第二讲：二进制编码表示

主　要　内　容

- 计算机的外部信息和内部数据
- 进位计数制
  - 十进制
  - 二进制
  - 八进制和十六进制
- 二进制数与其他计数制数之间的转换
  - R进制数与十进制数之间的转换
  - 二、十六进制数之间的转换
  - 十进制数→二进制数的简便方法

顺便再来看看（ISA）

计算机是如何工作的呢?

“存储程序”工作方式——程序（指令）、数据、执行

(计算机硬件能够理解并执行的只有机器指令)

(计算机硬件能够存储处理的只有二进制数据)

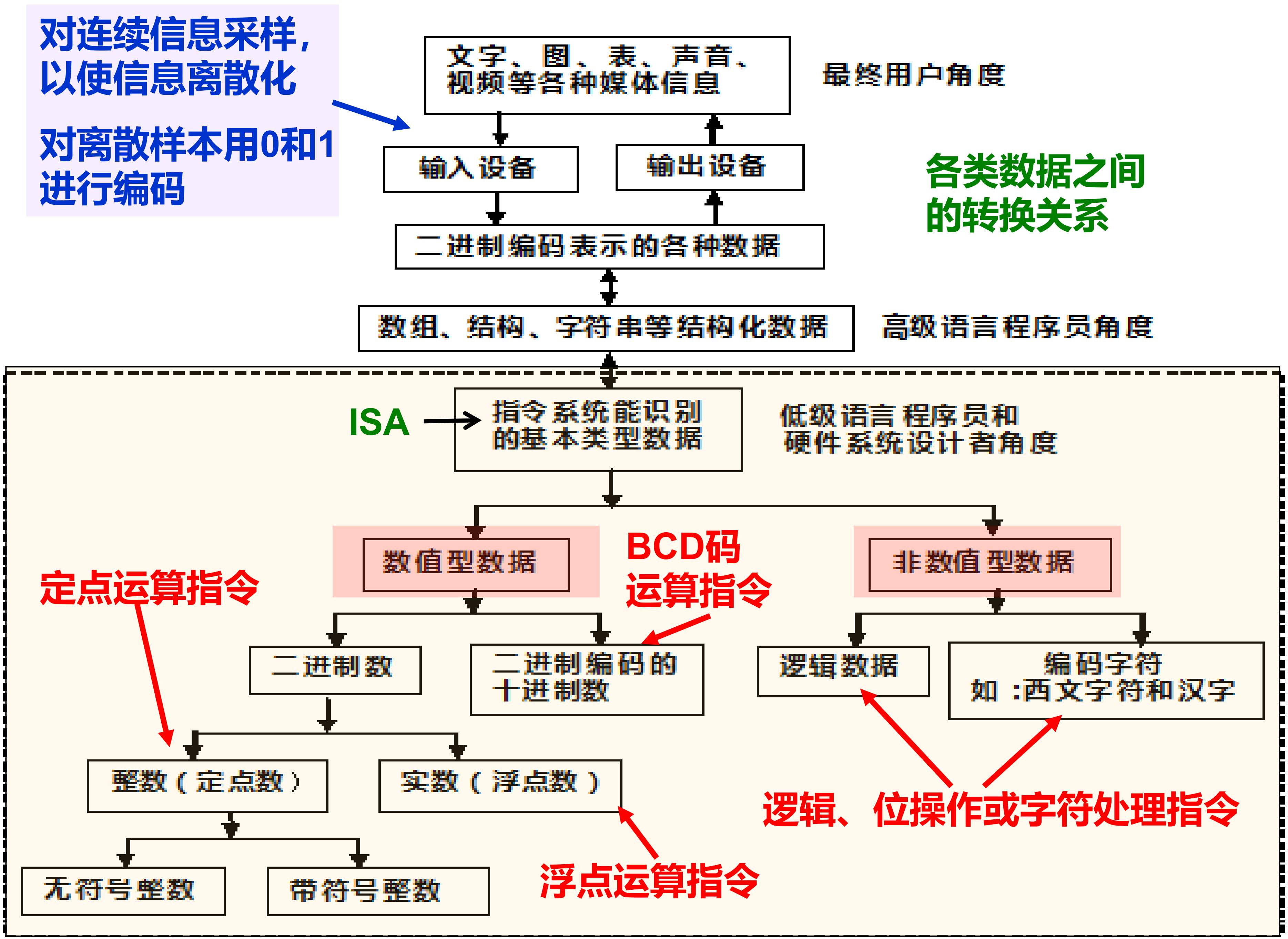
人脑接受一切信息，理解并处理

做菜：接受固定原材料，处理

◆ **ISA即指令集体系结构规定了如何使用硬件**

- 可执行的指令的集合；
- 指令可以接受的操作数的类型；
- 操作数能存放到哪些地方：寄存器，MM；
- 指令执行过程的控制方式，包括程序计数器等。

如果外部世界的一切信息都要用计算机来处理？



对连续信息采样，以使信息离散化
对离散样本用0和1进行编码
文字、图、表、声音、视频等各种媒体信息
最终用户角度
输入设备
输出设备
各类数据之间的转换关系
二进制编码表示的各种数据
数组、结构、字符串等结构化数据
高级语言程序员角度
ISA
指令系统能识别的基本类型数据
低级语言程序员和硬件系统设计者角度
数值型数据
BCD码运算指令
非数值型数据
定点运算指令
二进制数
二进制编码的十进制数
逻辑数据
编码字符 如：西文字符和汉字
整数（定点数）
实数（浮点数）
逻辑、位操作或字符处理指令
无符号整数
带符号整数
浮点运算指令

信息的二进制编码

- 机器级数据分两大类：
  - 数值数据：无符号整数、带符号整数、浮点数（实数）、十进制数
  - 非数值数据：逻辑数（包括位串）、西文字符和汉字
- 计算机内部所有信息都用二进制（即：0和1）进行编码
- 用二进制编码的原因：
  - 制造二个稳定态的物理器件容易
  - 二进制编码、计数、运算规则简单
  - 正好与逻辑命题对应，便于逻辑运算，并可方便地用逻辑电路实现算术运算
- 真值和机器数
  - 机器数：用0和1编码的计算机内部的0/1序列
  - 真值：机器数真正的值，即：现实中带正负号的数

Decimal / Binary（十 / 二进制数）

- The decimal number 5836.47 in powers of 10:

$$5 \times 10^{3} + 8 \times 10^{2} + 3 \times 10^{1} + 6 \times 10^{0} + 4 \times 10^{-1} + 7 \times 10^{-2}$$

- The binary number 11001 in powers of 2 :

$$1 \times \boxed{2^{4}} + 1 \times \boxed{2^{3}} + 0 \times \boxed{2^{2}} + 0 \times \boxed{2^{1}} + 1 \times \boxed{2^{0}}$$

$$= 16 + 8 + 0 + 0 + 1 = 25$$

- 用一个下标表示数的基（ radix / base)

或用后缀B-二进制（H-十六进制（前缀0x-）、O-八进制）

$11001_2 = 25_{10}$、11001B= 25

Octal / Hexadecimal (八 / 十六进制数)

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	1	1	0	1	0	0	0	0

$= 2000_{10}$

$$v = \sum_{i=0}^{n-1} 2^i b_i$$

$2^3=8$

03720

Octal - base 8

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

$2^4=16$

0x7d0

Hexadecimal - base 16

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - a
0011 - 3	1011 - b
0100 - 4	1100 - c
0101 - 5	1101 - d
0110 - 6	1110 - e
0111 - 7	1111 - f

一个8进制数字用3位二进制数字表示

一个16进制数字用4位二进制数字表示

计算机用二进制表示所有信息！

为什么要引入 8 / 16进制?

8 / 16进制是二进制的简便表示。便于阅读和书写！

它们之间对应简单，转换容易。

在机器内部用二进制，在屏幕或其他外部设备上表示时，转换为10进制或8/16进制数，可缩短长度

早期有用8进制数简便表示2进制数

现在基本上都用16进制数表示机器数

数制之间的转换

(1) 二、八、十六进制数的相互转换

① 八进制数转换成二进制数

$(13.724)_8=(\ 001\ 011\ .\ 111\ \ 010\ \ 100\ )_2=(1011.1110101)_2$

② 十六进制数转换成二进制数

$(2B.5E)_{16} = (00101011\ .\ 01011110)_2 = (101011.0101111)_2$

③ 二进制数转换成八进制数

$(0.10101)_2 = (\ 000\ .\ 101\ \ 010\ )_2 = (\ 0.52)_8$

④ 二进制数转换成十六进制数

$(11001.11)_2 = (\ 0001\ \ 1001\ .\ 1100\ )_2 = (\ 19.C\ )_{16}$

数制之间的转换

(2) R进制数 => 十进制数

按“权”展开 (a power of R)

例1: $(10101.01)_2=1\times2^4+1\times2^2+1\times2^0+1\times2^{-2}=(21.25)_{10}$

例2: $(307.6)_8=3\times8^2+7\times8^0+6\times8^{-1}=(199.75)_{10}$

例1: $(3A.1)_{16}=3\times16^1+10\times16^0+1\times16^{-1}=(58.0625)_{10}$

(3) 十进制数 => R进制数

整数部分和小数部分分别转换

① 整数(integral part)----“除基取余，上右下左”

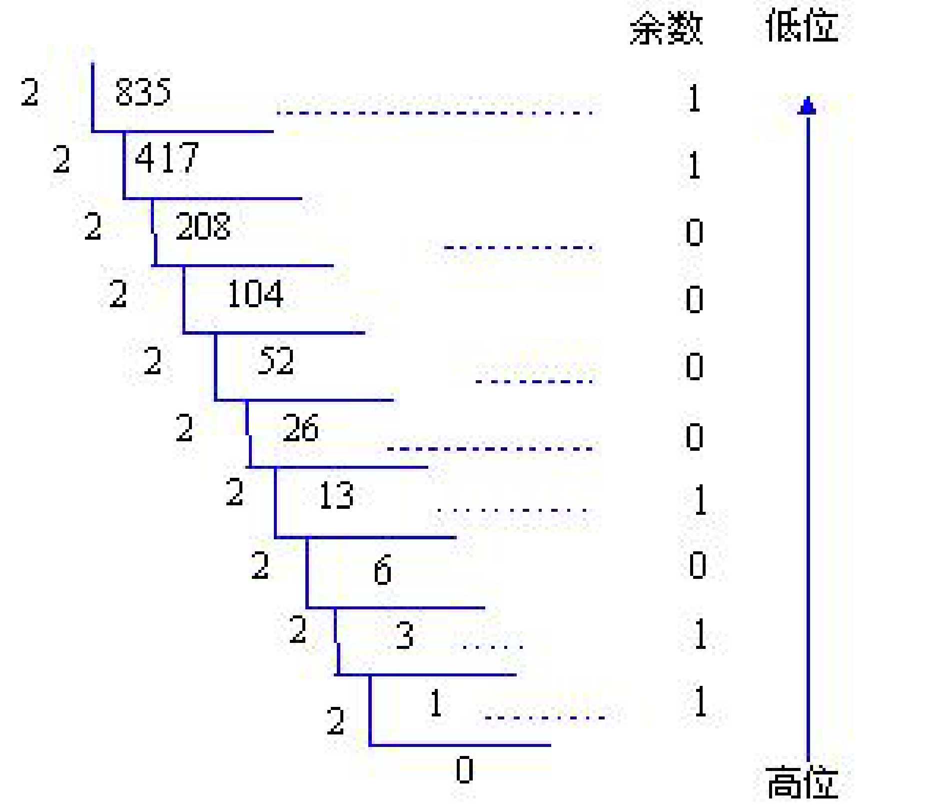
② 小数(fractional part)----“乘基取整，上左下右”

理论上的做法

10进制→2进制

例1: $(835.6785)_{10}=(1101000011.1011)_2$

整数----“除基取余，上右下左”　　小数----“乘基取整，上左下右”



$0.6875\times 2=1.375$	整数部分=1	(高位)
$0.375\times 2=0.75$	整数部分=0	↓
$0.75\times 2=1.5$	整数部分=1	↓
$0.5\times 2=1.0$	整数部分=1	(低位)

10进制→8进制

例2: $(835.63)_{10}=(1503.50243\cdots)_8$

小数----“乘基取整，上左下右”

有可能乘积的小数部分总得不到0，此时得到一个近似值。

除数	被除数	余数	
			低位
8	835		
8	104	3	↑
8	13	0	
8	1	5	
	0	1	高位

$0.63\times8=5.04$	整数部分=5	(高位)
$0.04\times8=0.32$	整数部分=0	
$0.32\times8=2.56$	整数部分=2	
$0.56\times8=4.48$	整数部分=4	
$0.48\times8=3.84$	整数部分=3	(低位)

数制之间的转换（简便方法）

(3) 十进制数 => R进制数

实际按简便方法先转换为二进制数，再按需转换为8/16进制数

整数：2、4、8、16、32、64、128、256、512、1024、2048、4096、8192、16384、32768、65536

小数：0.5、0.25、0.125、0.0625、0.03125、……

例： $4123.25 = 4096+16+8+2+1+0.25$

$= 1\ 0000\ 0001\ 1011.01B$

$= (101B.4)_{16}$

$4023 = (4096-1)-64-8$

$= 1111\ 1111\ 1111B - 100\ 0000B - 1000B$

$= 1111\ 1011\ 0111B$

$= FB7H = (FB7)_{16}$

第三讲：数值数据的编码表示

主　要　内　容

- 定点数的表示
  - 定点数的二进制编码
    - 原码、补码、移码表示
  - 定点整数的表示
    - 无符号整数、带符号整数
- 浮点数的表示
  - 浮点数格式和表示范围
  - IEEE754浮点数标准
    - 单精度浮点数、双精度浮点数
    - 特殊数的表示形式
- 十进制数的二进制编码（BCD码）

数值数据的表示

- 数值数据表示的三要素
  - 进位计数制
  - 定、浮点表示
  - 如何用二进制编码

  即：要确定一个数值数据的值必须先确定这三个要素。

  例如，机器数 01011001的值是多少? 答案是：不知道!

- 定/浮点表示（解决小数点问题）
  - 定点整数、定点小数
  - 浮点数（可用一个定点小数和一个定点整数来表示）
- 定点数的编码（解决正负号问题）
  - 原码、补码、反码、移码 （反码很少用）

Sign and Magnitude （原码的表示）

Decimal	Binary	Decimal	Binary
0	0000	-0	1000
1	0001	-1	1001
2	0010	-2	1010
3	0011	-3	1011
4	0100	-4	1100
5	0101	-5	1101
6	0110	-6	1110
7	0111	-7	1111

- 容易理解， 但是：
  - ✓ 0 的表示不唯一，故不利于程序员编程
  - ✓ 加、减运算方式不统一
  - ✓ 需额外对符号位进行处理，故不利于硬件设计
  - ✓ 特别当 a<b时，实现 a-b比较困难

从 50年代开始，整数都采用补码来表示
但浮点数的尾数用原码定点小数表示

补码特性 - 模运算（modular运算）

重要概念：在一个模运算系统中，一个数与它除以“模”后的余数等价。

时钟是一种模12系统（0 ≡ 12 ≡ 24 ≡ 36， 3 ≡ 15 ， 6 ≡ 18 ）

假定钟表时针指向10点，要将它拨向6点， 则有两种拨法：

① 倒拨4格：10- 4 = 6

② 顺拨8格：10+8 = 18 ≡ 6 (mod 12)

模12系统中： 10- 4 ≡ 10+8 (mod 12)

- 4 ≡ 8 (mod 12)

则，称8是- 4对模12的补码（即：- 4的模12补码等于8）。

同样有 -3 ≡ 9 （mod 12）

-5 ≡ 7 （mod 12）等

结论1： 一个负数的补码等于模减该负数的绝对值。（正数的补码就是它自己）

结论2： 对于某一确定的模，某数减去小于模的另一数，总可以用该数加上另一数的相反数的补码来代替。

补码（modular运算）：+ 和– 的统一

补码的表示

现实世界的模运算系统举例

例1：“钟表”模运算系统

假定时针只能顺拨，从10点倒拨5格后是几点？

$10-5 = 10+(12-5) = 10+7 = 5$ （mod 12）

取模即只留余数，超过模“12”的部分被丢弃！相当于17−12 = 5留下了

例2：“4位十进制数” 模运算系统

假定算盘只有四档，且只能做加法，则在算盘上计算9028-1713等于多少？

$9028-1713=9028+(10^4-1713)$

$=9028+8287$

$=\boxed{1}7315$

$=7315$（$\bmod 10^4$）

取模即只留余数，高位“1”被丢弃！

（超过模10000的部分被丢弃）

相当于只有低4位留在算盘上。

计算机中的运算器是模运算系统

模是多少？ 2^8

8位二进制加法器模运算系统

计算0111 1111 - 0100 0000 = ?

0111 1111 + (- 0100 0000) = 0111 1111 + (2^8- 0100 0000)

=0111 1111 + 1100 0000 = $\boxed{1}$ 0011 1111 (mod 2^8)

= 0011 1111

只留余数，1被丢弃

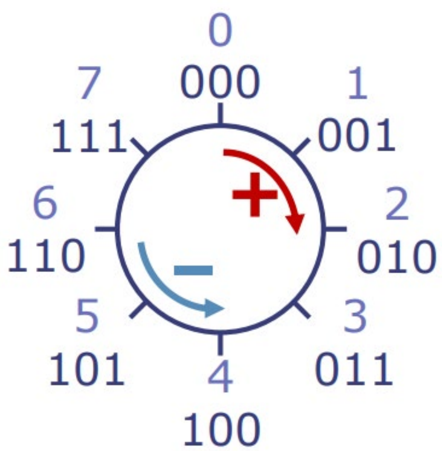
结论1： 一个负数的补码等于对应正数（该负数的绝对值）的“各位取反、末位加1”

注意：并不会真的用减法去计算 (2^8- 0100 0000)

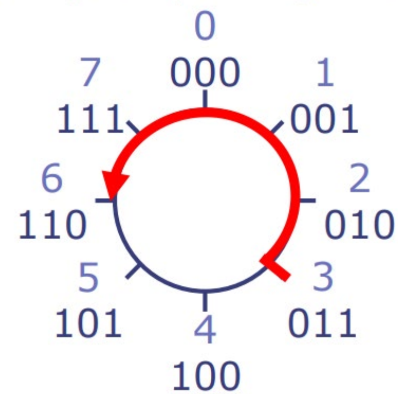
运算器是一个模运算系统

计算机中运算器只有有限位。假定为n位，则运算结果只能保留低n位，其模为2^n。

模为2^3



Example: (3 – 5) mod 2^3 ?



0~7

Compute 3 – 2 using 3-bit 2's complement addition

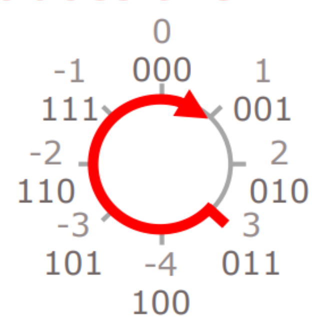
3: 011

2: 010

-2: 110

$$
\begin{array}{r}
\textit{11}\\
011\\
+\quad 110\\
\hline
\underbrace{\textit{1001}}_{\text{Keep only last 3 bits}}
\end{array}
$$

What does this 1 mean?



Zero crossing

0123
-4
-3
-2
-1

运算器是一个模运算系统

补码的定义　假定补码有n+1位，则：

定点整数：$[X]_{补} = 2^{n+1} + X \quad (-2^n \le X < 2^n, \bmod 2^{n+1})$

4位中最高位即为符号位
1代表负数
0代表正数

定点小数：$[X]_{补} = 2 + X \quad (-1 \le X < 1, \bmod 2)$

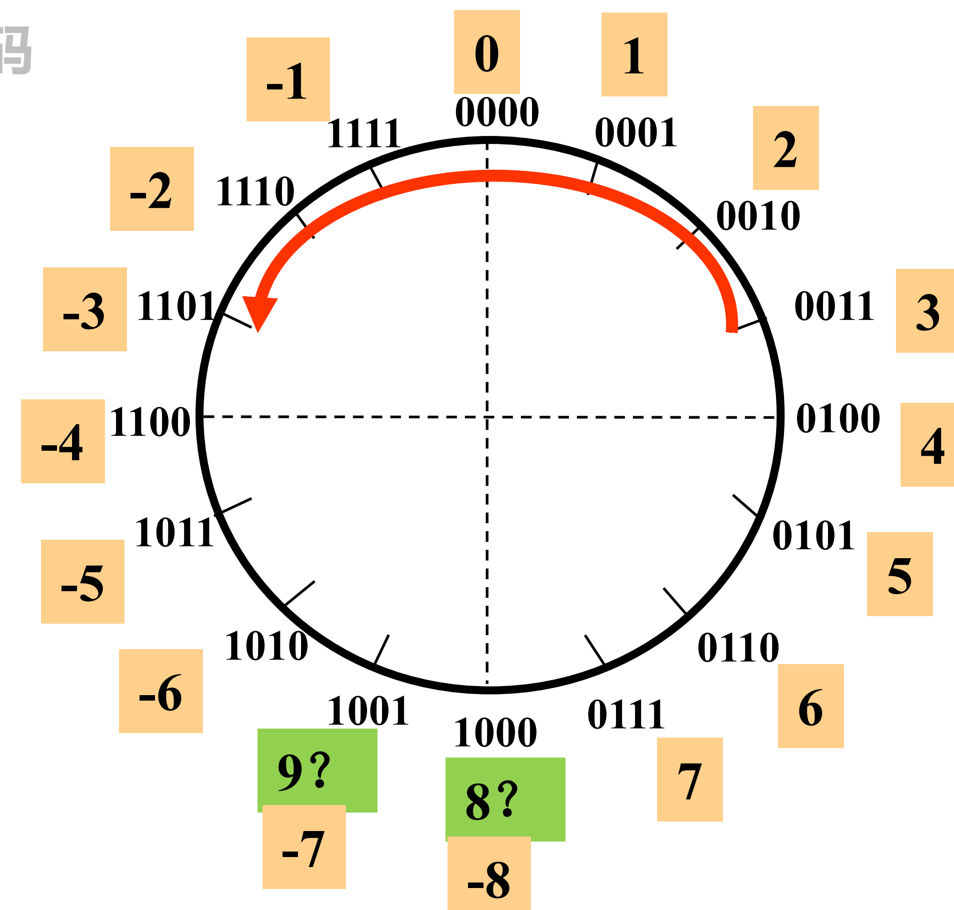
注：实际上在计算机中并不使用补码定点小数！无需掌握该知识点

当n+1=4时，共有16个机器数：0000 ~ 1111，可看成是模为 2^4 的钟表系统。

真值的范围为 -8 ~ +7

3: 0011
6: 0110
-6: 1010

$$\begin{array}{r} {\scriptstyle 1} \\ 0011 \\ +\ 1010 \\ \hline 1101 \end{array}$$



求特殊数的补码

假定机器数有n位

——也就是一个补码共占用n位，包括符号位

$[X]_{补}= 2^n + X \quad (-2^{n-1} \le X < 2^{n-1}，\mathrm{mod}\ 2^n)$

① $[-2^{n-1}]_{补}= 2^n - 2^{n-1} = 10\ldots0$（n-1个0）　（mod 2^n）

② $[-1]_{补}= 2^n - 0\ldots01 = 11\ldots1$（n个1）　（mod 2^n）

③ $[+0]_{补}= [-0]_{补}= 00\ldots0$（n个0）

补码与真值之间的简便转换

例: 设机器数有8位，求123和-123的补码表示。

如何快速得到123的二进制表示?

解: 123 = 127- 4 = 01111111B - 100B = 01111011B

-123= - 01111011B

$[+01111011]_{补} = 2^8 + 01111011 = 100000000 + 01111011$

$= 01111011 \pmod{2^8}$，即 7BH。

$[-01111011]_{补} = 2^8 – 01111011 = 10000\ 0000 – 0111\ 1011$

$= 1111\ 1111 – 0111\ 1011 +1$

1000 0100

$= 1000\ 0100 +1$ ← 各位取反，末位加1

$= 1000\ 0101$，即 85H。

Two’s Complement （补码的表示）

- 正数：符号位（sign bit）为0，数值部分不变
- 负数：符号位为1，数值部分“各位取反，末位加1”

变形（模4）补码：双符号，用于存放可溢出的中间结果。

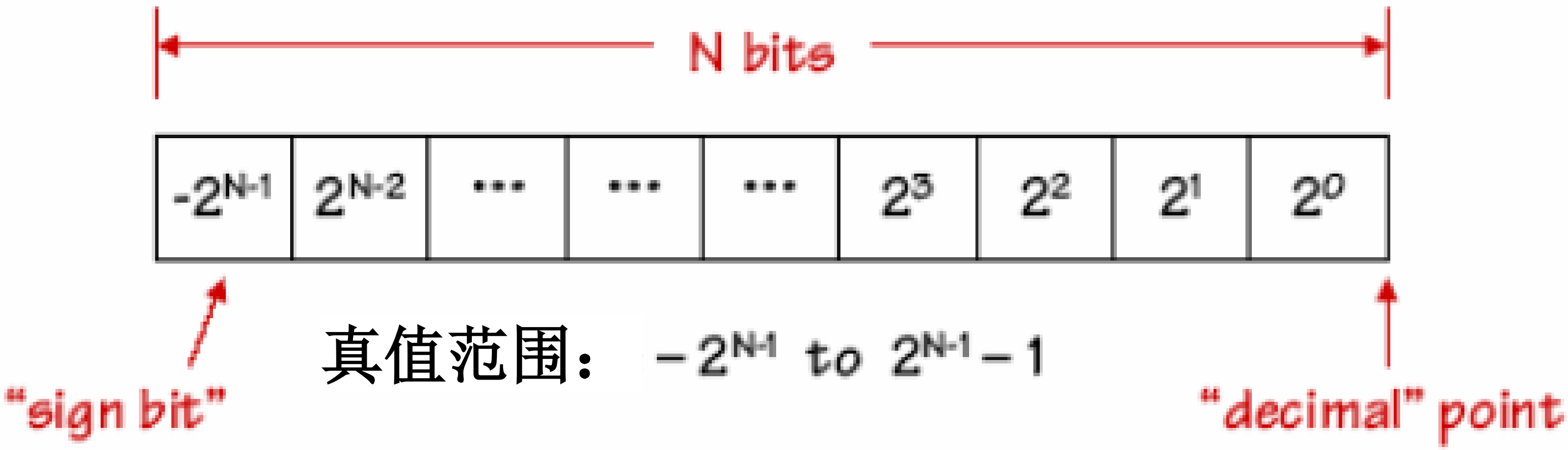
+0和-0表示唯一

Decimal	补码	变形补码	Decimal	Bitwise Inverse	补码	变形补码
0	0000	00000	-0	1111	0000	00000
1	0001	00001	-1	1110	1111	11111
2	0010	00010	-2	1101	1110	11110
3	0011	00011	-3	1100	1101	11101
4	0100	00100	-4	1011	1100	11100
5	0101	00101	-5	1010	1011	11011
6	0110	00110	-6	1001	1010	11010
7	0111	00111	-7	1000	1001	11001
8	1000	01000	-8	0111	1000	11000

值太大，用4位补码无法表示，故“溢出”！但用变形补码可保留符号位和最高数值位。

如何求补码的真值

根据补码各位上的“权”，可以求出一个补码的值



真值范围： -2^{N-1} to $2^{N-1}-1$

8-bit 2's complement example:

$$11010110 = -2^7 + 2^6 + 2^4 + 2^2 + 2^1 = -128 + 64 + 16 + 4 + 2 = -42$$

符号为0，则为正数，数值部分相同

符号为1，则为负数，数值各位取反，末位加1

例如：补码“11010110”的真值为：-0101010=-(32+8+2)=-42

Excess (biased) notion- 移码表示

- 什么是“excess (biased) notation-移码表示”？

  将每一个数值加上一个偏置常数（Excess / bias）

- 一般来说，当编码位数为n时，bias取 2^{n-1}

  Ex. n=4: $E_{biased} = E + 2^3$ (bias= 2^3 =1000B)

  -8 (+8) ~ 0000B

  -7 (+8) ~ 0001B

  …

  0 (+8) ~ 1000B

  …

  +7 (+8) ~ 1111B

0的移码表示唯一

此时移码和补码仅第一位正好相反

移码主要用来表示浮点数阶码！——为了简化浮点数的编码和计算

Unsigned integer(无符号整数)

- 一般在全部是正数运算且不出现负值结果的场合下，可使用无符号数表示。例如，地址运算，编号表示，等等
- 无符号数的编码中没有符号位
- 能表示的最大值大于位数相同的带符号整数的最大值
  - 例如，8位无符号整数最大是255（1111 1111）

    而8位带符号整数最大为127（0111 1111）
- 总是整数，所以很多时候就简称为“无符号数”

Signed integer（带符号整数）

- 计算机必须能处理正数(positive) 和负数(negative)，包含符号位
- 有三种定点编码方式
  - Signed magnitude （原码）

    现用来表示浮点（实）数的尾数
  - One’s complement （反码）

    现已不用于表示数值数据
  - Two’s complement （补码）

    50年代以来，所有计算机都用补码来表示定点整数
- 为什么用补码表示带符号整数？
  - 补码运算系统是模运算系统，加、减运算统一
  - 数0的表示唯一，方便使用
  - 比原码多表示一个最小负数（提示：原码+0和-0不一样）
  - 与移码相比，其符号位和真值的符号对应关系更直接

不同类型数值的相互转换

例：在32位机器上输出si, usi, i, ui的十进制（真值）和十六进制值（机器数）是什么？

16位带符号数　**short si = -32768;**

16位无符号数　**unsigned short usi = si;**

32位带符号数　**int i = si;**

32位无符号数　**unsingned ui = usi ;**

提示：

$32768=2^{15}$

=1000 0000 0000 0000B

机器数

真值	机器数
si = -32768	80 00
usi = 32768	80 00
i = -32768	FF FF 80 00
ui = 32768	00 00 80 00

真值

机器数不变，但真值变了

16位带符号数 →32位带符号数

16位无符号数 →32位无符号数

现象：

带符号整数：符号扩展

无符号整数：0扩展