第1章 概 论

1.1 计算机组成原理研究的内容

完整的计算机由硬件及软件组成这一点已经成为常识。因此需要通过一系列软件和硬件课程的学习才能理解什么是计算机，才能建立完整的计算机概念。

但是,什么是软件,什么是硬件,以及如何在计算机中建立它们的分界面就未必是‘常识”了，这需要掌握一定的计算机专业知识才能理解。

数字计算机是通过执行人们给出的指令来解决问题的机器。指令序列称为程序，它描述如何完成一个确定任务。每台计算机都只能识别和直接执行有限的、简单的基本指令。这些基本指令的功能都很简单，比如两个数相加、检查某数是否等于零、将一些数据从计算机内存的某些单元拷贝到其他的单元中或传送到寄存器中等。计算机的这些原始指令共同组成了一种可供人和计算机进行交流的语言，我们称其为机器语言，在数字计算机中它是由 0和 1组成的二进制代码序列。正因为如此，用机器语言编写程序就显得十分困难和乏味。设计一种新的计算机时，人们必须首先决定它的机器语言中包含哪些原始指令。这些原始指令的集合构成了计算机的指令系统。通常，原始指令应尽量简单，兼顾计算机的使用要求和性能要求，以降低实现电路的成本和复杂度。

由于机器语言与人类熟悉的语言（自然语言）相差甚远，所以人们不愿意使用机器语言编写程序，而更喜欢用人类的自然语言直接编写程序，但至少到目前为止，这仍然是一个没有解决的问题。尽管如此，人类却设计出了与自然语言相近的高级语言（如 C 语言、FORTRAN语言等),这些语言可以称为‘人工语言”。用高级语言编写的所有程序都必须在执行前转换成计算机基本指令构成的机器语言程序。这种转换是借助于翻译系统实现的。

有了以上的基本知识，就可以理解软件的概念了。软件是由算法（指明如何做某事的详细指令）及其在计算机中的表示——程序组成的。程序可被存储在硬盘、软盘、CD-ROM或其他存储介质上。但实质上，程序是指令的集合，而不是记录它们的物理介质。

用真正的计算机机器语言编写的程序能直接被计算机的电路执行，不需要经过任何的解释或翻译。这些电路和存储器及输入输出设备加在一起，构成了计算机的硬件。硬件是具体的对象——集成电路、印制电路板、电缆、电源、存储器和打印机等,而不是抽象

的概念、算法或指令。

软件和硬件的分界线就是指令系统（见图 1.1）。指令系统以上的部分是软件，指令系统以下的部分是硬件。

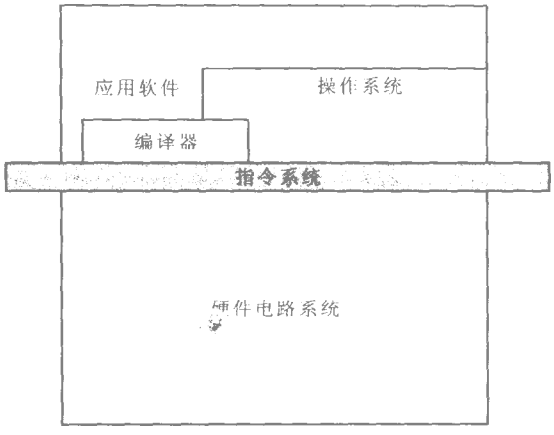


图 1.1 指令系统——软件和硬件的分界线

高级语言程序属于应用软件。通常，高级语言（如 C语言）编写的源程序不能被计算机的指令系统直接执行，需要经过编译后才能由指令系统执行。

设计一台新计算机时，首先要确定指令系统，在指令系统确定之后，就可以同时进行软件和硬件的设计了。软件由指令系统开始往上面设计，硬件由指令系统开始往下面设计(见图 1.2)。

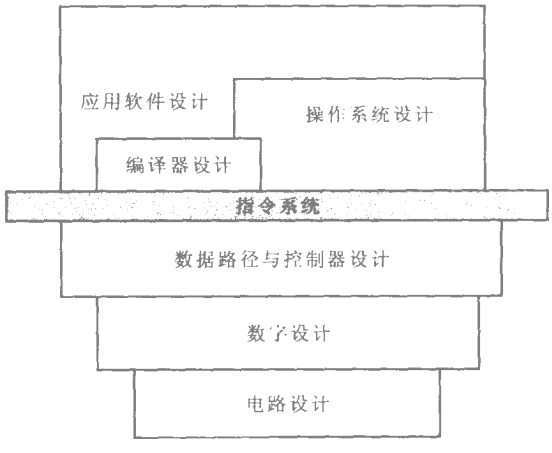


图 1.2 由指令系统决定软件和硬件的设计

对于指令系统中的每一条指令的执行，则需要用硬件来实现。计算机组成原理更关心的是硬件设计。在指令系统的下面，首先遇到的是数据路径与控制层次。在这一层次要描述数据路径的建立、控制器的设计（例如是硬布线或微程序）。再往下便是数字设计

层次，该层完成实现数据路径相关部件的数字逻辑设计。数字设计之后便是电路设计。计算机组成原理将把更多的注意力放在数据路径与控制层次上（见第 4 章）。

在本课程要研究的最底层——数字设计层，我们感兴趣的对象是门。每个门可以有一个或多个数字输入端(由 0 或 1 表示的信号），可计算并输出这些输入的一些简单逻辑函数(如与和或)的结果。门最多由几个晶体管构成，几个门可组成 1 位存储器，存放一个 0 或 1。1 位存储器可组合成(例如 16、32 或 64 一组）寄存器。每个寄存器可存放一个不超过某个最大值的二进制数。门本身也可组成主要的计算部件。在研究运算器时，会涉及到这一层次(见第 2 章)。

电路设计通常属于电子工程领域（不属于本书讨论的范围）。在该层，设计者见到的是单个的晶体管，这些对计算机设计人员来说是最底层的要素。至于里面的晶体管是如何工作的，则是固体物理研究的课题。

当然，计算机通常要与输入输出设备(如显示器、打印机等)交换信息，因此输入输出设备也是计算机系统要研究的对象。但是输入输出设备种类繁多，工作原理各不相同，深入理解输入输出设备需要许多其他专业知识。在“计算机组成原理”课程中不可能作详细的讨论。

1.2 计算机组成和体系结构

计算机组成(Computer Organization)和计算机体系结构(Computer Architecture)这两个概念对于想了解计算机系统的人来说是很重要的。虽然很难给出这两个术语的精确定义，但对它们所涉及的领域则存在着共识。

一般认为，计算机体系结构是指那些对程序员可见的系统属性。换句话说，这些属性直接影响到程序的逻辑执行。例如，计算机体系结构的属性包括指令系统、表示各种数据类型(例如整型、字符型)的比特数、输入／输出机制以及内存寻址技术。

计算机组成指的是实现计算机体系结构规范的操作单元及其相互连接。计算机组成的属性包括那些对程序员透明的硬件细节，如控制信号、存储器使用技术等。

可以通过一个例子来说明计算机体系结构和计算机组成的区别。例如，计算机是否有乘法指令是计算机体系结构设计问题。而这条指令是由特定的乘法单元实现，还是通过重复使用系统的加法单元来实现，则是一个计算机组成问题。决定使用哪种计算机组成需要考虑预期使用乘法单元的频度，考虑两种方案的相对速度，还需要考虑一个特定乘法单元的成本和物理尺寸等因素。

计算机体系结构则是从程序员（特别是系统程序员）的角度观察计算机系统具有哪些特征。计算机体系结构是计算机硬件和软件之间的接口。一个计算机体系结构可以用不同的计算机组成来实现。

计算机制造商往往提供一系列型号的计算机，它们都有相同的计算机体系结构，但计算机组成不同。一种计算机体系结构可能存在多年，但它的计算机组成则随着技术的进步而不断更新。技术的更新不仅影响了计算机的组成，还导致了更强大且更丰富的计算机体系结构。

1.3 冯·诺依曼计算机

要知道什么是冯·诺依曼(John Von Neumann)计算机，还得从世界上第一台通用电子数字计算机谈起。世界上第一台通用电子数字计算机的英文全名是 Electronic Numerical Integrator And Computer(电子数字积分计算机,缩写为 ENIAC)。

研制通用电子数字计算机(ENIAC)的目的是满足美国战时(第二次世界大战)的需要。美国军队的弹道研究实验室(BRL)——一个负责开发新式武器的射程和弹道表的机构，在提供数据表的精确性和及时性上遇到了困难。没有这些发射表，新式的武器和火炮对炮手来说是没有用处的。BRL 雇用了 200 多人，大多数是妇女。他们使用桌面计算器求解所需的火炮公式。为一件武器提供数据表将耗费几小时，甚至几天的时间。

美国 Pennsylvania 大学教授 Mauchly 和他的研究生 Eckert 提出用电子管制造通用计算机的设想,用以满足 BRL 的应用需求。1943 年，这个计划被军方采纳,ENIAC 项目开始启动。最终的机器体积庞大,重约 30 吨,占地 1 500 平方英尺(1 英尺 = 0.3048 m),采用了约 18 万个电子管，它工作时消耗 140 kW 的功率。但它比电子机械计算机要快许多，每秒钟能执行 5 000 次加法。

ENIAC 完成于 1946 年，是十进制而不是二进制机器。ENIAC 的主要缺点是，它必须通过手工设置分布于各处的 6 000 个开关和连接森林般的插头及众多的插座来编程。这显然是一件十分枯燥和乏味的工作。

为克服这一困难,冯·诺依曼提出了存储程序的概念，其要点如下：

1. 采用二进制代码表示指令和数据

数据和指令在代码的形式上没有区别，都是 0 和 1 组成的二进制数据，但其含义不同。程序信息本身也可以作为被处理的对象(如编译)。

2. 采用存储程序方式

将事先编制好的程序(包含指令和数据代码)存入主存储器中，计算机在程序运行时就能够自动地、连续地从存储器中依次取出指令并且执行。这是计算机高速自动运行的基础。

由于冯·诺依曼第一次描述了这一思想，因而这种机器被命名为冯·诺依曼(John Von Neumann)机。将事先编制好的程序(包含指令和数据代码)存入主存储器中,由 CPU 调用执行,这在现在看起来已经是普通的常识,而在当时,却是一个伟大的贡献。

计算机的工作体现为程序的执行。计算机功能的扩展很大程度上体现为存储程序的扩展。

冯·诺依曼机的这种工作方式,称为控制流(指令流)驱动方式。在这种方式下,始终以控制信息流(即指令流)为驱动工作的因素，而数据信息流则是被动地被调用处理。如何控制指令流呢？设置一个程序计数器 PC(Program Counter),让它存放当前指令所在的存储单元的地址。对顺序执行，每取出一条指令后 PC 的内容自动增加一个常量，指向下一条指令的地址。对程序的转移，就将转移后的地址送入 PC。PC 就像一个指针,一直指示着程序的执行进程，即指示着控制流(指令流)的形成。

1.4 计算机的发展简史

计算机的发展经历了第零代——机械计算机,第一代——电子管计算机(1945～1955年),第二代——晶体管计算机(1955～1965 年),第三代——集成电路计算机(1965～1980年),第四代——超大规模集成电路计算机(1980～ ?),见表 1.1。

表 1.1 计算机发展简史

年代	机器名称	制造者	说 明
1834	Analytical Engine	Babbage	建造数字计算机的第一次尝试
1936	Z1	Zuse	第一台使用继电器的计算机器
1943	COLOSSUS	英国政府	第一台电子计算机
1944	Mark I	Aiken	第一台美国通用计算机
1946	ENIAC I	Eekert/Mauchley	现代计算机历史从此开始
1949	EDSAC	Wilkes	第一台存储程序的计算机
1951	Whirlwind I	M.I.T.	第一台实时计算机
1952	IAS	Von Neumann	大多数现代计算机还用的设计
1960	PDP-1	DEC	第一台小型机(销售 50 台)
1961	1401	IBM	非常流行的小型商用机
1962	7094	IBM	20 世纪 60 年代早期的主流科学计算用机
1963	B5000	Burroughs	面向高级语言的第一台计算机
1964	360	IBM	系列机的第一个产品
1964	6600	CDC	第一台用于科学计算的超级计算机
1965	PDP-8	DEC	第一台占领市场的小型机(销售 50 000 台)
1970	PDP-11	DEC	20 世纪 70 年代的主导小型机
1974	8080	Intel	第一台在一个芯片上的 8 位计算机
1974	CRAY-1	Cray	第一台向量超级计算机
1978	VAX	DEC	第一台 32 位超级小型计算机
1981	IBM PC	IBM	开创现代个人计算机新纪元
1985	MIPS	MIPS	第一台商用 RISC 机
1987	SPARC	Sun	第一台基于 SPARC 的 RISC 工作站
1990	RS6000	IBM	第一台超标量体系结构计算机

1.5 计算机的应用

计算机的早期应用主要集中在数值计算领域中，比如，工程设计中要用到的各种计算、数学方程的求解等。而现在,计算机的应用更多的是信息处理,如 Internet 上的信息浏

览、文字处理等。在现代社会里，几乎没有哪个领域不使用计算机。人类已经越来越离不开计算机了。

不同的应用领域对计算机的要求也是不同的。在数值计算领域，人们追求的是高速度,因此,人们需要研究开发运算速度极高的巨型机 而在信息处理应用领域,为了使广大的普通用户能够接受，计算机的开发商在追求高性能的同时，必须考虑价格因素（如个人计算机）在控制领域,用得最多的要数单片机,其功能简单,价格便宜,体积很小。不同的应用领域，推动着计算机沿着不同的方向发展，可以说，应用是推动计算机发展的最大动力。

习 题

1. 人们在研究计算机系统时，常将其划分成一些层次，这有什么好处？
2. 简述计算机体系结构和计算机组成之间的区别。

第 2 章　运算方法基础与运算器

在计算机中，采用数字化方式来表示各种信息，通常采用二进制。因此，要想了解计算机的基本组成原理，就必须熟悉二进制的运算。数值运算是由 CPU 中的算术逻辑运算部件(ALU)完成的。ALU 可以完成加、减、乘、除四则运算，与、或、非、逻辑异或运算，移位、计数、取补等运算。这些基本运算构成了所有复杂运算的基础。本章讨论运算方法和 ALU 的逻辑结构。

2.1　数的机器码表示方法

我们在日常生活中经常使用的是十进制数，但是在表示时间时，则用十二进制或二十四进制表示小时，六十进制表示分和秒。在计算机中，除了二进制之外，也通常使用八进制、十六进制等，但基础都是二进制。

研究机器的数码表示，首先应当理解真值与机器数的概念。

真值：即数的实际值。真值就是人们常常习惯书写的数值，一般带有正负号（如 -010,1010）.

机器数（或机器码）：将真值按某种编码方式进行编码后的数值。

机器数又分为无符号机器数和带符号机器数，分别简称为无符号数和带符号数。

无符号数 无符号数的每一位都是数值位(没有符号位)，每位的权值不一样。

二进制有两个代码，用 0 和 1 表示 八进制有 8 个代码，用 0,1,2,3,4,5,6,7 表示。十进制数要有 10 个代码来区别各自的值，这就是我们非常熟悉的 0,1,2,3,4,5, 6,7,8,9;十六进制要求有 16 个代码，前 10 个借用了十进制的代码，后 6 个目前通常由字母 A,B,C,D,E,F 来表示。

例如：无符号二进制数 10011011，其真值的十进制形式为：

$$1\times2^7+0\times2^6+0\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0=155$$

除了无符号数以外，人们还要处理带符号的数据，因此会遇到以下的问题：如何表示负数？

带符号数：带符号数是将符号位和数值一起编码表示数的表示方法（如原码、补码、反码等)，即数的实际值在计算机内的表达形式(编码)。

例如 真值 -010 在计算机中应当如何表示？我们知道，计算机中的数据值有两个（0

和 1)，不能直接表示负号“ - ”。因此只能用两个代码(0或 1)之一来表示负号“ - ”,而另一个代码表示“ + ”号。

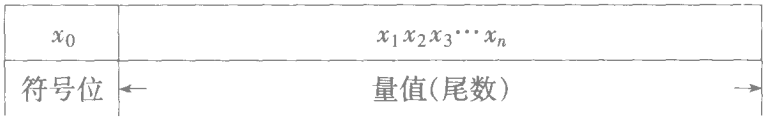
显然,与无符号数不同,带符号数虽然形式上也是由 0或 1 构成的字符串，但并非每一位都是数值位,其中有一位是符号位(通常是最高位)。

计算机数据通常用二进制来表示。一位二进制数可以表示的数据值有两个： 0 和 1。我们通常使用“ 字(word) ”来表示计算机能同时运算的二进制数。通常约定一个“ 字节 ”为 8 个二进制位，一个字由若干个字节组成。目前，关于“ 字 ”的位数定义尚无统一的标准。例如,较早期的计算机字为 16 位,现在多为 32 位或 64 位。尽管随着计算机技术的发展，字的位数在不断增加，但总是有限的，因此，它能表示的数的范围也是有限的。为了能够表示小数和整数，计算机中常用的数据格式有两种：定点格式和浮点格式，下面分别对此加以介绍。

2.1.1 计算机带符号定点数的表示方法

定点格式约定机器中所有数据的小数点位置是固定不变的。

设定点数 $x = x_0x_1\cdots x_n$，在定点机中可表示成如下形式：



如 x 为纯小数,小数点位于 x_0 与 x_1 之间，$x_1, x_2, \cdots, x_n$ 均为 0时，x 的绝对值最小，即 $|x|_{min} = 0$ 当 x 的各位均为 1时，x 的绝对值最大,即 $|x|_{max} = 1 - 2^{-n}$.

于是，定点小数的表示范围为：$0 \leqslant |x| \leqslant 1 - 2^{-n}$.

如 x 为纯整数，那么小数点位于最低位 x_n 的右边,此时同样 $|x|_{min} = 0$(当 $x_1, x_2, \cdots, x_n$ 各位均为 0 时）;当 x 的各位均为 1 时，x 的绝对值最大。于是定点整数的表示范围为：$0 \leqslant |x| \leqslant 2^n - 1$.

定点数的表示方法虽然简单，但是数据的表示范围十分有限。

2.1.2 计算机浮点数的表示方法

在考虑数据的表示范围时，人们会遇到以下的问题：

(1) 一个计算机字所能表示的最大的数是什么？

(2) 当计算结果比计算机所能表示的最大数还要大时会发生什么情况？

(3) 如何表示小数和实数？

浮点表示法：把一个数的有效数字和数的范围在计算机的一个存储单元中分别予以表示。这种把数的范围和精度分别表示的方法，相当于数的小数点位置随比例因子的不同而在一定的范围内可以自由浮动，称为浮点表示法。

例 2.1 一个十进制数可表示为

$$N_1 = 3.14159 = 0.314159 \times 10^1 = 0.0314159 \times 10^2$$

同样,一个二进制数可表示为

比例因子 10^{l} 及 2^{-l} 要分别存放在机器的某个存储单元中。

一般地说，一个任意二进制数 N 可以表示成

$$N = 2^{E} \times M$$

一个任意的 R 进制数可以表示成

$$N = R^{E} \times M$$

其中，M 称为浮点数的有效数，是一个小数；E 是比例因子的指数，称为浮点数的阶码，是一个整数，当 E 变化时，数 N 的有效数 M 中的小数点位置也随之左右移动； R 称为比例因子的基数，对于确定了计数制的机器来说是一个常数，通常，$R = 2, 8, 16$ 等等。

机器中表示一个浮点数的形式：

■■ 给出有效数，有效数以小数表示(有效数决定了浮点数的表示精度)；

给出阶码，阶码用整数表示，阶码指明小数点在数据中的位置，决定浮点数的表示范围。

浮点数的优点 数表示范围大(与定点数相比)。

如果按 标准考虑浮点数据的数据格式，则数符在最高位，阶符隐含。以后会详细介绍这方面的内容。

二进制带符号数的表示方法

原码表示方法

原码表示法是用机器数的最高一位代表符号，其余各位给出数值的绝对值的表示方法。通常，最高位为 代表正数，最高位为 代表负数。原码表示法也称为符号一绝对值表示法。

定点小数的原码表示

定义

式中， 是真值， 是 的原码。

例 2.2 如 则

如 则

一般形式 如 $x = +0.x_1x_2\cdots x_n$，则 $[x]_{原} = 0.x_1x_2\cdots x_n$；

如 $x = -0.x_1x_2\cdots x_n$，则 $[x]_{原} = 1.x_1x_2\cdots x_n$。

定点整数的原码表示

定义 设字长为 位，定点整数原码表示为

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

例 如 假设原码字长共有 位，则

$$[x]_{原} = x = 01001$$

如 $x = -1001$,假设原码字长共有 5 位,则

$$[x]_{原} = 2^4 + |x| = 10000 + 1001 = 11001$$

一般地(设字长为 $n+1$ 位):

如 $x = +x_1x_2\cdots x_n$,则 $[x]_{原} = 0x_1x_2\cdots x_n$;

如 $x = -x_1x_2\cdots x_n$,则 $[x]_{原} = 2^n + |x| = 1x_1x_2\cdots x_n$。

3. 原码的性质

(1) 真值 0 在原码表示中有两种形式,以定点整数为例:

$$[+0]_{原} = 0000$$

$$[-0]_{原} = 1000$$

(2) 原码表示的定点小数,其表示范围为

$$-1 < x < 1 \text{ ,即 } |x| < 1$$

原码表示的定点整数,其表示范围为

$$-2^n < x < 2^n \text{,即 } |x| < 2^n$$

(3) 可用数轴表示出原码表示的范围和可能的代码组合。以定点整数为例,设机器字长为 $n+1$ 位,如图 2.1 所示,数轴上方表示的是原码的代码组合,下方注明原码对应的真值。

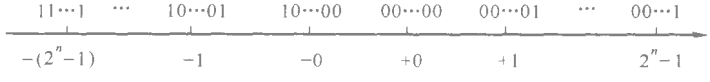


图 2.1 原码表示的数据沿数轴的分布示意图

4. 原码的特点

优点:(1)简单直观;(2)乘除运算简单(数值部分绝对值直接运算,单独处理符号位)。

缺点:原码加减运算较复杂。

例如用如下两个原码表示的数做加法运算:11101 + 01011 ,即(−13 + 11),表面上做加法,事实上由于两数异号,故做减法,即原码表示的数在做加法运算时,不仅要根据指令规定的操作性质(加或减),还要根据两数的符号,才能决定实际操作是加还是减。为了解决这一困难,人们找到了补码表示法。

2.2.2 补码表示方法

1. 概念与定义

先分析两个十进制数的运算:

$$58 - 35 = 23$$

$$58 + 65 = 123 = 100 + 23$$

如果使用两位十进制的运算器,在做加法 58 + 65 时,结果中的 100 超出该运算器的表示范围时,将会被自动舍去,运算器只能表示出结果为 23。在做减法 58 − 35 时,若用加

法 $58+65$ 能得到同样的结果 23。

在数学上可用同余式(同余式的概念见本章附录)来表示,即

$$\begin{aligned}58-35 &= 100+(58-35)\ (\mathrm{mod}\ 100)\\ &= 58+(100-35)\ (\mathrm{mod}\ 100)\end{aligned}$$

上式的 100 称作两位十进制运算器的模或溢出量。 $\mathrm{mod}\ 100$ 就指模为 100。以上运算称为有模运算。上式也可写成

$$58+(-35)=58+65\ (\mathrm{mod}\ 100)$$

减法变成了加法,或

$$-35=65\ (\mathrm{mod}\ 100)$$

也就是说,(-35)的补码(相对模 100 而言)是 $+65$。

结论 在引入补码后,减法可转换为加法。

注意:计算机中数的表示受计算机字长的限制,其运算都是有模运算。模在机器中是表示不出来的,若运算结果超出能表示的数值范围,则会自动舍去溢出量(模),只保留小于模的部分。

设字长为 $n+1$ 位,对定点小数 $x_0.x_1x_2\cdots x_n$,其溢出量为 $(10.\underbrace{00\cdots0}_{n})_2$ 即 2,则以 2 为模。

对定点整数 $x_0x_1x_2\cdots x_n$,其溢出量为 $1\underbrace{00\cdots0}_{n+1}$ 即 2^{n+1},则以 2^{n+1} 为模。

下面研究二进制数的补码。

补码定义 1 对一个给定的真值 x(二进制整数)以及一个正整数 M(M 为模,是二进制数),若 $-M/2\leqslant x<M/2$,则同余方程式

$$\begin{cases}y=x\ (\mathrm{mod}\ M)\\ 0\leqslant y<M\end{cases}\tag{2.1}$$

的解 y 称为真值 x 的关于模 M 的补码,并将该解 y 记为 $[x]_{补}$。

定义中的 M 是 2 的整数幂。M 的选择与机器的字长有关。

这里,应该把 $[x]_{补}$ 看成一个机器码。从形式上看,$[x]_{补}$ 的各个位或为 0 或为 1,其本身没有正负符号,可以看成是无符号的二进制数。但是,与 $[x]_{补}$ 对应的真值 x 是有符号的数据。以上定义是本书作者给出的。

定理 2.1 当 $x\geqslant 0$ 时,$[x]_{补}=x$ 当 $x<0$ 时,$[x]_{补}=M+x$。

证 同余方程式 $y=x(\mathrm{mod}\ M)$ 的解是 $\{x+kM,k\in\mathbf{Z}\}$,其中 $\mathbf{Z}$ 是整数。显然,由补码的定义(2.1)式知,对于 $-M/2\leqslant x<M/2$,当 $x\geqslant 0$ 时,$[x]_{补}=x$,此时,$0\leqslant[x]_{补}<M$ 对于 $-M/2\leqslant x<M/2$,当 $x<0$ 时,$[x]_{补}=M+x$,此时 $0<[x]_{补}<M$。综合上面两种情况可知,对于 $-M/2\leqslant x<M/2$,$0\leqslant[x]_{补}<M$,见(2.1)式。

证毕

定理 2.2 对于给定的 x 及 M,$[x]_{补}$ 是唯一的。

证 可由定理 2.1 直接得到。

证毕

利用定理 2.1,可以将补码定义写成:

补码定义 2 一个数 x 的补码记作 $[x]_{补}$，设模为 M，$-M/2 \leqslant x < M/2$，其补码表示定义为

$$[x]_{补} = M + x \pmod{M} \tag{2.2}$$

为了以后方便起见，(2.2)式省略了 $0 \leqslant [x]_{补} < M$（下同），上式是一个包含正数在内的统一定义式。若 $x \geqslant 0$，则模 M 作为溢出量被舍去，因而正数补码就是其本身。

应该指出的是，在数学上定义同余式时，其变量都是整数。计算机中的数据都可以看成是字长有限的整数。所谓的定点小数中的小数点是人们为了描述小数方便而人为加上的，计算机的数据中并不存在真正的小数点。从另一个角度看，即使把这些数据看成小数，由于计算机的字长是确定的，所以，可以把这些数据放大相同的倍数，以使它们都成为整数。正是由于上述原因，数学上的同余式概念完全可以用到这里的定点小数。

例 2.4 如 $x = 0.1011, M = 2$，求 $[x]_{补}$.

解 $[x]_{补} = M + x \pmod 2 = 2 + 0.1011 \pmod 2 = 0.1011$。

可见，正数的补码形式与其原码形式相同。虽然其符号位在形式上与原码相同，都是用 0 表示正，但这个 0 是通过模 2 运算求得的，它是数值的一部分，可直接参加运算。

例 2.5 如 $x = -0.1011, M = 2$，求 $[x]_{补}$.

解 因为 $x < 0$，所以

$$\begin{aligned}[x]_{补} &= M + x \pmod 2 = M - |x| \pmod M \\ &= 2 + (-0.1011) = 2 - 0.1011 = 1.0101 \pmod 2\end{aligned}$$

这里省略了 $0 \leqslant [x]_{补} < 2$。可见负数的补码形式与其原码形式不同，虽然其符号位在形式上与原码相同，都是用 1 表示负，但这个 1 是通过模 2 运算得到的，它是数值的一部分，可直接参加运算。正如前述的 +65 表示 −35 一样，负号在映射后已包含在数值之中。

2. 定点小数的补码表示

下面假设机器的字长为 $n+1$ 位。

由补码定义 1 及定义 2 有

$$[x]_{补} = \begin{cases} x & 0 \leqslant x < 1 \\ 2 + x = 2 - |x| & -1 \leqslant x < 0 \end{cases} \tag{2.3}$$

式中，$[x]_{补}$ 为机器数（补码），x 为真值。

一般情况

正数 $x = +0.x_1x_2\cdots x_n$，有 $[x]_{补} = 0.x_1x_2\cdots x_n$

负数 $x = -0.x_1x_2\cdots x_n$，有 $[x]_{补} = 10.00\cdots00 - 0.x_1x_2\cdots x_n$

↑ 对应十进制的 2

3. 定点整数的补码表示

由补码定义 1 及定义 2 可知，对定点整数，补码可表示成

$$[x]_{补} = \begin{cases} x & 0 \leqslant x < 2^n \\ 2^{n+1} + x = 2^{n+1} - |x| & -2^n \leqslant x < 0 \end{cases} \tag{2.4}$$

4. 补码与原码、真值间的转换

(1) 由真值、原码转换为补码

例 2.6 设机器字长为 5 位，采用定点整数表示，$x = +110$，求 $[x]_{原}$，$[x]_{补}$.

解　因为 $n+1=5$，所以 $n=4$，

$$[x]_{原}=0,0\ 1\ 1\ 0$$
$$[x]_{补}=0,0\ 1\ 1\ 0$$
$$\uparrow\ \uparrow\quad\quad\uparrow$$
$$x_0\ x_1\ \cdots\ x_4$$

显然，正数的原码与补码形式相同。

例 2.7　设机器字长8位并用定点整数表示，模为 2^8，若 $x=-110$，求 $[x]_{原}$、$[x]_{补}$。

解　因为 $n+1=8$，所以 $n=7$，

$$[x]_{原}=10000110$$
$$[x]_{补}=M+x=2^8+(-110)=(100000000)_2-110$$
$$=11111010$$

下面介绍由负数的原码转换为补码有两种实用的转换方法。

第一种方法是：符号位保持为“1”，其余各位变反，并在末位加 1（在定点小数中，末位加 1 相当于数值加 2^{-n}）。常将此法简称为“变反加 1”。

例 2.8　（重复上例）

$$[x]_{原}=\quad 10000110$$
变反　$\quad 11111001$
末位加 1　$+\quad\quad 1$
$$[x]_{补}=\quad 11111010$$

第二种方法是 符号位保持为 1，其余部分自低位向高位数，第一个 1 以及以前数过的各位 0保持不变，以后的各高位按位变反。

例 2.9　（重复上例）

$$[x]_{原}=1\quad 00001\quad 10$$
$$[x]_{补}=1\quad 11110\quad 10$$
不变　变反　不变

(2) 由补码求原码与真值

采用补码表示的计算机，其运算结果也是补码形式，不直观，因此有时需要将补码转换为真值或原码表示。转换方法是 对于正数，原码与补码相同，其真值在略去正号后，形式上与机器数相同 对于负数，保持符号位为 1，其余各位变反末位加 1（或采用第二种转换方法）即得到原码表示，将负数原码符号恢复为负号，即得到真值表示。

例 2.10　如 $[x]_{补}=00000110$，则 $[x]_{原}=00000110$，$x=+110$

如 $[x]_{补}=11111010$，则 $[x]_{原}=10000110$，所以 $x=-110$

$[x]_{原}$ 的具体计算如下：

$$[x]_{补}=\quad 11111010$$
变反　$\quad 10000101$
末位加 1　$+\quad\quad 1$
$$\quad\quad 10000110$$

5. 补码的性质

(1) 补码的最高位是符号位，在形式上同于原码，0 表示正，1 表示负。但应注意，原

码的符号位是人为地定义 0正 1 负 而补码的符号位是通过模运算得到的,它是数值的一部分，可直接参与运算。

(2) 正数的补码表示在形式上同于原码；而负数的补码表示在形式上则不同于原码，可用负数原码求补方法进行转换。

(3) 在补码表示中,真值 0 有唯一的编码,即

$$[+0]_{补}=[-0]_{补}=00\cdots0$$

以定点小数为例,设 $x=+0.0000000, y=-0.0000000,$

按补码定义有

$$\begin{aligned}[x]_{补}&=x=00000000\\[y]_{补}&=2+y\ (\mathrm{mod}\ 2)\\&=10.0000000+(-0.0000000)\ (\mathrm{mod}\ 2)\\&=10.0000000\ (\mathrm{mod}\ 2)\\&=0.0000000\end{aligned}$$

6. 补码的表示范围与代码组合

如图 2.2 所示,以定点整数为例,设机器字长为 $n+1$ 位。

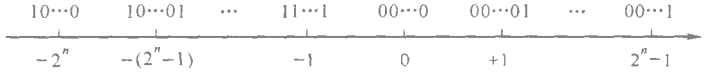


图 2.2 补码表示的数据沿数轴的分布示意图

对比图 2.2 和图 2.1 ,可以看出二者的主要区别是：

(1) 原码表示中有两种 0 的形式，而补码表示只有一种 0 的表示形式,即 00…0。

(2) 与原码表示相比，补码表示的负数域多一种组合，这是由于原码中“ -0”占去了一种组合。因此,对比定点整数,补码中‘ 最负 ’的值可到 -2^n,而原码中‘ 最负 ’的值只能到 $-(2^n-1)$。

2.2.3 反码表示方法

1. 定点小数的反码表示

反码表示法是用机器数的最高一位代表符号， 0 为正,1 为负。反码的数值位是对负数真值的各位取反的表示方法，其数学表达式为（定点小数）

$$[x]_{反}=\begin{cases}x & 0\leqslant x<1\\(2-2^{-n})+x & -1<x\leqslant 0\end{cases}\qquad(2.5)$$

其中,n 代表数值位数(不包括符号位),或者说字长是 $n+1$ 位。

第二式的证明：

设 $x=-0.x_1x_2\cdots x_n$,则有 $[x]_{反}=1.\overline{x}_1\overline{x}_2\cdots\overline{x}_n$。现将 x 的绝对值 $|x|$ 和 $[x]_{反}$ 相加，则得

$$[x]_{反}+|x|=1.11\cdots1=10.00\cdots0-0.00\cdots1=2-2^{-n}$$

所以

$$[x]_{反}=(2-2^{-n})-|x|=(2-2^{-n})+x$$

证毕

一般情况下,对于正数 $x=+0.x_1x_2\cdots x_n$,则有 $[x]_{反}=0.x_1x_2\cdots x_n$,对于负数 $x=-0.x_1x_2\cdots x_n$,则有 $[x]_{反}=1.\overline{x}_1\overline{x}_2\cdots\overline{x}_n$,对于 0,有

$$
\begin{aligned}
[+0]_{反}&=0.00\cdots0\\
[-0]_{反}&=(2-2^{-n})+(-0)\\
&=10.00\cdots0-0.00\cdots1\\
&=1.11\cdots1
\end{aligned}
$$

2. 定点整数的反码表示

对定点整数,反码表示的定义是

$$
[x]_{反}=\begin{cases}x & 0\leqslant x<2^n\\ (2^{n+1}-1)+x & 2^{-n}<x\leqslant 0\end{cases}\tag{2.6}
$$

3. 负数的反码与补码的关系

对定点小数

$$
\begin{aligned}
[x]_{反}&=2-2^{-n}+x\\
[x]_{补}&=2+x
\end{aligned}
$$

所以

$$
[x]_{补}=[x]_{反}+2^{-n}\tag{2.7}
$$

对定点整数

$$
\begin{aligned}
[x]_{反}&=(2^{n+1}-1)+x\\
[x]_{补}&=2^{n+1}+x
\end{aligned}
$$

所以

$$
[x]_{补}=[x]_{反}+1\tag{2.7a}
$$

以上两个关系都表示在最低位加 1。

以上两个公式告诉我们:若要一个负数变为补码,其方法是符号位置 1,其余各位求反(0变 1,1变 0),然后再在最末位加 1。

例 2.11 将十进制真值 $x(-127,-1,0,+1,+127)$ 列表表示成八位二进制数及原码、反码、补码。

解 二进制真值 x 及诸码值列于表 2.1,其中 0 在 $[x]_{原},[x]_{反}$ 中有两种表示。

表 2.1 将十进制真值 x 表示成八位二进制数及原码、反码、补码

真值 x (十进制)	真值 x (二进制)	$[x]_{原}$	$[x]_{反}$	$[x]_{补}$
-127	-01111111	11111111	10000000	10000001
-1	-00000001	10000001	11111110	11111111
0	00000000	00000000	00000000	00000000
		10000000	11111111	
+1	+00000001	00000001	00000001	00000001
+127	+01111111	01111111	01111111	01111111

2.2.4 移码表示方法

移码常用于表示浮点数的阶码，浮点数的阶码是带符号的定点整数。

若浮点数阶码的字长为 $n+1$ 位,假定定点整数移码形式为 $x_0x_1x_2\cdots x_n$ 时,移码的定义是

$$[x]_{移} = x + B \quad -2^n \leqslant x \leqslant 2^n - 1 \tag{2.8}$$

可见移码表示是将数值在数轴上平移了 B ,故称为移码。B 称为偏移量。如果取 $B=2^n$,则 $[x]_{移}$ 的取值范围为 $0 \leqslant [x]_{移} \leqslant 2^{n+1}-1$,如图 2.3 所示。

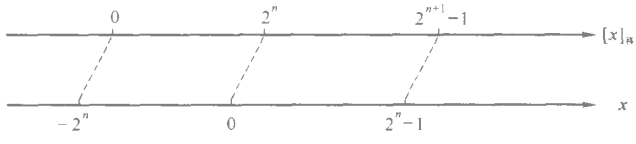


图 2.3 移码与真值的映射图

2.3 字符与字符串的表示方法

在计算机中，除了能处理数值数据信息外，还能处理大量的非数值数据信息，如字符、图像及汉字信息等，这些信息在计算机中也必须用二进制代码形式表示。要想用计算机对文本进行处理，首先遇到的一个问题是如何用二进制数来表示字符，即如何对字符编码。例如,字符“A”的编码是什么,字符“a”的编码又是什么？

本节主要讨论字符数据的表示。在使用各种高级语言或汇编语言编制程序时，除了使用数字外，常常大量使用英文字母及一些符号，因此信息的字符表示是不可缺少的。

字符表示主要涉及到选择哪些常用字符，采用什么编码表示字符，如何压缩编码信息以减少所占有的存储空间等问题。

目前被广泛应用的字符编码是由美国国家标准局（American National Standards)制定的美国信息标准码——ASCII(America Standard Code for Information Interchange)码。ASCII 码共定义了 128 个字符,每个字符的 ASCII 码用 7 位二进制数表示（见表 2.2)。用二进制数来表示字符(例如字符 A),即对字符进行编码,其中定义的有些字符是不能被打印或被显示出来的,我们称之为控制字符(见表 2.3)。

在计算机存储 ASCII 字符或用 ASCII 字符通信时，一个字符通常用一个字节（8 位)来表示。最高位或者设置成 0,或者做奇偶检验(parity)用。

设置最高位的方法如下：若是奇校验(odd parity),应使 8 位中 1 的个数为奇数;偶校验(even parity)时,应使得 1 的个数为偶数。

表 2.2 ASCII 字符

$d_6\ d_5\ d_4$ 二进制位 $d_3\ d_2\ d_1\ d_0$	000	001	010	011	100	101	110	111
	控制字符		符号与数字		大写字母		小写字母	
0 0 0 0	NUL	DEL	SP	0	@	P	‘	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	DC4	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	’	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[	k	{
1 1 0 0	FF	FS	,	<	L	\	l	|
1 1 0 1	CR	GS	–	=	M	]	m	}
1 1 1 0	SO	RS	’	>	N	Λ	n	~
1 1 1 1	SI	US	/	?	O	-	o	DEL

表 2.3 控制字符

字符	解 释	字符	解 释
NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of media
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete