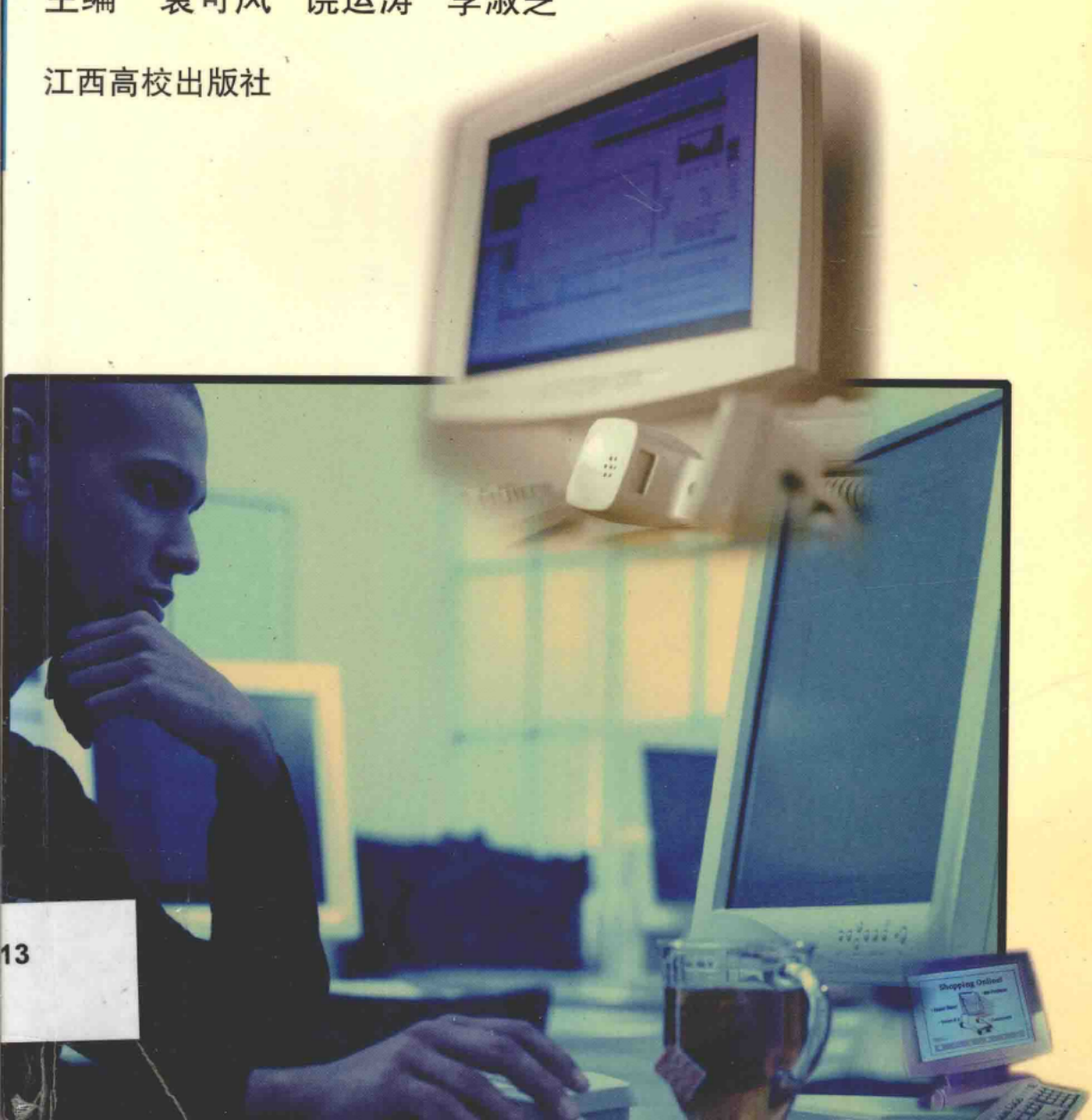
汇编语言程序设计

HuiBian YuYan ChengXu SheJi

主编　袁可风　饶运涛　李淑芝

江西高校出版社



汇编语言程序设计

主编　袁可风　饶运涛　李淑芝

江西高校出版社

图书在版编目(CIP)数据

汇编语言程序设计/袁可风,饶运涛,李淑芝主编 .—南昌:江西高校出版社,2001.12

ISBN 7-81075-257-X

Ⅰ.汇… Ⅱ.①袁…②饶…③李… Ⅲ.汇编语言-程序设计 Ⅳ.TP313

中国版本图书馆 CIP 数据核字(2001)第 094903 号

江西高校出版社出版发行

(江西省南昌市洪都北大道 96 号)

邮编:330046 电话:(0791)8512093,8504319

江西恒达科贸有限公司照排部照排

南昌市光华印刷厂印刷

各地新华书店经销

*

2001 年 12 月第 1 版 2001 年 12 月第 1 次印刷

787mm×1092mm 1/16 21 印张 500 千字

印数:1~2700 册

定价:36.00 元

(江西高校版图书如有印刷、装订错误,请随时向承印厂调换)

前　言

“汇编语言程序设计”是计算机专业及一些相关专业的核心课程之一，是从事计算机研究与应用，特别是软件研究的基础，是计算机专业人员必须接受的最重要的专业基础训练之一。该课程从系统软件和应用软件设计的角度出发，以目前使用最为广泛的 IBM－PC 机为例，详细介绍了汇编语言的基本概念、基本原理和程序设计的常用方法与技术，通过具体实例，叙述了用计算机解决实际问题的全过程，同时还介绍了在 IBM－PC 机上调试运行汇编源程序的方法。

由于汇编语言面向机器，它的学习难度比一些高级语言要大，不易学懂学透。作者根据多年以来教授本课程的丰富经验，对该书在内容的选取、概念的引入、文字的叙述以及例题和习题的选择等方面，都力求遵循面向应用、重视实践、便于自学的原则。全书力求深入浅出、讲清概念、突出重点难点，覆盖了汇编语言程序设计的各类问题。书中精选了大量的程序设计实例阐述汇编语言程序设计的方法和技巧，培养和启发读者的思维能力和创造性。

全书共分十章。由华东交通大学袁可风副教授担任第一主编，负责全书的统稿并编写了其中的第五、六、七、十章；华东地质学院饶运涛教授担任第二主编，编写其中的第三、四、九章；南方冶金学院李淑芝副教授担任第三主编，编写其中第一、二、八章及本书的附录。为了帮助读者更好地掌握汇编语言程序设计的特点，书中结合应用安排了丰富的例题，帮助自己进一步理解与掌握该章节中的重点与难点。希望读者用心地阅读这些例题，从中学习一些基本规律。程序设计是一门实践性很强的科学，既包含复杂的脑力劳动，又是一种极富有创造性的活动；因此，读者在学习过程中应多阅读程序，多编程序，多上机，只有这样，才能真正掌握程序设计的方法与技巧。本书的出版得到了编者各自学校的相关领导的大力支持和热情帮助，在此表示衷心的感谢。本书在编写过程中参考了不少文献资料，在此也向这些文献的作者表示衷心的感谢。

由于作者水平有限，书中不妥或错误之处在所难免，殷切希望广大读者批评指正。

编者

2001 年 12 月

计算机系列教材编写委员会

目　录

第一章　基本知识

1.1　计算机语言系统

1.1.1　机器语言

计算机之所以能自动连续地工作，是采用了“存储程序”的思想。这个思想是美籍匈牙利数学家冯·诺依曼(John Von Neumann)于1946年提出的，一直沿用到今。其中心思想是：将程序和原始数据预先存放在主存储器中，CPU(中央处理器)不断地取指令、分析指令、执行指令，最终获得结果。由此可知，计算机的工作过程就是执行程序的过程。程序是一组相关指令的有序集合，而指令是操纵计算机执行某种操作的命令。

人们用来编制程序，与计算机进行交流的工具就是计算机语言。目前所使用的计算机语言主要分四类：机器语言、汇编语言、高级语言、4GL(第四代)语言。

机器语言是用二进制编码的机器指令的集合及一组使用机器指令的规则。它是CPU能直接识别运行的惟一语言。机器语言中的每一条语句即为机器指令，机器指令与CPU有着密切的关系。通常，CPU的种类不同，对应的机器指令也就不同。但同一个系列的CPU指令集(指令系统)往往具有良好的向上兼容性。例如，Intel 80386指令集包含了8086指令集。机器指令在形式上表现为二进制编码，其一般形式为：

操作码	地址码

操作码指出要进行的操作或运算，如加、减、乘、除、传送、移位等；地址码指出参与操作或运算的对象，也指出操作或运算结果存放的位置，可为寄存器编号、存储单元地址和直接的数据值等。例如，将偏移地址为100的字存储单元中的内容加5，再送回该存储单元中去，如果用Intel 8086的机器指令来完成该操作，则相应的机器指令序列为：

10000011
00000110　} 16位的操作码，表示“加”运算，并指出了获得两个加数的寻址方式

01100100
00000000　} 16位的地址码，指出第一个加数(称目的操作数)存放单元的偏移地址为100(64H)，相加的结果也存入该单元

00000101　指出第二个加数(称源操作数)是直接操作数5

几乎没有人能够直接看出以上程序片段的功能，原因是0、1代码表示的机器指令难以理解记忆。故机器语言程序虽然能被CPU识别，无须翻译且执行速度快，但是人们早已不用它编程，主要是基于它的以下几个缺点：

(1)用机器语言编程难度大，调试困难，需要对硬件有相当的了解。

(2)用机器语言编制的程序可读性、可移植性、通用性都较差。

1.1.2 汇编语言

为了克服机器语言的上述缺点，人们采用便于记忆、并能描述指令功能的符号来表示指令的操作码，这些符号被称为指令助记符。助记符一般是说明指令功能的英语词汇或者词汇的缩写，同时也用符号表示操作数，如 CPU 的寄存器、存储单元地址等。这种用符号书写的、其主要操作与机器指令基本上一一对应的、遵循一定语法规则的计算机语言就是汇编语言。用汇编语言编写的程序称为汇编语言源程序。本教材主要介绍 Intel 8086 宏汇编语言。例如，前面例子中的机器语言程序片段用 8086 宏汇编语言来书写则为：

```
ADD WORD PTR [100],5
```

其中，ADD 是指令的助记符，[100]表示第一个加数及结果的存放单元都是当前数据段，偏移地址为 100 的单元，“WORD PTR”表明这个目的操作数是 16 位的二进制数，而第二个加数是 5，相加的结果存入目的操作数所在的单元中。

用指令助记符、地址符号等符号表示的指令称为汇编格式指令。显然，汇编格式指令比二进制编码的机器指令要容易掌握得多，用汇编语言编写的程序与机器语言编写的程序相比具有如下优点：

(1)更易于理解和记忆。

(2)调试和维护更容易。

(3)所占存储空间、执行速度与机器语言相仿。

由于 CPU 能够直接识别的惟一语言是机器语言，所以用汇编语言编写的源程序必须被翻译成用机器语言表示的目标程序后才能由 CPU 执行。这种把汇编语言源程序翻译成目标程序的语言加工程序称为汇编程序。汇编程序进行翻译的过程叫做汇编。在这里，汇编程序相当于一个翻译器，它加工的对象是汇编语言源程序，而加工的结果是目标程序。它们三者之间的关系如图 1.1 所示。

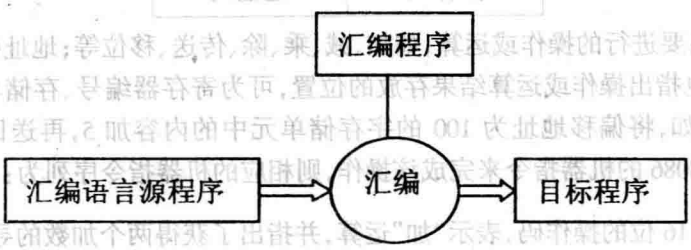


图 1.1 汇编过程示意图

为了能让汇编程序正确地完成翻译工作，必须在汇编源程序中告诉汇编程序，源程序应该从什么位置开始放置，汇编到什么位置结束，数据应放在什么位置，数据的类型是什么，留多少内存单元作临时存储区等。这些工作由伪指令(又称汇编控制命令)来完成。伪指令用法见第三章介绍。

1.1.3 高级语言

汇编语言也是面向机器的语言，汇编语言程序在可移植性、通用性方面受到一定的限制。汇编级程序员对硬件也要求有一定的了解，所以汇编语言编程对普通用户是望而却步

的。近几十年出现了各种接近于自然语言描述的程序设计语言，如 BASIC, FORTRAN, PASCAL, C, COBOL 等高级程序设计语言。使用这些程序设计语言编制程序简单易学，便于调试、维护，且程序设计人员可以不考虑具体计算机的结构特点，只需遵照某种程序设计语言的语法规则和解决问题的具体算法就可以编制程序。例如，用 C 语言描述前面的例子可以写成如下程序片断：

```
int x;
x = x + 5;
```

可见这种描述与自然语言相近，非常形象直观，普通用户比较容易入门。用高级语言编制的程序与用汇编语言编制的程序相比具有以下优点：

(1)编程、调试、维护更容易。

(2)高级语言的面向过程性使高级语言程序的可移植性、通用性更强。

(3)更容易推广使用。

但是，正因为这些高级语言与计算机结构无关，因此也就不能充分利用计算机的结构和特点(如某些寄存器的特殊功能，处理器状态字，一些特殊指令功能等)；而且用高级语言编制的源程序，必须经过编译程序翻译成目标代码才能运行；通常由此产生的目标代码较长，占用较多内存空间，所以这样的目标代码运行时间也较长。在某些情况下，用高级语言编制的程序不能满足要求时，应当使用接近机器语言的汇编语言编制程序，它既可以弥补高级语言的某些不足，又可以改善用机器语言编制程序的困难。

1.2 进一步理解汇编语言

1.2.1 为什么要学习汇编语言

凡是学过一门高级语言的用户，都会有高级语言“易学好用”的感觉。这是因为这些语言的语句是面向数学语言或自然语言的，因此容易接受、掌握。相对来说用汇编语言编制程序比用高级语言要困难些。既然如此，为什么至今还要学习和使用汇编语言呢？

(1)学习和使用汇编语言可以从根本上认识、理解计算机的工作过程。因为一台计算机执行一个任务，从本质上说就是执行一个计算机语言程序。通过汇编语言编程，可以更清楚地了解计算机是怎样完成各种复杂的工作的。在此基础上，程序设计者能更充分地利用机器硬件的全部功能，发挥机器的长处。

(2)现在的计算机系统中，某些功能仍然是靠汇编语言程序来实现的。例如机器自检、系统初始化、实际的输入输出设备的操作，至今仍然是用汇编语言编制的程序来完成的。

(3)汇编语言程序的效率通常高于高级语言程序。这里的“效率”是指程序的目标代码的长短和程序运行的速度。所以在节省内存空间和提高程序运行速度的重要指标的场合(如实时过程控制)，常常用汇编语言来编制程序。

鉴于以上理由，现在许多高级语言(如 C 语言等)都设置有与汇编语言程序接口的功能，以便于用户用汇编语言编制某些子程序，完成与机器联系紧密的特定功能，提高高级语言程序的效率。汇编语言程序设计是从事计算机研究与应用，特别是软件研究的基础，对于从事计算机研制和应用的广大科技工作者来说，掌握汇编语言及其程序设计技术是非常重

要的。

1.2.2 汇编语言程序的建立及汇编过程

图 1.2 表示了对汇编语言程序的处理过程。首先用编辑软件(如行编辑软件 EDLIN 或全屏幕编辑软件 EDIT)建立汇编语言源程序(扩展名为.ASM),源程序经过起翻译作用的汇编语言的翻译,转换成用二进制代码表示的目标文件(扩展名为.OBJ)。在转换过程中,汇编程序将对源程序进行二遍扫描,如果源程序中有语法错误,则汇编结束后,汇编程序将指出源程序中的错误,用户还可以用编辑软件来修改源程序中的错误,最后得到无语法错误的.OBJ 文件。

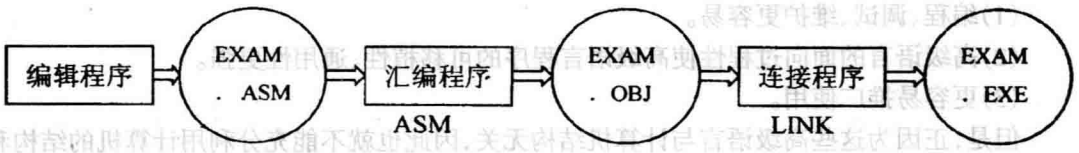


图 1.2 汇编语言程序的建立及汇编过程

.OBJ 文件虽然已经是二进制文件,但它还不能直接上机运行,必须经过连接程序(LINK)把目标文件与库文件或其他目标文件连接在一起形成可执行文件(.EXE 文件),这个文件可以由 DOS 装入内存,在机器上运行。

汇编程序的主要功能是:

(1)二遍扫描检查源程序。

(2)检测出源程序中的语法错误,并给出出错信息。

(3)产生源程序的目标程序,并可以有选择地给出列表文件(同时列出汇编语言和机器语言的文件称为.LST 列表文件)。

(4)展开宏指令。

(5)处理伪指令。

1.3 进位计数制及其相互转换

1.3.1 进位计数制

一个任意的十进制数可以表示为:

$a_n a_{n-1} \cdots a_0 . b_1 b_2 \cdots b_m$

其含义是:

$$a_n . 10^n + a_{n-1} . 10^{n-1} + \cdots + a_0 . 10^0 + b_1 . 10^{-1} + b_2 . 10^{-2} + \cdots + b_m . 10^{-m}$$

其中 $a_i(i=0,1,\cdots,n)$,$b_j(j=1,2,\cdots,m)$是 0~9 十个数码中的一个。

十进制数的基数为 10,即其数码的个数为 10,每个数位计满 10 就向高位进位,即"逢十进一"。上式中相应于每位数字的 10^k 称为该位数字的权,所以每位数字乘以其权所得到的乘积之和即为所表示数的值。

例如,一个十进制数 1998.9,可以按"权相加"写成:

$$(1998.9)_{10} = 1\times10^3 + 9\times10^2 + 9\times10^1 + 8\times10^0 + 9\times10^{-1}$$

在计算机科学中，为了便于物理实现，广泛采用二进制计数制。这是因为，二进制表示的数的每一位只取两个数码 0 和 1，因而可以用任何具有两个不同稳定状态的元件来表示，并且数据的存储和传送也可以用简单而可靠的方式进行。二进制的基数是 2，其计数规律是“逢二进一”。二进制数也如十进制数一样可以按“权相加”。

例如，一个二进制数 1101.101 可以写成：

$$(1011.101)_2 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} = (11.625)_{10}$$

采用二进制计数制，对计算机来说，运算、存储和传送极为方便。然而对人来说，二进制书写起来很不方便。为此人们经常采用八进制计数制和十六进制计数制。它们多被用于指令的书写、目的程序的输入或输出。

表 1.1　几种常用的进位计数制的基数和数码

进位计数制	基　数	数　　码
十六进制数	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
十进制数	10	0,1,2,3,4,5,6,7,8,9
八进制数	8	0,1,2,3,4,5,6,7
二进制数	2	0,1

按照同样的方法，读者可以很容易地掌握八进制和十六进制数的表示方法。在计算机里，通常用数字后面跟一个英文字母来表示该数的数制。十六进制数用 H(Hexadecimal)、十进制数用 D(Decimal)、八进制数用 O(Octal)、二进制数用 B(Binary)。

1.3.2　各种数制间的相互转换

1.十进制数转换成二进制数

这里只介绍比较简单的“除 2 取余”法、“乘 2 取整数积”法。

把要转换的十进制数的整数部分不断除以 2，并记下余数，直到商为 0 为止。此为“除 2 取余”法。

例如　M = 117D

117/2 = 58　　($a_0 = 1$)

58/2 = 29　　($a_1 = 0$)

29/2 = 14　　($a_2 = 1$)

14/2 = 7　　($a_3 = 0$)

7/2 = 3　　($a_4 = 1$)

3/2 = 1　　($a_5 = 1$)

1/2 = 0　　($a_6 = 1$)

所以 M = 117D = 1110101B。

对于被转换的十进制数的小数部分则应不断乘以 2，并记下其整数部分，直到结果的小数部分为 0 为止(若始终不能为 0，则考虑一定的精度要求)。此为“乘 2 取整数积”法。

例如　N = 0.8125D

$0.8125\times2 = 1.625$　　($b_1 = 1$)

$0.625 \times 2 = 1.25$ $(b_2 = 1)$

$0.25 \times 2 = 0.5$ $(b_3 = 0)$

$0.5 \times 2 = 1.0$ $(b_4 = 1)$

所以 N = 0.8125 D = 0.1101B。

2.八进制、十六进制数与二进制数的转换

由于数 $2^3 = 8$, $2^4 = 16$,所以一位八进制数所能表示的数值恰好相当于三位二进制数能表示的数值,而一位十六进制数与四位二进制数所能表示的数值正好相当。

例如 67.531O = 110 111.101 011 001B

3AD5.E2H = 0011 1010 1101 0101.1110 0010B

反之,从二进制数转换成八进制数时,只要从小数点开始,分别向左右两边把 3 位二进制数码划为一组,最左和最右一组不足 3 位用 0 补充,然后每组用一个八进制数码代替即可。

二进制数转换成十六进制数与此类似,只不过是四位二进制数码分为一组。

3.二进制、八进制、十六进制数转换成十进制数

均可按"权相加"的方法进行。

例如 $167O = 1 \times 8^2 + 6 \times 8^1 + 7 \times 8^0 = 64 + 48 + 7 = 119D$

$0.68H = 6 \times 16^{-1} + 1 \times 16^{-2} = 0.375 + 0.03125 = 0.40625D$

由于二进制数各位数码的权更容易记忆,所以八进制数、十六进制数转换成十进制数时,往往采用二进制数过渡。请读者思考如何通过二进制数过渡将上述例子转化为十进制数。

1.4 计算机中数的表示及编码

1.4.1 带符号数的表示

在计算机中正数与负数如何表示呢?为便于计算机识别与处理,通常在数的最高位上,用 1 位二进制数位来表示数的符号,0 表示正数,1 表示负数。日常书写用 + 或 - 表示符号的数叫真值;而把一个数连同其符号在内在机器中的表示加以数值化,这样的数叫机器数或机器码。例如:

真值	机器数
+1101	01101
-0.1101	1.1101

其实,小数点在机器硬件中并不存在,只是一种隐含的约定,通常只在书写时标示出来,供阅读使用。例子中的小数点仅代表小数点应该所在的位置。

带符号的机器数可以用原码、反码、补码、移码四种不同码制来表示,由于补码表示法在频繁的加减运算中所具有的优点,现在多数计算机都是采用补码表示法。IBM PC 及其系列机也是采用的补码表示法。为此下面将只对原码和补码分别介绍。

1.原码表示法

原码是一种比较直观的机器数表示法。用二进制数的最高位表示符号,数的有效数值

用二进制数的绝对值表示。若定点小数(小数点位置隐含在符号位之后)的原码形式为 x_0，$x_1, x_2 \cdots x_n$，则原码表示的定义是：

$$[x]_{原} = \begin{cases} x & 1 > x \geqslant 0 \\ 1 - x = 1 + |x| & 0 \geqslant x > -1 \end{cases}$$

式中$[x]_{原}$是机器数，x 是真值。

例如　$x = +0.1001$

则　$[x]_{原} = 0.1001$

又如　$x = -0.1001$

则　$[x]_{原} = 1.1001$

一般情况下，对于正数 $x = +0.x_1x_2\cdots x_n$，则有

$$[x]_{原} = 0.x_1x_2\cdots x_n$$

对于负数 $x = -0.x_1x_2\cdots x_n$，则有

$$[x]_{原} = 1.x_1x_2\cdots x_n$$

若定点整数(小数点位置隐含在数位的最后)的原码形式为 $x_n\, x_{n-1}\cdots x_2x_1x_0$，则原码表示的定义为：

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

通过分析，可得出有关原码的一些性质：

(1)对于真值 0，原码表示中往往有“+0”、“-0”之分，故有两种形式：

$[+0]_{原} = 0.00000\ldots0$

$[-0]_{原} = 1.00000\ldots0$

(2)原码表示的定点小数，其表示范围为 $-1 < x < 1$，即 $|x| < 1$；原码表示的定点整数，其表示范围为 $-2^n < x < 2^n$，即 $|x| < 2^n$。

(3)可用数轴表示出原码的表示范围和可能的数码组合。

以定点整数为例，设机器字长为 n+1 位，如图 1.3 所示，数轴上方表示的是原码的数码组合，下方注明原码对应的真值。

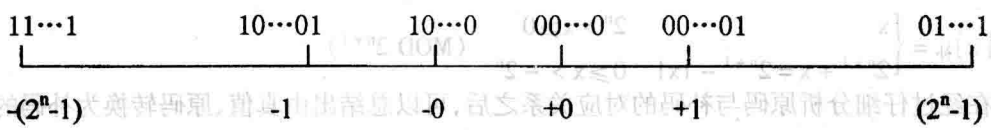


图 1.3　原码表示沿数轴的分布示意图

采用原码表示法简单易懂，但它的最大缺点是加减法运算复杂。为了解决这些问题，人们找到了补码表示法。

2.补码表示法

我们先以钟表对时为例说明补码的概念。假设现在的标准时间为 6 点正，而有一只表走快了，已经 9 点了。为了校准时间，可以采用两种方法：一是将时针逆时针方向(代表减法)退 9 - 6 = 3 格；二是将时针顺时针(代表加法)进 12 - 3 = 9 格。这两种方法都能对准到 6 点，由此看出，减 3 和加 9 是等价的，也就是说 9 是(-3)对 12 的补码，可以用数学公式表示为：

$$-3 \equiv +9 \quad (\text{MOD } 12)$$

MOD 12 的意思就是以 12 为模数，这个“模”表示被丢掉的数值。上式在数学上称为同余式。

上例中之所以 9 - 3 和 9 + 9(MOD 12)等价，原因就是表指针超过 12 时，将 12 自动丢掉，最后得到 18 - 12 = 6。从这里可以得到一个启示，就是在有模运算中，一个负数用其补码代替，将得到同样正确的运算结果。显然，在引入补码后，减法可转换为加法。

在计算机中数的表示及运算受字长限制，其运算都是有模运算。模在机器中是表示不出来的。若运算结果超出所能表示的数值范围，就会自动舍去溢出量，只保留小于模的部分。例如，设机器字长为 n + 1 位：

对于定点小数 $x_0.x_1x_2\cdots x_n$，其溢出量为 10.00...0B，即 2，故定点小数以 2 为模。

对于定点整数 $x_nx_{n-1}\cdots x_2x_1x_0$，其溢出量为 1000...0B，即 2^{n+1}，故定点整数以 2^{n+1} 为模。

一个数 x 的补码记作 $[x]_{补}$，设模为 M，其补码定义如下：

$$[x]_{补} = M + x \quad (\text{MOD } M)$$

上式是一个包含正负数在内的通用定义式。若 $x \geqslant 0$，则模 M 作为溢出量被丢掉，因而正数的补码就是其自身，形式上与原码相同。

例如　$x = 0.1101, M = 2$

则　$[x]_{补} = 2 + x = 2 + 0.1101 = 0.1101 \quad (\text{MOD } 2)$

若 $x < 0$，则

$$[x]_{补} = M + x = M - |x| \quad (\text{MOD } M)$$

例如　$x = -0.1101, M = 2$

则　$[x]_{补} = 2 + (-0.1101) = 2 - 0.1101 = 1.0011 \quad (\text{MOD } 2)$

可见负数的补码形式与其原码形式不同。设机器字长为 n + 1 位，以下分别导出定点小数和定点整数的补码定义。

若定点小数补码形式为 $x_0.x_1x_2\cdots x_n$，则补码表示的定义是

$$[x]_{补} = \begin{cases} x & 1 > x \geqslant 0 \\ 2 + x = 2 - |x| & 0 \geqslant x > -1 \end{cases} \quad (\text{MOD } 2)$$

若定点整数补码形式为 $x_nx_{n-1}\cdots x_2x_1x_0$，则补码表示的定义是

$$[x]_{补} = \begin{cases} x & 2^n > x \geqslant 0 \\ 2^{n+1} + x = 2^{n+1} - |x| & 0 \geqslant x > -2^n \end{cases} \quad (\text{MOD } 2^{n+1})$$

在经过仔细分析原码与补码的对应关系之后，可以总结出由真值、原码转换为补码的实用、快捷的方法：

(1)正数的补码与原码形式相同。

(2)由负数原码表示转换为补码表示时，符号位保持“1”，其余各位变反，并在末位加 1。常将此法叫“变反加 1”。

例如　$[x]_{原} = 10000110$

每位变反 11111001

末位加 1　　1

$[x]_{补} = 11111010$

(3)由负数原码转换为补码的第二种方法是：符号位保持为 1，尾数部分自低位向高位数，第一个 1 及低位的各位不变，更高的各位每位变反。

例如　$[x]_{原}=1\quad 00001\quad 10$

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

采用补码表示的计算机，其运算结果也是补码形式。但补码表示不直观，因此有时需将补码转换为真值或原码表示。其转换方法是：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，尾数每位变反末位加1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。

例如　$[x]_{补}=00000110$

则　$[x]_{原}=00000110$

$x=+110$

又如　$[x]_{补}=11111010$

则　$[x]_{原}=10000110$

$x=-110$

通过分析，可得出补码的一些性质：

(1)对于0，在补码情况下，$[+0]_{补}=[-0]$ 补 $=0.000\ldots0\ (\text{MOD}\ 2)$，即只有一种形式。

(2)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，可直接参加运算。

(3)可用数轴形式表示出补码的表示范围与代码组合。图1.4以定点整数为例，并设机器字长为n+1。

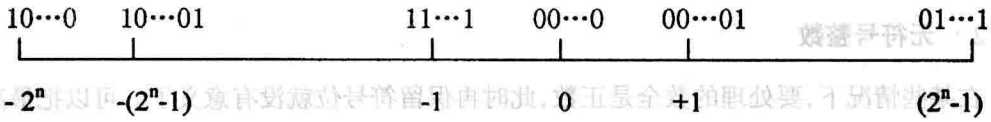


图1.4　补码表示沿数轴的分布示意图

3.补码的加减运算

(1)补码加法

补码加法的规则是：$[x]_{补}+[y]_{补}=[x+y]_{补}\quad(\text{MOD}\ 2)$

上式表示，在模2意义下，任意两数的补码之和等于该两数之和的补码。这是补码加法的理论基础，其结论也适用于定点整数。

例如　$X=+0.1011, Y=-0.0101$，求 $X+Y=?$。

解　$[x]_{补}=0.1011, [Y]_{补}=1.1011$

$$\begin{array}{rr} [x]_{补} & 0.1011 \\ +\quad [Y]_{补} & 1.1011 \\ \hline [x+Y]_{补} & \underline{1}\,0.0110 \end{array}$$

自动舍去1

所以　$X+Y=+0.0110$

(2)补码减法

数用补码表示时,减法运算规则是:

$$[x]_{补}-[y]_{补}=[x]_{补}+[-y]_{补}=[x-y]_{补} \quad (\text{MOD } 2)$$

上式表示,在模2意义下,求两数之差转化为求两数之和。这样可以在计算机系统中只设加法器电路,从而简化了计算机的设计。

从$[y]_{补}$求$[-y]_{补}$的方法是:对$[y]_{补}$包括符号位"每位变反末位加1"。

例如　$X=+0.1101, Y=+0.0110$,求$X-Y=?$。

解　$[x]_{补}=0.1101, [Y]_{补}=0.0110, [-Y]_{补}=1.1010$

$$\begin{array}{rr} [x]_{补} & 0.1101 \\ +\ [-Y]_{补} & 1.1010 \\ \hline [x-Y]_{补} & \underline{1}\ 0.0111 \end{array}$$

自动舍去1

所以　$X-Y=+0.0111$

由上述例子可知补码加减运算的共同特点:

(1)符号位要作为数的一部分一起参加运算。

(2)要在模2的意义下相加,即超过2的进位要丢掉。但机器为了某种需要将把这一进位值保留在标志寄存器的进位位CF中(将在第二章中说明)。

1.4.2 无符号整数

在某些情况下,要处理的数全是正数,此时再保留符号位就没有意义了。可以把最高有效位也作为数值处理,这样的数称为无符号整数。n位无符号数x的取值范围是$0 \leqslant x \leqslant 2^n-1$。例如,8位无符号整数的取值范围是0~255。

在计算机中无符号整数常用来表示地址值。此外,如双精度数(双字长数)的低位字也是无符号整数等。在某些情况下,带符号的数(往往表示成补码形式)与无符号数的处理是有差别的,如无符号数的逻辑移位和带符号数的算术移位问题。以后涉及到时请读者加以注意。

1.4.3 BCD码

虽然二进制数实现容易,并且二进制运算规律简单,但不符合人们的使用习惯,书写阅读都不方便。所以在计算机输入输出场合通常还是采用十进制来表示数,这就需要实现十进制与二进制间的转换。为了转换方便,常采用二进制编码的十进制,简称为BCD码(Binary Coded Decimal)。

BCD码就是用4位二进制数编码表示1位十进制数。表示的方法可有多种,常用的是8421自然BCD码(NBCD码),它的表示规则如表1.2所示。