第 2 章　基 础 知 识

2.1　常用数制及其相互转换

计算机内以二进制方式动作，人类熟悉的是十进制数，并不习惯使用二进制记数，所以在使用中根据需要，计算机中的编码和数经常用十进制、二进制、十进制和八进制形式表示。

为叙述方便，本书中若没有特别标注或说明，数均为十进制记数形式。

2.1.1　十进位记数制

十进位记数制(简称十进制)使用 10 个数码(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)表示数，低位向高位进位的规则是“逢 10 进 1”，或高位向低位进位的规则是“借 1 当 10”。

十进制数通常在数值后用下标 10 标识，或加 D(Decimal)或 d 来表示。

例如，十进制数 365，可以写成$(365)_{10}$，或写成 365D，365d。

2.1.2　二进位、八进位及十六进位记数制

1. 二进制数

二进位记数制(简称二进制)的基数为 2，使用 2 个数码 0 和 1 表示数。低位向高位进位的规则是“逢 2 进 1”，或高位向低位进位的规则是“借 1 当 2”。

二进制数通常在数值后用下标 2，或标以字母 B(Binary)或 b 以示区别。例如，二进制数 101101101，可记作$(101101101)_2$，或 101101101B, 101101101b。

二进制数所表示的数值可用下列权位展开式计算：

$$d = \sum b_{\mathrm{i}} \times 2^{i}$$

式中，b_i 取 0 或 1，i 在小数点左边(整数部分)自右至左依次取：0, 1, 2, …，小数点右边(小数部分)自左至右依次是：−1, −2, −3, …。

例如，$(1011.01)_2$ 所表示的数为：$1\times2^3+0\times2^2+1\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}=(11.25)_{10}$。

二进制数的四则运算除了使用 2 进位规则外，其运算法则类似十进制数运算。

1) 加法运算规则：

① 0 + 0＝0；② 0 + 1＝1；③ 1 + 0＝1；④ 1+ 1＝0(高位进 1)。

例 2.1　求二进制数 1101 0011 与 1001 0110 的和。

解：		1	1	0	1	0	0	1	1	(被加数)
+		1	0	0	1	0	1	1	0	(加数)
	1	0	1	1	0	1	0	0	1	(和)

所以，$(1101\,0011)_2 + (1001\,0110)_2 = (1\,0110\,1000)_2$。

2) 减法运算规则：

① 0 − 0＝0；② 0 − 1＝1(高位借 1)；③ 1 − 0＝1；④ 1 − 1＝0。

例 2.2　求二进制数 1101 1010 与 1010 1101 的差。

解：

$$\begin{array}{rcccccccccl}
 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & & \text{(被减数)} \\
+ & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & & \text{(减数)} \\
\hline
 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & & \text{(差)}
\end{array}$$

所以，$(1101\ 1010)_2 + (1010\ 1101)_2 = (0010\ 1101)_2$。

3) 乘法运算规则：

① $0 \times 0 = 0$；② $0 \times 1 = 0$；③ $1 \times 0 = 0$；④ $1 \times 1 = 1$

例 2.3 求二进制数 1101 与 1011 的积。

解：

$$\begin{array}{cccccccccl}
 & & & & & 1 & 1 & 0 & 1 & \text{(被乘数)} \\
 & & & & \times & 1 & 0 & 1 & 1 & \text{(乘数)} \\
\hline
 & & & & & 1 & 1 & 0 & 1 & \\
 & & & & 1 & 1 & 0 & 1 & & \\
 & & & 0 & 0 & 0 & 0 & & & \\
+ & & 1 & 1 & 0 & 1 & & & & \\
\hline
 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 &
\end{array}$$

所以，$(1101)_2 \times (1011)_2 = (1000\ 1111)_2$。

4) 除法运算规则：

① $0 \div 1 = 0$；② $1 \div 1 = 1$

例 2.4 求二进制数 11 1110 除以 1011 的商。

解：

$$\begin{array}{lccccccccccl}
 & & & & & & & & 1 & 0 & 1 & \text{(商)} \\
\text{(除数)} & 1 & 0 & 1 & 1\,) & 1 & 1 & 1 & 1 & 1 & 0 & \text{(被除数)} \\
 & & & & - & 1 & 0 & 1 & 1 & & & \\
\hline
 & & & & & 0 & 1 & 0 & 0 & 1 & 0 & \\
 & & & & & & - & 1 & 0 & 1 & 1 & \\
\hline
 & & & & & & & 0 & 1 & 1 & 1 & \text{(余数)}
\end{array}$$

所以，$(11\ 1110)_2 \div (1011)_2$ 的商为 $(101)_2$，余数为 $(0111)_2$。

2. 八进制数

八进位记数制(简称八进制)的基数为 8，使用 0, 1, 2, 3, 4, 5, 6, 7 共 8 个数码。低位向高位进位的规则是“逢 8 进 1”，或高位给低位借位的规则是“借 1 当 8”。

八进制数通常在数值后用下标 8，或标以字母 O(Octal)或 o 以示区别。

例如，八进制数 555 可以记作 $(555)_8$，或记作 555O，555o。

字母 O 易与数字 0 混淆，应用中多采用在数值后加 Q 或 q 来表示八进制数。

八进制所表示的数值可用下列权位展开式计算：

$$d = \sum q_{\mathrm{i}} \times 8^{i}$$

式中，q_i 取 0～7，I 在小数点左边(整数部分)自右至左依次是：0, 1, 2, …，小数点右边(小数部分)自左至右依次是：−1, −2, −3, …。

例如，$(13.2)_8$ 所表示的数值为：$1\times8^1+3\times8^0+2\times8^{-1}=(11.25)_{10}$。

一个数的二进制表示形式与八进制表示形式有着这样的对应关系：小数点左边(整数部分)自右至左，每 3 个二进制位对应一个八进制位，最左的数位不够三位用 0 补；小数点右边(小数部分)自左至右，每 3 个二进制位对应一个八进制位，最右边的数位不够三位用 0 补。反之亦然。三位二进制码与八进制数码间的对应关系如见表 2.1。

表 2.1 三位二进制数码与八进制数码对应关系表

三位二进制码	000	001	010	011	100	101	110	111
八进制数码	0	1	2	3	4	5	6	7

例如，在将$(1011.01)_2$转换成八进制数时，应把它理解成$(001\ 011.010)_2$，为$(13.2)_8$。又如，在把$(15.4)_8$转换成二进制数时，直接按顺序将一个八进制位展开成三位二进制位得到$(001\ 101.100)_2$，即$(1101.1)_2$。

3. 十六进制数

十六进位记数制(简称十六进制)的基数为 16，使用 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F 共 16 个数码，其中 A～F 也可用相应的小写字母 a～f，分别与 10, 11, 12, 13, 14, 15 这 6 个数值对应。低位向高位进位的规则是“逢 16 进 1”，或高位给低位借位的规则是“借 1 当 16”。

通常在十六进制数后用下标 16，或标以字母 H(Hexadecimal)或 h 来标识。

例如，十六进制数 16D 可以写成$(16D)_{16}$，或写成 16DH，16Dh。

十六进制数所表示的数值可用下列权位展开式计算：

$$d = \sum h_i \times 16^i$$

式中，h_i取 0～15，i在小数点左边(整数部分)自右至左依次是：0, 1, 2, …，在小数点右边(小数部分)自左至右依次是：−1, −2, −3, …。

例如，$(B.4)_{16}$所表示的数值为：$11\times16^0+4\times16^{-1}=(11.25)_{10}$。

一个数的二进制表示形式与十六进制表示形式有着这样的对应关系：小数点左边(整数部分)自右至左，每 4 个二进制位对应一个十六进制位，最左的数位不够四位用 0 补；小数点右边(小数部分)自左至右，每 4 个二进制位对应一个十六进制位，最右边的数位不够四位用 0 补。反之亦然。四位二进制数码与十六进制数码之间的对应关系见表 2.2。

表 2.2　四位二进制数码与十六进制数码对应表

四位二进制数码	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
十六进制数码	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

例如，在将$(1011.01)_2$转换成十六进制数时，应把它看成$(1011.0100)_2$，为$(B.4)_{16}$。又如，在把$(15.4)_6$转换成二进制数时，直接按顺序将一个十六进制位展开成四位二进制位得到$(0001\ 0101.0100)_2$，即$(10101.01)_2$。

2.1.3　数制间的转换

对于一个给定的数，可以用不同的进位记数制来表示，例如，可以验算，$-(21.25)_{10}$、$-(10101.01)_2$、$-(25.2)_8$、$-(15.4)$表示同一个数，它们除了外在表现形式不一样，数的本质是一样的。我们可以根据需要使用不同的数制来记数。

1. 二(八、十六)进制数转换为十进制数

对于二进制、八进制和十六进制等表示形式的数，直接用权位展开式计算即可得出其对应的十进制的值。例如：

$(10101.1011)_2=1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}=21.6875$

$(1207.3)_8=1\times8^3+2\times8^2+0\times8^1+7\times8^0+3\times8^{-1}=647.375$

$(1B2E.D)_{16}=1\times16^3+11\times16^2+2\times16^1+14\times16^0+13\times16^{-1}=6958.8125$

2. 十进制数转换成二(八、十六)进制数

十进制转换成二进制、八进制和十六进制，通常要区分数的整数部分和小数部分，并按除基取余数部分和乘基取整数部分两种不同的方法来完成。

(1) 十进制整数转换为二进制整数

十进制整数转换为二进制整数采用“除 2 取余，逆序排列”法。具体算法是：用 2 去除十进制整数，可以得到一个商和余数；再用 2 去除商，又得到一个商和余数，如此反复，直到商为零时为止，然后把先得到的余数作为二进制数的低位有效位，后得到的余数作为二进制数的高位有效位，依次排列起来。

例 2.5 将十进制数 105 转换成二进制表示形式。计算过程如下

2	105	余数为 1，即 $b_0=1$
2	52	余数为 0，即 $b_1=0$
2	26	余数为 0，即 $b_2=0$
2	13	余数为 1，即 $b_3=1$
2	6	余数为 0，即 $b_4=0$
2	3	余数为 1，即 $b_5=1$
2	1	余数为 1，即 $b_6=1$
	0	商为 0，终止

即转换后的二进制整数为：$(b_6b_5b_4b_3b_2b_1b_0)_2=(1101001)_2$。

(2) 十进制小数转换为二进制小数

十进制小数转换成二进制小数采用“乘 2 取整，顺序排列”法。具体算法是：用 2 乘十进制小数，可以得到乘积，将乘积的整数部分取出，再用 2 乘余下的小数部分，又得到一个乘积，再将乘积的整数部分取出，如此反复进行，直到乘积中的小数部分为零，或者达到所要求的精度为止。 然后把取出的整数部分按顺序排列起来，先取的整数作为二进制小数的高位有效位，后取的整数作为低位有效位。

例 2.6 将十进制小数 0.6875 转换成二进制小数。其计算过程如下：

	0.6875	
×	2	
	1.3750	整数部分为 1，即 $b_{-1}=1$
	0.3750	
×	2	
	0.7500	整数部分为 0，即 $b_{-2}=0$
	0.7500	
×	2	
	1.5000	整数部分为 1，即 $b_{-3}=1$
	0.5	
×	2	
	1.0000	整数部分为 1，即 $b_{-4}=1$
	0.0000	小数部分为 0，终止

即转换后的二进制小数为$(0.b_{-1}b_{-2}b_{-3}b_{-4})_2=(0.1011)_2$。

并不是所有的十进制小数都能精确地表示成二进制形式，这时要根据要求取有限位二进制数来近似表示该小数。也就是说，若小数转换不能算尽，那么只算到一定精度的位数为止，

当然这样做会产生一定的误差。(这也是计算机计算时产生的误差原因之一)

例 2.7　将十进制小数 0.32 转换成二进制小数的计算过程如下：

$$\begin{array}{rl}
32 & \\
\times\quad 2 & \\ \hline
\mathbf{0}.64 & \text{整数部分为 0，即 } b_{-1}=0 \\
0.64 & \\
\times\quad 2 & \\ \hline
\mathbf{1}.28 & \text{整数部分为 1，即 } b_{-2}=1 \\
0.28 & \\
\times\quad 2 & \\ \hline
\mathbf{0}.56 & \text{整数部分为 0，即 } b_{-3}=0 \\
0.56 & \\
\times\quad 2 & \\ \hline
\mathbf{1}.12 & \text{整数部分为 1，即 } b_{-4}=1 \\
0.12 & \\
\times\quad 2 & \\ \hline
\mathbf{0}.24 & \text{整数部分为 0，即 } b_{-5}=0
\end{array}
\qquad
\begin{array}{rl}
\times\quad 24 & \\
\times\quad 2 & \\ \hline
\mathbf{0}.48 & \text{整数部分为 0，即 } b_{-6}=0 \\
0.48 & \\
\times\quad 2 & \\ \hline
\mathbf{0}.96 & \text{整数部分为 0，即 } b_{-7}=0 \\
0.96 & \\
\times\quad 2 & \\ \hline
\mathbf{1}.92 & \text{整数部分为 1，即 } b_{-8}=1 \\
0.92 & \\
\times\quad 2 & \\ \hline
\mathbf{1}.84 & \text{整数部分为 1，即 } b_{-9}=1 \\
\cdots &
\end{array}$$

在转换过程中，小数部分始终不为 0，即 $0.32=(0.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}b_{-7}b_{-8}b_{-9}\cdots)_2=(0.010100011\cdots)_2$，是一个无限位二进制小数。若精度要求是 8 个二进制位，则可截取前 8 位作为近似结果：$0.32\approx(0.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}b_{-7}b_{-8})_2=(0.01010001)_2$。

对既有整数部分又有小数部分的十进制数，可以先分别转换，然后将得到的两部分结果合起来，就得到了转换后的最终结果。例如，$105.6875=(1101001.1011)_2$。

参照上述方法，也可以实现十进制到八进制、十进制到十六进制的转换过程。

例 2.8　将十进制数 725.703125 转换成十六进制表示形式的计算过程如下：

整数部分

$$\begin{array}{r|rl}
16 & 725 & \text{余数为 5，　即 } h_0=5 \\
16 & 45 & \text{余数为 13，即 } h_1=\mathrm{D} \\
16 & 2 & \text{余数为 2，　即 } h_2=2 \\
 & 0 & \text{商为 0，终止}
\end{array}$$

小数部分

$$\begin{array}{rl}
0.703125 & \\
\times\quad 16 & \\ \hline
\mathbf{11}.25 & \text{整数部分为 11，即 } h_{-1}=\mathrm{B} \\
0.25 & \\
\times\quad 16 & \\ \hline
\mathbf{4}.00 & \text{整数部分为 4，即 } h_{-2}=4 \\
0.00 & \text{小数部分为 0，终止。}
\end{array}$$

将转换后整数部分和小数部分合并得：$725.703125=(h_2h_1.h_{-1}h_{-2})_{16}=(\mathrm{2D5.B4})_{16}$。

2.2　数与字符的表示方法

对于数值、文字、图形、图像、音频和视频等这些数值数据和非数值数据，必须将它们以二进制编码形式表示出来，才能为计算机识别、存储、处理和传送。本节主要讲述几类基本的数值数据和非数值数据的二进制编码表示。

需要注意的是，二进制编码与二进制记数是两个有着本质不同的概念，一般来说，二进制编码所表示的并不是二进制数。

2.2.1　整数的表示

1. 无符号整数的表示

对于 0 和正整数(即非负整数)可直接用二进制编码来表示，即二进制编码就是它所表示数的二进制记数形式，这便是无符号整数。

例 2.9　二进制编码 1100 0010 所表示的无符号数是：$(1100\ 0010)_2=(194)_{10}$。

一个 *n* 位二进制码所表示的无符号整数范围为：$0\sim2^n-1$。计算机内部常用 1 字节、2 字节、4 字节和 8 字节等来表示整数，其能表示的无符号整数的范围见表 2.3。

表 2.3　常用数据类型所表示无符号整数的范围

类型/Bytes	二进制位数/bits	范　　围
1 字节	8 位	0～255
2 字节	16 位	0～65 535(64K−1)
4 字节	32 位	0～4 294 967 295(4G−1)
8 字节	64 位	0～18 446 744 073 709 551 615

两个 *n* 位的无符号数相加减，所得结果若超出它对应的范围，则产生结果溢出，此时，所得结果不正确，应当选用更大范围的数据类型。例如，采用 8 位无符号编码，65 和 194 分别为 01000001 和 11000010，但是 65+194＝259，65−194＝−129，两个结果均不在 8 位无符号数的范围内(0～255)，所以溢出。

无符号数溢出的判断方法较简单：当加法运算结果的最高位产生进位，或减法运算结果产生最高位借位时，则运算结果溢出。

例如，采用 8 位无符号数，65+194 和 65−194 的运算过程如下：

0100 0001
+　1100 0010
进位　$\boxed{1}$←00000011(0000 0011 对应 3)
有进位，产生溢出，结果不正确

0100 0001
−　1100 0010
借位　$\boxed{1}$←01111111(0111 1111 对应 127)
有借位，产生溢出，结果不正确

2. 有符号整数的表示

对于有符号数，一般处理的方法是：用 0 表示正数，用 1 表示负数，放在二进制码的最前面。根据编码规则，有符号整数有多种编码表示，如原码、反码、补码等。不同编码的主要区别在于二进制编码和数的对应规则不同。

在计算机系统中，目前普遍采用二进制补码来表示有符号整数，所以，这里只介绍二进制补码的相关内容。长度为 *n* 位二进制补码编码方案如下：

(1) 最高位(Most Significant Bit, MSB)是符号位，其余 $n-1$ 位是数值位。

(2) 符号位，0 表示正数，1 表示负数。

(3) 数值位，正数的编码是其自身的二进制记数形式，负数的编码是把其正数的二进制编码各位取反(0 变 1，1 变 0)再加 1，即正数是其数值本身，负数是其正数取反加 1。

例 2.10 设用 8 位补码表示十进制整数+62 和−62，试定出它们的二进制编码。

解：

+62 的补码	0011 1110
各位取反后	1100 0001
加 1	+ 1
得到−62 的补码	1100 0010

按照补码编码规则，当符号位为 0 时，该补码对应的是 0 和正数，其编码就是数的二进制记数形式；当符号位为 1，该补码对应的是负数，将其编码各位取反再加 1 后所得的二进制码就是该数绝对值的二进制记数形式。可用这个规则求出补码所表示的整数。

例 2.11 求下列补码所表示的数：(1) 00011000；(2) 11101111；(3) 10000000。

解：(1) 符号位是 0，它所表示的数为：$+(00011000)_2=+(18)_{16}=+(24)_{10}$。

(2) 符号位是 1，说明它所表示的是负数。将编码 1110 1111 各位取反再加 1 后得到 0001 0001，所以该补码所表示的数为：$-(00010001)_2=-(11)_{16}=-(17)_{10}$。

(3) 符号位是 1，说明它所表示的是负数。将编码 1000 0000 各位取反再加 1 后得到 1000 0000，所以该补码所表示的数为：$-(10000000)_2=-(80)_{16}=-(128)_{10}$。

一个 n 位补码所表示数的范围为：$-(2^{n-1})\sim+(2^{n-1}-1)$。表 2.4 列出了 8 位、16 位和 32 位补码所表示的有符号数的范围。

表 2.4 常用数据类型所表示有符号整数的范围

类型(Bytes)	二进制位数(bits)	范 围
1 字节	8 位	−128～+127
2 字节	16 位	−32 768～+32 767
4 字节	32 位	−2 147 483 648～+2 147 483 647
8 字节	64 位	−9 223 372 036 854 775 808～+9 223 372 036 854 775 807

对数 x 的补码(记作$[x]_补$)各位取反后再加 1，即可得到该数负值的补码，即

$$[x]_补 \xrightarrow{\text{取反加 1}} [-x]_补 \xrightarrow{\text{取反加 1}} [x]_补$$

例如，+69 和−69 的 8 位补码是$(01000101)_2$和$(10111011)_2$，分别对它们求反加 1：

$[69]_补$	0100 0101	$[-69]_补$	1011 1011
各位取反后	1011 1010	各位取反后	0100 0100
加 1	+ 1	加 1	+ 1
	1011 1011		0100 0101
	10111011 是−69 的补码		01000101 是+69 的补码

即有：$[69]_补 \xrightarrow{\text{取反加 1}} [-69]_补$，$[-69]_补 \xrightarrow{\text{取反加 1}} [x]_补$。

补码的加法规则是：$[x+y]_补=[x]_补+[y]_补$。

该规则表明，当有符号的两个数采用补码形式表示时，要完成计算两数之和的补码表示，只需用两数的补码直接执行加法运算即可，符号位与数值位同等对待，一起参加运算，若运算的结果不超出对应的补码所表示的范围，则结果的符号位和数值位同时为正确值。

例 2.12 用 8 位二进制补码完成十进制整数运算：62+65，62+(−65)，−62+(−65)。

解：	十进制加	二进制补码加法	
	62	0011 1110	
	+ 65	+ 0100 0001	
	127	0111 1111	(即 127 的补码)
	62	0011 1110	
	+ −65	+ 1011 1111	
	−3	1111 1101	(即−3 的补码)
	−62	1100 0010	
	+ −65	+ 1011 1111	
	−127	1←1000 0001	(1000 0001 即−127 的补码，丢弃进位，结果正确)

上例中，符号位和数值一起参加运算，所得结果正确，若产生进位则丢弃。

补码的减法规则是：$[x-y]_{补}=[x]_{补}-[y]_{补}=[x]_{补}+[-y]_{补}$。

该规则表明，当有符号的两个数采用补码形式表示时，要完成计算两数之差的补码表示，只需用两数的补码直接执行减法运算即可，同样符号位一起参加运算，若运算的结果不超出对应的补码所表示的范围，则结果的符号位和数值位同时为正确值。

例 2.13 用 8 位二进制补码完成下列十进制整数运算：62−65，62−(−65)，(−62)−65。

解：	十进制减	二进制补码减	
	62	0011 1110	
	− 65	− 0100 0001	
	−3	1←1111 1101	(1111 1101即−3的补码，丢弃借位，结果正确)
	62	0011 1110	
	− −65	− 1011 1111	
	127	1←0111 1111	(0111 1111即127的补码，丢弃借位，结果正确)
	−62	1100 0010	
	−) 65	− 0100 0001	
	−127	1000 0001	(即−127的补码)

上例中符号位和数值一起参加运算，所得结果正确。

式$[x]_{补}-[y]_{补}=[x]_{补}+[-y]_{补}$表明，补码的减法运算可转换成补码的加法运算，其中$[-y]_{补}$只须对$[y]_{补}$简单地进行求反加 1 即可得到。这一特性对于简化运算电路的实现非常有用，故主要应用于硬件设计中。

两个 n 补码的加、减运算结果若超出它所表示数的范围，则结果产生溢出。

例 2.14 用 8 位补码完成下列十进制整数运算：94+55，(−98)−59。

解：	十进制运算	二进制补码运算	
	94	0101 1110	
	+ 55	+ 0011 0111	
	149	1001 0101	(是−107的补码)
	−98	1001 1110	
	− 59	+ 1100 0101	
	−157	1←0110 0011	(0110 0011是99的补码)

可以看出，例 2.14 中十进制运算结果和其对应的补码运算结果不一致，即补码的运算结果不正确，为什么呢？主要原因是运算结果 149 和−157 超出了−128～+127 的范围。

只有当同号的补码数相加，或异号补码数相减时，才可能产生溢出，而对于同号数相减，或异符号数相加，其运算结果不会溢出。

3. 无符号整数与有符号整数的进一步说明

无符号整数与有符号整数的主要区别在于编码规则不同。如图 2.1 所示，对于无符号整数来说，所有的二进制位均表示数值，编码即无符号数；对于有符号整数来说，最高二进制位为符号位，其余的二进制位用来表示数值。

(1) 无符号整数编码　　　　(2) 有符号整数编码

图 2.1　无符号整数与有符号整数编码规则比较

一个 n 位二进制码可以表示 2^n 个整数，作为无符号整数，它表示整数的范围是 $0 \sim 2^n-1$；作为补码，它所表示整数的范围是 $-2^{n-1} \sim 2^{n-1}-1$，这是因为它们与数的对应规则不同所致。表 2.5 列出了 $n=8$ 时它们各自与整数的对应关系。

表 2.5　8 位二进制码在不同编码规则下与整数的对应关系

二进制码	作为无符号数对应的整数	作为补码对应的整数
0000 0000	0	0
0000 0001	1	+1
0000 0010	2	+2
⋮	⋮	⋮
0111 1110	126	+126
0111 1111	127	+127
1000 0000	128	−128
1000 0001	129	−127
1000 0010	130	−126
⋮	⋮	⋮
1111 1101	253	−128
1111 1110	254	−2
1111 1111	255	−1

如何确定一个二进制数是无符号数还是有符号数？这和具体的使用要求密切相关，既要看怎么用它，也要看具体的上下文，下面举例说明。

例 2.15　(1)用 8 位无符号数计算 226+11；(2)用 8 位补码数计算(−30)+11

解：(1) 226对应的二进制码　　1110 0010
　　　13对应的二进制码　　+　0000 1011
　　　　　　　　　　　　　　　1110 1101　对应的无符号整数为237

(2) −31对应的补码 1110 0010
13对应的二进制码 + 0000 1011
———————————
1110 1101 对应的有符号整数为−19

从例 2.15 中可以看出，二进制码 1110 1101 有两种解释，在(1)中我们将它解释为 237，为什么呢？因为约定用无符号编码规则；在(2)却又将它解释为−19，因为约定用补码规则。

所以，有符号整数和无符号整数都是使用者约定的表示方法。一个二进制码，若以有符号整数的方法来使用它时，则是有符号整数；若以无符号整数的方法使用时，则是无符号整数。也就是说，二进制码自身并不能标明是无符号数还是有符号数，能标明它们的是使用者。若将它作为无符号数使用时，那么就应该用无符号数的方法来处理，若把它作为有符号数使用时，则应该用有符号数的方法来处理。

例 2.15 还说明一点：无符号整数和补码使用相同的规则来处理加法运算，这一结论对减法运算也适用。事实上，无符号数和补码数的加法和减法的运算规则完全相同，但是由于约定的表示规则不一样，所以它们各自采用不同的溢出判断方法。

2.2.4 字符表示

计算机除了对数值类型数据处理外，还需处理字符(含各种符号、数字、字母等)，因此，需要为每个字符规定一个特定的编码，以便能够以二进制形式表示。

1. ASCII 字符编码

目前使用最普遍的是ASCII字符编码，即美国标准信息交换码(American Standard Code for Information Interchange, ASCII)。该编码已被国际标准化组织采纳，作为国际通用的信息标准交换代码。ASCII 码包括标准版和扩展版，通常所说的 ASCII 码指的是标准版。

标准 ASCII 码采用 7 位二进制码表示一个字符，共规定了 128 个字符的编码，见表 2.7。其中 96 个可打印字符，包括阿拉伯数码、英文字母、标点符号和运算符等，以及 32 个不能打印出来的控制符号。

表 2.7 中每个字符所在行与列所对应的十六进制码分别是该字符 ASCII 码值的低 4 位和高 3 位值，例如，'A'的 ASCII 码值为 41h，'b'的 ASCII 码值为 62h，ASCII 码值是 30h 所对应的字符是'0'。

为便于记忆，现将 ASCII 码表中的一些规律说明如下：

(1) 表中最前面的 32 个码(00h～1Fh)和最后一个码(7Fh)不对应任何可印刷的字符，主要用于对计算机通信中的通信控制或对计算机设备的控制，称控制码。这些字符中使用比较多的有：回车符 CR(0Dh)、换行符 LF(0Ah)、退格符 BS(08h)、水平跳格 HT(09h)。

(2) 空格字符 SP 的编码值是 32(20h)。

(3) 数字符'0'～'9'的编码值是 48～57(30h～39h)；大写英文字母'A'～'Z'的编码值是 65～90(41h～5Ah)，小写英文字母'a'～'z'的编码值是 97～122(61h～7Ah)，大、小写字母编码值相差 32(20h)。

表 2.7　标准 ASCII 码表

$b_6b_5b_4$ / $b_3b_2b_1b_0$	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	～
F	SI	US	/	?	O	_	o	DEL

虽然标准 ASCII 码是 7 位编码，但一般以一个字节来存放一个 ASCII 字符，每一个字节中多余出来的一位(最高位)通常置 0。

扩展 ASCII 码是在兼容标准 ASCII 码基础上制定的。它是 8 位二进制编码，最高位是 0 的编码 00h～7Fh(0～127)对应的是标准 ASCII 码，最高位为 1 的编码 80h～FFh(128～255)对应的是扩充的 128 个字符。

2.3　二进制码的基本逻辑运算

二进制符号 1 和 0 在逻辑上可以代表“真”与“假”、“是”与“否”、“有”与“无”等互相对立的两种状态，也就是说它具有逻辑属性，所以，二进制数可以进行逻辑运算。

二进制数的逻辑运算和算术运算处理规则是不同的，逻辑运算是按位进行的，位与位之间彼此独立，不像加减运算那样，位与位之间存在进位或借位的联系。

常用的基本逻辑运算有四种：“与”运算、“或”运算、“非”运算、“异或”运算。

1.“与”运算(AND)

“与”运算即逻辑乘法，通常用符号“∧”或“·”来表示。“与”运算规则如下：

$$0\wedge0=0 \qquad 0\wedge1=0 \qquad 1\wedge0=0 \qquad 1\wedge1=1$$

不难看出，只有当运算各方同时取值为 1 时，其结果才等于 1，否则结果为 0。

例 2.18　求二进制数 1101 0011 与 1001 0110 的逻辑“与”。

解：　1　1　0　1　1　0　1　0

∧　1　0　0　1　0　1　1　0

	1	0	0	1	0	0	1	0

所以，$(1101\ 1010)_2 \wedge (1001\ 0110)_2 = (1001\ 0010)_2$。

2.“或”运算(OR)

“或”运算即逻辑加法，常用符号“∨”或“+”来表示。“或”运算规则如下。

$$0 \vee 0 = 0 \qquad 0 \vee 1 = 1 \qquad 1 \vee 0 = 1 \qquad 1 \vee 1 = 1$$

不难看出，只有当运算各方同时取值为 0 时，其结果才等于 0，否则结果为 1。

例 2.19 求二进制数 1101 1010 与 1001 1001 的逻辑“或”。

解：

	1	1	0	1	1	0	1	0
∨	1	0	0	1	1	0	0	1
	1	1	0	1	1	0	1	1

所以，$(1101\ 1010)_2 \vee (1001\ 1001)_2 = (1101\ 1011)_2$。

3. “非”运算(NOT)

“非”运算即逻辑否定，通常用符号“¬”或在运算变量上面加一根横线表示。“非”运算规则如下。

$$\neg 1 = 0 \qquad \neg 0 = 1$$

不难看出，“非”运算就是“取反”运算。

例 2.20 求二进制数 1101 1010 的逻辑“非”。

解：

¬	1	1	0	1	1	0	1	0
	0	0	1	0	0	1	0	1

所以，$\neg(1101\ 1010)_2 = (0010\ 0101)_2$。

4. “异或”运算(XOR)

“异或”运算通常用符号“∀”或“⊕”来表示。“异或”运算规则如下。

$$0 \forall 0 = 0 \qquad 0 \forall 1 = 01 \qquad 1 \forall 0 = 1 \qquad 1 \forall 1 = 0$$

不难看出，只有当运算双方取值为不同时，其结果等于 1，双方取值相同时结果等于 0。

例 2.11 求二进制数 1101 1010 与 1001 1001 的逻辑“异或”。

解：

	1	1	0	1	1	0	1	0
∀	1	0	0	1	1	0	0	1
	0	1	0	0	0	0	1	1

所以，$(1101\ 1010)_2 \forall (1001\ 1001)_2 = (0100\ 0011)_2$。