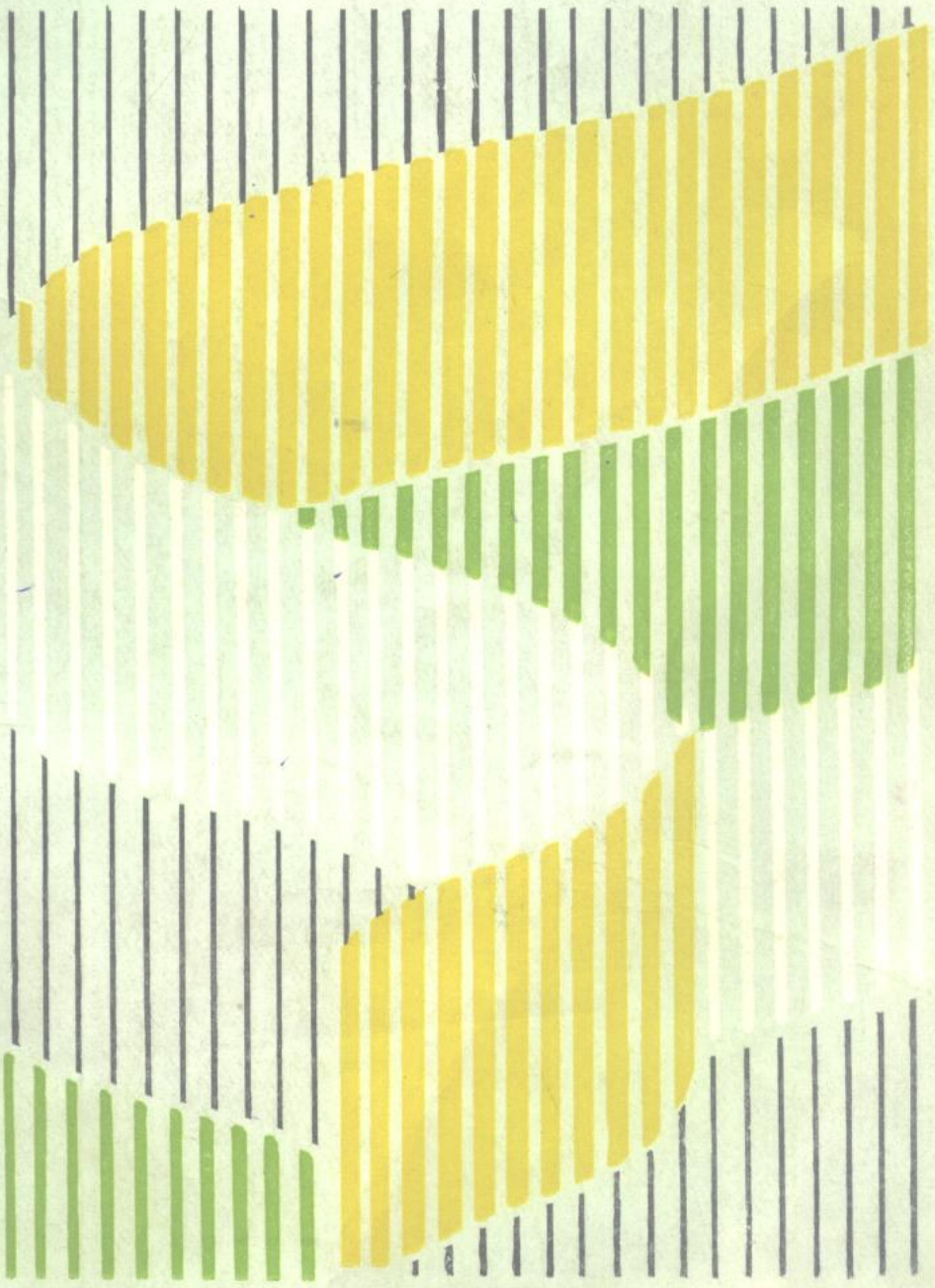
IBM PC

汇编语言

唐家益 主编



陕西师范大学出版社

IBM PC

汇 编 语 言

唐家益 主编

唐家益 冯德民 裘国永 编

陕西师范大学出版社

(陕)新登字 008 号

内 容 提 要

本书以国内的主流机种 IBM PC 系列及其兼容机为背景，系统地介绍了 8088 的指令系统和汇编语言程序设计，取材难度适中，且自成体系。书中配有丰富的例题和习题，适合用作计算机，电子工程，通信及自动控制等专业的教材，也可供工程技术人员，大专院校师生和业余爱好者自学参考。

IBM PC
汇编语言
唐家益等编
陕西师范大学出版社出版
(西安市陕西师大 120 信箱)
陕西省新华书店发行 陕西师范大学印刷厂印刷
开本 787×1092 1/16 印张 14.25 字数 329 千字
1991 年 12 月第 1 版 1991 年 12 月第 1 次印刷
印数: 1—2000
ISBN7-5613-0487-0
G·357 定价: 3.70 元

前　言

随着计算机科学技术的发展，形形式式的高级语言层出不穷，并且各怀绝技，而汇编语言对其生命力却仍充满信心。且不说操作系统，编译原理等系统软件多数是用汇编语言编写的，在其它许多对速度和效率要求较高的场合，如大型软件系统的核心模块，汇编语言也是挺身而出，当仁不让。这是什么原因呢？这是因为汇编语言是迄今为止用户在计算机上能够使用的速度最快，效率最高的语言，也是唯一能够充分发掘计算机硬件资源的语言。随着计算机应用领域的拓宽，特别是微型计算机在管理、控制及通信等方面的广泛应用，汇编语言势将逐步为广大微型计算机用户所掌握和运用。

最近几年，国内外极个别人，制造和传播计算机病毒，破坏宝贵的计算机资源，使广大用户特别是微型机用户蒙受重大损失。因而保护计算机资源，防止和清除计算机病毒，成了广大用户面临的十分紧迫的问题。在同各种病毒的斗争中，汇编语言更是南征北战，大显身手，立下了汗马功劳。可以毫不夸张地说，如果不精通汇编语言，要破译和分析病毒程序，并设计出好的反病毒软件几乎是不可能的。

作为一门课程，《汇编语言》对于训练学生掌握程序设计技术，熟练上机操作及程序调试等技能有着重要的作用，因而汇编语言是高等院校电子计算机硬、软件专业必修的核心课程之一，也是其它课程如微机原理，操作系统等必须的先修课。

本书的前身是在教学工作中为了应急而编写的讲义。在多年的教学实践中，该讲义几经修改，充实和重印，遂成此书。考虑到读者的不同背景，本书取材难度适中，且在内容安排上呈封闭体系。读者只要具有高级语言程序设计的基础，即使没有学习过计算机原理，也可以通过对本书的学习而掌握汇编语言程序设计的基本知识和技能，本书的第一章预备知识就是为这方面的读者安排的。学过计算机原理或者微机原理等课程的读者则可跳过第一章。其余各章的内容如下：第二章　IBM PC 组织结构，第三章　IBM PC 寻址方式和指令系统，第四章　DEBUG，第五章　汇编语言程序设计基础，第六章　分枝程序设计，第七章　宏汇编语言 MASM，第八章　循环程序设计，第九章　中断和磁盘文件管理，第十章　子程序，第十一章　汇编语言程序与高级语言程序的连接。每章都有一定数量的习题，可作为高等学校计算机类专业学生 60 学时的教材，也可供大专院校师生、工程技术人员参考。书中全部例题都在 IBM PC 系列计算机上考验通过，备有软盘，可免费为读者拷贝。

参加本书编写工作的有冯德民，裘国永和唐家益等同志。冯德民同志编写了第二章、第三章、第一章第五节的部分内容及附录三，裘国永同志编写了第四、九和第十一章及附录四，唐家益同志编写了第五、六、七、八、十和第一章的其余部分及附录一、二。全书由唐家益同志负责统稿工作。

本书在编写过程中得到了陕西师范大学计算机科学系、数学系和陕西师范大学教学委

员会的领导和老师的热情鼓励和大力支持．陕西师范大学计算机科学系曹豫莪副教授、陕西师范大学数学系张德荣教授、西北工业大学计算机科学与技术系胡正国副教授、西安冶金建筑工程学院计算机教研室刘家全教授分别审阅了书稿，并提出了许多宝贵意见。陕西师范大学计算机科学系袁凌涛副教授、王四万老师等也给予了很多的帮助；周天阳同志等试用过本书初稿；85级、86级、87级和88级的学生在学习《汇编语言》的过程中也提出过不少建议，陕西师范大学出版社张炜同志为本书的出版做了很多工作，郭建、张玉琴、张川民等同志，参加微机排版和制图，付出了辛勤的劳动，在此一并表示衷心的感谢。

限于编者水平，书中疏漏之处必不在少，恳请读者不吝指正。

编者

1991年6月于陕西师范大学

目　录

第一章 预备知识

本章着重介绍记数制、数的定点和浮点表示、各种进制数的互相转换以及常用的编码方法。

1.1 计算机语言的发展

我们知道，计算机之所以能高效地进行计算或处理，是由于人们事先把用计算机语言编的程序及其所需的数据存储在计算机中，而计算机只不过是按照程序的安排加以执行而已。因此，随着世界上第一台电子计算机的问世，用来编写程序的各种计算机语言也相继出现，并不断地由低级向高级发展。

1.1.1 机器语言

最早的计算机语言就是机器的指令系统，叫做机器语言。由于计算机只能识别 0 和 1,只能执行由 0 和 1 组成的机器指令，所以，在计算机出现之初，程序是直接用机器指令编写的，这样的程序叫做手编程序。

用机器语言编写程序有这样几个问题:

1. 烦琐: 由于机器指令就是一串二进制数，也就是一串 0 和 1 。例如 MV 系列计算机有 16 位指令，也有 32 位指令，也就是说它的指令有的由 16 个 0 和 1 组成，有的则由 32 个 0 和 1 组成，而指令数目有数百条之多，除了熟悉机器指令系统的专业人员以外，其他的人要记住这一串串的 0 和 1 ，并且用以编写程序确实不是一件容易的事情，这就大大的限制了计算机的推广应用。

2. 易错: 由于手编程序中出现的都是 0 和 1 ，稍不小心就会写错，而且错误的检查与修改也是十分困难的。

3. 效率低: 用机器语言编写程序，效率也很低。据统计，即便是熟悉机器指令系统的专业人员，用机器语言编写程序，平均每天只能写大约 9 条正确的指令，无怪乎一个大的系统程序往往要花费几十甚至几百、几千人年。这就是计算机的高速度与程序设计的低效率之间的矛盾。

4. 不通用: 因为各种类型的计算机其指令系统各异，用某一机器语言写的程序，拿到其它型号的机器上就无法使用. 对用户来说这是很不方便的，因为即使他学会了一种机器语言，换一种类型的机器，还得重新学习。

为了克服机器语言的这些缺点，加速计算机的推广应用，出现了一种新的语言——汇编语言。

1.1.2 汇编语言

汇编语言的基本思想，就是用一些助记符来取代由 0 和 1 组成的机器指令编写程序。

比如，用 ADD 代表一条加法指令，用 SUB 代表减法指令等等。显然，要记住这样的符号比记忆那些一串串的 0 和 1 要容易得多，这就给用户带来了一些方便，程序设计的效率提高了。我们把用汇编语言编写的程序叫做汇编源程序。不过计算机认识的还是那些 0 和 1，并不认识这些汇编符号。所以，必须首先把这些汇编符号翻译成二进制的代码，即把汇编源程序翻译成等价的机器语言程序（叫做目标程序），机器才能执行。完成这项任务的是汇编程序，它本身是用机器语言写成的。这样一来，就可以由少数熟悉机器语言的人事先写好汇编程序，而广大用户只需记住一些汇编符号就可以编写程序使用计算机了，这促进了计算机的推广应用。

但是，用汇编语言编写程序仍然有所不便，因为它要求用户熟悉计算机的符号系统，而且也还是容易出错。同时，各种类型的机器的符号系统是不一样的（它与机器指令是一一对应的），因而也缺乏通用性。计算机的高速度与程序设计的低效率之间仍然存在着很大的矛盾。

1.1.3 高级语言

为了进一步发挥计算机的作用，提高程序设计的效率，到了 50 年代中期，出现了许多高级语言，如 FORTRAN，ALGOL，COBOL，后来又出现了 PASCAL，BASIC 等语言。这些高级语言的共同特点是：与数学语言比较接近，容易掌握，程序的检查和修改也很方便，而且用户无需了解计算机的硬件构造，就可以使用它来解决问题。当然用高级语言编的源程序，计算机也不能直接执行，也要像汇编源程序一样，先进行翻译，然后机器才能执行。完成这项翻译工作的是编译程序或解释程序．编译程序与解释程序的差别在于，编译程序是把源程序翻译成机器可直接执行的目标程序，而解释程序是对源程序边解释边执行，下一次再运行还得从头到尾逐步进行解释，不像编译程序那样，只要源程序未加修改，也没有什么错误，编译（或者说翻译）1 次就行了，以后每次运行只要执行生成的目标程序就行了．和汇编程序一样，编译程序或解释程序也是由熟悉机器指令系统的人事先用机器语言写成的．用户的源程序送入计算机后，这种翻译工作会自动进行，这就给用户带来了极大的方便．

1.1.4 高级语言的发展

高级语言也是不断发展的．这种发展一方面是指新的、功能更多、更强的程序设计语言在不断涌现，另一方面是指许多原有的高级语言也在不断扩充和更新，推出新的版本，迎合用户的需要，而不愿意为其它高级语言所代替、接受被淘汰的局面。为了赢得用户的青睐，各种高级语言竞相登台献技，各显身手．这种百花争艳的局面，推动了计算机科学技术的发展。

最初的高级程序设计语言（如 FORTRAN 语言等）都是面向数值计算的，这与当时的计算机主要用于科学计算有关，若使用这些语言去作数据处理，就会显得很不方便，或者力不从心。而随着电子计算机应用的日益广泛，数值计算所占的比重越来越小，而数据处理方面所占的比重则越来越大，于是各种面向商业和数据处理等的高级语言应运而生，COBOL 语言便是其中之一。

上面这些高级语言，虽然处理的对象有所不同，但都属于命令式语言，或者叫做面向

过程的语言。利用这种类型的语言解题时，不必考虑计算机的内部逻辑，而是主要考虑解题、算法的逻辑和过程的描述。FORTRAN、COBOL、ALGOL、PASCAL 等都是面向过程的语言。

50 年代末到 60 年代初，人工智能异军突起，伴随而来的是一种新型的面向函数的程序设计语言脱颖而出，其中最有代表性的是 LISP 语言。函数型语言的出现，打破了过程型程序设计语言的一统天下，并展现了一种新型的程序设计风格。LISP 语言甚至被誉为人工智能的数学，它不仅对人工智能的机器实现有重要意义，而且是进行人工智能理论研究的重要工具。

新的程序设计语言仍在不断涌现，具有代表性的是面向逻辑的程序设计语言 PROLOG。它是在 LISP 语言的基础上发展起来的。PROLOG 语言以逻辑程序设计为基础，以知识处理为主要特色，已广泛应用于问题求解、数理逻辑、自然语言理解、专家系统等领域。

但是，PROLOG 语言没有也不可能将其它高级语言取而代之。继 PROLOG 语言之后，一种面向对象的程序设计方法逐渐引起人们的注意并迅速发展起来。SMALLTALK 就是面向对象的程序设计语言之一。这种语言可使程序自我构造，能实现并行推理，引起了程序设计的概念性变化。许多其它的高级语言纷纷效法，竞相增加面向对象的程序设计功能，推出新的版本。例如 CLOS (Common LISP Object-oriented System) 就是在 Common LISP 的基础上推出来的。

高级语言层出不穷，种类繁多。程序设计的风格从面向过程发展到面向函数、面向逻辑和面向对象，应用范围从科学计算发展到数据处理，又进一步发展到知识处理。高级语言还将不断地向更"高级"发展，支持更先进的程序设计方法，提供更强大的知识处理能力以及更简单友好的用户接口，使软件开发的效率大大提高，成本大大降低。

1.1.5 汇编语言的地位

既然高级语言发展如此迅速，功能更强、能给用户带来更多方便的程序设计语言又几乎是与日俱增，而汇编语言本身又存在着一些固有的缺点，如不通用，程序难写、难读、难改，并且要求程序设计人员对机器的性能有详细和深入的了解，这对于非计算机专业的人们来说，要求是苛刻的。那么，这是不是意味着汇编语言将退出历史舞台，或者说将会从计算机语言中被剔除出去呢？回答是否定的。

诚然，如果用机器语言设计程序不是比用汇编语言难得多，那么汇编语言的存在是不会成为问题的。而且，另一方面高级语言虽然容易学习、使用方便、通用性好，但也有某些不足之处，特别是:

1. 当程序要访问具体硬件时，高级语言则不如汇编语言方便。

2. 在有些情况下，用高级语言编写的程序，经编译后产生的目标程序，比好的程序员用汇编语言编写、经汇编所产生的目标程序，时空开销一般要多 0.5～2 倍。

所以，使用汇编语言编写程序，能比较充分地发挥机器硬件的作用，能高效地使用机器。汇编语言还是编写系统软件的工具。例如操作系统等大型软件的核心模块，常用汇编语言编写。因而学习和掌握汇编语言是重要的。

1.2 记 数 制

任何数都可以用一组统一的符号和规则表示，称为记数制．例如正数 N 可以表示成如下的形式:

$$N=\sum_{i=-k}^{m} A_i \times P^i$$

其中，P 为 2，3，4，...，

A_i 为 0，1，...，$P-1$。

P^i 称为A_i 的权，P称为基数，i为负表示小数部分．例如:

$$125=1\times 10^2+2\times 10^1+5\times 10^0$$

其中，A_0为 5，权为10^0即1，

A_1为 2，权为10^1即10，

A_2为 1，权为10^2即100。

因为 P 为 10，故这种记数制称为十进制，是我们最熟悉的一种记数方法．权值P^i通常可以省略，记作 125．所以只有明确了采用的记数制后，一串符号才有确定的意义，当 P 等于 2 时，称为二进制，P 等于 16 时，称为十六进制，如此等等．同样一串符号例如“100”，在不同的记数制中有不同的意义；在二进制记数系统中，它表示十进制数 4，在十进制记数系统中表示 100，而在十六进制记数系统中表示十进制数 256．为了避免二义性，采用特殊的符号后缀来标识不同的记数制，例如用 B 表示二进制数，用 H 表示十六进制数等等．于是下列符号串便有了确定的意义:

$(101)_B$

$(FF)_H$

在汇编语言中，允许使用二进制、八进制、十进制和十六进制数，因此以下只讨论这 4 种计数制。

1.2.1 二进制

数在机器中是以器件的物理状态来表示的，一个具有两种不同的稳定状态且能互相转换的器件，就可以用来表示 1 位二进制数。例如，用高电位表示“1”，低电位表示“0”，或者反过来，等等．所以，二进制表示起来既简单又可靠。计算机内的数全是用二进制表示的。

二进制数有以下特点:

1．只有两个数码，即 0 和 1．

2．逢二进位。即任意位置上的 2 个单位，形成高一位置（即左边一个位置，下同）上的 1 个单位。

这就是说，同一数码（0 除外）在不同的位置上表示的数值是不同的（因权不一样）。

例如:

111.11

小数点左边第一位的“1”，代表的值就是它本身，因为权是1；而小数点左边第二位的“1”，代表的值为 $1*2^1=2$，因为权为 2^1，等等。至于小数点右边第一位的“1”，表示1/2，因为权为 2^{-1}；小数点右边第二位的权为 2^{-2}，等等。

3. 运算规则简单。进行一位数的加法和乘法时，必须记住这两个数的和与积。因此如果是十进制数运算，人们必须记住55个和与积，若要计算机去记住这么多的加法口诀和乘法口诀，运算器就要做得十分庞大，控制线路也会非常复杂。而采用二进制，因为只有“0”和“1”，加法和乘法口诀都十分简单:

加法:

$0+0=0$

$1+0=1$

$0+1=1$

$1+1=(10)_B$

乘法:

$0\times0=0$

$1\times0=0$

$0\times1=0$

$1\times1=1$

例如，1111与11的和、差、积、商分别为:

```
  1111          1111
+   11        -   11
------        ------
 10010         1100

  1111          101
                ---
x   11       11)1111
------          11
  1111          ---
 1111            11
------           11
101101           --
                  0
```

二进制记数法的缺点是与人们的习惯差距较大，且位数通常较多，书写不便，难读，与十进制之间的转换也不太简便。因此，常用八进制和十六进制来弥补其不足。另外，在程序中可以使用人们习惯的十进制，而在计算时先将其转换成二进制，最后，再把结果转换成十进制输出。而这种烦琐的转换工作，完全由计算机自动完成。

1.2.2 十进制

十进制数具有如下特点:

1. 有10个数码，即0，1，2，3，4，5，6，7，8，9。每个十进制数均由这十个数码组成。

2. 逢十进一，任意位置上的10个单位形成高一位置上的1个单位。所以每个数字不仅本身有大小之差，而且其值还与所处位置有关。

1.2.3 八进制

采用八进制可以弥补二进制的一些不足，使程序易于书写和阅读，当然计算机中还是采用二进制运算，故八进制数一般也要转换成二进制，但这种转换比二进制与十进制之间的转换要容易得多。

八进制数有如下特点:

1. 有 8 个数码，即 0，1，2，3，4，5，6，7。八进制数由这 8 个数码组成。
2. 逢八进一，任意位置上的 8 个单位形成高一位置上的 1 个单位。例如:

$(177)_O$

表示八进制数 177，即十进制数 127 。数字后面的 O (Octal) 或 Q 是八进制数的标记。

1.2.4 十六进制

计算机中一般不用二进制的位 (bit) 作单位，而是以字节 (Byte，即 8 个二进制位) 为计量单位。1 个字节刚好可以表示十六进制数的 2 位，因此十六进制被广泛采用，尤其

表 1.1

十进制	十六进制	八进制	二进制
0	0	0	0 0 0 0
1	1	1	0 0 0 1
2	2	2	0 0 1 0
3	3	3	0 0 1 1
4	4	4	0 1 0 0
5	5	5	0 1 0 1
6	6	6	0 1 1 0
7	7	7	0 1 1 1
8	8	1 0	1 0 0 0
9	9	1 1	1 0 0 1
1 0	A	1 2	1 0 1 0
1 1	B	1 3	1 0 1 1
1 2	C	1 4	1 1 0 0
1 3	D	1 5	1 1 0 1
1 4	E	1 6	1 1 1 0
1 5	F	1 7	1 1 1 1
1 6	1 0	2 0	1 0 0 0 0

在源程序中。

十六进制数有如下特点:

1. 具有 16 个数码，即数字 0～9 及字母 A～F。

2. 逢十六进位。

十六进制数通常末尾标以 H，以示区别。十六进制数与十进制、八进制、二进制数之间的关系如表 1.1 所示:

当然，十六进制数也要转换成二进制形式，才能参加运算，就和十进制数需要转换一样。

1.3 数的定点和浮点表示

1.3.1 定点数和浮点数

任何数 N 都可以表示成如下形式:

$$N = P^j \times S$$

其中 j，P 为正整数，

S 为小数，其形式为:

$$\pm .A_{-1}A_{-2}\cdots A_{-m}$$

这里A_i为 0，1，...，P−1，P 为进位制，j 称为数 N 的阶码，S 称为数 N 的尾数。当 P 为 10 时，即为数.N 的十进制表示。特别，当 $P=2^K$ 时，N 可表示成:

$$N = (2^K)^j \times S$$

这里 K 是一固定整数。当 K＝1，即 N 的二进制表示；K＝3，即 N 的八进制表示；K＝4，即为 N 的十六进制表示；等等。同时，如果对任何数 N，阶码 j 也是固定不变的，则这种表示法称为数的定点表示法，这样的数称为定点数；如果 j 可以取不同值，则称为数的浮点表示，这样的数称为浮点数。当 S 满足

$$\frac{1}{2^K} \text{ 或 } \frac{1}{P} < |S| < 1$$

时，则称数 N 为规格化的浮点数，否则就不是规格化浮点数。

1.3.2 两种表示法的比较

由于在机器中，一般都采用二进制，故只讨论 K＝1 的情形，并设阶码 j 也取二进制形式。设机器中用 M 个二进制位表示 1 个数，则尾数 S 的位数为 M 减去阶码的位数，也就是说当阶码 j 的位数增加（表示的数的范围增大）时，尾数 S 的位数减少（有效数字减少）。浮点数进行加减运算，参与运算的两个操作数首先要“对阶”，然后才能对尾数进行运算；浮点数做乘除运算时，在对尾数进行乘除运算的同时，阶码也要做一次相应的加或减的运算。而定点数的加减乘除运算则要简单得多。另外，当 j＝0 时，定点数只能表示小数；当 j 的位数为 M 时，定点数只能表示整数。这是两种常用的定点数。

1.4 几种进制数之间的转换

如前所述，十进制、八进制或十六进制数在输入时通常要转换成二进制，计算机运算结果，在输出时又要转换成十进制，以符合人们的习惯。当然这种转换都是由计算机自动完成的。那么计算机是怎么进行转换的呢？下面我们就来讨论这个问题。

1.4.1 八进制数与二进制数互相转换

由于八进制数每一位都是由 0 ～ 7 这 8 个数码组成的，因此可以用二进制 3 位表示八进制数的 1 位，因而八进制数与二进制数之间的转换就非常简单，将八进制数转换成二进制数时，只要将八进制数的每一位都用二进制 3 位表示，再按顺序排列起来就行了。若原来的八进制数有 M 位小数，则转换成二进制数后应有 3M 位小数（包括未尾和开始的零）。例如：

```
2 .  5   3  =  0 1 0.1 0 1 0 1 1
↓    ↓   ↓
010  101 011
```

二进制数转换成八进制数的过程与上面的相反。从小数点开始向左向右每 3 位一组，将二进制数分成若干组，若两端的组不足 3 位，则用 0 补齐。然后将每组用对应的八进制码替换，即得所求八进制数。若原来的二进制数有 M 位小数，则结果八进制数中有 $\left\lceil \frac{M+2}{3} \right\rceil$ 位小数。例如：

```
1010.1101 = 001  010  .  110  100  =  (12.64)o
            ↓    ↓       ↓    ↓
            1    2       6    4
```

十六进制数与二进制数互相转换的过程与此类似，只是十六进制数的 1 位对应二进制数的 4 位，因而稍微复杂一点。

1.4.2 十进制数与八进制数互相转换

要将十进制整数 N 转换成如下八进制整数的形式：

$$N = f_m f_{m-1} \cdots f_2 f_0 \qquad (f_i \text{ 为 } 0 \sim 7)$$

那么问题就是求出 f_i（i＝0，1，2，...，m）。方法是除八取余。第一次，将十进制数除以 8，得商 N1，而余数（其值为 0～7 之间）即为所求之 f_0；第二次将 N1（仍为十进制）除以 8，得商 N2 和余数 f_1；如此重复，直到最后一次商为 0 时，相应的余数便是 f_m。

例 1.1 将十进制数 3927 转换成八进制数

```
8 | 3 9 2 7
  8 | 4 9 0      f0 = 7
    8 | 6 1      f1 = 2
      8 | 7      f2 = 5
        0        f3 = 7
```

故$3927=(7527)_O$

将十进制小数$N=0.a_{-1}a_{-2}\cdots a_{-k}$转换成八进制小数$(N)_O=0.f_{-1}f_{-2}\cdots f_{-m}$的形式，即求$f_i(i=-1,-2,\ldots,-m)$。

由于$N=(N)_O$，有$8N=8\times(N)_O$，即

$$a_0.a_{-1}'a_{-2}'\cdots a_{-k}'=f_{-1}.f_{-2}f_{-3}\cdots f_{-m}$$

因为两数相等必有整数部分和分数部分分别相等，即$a_0=f_{-1}$。

这样就求得了f_{-1}，同理可求f_{-2}，f_{-3}，等等．直到乘积的十进制小数部分为零或满足精度要求为止（十进制有限小数转换成八进制不一定仍为有限小数）。

例1.2　将十进制小数0.1875转换成八进制

$$\begin{array}{r} 0.1875 \\ \times\qquad 8 \\ \hline 1.5000 \end{array} \qquad f_{-1}=1$$

$$\begin{array}{r} 0.5 \\ \times\quad 8 \\ \hline 4.0 \end{array} \qquad f_{-2}=4$$

故$0.1875=(0.14)_O$

若十进制数既有整数部分又有小数部分，则按上述规则分别进行转换。

将八进制整数$(N)_O=f_mf_{m-1}\cdots f_2f_1f_0$转换成十进制整数用公式:

$$N=\sum_{i=0}^{m}f_i\times 8^i$$

将八进制小数$(N)_O=0.f_{-1}f_{-2}\cdots f_{-m}$转换成十进制小数的公式是:

$$N=8^{-m}\sum_{i=-1}^{-m}f_i\times 8^{i+m}$$

即将小数点右移M位，按整数方式求和，再将结果除以8^m，即得十进制小数。

例1.3 将$(15.64)_O$转换成十进制小数:

将小数点右移2位，加权求和，最后将和除以64即得。

$$\begin{aligned} &8^{-2}\times(1\times 8^3+5\times 8^2+6\times 8+4)\\ &=8^{-2}\times(512+320+48+4)\\ &=13.8125 \end{aligned}$$

1.4.3　十进制数与二进制数互相转换

十进制数转换成二进制数有两种方法．其一是以八进制数为过渡，进行间接转换，其二是直接从十进制转换成二进制。即对于十进制整数用除二取余法转换成二进制整数；对于十进制小数用乘二取整法转换成二进制数。与将十进制数转换成八进制数的方法完全

类似。

例 1.4 将十进制整数 29 转换成二进制数。

(1) 用“除二取余”法直接转换:

$$
\begin{array}{r|l}
2 & 29 \\
\hline
2 & 14 \quad f_0 = 1 \\
\hline
2 & 7 \quad f_1 = 0 \\
\hline
2 & 3 \quad f_2 = 1 \\
\hline
2 & 1 \quad f_3 = 1 \\
\hline
 & 0 \quad f_4 = 1
\end{array}
$$

(2) 间接转换:

$$29 = (35)_O$$

$$29 = (35)_O = (11101)_B$$

二进制数转换成十进制数也有两种方法。一是以八进制数为过渡，进行间接转换。二是用类似于八进制数转换成十进制数的方法，直接从二进制数转换成十进制数。

例1.5 将二进制数

$$(1101.1101)_B$$

转换成十进制数。

(1) 直接转换:

$$
\begin{aligned}
(1101.1101)_B &= 2^{-4} \times (2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^0) \\
&= 2^{-4} \times (221) \\
&= 13.8125
\end{aligned}
$$

(2) 间接转换:

$$(1101.1101)_B = (15.64)_O$$

利用例1.3的结果得

$$(1101.1101)_B = (15.64)_O = 13.8125$$

1.5 编　　码

如前所述，计算机中普遍采用二进制数，因为二进制数码“0”和“1”容易用电子器件的不同状态来表示。剩下来的问题是，数的符号在机器中怎样表示。

实际上，正号“+”和负号“−”也可以用器件的两种不同的状态来表示。例如，用表示数码“0”的状态表示正号“+”，用表示数码“1”的状态表示负号“−”。于是计算机中数的符号也就“量化”了：“0”代表正号，“1”代表负号.

假设机器中用二进制 8 位表示 1 个数，则第 1 位为符号，其余 7 位表示数值部分。于是二进制数

$$N_1 = (+1010101)_B \qquad N_2 = (-0101010)_B$$

在机器中表示为