计算机引论

山东建筑大学
计算机学院
秦松

3

数的表示

转换

Example

转换二进制到十进制 **10011 .**

Solution

写出每一位和他所对应的全值。将每一位与其权值相乘并记下相乘结果，将结果加起来的到十进制数。

二进制	**1**	**0**	**0**	**1**	**1**
权重	**16**	**8**	**4**	**2**	**1**

----

16 + 0 + 0 + 2 + 1

十进制 **19**

Example

转换十进制到二进制 35 。

Solution

将要转换的数写在右上角。连续被2除，写下每次的商和余数。商写在左边，余数写在各自运算下。直到上为零。

十进制	0 **ç**	1 **ç**	2 **ç**	4 **ç**	8 **ç**	17 **ç**	35 Dec.
二进制		**1**	**0**	**0**	**0**	**1**	**1**

Example

转换成十六进制数 0011100010.

Solution

每 4 位分成 1 组，转换为对应的十六进制数.（从右边开始）. 在这个例子中，需要在左边添加两个 0 时总位数能被 4 整除。原式变为 000011100010,

二进制	**0000**	**1110**	**0010**
十六进制	**0**	**E**	**2**

结果是 x0E2.

Example

转换成二进制 **x24C?**

Solution

将每个十六进制数据转换为二进制数，

十六进制	**2**	**4**	**C**
二进制	**0010**	**0100**	**1100**

结果是 **001001001100.**

Example

转换为八进制 **1100010.**

Solution

每 3 位分成 1 组，转换为对应的八进制数.（从右边开始）.在这个例子中，需要在左边添加两个 0 时总位数能被 3 整除。原式变为 000011100010,

二进制	**000**	**011**	**100**	**010**
八进制	**0**	**3**	**4**	**2**

结果是 **342_8.**

Example

转换成二进制 **24_8?**

Solution

将每个八进制数据转换为二进制数，

八进制	**2**	**4**
二进制	**010**	**100**

结果是 **010100.**

Example

写出八进制 **$(11110100.1110111)_2$.**

Solution

二进制	011	110	100	.	111	011	100
八进制	3	6	4	.	7	3	4

$(11110100.1110111)_2=(364.734)_8$

Example

写出二进制（351.65）$_8$?

Solution

八进制	3	5	1.	6	5
二进制	011	101	001.	110	101

$(351.65)_8=(11101001.110101)_2$

整数表示法

无符号整数

无符号整数就是没有符号符号的整数，无负数，**0**~无穷大；

计算机不可能表示，所以大多数计算机定义了一个最大的无符号整数的常量。

范围：**0~ (2^N-1)**，**N**代表？

无符号整数范围

Number of Bits	*Range*	
8	0	255
16	0	65,535

比如**int** 类型占用**4**个字节，**32**位。？

无符号整数格式

表示法：

1.将整数变成二进制数；

2.如果二进制位不足**N**位，则在二进制数的左边补**0**，使其总位数为**N**位。

Example

将**7**存储在**8**位存储单元中。

Solution

先将这个数转换为二进制数**111**。加**5**个**0**使总位数为**N(8)**，得**00000111**。在将该数存储到存储单元中。

Example

Store 258 in a 16-bit memory location.

Solution

First change the number to binary 100000010. Add seven 0s to make a total of N (16) bits, 0000000100000010. The number is stored in the memory location.

两类不同的计算机中无符号整数的存储和无符号整数的译解

Decimal	*8-bit allocation*	*16-bit allocation*
7	00000111	0000000000000111
234	11101010	0000000011101010
258	overflow	0000000100000010
24,760	overflow	0110000010111000
1,245,678	overflow	overflow

译解：将**N**位二进制数从二进制系统转换为十进制系统

Example

把以无符号整数形式存储的**00101011**译为十进制数。

Solution

写出每一位和他所对应的全值。将每一位与其权值相乘并记下相乘结果，将结果加起来的到十进制数。

二进制	0	0	1	0	1	0	1	1
权重	128	64	32	16	8	4	2	1

0 + 0 + 32 + 0 + 8 + 0 + 2 + 1

十进制 **43**

溢出与应用

溢出？

超范围，水杯

1.计数;

2.寻址.没有负地址

符号加绝对值（原码）格式

数值有正负之分,我们需要用1个二进制位表示符号(0 + + 1 -).

少一位最大值为无符号整数的一半。

why

范围: $-(2^{N-1}-1)\sim+(2^{N-1}-1)$

符号加绝对值（原码）格式的范围

# of Bits	*Range*	
8	–127	+127
16	–32767	+32767
32	–2,147,483,647	+2,147,483,647

符号加绝对值（原码）格式表示法

表示法:

1.先将数转换为二进制数；符号位忽略；

2.在数的左边补0，使他总位数为N-1位；

3如果为正数符号位为0；为负数符号位为1。

注:

在符号加绝对值（原码）表示法中，
最左边的位用于定义数的符号。如果是0，则表示该数为正数，如果是1，则为负数

0的表示

注:

0在符号加绝对值（原码）格式有两种:正0和负0。

例如在8位的存储单元中:

+0 è 00000000
-0 è 10000000

Example

用符号加绝对值（原码）表示法将+7存储在8位的存储单元中。

Solution

先将数转换为二进制111，加上4个0补足N-1 (7)位，0000111. 因为是正数额外在加1个0表示正数。结果是:

00000111

Example

用符号加绝对值（原码）表示法将–258存储在16位的存储单元中。

Solution

先将数转换为二进制100000010，加上6个0补足N-1 (15)位， 000000100000010. 因为是负数额外在加1个1表示负数。

结果是: 1000000100000010

符号加绝对值（原码）译解

译解:

1.忽略第一位.

2.把剩下的**N-1**位二进制数转换为十进制数

3.在数的最左边加上**+**或**–**

Example

把以符号加绝对值（原码）整数形式存储的**10111011**译为十进制数。

Solution

忽略最左边一位，剩下**0111011**。转换为十进制**59**。

原数最左边是**1**，所以结果**–59**。

二进制反码格式

对除符号位外的其余各位逐位取反就产生了反码.

表示正数，约定使用无符号整数.
表示负数，用正数的反码形式。

反码的取值空间和原码相同且一一对应.

范围: $-(2^{N-1}-1) \sim +(2^{N-1}-1)$

# of Bits	***Range***	
8	–127	+127
16	–32767	+32767
32	–2,147,483,647	+2,147,483,647

二进制反码表示法

表示法:

1.先将数转换为二进制数，符号被忽略。

2.在数的左边补**0**，使其总位数为**N.**

3.如果符号为正，则不需变动，符号为负，则将每一位换成它的反码形式（把**0**变成**1**或把**1**变成**0**）.

注:

在二进制反码表示法中，最左边**1**位定义数的符号。如果为**0**，数值为正，如果是**1**，数值为负。

数的反码的反码为本身

0的表示

注:

0在二进制反码中有两种表示方法，如在**8**位存储单元中:

+0 è 00000000
-0 è 11111111

Example

将**7**存储在**8**位存储单元中，用二进制反码。

Solution

先将数转换为二进制111，加上5个0补足N (8)位， 00000111. 因为是正数,

结果是: 00000111

Example

将–258存储在16位存储单元中，用二进制反码。

Solution

先将数转换为二进制100000010，加上7个0补足N (16)位， 0000000100000010. 因为是负数，所以把每一位换成它的反码形式，

结果是:　　　　　　1111111011111101

两类不同的计算机中二进制反码格式的存储和二进制反码格式译解

Decimal	*8-bit allocation*	*16-bit allocation*
+7	00000111	0000000000000111
−7	11111000	1111111111111000
+124	01111100	0000000001111100
−124	10000011	1111111110000011
+24,760	overflow	0110000010111000
−24,760	overflow	1001111101000111

译解：1.如果最左边的位为0，

a.把整个二进制数转换为十进制。 b.在数的前面加+.

2.如果最左边的位为1，

a.把整个二进制数转换成其反码形式（把所有的0变成1，把所有的1变成0. b.再把转换过的二进制数转换为十进制数。 c.在数的前边加-.

Example

把以二进制反码形式存储的11110110 译为十进制数。

Solution

最左边1位为1,所以该数为负数.先转换为反码，结果是00001001. 转换为十进制是9.

所以原来的数是–9.

注:

二进制反码表示法需要转换所有的位。正数的反码得到相应的负数，负数的反码得到相应的正数，一个数去两次反码，仍然是其本身。

应用

应用范围:

少用

少用的原因:

1、 0多种表示用起来麻烦

0在二进制反码中有两种表示方法，如在8位存储单元中:

+0 è 00000000

-0 è 11111111

2、进行加、减不方便

二进制补码

负数的补码就是对反码加一,而正数不变,正数的原码反码补码是一样的.

范围: $-(2^{N-1}) \sim +(2^{N-1}-1)$

二进制补码的范围

# of Bits	*Range*		
8	**-128**	0	+127
16	−32,768	0	+32,767
32	−2,147,483,648	0	+2,147,483,647

在补码中用(-128)代替了(-0),所以补码的表示范围为:(-128~0~127)共256个.

注意:(-128)没有相对应的原码和反码, (-128) = (10000000)

二进制补码表示法

表示法:

1.先将数转换为二进制数；符号被忽略

2.如果二进制数位数不足N位，在数的左边补0.

3.如果符号为正，不需变动，如果符号为负，则将最右边的所有0和首次出现的1保持不变，其余位取反。

注:

如果对正数求二进制补码，则得到相应的负数。如果对负数求其二进制补码，则得到相应的正数。如果对一个数取两次二进制补码，就得到原来的数。

0的表示

注:

二进制补码是目前最普遍、最重要、应用最广泛的整数表示法.在二进制补码表示法中，最左边的位定义为符号位。如果为0，数为正。如果为1，数为负。在二进制补码表示法中，0只有一个表示法。在8位的存储单元中:

0 è 00000000

Example

存储 +7 在8位存储器中用二进制补码.

Solution

先将数转换为二进制数111；在数的左边补5个0补足N (8)位， 00000111.符号为正，不需变动，

The result is: ***00000111***

Example

存储 –40 在16位存储器中用二进制补码.

Solution

先将数转换为二进制数101000；在数的左边补10个0补足N (16)位， 0000000000101000.符号为负，将最右边的所有0和首次出现的1保持不变，其余位取反，

The result is:
1111111111011000

两类不同的计算机中二进制补码格式的存储和二进制补码格式译解

Decimal	*8-bit allocation*	*16-bit allocation*
+7	00000111	0000000000000111
–7	11111001	1111111111111001
+124	01111100	0000000001111100
–124	10000100	1111111110000100
+24,760	overflow	0110000010111000
–24,760	overflow	1001111101001000

译解：1.如果最左边的位为0（正数）,把整个二进制数转换为十进制数，在数的前面加+.
2.如果最左边的位为1（负数）
a.从最右边开始到第一个1出现保持不变，其余的换成它的反码形式。 b.再把得到的二进制数转换为十进制数. c.在数的前面加-.

Example

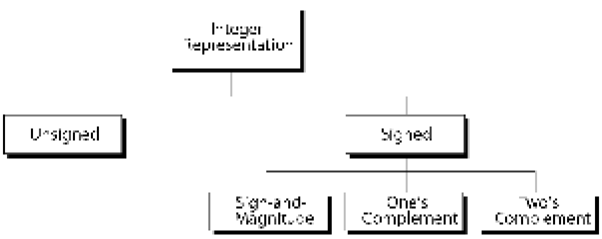
将二进制补码11110110译解成十进制数.

Solution

最左边是1，所以是负数. 右边的10不变，其余的位取反得00001010. 得到十进制10.

所以原来的数是 –10.

小结



整数表示法小结

Contents of Memory	Unsigned	Sign-and-Magnitude	One’s Complement	Two’s Complement
0000	**0**	+0	+0	+0
0001	**1**	+1	+1	+1
0010	**2**	+2	+2	+2
0011	**3**	+3	+3	+3
0100	**4**	+4	+4	+4
0101	**5**	+5	+5	+5
0110	**6**	+6	+6	+6
0111	**7**	+7	+7	+7
1000	**8**	-0	-7	-8
1001	**9**	-1	-6	-7
1010	**10**	-2	-5	-6
1011	**11**	-3	-4	-5
1100	**12**	-4	-3	-4
1101	**13**	-5	-2	-3
1110	**14**	-6	-1	-2
1111	**15**	-7	-0	-1

Answer

定点表示法一般采取两种简单的约定：

1、定点小数：小数点固定在最高位之前、符号位之后，参与运算的数是纯小数

2、定点整数：小数点固定在最低位之后，参与运算的数是纯整数

浮点表示法

浮点表示法

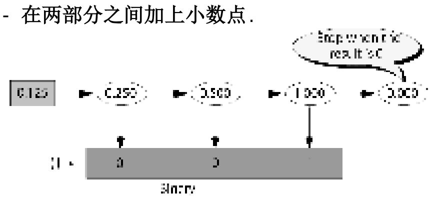
为了表示浮点数（即包含整数又包含小数），要经过几个步骤：

1、数被分为两部分：整数部分和小数部分。

例如：浮点数**14.234**=整数部分**14**+小数部分**:0.234**

2、把浮点数转换成二进制数：

- 把整数部分转换为二进制数.
- 将小数部分转换为二进制数.
- 在两部分之间加上小数点.



Example

将小数**0.875**转换为二进制

Solution

将小数写在左上角，不断用2乘，提取整数部分作为二进制位.直到该数为0.0.

0.875 **è** 1.750 **è** 1.5 **è** 1.0 **è** 0.0

0 **.** 1 1 1

Example

将小数**0.4**转换为**6**位的二进制数**.**

Solution

将小数写在左上角，不断用2乘，提取整数部分作为二进制位.直到该数小数点后为6位.

0.4 **è** 0.8 **è** 1.6 **è** 1.2 **è** 0.4 **è** 0.8 **è** 1.6

0 . 0 1 1 0 0 1

浮点表示法

3、规范化

为了表示1101.1101(13.8125)，需将符号、所有的位和小数点的位置存储起来，可行，但数的运算变难，如可表示为 $2^3 \times$ 1.1011101 or

$2^5 \times$ .011011101 or

$2^{-4} \times$ 11011101.

就需要一个浮点数标准表示法，解决的办法我们称之为规范化。

注

规范化:移动小数点使小数点左边只有1个1.

规范化

为了表示数的原始值，将它乘以2^e，e表示这个数规范化时小数点移动的位数，正数左移，负数右移，正负号加在数的最左边

规范化示例

Original Number	*Move*	*Normalized*
------------	------------	------------
+1010001.1101	ç 6	$+2^6$ x 1.01000111001
−111.000011	ç 2	-2^2 x 1.11000011
+0.00000111001	6 è	$+2^{-6}$ x 1.11001
−0.001110011	3 è	-2^{-3} x 1.110011

$$N = (\pm)2^{\pm E} \times S$$

浮点表示法

4、符号、幂和尾数

规范化后，只存储了这个数的三部分信息：符号、幂和尾数

例如:

数**+1000111.0101**规范化后,

+ 2^6 $\times$ 1.0001110101

符号 + 指数 **6** 尾数 **0001110101**

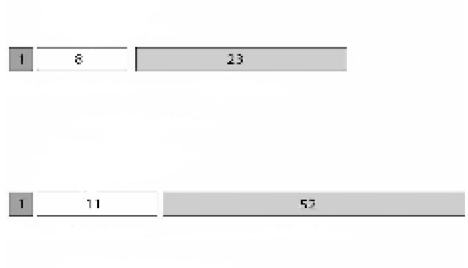
符号:可用一个二进制位来储存**(0 or 1).**

指数:定义小数点移动的位数。幂可以为正也可以为负.

尾数:是小数点右边的二进制数. 它定义了数的精度. 作为无符号整数存储.

IEEE 标准

电器和电子工程师协会（**IEEE**）定义了三种存储浮点数的标准。其中两种用于内存中存储数值（单精度**32**位和双精度**64**位）。



单精度表达法

1. 存储符号，**0 (**正数**) or 1 (**负数**).**

2. 存储指数.（2的幂）

3.以无符号整数存储尾数.

Example 给出下列规范化数的单精度表示法

+ 2^6 $\times$ 1.01000111001

Solution

符号为正，用0表示. 指数为6，在Excess_127 表示法中，指数加上127为133. 补0成为23位.得:

0 00000110 01000111001000000000000

sign exponent mantissa

浮点表示法译解和示例

1.用最左边一位表示符号.

2.将后面的**8**位转换为十进制.结果为指数.

3.在剩下的**23**位前面加一个**"1"**和小数点，可以忽略右边所有多余的**0.**

4.将小数点加在正确的位置。

5. 将整数部分转换为十进制.

6.将小数部分转换为十进制。

7.把他们组合起来.

Example

转换下列32位浮点数

1 00000011 11001100000000000000000

Solution

符号为负. 指数为3 .

结果是

$-2^3 \times 1.110011$