



資工導論

Data Representation in
Computer Systems

李哲榮

What are data?

		Inside computer
• 文字 text	ABC天地玄黃	01100001011000...
• 數字 number	123 3.14159	00000001000000...
• 聲音 sound		01001100010101...
• 圖像 image		10001001010100...
• 影片 video		00110000001001...

Binary system

- Computers use a binary system, 0 and 1, to represent and store all kinds of data.
- Why binary?
 - We need to find physical objects/phenomenon to **store**, **transmit**, and **process** data. Binary is the most straightforward representation.

有 無 上 下 黑 白 真 偽 勝 負
北 南 負 正 錯 對 陽 陰 關 開 1 0

Some jargons

- **Bit:** a binary digit (0 or 1)
- **Byte:** 8 bits
 - Basic storage unit in computer system
- **Hexadecimal notation:**
 - Represents each 4 bits by a single symbol
 - Example: A3 denotes
1010 0011

Bit pattern	Hexadecimal
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

More jargons

- **Kilobyte (KB):** 2^{10} bytes = 1024 bytes $\approx 10^3$ bytes
 - Example: 3 **KB** $\approx 3 \times 10^3$ bytes
- **Megabyte (MB):** 2^{20} bytes $\approx 10^6$ bytes
 - Example: 3 **MB** $\approx 3 \times 10^6$ bytes
- **Gigabyte (GB):** 2^{30} bytes $\approx 10^9$ bytes
 - Example: 3 **GB** $\approx 3 \times 10^9$ bytes
- **Terabyte (TB):** 2^{40} bytes $\approx 10^{12}$ bytes
 - Example: 3 **TB** $\approx 3 \times 10^{12}$ bytes
- **Petabyte (PB):** 2^{50} bytes $\approx 10^{15}$ bytes
 - Example: 3 **PB** $\approx 3 \times 10^{15}$ bytes

Text data

- Each character is assigned to 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[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

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

- Quiz: how many different bit patterns can be represented by 7 bits?

Big5 code

- For Chinese character encoding
- Uses 16 bits to represent a Chinese character
- Example

我	身	騎	白	馬
A7DA	A8AD	C34D	A5D5	B0A8

1010 0111 1101 1010



Unicode

- Uses 16-bits to represent the major symbols used in languages worldwide

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

Arabic char

偲	像	儻
3460	3470	3480
御	倭	儻
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

- Quiz: how many different bit patterns can be represented by 16 bits?

Encoding

- You need correct encoding to read meaningful texts.
- For example, on browser, you can choose the encoding.
- Most webpages have specified their encoding.

For HTML4:

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

For HTML5:

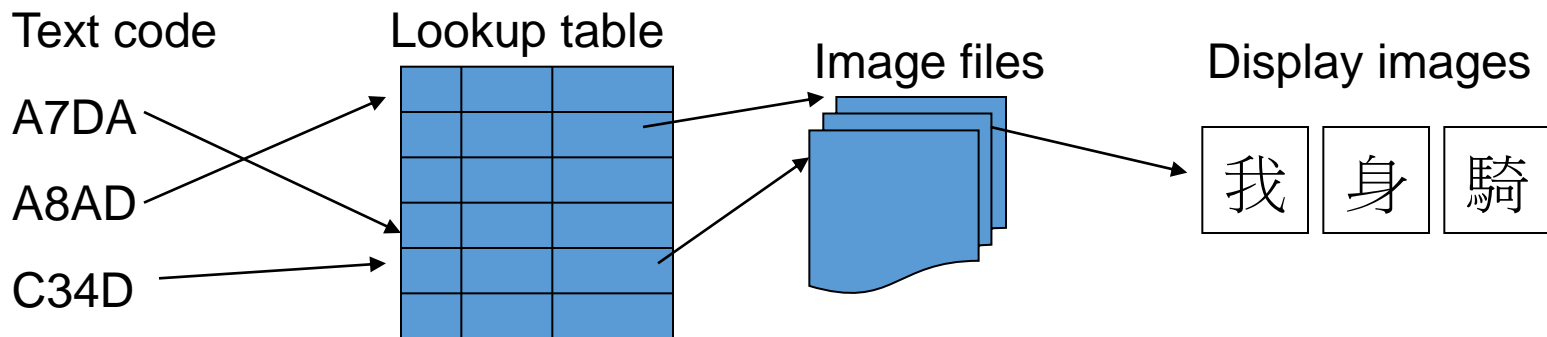
```
<meta charset="UTF-8">
```

æ 1æ “šè•é,|æ³•é™çæ–
‡ä»¶é;̄çº°i¼ŒĀ°çfē‡ē‡ç] šä¼4æ¥–
å@ ¶ã€ä,-ç•Œçÿ¥åæ%o<æ©ÿåºşå»
ã€Œå®é” é»»ã€(HTC)å...-å,è ‘fä°
<é•çkē>ªç´ ...i¼ŒĒé•
å^° æ•™æœfä°åf«æ...^å–



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**.
 - We will talk about images later.

Numbers

- Like texts, 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)
 - EX: 123 is coded 0001 0010 0011
- Problems
 - 4 bits binary can have 16 different patterns, but BCD only uses 10. It wastes 6 bit-patterns.
 - 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:

$$\begin{array}{r} 0101 \\ +0111 \\ \hline 0001\ 0010 \end{array}$$

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

Binary numeral system

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

Base 10 system



$$3 \times 10^2 + 7 \times 10^1 + 5 \times 10^0$$

Base 2 system

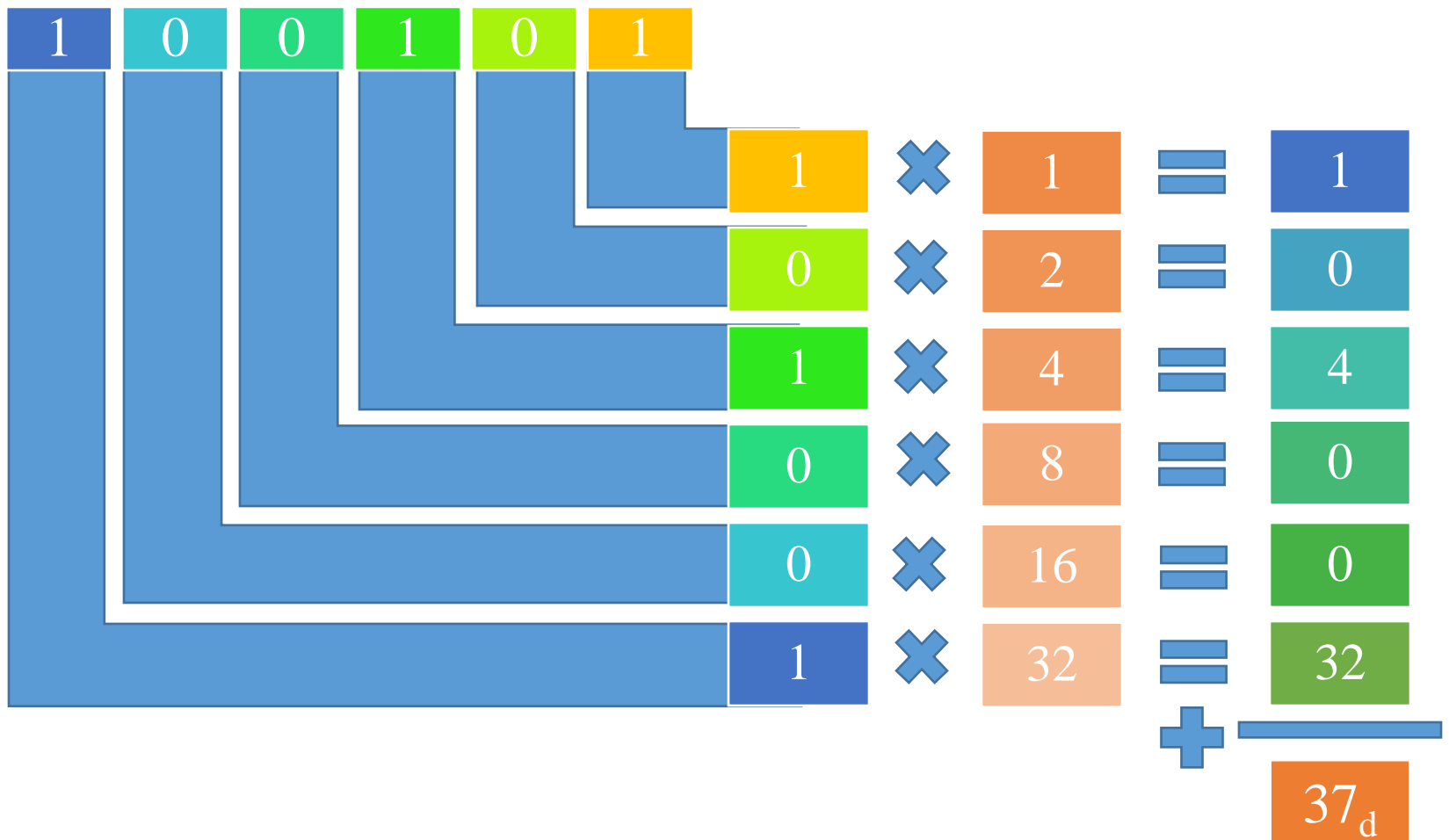


$$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

- 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 ?



Decimal to binary

- What is the binary number representation of 13_d ?
- First, how many bits do we need for 13.
 - Since $2^3=8<13<16=2^4$, we need at least 4 bits.

$$\begin{aligned} 13 &= b_3 b_2 b_1 b_0 \\ &= b_3 \times 8 + b_2 \times 4 + b_1 \times 2 + b_0 \times 1 \end{aligned}$$

- Remember b_3, b_2, b_1, b_0 are either 0 or 1.
- Can we decide what b_0 is? 0 or 1.
 - Because $b_3 \times 8 + b_2 \times 4 + b_1 \times 2$ is an even number, and 13 is an odd number, b_0 must be 1.

Decimal to binary

- What is b_1 ?

$$\begin{array}{r} 13 = b_3 \times 8 + b_2 \times 4 + b_1 \times 2 + b_0 \times 1 \\ - b_0 = b_0 \end{array}$$

$$\begin{array}{r} 12 = b_3 \times 8 + b_2 \times 4 + b_1 \times 2 \\ \div 2 = \div 2 \end{array}$$

$$6 = b_3 \times 4 + b_2 \times 2 + b_1 \times 1$$

- It becomes to decide the binary representation of 6.
 - Since 6 is an even number, b_1 must be 0.

Decimal to binary

- What is b_2 ?
- We can follow the same procedure.

$$\begin{array}{l} \boxed{6} = \boxed{b_3} \times 4 + \boxed{b_2} \times 2 + \boxed{b_1} \times 1 \\ \boxed{b_1} = \boxed{b_1} \end{array}$$

$$\begin{array}{l} \boxed{6} = \boxed{b_3} \times 4 + \boxed{b_2} \times 2 \\ \boxed{6} \div 2 = \boxed{3} \end{array}$$

$$\boxed{3} = \boxed{b_3} \times 2 + \boxed{b_2} \times 1$$

- Since 3 is an odd number, b_2 must be 1.

Decimal to binary

- What is b_3 ?

$$\begin{array}{r} \begin{array}{c} \text{3} \\ \text{---} \\ \text{b}_2 \end{array} = \begin{array}{c} \text{b}_3 \times 2 \\ \text{---} \\ \text{b}_2 \end{array} + \begin{array}{c} \text{b}_2 \times 1 \\ \text{---} \\ \text{b}_2 \end{array} \\ \hline \begin{array}{c} \text{2} \\ \text{---} \\ \text{2} \end{array} = \begin{array}{c} \text{b}_3 \times 2 \\ \text{---} \\ \text{2} \end{array} \\ \hline \begin{array}{c} \text{1} \\ \text{---} \\ \text{b}_3 \times 1 \end{array} \end{array}$$

- It shows b_3 is 1.
- So $13_d = b_3b_2b_1b_0 = 1101_b$.

Algorithm for decimal to binary

- An algorithm is a procedure to describe the steps for computers to solve some problems.
- Let x be the decimal number in question and y is the corresponding binary number representation.
- What we did can be described as the following steps.

Step 1	If x equals to 0 or 1, $y = x \circ y$.
Step 2	If x is even, b is 0. Otherwise b is 1. Let $y = b \circ y$.
Step 3	Let $x = (x - b)/2$ and go to step 1.

- The initial value of y is an empty bit pattern.
- The \circ operator means concatenation.

Algorithm for decimal to binary

- Problem: how a computer knows a number is even or odd?
- Use “integer division”
 - $x \mathbf{div} 2 = q \times 2 + r$, where q is quotient and r is remainder.
- The previous algorithm can be modified as follows

Step 1	If x equals to 0 or 1, $y = x \circ y$.
---------------	--

Step 2	Let $x \mathbf{div} 2 = q \times 2 + r$ and $y = r \circ y$.
---------------	---

Step 3	Let $x = q$ and go to step 1.
---------------	-------------------------------

Example

$$\begin{array}{r} 6 \\ 2 \overline{)13} \end{array}$$

$$\begin{array}{r} 3 \\ 2 \overline{)6} \end{array}$$

$$\begin{array}{r} 1 \\ 2 \overline{)3} \end{array}$$

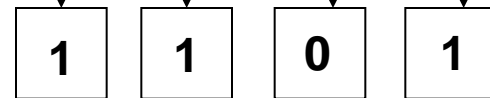
$$\begin{array}{r} 0 \\ 2 \overline{)1} \end{array}$$

Remainder = 1

Remainder = 0

Remainder = 1

Remainder = 1



Binary Presentation

Exercise: what is the binary for 57_d ?

Binary number calculations

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

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

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

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

Diagram illustrating the binary addition of 5 (0101) and 9 (1001) to get 14 (1110). Arrows point from the binary result to the decimal value 14. The numbers 5 and 9 are also shown with arrows pointing to their respective binary representations.

Another example

- Remember the rules of one bit addition

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

- What is $00111010_b + 00011011_b$?

$$\begin{array}{r} 1111 \\ + 00111010 \\ \hline 01010101 \end{array}$$

Negative numbers

- How to represent -1, -2, ... on a computer?
- Solution 1: use an extra bit to represent the negatives 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
- Problem 1: It will have 2 zeros, +0 and -0.
- Problem 2: how can we to do the addition $(-1) + (4)$ efficiently using the signed notation?

Example of $-1+4$ with sign notation

- One bit subtraction

$$\begin{array}{r}
 0 \quad 1 \quad 1 \quad 0 \\
 -0 \quad -0 \quad -1 \quad -1 \\
 \hline
 0 \quad 1 \quad 0 \quad -1 \quad 1 = -1*2+1*1=-1
 \end{array}$$

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

It can be very complicated !!

Solution 2

- The negative sign “-” just means the “opposite” or the “inverse”.
 - Ex, the opposite of East is West.
- For addition, the inverse of a number d , denoted $I(d)$, has the property: $I(d)+d=0$.
- We can use this property to define negative numbers.
 - For example, let x be the representation of -1 . It should satisfy the property $x + 1 = 0$.

What is -1?

- 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

$$\begin{array}{rcccc} 1 & 1 & 1 & 1 & \\ & b_3 & b_2 & b_1 & b_0 \\ + & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 \end{array}$$

- b_0 must be 1, and it will create a carry 1.
- b_1 must be 1, and it will create a carry 1.
- b_2 must be 1, and it will create a carry 1.
- b_3 must be 1, and it will create a carry 1.

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

Finite digit systems

- One common example of finite digit system is clocks.
- What is the next second after 23:59:59?

23:59:59



Two's complement

- Such number representation for negative numbers is called two's complement.
- The right table lists 4 bits 2's complement representation.
- Only one zero.
- Easy for addition.
- The leading bit of negative numbers is 1. (signed bit)

Bit pattern	Values
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

Calculation with 2's complement

- Calculation can be made easily 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

Computing 2's complement

- A simple algorithm to compute the 2's complement of a number
 1. Change each bit 0 to 1 and bit 1 to 0
 2. Add 1.

$$\begin{array}{r} 6_d = 0110_b \\ \downarrow \\ 1001_b \end{array} \quad \begin{array}{r} 1001_b \\ + 0001_b \\ \hline 1010_b = -6_d \end{array} \quad \begin{array}{r} 0110_b \quad (6) \\ + 1010_b \quad (-6) \\ \hline 1\ 0000_b \quad (0) \end{array}$$

truncated

Exercises

- What are the decimal numbers for the following 2's complement representations?

(a) 00000001

(b) 01010101

(c) 11111001

(d) 10101010

(e) 10000000

(f) 00110011

- Find the negative value represented in 2's complement for each number

Overflow

- What is $5+4$?

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

- This is called **overflow**
 - Adding two positive numbers results a negative number; or adding two negative numbers results a positive number.
 - A 4 bits 2's complement system can only represent $-8 \sim 7$

Bit pattern	Values
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

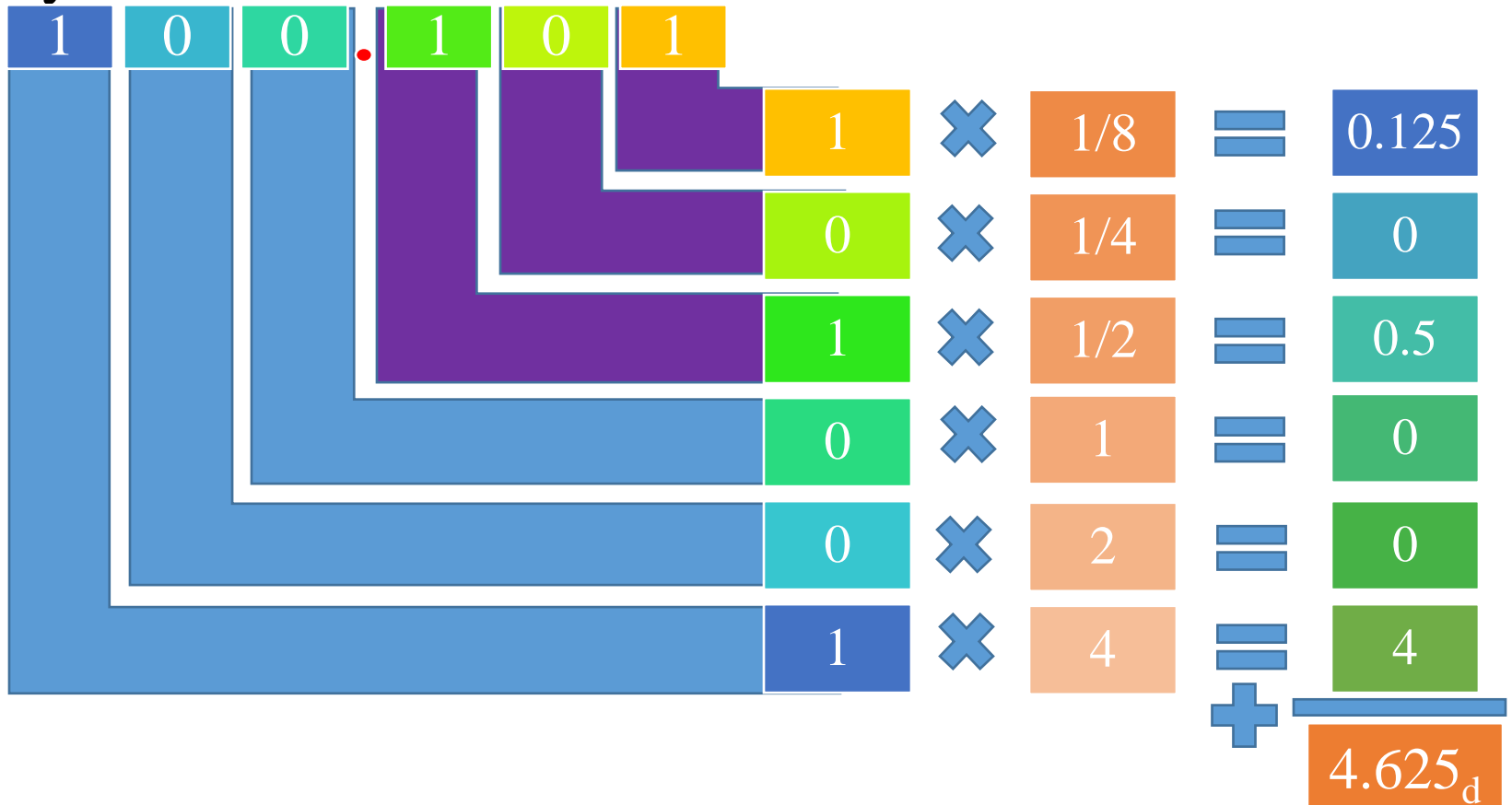
Data types

- How do we know 1001_b is 9 or -7?
- In computer system, every datum needs a data type, which describes the format of the data.
- For example, in C, there are many data types for integers.

Data types	Size	Range
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int	4 bytes	-2147483648 to 2147483647
unsigned int	4 bytes	0 to 4294967295

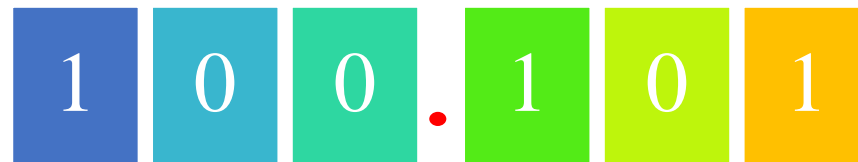
Numbers with the decimal point

- Real numbers can be represented by the binary system too.



Fixed point representation

- This representation is called fixed point, because the decimal point is at the fixed position.



- Fixed point representation is good for calculations, but the representable range is very limited.
 - Think about 1000000 or 0.0001. They will be all 0 for 6 bit fixed point representation.

Floating point

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

- It uses the scientific notation of numbers.

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

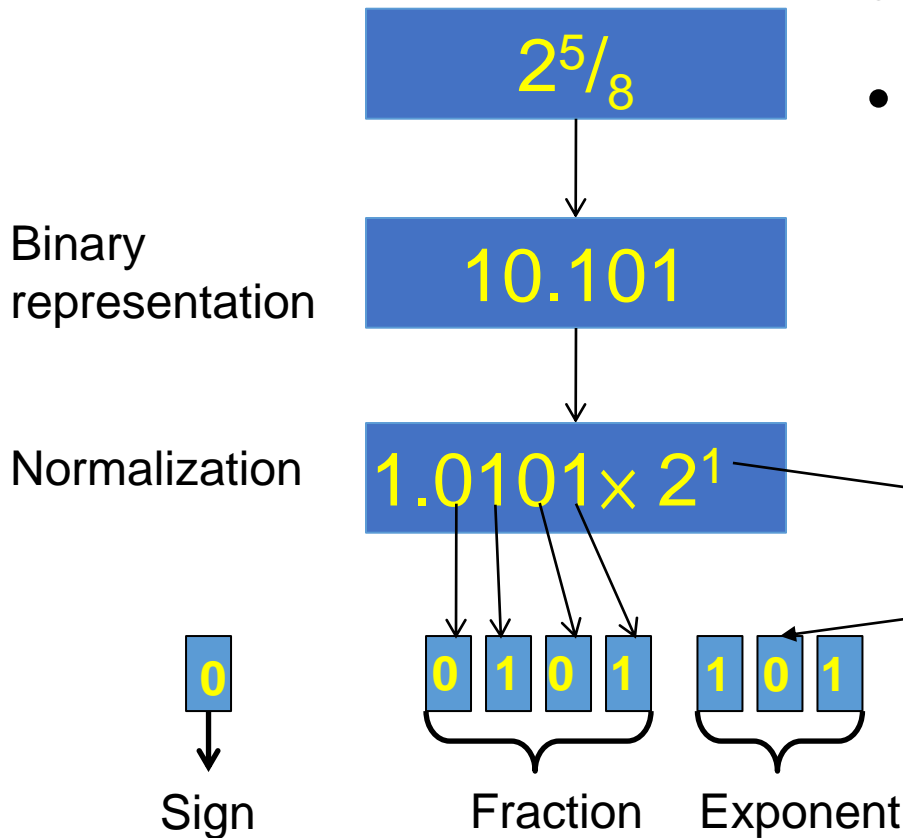
$$101.101_b = +1.01101_b \times 2^{2d} = \begin{matrix} + \\ \downarrow \end{matrix} 1.01101_b \times 2^{\begin{matrix} 10 \\ \downarrow \end{matrix}}_b.$$

- This is called the **floating point** representation of real numbers.

sign fraction exponent

Ex: floating point of $2^5/8$

- Sign bit: 0 for + and 1 for -.
- Exponent uses excess representation.



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

Why excess notation?

- Why excess notation?
- Why moving the exponent in front of the fraction?
- Both are for the same reason: easy to compare the numbers.
 - For two normalized numbers x and y , if x 's exponent is larger than y 's exponent, x is larger than y .
 - We can use the circuit of comparing integers to compare floating point numbers

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

Truncation error

- Consider the decimal number 0.1_d .
 - $0.1 = 1/16 + 1/32 + 1/256 + 1/512 + \dots$
 - So the binary representation of 0.1 is 0.000110011...
 - It is an infinite sequence in the binary notation.
- After normalization, it is $+1.10011\dots \times 2^{-4}$. The 8 bit floating point format can only represent $+1.1001 \times 2^{-4} = 25/256 = 0.09765625$
- The difference $|0.1 - 0.09765625| = 0.00234375$ is called **truncation error**.

IEEE Standard 754

- IEEE (Institute of Electrical and Electronics Engineers) has made a floating point number standard, IEEE-754, so that different computers can have a common ground.
- Two commonly used formats of floating numbers



William Morton Kahan

	Sign	Exponent	Fraction	Total
Single	1 bit	8 bits	23 bits	32 bits
Double	1 bit	11 bits	52 bits	64 bits

Exercises

- What are the fractions for the following floating number representations?
 - Suppose 1 bit for sign, 3 bits for exponent (using excess notation), 4 bits for mantissa
- (a) 01001010 (b) 01101101 (c) 11011100 (d) 10101011
- If direct truncation is used, what are the ranges of their possible values?