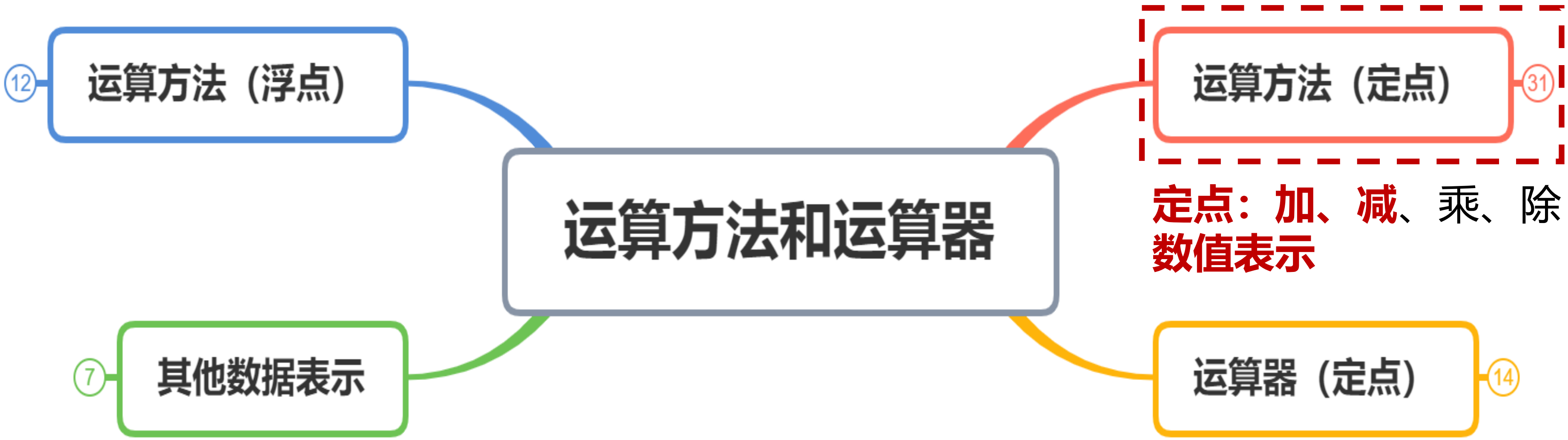
计算机组成与系统结构

第二章 运算方法和运算器（2）

吕昕晨
lvxinchen@bupt.edu.cn

网络空间安全学院

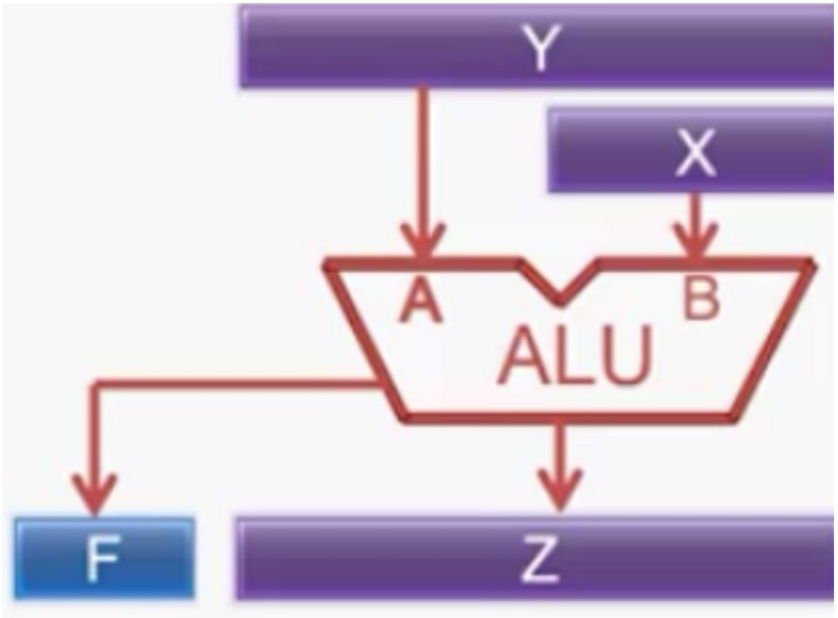
回顾：运算方法与运算器



定点：加、减、乘、除
数值表示

本周教学安排（定点简单运算）
——运算方法和运算器

- **硬件部分**（如何设计基本算数逻辑单元）
  - 逻辑运算
    - 非、与、或
  - 算数运算
    - 加、减法



- **理论部分**
  - 数的表示方法
    - 定点、浮点数、原、反、补码
    - 字符串与汉字等
  - 逻辑、加减运算

第二章 运算方法和运算器
——运算方法（理论）

- 定点与浮点表示法
- 数的机器码表示（原/反/补/移）

数值表示方法

- 定点加法、减法运算
- 溢出检测方法

回顾对照：
加减法硬件设计

- 字符表示与校验

非数值数据、差错控制

第二章 运算方法和运算器
——运算方法（理论）

- 定点与浮点表示法
- 数的机器码表示（原/反/补/移）
- 定点加法、减法运算
- 溢出检测方法
- 字符表示与校验

定点表示法

- 所有数据的小数点位置固定不变
- 理论上位置可以任意，但实际上将数据表示有两种方法（小数点按约定）
  - 纯小数
  - 纯整数

$x_0 . x_1 x_2 x_3 \ldots x_{n-1} x_n .$

纯整数：小数点固定于最后一位之后

符号

量值

纯小数：
小数点固定于符号位之后

数据表示范围

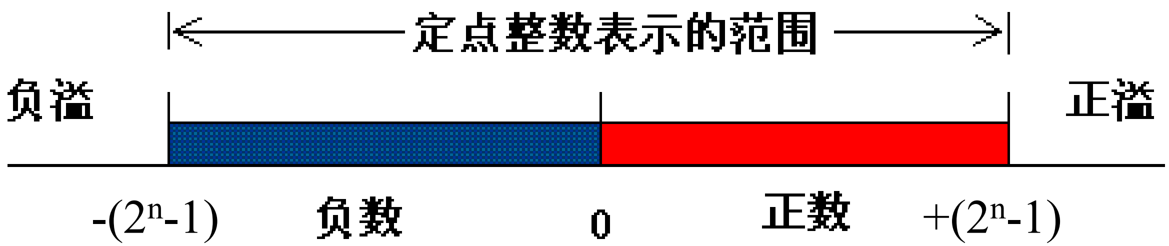
- 最大/小值：量值最大，符号相反
- 分辨率：最低位数值

纯小数

x=0.00...0 x=1.00...0	x=0	正0和负0都是0
x=0.11...1	$x=1-2^{-n}$	最大正数
x=0.00...01	$x=2^{-n}$	最接近0的正数
x=1.00...01	$x=-2^{-n}$	最接近0的负数
x=1.11...1	$x=-(1-2^{-n})$	最小负数

数据表示范围

- 纯整数表示范围



- 定点表示法的特点
  - 定点数表示数的范围受字长限制，数的范围有限;
  - 定点表示的精度有限

[例] 设机器字长16位，采用定点表示，符号位1位，尾数15位

(1)（原码）纯整数表示时，最大正数是多少？最小负数是多少?

0 111 111 111 111 111 最大正整数

$x = (2^{15} - 1)_{10} = (+32767)_{10}$

1 111 111 111 111 111 最小负整数

$x = -(2^{15} - 1)_{10} = (-32767)_{10}$

(2)（原码）纯小数表示， 最大正数是多少？最小负数是多少?

0 111 111 111 111 111 最大正小数

$x = (1 - 2^{-15})_{10}$

1 111 111 111 111 111 最小负小数

$x = -(1 - 2^{-15})_{10}$

浮点表示法

- 为什么需要浮点表示法?
  - 超大数-极小数超出定点表示法范围

    电子质量(克)： $9\times10^{-28} = 0.9\times10^{-27}$

    太阳质量(克)： $2\times10^{33} = 0.2\times10^{34}$
- 什么是浮点表示法?
  - 科学计数法（数字表示）
  - 浮点表示法
    - 数字：10进制、以10为底
    - 机器（浮点表示法）：2进制，以2为底

浮点表示法

- 浮点表示：小数点位置随阶码不同而浮动

  格式:$N=R^{E}.M$

  基数R，取固定的值2，隐含表示

  指数E

  尾数M

- 机器中表示

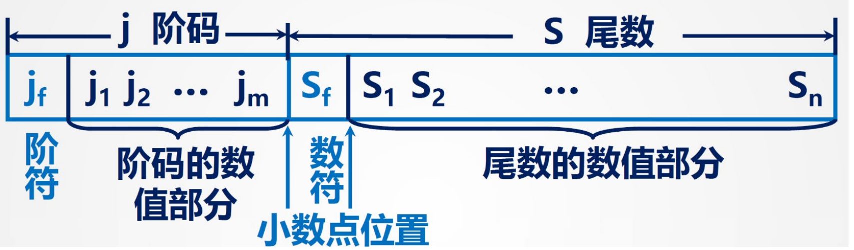
阶符	阶码	数符	尾数

问题1：表示范围?

问题2：表示效率?

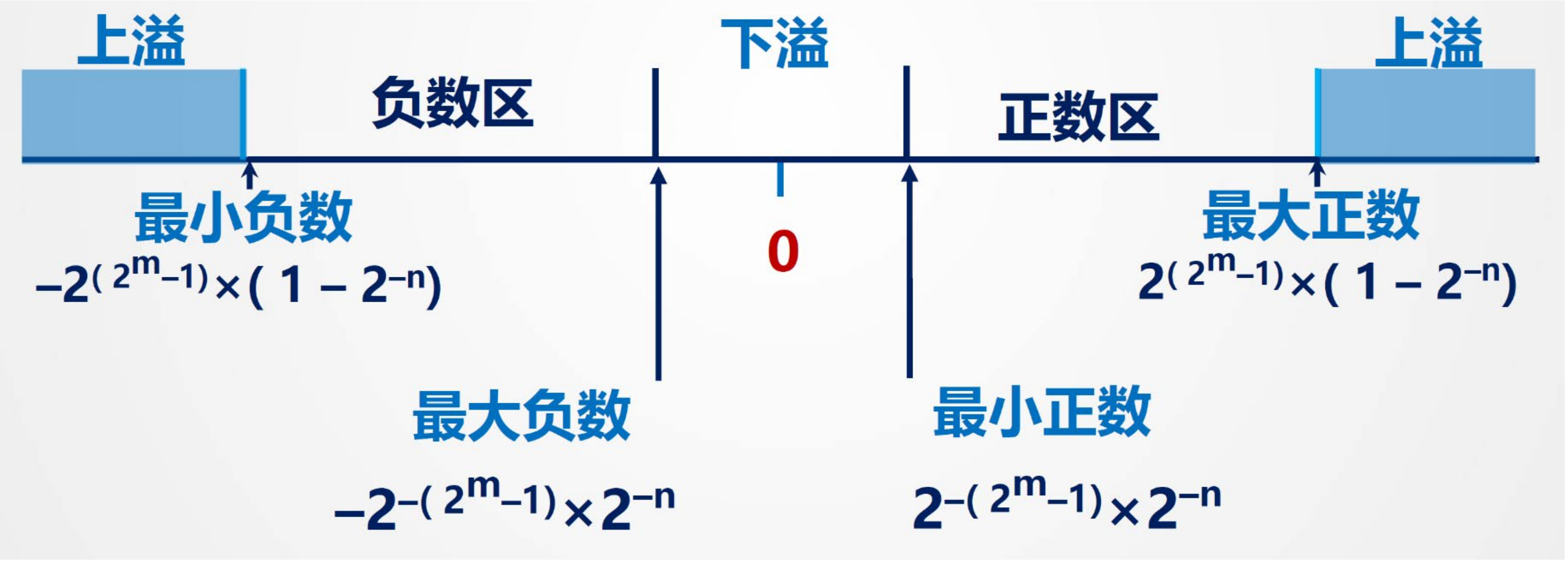
问题1：表示范围?

浮点数：表示范围



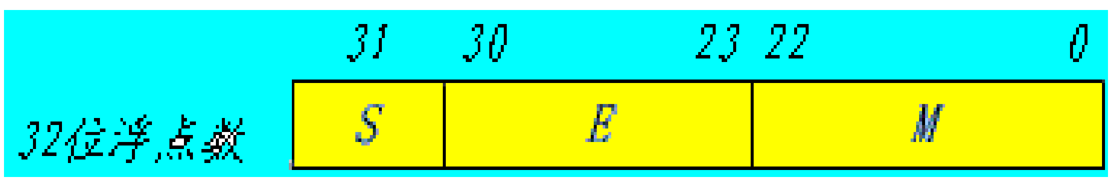
上溢　阶码>最大阶码　中断溢出处理

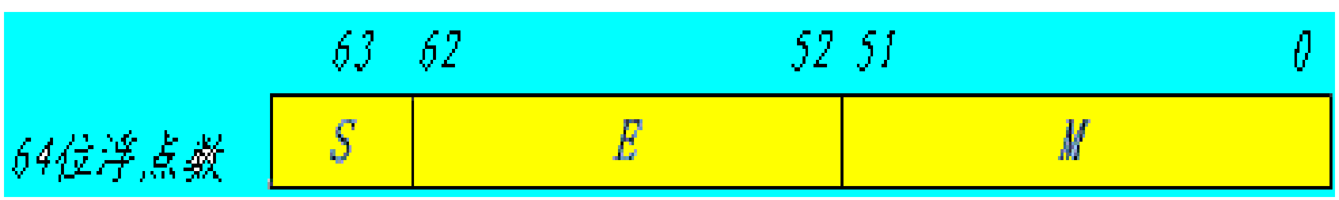
下溢　阶码<最小阶码　按机器零处理



IEEE 754标准（重要）

- IEEE （Institute of Electrical and Electronics Engineers）：电气与电子工程师协会
- IEEE 754标准(规定了浮点数的表示格式等)
  - 规则规定了**单精度(32)**和双精度(64)的基本格式
  - 规则中，尾数用原码，阶码用移码(便于对阶和比较)





IEEE 754标准—32位浮点数

- 基数R=2，基数固定，采用隐含方式来表示它。
- 32位的浮点数：
  - S数的符号位，1位，在最高位，0表示正数，1表示负数
  - M是尾数， 23位，在低位部分，采用纯小数表示
  - E是阶码，8位，采用移码表示。移码比较大小方便
  - 注意：
    - 关于尾数：**尾数域最左位(最高有效位)总是1，故这一位不予存储**，而认为隐藏在小数点的左边。
    - 关于阶码：由于采用移码，指数e加上一个固定的偏移值127(01111111)，即E=e+127

	31	30 … 23	22 … 0
32位浮点数	S	E	M

$$+/-（1.M）\times 2^{e}$$

$$E=e+127$$

例题1—754转十进制

例1 若浮点数x的754标准存储格式为$(41360000)_{16}$，求其浮点数的十进制数值。

解：将16进制数展开后，可得二制数格式为

0　100/0001/0　011/0110/0000/0000/0000/0000

S　阶码(8位)　尾数(23位)

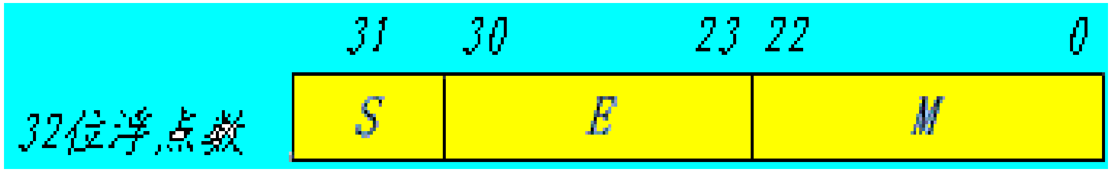
指数e=阶码-127=1000 0010 - 0111 1111=00000011=$(3)_{10}$

包括隐藏位1的尾数

1.M=1.011 0110 0000 0000 0000 0000=1.011011

于是有

x=+(S==0)1.M×2^e=+(1.011011)×2^3=+1011.011=$(11.375)_{10}$



+/-（1.M）×2^e

E=e+127

例题2—十进制转754

例2 将数$(20.59375)_{10}$转换成754标准的32位浮点数的二进制存储格式。

解:首先分别将整数和分数部分转换成二进制数：

20.59375=10100.10011

然后移动小数点，使其在第1，2位之间

$10100.10011=1.\underline{010010011}\times 2^4$

e=4于是得到：　　　M

S=0, E=4+127=131, E=1000 0011

最后得到32位浮点数的二进制存储格式为：

0 10000011 010010011 00000000000000=$(41A4C000)_{16}$

+/-（1.M）×2^e

E=e+127

第二章 运算方法和运算器
——运算方法（理论）

- 定点与浮点表示法
- 数的机器码表示（原/反/补/移）—重要
- 定点加法、减法运算
- 溢出检测方法
- 字符表示与校验

数的机器码表示

- 真值：一般书写的数
- 机器码：机器中表示的数，要解决在计算机内部数的正、负符号和运算问题
  - 原码
  - 反码
  - 补码
  - 移码

加减法运算：

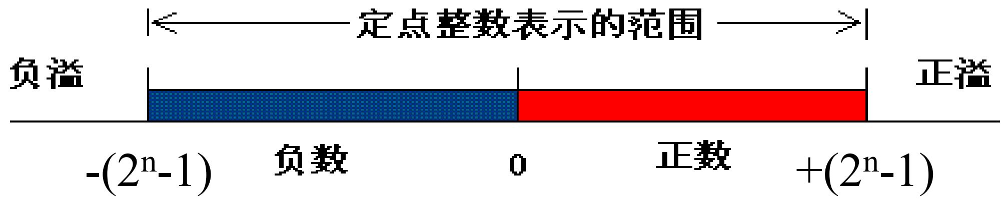
$x - y = x + (-y)$

原码——机器数与真值

真值 带符号的数	机器数 符号数字化的数	
+ 0.1011	0 ‖ 1011	小数点的位置
- 0.1011	1 ‖ 1011	小数点的位置
+ 1100	0 ‖ 1100	小数点的位置
- 1100	1 ‖ 1100	小数点的位置

原码表示法—纯整数

- 定点整数$X_0X_1X_2\ldots X_n$



$$[x]_{原}=\begin{cases} x & 2^n>x\geq 0 \\ 2^n-x & 0\geq x>-2^n \end{cases}$$

符号 $\begin{cases} 0，正数 \\ 1，负数 \end{cases}$

范围一半 $(-2^n, 2^n)$

说明:

- 有正0和负0之分
- 范围 $-(2^n-1)$ ~ $+(2^n-1)$

- 例：$x=+11001110$，$y=-11001110$

$[x]_{原}=011001110$ $[y]_{原}=111001110$

原码表示法—纯小数

- 定点小数$x_0. x_1x_2\ldots x_n$

$$[x]_{原}=\begin{cases} x, & 1>x\geq 0 \\ 1-x, & 0\geq x>-1 \end{cases}$$

符号 $\begin{cases} 0，正 \\ 1，负数 \end{cases}$

范围一半

(-1, 1)

- 有正0和负0之分
- 范围$-(1-2^{-n}) \sim +(1-2^{-n})$

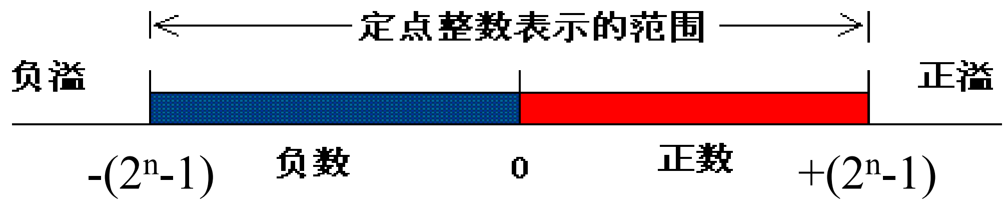
例：$x=+0.11001110$，$y=-0.11001110$

$[x]_{原}=0.11001110$　$[y]_{原}=1.11001110$

原码表示法

- 原码表示法特点：
  - 表示简单，易于同真值之间进行转换，实现乘除运算规则简单
  - 思路：最高符号位表示正负
  - 最直观的数据表示方法（乘除法运算：绝对值）
  - 缺点：进行加减运算十分麻烦

反码表示法



- 定义
  - 正数的表示与原码相同
  - 负数的反码符号位为1，数值位是将原码的数值按位取反
- 电路容易实现，触发器的输出有正负之分
- 方便加减法转换：符号位可参与运算（符号位循环进位至最低位）

扩展：反码加减运算示例

- $3-1 = (0000\ 0011)_{反}+(1111\ 1110)_{反} = (0000\ 0010)_{反} = 2$

连同符号位一起相加，符号位产生的进位循环至低位

- $1-1 = (0000\ 0001)_{反}+(1111\ 1110)_{反} = (1111\ 1111)_{反} = -0$

- 缺点：正负零，循环进位（不常用）

补码表示法

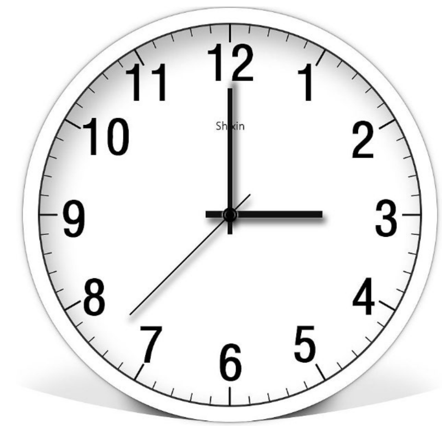
补码表示法（加减法统一）

- 生活例子：现为北京时间下午4点，但钟表显示为 7点。有两种办法校对：

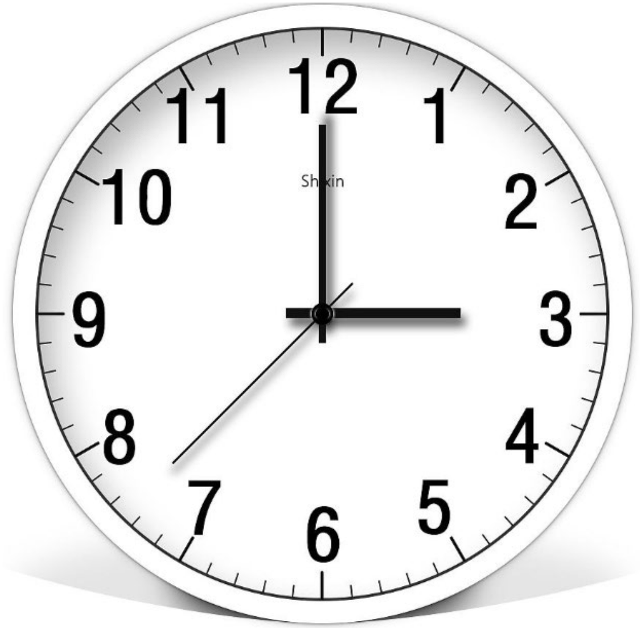
(1) 做减法 7-3 = 4 (逆时针退3格)

(2) 做加法 7+9 = 16 (顺时针进9格)

16 (mod 12) = 16-12 = 4 (以12为模，变成4)



补码表示法—总结



- 时钟：现在3点钟
  - 前拨4小时→11点
  - 后拨8小时→11点
  - 结论：- 4 = + 8 (mod 12)
  - 如果a = b (mod m)，c = d (mod m)
    a ± c = b ± d (mod m)
  - n位定点数？数范围→模数

- 定点小数$x_0.x_1x_2\ldots x_n$，以2为模 [-1,1]
- 定点整数$x_0x_1x_2\ldots x_n$，以2^{n+1}为模 $[-2^n,2^n-1]$
  - 定点小数$x_0.x_1x_2\ldots x_n$ **mod 2**

表示范围

$$[x]_{补}=\begin{cases} x & 1>x\geq 0 \\ 2+x & 0\geq x>-1 \end{cases}$$

符号 $\begin{cases} 0, & 正小数 \\ 1, & 负小数 \end{cases}$

补码加减运算

- 定义：
  - 正数：补码就是其本身
  - 负数：补码是在反码的基础上+1

- $3-1 = (0000\ 0011)_{补}+(1111\ 1111)_{补}= (0000\ 0010)_{补}= 2$
- $1-1 = (0000\ 0001)_{补}+(1111\ 1111)_{补}= (0000\ 0000)_{补}= 0$

连同符号位一起相加，符号位产生的进位自然丢掉

补码转换方法（对2求补）

- 例 x=-1011110，写出其原码与补码
  - 原码为1 10111 10
  - 补码为1 01000 10
- 补码转换性质
  - 按位取反，末位加一（加法器）
  - 最右端往左边扫描，直到第一个1的时候，该位和右边各位保持不变，左边各数值位按位取反

- $[x]_{原}$= 1 11110　　　　补：1 00010

不变，左边数值位取反

补码问题—如何比较大小

补码	原码	二进制	十进制
010101	010101	+10101	+21
101011	110101	-10101	-21
011111	011111	+11111	+31
100001	111111	-11111	-31

补码表示很难直接判断其真值大小

移码表示法（区别IEEE 754）

- 移码表示法（$127/2^n-1$(IEEE 754) v.s. $128/2^n$）
  - 定点整数定义 $[x]_{移}=2^n+x \quad 2^n > x \geq -2^n$
  - $00000000\sim11111111(-2^n\sim2^n-1)$
  - x=+1011111
  - x=-1011111

x=+1011111	x=-1011111
原码为01011111	原码为11011111
补码为01011111	补码为10100001
反码为01011111	反码为10100000
移码为11011111	移码为00100001

  - 特点：移码和补码数值位相同，符号位相反
  - 优点：方便比较大小

原码、反码、补码、移码（重要）

- 原码
  - 符号位加上真值的绝对值
- 反码
  - 正数：其本身
  - 负数：在其原码的基础上，符号位不变，其余各位取反
- 补码
  - 正数：其本身
  - 负数：在反码的基础上+1
- 移码
  - 补码的符号位取反（无论正负）

例—将十进制真值(- 127, - 1,0, + 1, + 127)列表表示成二进制数及原码、反码、补码、移码值

真值x（十进制）	真值x（二进制）	[x]原	[x]反	[x]补	[x]移
-127	-01111111	11111111	10000000	10000001	00000001
-1	-00000001	10000001	11111110	11111111	01111111
		00000000	00000000		
0	00000000			00000000	10000000
		10000000	11111111		
+1	+00000001	00000001	00000001	00000001	10000001
+127	+01111111	01111111	01111111	01111111	11111111

注意：正负0的不同表示

第二章 运算方法和运算器
——运算方法（理论）

- 定点与浮点表示法
- 数的机器码表示（原/反/补/移）
- 定点加法、减法运算
- 溢出检测方法
- 字符表示方法

补码加减法公式

- 加法

整数 $[A]_{补} + [B]_{补} = [A+B]_{补} \pmod{2^{n+1}}$

小数 $[A]_{补} + [B]_{补} = [A+B]_{补} \pmod{2}$

- 减法

$$A-B = A+(-B)$$

整数 $[A-B]_{补} = [A+(-B)]_{补} = [A]_{补} + [-B]_{补}$

小数 $[A-B]_{补} = [A+(-B)]_{补} = [A]_{补} + [-B]_{补}$

连同符号位一起相加，符号位产生的进位自然丢掉

补码加法—证明

- 补码加法

  公式：$[x+y]_{补}=[x]_{补}+[y]_{补} \pmod{2^{n+1}}$

  - 补码表示：定点整数$x_0\, x_1x_2\ldots x_n$

$$[x]_{补}=\begin{cases} x & 2^n>x\geq 0 \\ 2^{n+1}+x & 0\geq x>-2^n \end{cases}$$

符号 $\begin{cases} 0，正整数 \\ 1，负整数 \end{cases}$

$[x]_补 + [y]_补 = [x + y]_补$证明（1）

- 假设 $|x| < 2^{n}$-1, $|y| < 2^{n}$-1, $|x + y| < 2^{n}$-1
- 现分四种情况来证明

$$[x]_补 = \begin{cases} x & 2^n > x \geq 0 \\ 2^{n+1}+x & 0 \geq x > -2^n \end{cases}$$

(1) $x > 0$， $y > 0$，则 $x + y > 0$

$[x]_补=x$, $[y]_补=y$, $[x + y]_补=x+y$

所以等式成立.

(2) $x > 0$， $y < 0$

$[x]_补=x$， $[y]_补=2^{n+1}+y$,

$[x]_补 + [y]_补=x+ 2^{n+1} +y= 2^{n+1} +(x+y)$

a) 当 $x + y>0$时, $[2^{n+1} + (x + y)] \bmod 2^{n+1} = x + y$,

故 $[x]_补 + [y]_补 = x + y = [x + y]_补$

b) 当 $x + y<0$时, $[2^{n+1} + (x + y)] \bmod 2^{n+1} =2^{n+1} + (x + y)$,

故 $[x]_补 + [y]_补 = 2^{n+1} + (x + y) = [x + y]_补$

所以上式成立

$[x]_{补} + [y]_{补} = [x + y]_{补}$证明（2）

(3) $x<0$， $y>0$

这种情况和第2种情况一样,把 x和 y的位置对调即得证。

(4) $x<0$， $y<0$，则 $x + y<0$

相加两数都是负数,则其和也一定是负数。

$\because [x]_{补} = 2^{n+1} + x$， $[y]_{补} = 2^{n+1} + y$

$\therefore [x]_{补} + [y]_{补} = 2^{n+1} + x + 2^{n+1} + y$

$= [2^{n+1} + (2^{n+1} + x + y)] \bmod 2^{n+1}$

$= 2^{n+1} + (x + y)$

$= [x + y]_{补}$

$$[x]_{补} = \begin{cases} x & 2^n > x \geq 0 \\ 2^{n+1} + x & 0 \geq x > -2^n \end{cases}$$

补码加法——例题

- [例] x=+1001，y=+0101，求 x+y=?

补码加法——例题

- [例] x=+1001 , y=+0101 , 求 x+y=?

  解：$[x]_{补} = 01001$, $[y]_{补} = 00101$

$$
\begin{array}{rll}
 & [x]_{补} & 0\ 1\ 0\ 0\ 1 \\
+ & [y]_{补} & 0\ 0\ 1\ 0\ 1 \\
\hline
 & [x+y]_{补} & 0\ 1\ 1\ 1\ 0
\end{array}
$$

x+y = +1110

第二章 运算方法和运算器
——运算方法（理论）

- 定点与浮点表示法
- 数的机器码表示（原/反/补/移）
- 定点加法、减法运算
- 溢出检测方法
- 字符表示方法

溢出示例

[例] x=+1101，y=+1001，求 x+y。

溢出示例

[例] $x=+1101$，$y=+1001$，求 $x+y$。

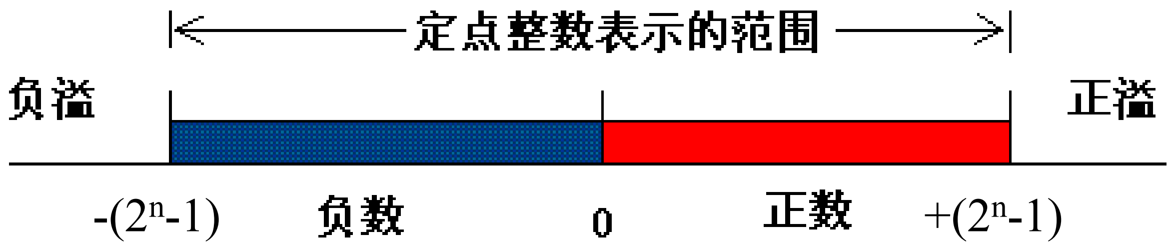
解：$[x]_{补}=01011$，$[y]_{补}=01001$

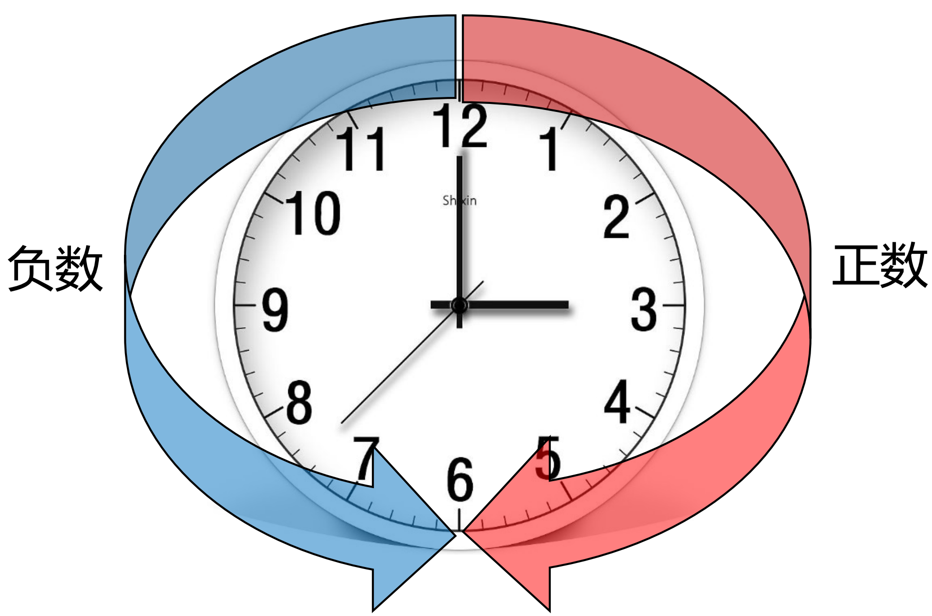
$$
\begin{array}{lll}
 & [x]_{补} & 0\ 1\ 0\ 1\ 1 \\
+ & [x]_{补} & 0\ 1\ 0\ 0\ 1 \\
\hline
 & [x+y]_{补} & 1\ 0\ 1\ 0\ 0
\end{array}
$$

两个正数相加的结果成为负数，表示正溢。

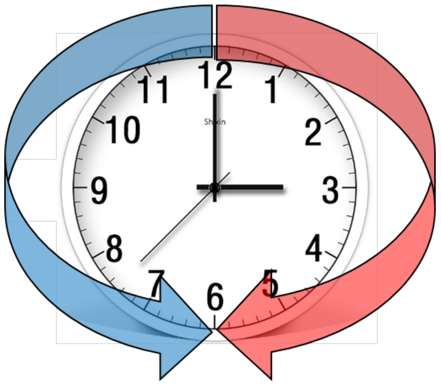
溢出概念

溢出的概念





溢出概念



- 溢出的概念
  - 可能产生溢出的情况
    - 两正数加，变负数，正溢（大于机器所能表示的最大数）
    - 两负数加，变正数，负溢（小于机器所能表示的最小数）
  - 注意
    - 仅在有符号数运算可能产生溢出
    - 加法溢出需要两个数符号相同

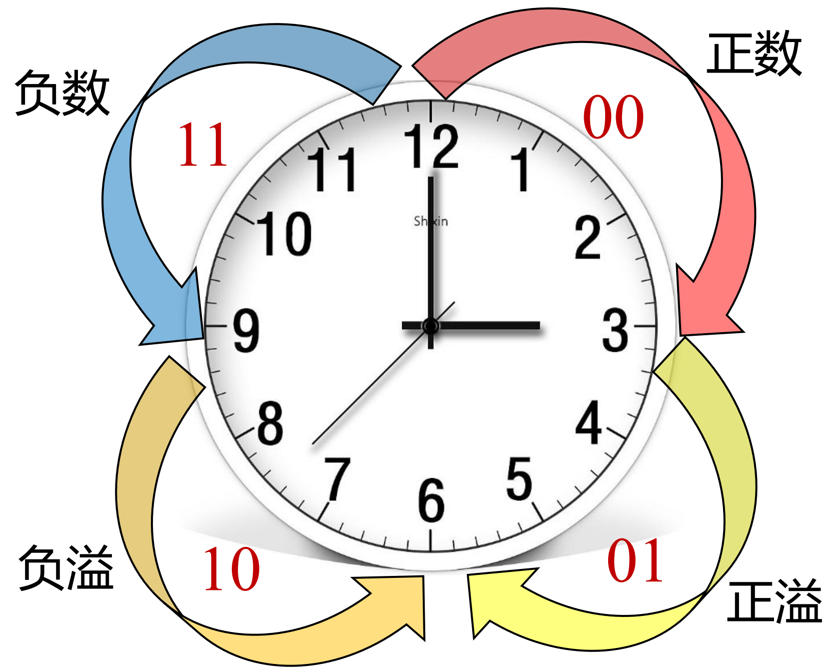
双符号位溢出检测——手算过程

1、双符号位法

（变形补码—扩大一倍范围）

$[x]_{补} = 2^{n+2} + x \pmod{2^{n+2}}$

S_{f1}	S_{f2}	
0	0	正确（正数）
0	1	正溢
1	0	负溢
1	1	正确（负数）



S_{f1} 表示正确的符号，逻辑表达式为$V=S_{f1} \oplus S_{f2}$，可以用异或门来实现

双符号位检测——例题1

[例] x=+1100，y=+1000，求 x+y 。

双符号位检测——例题1

[例] x=+1100 , y=+1000 , 求 x+y 。

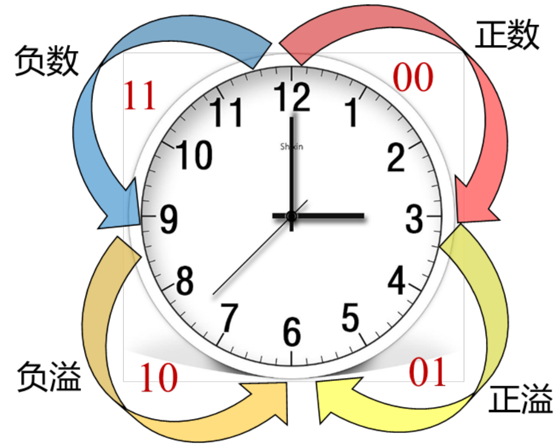
解： $[x]_{补} = 001100$, $[y]_{补} = 001000$

	$[x]_{补}$	0 0 1 1 0 0
+	$[y]_{补}$	0 0 1 0 0 0
	$[x+y]_{补}$	0 1 0 1 0 0 (表示正溢)

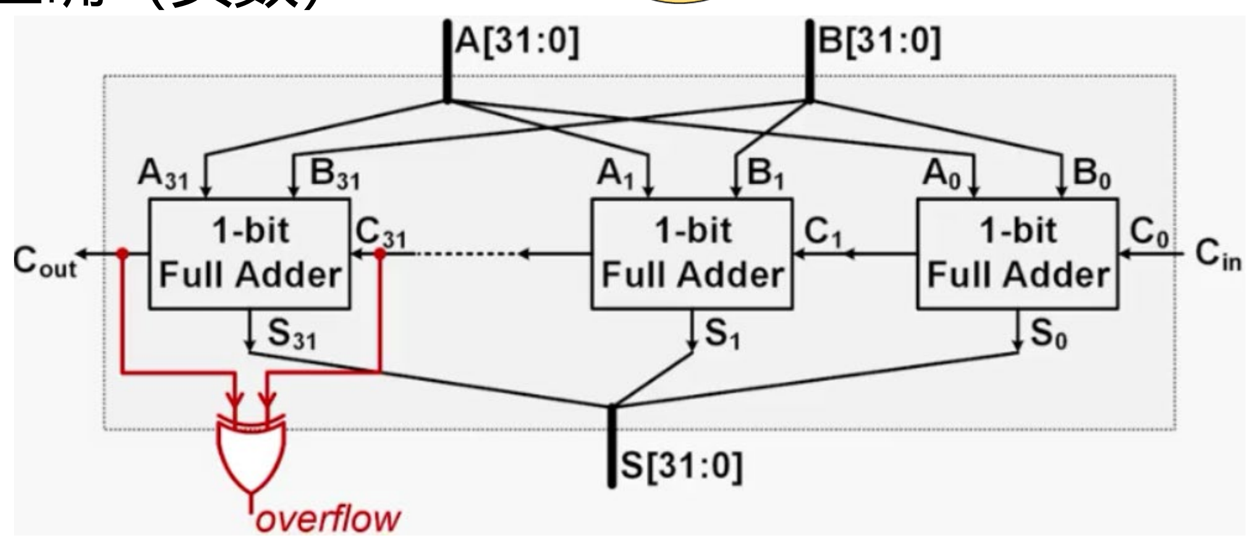
单符号位溢出检测——硬件实现

- Cf为符号位产生的进位，C0为最高有效位产生
- C_f　C_0

C_f	C_0	
0	0	正确（正数）
0	1	正溢
1	0	负溢
1	1	正确（负数）



$$\boxed{V=C_f \oplus C_0}$$



第二章 运算方法和运算器
——运算方法（理论）

- 定点与浮点表示法
- 数的机器码表示（原/反/补/移）
- 定点加法、减法运算
- 溢出检测方法
- 字符表示方法

字符和字符串的表示方法

- 符号数据：字符信息用数据表示，如ASCII等
- 字符表示（ASCII码）
  - 用一个字节来表示，低7位用来编码，最高位为校验位
  - 扩展表：8位均进行编码，无校验位

ASCII扩展表

(American Standard Code for Information Interchange 美国标准信息交换代码)

高四位 / 低四位		1000		1001		1010		1011		1100		1101		1110		1111	
		8		9		A		B		C		D		E		F	
		十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符
0000	0	128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
0001	1	129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
0010	2	130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
0011	3	131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
0100	4	132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
0101	5	133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
0110	6	134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
0111	7	135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
1000	8	136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
1001	9	137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	•
1010	A	138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
1011	B	139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
1100	C	140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
1101	D	141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
1110	E	142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	∈	254	■
1111	F	143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	ÿ

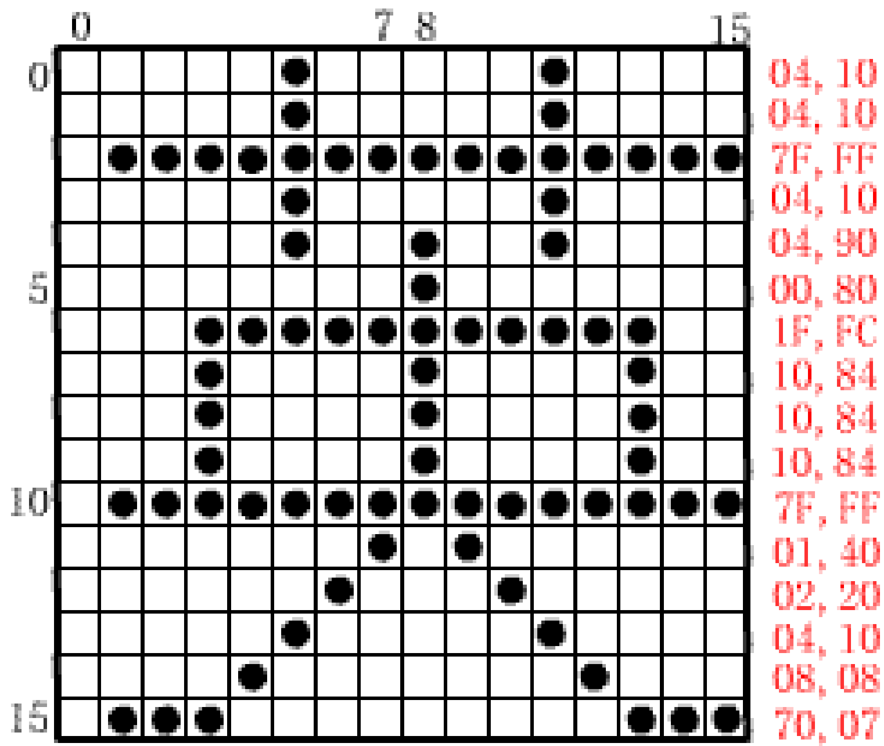
汉字的表示方法

- 汉字的表示方法
  - 输入码
    - 拼音
    - 五笔
  - 国标码（初代）

  - 汉字机内码：汉字信息的存储，交换和检索的机内代码，两个字节组成，每个字节高位都为1

汉字的存放

- 汉字字形码：汉字字形存储
  - 点阵



校验码

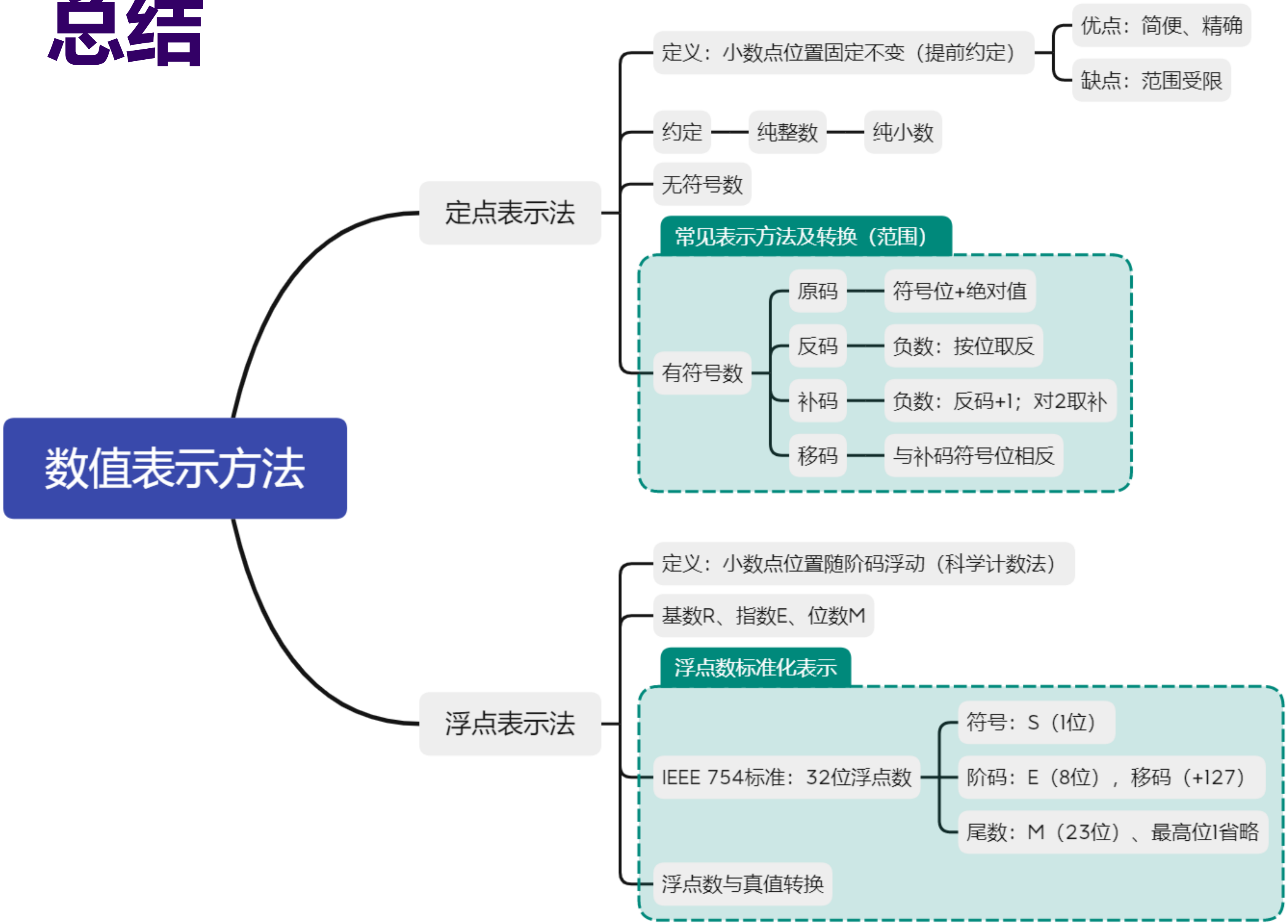
- 校验码（只介绍奇偶校验码）
  - 引入：信息传输和处理过程中受到干扰和故障，易出错
  - 解决方法：是在有效信息中加入一些冗余信息（校验位）
  - 设 $x=(x_0 x_1 \ldots x_{n-1})$是一个n位字，则奇校验定义为
    - $\overline{C}=x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$只有当x中包含有奇数个1时，才使$\overline{C}=1$，即$C=0$。
  - 接收端进行检查，是否满足奇偶校验规则
  - **只能检查出奇数位错；不能纠正错误**

原始码	奇校验	偶校验
	奇数个1	偶数个1
1011000	10110000	10110001
1010000	10100001	10100000
0011010	00110100	00110101
0001000	00010000	00010001

奇偶校验码——实例

- 奇偶校验码——ASCII码实例
  - 大写字母A，十六进制0x41
    - 二进制：0100 0001
    - 奇校验：1100 0001 （奇数个1）
    - 偶校验：0100 0001 （偶数个1）
  - 接收端（奇校验，奇数个1—校验通过）
    - 正确码流：1100 0001
    - 错1位：1110 0001 （偶数个1，传输出错）
    - 错2位：1110 0101 （无法检出错误）

总结


数值表示方法
定点表示法
定义：小数点位置固定不变（提前约定）
优点：简便、精确
缺点：范围受限
约定
纯整数
纯小数
无符号数
常见表示方法及转换（范围）
有符号数
原码
符号位+绝对值
反码
负数：按位取反
补码
负数：反码+1；对2取补
移码
与补码符号位相反
浮点表示法
定义：小数点位置随阶码浮动（科学计数法）
基数R、指数E、位数M
浮点数标准化表示
IEEE 754标准：32位浮点数
符号：S（1位）
阶码：E（8位），移码（+127）
尾数：M（23位）、最高位1省略
浮点数与真值转换

总结

- 加减法运算
  - 补码加/减法正确性证明
    - 根据补码表示定义式，数学公式证明
    - x,y><0分四类情况讨论
  - 加减法手算
    - 补码
    - 原码
  - 溢出判断
    - 条件
      - 有符号数
      - 分类：正溢、负溢
    - 补码溢出判定方法
      - 双符号位判定
        - 扩展补码符号位：时钟表示
        - 手算判定流程
      - 单符号位检测
        - 硬件实现
        - 最高位进位输入输出异或

- 字符表示与校验
  - 英文字符
    - ASCII码（8位）
  - 中文字符
    - 输入码
    - 字形码
    - 机内码
  - 校验方法
    - 奇偶校验