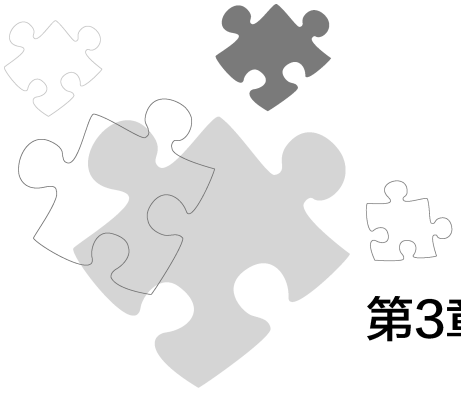


第3章　计算机中数据的表示与运算

本章学习目标

- 了解计算机内数值数据和非数值数据的组织格式和编码规则
- 掌握数值数据中指导计算机进行算数运算的理论基础——进位计数制、小数点的处理以及符号的表示
- 熟练掌握进位计数制、补码及浮点数

计算机所要处理的对象是数据信息，而指挥计算机操作的信息是控制信息。因此，可以把计算机的内部信息分为数据信息和控制信息两大类。其中，数据信息包括数值数据和非数值数据，非数值数据包括逻辑数据、字符数据（字母、符号、汉字）以及多媒体数据（图形、图像、声音）等，控制信息包括指令和控制字。

在计算机内部，信息的表示依赖于机器硬件电路的状态。数据采用什么表示形式，直接影响到计算机的性能和结构。应该在保证数据性质不变和工艺许可的条件下，尽量选用简单的数据表示形式，以提高机器的效率和通用性。

本章将使学生在了解计算机内数值数据和非数值数据的组织格式和编码规则的基础上，掌握计算机内数据信息的表示。其中，数值数据中指导计算机进行算术运算的理论基础——进制、补码、浮点数将作为学习的重点。

3.1　数 值 数 据

数值数据是指具有确定的数值，能表示其大小，在数轴上能够找到对应点的数据。

在现实生活中习惯采用十进制来表示数据，而计算机却用二进制表示信息。计算机采用二进制原因如下：

（1）二进制表示数据便于物理实现。在物理器件中，具有两个稳定状态的物理器件是很多的（如具有开关两个状态的灯、具有导通和截止两个状态的二极管、具有闭合和断开两个状态的开关等），恰好可以利用器件的两个状态对应表示二进制的 0 和 1 两个数字符号；而十进制具有从 0 到 9 的 10 个数字符号，要找到具有 10 个稳定状态的物理器件几乎是不可能的。这也是计算机采用二进制表示数据的最重要的理由。

(2) 二进制表示数据运算简单。用二进制表示数据,在做加法和乘法运算时,只需要记住 0 和 0、0 和 1、1 和 1,2×(2+1)/2 共 3 对数的和与积就可以了,而十进制运算却要考虑 10×(10+1)/2 共 55 对数的和与积。计算机采用二进制表示数据,其运算器件的电路实现起来十分简单。

(3) 二进制表示数据工作可靠。如果要采用十进制表示信息,那么,由于具有 10 个稳定状态的物理器件是不存在的,只能用器件的物理量来代表十进制的数字符号(如用电流量、电压的高低,假如用流过导线的电流量表示信息,0～0.1 安培代表"0"、0.1～0.2 安培代表"1"、0.2-0.3 安培代表"2"……);而二进制的表示,完全可以用物理器件的"质"来表示,比如用二极管的导通和截止或灯的亮与灭分别代表二进制的"0"和"1"。如果电路电压不够稳定,通过导线的电流量变化零点几安培是完全有可能的,而二极管导通与截止、灯的亮与灭的状态的跳变却不可能由电路电压的不稳定造成。也就是说用"质"来区分数字符号的二进制要比用"量"来区分数字符号的十进制工作起来稳定得多。

(4) 二进制表示数据便于逻辑判断。逻辑判断中,只有"是"和"非"两种状况,似是而非的状况是不存在的。正好可以用二进制的 0 和 1 分别代表逻辑判断的是与非。

当然,二进制表示数据也有它的不足,即它表示的数容量小。同样是 n 位数,二进制最多可表示 2^n 个数,而十进制可以表示 10^n 个数。如 3 位二进制数,可以表示从二进制的 000 到二进制的 111 一共 8 个数;而 3 位十进制数却可以表示从十进制的 000 到十进制的 999 共 1000 个数。

尽管二进制表示数据也有它的缺点,但基于它所带来的方便与简洁,计算机采用了二进制作为信息表示的基础。

十进制和二进制表示数据都是采用进位计数制。因此,要研究计算机内数值数据的表示,首先要研究进位计数的理论。不仅如此,还要考虑小数点和符号在计算机内如何处理。

所以,表示一个数值数据要有三个要素:进制、小数点和符号。

3.1.1　进位计数制及进制间的相互转换

本节介绍各种进位制结构的特性,以及它们之间的相互转换。

1. 进位计数制

为了协调人与计算机所用进制之间的差别,必须研究数字系统中各种进位制结构的特性,以及它们之间的相互转换,从中找出规律性的东西。下面,从我们习惯的十进制开始,系统研究进位计数制。

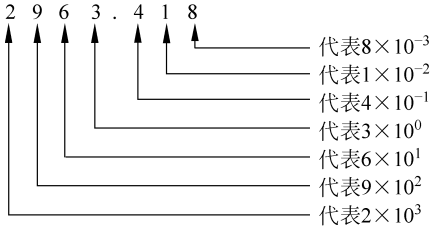
1) 十进制及十进制数

十进制采用逢十进一的进位规则表示数字。具体规则如下:

(1) 十进制用 0,1,…,9 十个数字符号分别表示 0,1,…,9 十个数。

(2) 当要表示的数值大于 9 时,用数字符号排列起来表示,表示规则如下:

- 数字符号本身具有确定的值。
- 不同位置的值由数字符号本身的值乘以一定的系数表示。
- 系数为以 10 为底的指数。



(3) 一个数的实际值为各位上的实际值总和。如：

$$1\,966\,298.735 = 1\times10^6 + 9\times10^5 + 6\times10^4 + 6\times10^3 + 2\times10^2 + 9\times10^1 + 8\times10^0 + 7\times10^{-1} + 3\times10^{-2} + 5\times10^{-3}$$

2) R 进制及 R 进制数

通过对十进制的总结，可以得出任意(R)进位数按逢 R 进一的规则表示数字的规则如下。

(1) R 进制用 0,1,…,$R-1$ 共 R 个数字符号分别表示 0,1,…,$R-1$ 共 R 个数。这里的 R 为数制系统所采用的数字符号的个数，被称为基数。

(2) 当要表示的数值大于 $R-1$ 时，用数字符号排列起来表示，表示规则如下。

- 数字符号本身具有确定的值。
- 不同位置的值由数字符号本身的值乘以一定的系数表示。
- 系数为以 R 为底的指数。
- 假设数字符号序列为

$$x_{n-1}x_{n-2}\cdots x_i\cdots x_1x_0.x_{-1}x_{-2}\cdots x_{-m}$$

通常在数字符号序列后面加上标注以示声明，如上面的 R 进制数表示为：$(x_{n-1}x_{n-2}\cdots x_i\cdots x_1x_0.x_{-1}x_{-2}\cdots x_{-m})_R$。$x_i$ 为 0 和 $R-1$ 之间的整数；x_i 的下标为数字符号的位序号，它所代表的值为 $x\times R^i$。系数 R^i($R^{\text{位序号}}$)被称为 x_i 所在位置的权。

(3) 一个数的实际值为各位上的实际值总和。如

$$x = x_{n-1}x_{n-2}\cdots x_i\cdots x_1x_0.x_{-1}x_{-2}\cdots x_{-m}$$

$$V(x) = x_{n-1}\times R^{n-1} + x_{n-2}\times R^{n-2} + \cdots x_i\times R^i + \cdots x_1\times R + x_0\times R^0 + x_{-1}\times R^{-1} + x_{-2}\times R^{-2} + \cdots x_{-m}\times R^{-m}$$

即

$$V(x) = \sum_{i=0}^{n-1} x_i\times R^i + \sum_{i=-1}^{-m} x_i\times R^i$$

$V(x)$表示 x 的值，m、n 为正整数。

3) 二进制及二进制数的运算

二进制采用逢二进一的进位规则表示数字，采用 0 和 1 两个数字符号。二进制的运算规则如下。

(1) 加法规则："逢 2 进 1"。

$$0+0=0 \quad 0+1=1+0=1 \quad 1+1=10$$

【例 3-1】 求 1010.110+1101.010。

解：

$$\begin{array}{r} 1010.110 \\ +\ 1101.010 \\ \hline 11000.000 \end{array}$$

结果：1010.110＋1101.100＝11000.000

（2）减法规则：“借1当2”。

$$0-0=0 \quad 1-0=1 \quad 1-1=0 \quad 10-1=1$$

【例3-2】 求11000.000－1101.010。

解：

$$\begin{array}{r} 11000.000 \\ -\ \ 1101.010 \\ \hline 1010.110 \end{array}$$

结果：11000.000－1101.010＝1010.110

（3）乘法规则。

$$0\times0=0 \quad 0\times1=0 \quad 1\times0=0 \quad 1\times1=1$$

由规则可以看出，二进制乘法要远比十进制乘法简单。

【例3-3】 求1010.11×1101.01。

解：

$$\begin{array}{r} 1010.11 \\ \times\ 1101.01 \\ \hline 10.1011 \\ 000.000 \\ 1010.11 \\ 00000.0 \\ 101011 \\ 101011 \\ \hline 10001110.0111 \end{array}$$

结果：1010.11×1101.01＝10001110.0111

在乘法运算的过程中，由于乘数的每一位只有0和1两种可能，那么，部分积也只有0和乘数本身两个值（不考虑小数点的位置）。根据这一特点，可以把二进制的乘法归结为移位和加法运算，即通过测试乘数的相应位是0还是1决定要加的部分积是0还是被乘数。

（4）除法规则。

【例3-4】 求10001110.0111÷1010.11。

解：

$$\begin{array}{r} 0000001101.01 \\ 101011\overline{)\ 1000111001.11} \\ \underline{\ \ 101011} \\ 0111000 \\ \underline{\ \ 101011} \\ 00110101 \\ \underline{\ \ 101011} \\ 001010\ 11 \\ \underline{\ \ 1010\ 11} \\ 0000\ 00 \end{array}$$

结果：10001110.0111÷1010.11＝1101.01

除法是乘法的逆运算，可以归结为与乘法相反方向的移位和减法运算。因此，在计算机

中，使用具有移位功能的加法/减法运算，就可以完成四则运算。

这里所举的例子恰好是可以整除的，最后的余数是 0000.00。如果是不可以整除的，那么在商达到了足够的精度后，最下面的部分就是余数。

4）八进制与十六进制

除了二进制与十进制外，八进制与十六进制由于其与二进制的特殊关系（$8=2^3$，$16=2^4$）也常被使用。一般在机器外部，为了书写方便，也为了减少书写错误，常采用八进制与十六进制。八进制的基数为 8，采用逢八进一的原则表示数据，权值为 $8^{位序号}$，数字符号为 0、1、2、3、4、5、6、7；十六进制的基数为 16，采用逢十六进一的原则表示数据，权值为 $16^{位序号}$，数字符号为 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。十六进制后面常加后缀 H 以示用于表示数字符号的 A、B、C、D、E、F 与字母的区别，如 13AH、E25 等。

二、八、十、十六进制之间的关系表示如表 3.1 所示。

表 3.1　四种进位计数制

二进制数	八进制数	十进制数	十六进制数
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
⋮	⋮	⋮	⋮
101101	55	45	2C
01101011	153	107	6B
110011010	632	410	19A

2. 进制间的相互转换

1）十进制转换为 R 进制

将十进制数转换为 R 进制数时，可以将数分为整数和小数两部分分别转换，然后再组合起来即可实现整个转换。

假设某十进制的数已转换为 R 进制的数，数字符号序列为：

$$x_{n-1}x_{n-2}\cdots x_1x_0.x_{-1}x_{-2}\cdots x_{-m}$$

（1）整数部分。

$$V(x)=x_{n-1}x_{n-2}\cdots x_i\cdots x_1x_0$$

$$=x_0+x_1\times R^1+x_2\times R^2+\cdots x_i\times R^i+\cdots x_{n-2}\times R^{n-2}+x_{n-1}\times R^{n-1}$$

若将其除以 R，可得：

$$V(x)/R = Q_0/R$$
$$= x_0 + R \times (x_1 + x_2 \times R^1 + \cdots + x_i \times R^{i-1} + \cdots + x_{n-1} \times R^{n-2})$$
$$= x_0 + Q_1$$

其中，x_0 为小于 R 的数，所以 x_0 为余数，Q_1 为商。

再将 Q_1 除以 R，可得

$$Q_1/R = x_1 + Q_2$$

x_1 为新得到的余数。

依此类推，$Q_i/R = x_i + Q_{i+1}$

如此循环下去，直到商为 0，就得到了从 x_0、x_1 一直到 x_{n-1} 的数字符号序列。

也就是说，要把十进制的整数转换为 R 进制的整数时，只需将十进制的整数连续地除以 R，其逐次所得到的余数即为从低位到高位的 R 进制的数字符号序列。

【例 3-5】 将 $(58)_{10}$ 转换为二进制的数。

	商	余数	
2	58		
	29	0	低位
	14	1	
	7	0	
	3	1	
	1	1	
	0	1	高位

由此可得 $(58)_{10} = (111010)_2$

【例 3-6】 将 $(58)_{10}$ 转换为八进制的数。

	商	余数	
8	58		
	7	2	低位
	0	7	高位

由此可得 $(58)_{10} = (72)_8$

【例 3-7】 将 $(58)_{10}$ 转换为十六进制的数。

	商	余数	
16	58		
	3	10——十六进制的A	低位
	0	3——十六进制的3	高位

由此可得$(58)_{10}=(3A)_{16}$

(2) 小数部分。

$$V(x)=0.x_{-1}x_{-2}\cdots x_{-m}$$
$$=x_{-1}\times R^{-1}+x_{-2}\times R^{-2}+\cdots+x_{-m}\times R^{-m}$$

若将其乘以 R,可得:

$$V(x)\times R=F_0\times R$$
$$=x_{-1}+x_{-2}\times R^{-1}+x_{-3}\times R^{-2}+\cdots+x_{-m}\times R^{-(m-1)}$$
$$=x_{-1}+F_1$$

其中,x_{-1} 为大于 1 的数,所以 x_{-1} 为整数,F_1 为小数部分。

再将 F_1 乘以 R,可得:

$$F_1\times R=x_{-2}+F_2$$

x_{-2} 为新得到的整数。

依此类推,$F_i\times R=x_{-(i+1)}+F_{i+1}$

如此循环下去,直到小数部分为 0 或商的精度达到要求为止,就得到了从 x_{-1}、x_{-2} 一直到 x_{-m} 的数字符号序列。

也就是说,要把十进制的小数转换为 R 进制的小数数时,只需将十进制的小数连续地乘以 R,其逐次所得到的整数即为从 x_{-1} 到 x_{-m} 的 R 进制小数的数字符号序列。

【例 3-8】 将$(0.5625)_{10}$转换为二进制的数。

	余数	
0.5625	整数	
× 2		
0.125	1	高位
× 2		
0.25	0	
× 2		
0.5	0	
× 2		
0.0	1	低位

由此可得$(0.5625)_{10}=(0.1001)_2$

【例 3-9】 将$(0.5625)_{10}$转换为八进制的数。

	余数	
0.5625	整数	
× 8		
0.5000	4	高位
× 8		
0.0	4	低位

由此可得$(0.5625)_{10}=(0.44)_8$

【例 3-10】 将$(0.5625)_{10}$转换为十六进制的数。

	余数	
0.5625		整数
× 16		
0.0000	9	高位

由此可得$(0.5625)_{10}=(0.9)_{16}$

【例 3-11】 将$(0.6)_{10}$转换为二进制的数。

	余数	
0.6		整数
× 2		
0.2	1	高位
× 2		
0.4	0	
× 2		
0.8	0	
× 2		
0.6	1	
× 2		
0.2	1	
× 2		
0.4	0	
× 2		
0.8	0	
× 2		
0.6	1	低位
× 2		
⋮	⋮	

由此可得$(0.6)_{10}=(0.10011001\cdots\cdots)_2$

小数部分在转换过程中出现了循环，永远也不可能出现0。就要根据需要的精度(或说计算机可能表示的精度)进行截止舍入。

假如要保留小数点后n位，那么至少要求出$n-1$位整数，然后进行舍入。

下面介绍一下二进制的舍入问题。

(3) 二进制的舍入。

二进制的舍入有两种方法。下面对比进行介绍。

- 0 舍 1 入法：被舍去的部分最高位如果为 1，就将其加到保留部分的最低位，否则直接舍去。
- 恒 1 法：被舍去的部分如果含有真正的有效数位(即 1)，就使保留部分的最低位为 1(不管其原来是 0 还是 1)。

【例 3-12】 将 0.101100101 保留到小数点后 5 位。

解：

按 0 舍 1 入法保留后的结果为 0.10110。

按恒 1 法舍入后的结果为 0.10111。

【例 3-13】 将 0.101111101 保留到小数点后 5 位。

解：

按 0 舍 1 入法保留后的结果为 0.11000。

按恒 1 法舍入后的结果为 0.10111。

由上可知，$(0.6)_{10}$ 转换为二进制后，若小数点后保留 5 位，则无论采用 0 舍 1 入法还是恒 1 法，其结果都为$(0.10011)_2$。

上面分别介绍了从十进制到 R 进制的整数和小数部分的转换。

从上面的例题中可得

$$(58.5625)_{10}=(111010.1001)_2$$

$$(58.5625)_{10}=(72.44)_8$$

$$(58.5625)_{10}=(3A.9)_{16}$$

2) R 进制转换为十进制

按照求值公式得

$$x=x_{n-1}x_{n-2}\cdots x_1x_0.x_{-1}x_{-2}\cdots x_{-m}$$

$$V(x)=\sum_{i=0}^{n-1}x_i\times R^i+\sum_{i=-1}^{-m}x_i\times R^i$$

基数为 R 的数，只要将各位数字与它所在位置的权 R^i($R^{\text{位序号}}$)相乘，其积相加(按逢十进一的原则)，即为相应的十进制数。

【例 3-14】 将$(21A.8)_{16}$、$(3A.9)_{16}$ 转换为十进制的数。

解：

$$\begin{aligned}V((21A.8)_{16})&=2\times16^2+1\times16^1+10\times16^0+8\times16^{-1}\\&=2\times256+1\times16+10\times1+8/16=538.5\end{aligned}$$

$$\begin{aligned}V((3A.9)_{16})&=3\times16^1+10\times16^0+9\times16^{-1}\\&=3\times16+10\times1+9/16=58.5625\end{aligned}$$

由此可得$(21A.8)_{16}=(538.5)_{10}$，$(3A.9)_{16}=(58.5625)_{10}$

【例 3-15】 将$(72.44)_8$ 转换为十进制的数。

解：

$$\begin{aligned}V((72.44)_8))&=7\times8+2\times8^0+4\times8^{-1}+4\times8^{-2}\\&=7\times8+2\times1+4/8+4/64=58.5625\end{aligned}$$

由此可得$(72.44)_8=(58.5625)_{10}$

【例 3-16】 将$(111010.1001)_2$转换为十进制的数。

解：

$$V((111010.1001)_2)=1\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0$$
$$+1\times2^{-1}+0\times2^{-2}+0\times2^{-3}+1\times2^{-4}$$
$$=32+16+8+2+0.5+0.625=58.625$$

由此可得$(111010.1001)_2=(58.5625)_{10}$

3）二、八、十六进制间的相互转换

二进制、八进制与十六进制之间的转换由于它们之间存在着权的内在联系而得到简化。由于$2^4=16$、$2^3=8$，所以，每一位十六进制数相当于四位二进制数，而每一位八进制数相当于三位二进制数。

（1）二进制转换为八进制或十六进制。

可将二进制的3位或4位一组转换为一位八进制或十六进制数。在转换中，位组的划分是以小数点为中心向左右两边延伸的，不足者补齐0。整数部分在高位补0，小数部分在低位补0。

【例 3-17】 将$(111010.1001)_2$转换为八进制的数。

解：

位组划分：　111　010 . 100 100

八进制数：　7　2　4　4

由此可得$(111010.1001)_2=(72.44)_8$

【例 3-18】 将$(111010.1001)_2$转换为十六进制的数。

解：

位组划分：　0011　1010 . 1001

十六进制数：　3　A　9

由此可得$(111010.1001)_2=(3A.9)_{16}$

（2）八进制或十六进制转换为二进制。

将每一位八进制或十六进制数转换为3位或4位的一组二进制数。

【例 3-19】 将$(D3A.94)_{16}$转换为二进制的数。

解：

十六进制数：　D　3　A . 9　4

↓　↓　↓　↓　↓

相应二进制数：1101 0011 1010 1001 0100

由此可得$(D3A.94)_{16}=(110100111010.100101)_2$

【例 3-20】 将$(376.52)_8$转换为二进制的数。

解：

八进制数：　3　7　6 . 5　2

↓　↓　↓　↓　↓

相应二进制数：011 111 110 101 010

由此可得$(376.52)_8=(11111110.10101)_2$

掌握了二、八、十六进制之间的内在联系，在它们之间的数制转换就不必用十进制作为

桥梁，既方便又不容易出错。

3.1.2 定点数与浮点数

在前面的内容中，介绍了进位计数制。在掌握了计算机内的二进制表示及二进制与其他进位计数制之间的相互转换之后，再来看一下小数点在计算机内是如何处理的。

数既可以是整数，也可以是小数。但是，计算机并不识别小数点。这就引出了小数点在机器内如何处理的问题。

计算机处理小数点的方式有两种：定点表示法和浮点表示法。定点表示法中，所有数的小数点都固定到有效数位间的同一位置；浮点表示法中，一个数的小数点可以在有效数位间任意游动。

小数点的位置固定不变的数称为定点数；小数点可以在有效数位间任意游动的数为浮点数。

采用定点表示法的计算机被称为定点机；采用浮点表示法的计算机称为浮点机。

假设一个二进制数 X，可以表示为

$$X = 2^E \times M$$

其中，E 是一个二进制整数，称为 X 的阶；2 为阶的基数；M 称为数 X 的尾数。尾数表示 X 的全部有效数字，而阶 E 指明该数的小数点的位置，阶和尾数都是带符号的数。在机器内部表示时，需要表示尾数和阶，至于基数和小数点，是不需要用任何设备表示的。关于正负号表示，将在后面进行介绍。

定点表示法中所有数的 E 值都相同。浮点表示法中一个数的 E 值就可以有多个，不同的 E 值，其尾数中小数点的位置就不同。

比如：

0.10、10.101 和 1011.011 这三个数，在八位字长的时候，如果小数点固定在第 4 位和第 5 位之间，那么它们分别为 0000.1000、0010.1010 及 1011.0110。

而浮点表示中，一个数就可以因小数点在有效数位间任意移动而有多种表示。如：

$$0.1011011 = 0.001011011 \times 2^2 = 101.1011 \times 2^{-3}$$

$$100.10000 = 0.1001 \times 2^3 = 1001 \times 2^{-1}$$

下面分别介绍定点数和浮点数。

1. 定点数

1）定点整数和定点小数

计算机内，通常采取两种极端的形式表示定点数。要么所有数的小数点都固定在最高位，称为定点的纯小数机；要么所有数的小数点都固定在最低位，称为纯整数机。

在定点的纯小数机中，若不考虑符号位，那么数的表示可归纳为：$0.x_{-1}x_{-2}\cdots x_{-m}$，其中 x_i 为各位数字符号，m 为数值部分所占位数；0 和小数点不占表示位，只是为了识别方便，在表示的时候才书写出来，而在机器中，小数点的位置是默认的，无须表示。

定点的纯整数机中，数的表示可归纳为 $x_{n-1}x_{n-2}\cdots x_i\cdots x_1x_0$，其中，$x_i$ 为各位数字符号；n 为数值部分所占位数。

2）定点数的表示范围

（1）n 位定点小数的表示范围。

最大数：0.11…11

最小数：0.00…01

范围：$2^{-n} \leqslant |x| \leqslant 1-2^{-n}$

（2）n 位定点整数的表示范围。

最大数：11…11

最小数：0

范围：$0 \leqslant |x| \leqslant 2^{n}-1$

如果运算的数小于最小数或大于最大数，则产生溢出。这里所说的溢出是指数据大小超出了机器所能表示的数的范围。

当数据大于机器所能表示的最大数时，就产生了上溢；而数据小于机器所能表示的最小数时，就产生了下溢。

一般下溢可当成 0 处理，不会产生太大的误差。如果参加运算的数、中间结果或最后结果产生上溢，就会出现错误的结果。因此，计算机要用溢出做标志迫使机器停止运行或转入出错处理程序。早期，程序员使用定点机进行运算要十分小心，常常通过选用比例因子来避免溢出的发生。

3）比例因子及其选取原则

在纯小数机或纯整数机中，若要表示的数不在纯小数或纯整数的范围之内，就要将其乘上一定的系数进行缩小或扩大为纯小数或纯整数以适应机器的表示，在输出的时候再做反方向调整即可。这个被乘的系数，称之为比例因子。

从理论上讲，比例因子的选择是任意的，因为尾数中小数点的位置可以是任意的。

比例因子不能过大。如果比例因子选择太大，将会影响运算精度。比如 $N=0.11$，机器字长为 4 位，则：

当比例因子为 2^{-1} 时，相乘后的结果为 0.011；

当比例因子为 2^{-2} 时，相乘后的结果为 0.001；

当比例因子为 2^{-3} 时，相乘后的结果为 0.000。

比例因子也不能选择过小。比例因子太小有可能使数据超出机器范围。如 $0.0110+0.1101=1.0011$。纯小数相加，产生了整数部分。

在选取比例因子时，必须要保证初始数据、预期的中间结果和运算的最后结果都在定点数的表示范围之内。

4）定点数的优缺点

定点数的最大优点是其表示简单，电路相对实现起来就容易，速度也比较快，但由于其表示范围有限，所以很容易产生溢出。

2. 浮点数

1）浮点数的两部分

在机器内部，浮点数由阶和尾数两部分构成。尾数部分必须为纯小数；而阶的部分必须为纯整数。

例如：$-0.101100\times 2^{-1011}$。

在表示浮点数的时候，除了要表示尾数和阶的数值部分，还要表示它们的符号。所以，要完整地表示一个浮点数，须包括阶的符号(阶符)、阶的数值(阶码)、尾数的符号(尾符)、尾数的数值(尾码)四部分。它们的顺序与位置在不同的机器中会有所不同，但必须完整表示这四部分。

2) 浮点数的表示范围

假设浮点数的尾数部分数值位为 n 位，阶的部分数值位为 l 位，那么，它的表示范围为

最大数：$0.11\cdots\cdots11\times2^{+11\cdots11}$

最小数：$0.00\cdots\cdots01\times2^{-11\cdots11}$

范围：$2^{-(2^l-1)}\times2^{-n}\leqslant|x|\leqslant(1-2^{-n})\times2^{+(2^L-1)}$

可以看出，浮点数的表示范围要远远超过定点数的表示范围。浮点数的最大数和最小数要比定点小数的最大数和最小数大或小 $2^{+(2^L-1)}$ 倍。

3) 浮点数的优缺点

从上面的形式可以看出，要表示一个浮点数，其实现要比定点数的复杂，因而速度也会有所下降；但它的表示范围和数的精度要远远高于定点数。

4) 浮点数的规格化

一个数所能保留的有效数位越多，其精度也就越高。

假如有下面三个浮点数：

$A=0.001011\times2^{00}$；

$B=0.1011\times2^{-10}$；

$C=0.00001011\times2^{+10}$。

这实际是同一个数的三个不同表示形式。

现在依据机器的要求，尾数的数值部分只能取 4 位，那么，在考虑舍入的情况下，三个数变为：

$A=0.0010\times2^{00}$；

$B=0.1011\times2^{-10}$；

$C=0.0000\times2^{+10}$。

A 只保留了两个有效数位，B 保留了全部的有效数位，而 C 却丢失了全部的有效数位。

如何尽可能地利用有限的空间保留尽可能多的有效数位呢？总结一下，可以发现：尾数部分的有效数位最高位的 1 越接近小数点，它的精度就越高。于是，要想办法使浮点数的尾数部分的最高位为 1 赢取最高的精度。这就涉及浮点数的规格化的问题。

所谓浮点数的规格化是指：在保证浮点数值不变的前提下，适当调整它的阶，以使它的尾数部分最高位为 1。

规格化浮点数尾数部分的数值特征为 0.1X……XX，即 $\frac{1}{2}\leqslant|m|<1$。

3. 定点数与浮点数的比较

一台计算机究竟采用定点表示还是浮点表示，要根据计算机的使用条件来确定。定点表示与浮点的比较见表 3.2。

表 3.2　定点表示与浮点表示的比较

比　较　项	定 点 表 示	浮 点 表 示
表示范围	较小	比定点范围大
精度	决定于数的位数	规格化时比定点高
运算规则	简单	运算步骤多
运算速度	快	慢
控制电路	简单，易于维护	复杂，难于维护
成本	低	高
程序编制	选比例因子，不方便	方便
溢出处理	由数值部分决定	由阶大小判断

从上面的介绍可以看出，定点数无论在数的表示范围、数的精度还是溢出处理方面，都不及浮点数。但浮点数的线路复杂，速度低。因此，在要求精度和数的范围的情况下，采用定点数表示方法往往更快捷、经济。

一台机器可以采用定点数表示，也可以采用浮点数表示；但同时只能采用一种方式。相应地，机器被称为定点机或浮点机。

3.1.3　数的符号表示——原码、补码、反码

前面讨论了数的进制和小数点的处理。要真正表示一个数，还要考虑它的符号。数的符号是如何被处理的呢？下面来讨论计算机内处理带符号数的二进制编码表示系统——码制。

1. 机器数与真值

二进制的数也有正负之分，如 A＝＋1011，B＝－0.1110，A 是一个整数，而 B 是一个负数。然而，机器并不能表示“＋”“－”。为了在计算机中表示数的正负，引入了符号位，即用一位二进制数表示符号。这被称之为符号位的数字化。

为了方便区分计算机内的数据和实际值，引入机器数和真值的概念。

真值：数的符号以通常的习惯用“＋”“－”表示。

机器数：数的符号数字化后用“0”“1”表示。

数的符号数字化后，是否参加运算？符号参加运算后数值部分又如何处理呢？计算机内有原码、补码、反码三种机器数形式，还有专门用于阶的移码形式。下面分别介绍。

2. 原码表示法

数的符号数字化后用“0”和“1”来表示，用户最自然的是想到用“0”和“1”在原来的“＋”“－”号位置上简单取代。这也正是原码表示法的基本思想。

在原码表示法中，用机器数的最高位表示符号，0 代表正数，1 代表负数；机器数的其余各位表示数的有效数值，为带符号数的二进制的绝对值。所以，原码又称符号-绝对值表示法。

【例 3-21】 原码表示法。

$$[+1010110]_{原}=01010110$$

$$[-1010110]_{原}=11010110$$

$$[+0.1010110]_{原}=0.1010110$$

$$[-0.1010110]_{原}=1.1010110$$

数的符号数值化后，就可把原码的定义数学化出来。下面系统研究原码。

1）原码的数学定义

n 字长定点整数(1 位符号位，$n-1$ 位数值位)的原码数学定义为

$$[x]_{原}=\begin{cases}x, & 0\leqslant x<2^{n-1}\\ 2^{n-1}+|x|, & -2^{n-1}<x\leqslant 0\end{cases}$$

n 字长定点小数的原码数学定义为

$$[x]_{原}=\begin{cases}x, & 0\leqslant x<1\\ 1+|x|, & -1<x\leqslant 0\end{cases}$$

【例 3-22】 求在 16 位字长的机器中，+1010110、−1010110 和+0.1010110、−0.1010110 的原码。

解：

$$[+0000000001010110]_{原}=0000000001010110$$

$$[-0000000001010110]_{原}=1000000001010110$$

$$[+0.101011000000000]_{原}=0.101011000000000$$

$$[-0.101011000000000]_{原}=1.101011000000000$$

数值部分不足 $n-1$ 位的时候，要在整数数值位前面和小数数值位后面补足 0。

2）关于零的原码

对于 0 来讲，正负 0 的原码是不同的。

$$[+00\cdots\cdots00]_{原}=000\cdots\cdots00$$

$$[-00\cdots\cdots00]_{原}=100\cdots\cdots00$$

3）已知原码求真值

在已知数原码的情况下，要求得它的真值也非常简单。

符号位为 1 的原码减去 1 或 2^{n-1} 就可得出真值的绝对值，符号位填上“−”就可得到真值。而符号位为 0 的原码，其本身就是真值的绝对值，只需把 0 改为“+”号或直接在前面加“+”(对于纯小数)即可。

【例 3-23】 已知原码求真值。

原码 10101100 的真值为

$$-(10101100-2^{8-1})=-(10101100-10000000)=-0101100$$

原码 1.101011000000000 的真值为

$$-(1.101011000000000-1)=-0.101011000000000$$

原码 00101100 的真值为+0101100，原码 0.101011000000000 的真值为+0.101011000000000。

也可以简单地把原码的符号位的“1”改为“−”、把“0”改为“+”而求得真值。

4）原码的运算

从上面的介绍可以看出，原码表示法只是简单地把“+”“−”号数字化成了“0”和“1”。其他的与真值是相同的。

所以，原码的运算与真值的运算规则是相同的，即把符号和数值部分分开来处理。

这对于乘除法来讲是十分适合的，因为两个数相乘除时，符号和绝对值就是分别处理

的。而对于加减法来讲，似乎就比较麻烦。两个数进行加减的时候，要先比较它们的绝对值，然后再决定是做加法还是减法。也就是说两个数相加，实际做的有可能不是加法而是减法，反之也一样。

那么，可不可以把加减法变得简单起来呢？比如只做加法而不必做减法。下面引进的补码表示法就大大简化了加减法。

3. 补码表示法

1）补的概念及模的含义

为了引进“补”的概念，先来看看日常使用的时钟。

时钟若以小时为单位，表盘上有 12 个刻度。时针每转动一周，其计时范围为 1～12 点。若把 12 点称作 0 点，计时范围为 0～11，共 12 个小时。

假设现在时针指向 3。那么，要想让时针指向 9，可有两种方法。

方法一：让时针顺时针转 6 个刻度。可表示为

$$3+6=9$$

方法二：让时针逆时针转 6 个刻度。表示为

$$3-6=9\text{（在共有 12 个数的前提下）}$$

再来看时针指向 8 的情形。如果把时针顺时针转动 7 个刻度，它指向 3；逆时针转 5 个刻度也会指向 3。可表示为：$8+7\equiv 8-5$（在共有 12 个数的前提下）。

为什么加一个数和减一个数会是等价的呢？是因为表盘只有 12 个刻度，是有限的。为了系统研究加与减等价的问题，再来看一个实例。

假设某二进制计数器共有 4 位，那么它能记录 0000～1111 即十进制的 0～15 共 16 个数。它能记录的数是有限的。如果它现在的内容是 1011，那么把它变为 0000 也有两种方法：

$$\begin{array}{r} 1011 \\ -\ 1011 \\ \hline 0000 \end{array} \qquad \begin{array}{r} 1011 \\ +\ 0101 \\ \hline 10000 \end{array}$$

可以得出：$1011-1011=0000$，$1011+0101=0000$（在只有 4 位二进制共可表示 2^4 即 16 个数的前提下）。第二个式子中最高位的 1 会因只有 4 位而自动丢失。可以表示为：$1011-1011\equiv 1011+0101$（在只有 16 个数的前提下）。

仔细观察上面的例子可以得出结论：在计数系统容量有限的前提下，加一个数和减一个数可以等价；并且它们的绝对值之和就等于这个计数系统的容量。如对于表盘来讲，$-6\equiv+6$，$-5\equiv+7$，6 与 6 之和以及 7 与 5 之和都为表盘刻度的总数 12；对于 4 位二进制计数器，1011 和 0101 之和为计数器的容量 16。

可以依此类推，假设计数系统的容量为 100，可有下面的式子存在：

$$97+7\equiv 97-93,25+67\equiv 25-33$$

前面的 3 个系统中，12、16 和 100 是计数系统的容量。在计算机科学中称为“模”。

所谓模就是指一个计量系统的量程或它所能表示的最多数。

在有了模的概念之后，上面的等价式子可以表示为

$$+7\equiv -5(\bmod 12)$$

$$-1011 \equiv +0101 (\bmod 2^4)$$
$$+7 \equiv -93 (\bmod 100)$$

这里的+7与-5、-1011与+0101以及+7与-93互称为在模12、2^4和100下的补数。

电子计算机系统是一种有限字长的数字系统。因此，它所有的运算都是有模运算。在运算过程中超过模的部分都会自然丢失。

补码的设计就是利用有模运算的这种自然丢失的特点，把减法变成加法，从而使计算机中的运算变得简单明了。

2）正数的补码和负数的补码

在有模运算中，加上一个正数（加法）或加上一个负数（减法）可以用加上一个负数或加上一个正数来等价。如果加一个负数在运算过程中用加一个正数来等价，就把减法变成了加法；反过来，如果一个正数用一个负数来等价，就把加法变成了减法。后者是我们所不希望的。所以，为了简化加减运算，在运算过程中，把正数保持不变，负数用它的正补数来代替。这就引出了补码的概念。将补码简单定义为

$$[x]_{补} = \begin{cases} x, & x \geqslant 0 \\ x \text{ 的正补数}, & x \leqslant 0 \end{cases}$$

考虑到互为补数的两个数的绝对值之和为计数系统的模，又可将补码进一步定义为：

$$[x]_{补} = \begin{cases} x, & x \geqslant 0 \\ 模 - |x|, & x \leqslant 0 \end{cases}$$

对于用二进制表示信息的计算机系统来讲，如果不考虑符号位，n位二进制数可表示的数从00…00到11…11共2^n个数，其模即为2^n。

3）补码的数学定义

n字长定点整数（1位符号位，$n-1$位数值位）的补码数学定义为

$$[x]_{补} = \begin{cases} x, & 0 \leqslant x < 2^{n-1} \\ 2^n - |x|, & -2^{n-1} \leqslant x < 0 \end{cases} \quad (\bmod 2^n)$$

n字长定点小数的补码数学定义为

$$[x]_{补} = \begin{cases} x, & 0 \leqslant x < 1 \\ 2 - |x|, & -1 \leqslant x < 0 \end{cases} \quad (\bmod 2)$$

这里，有必要对补码数学定义中的模进行说明。$n-1$位数值位可表示的数据个数为2^{n-1}个，其模应为2^{n-1}，但前面定义中的模却为2^n。

下面以4位二进制数为例。4位数中，符号占1位，3位数值位。如果以2^3为模，做如下运算：

$$\begin{array}{r} 1000 \\ -\ 101 \\ \hline 0011 \end{array}$$

可以看到，得到的数为0011。至于这个0011是-101的补数，还是+011本身，无从知晓。

倘若以2^4为模，运算如下：

$$\begin{array}{r} 10000 \\ -\quad 101 \\ \hline 1011 \end{array}$$

此时的结果1011，就可以与+011区分开来。

由此可知，用2^n做模求出来的负数的补数在最高位带出了特征位“1”，而用2^{n-1}做模却达不到这样的效果。小数补码的模由1变为2，也是同样的道理。

【例3-24】 已知真值求补码。

$$[+1010110]_{补}=01010110\ (\mathrm{mod}\ 2^8)$$

$$[-1010110]_{补}=10101010\ (\mathrm{mod}\ 2^8)$$

$$[+0.1110010]_{补}=0.1110010\ (\mathrm{mod}\ 2)$$

$$[-0.1110010]_{补}=1.0001110\ (\mathrm{mod}\ 2)$$

4）求补码的方法

由定义可知：正数的补码只要把真值的符号位变为0，数值位不变（n位字长，数值位应为$n-1$位。超过$n-1$位时要适当舍入，不足$n-1$位时，要在整数的高位或小数的低位补足0）即可求得。下面介绍的补码求法主要是针对负数而言。假设真值的数值位为$n-1$位。

方法一：按补码的数学定义求。

方法二：从真值低位向高位检查，遇到0的时候照写下来，直到遇到第一个1，也照写下来；第一个1前面的各位按位取反（0变成1，1变成0），符号位填1。

【例3-25】 已知：$X=1101100$，求X在8位机中的补码。

解：补码求法示意为

真值：$-\underbrace{1101}\quad\underbrace{100}$

↓　↓变反　↓不变

补码：10010100

$[-1101100]_{补}=10010100\ (\mathrm{mod}\ 2^8)$

方法三：对其数值位各位按位取反末位加1符号位填1。

【例3-26】 已知：$X=-0.1010100$，求X在8位机中的补码。

解：

真值：$-\underbrace{0.1010100}$

↓变反

.0101011

\+　　1

符号填1 .0101100

$[-0.1010100]_{补}=1.0101100\ (\mathrm{mod}\ 2)$

用同样的方法可以求得：$[-1101100]_{补}=10010100\ (\mathrm{mod}\ 2^8)$

计算机中常采用此种方法求补码。

5）关于零的补码

对于0来讲，正负0的补码是相同的。

$$[+00\cdots\cdots00]_{补}=000\cdots\cdots00$$

$$[-00\cdots\cdots00]_{补}=000\cdots\cdots00$$

6）已知补码求真值

先判断补码的最高位，若为0，则表明该补码为正数的补码，只要将最高位用正号表示，即得到其真值。若为1，则表示该补码为负数的补码，只需将其数值部分再求一次补，将最高位用负号表示，便得到其真值。

【例3-27】 已知：$[X]_{补}=10110110$，$[Y]_{补}=0.0101011$，求 X，Y。

解：$[X]_{补}$最高位为1，所以 X 为负数。数值部分按求补的方法变换后为1001010，因此，$X=-1001010$。

$[Y]_{补}$最高位为0，所以 Y 为正数，数值部分不变。$Y=+0.0101011$。

7）补码的运算

补码在运算的时候，符号和数值部分一起参加运算。补码的引进，为加减法带来了极大便利。限于篇幅，关于补码运算方法，此处不再赘述。

4. 反码表示法

补码表示法确实为加减法运算带来了方便。但是，按照定义求负数的补码时却用到了减法。下面来做一下变换，以整数为例。

令 $X=-X_{n-2}X_{n-3}\cdots X_1X_0$，则有

$$\begin{aligned}[X]_{补}&=2^n+X\\&=1000\cdots00-X_{n-2}X_{n-3}\cdots X_1X_0\\&=111\cdots11+00\cdots01-X_{n-2}X_{n-3}\cdots X_1X_0\\&=111\cdots11+00\cdots01-X_{n-2}X_{n-3}\cdots X_1X_0\\&=1\ \overline{X}_{n-2}\overline{X}_{n-3}\cdots\overline{X}_1\overline{X}_0+00\cdots01\end{aligned}$$

上式中的 $1\ \overline{X}_{n-2}\overline{X}_{n-3}\cdots\overline{X}_1\overline{X}$ 即为负数 X 的反码。

一个负数的反码即为对其原码除符号位以外的各位按位取反，或补码减去末位的1。

1）反码的数学定义

n 字长定点整数（1为符号位，$n-1$ 位数值位）的反码数学定义为

$$[x]_{反}=\begin{cases}x, & 0\leqslant x<2^{n-1}\\ 2^n-1-|x|, & -2^{n-1}<x\leqslant 0\end{cases}\quad \mathrm{mod}(2^n-1)$$

n 字长定点小数的反码的数学定义为

$$[x]_{反}=\begin{cases}x, & 0\leqslant x<1\\ 2-2^{-(n-1)}-|x|, & -1<x\leqslant 0\end{cases}\quad \mathrm{mod}(2-2^{-(n-1)})$$

反码又被称为“1”补码。

【例3-28】 求16位字长的机器中，+1010110、−1010110和+0.1010110、−0.1010110的

反码。

解：

$$[+0000000010101 10]_{反}=0000000001010110$$

$$[-0000000010101 10]_{反}=1111111110101001$$

$$[+0.10101100000000 0]_{反}=0.101011000000000$$

$$[-0.10101100000000 0]_{反}=1.010100111111111$$

数值部分不足 $n-1$ 位的时候，要在整数数值位前面和小数数值位后面补足 0。

2）关于零的反码

对于 0 来讲，正负 0 的反码是不同的。

$$[+00\cdots\cdots00]_{反}=000\cdots\cdots00$$

$$[-00\cdots\cdots00]_{反}=111\cdots\cdots11$$

3）已知反码求真值

在已知数的反码的情况下，要求得它的真值也非常简单。

可以按照公式，符号位为 1 的反码用 1.11……11 或 11……11（n 个 1）减去反码就可得出真值的绝对值，符号位填上“－”就可得到真值。而符号位为 0 的反码，其本身就是真值的绝对值，只需把 0 改为“＋”号或直接在前面加“＋”（对于纯小数）即可。

【例 3-29】 已知反码求真值（整数）。

解：

反码 10101100 的真值为

$$-(11111111-10101100)=-1010011$$

反码 1.101011000000000 的真值为

$$-(1.111111111111111-1.101011000000000)=-0.010100111111111$$

反码 00101100 的真值为＋0101100，反码 0.101011000000000 的真值为＋0.101011000000000。

也可以把负数反码的符号位的“1”改为“－”、把数值部分各位按位取反来求得真值。

【例 3-30】 已知反码求真值（小数）。

解：

反码：1.1010100

变反

真值：-0.0101011

反码：11010010

变反

真值：-0101101

4）反码的运算

反码在运算的时候，符号和数值部分一起参加运算。关于反码运算方法，此处不予

详述。

5. 原码、补码、反码三种机器数的比较

1）正数与负数的不同码制

三种码制最高位均为符号位。

真值为正时：三种码制相同。符号位为0，数值部分与真值同。

真值为负时：符号位均为1。原码的数值部分与真值同，反码为原码的各位按位取反，补码为反码的末位加1。

2）数的范围

0的表示原、反码各有两种，补码只有一种。

原、反码表示的正、负数范围相对于0对称。

$$\text{对于整数：} -2^{n-1} < X < +2^{n-1}$$

$$\text{对于小数：} -1 < X < +1$$

补码表示的负数范围比正数范围大，多表示一个最小负数100……00，真值为-2^{n-1}或-1，无可被表示的最大正数与之对应。

$$\text{对于整数：} -2^{n-1} \leqslant < X < +2^{n-1}$$

$$\text{对于小数：} -1 \leqslant X < +1$$

3）运算规则

补码、反码的符号位与数值位一起参加运算，原码符号位与数值位分开处理。

6. 综合例题

计算机要表示一个数值数据，需要考虑三个方面的因素：数制的处理，小数点的处理和符号的处理。计算机内的数值数据用二进制来表示；小数点的处理方法有定点表示法和浮点表示法；符号的处理有原码表示法、补码表示法、反码表示法以及阶的移码表示法。表3.3是8字长计算机中常用数据的各种码制对照表。

表3.3　8字长计算机常用数据的各种码制对照表

真　值	原码表示	反码表示	补码表示	移码表示
127	01111111	01111111	01111111	11111111
126	01111110	01111110	01111110	11111110
1	00000001	00000001	00000001	10000001
+0	00000000	00000000	00000000	10000000
−0	10000000	11111111	00000000	10000000
−1	10000001	11111110	11111111	01111111
−127	11111111	10000000	10000001	00000001
−128	无法表示	无法表示	10000000	00000000

下面，通过两个具体的实例来体会计算机中数值数据的表示。

【例3-31】 已知$X=+13/128$，试用二进制表示成定点数和浮点数（尾数数值部分取7位，阶码部分取3位，阶符、尾符各占1位），并写出它们在定点机和浮点机中的机器数形式。

解：

$$X=+1101/2^7=+0.0001101$$

定点：$X=+0.0001101$

规格化浮点：$X=+0.1101000\times2^{-011}$

定点机中：

$$[X]_{原}=[X]_{补}=[X]_{反}=00001101$$

浮点机中：

$$[X]_{原}=1011\quad 01101000$$
$$[X]_{补}=1101\quad 01101000$$
$$[X]_{反}=1100\quad 01101000$$

【例 3-32】 已知 $X=-17/64$，试用二进制表示成定点数和浮点数，要求同上例。

解：

$$X=-10001/2^{6}=-0.010001$$

定点：$X=-0.0100010$

规格化浮点：$X=-0.1000100\times2^{-001}$

定点机中：

$$[X]_{原}=10100010$$
$$[X]_{补}=11011110$$
$$[X]_{反}=11011101$$

浮点机中：

$$[X]_{原}=1001\quad 11000100$$
$$[X]_{补}=1111\quad 10111100$$
$$[X]_{反}=1110\quad 10111011$$

注：因小数点可以在有效数位间任意位置，一个数的浮点表示可以有很多种。这里只表示出规格化后的形式。

另外，在机器中，浮点数的阶、尾两部分不一定都用同一种码制表示。这里如此列出，只是为了方便。

3.2　非数值数据

计算机不仅能够对数值数据进行处理，还能够对逻辑数据、字符数据（字母、符号、汉字）以及多媒体数据（图形、图像、声音）等非数值数据信息进行处理。非数值数据是指不能进行算术运算的数据。

3.2.1　逻辑数据的表示与逻辑运算

1. 逻辑数据的表示

逻辑数据是用二进制代码串表示的参加逻辑运算的数据。其主要应用于逻辑判断。

逻辑数据由若干位无符号二进制代码串组成，位与位之间没有权的内在联系，只进行本位操作。每一位只有逻辑值：“真”或“假”。一般情况下，0 对应逻辑假，1 对应逻辑真，如 10110001010。从表现形式上看，逻辑数据与数值数据区别不大，要由指令来识别是否为逻辑数据。

2. 逻辑运算

逻辑运算包括逻辑与、逻辑或、逻辑非、逻辑异或、逻辑同或、逻辑移位等。

1）逻辑与运算

逻辑与运算又被称为逻辑乘。其运算规则为

$$0 \wedge 0=0,\quad 0 \wedge 1=0,\quad 1 \wedge 0=0,\quad 1 \wedge 1=1$$

逻辑与运算可以简单描述为：当且仅当两个操作数都为逻辑真时，逻辑与运算的结果才为真；其他情况时，运算结果均为假。

2）逻辑或运算

逻辑或运算又被称为逻辑加。其运算规则为

$$0 \vee 0=0,\quad 0 \vee 1=1,\quad 1 \vee 0=1,\quad 1 \vee 1=1$$

逻辑或运算可以简单描述为：当且仅当两个操作数都为逻辑假时，逻辑或运算的结果才为假；其他情况时，运算结果均为真。

3）逻辑非运算

逻辑非运算又被称为逻辑取反。其运算规则为

$$\overline{1}=0,\quad \overline{0}=1$$

逻辑非运算可以简单描述为：非假即真，非真即假。

4）逻辑异或运算

逻辑异或运算的规则为

$$0 \oplus 0=0,\quad 0 \oplus 1=1,\quad 1 \oplus 0=1,\quad 1 \oplus 1=0$$

逻辑异或运算可以简单描述为：两个操作数相同时，异或运算结果为假；两个操作数不同时，异或运算结果为真。

5）逻辑同或运算

逻辑同或运算的规则为

$$0 \odot 0=1,\quad 0 \odot 1=0,\quad 1 \odot 0=0,\quad 1 \odot 1=1$$

逻辑同或运算与逻辑异或运算结果正好相反：两个操作数相同时，同或运算结果为真；两个操作数不同时，同或运算结果为假。

6）逻辑移位运算

逻辑移位运算分为逻辑左移和逻辑右移。逻辑左移时，所有位向左移动移位，最高位丢弃，最低位补0；逻辑右移时，所有位向右移动移位，最低位丢弃，最高位补0。

【例 3-33】 计算11010001和01010000两个数据的逻辑与、逻辑或、逻辑异或、逻辑同或的结果。

解：根据逻辑运算规则，4种逻辑运算结果如下：

11010001	11010001	11010001	11010001
$\wedge$ 01010000	$\vee$ 01010000	$\oplus$ 01010000	$\odot$ 01010000
01010000	11010001	10000001	10000001

【例 3-34】 计算11010001的逻辑非、逻辑左移一位、逻辑右移两位的结果。

解：11010001逻辑非的结果为：00101110；

11010001逻辑左移一位的结果为：10100010；

11010001逻辑右移两位的结果为：00110100。

用计算机处理数据时，若需要把数据中的某些位变为0而其他位保持不变时，可以用与运算来实现；若需要把数据中的某些位变为1而其他位保持不变时，可以用或运算来实现；若需要把数据中的某些位取反而其他位保持不变时，可以用异或运算来实现。

3.2.2　十进制数字编码

计算机内毫无例外地都使用二进制数进行运算，但通常采用八进制和十六进制的形式读写。

计算机技术专业人员理解这些数的含义是没问题的，但非专业人员理解起来就不那么容易了。由于日常生活中，人们最熟悉的数制是十进制，因此专门规定了一种二进制的十进制码，简称BCD码(Binary-Coded Decimal)，它是一种以二进制表示十进制数的编码。

BCD编码是用4位二进制码的组合代表十进制数的0、1、2、3、4、5、6、7、8、9十个数字符号。4位二进制数码有16种组合，原则上可任选其中的10种作为代码，分别代表10个数字符号。因$2^4=16$，而十进制数只有十个不同的数码，故16种组合中选取10组，可组成多种BCD码方案。

根据4位代码中每位是否有确定的位权进行划分，分为有权码和无权码两类。

在有权码中使用最普遍的是8421码，即4个二进制位的位权从高到低分别为8、4、2、1。有权码还有2421码、5211码及4311码。无权码中常用的是余3码和格雷码。余3码是在8421码的基础上，把每个代码加0011而构成。格雷码的编码规则是相邻的两个代码之间只有一位不同。常用的二-十进制编码见表3.4。

表3.4　常用的二-十进制编码表

十进制数	8421码	2421码	5211码	4311码	余3码	格雷码
0	0000	0000	0000	0000	0011	0000
1	0001	0001	0001	0001	0100	0001
2	0010	0010	0011	0011	0101	0011
3	0011	0011	0101	0100	0110	0010
4	0100	0100	0111	1000	0111	0110
5	0101	1011	1000	0111	1000	0111
6	0110	1100	1010	1011	1001	0101
7	0111	1101	1100	1100	1010	0100
8	1000	1110	1110	1110	1011	1100
9	1001	1111	1111	1111	1100	1101

【例3-35】 求十进制数的编码。

解：

$$
\begin{aligned}
&(1945.628)_{10}\\
=&(0001100101000101.011000101000)_{8421}\\
=&(0100110001111000.100101011011)_{余3码}\\
=&(0001110101100111.010100111100)_{格雷码}
\end{aligned}
$$

3.2.3 字符数据编码

字符数据是指用二进制代码序列表示的字母、数字、符号等的序列。字符数据主要用于主机与外设之间进行信息交换。

字符数据也是一种编码。编码最早源于电报的明码。例如，北京为 0554 0019，4 位十进制数表示一个汉字。在计算机中，关于字符数据的编码包括表示最基本字符的 ASCII 字符编码、汉字及其他文字编码等。

1. ASCII 字符编码

我们在使用计算机进行输入/输出操作及各种动作的时候，基本上要用到 95 种可打印字符（能用键盘输入并可显示的字符，包括大小写英文字母 A～Z；数字符号 0～9；标点符号；特殊字符）和 32 种控制字符（不可打印的 Ctrl、Shift、Alt 等）。需将它们进行数字化处理之后才能输入计算机。数字化处理后的数据即为字符数据。

目前，国际上广泛使用的字符是美国信息标准码，简称 ASCII 码。每个 ASCII 字符用 7 个二进制位编码，共可表示 $2^7=128$ 个字符。ASCII 字符表如表 3.5 所示。

表 3.5 ASCII 码字符表

$b_3\,b_2\,b_1\,b_0$	$b_6\,b_5\,b_4$							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	、	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYM	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESU	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	↑	n	Esc
1111	SI	US	/	?	O	↓	o	Del

为了构成一个字节，ASCII 码允许加一位奇偶校验位，一般加在一个字节的最高位，用作奇偶校验。通过对奇偶校验位设置“1”或“0”状态，保持 8 位字节中的“1”的个数总是奇数（称奇校验）或偶数（称为偶校验），用以检测字符在传送（写入或读出）过程中是否出错。

表 3.5 中的 ENQ（查询）、ACK（肯定回答）、NAK（否定回答）等，是专门用于串行通信的控制字符。

在 ASCII 码字符表中查找一个字符所对应的 ASCII 码的方法是：向上找 $b_6\,b_5\,b_4$ 向左

找 $b_3b_2b_1b_0$。例如，字母 J 的 ASCII 码中的 $b_6b_5b_4$ 为 100B(5H)，$b_3b_2b_1b_0$ 为 1010B(AH)。因此，J 的 ASCII 码为 1001010B(5AH)。

ASCII 码也是一种 0，1 码，把它们当作二进制数看待，称为字符的 ASCII 码值。用它们代表字符的大小，可以对字符进行大小比较。

1981 年，我国参照 ASCII 码颁布了国家标准《信息处理交换用汉字编码字符集——基本集》，与 ASCII 码基本相同。

2. 汉字编码

由于信息在计算机中都是以二进制形式存在的，若想让计算机能够存储和处理汉字，也必须对汉字进行编码，为每个汉字分配一个唯一的二进制代码。汉字信息处理必须考虑汉字的输入、存储以及显示。汉字编码包括将汉字输入计算机的汉字输入码、将汉字存储在计算机内的汉字机内码以及将汉字在输出设备上显示出来的汉字字形码。

计算机进行汉字处理的过程如图 3.1 所示。

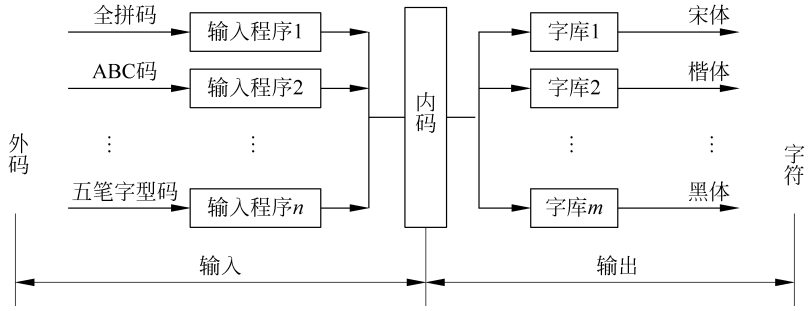


图 3.1　计算机进行汉字处理的过程

用户按照一定的输入方法(拼音、偏旁部首、数字等)输入汉字信息，相应的输入程序将输入码转换成计算机机内码存储在计算机内；需要输出的时候，相应的程序再通过调用相应字库里的字模信息将机内码转换成字形码，并以汉字的形式显示出来。

内码与字符是一一对应的，而外码(输入码)与内码具有多对一的关系，字库(输出码)与内码也是多对一的关系。

1) 汉字输入码

将汉字输入计算机，有模式识别输入和汉字编码输入两种途径。

(1) 通过模式识别输入。模式识别是指对表征事物或现象的各种形式的(数值的、文字的和逻辑关系的)信息进行处理和分析，以及对事物或现象进行描述、辨认、分类和解释的过程，是信息科学和人工智能的重要组成部分。模式识别法输入即指计算机通过“视觉”“听觉”及“触觉”装置(如扫描仪、话筒、手写板、触摸屏等)提取相应的输入信息，再经过模式识别软件的处理、辨识与解释，形成相应的汉字并转换成机内码的过程。

(2) 通过汉字编码输入。用户借助输入设备(通常为键盘)并根据一定的编码方法通过相应的输入方法将汉字输入计算机。如何利用标准英文键盘输入汉字，目前已提出的方法约有 2000 种，可分为以下几类：

- 汉字拼音输入码，如全拼码、双拼码等；
- 汉字字形编码，如五笔字型码、首尾码、101 码等；

- 汉字音形编码；
- 汉字数字编码，如区位码、电报码等。

每一种汉字输入程序的基本功能都是将输入码转换成机内码。

2）汉字机内码

机内码是指计算机系统内部处理和存储字符时使用的代码。

由于英文处理比较简单，所以其机内码就是 ASCII 码。ASCII 码用 8 位二进制数表示一个字符，其中第 1 位是奇偶校验位。

汉字被许多国家和地区所使用，所以目前存在多种汉字机内码标准。常用的汉字编码标准有 GB 2312—1980、GB 18030—2000 和 BIG5 码。这些码简称国标码，经过适当的转换（以区别于基本字符编码 ASCII 码），称为汉字在计算机内存储的机内码。

（1）GB 2312—1980 码。GB 2312—1980 码也称为国标交换码、国标码和 GB 码，是中华人民共和国国家汉字信息交换用途码，全称《信息处理交换用汉字编码字符集——基本集》，由国家标准总局于 1981 年发布。GB 2312 共收录 6763 个汉字及 682 个图形字符。

（2）GB 18030—2000 码。GB 18030—2000 码是最新的汉字编码字符集国家标准，向下兼容 GBK 和 GB 2312—1980 标准。GB 18030 编码根据 Unicode 标准对 GBK 进行了扩充，在双字节接触上对生僻字采用四字节进行编码，共收录了 27 533 个汉字，还收录了日文、朝鲜语和中国藏族、蒙古族等少数民族的文字。

（3）BIG5 码。BIG5 码是通行于我国台湾、香港地区的繁体汉字编码方案，也称为大五码。它是双字节编码方案，共收录了 13 461 个汉字和符号。

总之，不管是用哪一种输入码输入的汉字，以什么编码标准方案进行的转换，在计算机内部存储时，都使用机内码。这也是为什么用一种汉字输入法输入的文档也可以用另一种汉字输入法对其进行修改的原因之所在。

3）汉字字形码

字库也称为字形码或字模码，与机内码是多对一的关系，一个机内码对应多个字模码，用于输出不同的字形，可以将内码转换成各种不同的字形。或者说，不同的字体有不同的字库。简单地说，输出时机内码作为字库的地址，选定了字体后，每一个机内码驱动一个字将其输出。全部汉字字形的集合叫作汉字字形库（简称汉字库）。汉字的字库分为点阵字库和矢量字库两类。

（1）点阵字库。点阵字库就是将每个汉字（包括一些特殊符号）看成是一个矩形框内的一些横竖排列的点的集合，有笔画的位置用黑点表示，无笔画的位置用白点表示，分别对应二进制的 1 和 0，将这些点阵信息记录下来，就成了字库。一般汉字系统中汉字字形的点阵规格有 16×16、24×24、48×48 几种。点阵越大，每个汉字的笔画越清晰，打印质量也就越高。点阵字库显示或打印速度快，但占用存储空间大，且不能缩放。假设每个汉字由 16×16 点阵组成，那么，需要占 32 个字节；24×24 点阵需要占 72 个字节（每个汉字字模占用的字节数＝点阵行数×点阵列数/8）。点阵字库常用于针式打印机和屏幕显示。

（2）矢量字库。保存的是汉字的笔画轮廓信息，包括组成汉字的每一个笔画的起点坐标、终点、半径、弧度等。在显示和打印这类字库的字形，要经过一系列的数学运算才能输出结果。矢量字库的汉字显示速度慢，但占用存储空间小，汉字可任意缩放。缩放后的笔画轮廓仍能保持圆滑、流畅。矢量字库多用于激光打印机和绘图仪。

3.2.4　多媒体数据

多媒体数据包括文字、声音、图形、图像及视频数据。文字可以理解为字符。在前面已经叙述过了，此处不再赘述。

1. 声音编码

1）音调、音强和音色

声音是通过声波改变空气的疏密度，引起鼓膜振动而作用于人的听觉的。从听觉的角度，音调、音强和音色称为声音的三要素。

音调决定于声波的频率。声波的频率越高，则声音的音调越高；声波的频率越低，则声音的音调越低。人的听觉范围为 20Hz～20kHz。

音强又称响度，决定于声波的振幅。声波的振幅越高，则声音越强；声波的振幅越低，则声音越弱。

音色决定于声波的形状。混入音波基音中的泛音不同，得到不同的音色。

图 3.2 形象地说明了两种不同的声音的三要素的不同。

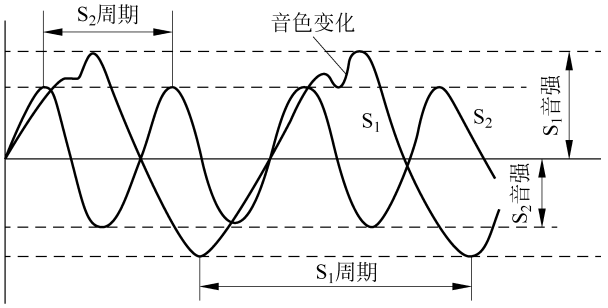


图 3.2　两种不同声音的三要素

显然，声音媒体的质量主要决定于它的频宽。

2）波形采样量化

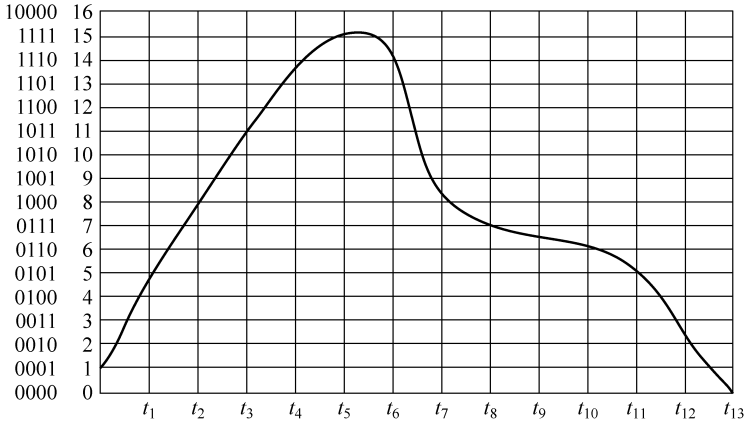
任何用符号表示的数字都是不连续的。如图 3.3 所示，波形的数字化过程是将连续的波形用离散的(不连续的)点近似代替的过程。在原波形上取点，称为采样。用一定的标尺确定各采样点的值(样本)，称为量化(见图 3.3 中的粗竖线)。量化之后，很容易将它们转换为二进制(0，1)码(见图 3.3 中的表)。

3）采样量化的技术参数

一个数字声音的质量，决定于下列技术参数。

(1) 采样频率。

采样频率即一秒内的采样次数，它反映了采样点之间的间隔大小。间隔越小，丢失的信息越少，数字声音越细腻逼真，要求的存储量也就越大。由于计算机的工作速度和存储容量有限，而且人耳的听觉上限为 20kHz，所以采样频率不能也不需要太高。根据奈奎斯特采样定律，只要采样频率高于信号中的最高频率的两倍，就可以从采样中恢复原始的波形。因此，40kHz 以上的采样频率足以使人满意。目前，多媒体计算机的标准采样频率有 3 个：44.1kHz、22.05kHz 和 11.025kHz。CD 唱片采用的是 44.1kHz。



样点	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}
幅值	0101	1000	1011	1110	1111	1110	1000	0111	0110	0110	0101	0010	0000

图 3.3　波形的采样与量化

(2) 测量精度。

测量精度是样本在纵向方向的精度,是样本的量化等级,它通过对波形纵向方向的等分而实现。由于数字化最终要用二进制数表示,所以常用二进制数的位数表示样本的量化等级。若每个样本用 8 位二进制数表示,则共有 $2^8=256$ 个量级。若每个样本用 16 位二进制数表示,则共有 $2^{16}=65\,536$ 个量级。量级越多,采样越接近原始波形;数字声音质量越高,要求的存储量也越大。目前,多媒体计算机的标准采样量级有 8 位和 16 位两种。

(3) 声道数。

声音记录只产生一个波形,称为单声道。声音记录产生两个波形,称为立体声双声道。立体声比单声道声音丰满、空间感强,但需两倍的存储空间。

2. 图形与图像编码

图形(Graphic)和图像(Image)是画面在计算机内部的两种表示形式。

图像表示法是将原始图画离散成 $m\times n$ 个像点-像素组成的矩阵。每一个像素根据需要用一定的颜色和灰度表示。对于 GRB 空间的彩色图像,每个像素用 3 个二进制数分别表示该点的红(R)、绿(G)、蓝(B)3 个彩色分量,形成 3 个不同的位平面。此外,每一种颜色又可以分为不同的灰度,当采用 256 个灰度等级时,各位平面的像素位数都为 8 位。这样,对于 RGB 颜色空间,且具有 256 个灰度级别时,要用 24 位二进制数表示(称颜色深度为 24)。24 位共可以表示 2^{24} 种颜色。图像表示常用于照片以及汉字字形的点阵描述。

图形表示是根据图画中包含的图形要素——几何要素(点、线、面、体)、材质要素、光照环境和视角等进行描述表示。图形表示常用于工程图纸、地图以及汉字字形的轮廓描述。

3.3　数据校验编码

计算机系统工作过程中,由于脉冲噪声、串音、传输质量等原因,有时在信息的形成、存取、传送中会发生错误。为减少和避免这些错误,一方面要提高硬件的质量,另一方面可以

先对要传输的数据进行编码，使编码后的数据具有检测或者更正错误的功能，再将编码后的数据进行传输。这种具有发现错误，甚至能够更正少量错误的数据编码，称为抗干扰码或数据校验码。

数据校验码的工作原理是：按一定的规律在有用信息的基础上再附加上一些冗余信息，使编码在简单线路的配合下能发现错误、确定错误位置甚至自动纠正错误。

通常，一个 k 位的信息码组应加上 r 位的校验码组，组成 $k+r$ 位抗干扰码字(在通信系统中称为一帧)。

在现代计算机硬件制造和数据通信领域广泛采用数据编码校验技术来提高数据的可靠性。常用的校验技术有奇偶校验码、循环冗余校验码和汉明校验码等。其中，奇偶校验码和循环冗余校验码主要用于检测数据错误，奇偶校验码主要用于单个字符的错误检测，循环冗余校验码主要用于一批数据的错误检测；而汉明校验码不但可以检测数据错误，还可以纠正错误。

3.3.1　奇偶校验码

奇偶校验码是奇校验码和偶校验码的统称，是一种最基本的检错码。它是在被传输的 n 位二进制信息元上额外加上 1 位二进制校验位。如果是奇校验码，在附加上 1 位校验位后，码长为 $n+1$ 位的码字中 1 的个数必须保证为奇数个；如果是偶校验码，则在附加上 1 位校验位后，码长为 $n+1$ 位的码字中 1 的个数必须保证为偶数个。当采用奇偶校验码时，通信双方必须采用统一的校验方式(奇校验或偶校验)才可正常通信。表 3.6 给出了几个奇偶校验的例子。

表 3.6　奇偶校验码示例

原 始 数 据	奇校验编码结果	偶校验编码结果
0101110	10101110	00101110
0010000	00010000	10010000
1010110	11010110	01010110

表 3.6 中编码结果中最高位为附加的校验位，低 7 位为有效数据位，在数据传输和存储前计算校验信息时，有效数据位中的信息要保持不变。

奇偶校验码可以检测出被传输的数据在传输过程中是否出现了差错。当数据进行通信时，需要将位信息元和 1 位校验元构成的 $n+1$ 位数据码一起发送，接收方收到数据后重新计算数据中 1 的个数，由此可知道数据是否正确。例如，通信系统双方约定采用奇校验，发送方发送的每个码字中 1 的个数一定都是奇数，如果某次通信时因受到外界干扰发生错误，数据中某个位由 0 变成 1，或者由 1 变成 0，则接收到的数据中 1 的个数就变成了偶数，接收方即可知道数据通信出错。

奇偶校验是一种有效地检测单个错误的方法，但无法判断错误出现的位置。之所以将注意力集中在检测或者纠正单个错误，主要是因为在现代数字通信中发生单个错误的概率要比发生两个或多个错误的概率大得多，要检测或者纠正多位错误，首先要解决单个错误。用奇校验码来检测单个错误，在低速、小批量数据通信情况下会取得良好的效果。另外，奇偶校验码的编码效率很高，$n+1$ 的码字中有 n 位有效数据，其通信效率可达到 $n/(n+1)$，

随 n 的增大而趋近于 1。

在数字信息传输中，奇偶校验码的编码生成以及编码校验可以用软件实现，也可用硬件电路实现。

假设有效 n 位数据为 $X=X_0X_1\cdots X_{n-1}$，校验位为 C，则 C 可以由下式计算生成：

$$C=X_0\oplus X_1\oplus\cdots\oplus X_{n-1} \quad \text{偶校验}$$

$$C=X_0\oplus X_1\oplus\cdots\oplus X_{n-1}\oplus 1 \quad \text{奇校验}$$

其中，⊕表示异或运算。

接收方收到数据后，可以用下列验证方程进行校验，若满足方程，则数据正确；若不满足方程，则接收到的数据有错误。

偶校验方程：$C\oplus X_0\oplus X_1\oplus\cdots\oplus X_{n-1}=0$

奇校验方程：$C\oplus X_0\oplus X_1\oplus\cdots\oplus X_{n-1}=1$

奇偶校验码目前广泛应用于计算机中内存的数据读/写校验以及单个 ASCII 码字符传输过程中的数据校验。

3.3.2 汉明校验码

汉明校验码是 Richard Hamming 于 1950 年提出的，目前被广泛采用的一种很有效的校验方法。它只要增加少数几个校验位，就能提供多位检错信息，以指出最大可能是哪位出错，从而将其纠正。

一般数据校验的基本原理是，在合法的数据编码之间加入一些非法数据编码。发送时，只发送合法的数据编码。如果数据传输过程中出错，将变为非法数据编码，这样接收方即可检测出来。合理安排非法编码数量和编码规则，可以提高检测错误能力，还可以达到纠正错误的目的。

汉明校验码的实现原理是，在 k 个数据位之外加上 r 个校验位，从而形成一个 $k+r$ 位的新的码字，使新的码字的码距比较均匀地拉大。把数据的每一个二进制位分配在几个不同的偶校验位的组合中，当某一位出错后，就会引起相关的几个校验位的值发生变化，这不但可以发现出错，还能指出是哪一位出错，为进一步自动纠错提供了依据。

假设为 k 个数据位设置 r 个校验位，则校验位能表示 2^r 个状态，可用其中的一个状态指出“没有发生错误”，用其余的 2^r-1 个状态指出错误发生在哪一位。

3.3.3 循环冗余码校验码

循环冗余校验(Cyclic Redundancy Check，CRC)是利用除法及余数的原理来实现错误检测。在发送端根据要传送的 k 位二进制码序列，用一定的生成多项式产生一个校验用的位——r 位校验码(即 CRC 码)，并附在信息后边，构成一个新的二进制码序列，共 $k+r$ 位，最后一起发送出去。接收方使用相同的生成多项式进行校验，用接收到的数据除以生成多项式，如果能够除尽，则数据正确；如果不能除尽，则数据错误，并且余数给出了出错位的有关信息，可用于纠正错误。

CRC 校验中最关键的是找到满足一定条件的生成多项式。下面列出了国际上常用的

CRC 循环冗余校验标准生成多项式。

$$\text{CRC-12} = X^{12} + X^{11} + X^3 + X^2 + X + 1$$

$$\text{CRC-16} = X^{16} + X^{15} + X^2 + 1$$

$$\text{CRC-CCITT} = X^{16} + X^{12} + X^5 + 1$$

$$\text{CRC-32} = X^{32} + X^{26} + X^{23} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

用 CRC-12 生成的 CRC 码为 12 位，CRC-16 和 CRC-CCITT 生成的 CRC 码为 16 位，CRC-32 生成的 CRC 码为 32 位。CRC 校验可以完全检测出所有奇数个随机错误和长度小于或等于 k（生成多项式的阶数）的突发错误。所以，CRC 的生成多项式的阶数越高，那么误判的概率就越小，当然复杂性也随之增加。

CRC 校验在磁表面存储器例如硬盘、磁带方面以及计算机高速通信领域得到了应用。例如，著名的通信协议 X.25 的 FCS（帧检错序列）采用的是 CRC-CCITT，ARJ 和 LHA 等压缩工具软件采用的是 CRC-32，磁盘驱动器的读/写采用了 CRC-16，通用的图像存式 GIF、TIFF 等也采用 CRC 作为检错手段。

关于汉明码和 CRC 校验码的更多内容，此不详述。有兴趣的读者可查阅相关文献。

3.4　本 章 小 结

本章从进制转换、小数点处理、符号表示几方面系统介绍了计算机内数值数据的表示方法。有关非数值数据的组织格式和编码规则，也作了简要介绍。在学习中，学生要重点掌握进制、补码以及浮点数。

习　　题

一、选择题

1. 下列数中，最小的数是（　　）。

A. $(101001)_2$　　B. $(52)_8$　　C. $(2B)_{16}$　　D. 45

2. 下列数中，最大的数是（　　）。

A. $(101001)_2$　　B. $(52)_8$　　C. $(2B)_{16}$　　D. 45

3. 字长 16 位，用定点补码小数表示时，一个字能表示的范围是（　　）。

A. $-1 \sim (1-2^{-15})$　　B. $0 \sim (1-2^{-15})$

C. $-1 \sim +1$　　D. $-(1-2^{-15}) \sim (1-2^{-15})$

4. 若 $[X]_{补} = 10000000$，则十进制真值为（　　）。

A. -0　　B. -127　　C. -128　　D. -1

5. 定点整数 16 位，含 1 位符号位，原码表示，则最大正数为（　　）。

A. 2^{16}　　B. 2^{15}　　C. $2^{15}-1$　　D. $2^{16}-1$

6. 当 $-1 < x < 0$ 时，$[x]_{原} =$（　　）。

A. X　　B. $1-x$　　C. $4+x$　　D. $(2-2^n)-1\times 1$

7. 8位反码表示数的最小值为(　　),最大值为(　　)。

A. −127　　B. +255　　C. +127　　D. −255

8. $N+1$位二进制正整数的取值范围是(　　)。

A. $0\sim2^{n}-1$　　B. $1\sim2^{n}-1$　　C. $0\sim2^{n+1}-1$　　D. $1-2^{n+1}-1$

9. 浮点数的表示范围和精度取决于(　　)。

A. 阶的位数和尾数的位数　　B. 阶的位数和尾数采用的编码

C. 阶采用的编码和尾数采用的编码　　D. 阶采用的编码和尾数的位数

10. 在浮点数编码表示中,(　　)在机器数中不出现,是隐含的。

A. 尾数　　B. 符号　　C. 基数　　D. 阶码

11. 正0和负0的机器数相同的是(　　)。

A. 原码　　B. 真值　　C. 反码　　D. 补码

12. 不区分正数和负数的机器数是(　　)。

A. 原码　　B. 移码　　C. 反码　　D. 补码

13. 最适合做乘法运算的码制是(　　)。

A. 原码　　B. 移码　　C. 反码　　D. 补码

14. 做加减法运算最方便的码制是(　　)。

A. 原码　　B. 移码　　C. 反码　　D. 补码

15. 下面真值最大的补码数是(　　)。

A. $(10000000)_2$　　B. $(11111111)_2$

C. $(01000001)_2$　　D. $(01111111)_2$

16. 下面最小的数字是(　　)。

A. $(123)_{10}$　　B. $(136)_8$　　C. $(10000001)_2$　　D. $(8F)_{16}$

17. 整数在计算机中通常采用(　　)格式存储和运算。

A. 原码　　B. 反码　　C. 补码　　D. 移码

18. 下面不合法的数是(　　)。

A. $(11111111)_2$　　B. $(139)_8$　　C. $(2980)_{10}$　　D. $(1AF)_{16}$

19. −128的8位补码机器数是(　　)。

A. $(10000000)_2$　　B. $(11111111)_2$　　C. $(01111111)_2$　　D. 无法表示

20. 8位字长补码表示的整数N的数据范围是(　　)。

A. −128～127　　B. −127～127　　C. −127～128　　D. −128～128

21. 8位字长原码表示的整数N的数据范围是(　　)。

A. −128～127　　B. −127～127　　C. −127～128　　D. −128～128

22. 若用8421BCD码表示十进制的16,应该是(　　)。

A. 10000　　B. 00010110　　C. 11　　D. 10110

23. ASCII码是对(　　)进行编码的一种方案。

A. 字符、数字、符号　B. 汉字　　C. 多媒体　　D. 声音

24. 汉字在计算机中存储所采用的编码是(　　)。

A. 国标码　　B. 输入码　　C. 字形码　　D. 机内码

25. 下列(　　)编码是常用的英文字符编码。

A. ASCII　　B. Unicode　　C. GB 2312　　D. GBK

26. 最简单的数据校验法是(　　)。

A. 循环冗余码校验　　B. 汉明码校验

C. 判别校验　　D. 奇偶校验

二、填空题

1. 二进制中的基数为_____，十进制中的基数为_____，八进制中的基数为_____，十六进制中的基数为_____。

2. $(27.25)_{10}$ 转换成十六进制数为_____。

3. $(0.65625)_{10}$ 转换成二进制数为_____。

4. 在原码、反码、补码三种编码中，_____数的表示范围最大。

5. 在原码、反码、补码三种编码中，符号位为0，表示数是_____。符号位为1，表示数是_____。

6. 0的原码为_____；0的补码为_____；0的反码为_____。

7. 在_____表示的机器数中，零的表示形式是唯一的。

8. 8字长的机器中，－1011011的补码为_____，原码为_____，反码为_____。

9. 8字长的机器中，＋1001010的补码为_____，原码为_____，反码为_____。

10. 浮点数的表示范围由_____部分决定。浮点数的表示精度由_____部分决定。

11. 在浮点数的表示中，_____部分在机器数中是不出现的。

12. 计算机定点整数格式字长为8位(包含1位符号位)，若 x 用补码表示，则 $[x]_{补}$ 的最大正数是_____，最小负数是_____。(用十进制真值表示)

13. 真值为－100101的数在字长为8的机器中，其补码形式为_____。

14. 浮点数一般由_____和_____两部分组成。

15. 在计算机中，数据信息包括_____和_____。

16. 模是指_____。

17. 表示一个数据的基本要素是_____、_____、_____。

18. 在计算机内部信息分为两大类，即_____和_____。

19. 设字长为8位，则－1的原码表示为_____，反码表示为_____，补码表示为_____。

20. 设字长为 n 位，则原码表示范围为_____，补码的表示范围为_____。

21. $(200)_{10}=(\quad)_2=(\quad)_8=(\quad)_{16}$。

22. $(326.2)_8=(\quad)_2=(\quad)_{16}$。

23. $(528.0625)_{10}=(\quad)_{16}$。

24. 一个 R 进制数转换为十进制数常用的办法是_____，一个十进制数转换为 R 进制数时，整数部分常用的方法是_____，小数部分常用的方法是_____。

25. 国际上常用的英文字符编码是_____。它采用7位编码，可以对_____种符号进行编码。

26. 若字母A的ASCII编码是65，则B的ASCII编码是_____。

三、解答题

1. 设字长为 8 位，分别用原码、反码、补码表示－127 和 127。

2. 将 63 表示为二进制、八进制、十六进制数。

3. 将$(3CD.6A)_{16}$转换为二进制和八进制数。

4. 设字长为 8 位，$X=10100101$，$Y=11000011$，求 $X \wedge Y$，$X \vee Y$，$X \oplus Y$ 的结果。

5. 将二进制数－0.0101101 用规格化浮点数格式表示。格式要求：阶 4 位，含 1 位符号位；尾数 8 位，含 1 位符号位。阶和尾数均用补码表示。

6. 将二进制数＋1101.101 用规格化浮点数格式表示。格式要求：阶 4 位，含 1 位符号位；尾数 8 位，含 1 位符号位。阶和尾数均用补码表示。

7. 什么是机器数？

8. 数值数据的三要素是什么？

9. 在计算机系统中，数据包括哪两种？简要解释。