《程序设计基础》第二讲 剖析二进制

原创 2016-09-25 李骏扬 骏扬工作室

我们之前介绍了R进制，虽然R进制实际上已经包含了二进制，但是我们这里着重对二进制进行详述。

2.A 二进制的数数

根据R进制拥有R个符号的特点，二进制显然是只有两种符号：0和1。因为二进制中最大的数字是1，如果数数的话，1之后就得进位了。比如，二进制第一个数字是0，第二个数字是1，再往下数，1之后就得进位，是10，也就是十进制中的2，二进制的10之后是11，11之后又得连续进位，得到100，也就是十进制中的4。二进制的前16个数字如下表：

十进制	二进制	十进制	二进制
0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111

二进制数字的每一位同样是拥有位权的。例如1001这个４位二进制数字，从左至右４位的位权分别是2的3次方、2的2次方、2的1次方和2的0次方。因此，二进制数字1001表达的数值为：

$$(1001)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 0 + 0 + 1 = 9.$$

对一个n+1位的二进制数字

$$r_n r_{n-1} r_{n-2} \cdots r_2 r_1 r_0 \quad (\forall i,\ r_i \in \{0,1\})$$

来说，其表达的值为：

$$
\begin{aligned}
&(r_n r_{n-1} r_{n-2} \cdots r_2 r_1 r_0)_2 \\
&= r_n 2^n + r_{n-1} 2^{n-1} + r_{n-2} 2^{n-2} + \ \cdots \ + r_2 2^2 + r_1 2^1 + r_0 2^0 \\
&= \sum_{i=0}^{n} r_i 2^i
\end{aligned}
$$

除了0~15之外，以下的一些二进制数字也经常会用到：

十进制	二进制	十进制	二进制
$2^0 = 1$	1	$2^0 - 1 = 0$	0
$2^1 = 2$	10	$2^1 - 1 = 1$	1
$2^2 = 4$	100	$2^2 - 1 = 3$	11
$2^3 = 8$	1000	$2^3 - 1 = 7$	111
$2^4 = 16$	10000	$2^4 - 1 = 15$	1111
$2^5 = 32$	100000	$2^5 - 1 = 31$	11111
$2^6 = 64$	1000000	$2^6 - 1 = 63$	111111
$2^7 = 128$	10000000	$2^7 - 1 = 127$	1111111
$2^8 = 256$	100000000	$2^8 - 1 = 255$	11111111
$2^9 = 512$	1000000000	$2^9 - 1 = 511$	111111111
$2^{10} = 1024$	10000000000	$2^{10} - 1 = 1023$	1111111111

2.B R进制与二进制的计算

任何一种进制的计算都只需要掌握两个要点：1. 进位加法，2. 乘法表。

例如，在十进制中，如果你掌握了进位加法，减法自然也不在话下。如果你掌握了九九乘法表，那么多位乘法和除法自然也不在话下。

但是，如果是八进制呢？好吧，首先我们来看进位加法。比如：a + b会引发进位，而 a + c又恰好为8，那么a + b = (a + c) + (b - c) = 10 + (b - c)，注意，这里的“10”是在八进制中，在八进制中就表示“8”。所以就得到以下八进制加法：

7+1 = 10

6+5 = (6 + 2) + 3 = 10 + 3 = 13

4+6 = (4 + 4) + 2 = 10 + 2 = 12

如果是二进制的加法呢？遇到进位的加法规则只有一条：1 + 1 = 10

对于多位数的二进制加法，当然也可以仿照小学时候学过的竖式加法：

$$
\begin{array}{r}
101 \\
+\quad {}_1 1 {}_1 1\, 0 {}_1 1 \\
\hline
10010
\end{array}
$$

减法呢，退位的方法和加法类似，相信读者的能力，这里就不赘述了。

下面说R进制，乃至R进制的乘法。乘法的数学含义和进制是无关的，所以这里不再啰嗦了。不过我们日常在做十进制的乘法时，已经严重依赖于乘法口诀表了，所以如果要灵活的对R进制进行乘法计算，有一个乘法口诀表显然是最好的。

比如，如果要做八进制乘法，最好首先知道八进制的乘法口诀表，前提是你愿意背诵：

1×1=1，1×2 = 2 …… 2×4=10，2×5=12，……4×4=20……7×7=61

若是四进制，乘法口诀表要简单一些，就下面这么几条：

1×1=1，1×2 = 2，1×3 = 3，2×2 = 10，2×3 = 12，3×3 = 21

若是三进制，乘法口诀表就更简单了：

1×1=1，1×2 = 2，2×2 = 11

如果是二进制呢？乘法口诀表只剩下一条了：

1×1=1

啥？就这一条？？是啊！那0×1=0 呢？呵呵，在十进制乘法口诀表里面也没有出现0×1啊，嘿嘿~~

进行多位二进制的乘法和除法其实特别简单，读者可以自己感受一下下面的竖式多位乘法和除法：

$$
\begin{array}{r}
1011 \\
\times\quad 1101 \\
\hline
1011 \\
0000 \\
1011 \\
{}_1 1 {}_1 0 {}_1 1 {}_1 1 \\
\hline
10001111
\end{array}
\qquad\qquad
\begin{array}{r}
11 \\
111\,\overline{)\,11001} \\
111 \\
\hline
1011 \\
111 \\
\hline
100
\end{array}
$$

由于只有0和1的乘法，竖式乘法每一行中间结果，要么相乘得到0，要么只需要把被乘数抄一遍，这要比十进制乘法简单得多。同样，在做除法时，试商只需要通过比大小就可以了。上图右侧的式子就是二进制竖式乘除法，完成了25除以7商3余4的计算。

其实对于R进制的乘法和除法，还有一些和进制无关的通用规则，比如，在R进制中，任何一个数字乘以10的N次方，小数点向右移动N位，任何一个数除以10的N次方，小数点向左移

动N位（注意这里的10是R进制的10，不是十进制中的十）。再比如八进制中 271×10 = 2710，十六进制中，8FF00 ÷100 = 8FF。关于这个规律，读者可以自己琢磨一下~~

2.C 进制的转换

2.C.1 R进制 → 十进制

R进制转换为十进制相对比较容易，只要依照公式

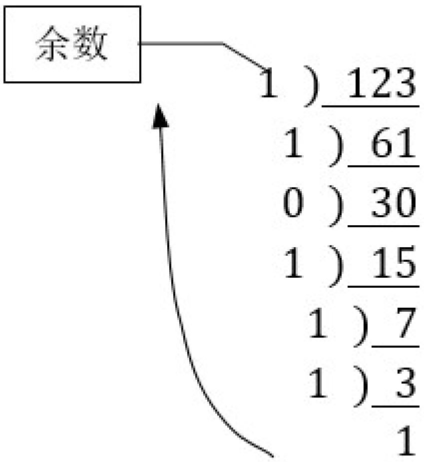
$$
\begin{aligned}
&(r_n r_{n-1} \dots r_1 r_0)_R \\
&= r_n R^n + r_{n-1} R^{n-1} + \cdots + r_1 R^1 + r_0 R^0 \\
&= \sum_{i=0}^{n} r_i R^i
\end{aligned}
$$

进行计算就可以了。当然，对二进制，我们对数字应该具有敏感性，比如二进制中的10010，你是否可以快速得到16+2=18？

2.C.2 十进制 → 二进制

考虑十进制转换为二进制，首先如果我们能将一个数字快速拆分为2的N次方的和，那么就可以快速写出其二进制表达，例如99 = 64 + 32 + 2 + 1，分别是2的6次方、5次方、1次方和0次方之和，所以99的二进制表达为：1100011。

当然，我们还有一种标准做法，该方法任何一本书上都会有介绍。举个例子：比如十进制数字123。我们的标准做法是将123除以2，得到61和余数1，再用61除以2，得到30和余数1，如此循环，直到除到1。此过程如下：



最后，将余数**从下至上**排列，得到十进制数字123的二进制表达：1111011。

在上课的时候，有同学提出了一个问题：“为什么呢？”，也就是为什么这么做就能得到一个数字的二进制表达呢？且看下面的分析：

首先看一个十进制数据：比如说123，将这个数字除以十，得商12，余数3，这个余数3就是123这个数字的个位数，然后将商12再次除以十，得到1余2，那么这时候余数2就是123的倒数第二位，当然最后一个数字1是第一位。

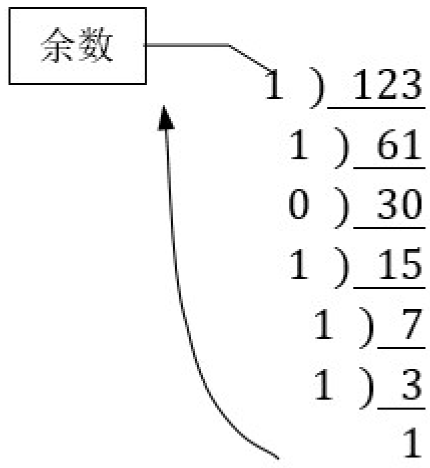
通过这个分析我们不难发现，如果让932转换成R进制，只需要进行以下循环步骤：

首先，123除以R，得到的商假设为N0，余数为r0，那么r0则为R进制的最后一位；

然后，商N0再次除以R，得到商N1，余数r1，那么r1则为R进制表达的倒数第二位；

如此类推，就可得到123这个数字的完整 R 进制表达。

当然，如果 R =2，即将一个数字转换为二进制，则先将这个数字除以2，其余数就是二进制表达的最后一位，将刚才除法得到的商再次除以2，得到二进制倒数第二位，然后将上一次除法的商再除以2，以此类推，得到二进制的每一位，直到那个数字最后为1。其实读者不难发现，这个过程，就是我们之前介绍的拆分123的过程。再看一下这个过程：



由于第一次除法的余数得到的是最后一位，其次得到的倒数第二位......直到得到第一位，所以在这个过程之后，从下往上倒过来排列余数就是这个数字的二进制表达。

2.C.3 十六进制与二进制的相互转换

首先探讨一下十六进制，十六进制在之前谈到R进制的时候已经说过了，采用16个字符表示，分别是0~9，A~F。当你第一眼看到一个十六进制数字“CD”的时候，你的第一反应是什么？

这时候千万不要去将“CD”翻译为十进制。其实“CD”就是一个数字，可能和十进制数字“78”没什么区别。只是我们习惯了看十进制数字，却不太习惯看十六进制数字，这也有点像我们初学英语的时候，看着所有的英文词句都要将它们翻译成中文才能理解一样。可是，你既然不会去深究“78”，不会去细细的想“78”是由10个7，1个8来组成，又何必去细究“CD”是多少呢？你只要知道“CD”在“CC”之后，在“CE”之前就行了。

我们在计算机中经常会使用十六进制，所以看到十六进制数据也要“坦然”。这也有点像学英文，真正的融会贯通者会学着直接用英文去思考，我们看到十六进制数字，也用十六进制去思考吧！若是遇到简单的计算，也可以直接用十六进制去计算。比如CD + 1 = CE，CD + 3 = D0。

好了，现在我们来做一个数学推理，以便于了解十六进制和二进制的微妙关系。

我们把一个多位的二进制数字，四位一组来书写（不够的数字前边用 0 补足），下面这个二进制数字有n+1组，一共4(n+1)位：

$$r_{4n+3}r_{4n+2}r_{4n+1}r_{4n}r_{4(n-1)+3}r_{4(n-1)+2}r_{4(n-1)+1}r_{4(n-1)}\dots r_3r_2r_1r_0$$

若这个二进制数字中间的任意一组为

$$r_{4i+3}r_{4i+2}r_{4i+1}r_{4i} \quad n \le i \le 0$$

那么可以有以下的推导

$$(r_{4i+3}r_{4i+2}r_{4i+1}r_{4i})_2$$

$$= r_{4i+3}\cdot 2^{4i+3} + r_{4i+2}\cdot 2^{4i+2} + r_{4i+1}\cdot 2^{4i+1} + r_{4i}\cdot 2^{4i}$$

$$= (r_{4i+3}\cdot 2^3 + r_{4i+2}\cdot 2^2 + r_{4i+1}\cdot 2^1 + r_{4i})\cdot 2^{4i}$$

$$= (r_{4i+3}\cdot 2^3 + r_{4i+2}\cdot 2^2 + r_{4i+1}\cdot 2^1 + r_{4i})\cdot 16^i$$

$$= x_i 16^i$$

其中

$$r_{4i+3}\cdot 2^3 + r_{4i+2}\cdot 2^2 + r_{4i+1}\cdot 2^1 + r_{4i} = x_i$$

从推导的过程不难看出

$$(r_{4i+3}r_{4i+2}r_{4i+1}r_{4i})_2 = (x_i)_{16}$$

考虑到i介于0到n之间，那么

$$(r_{4n+3}r_{4n+2}r_{4n+1}r_{4n}r_{4(n-1)+3}r_{4(n-1)+2}r_{4(n-1)+1}r_{4(n-1)}\dots r_3r_2r_1r_0)_2$$

$$= (x_nx_{n-1}\dots x_0)_{16}$$

其中

$$(r_{4n+3}r_{4n+2}r_{4n+1}r_{4n})_2 = (x_n)_{16}$$

$$(r_{4(n-1)+3}r_{4(n-1)+2}r_{4(n-1)+1}r_{4(n-1)})_2 = (x_{n-1})_{16}$$

$$\dots$$

$$(r_3r_2r_1r_0)_2 = (x_0)_{16}$$

上面这么一大堆的推导过程，无非说明一个简单的问题：二进制与十六进制之间有天然的4:1的关系，4位二进制数字对应1位十六进制数字。下例为十六进制数字1FA换位二进制：

$$\begin{array}{ccc} 1 & F & A \\ \downarrow & \downarrow & \downarrow \\ 0001 & 1111 & 1010 \end{array}$$

又如将二进制数字110110110转换为十六进制数字：

$$\begin{array}{ccc} 1 & 1011 & 0110 \\ \downarrow & \downarrow & \downarrow \\ 1 & 9 & 6 \end{array}$$

同样的，八进制和二进制之间有三位对一位的关系，这里就不再列举了。

2.D 整数的表达

计算机中，一个二进制数字，0或者1，我们称作1位，也称作一个比特（bit），而8个二进制数字称作一个字节（Byte）。通常，比特使用小写字母“b”表示，字节使用大写字母“B”表示。

计算机中，通常采用有限个比特位来表示一个整数。例如，如果一个比特表示一个整数，那只能表达两个数字：0或者1。如果使用2个比特，那么就可以表达四个整数：00、01、10和11（也就是0、1、2、3）。如果是4个比特，则可以表达16个数字0000、0001、0010、0011、……1110、1111，也就是0到15。显然，如果有 N 个比特位，则可以表示 2的N次方个数据，从0至2的N次方减1。

通常在计算机中，采用8比特（1字节）、16比特（2字节）、32比特（4字节），或者64比特（8字节）来表示一个整数。不同的比特数表示的整数范围如下

字节数	位数	最小值	最大值
1	8	0	$2^8-1=255$
2	16	0	$2^{16}-1=65,535$
4	32	0	$2^{32}-1=4,294,967,295$
8	64	0	$2^{64}-1=18,446,744,073,709,551,615$
$N/8$	N	0	2^N-1

2.E 溢出问题

计算机在表达整数的时候，由于选择不同的表达位数，其表达范围是有限的，这时候计算机在计算的时候，有可能超出其表达范围，比如，单字节整数，在上表中，最大的可以标识的数字是255，如果在存储单元中，有一个数字是255，结果做了一个计算255+1，结果会是多少呢？这要看计算机中的二进制是如何计算的了。

单字节整数255的二进制表达为1111 1111，刚好用满8位，加1之后得到1 0000 0000，结果需要9位来存储，但是计算机依旧只会提供8位来存储，因此会将最高位的1舍去，那么1 0000 0000居然变成了0000 0000，这样计算结果就出错了，也就是说在8位单字节的计算中，255+1会得到0，这种在计算过程中超出了计算机表达范围而导致的数据错误，我们叫做**溢出**。同样的，在单字节的计算中，254+4会得到2，200+200会得到144。

溢出不仅仅发生在加法，减法、乘法都会发生溢出。解决溢出的方案基本上只有两个：第一，扩大数据的表达范围，比如在双字节，或四字节、八字节的计算中，255+1、254+2和200+200都不会发生溢出。第二，当然是“小心谨慎”的计算，在连续加减或乘除计算的时

候，调整计算次序防止溢出。

今天的讲解就到这里，关于二进制，我们还有两个重要的内容没有阐述，一个负数如何被表达，另一个，小数如何被表达。我们下一讲再逐步讲述。

关注计算机教育，关注骏扬工作室