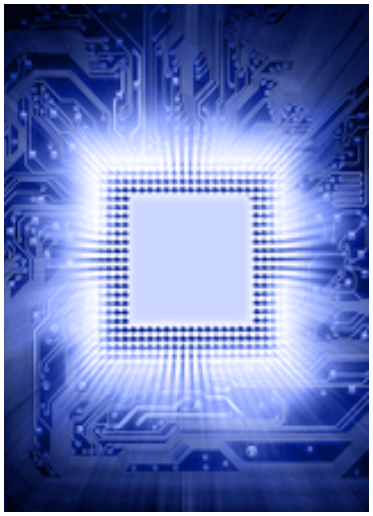
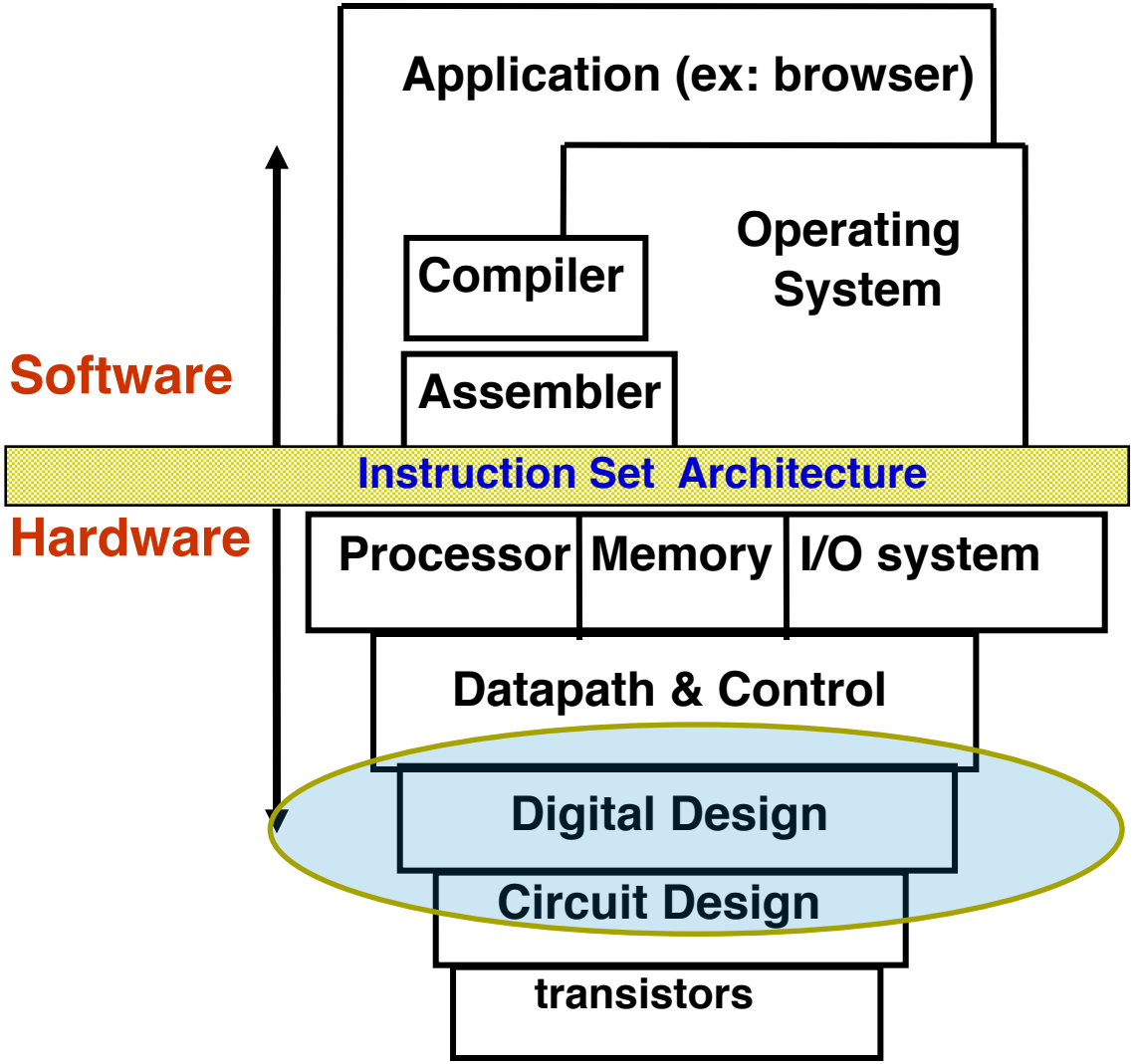
第二章 数制和编码



南京大学人工智能学院

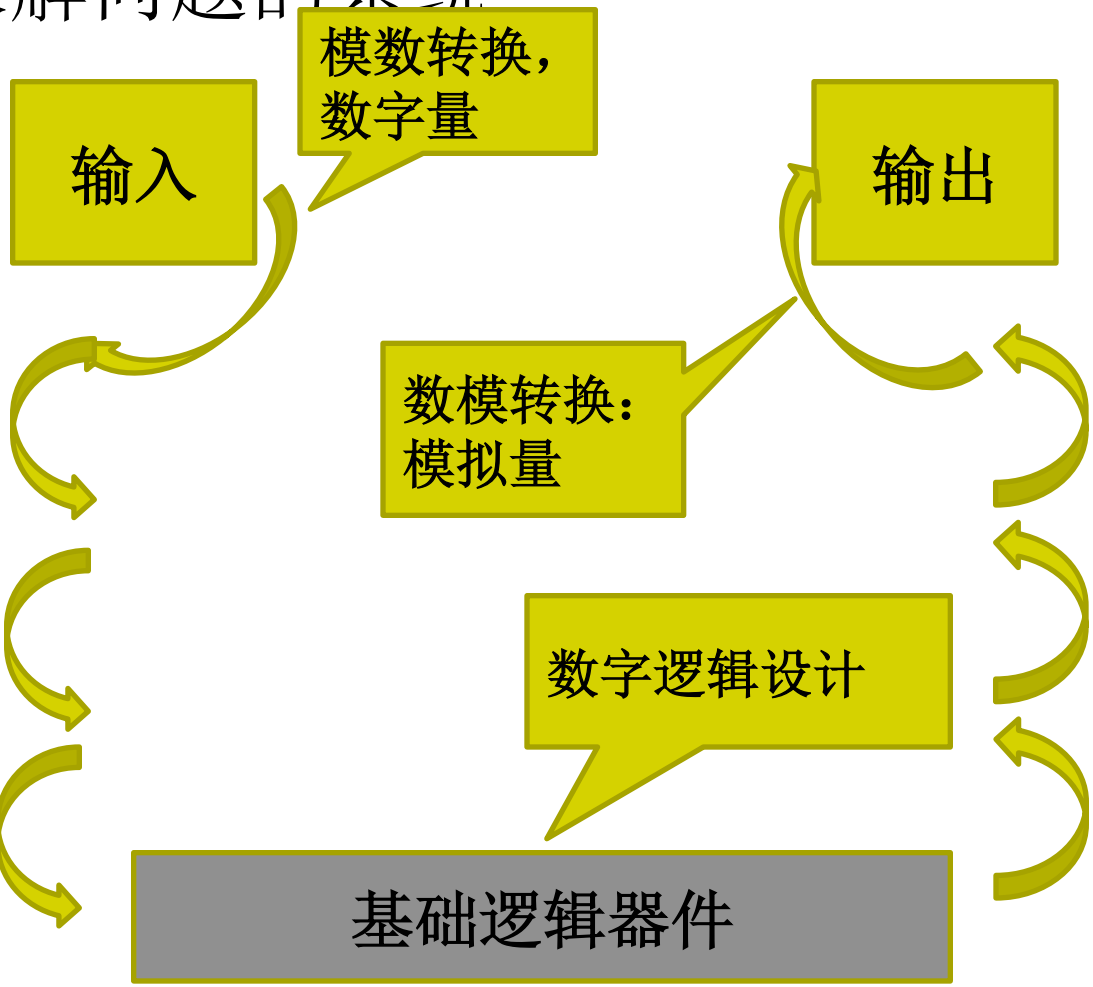
2018-2019 春季

前课回顾


Application (ex: browser)
Operating System
Compiler
Assembler
Software
Instruction Set Architecture
Hardware
Processor
Memory
I/O system
Datapath & Control
Digital Design
Circuit Design
transistors

前课回顾

- 构建求解问题的系统



本章学习导引

- 简单地看，数字系统就是处理二进制数码0和1的电路，其中：
  - “二进制数码”就是现实世界中的对象在数字系统中的表示；（数字化的结果）
  - “处理”的方式和二进制数码表示的对象种类有关系。（数字设计的需求）
- 本章学习的目的：初步了解数字系统中
  - 数值量可以如何表示和处理；
  - 非数值对象可以如何表示。

主要内容

- 数制
  - 按位计数制简介
  - 带符号数值表示及处理
    - 加、减、乘、除
- 编码
  - 十进制数的二进制编码
  - 字符编码
  - 动作、条件和状态编码
  - 检错码和纠错码
  - 面向传输与存储的编码

1、按位计数制

- 用一串数码表示一个数，每个数码的位置对应一个相关的权（**weight**），该数的值就等于所有数码按权展开相加的和。
- 位置表示法

$$N = (a_{n-1}a_{n-2} \ldots a_1a_0 \,.\, a_{-1}a_{-2} \ldots a_{-m})_r \qquad (1.1)$$

where . = radix point

r = radix or base

n = number of integer digits to the left of the radix point

m = number of fractional digits to the right of the radix poin

a_{n-1} = most significant digit (MSD)

a_{-m} = least significant digit (LSD)

1、按位计数制

- 多项式表示法

$$N = a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \ldots + a_0 \times r^0 + a_{-1} \times r^{-1} \ldots + a_{-m} \times r^{-m}$$

$$= \sum_{i=-m}^{n-1} a_i r^i \qquad (1.2)$$

r^i:weight(权)

$$N = (251.41)_{10} = 2 \times 10^2 + 5 \times 10^1 + 1 \times 10^0 + 4 \times 10^{-1} + 1 \times 10^{-2}$$

$(327.86)_{10} = 3\times10^2 + 2\times10^1 + 7\times10^0 + 8\times10^{-1} + 6\times10^{-2}$

1、按位计数制

- 二进制数
  - 每一位只取{0，1}。
  - 可以用两个不同稳定状态的电子元件来表示，存储可靠。
  - 基数为2，计数规律是逢二进一。

$(1101.101)_2=$

$1x2^3+1x2^2+0x2^1+1x2^0+1x2^{-1}+0x2^{-2}+1x2^{-3}$

1Byte = 8bit, 1KB(kilo) = 2^{10}Byte = 1024Byte

1MB(mega) = 2^{20}Byte = 1048576Byte

1GB (giga) = 2^{30}Byte

1TB(TeraByte) = 2^{40}Byte

1、按位计数制

- 二进制数的书写不太方便，平时喜欢使用八进制或十六进制书写。
- 八进制数
  - 每一位只取{0，1，2，3，4，5，6，7}。
  - 基数为8，计数规律是逢八进一。

$(67.731)_8=6x8^1+7x8^0+7x8^{-1}+3x8^{-2}+1x8^{-3}$

- 十六进制数
  - 每一位只取{0，…，9，A,B,C,D,E,F}。
  - 基数为16，计数规律是逢十六进一。

$(8AE6)_{16}=8x16^3+Ax16^2+Ex16^1+6x16^0=(35558)_{10}$

推广到r进制数

- r：基数（base或radix），$r \geq 2$
- 一个p位的r进制的整数：

$$D_1 : (d_{p-1}d_{p-1}d_{p-3}\ldots d_1 d_0)_r$$

- 其值为：

$$D_1 = \sum_{i=0}^{p-1} d_i \times r^i$$

- 其中 $\leqslant d_i \leqslant$

任意进制数

$$D : (d_{p-1}d_{p-2}\ldots d_1 d_0 \cdot d_{-1}d_{-2}\ldots d_{-n})_r$$

$$D = \sum_{i=0}^{p-1} d_i \times r^i$$

各进制特点对照表

数制	基数	数码	计数规则	一般表达式	计算机中英文表示
十进制	10	0~9	逢十进一	$N_{10}=\sum_{i=-m}^{n-1}a_i 10^i$	D
二进制	2	0、1	逢二进一	$N_2=\sum_{i=-m}^{n-1}b_i 2^i$	B
八进制	8	0~7	逢八进一	$N_8=\sum_{i=-m}^{n-1}a_i 8^i$	O
十六进制	16	0~9、ABCDEF	逢十六进一	$N_{16}=\sum_{i=-m}^{n-1}a_i 16^i$	H
N进制	N	0~（N-1）	逢N进一	$N_N=\sum_{i=-m}^{n-1}a_i N^i$	

常用的几种按位计数制

二进制	十进制	八进制	十六进制	4bit binary
0	0	0	0	0000
1	1	1	1	0001
10	2	2	2	0010
11	3	3	3	0011
100	4	4	4	0100
101	5	5	5	0101
110	6	6	6	0110
111	7	7	7	0111
1000	8	10	8	1000
1001	9	11	9	1001
1010	10	12	A	1010
1011	11	13	B	1011
1100	12	14	C	1100
1101	13	15	D	1101
1110	14	16	E	1110
1111	15	17	F	1111

按位计数制的相互转换

- 替代法
  - 将待转换的数字表示成多项式的格式

$$N = a_{n-1}r^{n-1} + \ldots + a_0 r^0 + a_{-1}r^{-1} + \ldots + a_{-m}r^{-m}$$

  - 转换过程：（A进制向B进制转）
    - 将数字表示成A进制的多项式格式。
    - 将该多项式中的所有数字，转换成B进制中的数
    - 用B进制的运算，重新计算多项式的值。
      - $(11010)_2 \rightarrow (\ ?\ )_{10}$

$$\begin{aligned} N &= 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0 \\ &= (16)_{10} + (8)_{10} + 0 + (2)_{10} + 0 \\ &= (26)_{10} \end{aligned}$$

      - $(627)_8 \rightarrow (\ ?\ )_{10}$

$$\begin{aligned} N &= 6\times8^2 + 2\times8^1 + 7\times8^0 \\ &= (384)_{10} + (16)_{10} + (7)_{10} \\ &= (407)_{10} \end{aligned}$$

按位计数制的相互转换

例：　$10011101.01)_2 = (\underline{010}\ \underline{011}\ \underline{101}.\ \underline{010})_2$

$= (235.2)_8$

$(753.4)_8 = (\underline{111}\ \underline{101}\ \underline{011}.\ \underline{100})_2$

$= (111101011.1)_2$

$(1011101000.011)_2 = (\underline{0010}\ \underline{1110}\ \underline{1000}.\ \underline{0110})_2$

$= (2E8.6)_{16}$

$(3FD.B)_{16} = (\underline{0011}\ \underline{1111}\ \underline{1101}.\ \underline{1011})_2$

$= (1111111101.1011)_2$

按位计数制的相互转换

- 除以基数取余法
  - 主要用于整数的转换，A->B
  - 理论基础:
    - $(N_I)_A = b_{n-1}B^{n-1} + \ldots + b_0B^0$

      这里, b_i's represents the digits of $(N_I)_B$ in base A.
    - $N_I / B = (b_{n-1}B^{n-1} + \ldots + b_1B^1 + b_0B^0) / B$

      $=$ (商 Q_1: $b_{n-1}B^{n-2} + \ldots + b_1B^0$) + (余数 R_0: b_0)
    - 一般来说, $(b_i)_A$ 就是Q_i 除以$(B)_A$ 的余数R_i.

按位计数制的相互转换

- 除以基数取余法
  - 转换过程

    1. $(N_l)_A$ 除以 $(B)_A$, 得到Q_1 和 R_0.
       R_0 是结果的最低位（LSD）d_0.
    2. 计算d_i, $i = 1 \ldots n-1$
       用Q_i 除以 $(B)_A$, 得到 Q_{i+1} 和R_i, R_i 就是d_i.
    3. 当$Q_{i+1} = 0$时，停止过程.

按位计数制的相互转换

→把十进制的173转换为二进制

$$D:(d_{p-1}d_{p-1}...d_1d_0)_2$$

$$D=\sum_{i=0}^{p-1}d_i\times 2^i$$

$$=d_{p-1}\cdot 2^{p-1}+...+d_2\cdot 2^2+d_1\cdot 2^1+d_0$$

=173

2	173	……余1
2	86	……余0
2	43	……余1
2	21	……余1
2	10	……余0
2	5	……余1
2	2	……余0
2	1	……余1
	0	

$173_{10}=10101101_2$

按位计数制的相互转换

- 除以基数取余法
  - ***Examples***
    - $(315)_{10} = (473)_8$

$$
\begin{array}{r|l|l}
8 & 315 & 3 \\
8 & 39 & 7 \\
8 & 4 & 4 \\
 & 0 &
\end{array}
\quad \uparrow \quad \text{LSD} \quad \text{MSD}
$$

    - $(315)_{10} = (13B)_{16}$

$$
\begin{array}{r|l|l}
16 & 315 & B \\
16 & 19 & 3 \\
16 & 1 & 1 \\
 & 0 &
\end{array}
\quad \uparrow \quad \text{LSD} \quad \text{MSD}
$$

    - $(315)_7 = (237)_8$?

按位计数制的相互转换

- **乘以基数取整法**
  - 用于转换小数部分.
  - 理论基础:
    - $(N_F)_A = b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-m}B^{-m}$

      这里, $(N_F)_A$ 是A进制小数， b_i's 是 $(N_F)_B$ 的A进制数字。
    - $B \times N_F = B \times (b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-m}B^{-m})$

      $=$ (整数 I_{-1}: b_{-1}) + (小数 F_{-2}: $b_{-2}B^{-1} + \ldots + b_{-m}B^{-(m-1)}$)
    - 一般来说, $(b_i)_A$ 是$F_{-(i+1)} \times (B_A)$的整数部分 I_{-i}.

按位计数制的相互转换

- 乘以基数取整法
  - 转换过程

1. 设 $F_{-1} = (N_F)_A$.

2.用 F_{-i} 乘以 $(B)_A$, 计算 $(b_{-i})_A$, 其中$i = 1 \ldots m$, 得到整数 I_{-i}, 表示 $(b_{-i})_A$, 以及小数$F_{-(i+1)}$.

3. 把 $(b_{-i})_A$ 改写成*B进制数*.

4. 直到 fraction=0或到达最大有效数字

乘基数取整法

→把十进制的0.8125转化为二进制

$$D=\sum_{i=-n}^{-1} d_i \times 2^i$$

$$=d_{-1}\cdot 2^{-1}+d_{-2}\cdot 2^{-2}+\ldots+d_{-n}\cdot 2^{-n}$$

0.8125 × 2 = 1.6250 ……整数部分＝1

0.6250 × 2 = 1.2500 ……整数部分＝1

0.2500 × 2 = 0.5000 ……整数部分＝0

0. 5000 × 2 = 1. 0000 ……整数部分＝1

$0.8125_{10}=0.1101_2$

按位计数制的相互转换

- *Examples*
  - $(0.479)_{10} = (0.3651\ldots)_8$

    MSD $3.832 \leftarrow 0.479 \times 8$

    $6.656 \leftarrow 0.832 \times 8$

    $5.248 \leftarrow 0.656 \times 8$

    LSD $1.984 \leftarrow 0.248 \times 8$

    …

  - $(0.479)_{10} = (0.0111\ldots)_2$

    MSD $0.9580 \leftarrow 0.479 \times 2$

    $1.9160 \leftarrow 0.9580 \times 2$

    $1.8320 \leftarrow 0.9160 \times 2$

    LSD $1.6640 \leftarrow 0.8320 \times 2$

    …

按位计数制的相互转换

- 通用转换过程
  - 算法1

    A进制数N转换成B进制数。

    (a) 用B进制数取代展开序列中的数字，并计算结果，或

    (b) 基于A进制运算，计算B基数的乘除法。

  - 算法2

    A进制数N转换成B进制数。

    (a) 用序列替代法将A进制数转换成10进制。

    (b) 用基数乘除法，将10进制数转换成B进制数。

    这种方法稍麻烦，但管用。

按位计数制的相互转换

- ***Example***

  $(18.6)_9 = (\ ?\ )_{11}$

  (a) 10进制的序列替代法:

  $$N_{10} = 1 \times 9^1 + 8 \times 9^0 + 6 \times 9^{-1}$$
  $$= 9 + 8 + 0.666\ldots$$
  $$= (17.666\ldots)_{10}$$

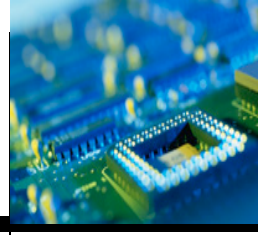
  (b) 11进制的基数乘除法

  $11 \mid 17 \quad 6$
  $11 \mid 1 \quad 1$
  0

  $7.326 \leftarrow 0.666 \times 11$
  $3.586 \leftarrow 0.326 \times 11$
  $6.446 \leftarrow 0.586 \times 11$

  $N_{11} = (16.736\ \ldots)_{11}$

进制数转换的启示



- 自然界中的量都可以用某个数制表示出来吗？
- 各种数制之间的表达能力一样吗？
- 是否存在最强表达能力的数制？

按位计数制的加减法

- 和十进制的运算过程差不多，就是加法表/减法表不同。

加法

X	190
Y	+ 141
$X+Y$	331

减法

X	229
Y	− 46
$X-Y$	183

		0	10	1	1	10	10	
	1	1	1	0	0	1	0	1
−	0	0	1	0	1	1	1	0
	1	0	1	1	0	1	1	1

X	210
Y	− 109
$X-Y$	101

	0	10	10	0	1	10	0	10
	1	1	0	1	0	0	1	0
−	0	1	1	0	1	1	0	1
	0	1	1	0	0	1	0	1

2、带符号数值表示及处理

- 符号-数值表示法
- 补码数制
- 余码
- 二进制补码加减法
- 二进制反码加减法
- 二进制乘法
- 二进制除法

主要讨论的是负数的表示方法，但要和正数统一来考虑。而且都是在n位的表示空间中讨论相关表示法。

符号-数值表示法

- 一个数是由表示该数为正或者负的符号和数值两部分组成。
  - +98，-57，+123.5，-13，
  - +0=-0
- 应用于二进制时（原码），增加符号位：
  - [+43]的8位原码为：00101011
  - [-43]的8位原码为：10101011
- 特点：
  - 正整数和负整数数量相同，零有两种。
  - n 个二进位的原码可表示的数值范围是：$-2^{n-1}+1 \sim 2^{n-1}-1$

补码数制

- 两种：
  - 基数补码表示和基数减1补码表示法（反码）
- 基数补码表示法计算方法：
  1. 基数为r的n位数的补码等于从r^n中减去该数。
  2. 按位取反加一。
     - $r^n-D=((r^n-1)-D)+1$; $(r^n-1)-D$表示反码，r-1-d
- 补码的性质：
  - 0是唯一表示的(用全0表示数值“0”)
  - 不对称，负整数比正整数多1个（例如，10000000表示-128）
  - n 个二进位的补码可表示的数值范围是：-2^{n-1} ～$2^{n-1}-1$

二进制数的反码和补码表示法

- 二进制数的反码

二进制数 11101010

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

反码 00010101

逐位取反

- 二进制数的补码

二进制数 11101010

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

反码 00010101

\+ 1

补码 00010110

逐位取反，再加1

•反码具有对称性，但0有两种表示。
•符号数在进行位扩展时，符号位也要扩展。

计算二进制数补码的两种其它方法

第一种，从右边开始数，将第一个1之后的位取反

二进制数 11101010
补码 00010110

11101000
00011000

第二种，对于一个N位的二进制数，其补码 = 2^N-二进制数

例如：对于11101010_2

100000000
_ 11101010
补码 00010110

11101000_2

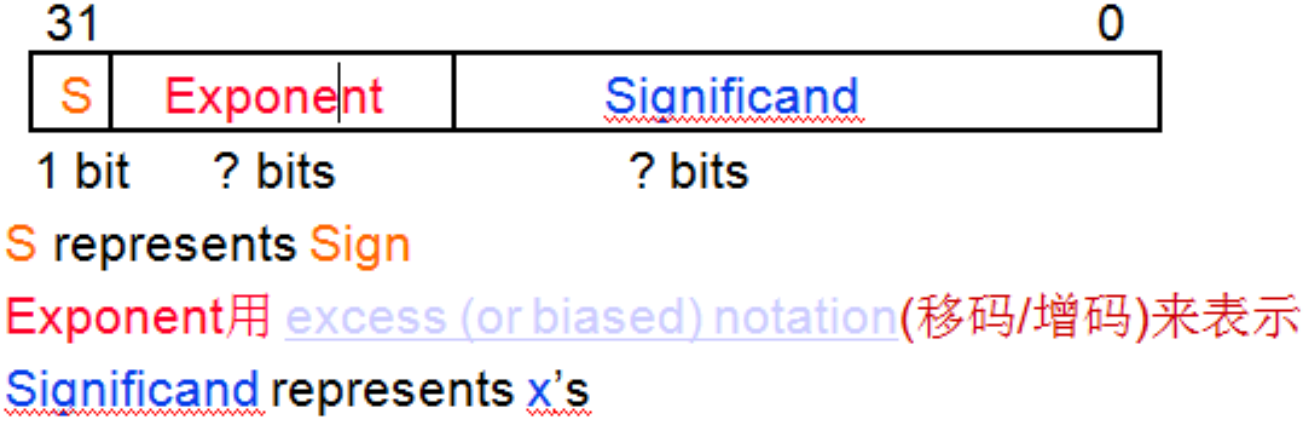
100000000
− 11101000
补码 00011000

注意：

- 谈论一种码制，如：补码、反码，其实同时意味着两个含义：
  - 数值编码的计算方法
    - 取反加一
    - 按位取反…
  - 某种应用（设计场合）下，具体的编码方案，即：某个具体数值，其对应编码的表示形式。
    - 需要表示的数值范围
    - 具体表示形式
    - …

余码

- M位码串的余B表示法
  - 直接用来表示M-B的有符号整数，其中B叫做数制的偏离。
- 例如：
  - 余2^{M-1}数制中，将【-2^{M-1}，$+2^{M-1}-1$】范围的数X+2^{M-1}数表示。
  - 非负，小于2^{M}。
- 余码表示用在浮点数系统中指数的表示。



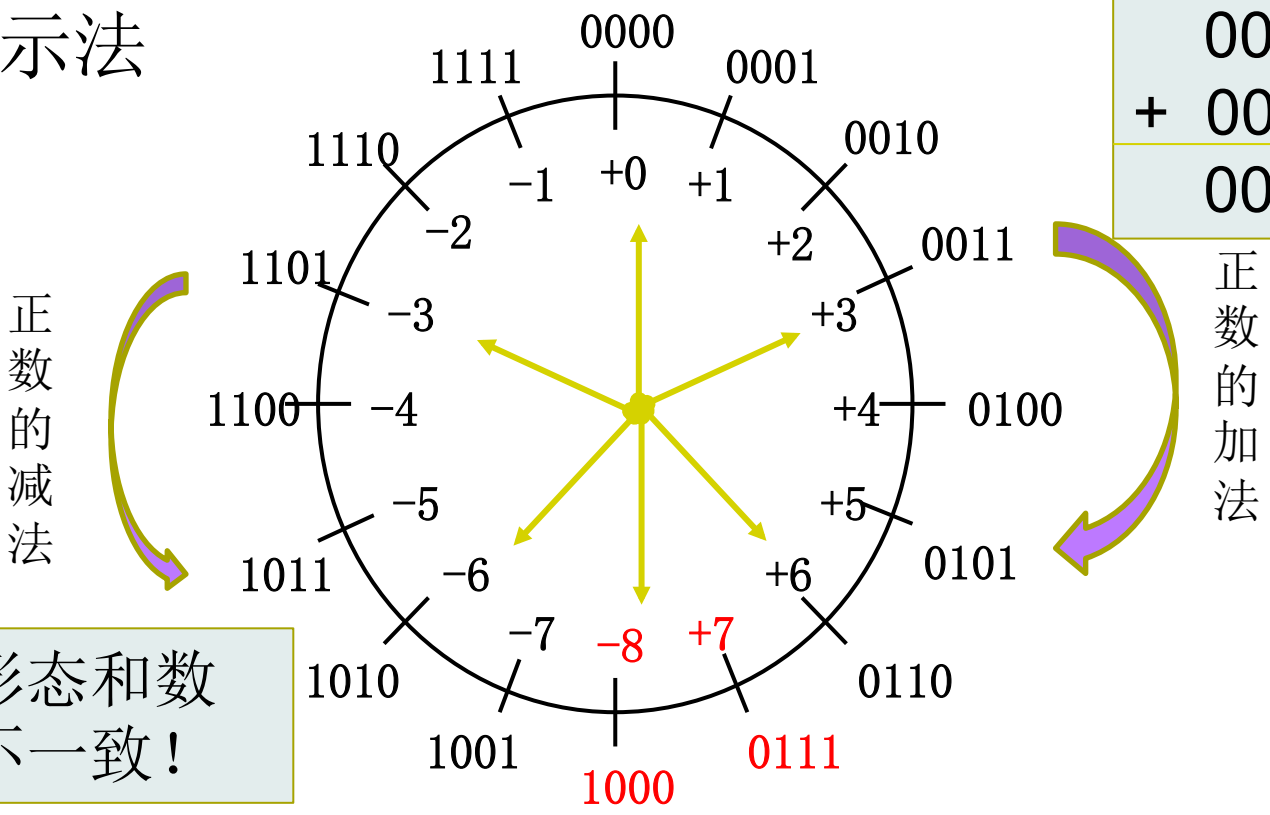
- 现代计算机中，带符号整数都使用补码表示，CPU直接对补码进行运算和处理！
- 原码、反码、补码、余n码等都是数值类量的编码方案。
- 一个二进制表示，要知道其编码方案才能知道具体的值：

  11101010

  - 原码？反码？补码？余2^7码？
  - 严格意义上应该叫：用*码编制的8位二进制整数

二进制补码加减法

- 加法规则
  - 二进制补码数可以按照普通的二进制加法相加，只要不超过计数系统的范围，该结果就是正确的
- 图示法



例1：15 加12

```
  00001111 (15)
+ 00001100 (12)
  00011011 (27)
```

有两处形态和数值上的不一致！

二进制补码加减法

- 减法处理方法：
  - 1、和普通无符号二进制数一样进行减法运算。
  - 2、转为补码加法
    - (1)将减数变负取补码;
    - (2)将减数和被减数按正常的加法规则相加即可。
  - 示例

例1：15 减12

```
    00001111  (15)
+   11110100  (-12 )补
  100000011  (3)
```

例1：15 减-12

```
    00001111  (15)
+   00001100  (12 )补
  000011011  (27)
```

- 通过检查被减数和取补后减数的符号可以检测溢出。

进位与溢出

- **进位：**是指运算结果的最高位向更高位的进位。
  - 可用来判断无符号数运算结果是否超出了计算机所能表示的最大无符号数的范围。
- **溢出(overflow)：**运算结果是否超出了码制表示的数值范围
  - 如带符号数补码运算结果超出了补码所能表示的范围，就叫补码溢出，简称溢出。
- 补码溢出判断：
  - 异号数相加决不会溢出。
  - 加数符号相同，而和不同，则有溢出。
  - 也可以通过计算过程中的进位来判断。
    - 进位和符号位不一致

进位与溢出

$(63)_{10}+(67)_{10}=$?

$$
\begin{array}{rr}
 & 0\ 011,1111 \\
+) & 0\ 100,0011 \\
\hline
 & \textcircled{1}\ 000,0010
\end{array}
$$

$(-65)_{10}+(-67)_{10}=$?

$$
\begin{array}{rr}
 & 1\ 011,1111 \\
+) & 1\ 011,1101 \\
\hline
 & 1\ \textcircled{0}\ 111,1100
\end{array}
$$

进位与溢出

- **溢出的判断方法，**常见的有：

① 通过参加运算的两个数的符号及运算结果的符号进行判断

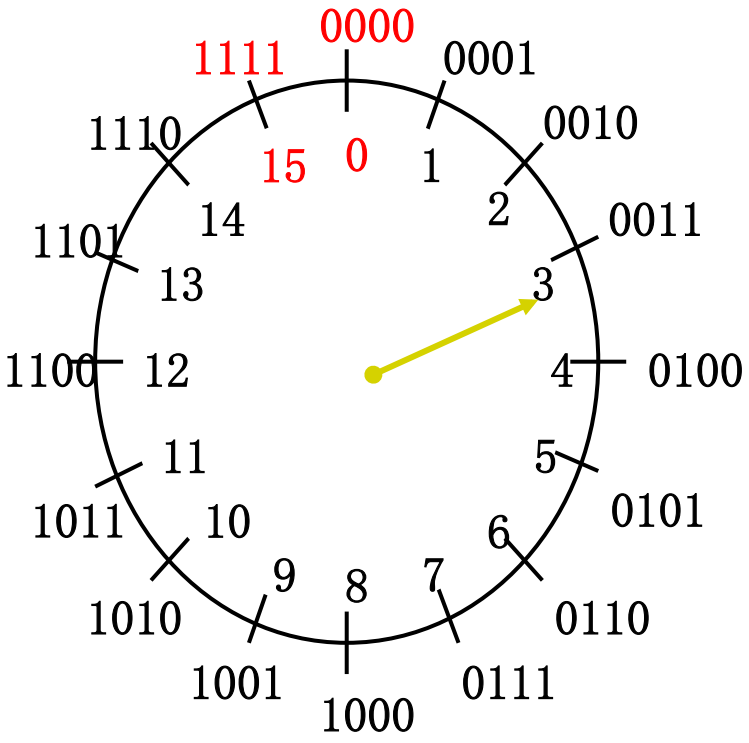
② ***单符号位法***。该方法通过符号位和数值部分最高位的进位状态来判断结果是否溢出。

③ ***双符号位法***，又称为变形补码法。它是通过运算结果的两个符号位的状态来判断结果是否溢出。

第一种方法仅适用于手工运算
其他两种方法在计算机中都有使用。

二进制补码和无符号二进制数

- 二进制补码和同字长的二进制无符号数的加减算法相同，所以计算机或其他数字系统可以用相同的加法器。
  - 结果的解释不同；
  - 溢出判定方法不同。
- 图示解释：
  - 有符号数：-8—7
  - 无符号数：0 —15



二进制反码加减法

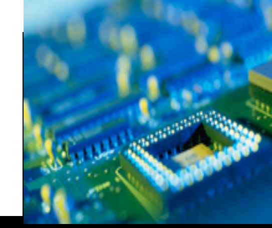
- 反码加法：
  - 采用标准的二进制加法，当计数经过最大值时，结果要额外多加一。
- 反码减法：
  - 将减数取反，做加法。
  - （和补码情况相同）
- 溢出判定：
  - 加数符号相同，而和不同，则有溢出。
  - （和补码情况相同）

+3	0011		+4	0100			+5	0101			
+ +4	+ 0100		+ −7	+ 1000			+ −5	+ 1010			
+7	0111		−3	1100			−0	1111			
−2	1101		+6	0110			−0	1111			
+ −5	+ 1010		+ −3	+ 1100			+ −0	+ 1111			
−7	10111		+3	10010			−0	11110			
	+ 1			+ 1				+ 1			
	1000			0011				1111			

二进制数加法和减法规则总结

数制	加法规则	变负规则	减法规则
无符号数	操作数相加；如果MSB产生进位，则结果超出范围	不适用	被减数减去减数，如果MSB产生借位，则结果超出范围
符号-数值二进制原码	（同号）绝对值相加，如果MSB产生进位，则溢出，结果符号与操作数符号相同 （异号）较大绝对值减去较小绝对值，结果符号与较大绝对值的符号相同	改变操作数的符号位	改变减数的符号位后同加法一样进行计算
二进制补码	做加法，忽略MSB的进位，如果向符号位的进位输入和从符号位的进位输出不同，则产生溢出	逐位取反，末位加1	减数取补后，与被减数相加
二进制反码	做加法，如果MSB有进位，结果加1；如果向符号位的进位输入和从符号位的进位输出不同，则产生溢出	逐位取反	减数逐位取反，同加法一样计算

二进制乘法



- 采用移位—累加方法实现无符号数的乘法。

```
   111           7
×  101        ×  5
------       -----
   111          35
  000
+111
------
100011
```

- 有符号数：
  - 同号相乘为正，异号为负；
  - 取操作数的绝对值，按无符号数乘法计算积。

- 二进制补码：
  - 最高位的权是负的，要特别处理。

```
   –5             1011   multiplicand
× –3         ×    1101   multiplier
             ---------
                 00000   partial product
                 11011   shifted multiplicand
             ---------
                111011   partial product
                00000↓   shifted multiplicand
             ---------
               1111011   partial product
               11011↓↓   shifted multiplicand
             ---------
              11100111   partial product
              00101↓↓↓   shifted and negated multiplicand
             ---------
              00001111   product
```

二进制除法

- 基本方法：
  - 移位—减法。
- 除数为零会产生除法溢出。
- 有符号数的处理方法和乘法相同。

```
      11
10 ) 110
     10
     10
     10
      0
```

```
      3
2 ) 6
    6
    0
```