计 算 机 算 术 运 算

原理、结构与设计

黄 铠 著

李三立 林定基 译

内 容 简 介

本书较全面地介绍了近代数字计算机中的算术运算，其中包括基本原理的阐述、结构的分析、设计的方法和实例的研究.

全书共十一章. 第一、二章介绍计算机算术运算的基本概念；第三、四章讲述双操作数和多操作数加法器的原理与设计；第五至第八章详细论述了各种类型的乘法器和除法器；第九、十章讨论浮点运算处理器；第十一章论述了基本函数、流水线运算以及运算误差的控制.

本书可供从事计算机研究与设计的科技人员阅读，也可供高等院校计算机专业的师生学习参考.

Kai Hwang

COMPUTER ARITHMETIC

Principles, Architecture And Design

John wiley & Sons, 1979

计 算 机 算 术 运 算

原理、结构与设计

黄 铠 著

李三立 林定基 译

*

科 学 出 版 社 出 版

北京朝阳门内大街 137 号

中国科学院印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1980 年 12 月第 一 版 开本：787×1092 1/16

1980 年 12 月第一次印刷 印张：19

印数：0001—12,900 字数：437,000

统一书号：15031·301

本社书号：1849·15—8

定价：2.35 元

著者为中译本写的序言

本书内容包括现代数字计算机算术运算原理，计算方法，以及运算处理机之系统结构，逻辑设计与大、中规模集成电路运算装置的设计与技术工艺等．可适用于大学及理工学院，作为计算机科学与计算工程设计方面的教科书．也可以用为设计，发展与应用计算机的专业参考书．读者需要有最基本的逻辑设计的背景，学过微电子学或集成电路会有一些帮助，但非必要．本书的特点在于用运算流程图、算术表达式、布尔方程、原理逻辑电路图、数字列表与微指令控制矩阵等来解释计算机算术运算的方法，程序及线路硬件装置等．以往二十年来的重要新发展都包括在内．资料文献的来源都在每章之后详细注明，搜集至 1978 年初为止．每章后所附习题供读者练习设计之用．

全书共分十一章．头一章介绍现有的五种运算数字系统与机器内的数字表示储存格式，特别强调定点、浮点、反码、补码与冗余码的应用．第二章简短介绍常用的中规模及大规模集成电路部件，包括运算及逻辑装置，寄存器与固态存储器等．此章是为缺乏微电子数字装置训练的读者而写．第三章讲解基本加法器及减法器的原理、设计与评价．特别强调 n 进制反码及补码加减法，以及先行进位、进位存储、进位完成等技术．第四章讲解多操作数加法，进位存储加法树，冗余码带符号数字的并行算术运算原理以及集成电路运算-逻辑器的设计、控制与应用．

第五、六两章讲解各种乘法器的设计．先介绍常规的移位乘法器以及再编码乘法原理与其评价．然后再详细说明集成电路高速网络乘法器的各种设计方法与装置条件．包括用只读存储器与加法器混合的乘法设计，或用对数变换的乘、除法．第七、八两章讲解各式的除法原理与装置．先叙述标准移位除法器，包括各种基数的恢复与不恢复除法，以及 SRT 除法原理与在 ILLIAC III 运算处理机中的应用．然后介绍以复乘法为基础的收敛性除法并以 IBM 360/91 计算机中收敛除法为例．最后讲解三种适合集成电路的网络式除法器的设计与评价．

第九章讲解浮点运算处理机的设计．先制定规格化浮点加、减、乘、除的基本程序，以及部件装置与安排．然后以 IBM 360/91 计算机中的全部浮点运算为实例，介绍浮点系统的设计．第十章分析浮点运算的误差精度．先介绍非规格化浮点运算的利弊，接着讨论误差上限与各种舍入方式以控制精确度．最后比较浮点指令，讲双精度浮点运算的程序与格式．第十一章讲解初等函数的求值，包括开平方根、求平方值、多项式的值、三角函数、对数以及其他超越函数等．同时介绍 CORDIC 计算方法与 Chen 氏收敛算法．然后讲解流水线算术运算原理，以 TI-ASC 及 CDC-STAR 两个超型计算机为实例，解释适合处理向量的多条流水线、并行运算器的设计与控制．并介绍多功能的网络式流水设计，以及流水式的高速傅里叶变换器设计．最后本书讨论算术运算处理机中各种错误控制方法与设施．

此书承清华大学计算机工程与科学系的李三立副教授及林定基讲师在百忙中细心翻

译为中文，著者在此对他们认真工作、热心祖国科技教育事业的崇高精神，表示衷心的敬意．并特别致谢科学出版社同仁在校稿，印刷及出版上的精美工作．

最后著者愿意与国内同胞共同勉励，为我们祖国的繁荣富强、早日完成中国式的现代化而努力奋斗．

黄铠　识

1979 年 7 月 27 日于美国普度大学

目　录

第一章　计算机算术运算和数的系统

1.1　计算机算术运算概论

算术运算在人类文明中，尤其在科学、工程和工艺技术领域中起着重要的作用．机器的算术运算可以追溯到公元前五百年在中国采用算盘的形式．在整个计算机的历史中，运算处理器是计算机的主要工作力量．运算处理器用来执行算术运算操作，并且为所要计算的问题产生数值解．由于中/大规模集成（MSI/LSI）电路的出现，愈来愈多的高级复杂的运算处理器已成为今日高性能数字计算机系统标准硬件的特点．一个现代化的数字计算机可以配备多个硬件运算处理器，这些处理器可以用来处理不同格式的数据，或者求解通用的和专用的特殊算术运算函数．这些硬件运算处理器提供比一般计算手段更快的计算速度．选择一个合适的算术运算系统，对计算机结构设计和程序设计两者都有重要影响．

令 **R** 是所有实数的集合．一个**实数算术运算**可以定义为一个代数映象

$$\mathbf{f}:\mathbf{R}\times\mathbf{R}\to\mathbf{R} \tag{1.1}$$

式中 **f** 的典型例子就是标准的两个操作数的算术运算操作 +，−，×，÷．令 **M** 是机器表示数的集合．在集合 **M** 中每个数只能有有限数目的数位．集合 **M** 必须是 **R** 的一个合适的有限子集，即 $\mathbf{M}\subset\mathbf{R}$．一个**机器的算术运算**可以用以下映象建立模型：

$$\mathbf{g}:\mathbf{M}\times\mathbf{M}\to\mathbf{M} \tag{1.2}$$

机器算术运算与实数算术运算不同之处在于数的精度的基本结果．用运算处理器只能执行有限精度的计算，而实数运算可以产生字长无限制的任意精度的结果．

换句话说，我们可以把机器算术运算看成是一种具有恰当舍入控制的近似的实数算术运算．用函数映象表示，以上机器算术运算函数 **g** 与实数算术运算函数 **f** 之间的关系，可用下列合成函数联系起来

$$\mathbf{g}=\mathbf{h}\circ\boldsymbol{\rho} \tag{1.3}$$

映象 **h** 定义为

$$\mathbf{h}=\mathbf{f}|\mathbf{M}\times\mathbf{M}\to\mathbf{R} \tag{1.4}$$

实数 **f** 限制在机器域（Machine Domain）$\mathbf{M}\times\mathbf{M}$ 以内．定义为下式的映象

$$\boldsymbol{\rho}:\mathbf{R}\to\mathbf{M} \tag{1.5}$$

是所选择的舍入方案，它将使任意长度的结果修整为一个机器可表示的数．注意，数字计算机与一般普通的算术运算设备对比，在给定足够的存储容量的情况下，可以执行任意精度的计算．有限精度的结果，只是指在受限制的字长（在寄存器或加法器）和局限的存储容量的机器中所得的结果．

数字计算机的算术运算是一个涉及数字计算机的系统结构和逻辑设计的领域．因此算术运算的设计要包括算术运算算法的发展及其逻辑上的实现．在计算机结构领域它与

实现算术运算操作的效率互相有关．在数值分析领域它又关系到近似实数算术运算的精度．图 1.1 示出影响计算机算术运算系统设计与其它有关计算机学科的相互作用关系．可以说，算术运算系统的设计涉及到计算机工程的许多领域，它是计算机科学和数字工程的混合学科．

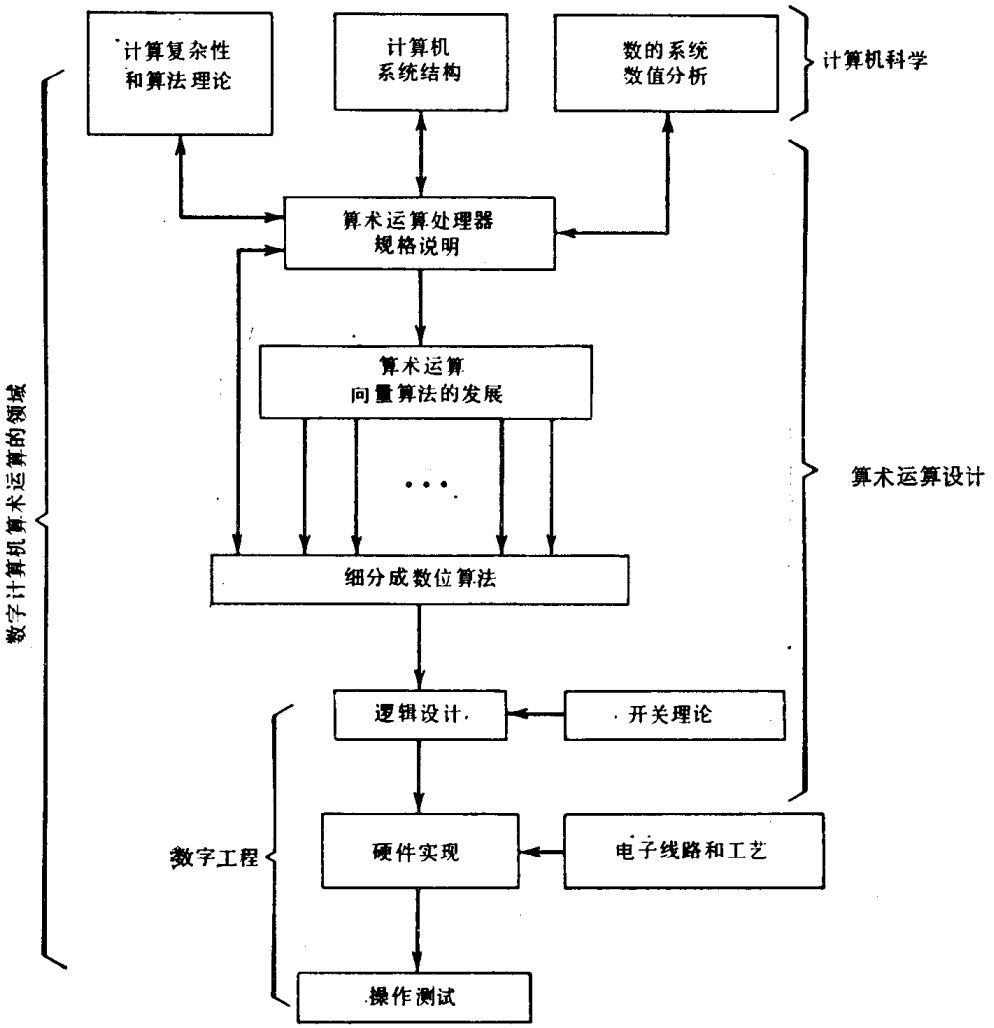


图 1.1　算术运算设计与其它计算机科学和数字工程学科的相互关系

本书描述计算机运算系统的四个主要领域的原理和设计技术：基数算术运算，带符号数字的算术运算，初等函数计值以及流水线算术运算．第一和第二章叙述数的系统和数字电路工艺，并为以下章节提供基础．基数算术运算，正如大多数计算机中遇到的，可分成定点的和浮点的系统．在阐明标准的和非标准的加法、减法、乘法和除法的算法以及它们如何实现时，更强调用新的方法说明，诸如多操作数进位存储加法器，乘数再编码，收敛和 SRT 除法，单元阵列乘法器和除法器，带符号数字的算术运算以及其他．对浮点算术运算从初级的到高等的水平都将予以说明．最后一章讨论初等函数和流水线算术运算的设计．

笔者建议有兴趣的读者查阅每章最后的文献目录单和参考书单，可以得到进一步资料来源．这些材料的大多数是选自一般杂志或会议刊物．习题的安排将有助于学生消化

本书的要点，并希望能激励学生们发展自己新的和改进的技术.

1.2 机器算术运算的数的系统

在数字计算机中，算术运算算法的实现，在很大程度上取决于数值数据是如何存储在存储器和寄存器中的. 不同的内部数的表示方法会产生不同的运算硬件的设计. 选择一种合适的数的系统会影响从设计者角度看来的计算机结构，同时也影响用户所使用的数值分析方法.

因为在数字计算机中只实现有限精度的算术运算，所有允许的数值表示必须限制在有限的字长内. 用机器实现具有有限精度的实数算术运算操作是由于字长受到限制. 选择一个好的算术运算系统和内部数的表示方法会影响机器操作的有效实现以及近似的实数算术运算的精度这两个方面.

算术运算的设计者在解决范围很广的应用问题时，必须注意时间和空间效率结构方面的现实性以及提供充分精度的数值分析方面的现实性. 一般机器的算术运算中，数的系统可分为以下五种范畴.

普通基数的数的系统

普通计算机采用基数 $r \geqslant 2$ 的**固定基数**的算术运算系统，而且数字的集合为:

$$\{0, 1, \cdots, r-1\} \tag{1.6}$$

一个数的所有数字都用正的权，而每个数是单值地表示的. 有些特殊系统可能采用**混合基数**的数，其中一个数的不同数字位采取不同的基本数值. 我们将在第 1.4 节正式讨论固定基数的数的系统.

带符号数字的数的系统

这种系统可以认为是固定基数系统的一种扩展情况，其中正权和负权的数字都允许在一个数字集合中

$$\{-\alpha, \cdots, -1, 0, 1, \cdots, \alpha\} \tag{1.7}$$

其中 α 是一个有界的正整数. 对于一个给定的数值，带符号数字表示法可能不是单值的. 因为对于一个给定的数，可以有多于一个的带符号数字的表示方法，这种数的系统可以认为是冗余的. 冗余的带符号数字的数的系统将在第1.5节中详细讨论.

残数的数的系统

一个残数的每个数位并不赋予权的因数；因此，数字的次序对于决定这个数的值来说是不重要的. 而且混合基数也可赋给不同数位. 一个残数 $\mathbf{x}$ 是用一个 $\mathbf{n}$ 重元表示的整数

$$\mathbf{x} = \{r_1, r_2, \cdots, r_n\}_m \tag{1.8}$$

相对应于另一个 n 重元

$$\mathbf{m} = \{m_1, m_2, \cdots, m_n\} \tag{1.9}$$

每个 r_i 叫做模数为 m_i 的 $\mathbf{x}$ 的残数，其中所有 n 模值 $\{m_i | i = 1, 2, \cdots, n\}$ 双双相对地互为素数. 所有这 n 个残数数字 r_i, $i=1, 2, 3, \cdots, n$, 可以各自独立地处理. 由于这一

点，在残数算术运算中，加法和乘法是固有地无进位的．残数算术运算最先是由 Svoboda[16] 提出的，是用来设计可靠的计算系统时减少算术运算的错误．建议读者可阅读 Szabo 和 Tanaka[15]，以便了解残数算术运算及其应用．

有理数的系统

这种算术运算系统所表示的数值量，如同按照分子-分母整数对来表示分数．有理数的加法、减法、乘法和除法永远产生有理数，所以这些操作是可以有限制的而无需采用无限的精度，如数 $\frac{1}{3}=0.3333\cdots$．但实际的限制使得这种理想情况难以处理，即其中分子和分母就是在中等规模的计算中也会变得很大．

曾经建议对这种近似算术运算设计范围的研究只在于一个有限精度的有理系统中，其中精度的限制是通过约束分数的分子和分母大小来实现的．这种有限精度的约束可以分别或共同做，即限制分子和分母数位的数目，产生名称为固定分切或浮动分切的有理数系统．就考虑其实现来说，有理算术运算还只处在假设阶段．有兴趣读者如要进一步研究，可参阅 Matula[12] 以及 Hwang 和 Chang[7] 的著作．

对数的数的系统

这个系统采用一个实数 $\mu>1$ 作为基数．实数的集合是用下列对数空间 $\mathbf{L}_\mu$ 来定义：

$$\mathbf{L}_\mu=\{\mathbf{x}\mid|\mathbf{x}|=\mu^i,\ i\text{ 是整数}\}\cup\{0\} \tag{1.10}$$

应用指数表达式的表示方法就是建议用几何的舍入法，而不是用算术的舍入法，以便提高数的精度．通过整数算术运算和对数-反对数子程序以实现对数算术运算的详细论述，见 Marasa[10]．

本书中描述的算术运算设计主要以普通基数的数的系统或者以冗余带符号数字的数的系统为基础．残数，有理数和对数的数的系统不在本书范围内．

1.3 算术运算操作的分类

从用户和设计者的角度来看，在现代数字计算机中算术运算指令一般可以分成三种类型．

标准算术运算操作

这一类型主要包括四种简单的算术运算功能：定点或浮点形式的加法，减法，乘法和除法．其它所有的数学函数都可以用这四种标准操作来表示．下面我们主要区别两种操作形式．详细的含意将在以后章节中讨论．

（1）**定点**（以后简称 **FXP**）算术运算．通常用于基数点固定的数据中，如商业或统计计算中遇到的题目．**FXP** 操作还可根据其基数点出现的位置进一步分成两小类．在整数算术运算中所有结果都在寄存器的右端排列起来，如同在最右端有一个基数点．而在分数算术运算中，所有结果不管其长度如何，都是在寄存器的左端排列．

(2) **浮点** (以后简称 **FLP**) 算术运算. 主要在科学和工程计算中使用,其中需要经常改变数值大小的比例尺度. **FLP** 操作也可按照其数据格式分成两类. 当必须使用规格化的数据操作数时,我们称为**规格化 FLP 算术运算**操作. 当操作数在中间和最后过程中无需规格化,则称为**非规格化 FLP 算术运算**操作. 目前使用的大部分计算机都采用规格化操作.

基本算术运算函数

基本函数指的是那些在数学计算中经常用到的特殊算术运算操作. 这些函数包括指数、平方根、对数、三角、双曲函数及其他. 但并非所有计算机都把这些函数作为标准硬件特征加以包括. 目前大部分计算机是用软件或固件例行子程序来实现这些基本函数的. 随着硬件价格的下降,正在愈来愈普遍地采用专用的硬件函数求值器来产生这些基本函数.

伪算术运算操作

这种类型的操作需要一定程度的算术计算,但主要是在执行计算机程序时作为专门的应用. 下面描述伪算术运算操作的两类.

(1) **地址运算**的执行基本上是为了计算有效的存储器地址,诸如变址,间址,相对地址或偏置编址方式.

(2) **数据编辑运算**包括逻辑上字母符号和数据传送操作,诸如比较,求补,输入,存储,组合,拆分,移位,规格化及其他. 这些操作用来把数据从一种格式变换到另一种,检查与源格式的一致性,或者为控制程序的顺序而作的测试.

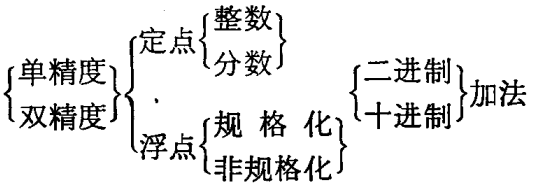
我们主要讨论标准的定点和浮点运算以及某些基本函数的计算. 地址运算和数据编辑操作并不是我们的主要兴趣. 本书中提出的理论和设计技术集中在前两种算术运算操作. 在很多计算机中,算术运算指令是按照执行的数据精度来细分的. 多倍精度的运算对定点和浮点操作都可应用.

(1) **单精度 (SP) 运算**指的是这样一些操作,其中所用的标准数据操作数的字长是和一个存储器字的字长相等的.

(2) **双精度 (DP) 运算**使用的每个操作数的字长加了一倍. 三倍字长或更高精度的操作可以相似地定义.

某些计算机器提供分开的运算硬件来处理二进制和十进制数据,如 IBM 360 系统.在这些机器中,可以直接规定对十进制操作数进行操作. 通常需要有码的转换,组合或拆分指令来直接处理十进制格式的数据.

表 1.1 对以上算术运算操作的分类进行总结. 举例说,一个加法 (ADD) 操作可以分成 16 种,如下所示:



在一个数字计算机中,加法操作的以上 16 种规定取决于数的精度,整数或分数的定点运

算，规格化或非规格化的浮点运算，以及对二进制还是对十进制数据进行操作.

假如对硬件的成本没有限制，而其应用确实需要大量的硬件运算功能以加快处理速度，那么我们还可以设计一个算术运算部件来处理实数或复数，或者设计某些专用的硬件机器来做一些专门的算术计算，诸如超高速傅里叶变换 (SFFT)，矩阵处理器及其他. 随着工艺的进步，这种高级专用的运算机器确实是可行的. 本书所谈到的设计技术为制造专用或通用计算机器提供了必要的知识.

表 1.1 算术运算操作的分类

标准算术运算操作	基本算术运算函数	伪算术运算操作
加法，减法，乘法和除法的操作方式：单精度、双精度、定点整数、定点分数、规格化浮点、非规格化浮点、二进制运算、十进制运算或高基数运算、负操作数	指数、对数、平方根、立方根、平方、立方、高次幂或高次根、三角和双曲线函数、特殊多项式、超越函数	**数据编辑运算**：比较、范围、求补、求反、输入，存储、组合、拆分、增量、减量、移位，旋转、规格化 **地址运算**：相对寻址、变址、基数寻址、偏置寻址

1.4 普通基数的数的系统

大多数现有的算术运算处理器的基础是具有普通基数的数的系统. 只有很少的机器用冗余带符号数字的算术运算[14]. 其余三种算术运算系统采用残数，有理数和对数的数的系统，目前尚未广泛应用.

一个以 r 为基数的数 $\mathbf{X}$ 在数字计算机中是用 $(n+k)$ 重数字向量来表示的

$$\mathbf{X}=(x_{n-1},\cdots,x_0.x_{-1},\cdots,x_{-k})_r \tag{1.11}$$

其中每个分量 x_i (对于 $-k\leqslant i\leqslant n-1$) 称为向量 $\mathbf{X}$ 的第 i 个数字. 每位数字可以有 r 个不同的值

$$\{0,1,\cdots,r-1\} \tag{1.12}$$

其中 r 是数的系统的基数. 在一个固定基数的数的系统中，所有各位数字都具有相同的基数值. 常用的十进制数的表示法属于 $r=10$ 的范畴.

数 $\mathbf{X}$ 的整数部分为前 n 个数位 $(x_{n-1},\cdots,x_1,x_0)$，而其余 k 个具有负下标的数位 $(x_{-1},x_{-2},\cdots,x_{-k})$ 形成数 $\mathbf{X}$ 的分数部分. 基数点 (即通常所说的小数点) 是用来分隔这两部分的. 在实际的计算机线路中，这个基数点是不标明的；就是说，它在存储设备中并不占有一个物理位置. 我们将讨论基数为正整数 $r\geqslant 2$ 的数系统. 以负数，分数和复数为基数的数系统不在本书的讨论范围内.

混合基数的数是指这样一种数，它在不同数位上具有不同的基数值. 我们用来计时的方法(小时，分，秒)就是采用了混合基数 (24, 60, 60). 在具有权的基数的数系统中，我们把每个数字向量 $\mathbf{X}$ 指定一个单值，表示为

$$\mathbf{X}_v=\sum_{i=-k}^{n-1}x_i\cdot\omega_i \tag{1.13}$$

其中每个 ω_i 称为第 i 位数字的加权因子. $n+k$ 个加权因子形成一个权向量，表示为

$$\mathbf{W}=(\omega_{n-1},\cdots,\omega_0,\omega_{-1},\cdots,\omega_{-k}) \tag{1.14}$$

数 $\mathbf{X}$ 的值可以用 $\mathbf{X}\cdot\mathbf{W}$ 求出，即两个向量 $\mathbf{X}$ 和 $\mathbf{W}$ 的点积. 特别是权向量

$$\mathbf{W} = (r^{n-1}, \cdots, r^0, r^{-1}, \cdots, r^{-k}) \tag{1.15}$$

将引出一个基数为 r 的数 $\mathbf{X}$ 的常用记数表示法，其值

$$\mathbf{X}_v = \mathbf{X} \cdot \mathbf{W} = \sum_{i=-k}^{n-1} x_i \cdot \omega_i = \sum_{i=-k}^{n-1} x_i \cdot r^i \tag{1.16}$$

实际上，经常使用的有四种基数的数系统，即分别相应于基数值 $r = 2, 8, 10$ 和 16 的二进制，八进制，十进制和十六进制的数系统.

基数值 r 愈高，需要有更多的二进位数字(位数)来为每个基数 r 的数字进行编码.通常，至少要有 k 位数来为一个基数 r 的数字编码，其中

$$k = \lceil \log_2 r \rceil \tag{1.17}$$

符号 $\lceil x \rceil$ 表示不小于实数 x 的最小整数. 举例说，我们想用 $k = \lceil \log_2 10 \rceil = 4$ 个二进位来对每个十进制数字 ($r = 10$) 编码. 用于表示十进制数字的若干种 4 位二进制码见表 1.2，其中二-十进制和葛莱码 (Gray code) 是最常用的.

表 1.2 若干表示十进制数字的二进制码

十进制数字	二-十进制码 (8-4-2-1 码)	葛 莱 码 (Gray code)	5-2-1-1 码	余-3 码
0	0 0 0 0	0 0 1 0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 1 0	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 1 1	0 0 1 1	0 1 0 1
3	0 0 1 1	0 1 0 1	0 1 0 1	0 1 1 0
4	0 1 0 0	0 1 0 0	0 1 1 1	0 1 1 1
5	0 1 0 1	1 1 0 0	1 0 0 0	1 0 0 0
6	0 1 1 0	1 1 0 1	1 0 1 0	1 0 0 1
7	0 1 1 1	1 1 1 1	1 1 0 0	1 0 1 0
8	1 0 0 0	1 1 1 0	1 1 1 0	1 0 1 1
9	1 0 0 1	1 0 1 0	1 1 1 1	1 1 0 0

人类已经习惯于实行十进制系统，而在数字计算机中所有数据都是二进制编码的.内部二进制编码的恰当选择要立足于表示方法的效率，算术运算的设备以及操作的可靠性.

设计者必须注意内部基数转换对非二进制机器是需要的. 然而，用户不必注意必要的码的转换. 计算机对用户看来可以是八进制，十进制，十六进制或其它基数的机器. 建议用更高基数的方法来设计高速硬件乘法和除法部件，如在 ILLIAC III 计算机中，用于乘法和除法的基数 $r = 256$[6].

在数字计算机中，正和负的普通基数的数主要有两种不同的表示方法. 方程式 1.6 所示的表示式相应于所谓定点表示法. 另一种存储基数的数的方法是浮点表示法. 我们将在第 1.8 和第 1.9 节中描述这两种基数的数的表示方法.

1.5 带符号数字的数的系统

冗余数的系统对于人工计算可能是不方便的，但它们在设计高速算术运算机器时是有用的. 带符号数字 (**SD**) 的数的表示方法可以有冗余性. 每个带符号的数字可以用几个二进位来表示它 (包括每个 **SD** 的符号位)，这样会增加数据存储空间并需要更宽的数

据总线. 然而，提高速度这一点却是它的一大优点. 我们将阐明使用 **SD** 表示法来设计第四章中的完全并行的加法器，第五章中的再编码乘法器，及第七章中的高基数除法器.

SD 数正式定义如下： 给定一个基数 r，则一个 **SD** 数的每一个数字可具有以下 $2\alpha+1$ 个值

$$\Sigma_r = \{-\alpha, \cdots, -1, 0, 1, \cdots, \alpha\} \tag{1.18}$$

其中最大数字的数值 α 必须在以下范围内：

$$\left\lceil \frac{r-1}{2} \right\rceil \leqslant \alpha \leqslant r-1 \tag{1.19}$$

因为整数 $\alpha \geqslant 1$，必须认为 $r \geqslant 2$. 为了在对称的数字集合 Σ_r 中得到最小的冗余，可选择以下值作为最大数值

$$\alpha = \left\lfloor \frac{r}{2} \right\rfloor \tag{1.20}$$

其中符号 $\lfloor x \rfloor$ 表示小于或等于实数 x 的最大整数. 因此，其中如果 r_e 是偶数，$\alpha = \frac{r_e}{2}$. 而如果 r_0 是奇数，则 $\alpha = \frac{(r_0-1)}{2}$. 这意思是说相邻奇偶基数值可产生相同的数字集合. 相应于这种 α 选择的数字集合可表示为以下两种不同方式，但实际上在 $r_0 = r_e + 1$ 时，两者是相同的集合.

$$\Sigma_{r_0} = \left\{-\frac{r_0-1}{2}, \cdots, -1, 0, 1, \cdots, \frac{r_0-1}{2}\right\}$$

$$\Sigma_{r_e} = \left\{-\frac{r_e}{2}, \cdots, -1, 0, 1, \cdots, \frac{r_e}{2}\right\} \tag{1.21}$$

举例说，基数为 2 的 **SD** 系统具有数字集合 $\Sigma_2 = \{-1, 0, 1\}$，而基数为 4 的 **SD** 系统具有数字集合 $\Sigma_4 = \{-2, -1, 0, 1, 2\}$，以此类推. 一个 **SD** 数

$$\mathbf{Y} = (y_{n-1} \cdots y_0 y_{-1} \cdots y_k)_r \tag{1.22}$$

的代数值 $\mathbf{Y}_r$ 可以计算如下

$$\mathbf{Y}_r = \sum_{i=-k}^{n-1} y_i \cdot r^i \tag{1.23}$$

这和方程式 1.16 相似，只不过现在的 $\mathbf{Y}$ 本身可以是正的或负的，而无需一个显式符号. 零只有一个唯一的表示式，即对方程式 1.22 中所有的 i，有 $y_i = 0$. 一个 **SD** 数 $\mathbf{Y}$ 的负数 $-\mathbf{Y}$ 是通过改变 $\mathbf{Y}$ 中的所有非零数字的符号直接推导出来的. 注意，数字“零”不考虑符号. 实际上，只使用 2 的幂次作为基数值，即 $r = 2^k$.

使用 **SD** 数系统的最初目的是要去除在加法(或减法)中的进位链延迟. 为了断开进位链，α 的下界应该缩紧一些

$$\left\lceil \frac{r+1}{2} \right\rceil \leqslant \alpha \leqslant r-1 \tag{1.24}$$

在第四章中我们将考虑使用的 α 在上述区间之内的 **SD** 加法. 对于使用 **SD** 表示法的除法，方程式 1.19 中给出的界限是充分紧缩的.

让我们考虑若干数值例子. 给定一个明确的数值 $\mathbf{Y}_r = -3$，其字长 $n=4$，$k=0$，基数为 2 的 **SD** 系统可以认为具有一个数字集合 $\{\bar{1}, 0, 1\}$. 为了清楚起见，负的数字

-1 用上面带一道横线表示，即 $\bar{1}$. 有五种合法的 **SD** 表示法可产生数值 -3.

$$\begin{aligned}\mathbf{Y} &= (00\bar{1}\bar{1})_2 = -2 - 1\\ &= (0\bar{1}01)_2 = -4 + 1\\ &= (\bar{1}101)_2 = -8 + 4 + 1\\ &= (0\bar{1}1\bar{1})_2 = -4 + 2 - 1\\ &= (\bar{1}11\bar{1})_2 = -8 + 4 + 2 - 1\end{aligned}$$

这些表示方法具有不同的零和非零数字的分布. 在一个具有 n 个数字其值为 $\mathbf{Y}_v$ 的 **SD** 向量 $\mathbf{Y}$ 中，非零数字的数目叫做**权** $\omega(n, \mathbf{Y}_v)$. 上面的数所具有的权是从 2 到 4. 一般，一个二进制的 n 位数字的 **SD** 向量，它的权定义如下：

$$\omega(n, \mathbf{Y}_v) = \sum_{i=0}^{n-1} |y_i| \tag{1.25}$$

其中当 $y_i \neq 0$ 时，$|y_i| = 1$.

对应于给定的 n 值和 $\mathbf{Y}_v$ 值，具有最小权的 **SD** 向量叫做最小 **SD** 表示法. 在以上例子中，有两个 **SD** 向量最小，即为 $(00\bar{1}\bar{1})_2$ 和 $(0\bar{1}01)_2$. 最小 **SD** 表示法对于第五章中所讨论的乘数再编码是特别有意义的.

普通的基数为 r 的数可以按照下面的方法很容易转换成一个等价的 **SD** 形式：

令 $\mathbf{X} = (x_{n-1}, \cdots, x_1, x_0)_r$ 是一个普通的基数为 r 的数，而 $\mathbf{Y} = (y_{n-1}, \cdots, y_1, y_0)_r$ 是等价的 **SD** 数. 等价的意思是它们代表相同的代数值. 对于每一个普通数字 x_i，我们形成一个中间的差数字 d_i 为

$$d_i = x_i - r \cdot b_{i+1} \tag{1.26}$$

其中借位数字，

$$b_{i+1} = \begin{cases} 0, & \text{如 } x_i < \alpha \\ 1, & \text{如 } x_i \geqslant \alpha \end{cases} \tag{1.27}$$

于是第 i 位 **SD** 数字 y_i 可以用 d_i 和 b_i 相加来获得

$$y_i = d_i + b_i \tag{1.28}$$

举例说，给定 $\mathbf{X} = (0648)_{10}$，其 $r = 10$，$n = 4$，对给定的 **SD** 集合 $\{\bar{6}, \cdots, \bar{1}, 0, 1, \cdots, 6\}$，$\alpha = 6$. 我们按照表 1.3 所经过的转换顺序，可得到 $\mathbf{Y} = (1\bar{4}5\bar{2})_{10}$. 值得注意的是，还可以从另一端开始形成 **SD**；实际上这里没有借位的传播. 正如上例可看到的，所有带符号的数字是可以独立生成的.

表 1.3 一个普通十进制数转换成一个等价的 SD 形式

数字位置 i	4	3	2	1	0	注 释
x_i		0	6	4	8	普通十进制数
d_i		0	$\bar{4}$	4	$\bar{2}$	中间的差
b_i	0	1	0	1		借位数字
y_i		1	$\bar{4}$	5	$\bar{2}$	**SD** 形式

从一个 **SD** 数 $\mathbf{Y}$ 转换到普通基数形式 $\mathbf{X}$ 是靠两个数 $\mathbf{Y}^+$ 和 $\mathbf{Y}^-$ 的相加，这两个数分别由 $\mathbf{Y}$ 中正的数字和负的数字来形成. 上面的例子表示

$$\mathbf{Y} = \mathbf{Y}^+ + \mathbf{Y}^- = (1\bar{4}5\bar{2})_{10}$$

其中