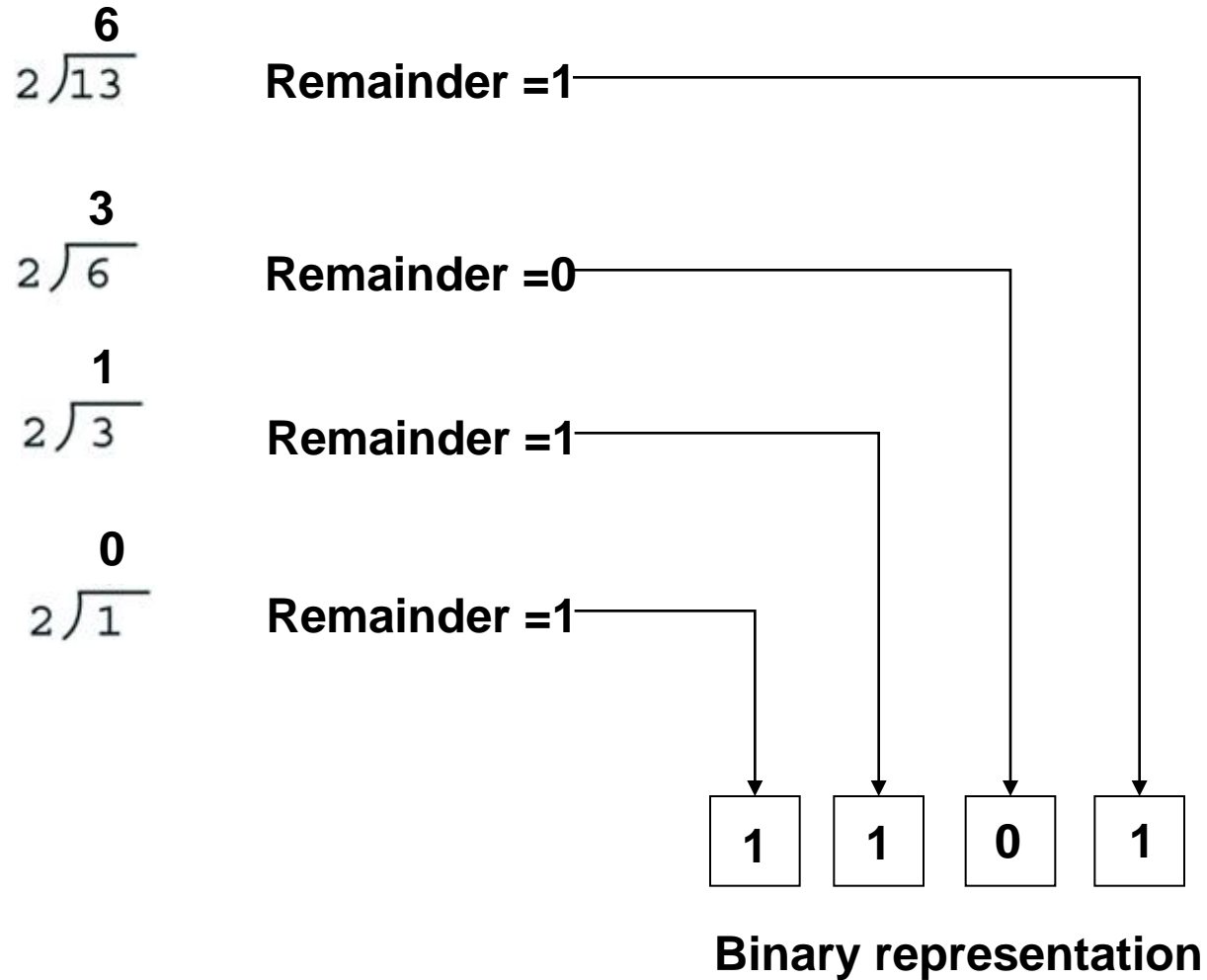
- What is the binary number of 13_d ?
 - How many bits we need for 13?
 - Since $13 < 16 = 2^4$, 4 bits can represent 13

$$13 = \boxed{b_3} \boxed{b_2} \boxed{b_1} \boxed{b_0} = b_3 \times 8 + b_2 \times 4 + b_1 \times 2 + b_0 \times 1$$

- 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.

(Fig. 1.17)

Running the Algorithm



(Fig. 1.18)

Binary Number Calculations

- Binary number is easy for calculations
- For example, the one bit addition

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

(Fig. 1.19)

Carry

- So, what is $5_d + 9_d$ in binary number form?

$$\begin{array}{r} 0101 \\ + 1001 \\ \hline 1110 \end{array}$$

5
9
14

Another Example

$$\begin{array}{r} \text{111 1} \\ 00111010 \\ + 00011011 \\ \hline 01010101 \end{array}$$

Carry

0	1	0	1
+0	+0	+1	+1
0	1	1	10

The binary addition facts

1.2 Negative Numbers (Sec. 1.6)

How to represent -1, -2, ... on a computer?

- **Solution 1:** use an extra bit to represent the negative sign
 - ❑ It is called *the sign bit*, in front of numbers
 - ❑ Usually, 0 is for positives; 1 is for negatives
 - ❑ Example: 1 0001 is -1 and 0 0100 is +4
- **Note:** *The sign bit does not carry value (it is not part of the value)*

4-bit Representation

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

Decimal	Hexadecimal	Binary
+0	0	0 000
+1	1	0 001
+2	2	0 010
+3	3	0 011
+4	4	0 100
+5	5	0 101
+6	6	0 110
+7	7	0 111
-0	8	1 000
-1	9	1 001
-2	A	1 010
-3	B	1 011
-4	C	1 100
-5	D	1 101
-6	E	1 110
-7	F	1 111

Solution-1 Representation

- Example: $\boxed{1}001$ is -1 and $\boxed{0}100$ is +4
- How can we do the addition $(-1) + (4)$ efficiently?
- **Question:**
Can we use “addition” to do addition and subtraction?
with and without signs
- **Solution idea:**
Use a different representation!

Solution 2

- The negative sign “−” just means the “opposite” or the “inverse”
 - For addition, the inverse of a number d , denoted $I(d)$, has the property: $I(d)+d=0$
 - We can use this to define negative numbers

- If we use four bits to represent a number, zero is 0000, and one is 0001. What is -1?

- Find b_3, b_2, b_1, b_0 such that

This 1 will be “dropped” since it is a 4 bits numbering system.

$$\begin{array}{r}
 b_3 \ b_2 \ b_1 \ b_0 \\
 + \ 0 \ 0 \ 0 \ 1 \\
 \hline
 \boxed{1} \ 0 \ 0 \ 0 \ 0
 \end{array}$$

- The solution is 1111
- You can use the same method to find other numbers
- Observe: the leading bit is 1 for negative values → *sign bit*

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

(Fig. 1.21)

Two's Complement

- A simple algorithm to find the inverse
Step1. Change each bit 0 to 1 and bit 1 to 0
Step 2. Add 1

$$\begin{array}{rcl} 6_d = 0110_b & \xrightarrow{\text{invert}} & 1001_b \\ \downarrow & & \downarrow \\ 1001_b & + & 0001_b \\ \hline & & 1010_b = -6_d \end{array}$$

$$\begin{array}{rcl} 1001_b & + & 1010_b & (6) \\ \hline & & 1\ 0000_b & (-6) \\ & & & (0) \end{array}$$

truncated (points to the carry bit 1)

Exercises

Calculation with 2's Complement

- Calculation can be made easy for two's complement representation

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

(Fig. 1.22)

Overflow

- Suppose computer only allow 4 bits
- What is $5+4$?

$$5_d + 4_d = 0101_b + 0100_b = 1001_b$$

- This is called **overflow**
 - ❑ Adding two positive (negative) numbers results in a negative (positive) number
→ Check Sign bit!
 - ❑ A 4-bit 2's complement system can only represent $7 \sim -8$

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

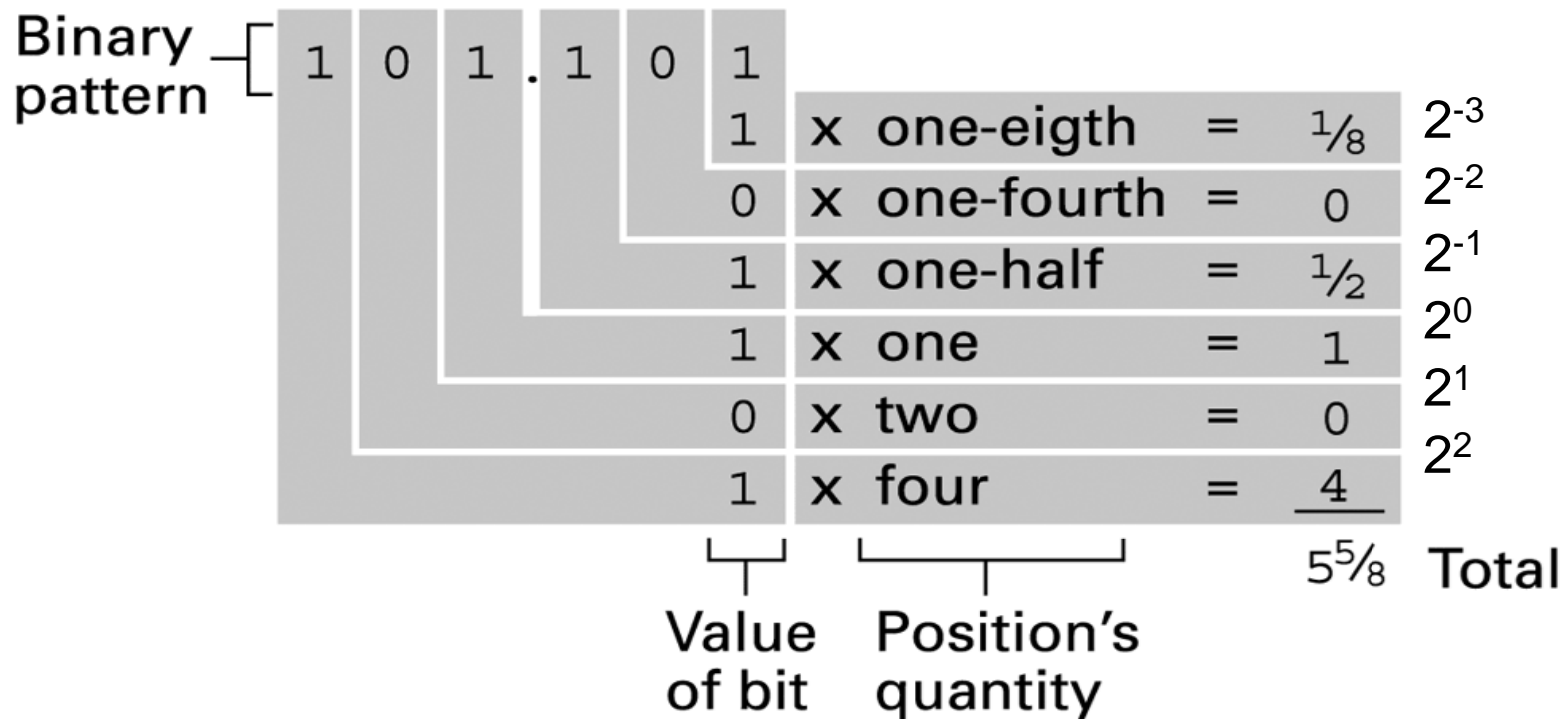
Excess Notation

- Relation to Binary Numerical Notation?

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

1.3 Fractions

- The binary representation of fractions
 - Problem: where to put the decimal point?



(Fig. 1.20)

1.4 Floating Point

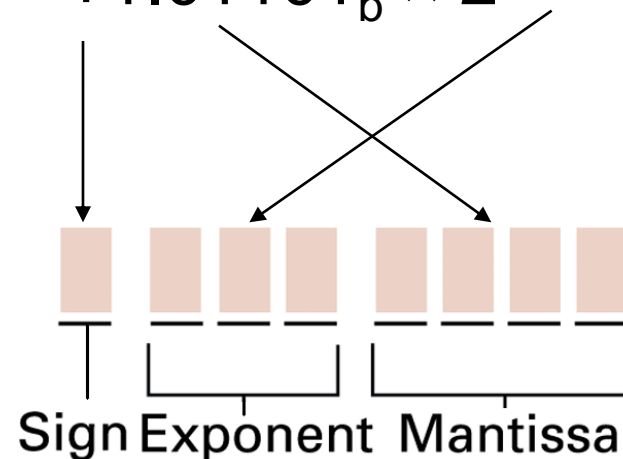
- To represent a wide range of numbers, we allow the decimal point to “float”

$$40.1_d = 4.01_d \times 10^1 = 401_d \times 10^{-1} = 0.401_d \times 10^2$$

- It is just like the scientific notation of numbers

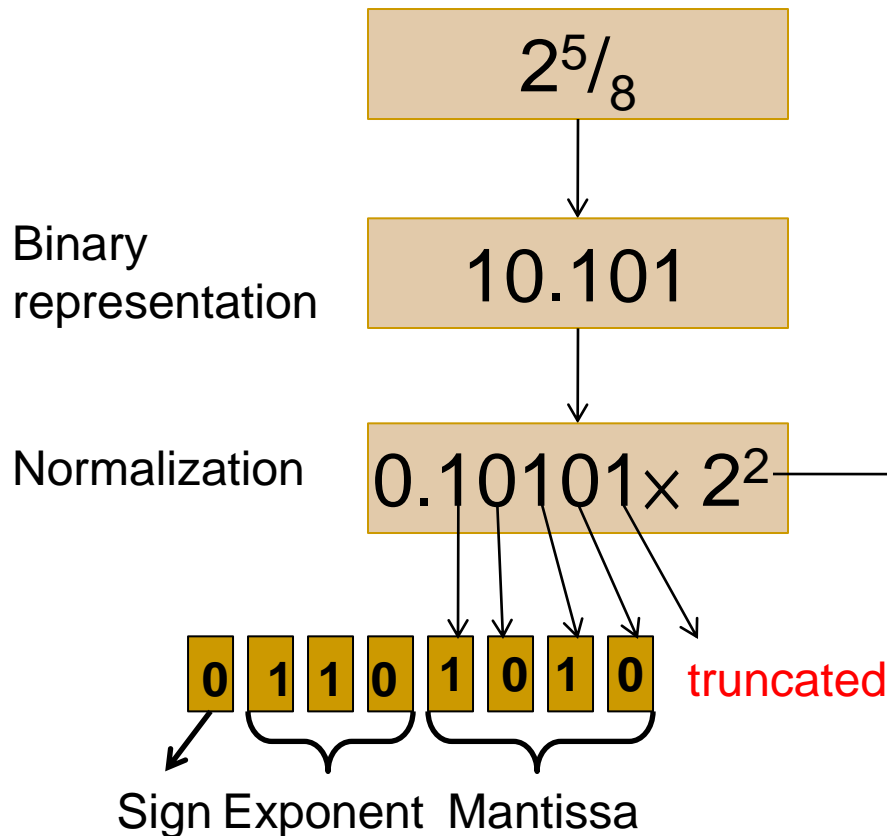
$$101.101_b = +1.01101_b \times 2^{2_d} = +1.01101_b \times 2^{10_b}$$

- This is called the **floating point** representation of fractions



Note: Exponent has a sign too! (Fig. 1.26)

Coding the Value of $2^5/8$



(Fig. 1.27)

- Exponent uses excess notation

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

(Fig. 1.25) "000" represents the smaller number

Truncation Error

- Mantissa field is not large enough
 - $2^5/8 = 2.625 \Rightarrow \mathbf{2.5 + round\ off\ error\ (1/8=0.125)}$
- Solution? Now 32 bits for float point.
- Order of computation: (p.60)
 $2.5 + 0.125 + 0.125 \Rightarrow \mathbf{2.5 + 0 + 0}$

Outline

- 1. The Binary System: Integers, Fractions (Sec 1.5~ 1.7)
- 2. Representing Information as Bit Patterns (Sec 1.4)
 - Text, Image, Sound, Video
- 3. Binary operations and logic gates (Sec 1.1)
- 4. Data Storage (Sec 1.2~1.3)
- 5. Data Processing (Sec 1.8~1.9)

Text Data

- Each character is assigned a **unique bit pattern**
- ASCII code
 - ❑ **American Standard Code for Information Interchange**
 - ❑ Uses **7 bits** to represent most symbols used in English text

```
!"#$%&'()*+,-./  
0123456789:;<=>?  
@ABCDEFGHIJKLMNO  
PQRSTUVWXYZ[\]^_  
`abcdefghijklmnopqrstuvwxyz{|}~
```

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

(Fig. 1.13)

Big5 Code

- For Chinese character encoding
- Uses 16 bits to represent a character
 - 1st byte: 0x81 (1000 0001) ~ 0xfe (1111 1110)
 - Second byte: 0x40 to 0x7e, 0xa1 to 0xfe
 - But does not use all (A140-F9FF)
- Example

我	身	騎	白	馬
A7DA	A8AD	C34D	A5D5	B0A8

Unicode

- Uses **16 bits** (65,536 symbols) to represent the major symbols used in languages worldwide

ت	چ	ي
FB62	FB72	FB82
ت	چ	ي
FB63	FB73	FB83

Arabic char

偁	倅	倅
3460	3470	3480
倅	倅	倅
3461	3471	3481

CJK char

Ê	Ú	ê
00CA	00DA	00EA
Ë	Û	ë
00CB	00DB	00EB

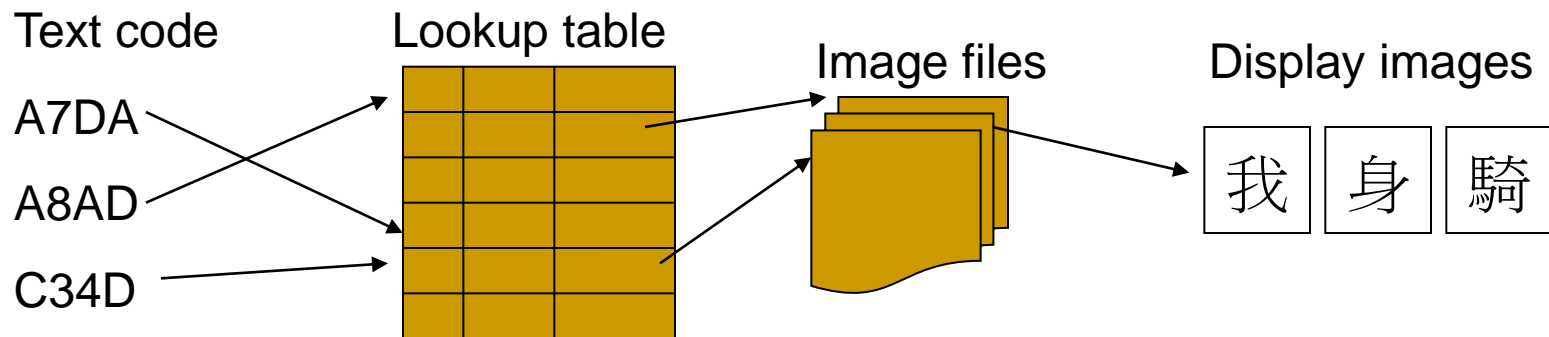
Latin char

আ	খ	দ
0986	0996	09A6
ই	গ	ধ
0987	0997	09A7

Indic char

Display Characters

- Computer doesn't show the codes directly to us. It displays what we can read



- Those images for displaying characters are called **fonts**

BCD Representation

- We can use **4 bits** to represent decimal digits 0,1,2,3,4,5,6,7,8,9
 - This is called “Binary-coded decimal” (BCD) representation
 - Example: 317=0011 0001 0111
- Problems
 - We waste last 6 bit-patterns of 4 bits
 - **Difficult to do calculation (+-*/)**

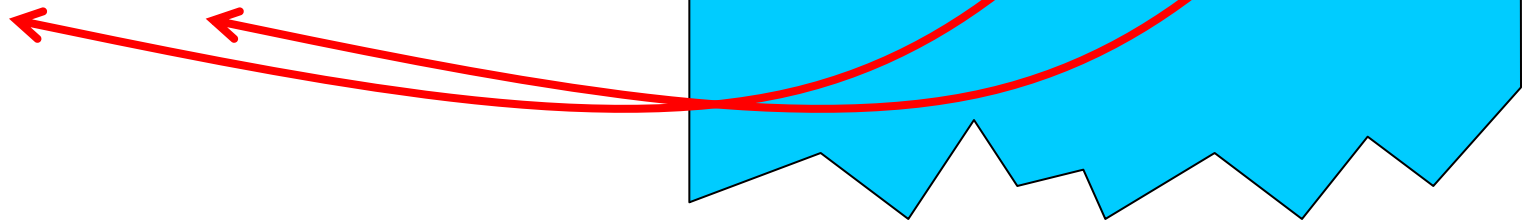
	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Example of Adding BCDs

- Using lookup table
- EX: 5+7
- In BCD:

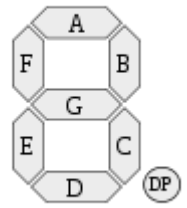
0101
+0111
0001 0010












a	b	carry	sum
:	:	:	:
0101	0110	0001	0001
0101	0111	0001	0010
0101	1000	0001	0011
0101	1001	0001	0100
:	:	:	:



Images

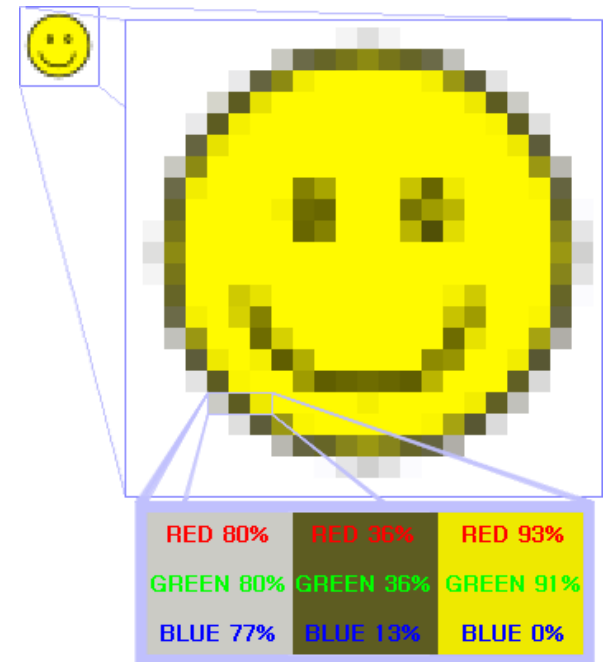
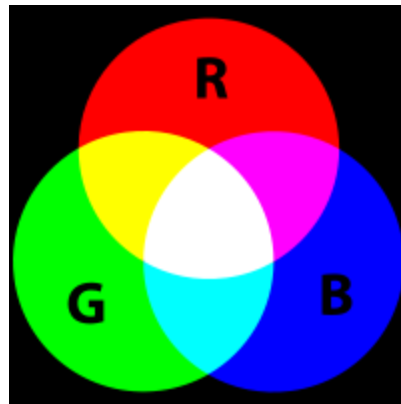
- Image representation depends on what the output device can display
 - For example, an image on the **seven segment** can be represented by 7 bits



No	Img	Repre.	3		1111001	7		1100000
0		1111110	4		0110011	8		1111111
1		0110000	5		1011011	9		1111011
2		1101101	6		1011111	A		1110111

Raster Image (bitmap)

- Represent an image by a rectangular grid of *pixels* (short for “picture element”)
- Binary image: 1 bit
- Gray image : 8 bits (0~255)
- Color image: 24 bits
- Each pixel is composed by three values: R, G, B.



Vector Graph Image

- When scaled up, a bitmap image shows the **zigzag effect**
- *Vector graph images* store the mathematical formula for lines, shapes and colors of the objects in an image
 - Example: *TrueType font (Scale-able font)*



Courier New AA A A A A

Courier AA A A A A

Sound

- Sound is an acoustic wave

- A simple wave can be characterized by *amplitude* and *frequency*.

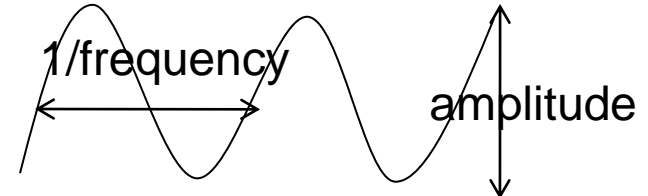
- The larger amplitude the louder the sound

- The higher frequency the higher pitch

- All sounds can be composed by simple waves.

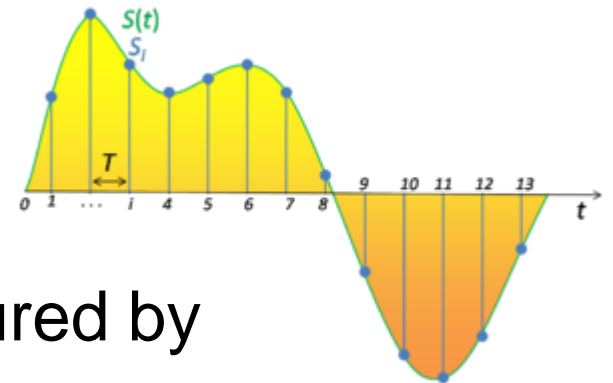
- MIDI file (Musical Instrument Digital Interface)

- Represent sounds by the amplitude and frequency of composed simple waves.



Sampled Sound

- The sound composed by simple waves may not sound real
- Alternatively, sampling the real sound and record it
- Quality of sampled sound is measured by
 - ❑ *Sampling rate*: how often to do the sampling
 - ❑ *Bit depth*: bits used for one sample
 - ❑ CD music has sampling rate 44.1kHz and uses 16 bits (stereo 32 bits) for each sample



Video

- Digital video is composed by a sequence of “continuous” images and synchronized sound tracks
 - Each image is called a “*frame*”
 - Each frame is flashed on the screen for a short time ($1/24$ seconds or $1/30$ seconds)

Outline

- 1. The Binary System: Integers, Fractions (Sec 1.5~ 1.7)
- 2. Representing Information as Bit Patterns (Sec 1.4)
- 3. Binary operations and logic gates (Sec 1.1)
 - 3.1 Basic operations for binary data
 - 3.2 Physical devices
- 4. Data Storage (Sec 1.2~1.3)
- 5. Data Processing (Sec 1.8~1.9)

The AND Function

- We can use the **AND** function to represent the statement

Room is dark A	Someone in the room B	Light is on A .AND. B
0	0	0
0	1	0
1	0	0
1	1	1

Input

Output

Boolean Operators

- The AND function is a **Boolean operator**
- Boolean operator is an operation that manipulates one or more 0/1 values
- Other common Boolean operations

OR

XOR (exclusive or)

NOT

Input		Output
0	0	0
0	1	1
1	0	1
1	1	1

Input		Output
0	0	0
0	1	1
1	0	1
1	1	0

Input	Output
0	1
1	0

(Fig. 1.2)

Logic Gate

- There are devices to implement Boolean operations
→ gate
- Pictorial representation of gates

AND



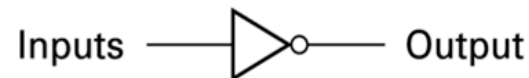
OR



XOR



NOT



(Fig. 1.2)

BIG Idea

- Computers store and process **binary #**
- Logic **true** and **false** can be used to represent binary **1** and **0**
- Logic operations can be implemented by logic **gates**
 - and in turn by **ON/OFF switches**
- Computers can be implemented using logic gates → for storing and processing

Example

- Almost **all operations** of computers can be carried out by logic gates
 - The textbook uses flip-flop as an example
 - We will use “one bit adder” as an example
- One bit adder has two inputs and two outputs (S: sum, C: carry)

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

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

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

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

A: input 1
B: input 2
S: output 1
C: output 2

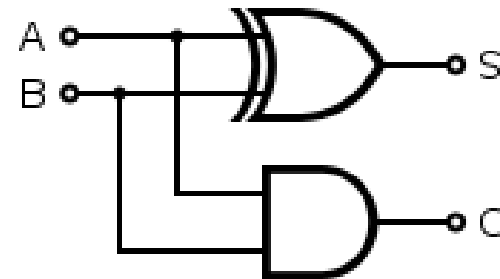
One Bit Adder

- The truth table of an one-bit adder
- Compare it to the truth table of Boolean function AND, OR, XOR, NOT

$$- S = A \text{ .XOR. } B$$

$$- C = A \text{ .AND. } B$$

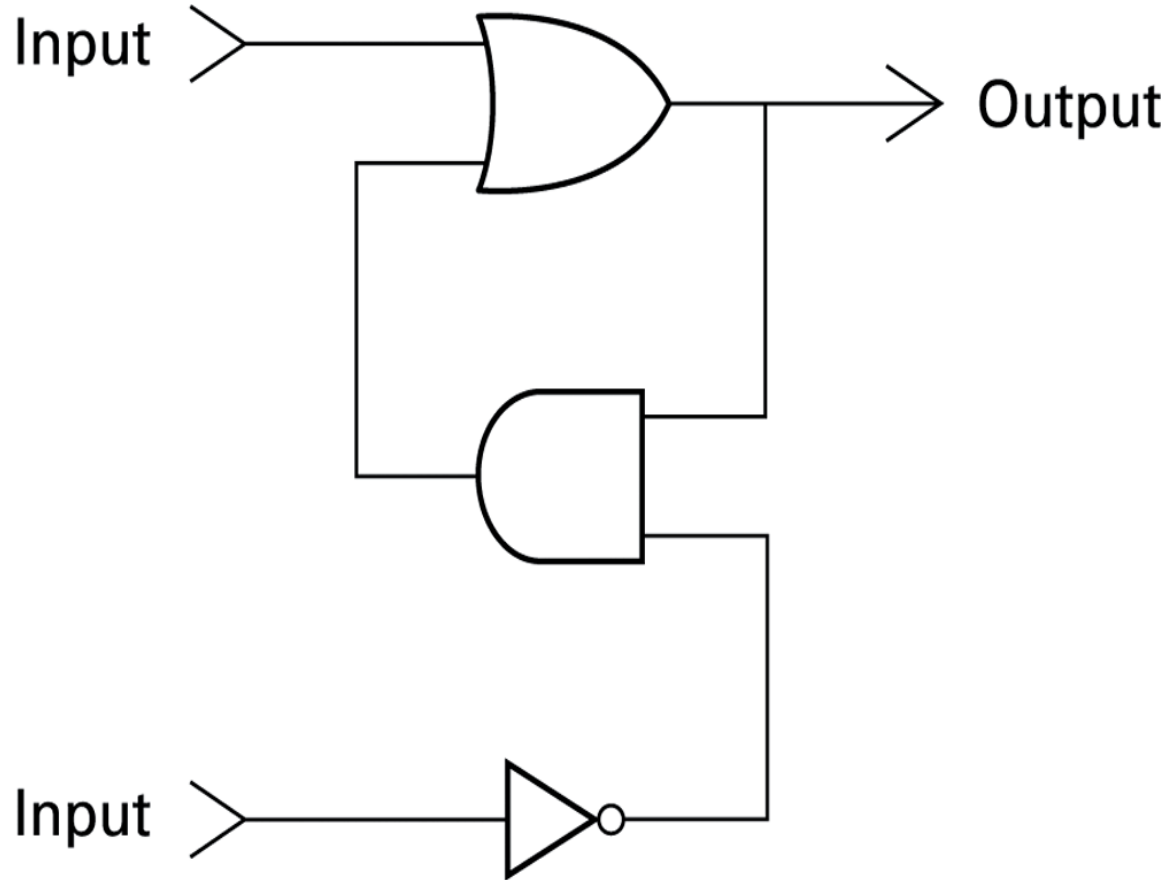
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Flip-flops

- Flip-flop: a circuit built from gates that can **store one bit**
 - One input line to set its stored value to 1
 - One input line to set its stored value to 0
 - While both input lines are 0, the most recently stored value is preserved
→ for “storing” data

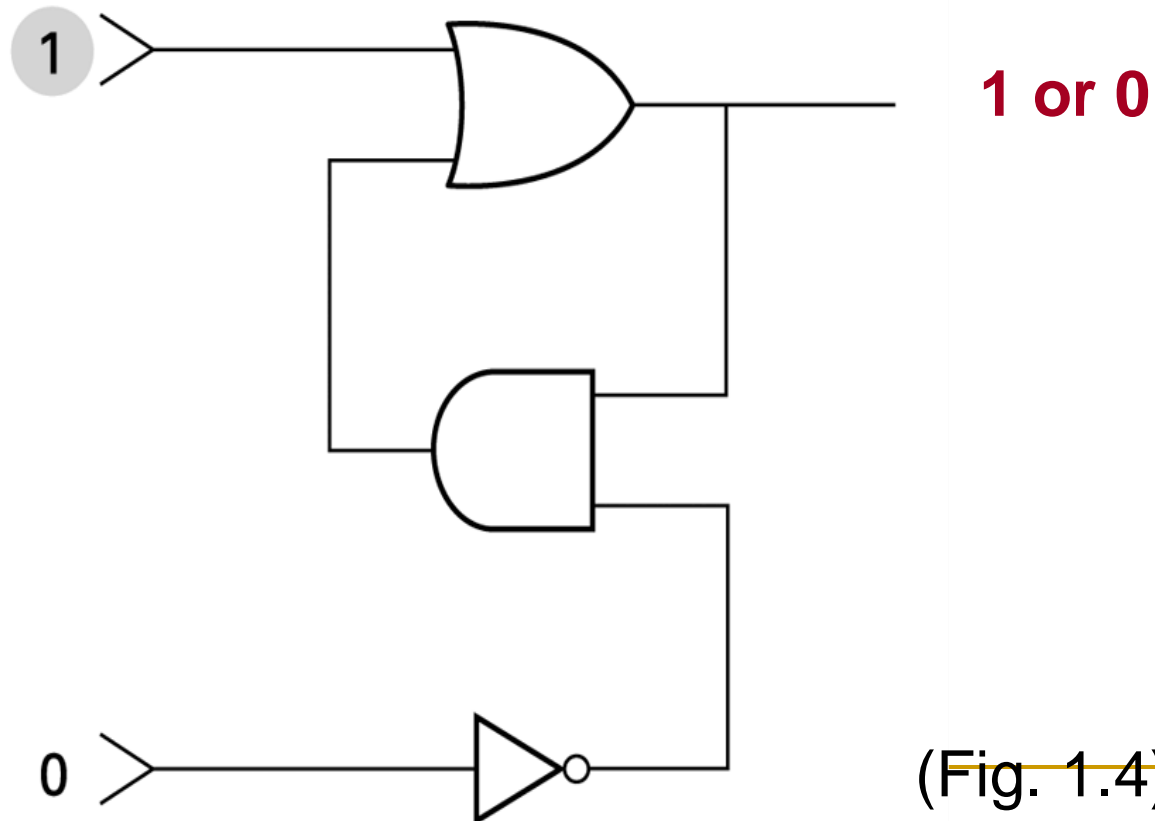
A Simple Flip-flop Circuit



(Fig. 1.3)

Setting Output to 1

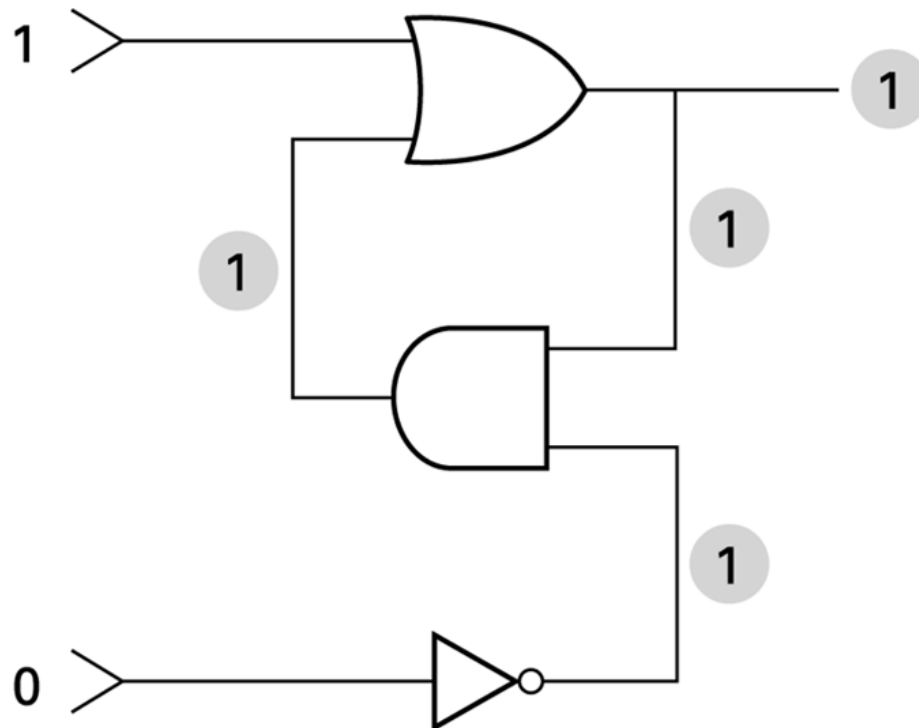
a. 1 is placed on the upper input.



(Fig. 1.4)

Setting Output to 1 (cont.)

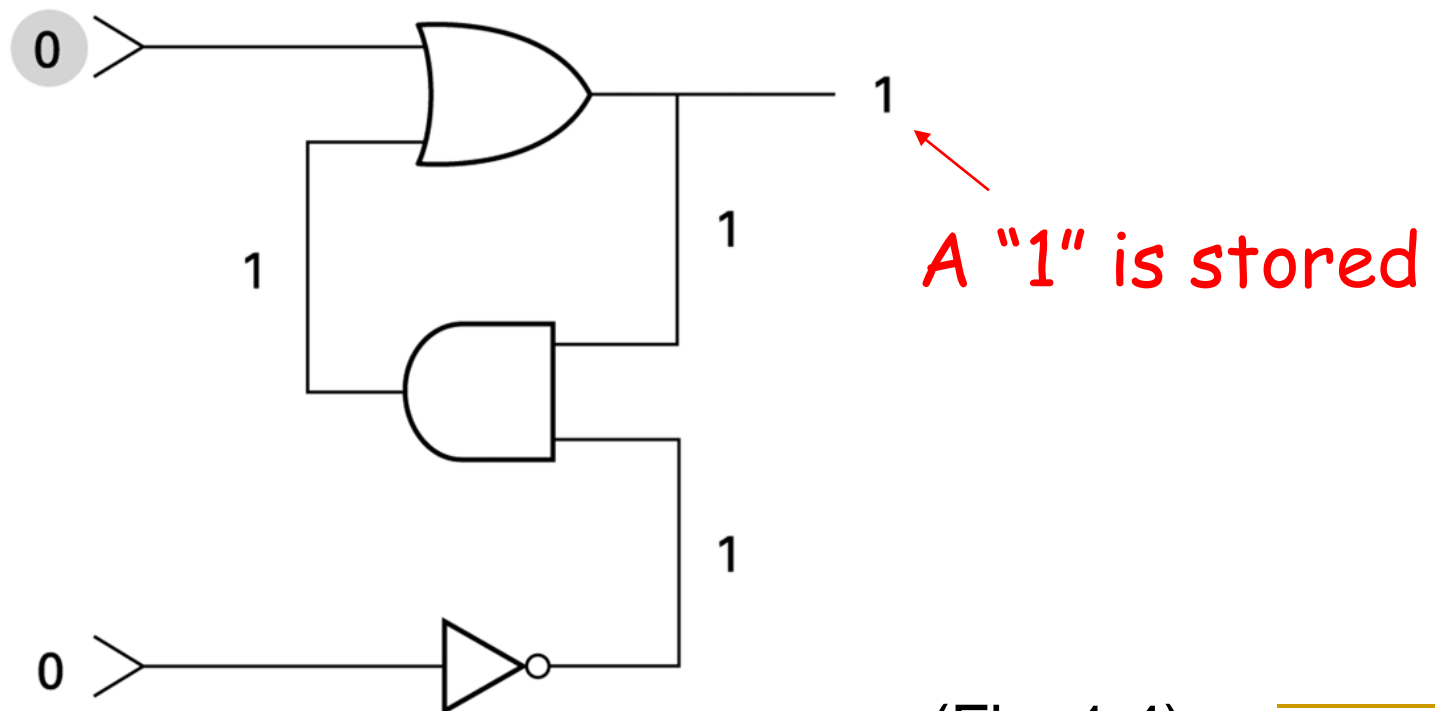
- b.** This causes the output of the OR gate to be 1 and, in turn, the output of the AND gate to be 1.



(Fig. 1.4)

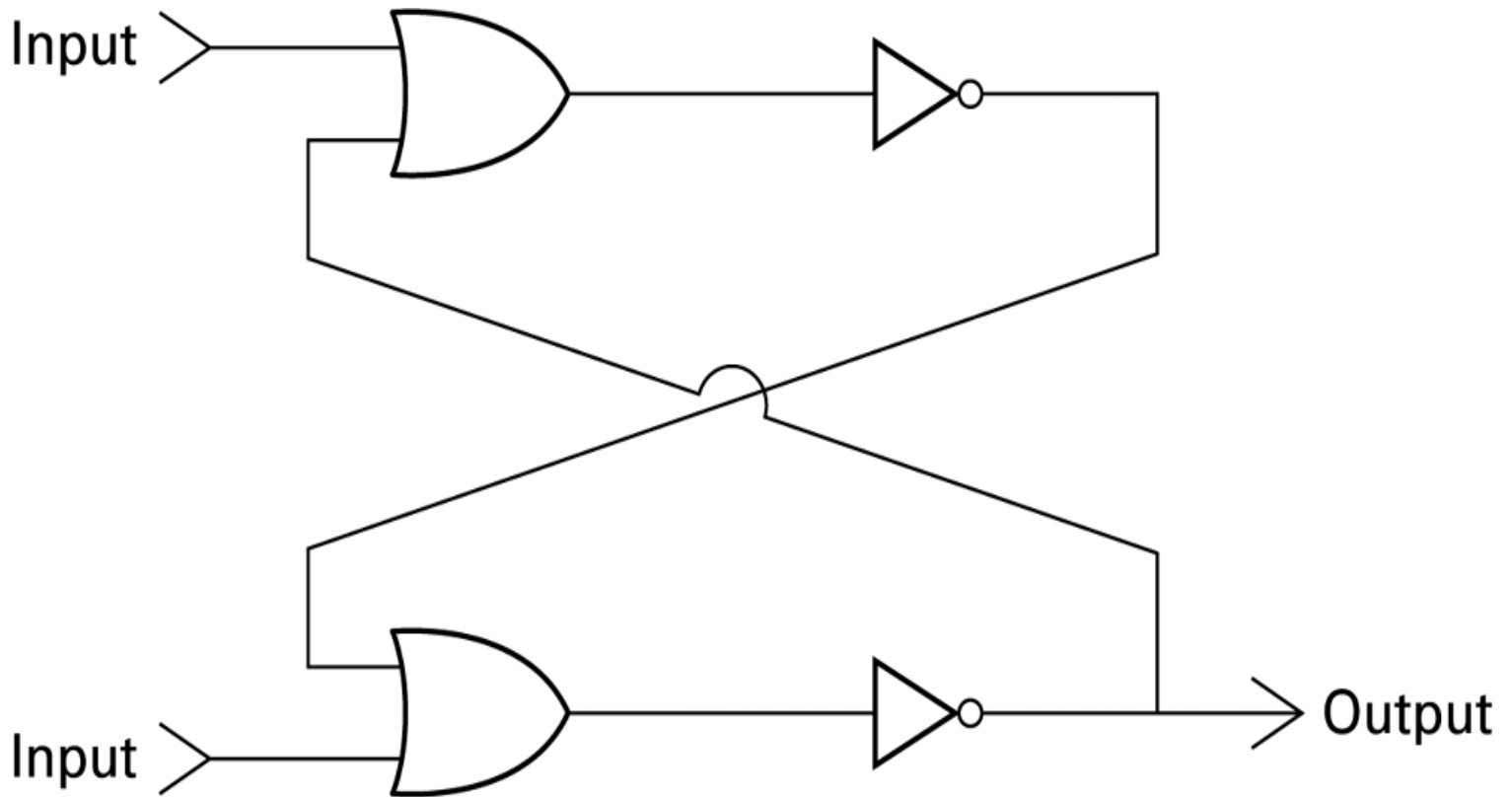
Setting Output to 1 (cont.)

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



(Fig. 1.4)

Another Way



(Fig. 1.5)

- Hierarchical structure:

Gates -> Flip-Flop (or other fundamental circuit elements) -> More complicated circuit

Outline

- 1. The Binary System: Integers, Fractions (Sec 1.5~ 1.7)
- 2. Representing Information as Bit Patterns (Sec 1.4)
- 3. Binary operations and logic gates (Sec 1.1)
- 4. Data Storage (Sec 1.2~1.3)
 - Memory, RAM, CD/DVD, hard disk, flash memory
- 5. Data Processing (Sec 1.8~1.9)

Storage Media

- Physical objects that can store bits and retrieve them can be a storage media
- *Volatile* (temporary) memory:
 - DRAM, SRAM, SDRAM
- *Non-volatile* storage (massive storage)
 - Optical systems: CD, DVD
 - Magnetic systems: hard disk, tape
 - Flash drives: iPod, cell phone, USB drivers...

Memory

- Memory is used inside computers for temporary storages
- They are often called **RAMs**
 - ❑ **R**andom **A**ccess **M**emory: data can be accessed in any order
 - ❑ Dynamic RAM (DRAM):
 - ❑ Synchronous DRAM (SDRAM)
 - ❑ Static RAM (SRAM)



Data Storage Unit

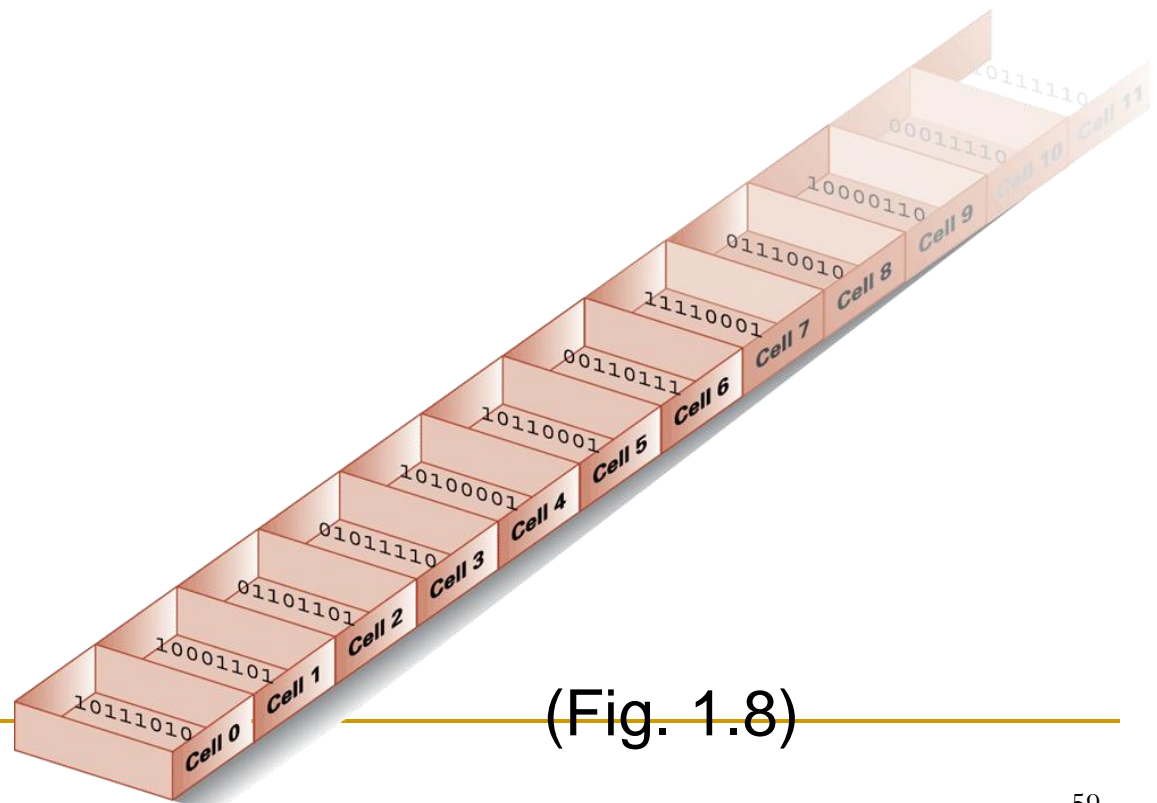
- To efficiently access data, computers use 8 bits (a **byte**) as a smallest storage unit (**cell**)
- A byte (8 bits)
 - **Most significant bit:** at the high-order end
 - **Least significant bit:** at the low-order end



(Fig. 1.7)

Memory Address

- Each storage unit in memory is numbered by an **address** so that data can be stored and loaded
 - These numbers are assigned consecutively starting at zero (定址法)

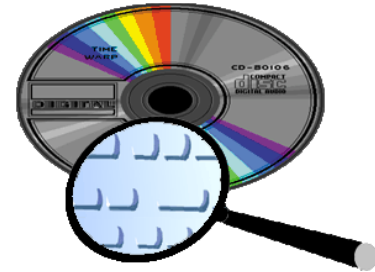


(Fig. 1.8)

Mass Storage

■ CD/DVD

- ❑ CD: Compact Disk
- ❑ DVD: Digital Video Disk



■ Hard Disks (HDD)

- ❑ A **hard platter** holds the **magnetic** medium
- ❑ Use magnetic field to represent 0/1

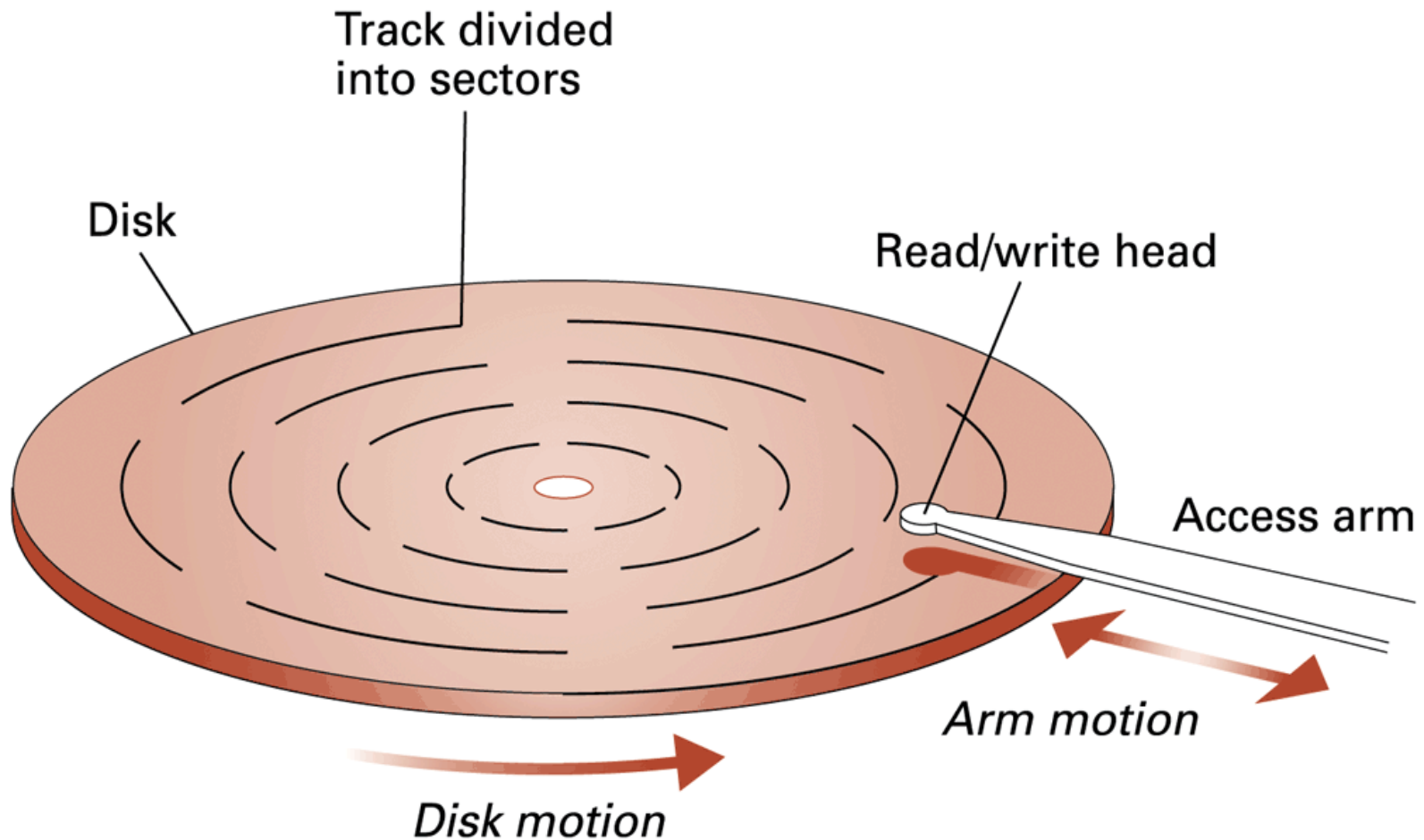


■ Flash Memory

- ❑ Use **electrical charge** to represent 0/1



Some Terms of Hard Disk



(Fig. 1.9)

HDD

- Evaluation Performance of HDD System
 - Seek Time:
 - Rotation Delay (Latency Time):
 - Access Time = Seek Time Latency Time
 - Transfer Rate

Data Communication Rates

- Measurement units
 - Bps: **Bits** per second
 - Kbps: Kilo-bps (1,000 bps)
 - Mbps: Mega-bps (1,000,000 bps)
 - Gbps: Giga-bps (1,000,000,000 bps)
- Bandwidth: maximum available rate

Outline

- 1. The Binary System: Integers, Fractions (Sec 1.5~ 1.7)
- 2. Representing Information as Bit Patterns (Sec 1.4)
- 3. Binary operations and logic gates (Sec 1.1)
- 4. Data Storage (Sec 1.2~1.3)
- 5. Data Processing (Sec 1.8~1.9)
 - 5.1 Compression (Sec 1.8)
 - 5.2 Error correction (Sec 1.9)

5.8 Data Compression

- Purpose: reduce the data size so that data can be stored and transmitted efficiently
- Example: what is the size of the video ...
 - 43 sec, 720x480, 29 frames/sec
 - $720 \times 480 \times 3 \times 29 \times 43 = 1,292,889,600$ bytes

5.8 Data Compression (Cont.)

- For example:
 - ❑ 000000000011111111 can be compressed as (10,0,9,1)
 - ❑ 123456789 can be compressed as (1,1,9)
 - ❑ AABAAAABAAC can be compressed as 1101111011100, where A, B, C are encoded as 1, 01, and 00 respectively

Many Compression Techniques

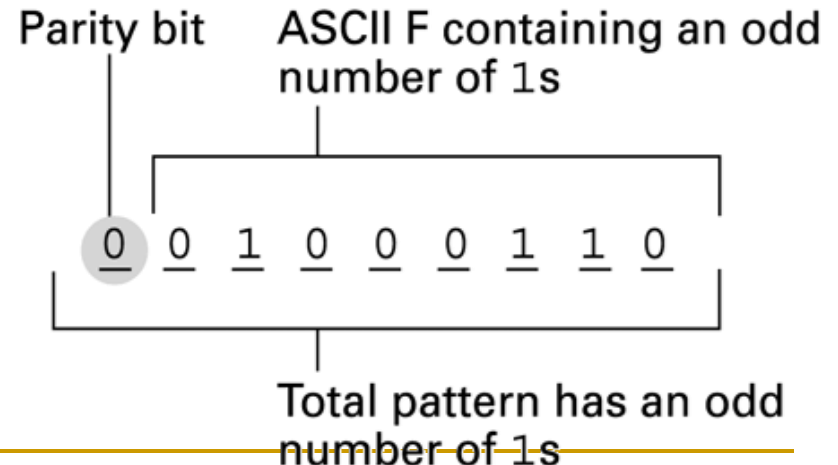
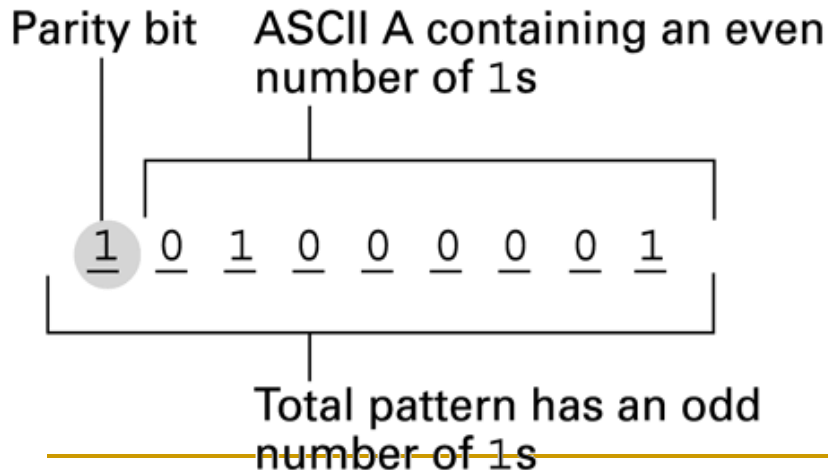
- Lossy versus lossless
- Run-length encoding
- Frequency-dependent encoding (Variable-length codes)
 - Huffman codes
- Relative encoding/differential encoding
 - Ex. Video, Mpeg
- Dictionary encoding (includes adaptive dictionary encoding such as LZW encoding)
 - Word processor for text compression
 - Word is composed of six alphabet-> 15 bits (0~24,999)
ASCII $6 \times 7 = 42$ bits , Unicode $16 \times 6 = 96$ bits

Different Data Has Different Compression Methods

- Image data
 - GIF: Good for cartoons
 - JPEG: Good for photographs
 - TIFF: Good for image archiving
- Video: MPEG
 - High definition television broadcast
 - Video conferencing
- Audio: MP3
 - Temporal masking, frequency masking

5.9 Error Detection

- During transmission, error could happen
 - For example, bit 0 \rightarrow 1 or bit 1 \rightarrow 0
- How could we know there is an error?
 - Adding a parity bit (even versus odd)



(Fig. 1.28)

Error Correction

- Can we find a way that not only detects an error, but also corrects errors?
 - ❑ Yes, by carefully designing the code
 - ❑ Suppose 010100 is received

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

(Fig. 1.30)

Smallest distance