



本章要点:

- 数制表示法
- 数制之间的转化
- 二进制数运算
- 有符号数的表示
- ASCII码



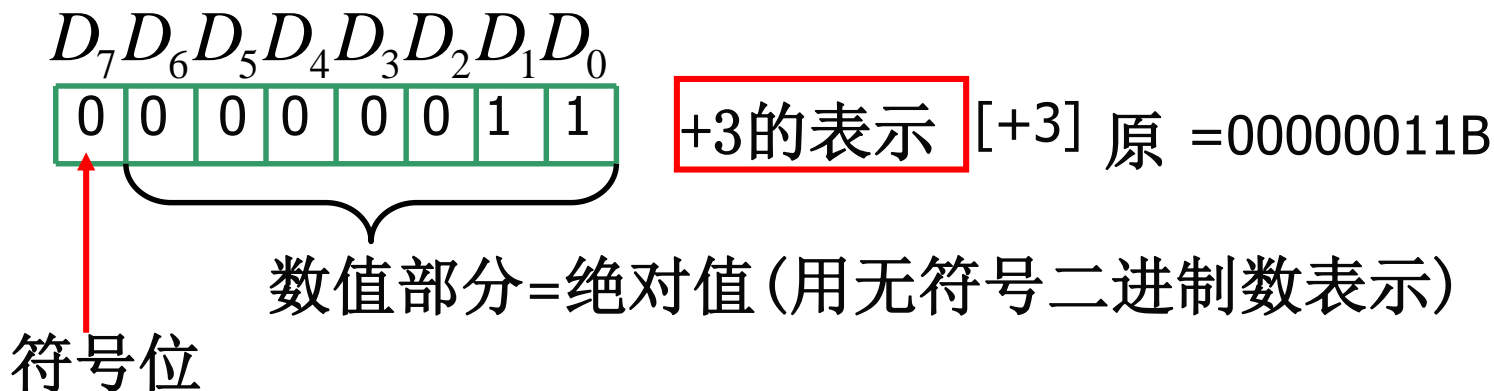
第1章 数制与码制

1. 有符号二进制数的表示方法及溢出问题

1.1 原码表示法

如果正数的符号位用0表示，负数的符号位用1表示，绝对值的编码规则与前面讲的无符号数编码规则相同，这种表示方法称为**原码表示法**。

一个数X的原码记为： $[X]_{\text{原}}$





第1章 数制与码制

1. 有符号二进制数的表示方法及溢出问题

1	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

-3的表示

$[-3]_{\text{原}} = 10000011\text{B}$

$[+0]_{\text{原}} = 00000000\text{B}$

$[-0]_{\text{原}} = 10000000\text{B}$

数0的两种表示方法

对8位有符号二进制数用原码表示的范围：

正数从00000000 \sim 01111111, $+0 \sim +127$

负数从10000000 \sim 11111111, $-0 \sim -127$

1. 有符号二进制数的表示方法及溢出问题

有符号二进制数用原码表示的优缺点：

优点：表示简单，易于理解，真值转换方便。

缺点：+、-运算麻烦。因为它仅仅是将其值的符号用一位二进制数表示，因而它的原码数的+、-运算完全同笔算。如两个正数相减，计算机首先要判断被减数的绝对值与减数的绝对值的大小，然后决定是颠倒过来相减，还是直接相减。最后在结果的前面加上正确的正负号。所以，势必增加运行时间，降低速度，使运算器的逻辑复杂化。为了改进它，引进了**补码**的概念。

1. 有符号二进制数的表示方法及溢出问题

1.2 补码表示法

(1) 补码的概念

一个数X的补码记为 $[X]_{\text{补}}$ ，补码可定义为：

$$[x]_{\text{补}} = \begin{cases} x & \text{当 } 0 \leq x < 2^{n-1} \\ 2^n + x & \text{当 } -2^{n-1} \leq x < 0 \end{cases} \pmod{2^n}$$

从定义可见，正数的补码=原码，即 $[x]_{\text{补}} = [x]_{\text{原}}$ ，

所以，只有负数求补的问题。

1. 有符号二进制数的表示方法及溢出问题

(2) 一个数的补码的求法

■ 根据定义求补码

$$[x]_{\text{补}} = 2^n + x = 2^n - |x|, \quad x < 0$$

即负数 x 的补码等于模 2^n 加上其真值（或减去其真值的绝对值）。

如： $x = -101\ 0111\text{B}$ （-87）， $n=8$ ，则

$$\begin{aligned} [X]_{\text{补}} &= 2^8 + (-1010111\text{B}) \\ &= 1\ 0000\ 0000\text{B} - 101\ 0111\text{B} \\ &= 1010\ 1001\text{B} \quad (\text{mod } 2^n) \end{aligned}$$

这种方法因要做一次减法，很不方便。

1. 有符号二进制数的表示方法及溢出问题

利用原码求补码

一个负数的补码等于其原码除符号位保持不变外，其余各位按位取反，再在最低位加1。

(-87)

如: $x = -101\ 0111B$ $[X]_{\text{原}} = 1101\ 0111B$

$[X]_{\text{补}} = 1010\ 1000B + 1 = 1010\ 1001B$

值的注意的是: 0的补码只有唯一的形式，符号位和数值位均为0。无正负0之分。

1. 有符号二进制数的表示方法及溢出问题

(3) 数的补码表示转换为原码表示

一个用补码表示的负数，如将 $[X]_{\text{补}}$ 再求一次补，即将 $[X]_{\text{补}}$ 除符号位外取反加1，就可得到 $[X]_{\text{原}}$ ，用下式表示：

$$[[X]_{\text{补}}]_{\text{补}} = [X]_{\text{原}}$$

如： $[X]_{\text{补}} = 1010\ 1001\text{B}$

$$[[X]_{\text{补}}]_{\text{补}} = 1101\ 0111\text{B} = [X]_{\text{原}}$$

1. 有符号二进制数的表示方法及溢出问题

(4) 补码的运算规则

第一个公式：两个n位二进制数之和的补码等于这两数补码之和，即：

$$[X+Y]_{\text{补}} = [X]_{\text{补}} + [Y]_{\text{补}}$$

[illegible]

1. 有符号二进制数的表示方法及溢出问题

第二个公式：两个n位二进制数之差的补码等于这两数补码之差，即：

$$[X-Y]_{\text{补}} = [X]_{\text{补}} - [Y]_{\text{补}}$$

例: $(+33) - (-15)$

0	0	1	0	0	0	0	1	B	$[+33]_{\text{补}}$	
-	1	1	1	1	0	0	0	1	B $[-15]_{\text{补}}$	
<hr/>										
借位, 丢掉	→	[1]	0	0	1	1	0	0	0	B $[+48]_{\text{补}}$



1. 有符号二进制数的表示方法及溢出问题

► 第三个公式：补码减法运算时，也可以利用加法基本公式，即：

$$[X-Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}} \quad (\text{mod } 2^n)$$

因为： $X-Y = X+(-Y)$

所以： $[X-Y]_{\text{补}} = [X+(-Y)]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$

一般称已知 $[Y]_{\text{补}}$ ，求得 $[-Y]_{\text{补}}$ 的过程叫变补或求负。



第1章 数制与码制

1. 有符号二进制数的表示方法及溢出问题

变补或求负是一种很有用的运算。求法：

若已知 $[Y]_{\text{补}} = Y_{n-1}Y_{n-2} \dots Y_1Y_0$ ，则对 $[Y]_{\text{补}}$ 的**每一位**（包括符号位）都按位取反，然后再加1，结果即 $[-Y]_{\text{补}}$ 。

例： $(+33) - (+15)$

$$\begin{array}{r} 00100001\text{B} \quad [+33]_{\text{补}} \\ - 00001111\text{B} \quad [+15]_{\text{补}} \\ \hline 00010010\text{B} \quad [+18]_{\text{补}} \end{array}$$

（公式二）

$$\begin{array}{r} 00100001\text{B} \quad [+33]_{\text{补}} \\ + 11110001\text{B} \quad [-15]_{\text{补}} \\ \hline [1] 00010010\text{B} \quad [+18]_{\text{补}} \end{array}$$

借位，丢掉

（公式三）

1. 有符号二进制数的表示方法及溢出问题

有符号数运算的溢出问题

如果计算机的字长为n位，n位二进制数的最高位为符号位，其余n-1位为数值位，采用补码表示法时，可表示的数X的范围为：

$$-2^{n-1} \leq X \leq 2^{n-1} - 1$$

当n=8时，可表示的有符号数的范围为：

$$-128 \quad \sim \quad +127$$

当n=16时，可表示的有符号数的范围为：

$$-32768 \quad \sim \quad +32767$$



1. 有符号二进制数的表示方法及溢出问题

两个有符号数进行加减运算时，如果运算结果超出可表示的有符号数的范围时，就会发生溢出，使计算结果出错。

很显然，溢出只能出现在两个同号数相加或两个异号数相减的情况下。



第1章 数制与码制

1. 有符号二进制数的表示方法及溢出问题

例： $(+72) + (+98) = +170 > +127$ 溢出

$$\begin{array}{r} 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ \text{B} \quad [+72]_{\text{补}} \\ +\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ \text{B} \quad [+98]_{\text{补}} \\ \hline 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ \text{B} \quad [-86]_{\text{补}} \end{array}$$

↑ ↑
有进位 $C_p = 1$
无进位 $C_s = 0$

溢出，结果出错（正溢出）

$$C_s \vee C_p = 1$$



第1章 数制与码制

1. 有符号二进制数的表示方法及溢出问题

例: $(-83) + (-80) = -163 < -128$ 溢出

$$\begin{array}{r} 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ \text{B} \quad [-83]_{\text{补}} \\ +\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ \text{B} \quad [-80]_{\text{补}} \\ \hline [1]0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ \text{B} \quad [+93]_{\text{补}} \end{array}$$

Diagram illustrating the addition of two 8-bit two's complement numbers. The first number is 10101101_{B} (labeled $[-83]_{\text{补}}$) and the second is 10110000_{B} (labeled $[-80]_{\text{补}}$). The result is $[1]01011101_{\text{B}}$ (labeled $[+93]_{\text{补}}$). The carry-out of the 8th bit is 1, and the carry-in of the 9th bit is 1. The carry-out of the 9th bit is 0.

无进位 $C_p = 0$
有进位 $C_s = 1$

溢出, 结果出错 (负溢出)

$$C_s \vee C_p = 1$$



1. 有符号二进制数的表示方法及溢出问题

结论：对于加法运算

- (1) 如果次高位有进位而最高位无进位，则结果溢出；
- (2) 如果次高位无进位而最高位有进位，则结果溢出。

这两种情况分别是：

- (1) 两负数相加，结果超出范围，形式上变为正数；
- (2) 两正数相加，结果超出范围，形式上变为负数。



1. 有符号二进制数的表示方法及溢出问题

同理，可得出结论：对于减法运算

- (1) 如果次高位有借位而最高位无借位，则结果溢出；
- (2) 如果次高位无借位而最高位有借位，则结果溢出。

2. 二进制编码的十进制数（BCD编码）

2.1 8421BCD码

BCD码是用四位二进制数表示1位0 —9的十进制数，而4位二进制数码有16种组合，原则上可任选10种作为代码，但为便于记忆和比较直观，最常用的是8421BCD码，8、4、2、1分别是4位二进制数的位权值。



第1章 数制与码制

2. 二进制编码的十进制数（BCD编码）

十进制数	8421BCD码
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

2. 二进制编码的十进制数（BCD编码）

如：十进制数和BCD码相互转换

75.4 \longrightarrow BCD码

$$75.4 = (0111 \mid 0101 \mid .0100)_{BCD}$$

BCD码10000101.0101 \longrightarrow 十进制数

$$(1000 \mid 0101 \mid .0101)_{BCD} = 85.5$$

同一个8位二进制代码表示的数，当认为它表示的是二进制数和认为它表示的是二进制编码的十进制数，数值是不相同的。如：

$$(00011000)_2 = 24$$

$$(0001 \mid 1000)_{BCD} = 18$$



第1章 数制与码制

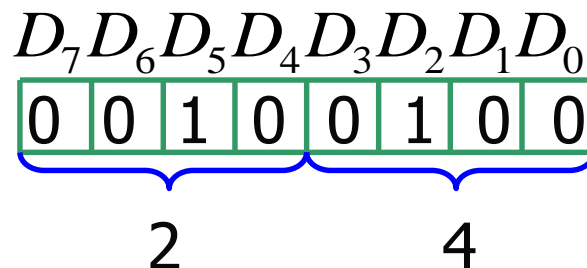
2. 二进制编码的十进制数（BCD编码）

在计算机中，BCD码有两种基本格式

- a. 组合式BCD码
- b. 分离式BCD码

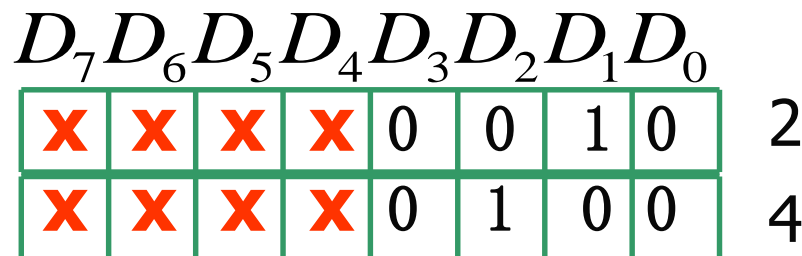
a. 组合式BCD码

两位十进制存放在一个字节中，如数**24**的存放格式：



b. 分离式BCD码

每位数存放在8位字节的低4位，高4位的内容与数值无关。如数24的存放格式：





2. 二进制编码的十进制数（BCD编码）

2.2 BCD码的加减法运算

由于BCD编码是将每个十进制数用一组4位二进制数表示，若将这种BCD编码直接交计算机运算，计算机总是把它按二进制数处理，所以结果可能出错。

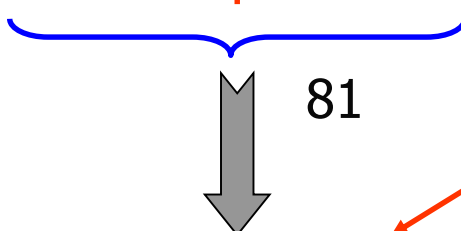


第1章 数制与码制

2. 二进制编码的十进制数（BCD编码）

如：38+49=87

0	0	1	1	1	0	0	0	$_{BCD}$	38
0	1	0	0	1	0	0	1	$_{BCD}$	49
<hr/>									
1	0	0	0	0	0	0	1		87

 81

显然，结果出错。

出错原因：十进制相加应逢十进一，但计算机按二进制运算，每四位一组，低四位向高四位进位相当十六进制运算，“逢十六进一”。所以当结果超过9时将比正确值少6。

解决办法：加六修正



2. 二进制编码的十进制数（BCD编码）

加六修正规则：

- （1）如果两个BCD码位相加没有进位，并且结果 ≤ 9 ，则该位不需修正。
- （2）如果两个BCD码位相加有进位，或者其结果 ≥ 10 ，该位进行加六修正。
- （3）低位修正结果使高位 > 9 时，高位进行加六修正。



第1章 数制与码制

2. 二进制编码的十进制数（BCD编码）

例：94+7=101

1 0 0 1	0 1 0 0	94
+ 0 0 0 0	0 1 1 1	7
<hr/>		
1 0 0 1	1 0 1 1	低4位满足法则1
+ 0 0 0 0	0 1 1 0	加六修正
<hr/>		
1 0 1 0	0 0 0 1	高4位满足法则3
+ 0 1 1 0	0 0 0 0	加六修正
<hr/>		
[1]0 0 0 0	0 0 0 1	101结果正确



2. 二进制编码的十进制数（BCD编码）

减六修正规则：

- （1）如果两个BCD码位相减没有借位，则该位不需修正。
- （2）如果两个BCD码位相减有借位，则该位进行减六修正。



第1章 数制与码制

2.3 二进制编码的十进制数（BCD编码）

例：50－29＝21

0	1	0	1	0	0	0	0	50
—	0	0	1	0	1	0	0	29
<hr/>								
0	0	1	0	0	1	1	1	低码位有借位
—	0	0	0	0	0	1	1	减六修正
<hr/>								
0	0	1	0	0	0	0	1	21结果正确



3. ASCII字符代码

ASCII —— 美国国家信息标准交换码

ASCII → 用**7**位二进制代码对任一字符编码，包括：

共128个 {
32个通用控制符
0—9 10个数字
52个英文大小写字母
34个专用符号

要求掌握常用字符的**ASCII**码：

0~9, A~Z, a~z, 空格, 回车, 换行, Esc

第1章 数制与码制



作业