3 数据表示、数据运算算法及其线路实现

运算器最重要的功能是加工数据，因此必须熟练地掌握各种类型的数据及其在计算机内的表示、存储方式、完成运算所用的算法，实现这些算法所用的原理性电路。这些内容又以数字化信息编码、布尔代数、数字电路与逻辑设计为基础。布尔代数、数字电路与数字逻辑设计在本书的第2章进行了简要介绍，因此，本章将从数字化信息编码讲起，介绍二进制编码、数制转换，插入部分检错纠错码的知识；并介绍各种基本类型数据的表示以及数值数据算术运算的有关方法。

二进制编码、数制转换以及定点小数和整数的原、反、补码表示是核心重点内容。

检错纠错码部分将介绍两种常用的检错、纠错码的实现原理，应对其深入理解，而对它们所用的具体线路一般了解即可。

应该掌握定点小数、整数、浮点数在计算机内的表示，补码加减法的运算规则，原码一位乘除法的实现算法和完成算术运算用到的原理性逻辑线路。补码乘除法、加速乘除运算的可行方案一般了解即可。

本章后面的习题较多，这些对深入理解所学的知识并熟练地完成必要的计算很有必要，也是更好地学习运算器知识的基础，应认真完成。

3.1 数字化信息编码的概念和二进制编码知识

3.1.1 数字化信息编码的概念

计算机最重要的功能是处理信息，如数值、文字、符号、语音、图形和图像等。在计算机内部，各种信息都必须采用数字化的形式保存、加工与传送，掌握信息编码的概念和技术是至关重要的。

所谓编码就是用**少量、简单的基本符号**，利用**一定的组合规则**，表

示大量复杂多样的信息。基本符号的种类和这些符号的组合规则构成编码的两大要素。例如,用10个阿拉伯数字表示数值,用26个英文字母构成英文词汇,就是现实生活中编码的典型例子。

当要使用的基本符号数量较多时,往往还要采取措施,首先使用更少量的简单符号来编码,以表示那些量大而复杂的基本符号,再用这些基本符号来表示信息,这就构成了多重编码。多重编码的典型例子是汉字编码,若把上万个汉字都作为基本符号就太多了,可以首先用笔形字画、偏旁部首、拼音或其他方式对它们进行编码,以解决汉字的输入问题。

在计算机中,广泛采用的是仅用"0"和"1"两个基本符号组成的基2码,亦称为二进制码。这主要有以下几个原因。

(1) 基2码在物理上最容易实现,即容易找到具有两个稳定状态且能方便地控制状态转换的物理器件;可以用两个状态分别表示基本符号"0"和"1"。

(2) 用基2码表示的二进制数,其编码、计数和算术运算规则简单,容易用数字电路实现,为提高计算机的运算速度和降低实现成本奠定了基础。

(3) 基2码的两个基本符号"0"和"1"能方便地与逻辑命题的"否"和"是"或"假"和"真"相对应,为计算机中的逻辑运算和程序中的逻辑判断提供了便利条件。

计算机中各种类型的数据通常都是以二进制编码的形式来表示、存储、处理和传送的。

3.1.2 二进制编码和码制转换

1. 数制与进位记数法

在采用进位记数的数字系统中,如果只用 r 个基本符号(例如0,1,2,…,$r-1$),通过排列起来的符号串表示数值,则称其为基 r 数制(radix-r number system),r 称为该数制的基(radix)。假定用 $m+k$ 个自左向右排列的符号 $D_i(-k\leqslant i\leqslant m-1)$ 表示数值 N,即

$$N = D_{m-1}D_{m-2}\cdots D_1D_0D_{-1}D_{-2}\cdots D_{-k} \tag{3.1}$$

式中的 $D_i(-k\leqslant i\leqslant m-1)$ 为该数制采用的基本符号,可取值0,1,2,…,$r-1$,小数点位置隐含在 D_0 与 D_{-1} 位之间,则 $D_{m-1}\cdots D_0$ 为 N 的整数部分,$D_{-1}\cdots D_{-k}$ 为 N 的小数部分。

如果每个 D_i 的单位值都赋以固定的值 W_i,则称 W_i 为该位的权(weight),此时的数制称为有权的基 r 数制(weighted radix-r number system)。此时 N 代表的实际值可表示为:

$$N = \sum_{i=-k}^{m-1} D_i \times W_i \tag{3.2}$$

如果该数制编码还符合逢 r 进位的规则,则每位的权(简称位权)可表示为:

$$W_i = r^i$$

式中的 r 是数制的基,i 为位序号。式(3.2)又可以写为:

$$N=\sum_{i=-k}^{m-1}D_i\times r^i \tag{3.3}$$

式中：

r 是这个数制的基(radix)。

i 表示这些符号的排列次序，即位序号。

D_i是位序号为 i 的一位上的符号。

r^i是第 i 位上的一个 1 所代表的值(位权)。

$D_i\times r^i$是第 i 位上的符号所代表的实际值。

$\sum$表示对 $m+k$ 位中各位的实际值执行累加求和。

N 代表一个数值。

此时该数制被称为 r 进位数制 (positional radix-r number system)，简称 r 进制。下面是计算机中常用的几种进位数制。

二进制为 $r=2$，基本符号为 0,1

八进制为 $r=8$，基本符号为 0,1,2,3,4,5,6,7

十六进制为 $r=16$，基本符号为 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,其中 A～F 分别表示十进制数 10,11,12,13,14,15。

十进制为 $r=10$，基本符号为 0,1,2,3,4,5,6,7,8,9。

如果每一数位都具有相同的基，即采用同样的基本符号集来表示，则称该数制为固定基数制 (fixed radix number system)，这是计算机内普遍采用的方案。在个别应用中，也允许对不同的数位或位段选用不同的基，即混合采用不同的基本符号集来表示，则该数制称为混合基数制 (mixed radix number system)。

2. 二进制编码和二进制数据

二进制编码是计算机内使用最多的码制，它只使用两个基本符号“0”和“1”，并且通过由这两个符号组成的符号串来表示各种信息(各种类型的数据)。二进制数值类型的数据亦是如此，计算其所代表的数值的运算规则如下：

$$N=\sum_{i=-k}^{m-1}D_i\times 2^i \quad (D_i \text{ 的取值为 0 或 1}) \tag{3.4}$$

例如：$(1101.0101)_2=(13.3125)_{10}$。

等号左右两边括号内的数字为两个不同进制的数字，括号右下角的 2 和 10 分别表示左右两边的数字为二进制和十进制的数。按式(3.4)计算二进制的 1101.0101 的实际值为：

$$1\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0+0\times 2^{-1}+1\times 2^{-2}+0\times 2^{-3}+1\times 2^{-4}$$
$$=8+4+1+0.25+0.0625=13.3125$$

从上式可以看出，由于二进制只用 0 和 1 两个符号，在计算二进制位串所代表的实际

值时,只需把符号为1的那些位的位权相加即可,则上式变为:

$$2^3+2^2+2^0+2^{-2}+2^{-4}=13.3125$$

牢记二进制数每位上的位权是有好处的。当位序号为0～12时,其各位上的位权分别为1、2、4、8、16、32、64、128、256、512、1 024、2 048和4 096。

3. 数制转换

计算机中常用的几种不同的进位数制包括二(八、十六)进制和十进制。二进制数据更容易用逻辑线路处理,更接近计算机硬件能直接识别和处理的数字信息的使用要求,而使用计算机的人更容易接受十进制的数据类型。二者之间的进制转换是经常遇到的问题。

(1) **二(八、十六)进制与十进制数据的转换**

式(3.3)确定的运算规则是不同进位记数制数据之间完成进位制转换的依据。

十进制到二进制的转换通常要区分数据的整数部分和小数部分,并分别按除2取余数部分和乘2取整数部分两种不同的方法来完成。

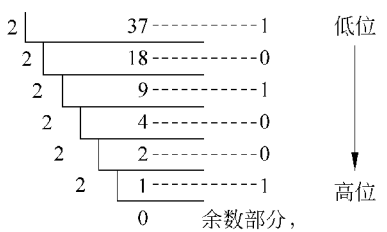
对整数部分,要用除2取余数的方法完成十进制到二进制的转换,其规则如下:

① 用2除十进制数的整数部分,取其余数为转换后的二进制数整数部分的低位数字。

② 再用2去除所得的商,取其余数为转换后的二进制数高一位的数字。

③ 重复执行第②步的操作,直到商为0,结束转换过程。

例如,将十进制的37转换成二进制整数的过程如下:



即转换后的结果为$(100101)_2$。

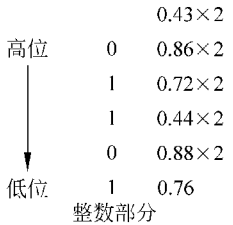
对小数部分,要用乘2取整数的方法完成十进制到二进制的转换,其规则如下:

① 用2乘十进制数的小数部分,取乘积的整数为转换后的二进制数的最高位数字。

② 再用2乘上一步乘积的小数部分,取新乘积的整数为转换后二进制小数部分的低一位数字。

③ 重复第②步的操作,直至乘积部分为0,或已得到的小数位数满足要求,结束转换过程。

例如,将十进制的0.43转换成二进制小数的过程如下(假设要求小数点后取5位):



即转换后的二进制小数为$(0.01101)_2$。

在对小数进行转换的过程中，若转换后的二进制已达到要求的位数，而最后一次乘积的小数部分不为0，则会使转换结果存在误差，其误差值小于求得的最低一位的位权。

对既有整数部分又有小数部分的十进制数，可以先转换其整数部分为二进制数的整数部分，再转换其小数部分为二进制数的小数部分，通过把得到的两部分结果合并起来即可得到转换后的最终结果。例如，$(37.43)_{10}=(100101.01101)_2$。

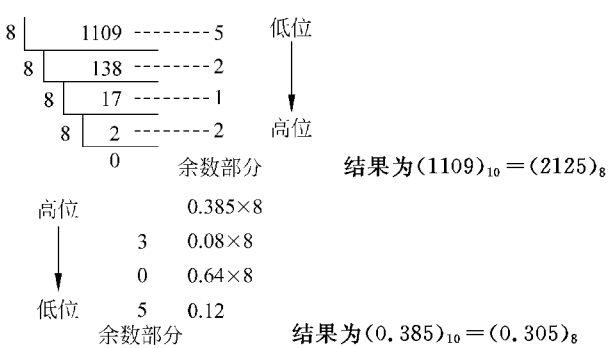
在实现手工转换时，如果对二进制数已经比较熟悉，基本记住了以2为底的指数值，即二进制数每一位上的权，则对十进制数进行转换时，也可以不采用上述规则，直接写出来即可。

例如：

$$(45.625)_{10}=32+8+4+1+0.5+0.125$$
$$=(10\ 1\quad 1\ 01.\quad 10\quad\ 1)_2,\quad 即(101101.101)_2。$$
$$(1105)_{10}=1024+81=1024+64+16+1$$
$$=(1000\ 10\ 10001)_2,\quad 即(10001010001)_2。$$

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。

例如：① 完成八进制和十进制数据之间的转换。



② 十进制到十六进制数的转换方法与前面介绍的方法类似，只是乘除16时，手工运算不大方便。

(2) 二进制与八进制及二进制与十六进制的转换

用二进制表示一个数值N，所用的位数K为$\log_2 N$，如表示4 096，则K为13，写起来

位串较长。因此,计算机中也常常采用八进制和十六进制来表示数值数据。为表示数值N,分别有如下对应关系:

$$N=\sum_{i=-k}^{m-1}D_i\times 8^i \quad (D_i \text{ 的取值为 } 0\sim 7) \tag{3.5}$$

例如:

$$(7.44)_8=7\times 8^0+4\times 8^{-1}+4\times 8^{-2}=(7.562\,5)_{10}。$$

$$N=\sum_{i=-k}^{m-1}D_i\times 16^i \quad (D_i \text{ 的取值为 } 0\sim 9 \text{ 和 } A\sim F) \tag{3.6}$$

例如:$(1A.08)_{16}=1\times 16^1+10\times 16^0+8\times 16^{-2}=(26.03125)_{10}$。

上述两式中所用符号的意义与式(3.3)中所用符号的意义类同,但这里 D_i 包含的基本符号分别限于0～7和0～9、A～F,各位的码权分别为 8^i 和 16^i。

把用二进制、八进制、十六进制表示的数转换成十进制数,能更容易衡量这个数值的大小。

(3) 二进制数与八进制、十六进制数的关系

由于 $\log_2 8=3$,$\log_2 16=4$,即一位八进制数可以用三位二进制数重编码得到,一位十六进制数可以用4位二进制数重编码得到,故人们通常认为,在计算机领域内,八进制和十六进制数只是二进制数的一种特殊的表示形式。表3.1中列出了少量二、八、十六和十进制数的对应关系。

表3.1 二、八、十六和十进制数的对应关系

二进制数	八进制数	十六进制数	十进制数的值
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

在把二进制数转换成八进制或十六进制数时，应从小数点所在的位置分别向左、向右对每3位或每4位二进制位进行分组，写出每一组所对应的一位八进制数或十六进制数。若小数点左侧(即整数部分)的位数不是3或4的整数倍，可以按在数的最左侧补零的方法处理；对小数点右侧(即小数部分)，应按在数的最右侧补零的方法处理，否则容易转换错误。对不存在小数部分的二进制数(整数)，应从最低位开始向左把每3位划分成一组，使其对应一个八进制位，或把每4位划分成一组，使其对应一个十六进制位。例如，$(10.101)_2$ 变成八进制时，应首先变为$(010.101)_2$，是$(2.5)_8$，即八进制的2.5。当把它转换为十六进制时，应首先变为$(0010.1010)_2$，是$(2.A)_{16}$，即十六进制的2.A，而不是$(2.5)_{16}$。又如

$$(1100111.10101101)_2=(147.532)_8$$

$$(1100111.10101101)_2=(67.AD)_{16}$$

八进制和十六进制之间的转换不常用，经过二进制的中间结果进行转换更方便。

4. 二进制数的运算规则

二进制数之间可以执行算术运算和逻辑运算，其规则简单，容易实现。

(1) **加法运算规则**

0+0=0
0+1=1
1+0=1
1+1=0 (产生进位)

例如

$$\begin{array}{r} 1101 \\ +)\ 1001 \\ \hline 10110 \end{array}$$

(2) **减法运算规则**

0 − 0=0
0 − 1=1 (产生借位)
1 − 0=1
1 − 1=0

例如

$$\begin{array}{r} 1101 \\ -)\ 0111 \\ \hline 0110 \end{array}$$

(3) **乘法运算规则**

二进制数乘法的计算方法与十进制数乘法的计算方法类似，按乘数每一位的值计算出部分积，对全部部分积求累加和即可得到最终的乘积。

0×0=0
0×1=0
1×0=0
1×1=1

例如

$$\begin{array}{r} 1101 \\ \times\ 1001 \\ \hline 1101 \\ 0000 \\ 0000 \\ 1101 \\ \hline 1110101 \end{array}$$

(4) 除法运算规则

二进制数除法的计算方法与十进制数除法类似,也由减法、逐位上商等步骤完成。

例如:

$$
\begin{array}{r}
1101 \\
1001\overline{)1110101} \\
\underline{1001} \\
1011 \\
\underline{1001} \\
1001 \\
\underline{1001} \\
0
\end{array}
$$

逻辑运算是在对应的两个二进制位的逻辑值之间进行的,与相邻的高低位的值均无关,即不存在进位、借位等问题。

(5) 逻辑或运算规则(运算符为 ∨)

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

例如

$$
\begin{array}{r}
1100 \\
\underline{\vee\ 1010} \\
1110
\end{array}
$$

(6) 逻辑与运算规则(运算符为 ∧)

$$0 \wedge 0=0 \qquad 0 \wedge 1=0 \qquad 1 \wedge 0=0 \qquad 1 \wedge 1=1$$

例如

$$
\begin{array}{r}
1100 \\
\underline{\wedge\ 1010} \\
1000
\end{array}
$$

(7) 逻辑非运算规则(运算符为¬)

$$\neg 0=1$$

$$\neg 1=0$$

逻辑非用于实现对单个逻辑值的处理,而不是对两个逻辑值的运算,逻辑非又被称为逻辑取反操作。对逻辑数 1011 逐位进行取反,其结果为 0100。

(8) 逻辑异或运算规则(运算符为 ∀)

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

例如

$$
\begin{array}{r}
1100 \\
\underline{\forall\ 1010} \\
0110
\end{array}
$$

与、或、非操作是三种最基本的逻辑操作,用它们可以实现任何逻辑运算功能。某些情况下,还要用到逻辑异或操作。逻辑异或实现的是按位加功能,只有参与异或操作的两

个逻辑值不同时(一个为0,另一个为1),结果才为1。它与或操作结果的差异表现为:或操作中1或1=1,而异或操作中则是1异或1=0。

3.1.3 检错纠错码

1. 检错纠错的有关概念和实现思路

数据在计算机系统内进行加工、存取和传送的过程中可能产生错误。为减少和避免这类错误,一方面需要精心选择各种电路,改进生产工艺与测试手段,尽量提高计算机硬件本身的可靠性;另一方面需要在数据编码上找出路,即采用带有某种特征能力的编码方法,通过少量的附加电路,使之能发现某些错误,甚至能准确地确定出错位置,进而提供自动纠正错误的能力。

数据校验码就是一种常用的具有发现某些错误,甚至具有一定自动改错能力的数据编码方法。它的实现原理是在合法的数据编码之间加一些不允许出现的(非法的)编码,使合法的数据编码出现某些错误时,成为非法编码。这样可以通过检查编码的合法性来达到发现错误的目的。合理地设计编码规则,安排合法、不合法的编码数量,就可以具备发现错误的能力,甚至达到自动改正错误的目的。这里用到一个**码距**(最小码距)的概念。码距是指任意两个合法码之间至少有几个二进制位不相同,若仅有一位不同,则称其(最小码距)为1。例如用4位二进制表示16种状态,则16种编码都用到了,此时码距为1,也就是说,任何一个编码状态的4位码中的一位或几位出错,都会变成另一个合法码,此时无检错能力。若用4个二进制位表示8种合法状态,就可以只用其中的8个编码来表示,而把另外8种编码作为非法编码,此时可能使合法码的码距为2。一般说来,合理地增大编码的码距,就能提高发现错误的能力,但表示一定数量的合法码所使用的二进制位数增多,则会增加电子线路的复杂性和数据存储、数据传送的数量。在确定与使用数据校验码的时候,通常要考虑在不过多增加硬件开销的情况下,尽可能地发现较多的错误,甚至能自动改正某些最常出现的错误。常用的数据校验码是奇偶校验码、汉明校验码、循环冗余校验码等。纠错编码是对检错编码更进一步的发展和应用。

计算机内经常遇到的错误有两大类:随机错误和突发错误。前者是指孤立出现的一个错误,后者是指连续产生的一批(彼此之间可能有关联)错误。对它们进行处理的难度和复杂度会有很大不同,在本书中基本不涉及对突发错误的检查与纠正问题,有兴趣的读者可自行查阅有关资料。纠错编码的分类方案如图3.1所示。

从校验位与信息位的**形成关系**区分,可分成线性码和非线性码。

从校验位与信息位的**约束条件**区分,可分成分组码和卷积码。

从码字本身的**结构特性**区分,可分成循环码和非循环码。

信息位和校验位的**排列位置**,数据位在前,校验位跟随其后时为系统码。

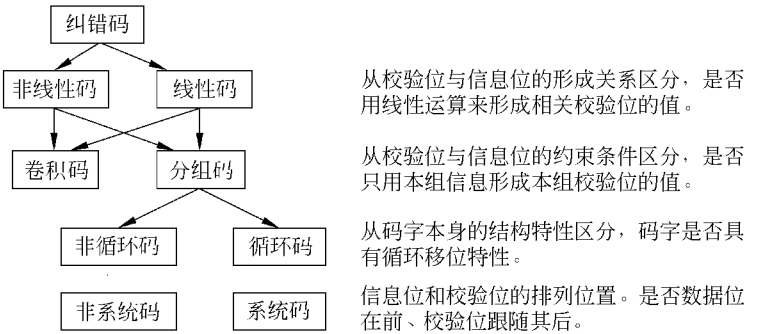


图 3.1　纠错码的分类

2. 三种常用的检错纠错码

(1) 奇偶校验码

奇偶校验码是一种开销最小，能发现数据代码中一位出错情况的编码，常用于存储器读写检查，或 ASCII 字符、其他类型信息传送过程中的出错检查。它的实现原理是使原来合法的编码码距由 1 增加到 2。若合法编码中有一个二进制位的值出错，由 1 变成 0 或由 0 变成 1，这个码必将成为非法编码。实现的具体方法通常是为一个字节补充一个二进制位，称为校验位，通过设置校验位的值为 0 或 1，使字节自身的 8 位和校验位为 1 值的位数一定为奇数或偶数。在使用奇数个 1 的方案进行校验时，称为奇校验，反之则称为偶校验。依据 8 位数据位中为 1 值的个数确定校验位的值，是由专设的线路实现的，通常使用几级异或门电路产生校验位的值，其输入是数据位的信号或异或运算的中间结果，最后的输出信号是校验位的值。例如，当要把一个字节的值写进主存时，首先用此电路产生校验位的值，然后将这 9 位代码作为合法的数据编码写进主存。当下一次读出这一代码时，再用相应线路检测这 9 位码的合法性。若在主存写进、存储或读出的过程中，某一个二进制位上出现错误，则得到的 9 位码必变成非法编码，从而可以发现一定是哪一位上出现了错误。这种方案只能发现一位错或奇数个位出错，但不能确定是哪一位错，也不能发现偶数个位出错。考虑到一位出错的概率比多位同时出错的概率高得多，该方案还是有很好的实用价值。

下面给出对几个字节值进行奇偶校验的编码结果。

数据	奇校验的编码	偶校验的编码
00000000	**1**00000000	**0**00000000
01010100	**0**01010100	**1**01010100
01111111	**0**01111111	**1**01111111
11111111	**1**11111111	**0**11111111