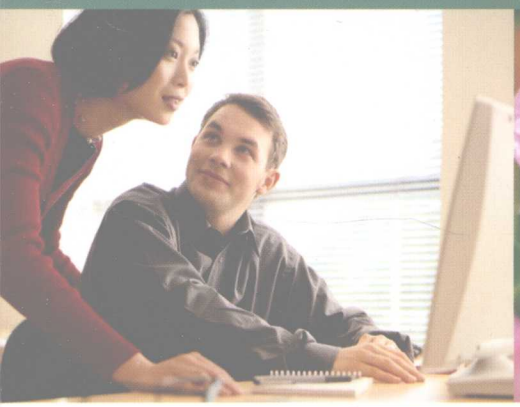
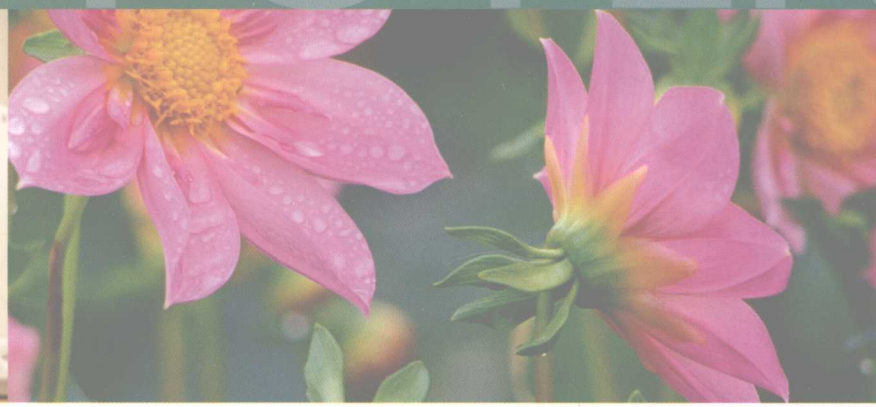
高等学校计算机系列规划教材

COMPUTER





汇编语言程序设计

（第3版）

丁 辉 主编　　张丽虹　魏远旺　副主编

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY http://www.phei.com.cn

高等学校计算机系列规划教材

汇编语言程序设计

（第3版）

丁　辉　主编

张丽虹　魏远旺　副主编

電子工業出版社

Publishing House of Electronics Industry

北京・BEIJING

内 容 简 介

本书内容主要包括微机基础知识， Intel 8086/8088 指令系统，Intel 80x86、Pentium 增强和扩展指令，程序设计方法，高级汇编技术，系统功能调用，汇编语言与 C/C++的混合编程技术，上机操作方法。在程序设计各章中在给出一般例题的基础上，特别设置了综合举例章节；在系统功能调用、汇编语言与 C/C++的混合编程两章中更特地设置了实例章节。每章附有习题，书后附有上机实验指导。

本书可作为高等学校、高等职业学校计算机专业或相近专业汇编语言程序设计课程教材，微型计算机原理课程辅助教材，亦可供软件开发人员参考。

图书在版编目（CIP）数据

汇编语言程序设计/丁辉主编. —3 版. —北京：电子工业出版社，2009.3
高等学校计算机系列规划教材
ISBN 978-7-121-08033-3

Ⅰ. 汇… Ⅱ. 丁… Ⅲ. 汇编语言—程序设计—高等学校—教材 Ⅳ. TP313

中国版本图书馆 CIP 数据核字（2008）第 206398 号

策划编辑：吕 迈
责任编辑：吕 迈
印 刷：北京东光印刷厂
装 订：三河市万和装订厂
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1 092 1/16 印张：18 字数：461 千字
印 次：2009 年 3 月第 1 次印刷
印 数：4 000 册 定价：27.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

在众多程序设计语言中，汇编语言属于低级语言。“低级”主要是指在面向用户方面，汇编语言不及C/C++、Java等程序设计语言。而在面向机器方面，汇编语言之“高级”则无与伦比。汇编语言可以充分利用计算机的硬件特性，编制对时间和空间要求很高的程序，在实时控制场合，汇编语言更是无可替代，由此决定了汇编语言程序设计是计算机专业及相近专业人员的必备知识。

本书以Intel 8086/8088系列微机作为基础机型介绍汇编语言程序设计知识。在介绍8086/8088 CPU寻址方式和指令系统的基础上，详细介绍了汇编语言程序设计的基本方法和技巧，掌握这些内容，可以为Intel 80x86及Pentium系列微机的汇编语言程序设计奠定基础。考虑到Intel 80x86及Pentium系列微机的广泛应用，本书设置了关于Intel 80x86及Pentium的增强和扩展指令内容，在介绍各种程序设计方法的例题中也兼顾了这些指令的应用。本书的主体内容为Intel 8086/8088指令系统和各种程序设计方法，第1章和第2章则提供了学习汇编语言的基础知识，第9章和第10章提供了进行高效率、大规模汇编语言程序设计的必备知识；第11章讲述了用C/C++进行混合编程的基本方法。

本书以编者长期使用的该课程讲稿为主体，以前两版本为基础，进行了系统的整合和内容的扩充，力求难点分散、循序渐进，为避免大量的汇编语言指令集中堆砌，将部分指令融于相关程序设计方法的介绍之中。对于同类内容讲透一点，以点带面。例题和习题的设置力图紧扣重点，举一反三，不仅有一般例题，更有综合举例和应用实例。每项实验均设有验证和设计两种类型的实验题，以便读者在巩固书本知识的基础上，提高应用和创新的能力。

本书由丁辉主编，张丽虹、魏远旺为副主编。第5章至第9章，以及上机实验指导由丁辉编写；第1章至第4章由张丽虹编写；第10、11章由魏远旺编写；全书由丁辉统稿。陈书谦、伍俊明、傅扬烈、姜宏岸、邵峥嵘、冯亚东、常赵罡为本书的编写提出了不少有益的建议，并参与了资料的整理工作。在编写过程中参考了相关书籍，包括书后参考文献中未能列出者，在此对相关作者表示诚挚的谢意。由于编者水平有限，书中难免存在疏漏，敬请同行专家指正。

在本书的编写过程中，得到了电子工业出版社的热情支持，在此一并表示衷心的感谢。

编者

2008年10月

目　录

第 1 章　基础知识 …… 1

1.1　汇编语言与汇编语言程序设计 …… 1

1.1.1　汇编语言 …… 1

1.1.2　汇编语言程序设计 …… 1

1.2　进位计数制 …… 2

1.2.1　常用计数制及其数的算术运算 …… 2

1.2.2　数制转换 …… 4

1.3　计算机中数和字符的表示 …… 8

1.3.1　数的表示 …… 8

1.3.2　字符的表示 …… 11

第 2 章　IBM-PC 计算机系统概述 …… 13

2.1　Intel 8086/8088 CPU 的功能结构 …… 13

2.1.1　执行单元与接口部件单元 …… 14

2.2　Intel 8086/8088 存储器的组织 …… 17

2.2.1　存储单元的地址和内容 …… 17

2.2.2　Intel8086/8088 存储器的组织 …… 17

2.2.3　堆栈 …… 19

2.3　Intel 80x86 系列微处理器简介 …… 20

2.3.1　80386 微处理器 …… 20

2.3.2　Pentium 微处理器 …… 22

2.4　外部设备 …… 25

第 3 章　指令系统 …… 28

3.1　指令格式 …… 28

3.2　寻址方式 …… 28

3.2.1　固定寻址（Inherent Addressing） …… 29

3.2.2　立即寻址（Immediate Addressing） …… 29

3.2.3　寄存器寻址（Register Addressing） …… 29

3.2.4　存储器寻址 …… 29

3.3　指令的执行时间 …… 34

3.4　Intel8086/8088 指令系统 …… 35

3.4.1　数据传送指令 …… 36

3.4.2　算术运算指令 …… 41

3.4.3　位操作指令 …… 45

3.4.4　串操作指令 …… 48

3.4.5 转移指令 …… 48
3.4.6 处理器控制指令 …… 48
3.5 Intel 80x86 及 Pentium 指令系统 …… 48
3.5.1 Intel80386 新增和扩充指令 …… 48
3.5.2 Pentium 新增指令 …… 57
第 4 章 汇编语言与汇编语言程序 …… 63
4.1 汇编语言程序与汇编程序 …… 63
4.2 汇编语言程序的格式和组成元素 …… 63
4.2.1 标识符 …… 64
4.2.2 保留字 …… 64
4.2.3 表达式 …… 64
4.3 伪指令 …… 69
4.3.1 符号定义伪指令 …… 70
4.3.2 变量定义伪指令 …… 70
4.3.3 段定义伪指令 …… 71
4.3.4 过程定义伪指令 …… 73
4.3.5 80x86 指令集选择伪指令 …… 73
4.4 汇编语言程序的上机过程 …… 73
4.4.1 MSAM 汇编环境 …… 74
4.4.2 TASM 汇编环境 …… 84
第 5 章 顺序程序设计 …… 91
5.1 汇编语言程序设计的基本步骤 …… 91
5.2 顺序程序设计 …… 91
5.2.1 十进制算术运算 …… 91
5.2.2 汇编语言程序中的输入/输出功能调用 …… 94
5.3 顺序程序设计综合举例 …… 97
第 6 章 分支程序设计 …… 104
6.1 分支程序结构 …… 104
6.2 转移指令 …… 105
6.2.1 条件转移指令 …… 105
6.2.2 无条件转移指令 …… 108
6.3 分支程序设计 …… 110
6.3.1 测试法分支程序设计 …… 110
6.3.2 跳转表法分支程序设计 …… 114
6.4 分支程序设计综合举例 …… 117
第 7 章 循环程序设计 …… 124
7.1 循环程序结构 …… 124
7.2 循环指令 …… 126
7.2.1 重复控制指令 …… 126

7.2.2 串操作指令及重复前缀 …… 128
7.3 循环程序设计 …… 132
7.3.1 计数控制的循环程序设计 …… 132
7.3.2 条件控制的循环程序设计 …… 134
7.3.3 多重循环程序设计 …… 139
7.4 循环程序设计综合举例 …… 142
第 8 章 子程序设计及系统调用 …… 148
8.1 调用程序与子程序 …… 148
8.2 调用与返回指令 …… 148
8.3 子程序设计 …… 150
8.3.1 子程序的定义 …… 150
8.3.2 子程序的调用与返回 …… 151
8.3.3 保护现场与恢复现场 …… 155
8.3.4 参数的传递 …… 156
8.4 程序的嵌套和递归 …… 163
8.4.1 子程序的嵌套 …… 163
8.4.2 子程序的递归 …… 166
8.5 子程序调用与系统功能调用 …… 167
8.5.1 子程序调用与系统功能调用间的关系 …… 167
8.5.2 系统功能调用的方法 …… 168
8.6 子程序设计综合举例 …… 169
第 9 章 高级汇编语言技术 …… 176
9.1 宏汇编 …… 176
9.1.1 宏定义 …… 176
9.1.2 宏调用和宏扩展 …… 177
9.1.3 宏定义和宏调用中参数的使用 …… 178
9.1.4 宏嵌套 …… 182
9.2 重复汇编 …… 183
9.2.1 使用 REPT 伪指令的重复汇编结构 …… 183
9.2.2 使用 IRP 伪指令的重复汇编结构 …… 184
9.2.3 使用 IRPC 伪指令的重复汇编结构 …… 185
9.3 条件汇编 …… 186
9.3.1 条件汇编的概念及条件汇编的结构 …… 186
9.3.2 条件汇编伪指令 …… 186
9.4 库的使用 …… 190
9.4.1 库的建立 …… 190
9.4.2 库的使用 …… 191
9.5 模块化程序设计 …… 191
9.5.1 模块化程序设计概述 …… 191
9.5.2 段的定义 …… 192

9.5.3 模块间的通信 …… 198
9.5.4 模块的连接 …… 201
第 10 章 系统功能调用及实例 …… 204
10.1 中断 …… 204
10.1.1 中断的基本概念 …… 204
10.1.2 中断的处理过程 …… 206
10.2 系统功能调用方法 …… 208
10.2.1 DOS 功能调用 …… 208
10.2.2 BIOS 功能调用 …… 211
10.3 系统功能调用应用实例 …… 216
第 11 章 汇编语言与 C/C++混合编程及实例 …… 225
11.1 Turbo C 嵌入汇编方式 …… 225
11.1.1 嵌入汇编语句的格式 …… 225
11.1.2 汇编语句访问 C 语言的数据 …… 226
11.1.3 嵌入汇编的编译过程 …… 228
11.1.4 Turbo C 模块连接方式 …… 229
11.2 汇编语言在 Visual C++中的应用 …… 233
11.2.1 嵌入汇编语言指令 …… 234
11.2.2 调用汇编语言过程 …… 236
11.2.3 使用汇编语言优化 C++代码 …… 238
11.2.4 使用 Visual C++开发汇编语言程序 …… 239
11.3 汇编语言与 C/C++的混合编程实例 …… 241
附录 A 上机实验 …… 246
实验一 程序的编辑、汇编、连接和调试 …… 246
实验二 分支程序设计 …… 246
实验三 循环程序设计 …… 247
实验四 子程序 …… 247
实验五 高级汇编语言技术 …… 248
实验六 DOS 功能调用与 BIOS 中断调用 …… 248
实验七 C/C++语言与汇编语言的混合编程 …… 249
附录 B ASCII 码表 …… 250
附录 C 80x86 指令表 …… 251
附录 D MASM 5.0 宏汇编程序出错信息 …… 264
附录 E DEBUG 命令表 …… 269
附录 F BIOS 和 MS-DOS 功能调用 …… 271
参考文献 …… 279

第1章 基础知识

本章提供了学习汇编语言程序设计所需的一些基础知识。首先对汇编语言程序设计进行了概述，其次对计算机常用的几种数制及其相互间的转换方法进行了讲解，并且介绍了数值数据和非数值数据在计算机中的表示方法。

1.1 汇编语言与汇编语言程序设计

1.1.1 汇编语言

计算机程序设计语言是人机交流的重要工具，可分为机器语言、汇编语言和高级语言。

机器语言是机器指令的集合，是一种面向机器的程序设计语言。机器指令是由0和1构成的二进制代码，不同种类的计算机具有各自的机器语言。其优点是可为计算机直接接受，用其编写的机器语言程序执行速度快，占内存空间小，可充分利用计算机的硬件特性；缺点是指令难记，用其编写的机器语言程序难以阅读且通用性差。

高级语言是面向问题求解过程或面向对象的程序设计语言。典型的高级语言有 Pascal，FORTRAN，C++，Java 等。高级语言接近于人类的自然语言，而且通用于各种计算机。其优点是易学易记，用其编写的高级语言程序易读易改，通用性强；其缺点是高级语言程序需经过编译或解释方能被计算机接受，执行速度慢，占内存空间大，不能直接利用计算机的硬件特性。

汇编语言又称为符号语言，实际上是一种符号化的机器语言。它将机器指令的操作码、操作数由二进制代码改为人们所熟悉的符号，例如

```
ADD  AL，5
```

表示将数字5加到AL中。汇编语言程序需经过汇编才能为计算机接受，这一点不如机器语言方便。虽然所用符号为人们所熟悉，然而不如高级语言那样接近人类的自然语言，程序编写和交流较为困难。除此以外，汇编语言几乎具备了机器语言的所有优点，一定程度上弥补了机器语言的缺陷，而且不存在高级语言的上述缺点。可以认为，汇编语言是目前使用的唯一直接利用计算机硬件特性的程序设计语言。

1.1.2 汇编语言程序设计

汇编语言程序设计是指使用汇编语言设计程序的过程。为什么要学习汇编语言程序设计？其原因至少有以下几点。

（1）通过汇编语言程序设计，人们可以高效地使用计算机解决现实问题。在解决同一现实问题时，汇编语言程序与高级语言程序相比，占用内存更小，执行速度更快。

（2）通过汇编语言程序设计，人们可以直接利用计算机的硬件特性，准确计算解决某一问题所需的时间，从而可实现实时控制。这一点是高级语言程序难以替代的。

（3）进行汇编语言程序设计，必然要从原理上认识、理解计算机的工作过程。因此，学

习汇编语言程序设计不仅可以掌握一种高效的程序设计方法，而且对于学习计算机组成原理、计算机原理也大有帮助。

1.2 进位计数制

进位计数制是计数的一种方法。人们普遍习惯的进位计数制为十进制。十进制数的基为10，有10个数码：0、1、2、3、4、5、6、7、8及9，遵循逢十进一的规则。二进制数的基为2，有2个数码：0、1，遵循逢二进一的规则。以此类推，N进制数的基为N，有N个数码：0、1、2、…、N−1，遵循逢N进一的规则。常用的几种进位计数制的情况如表1.1所示。

表1.1 常用的进位计数制

数　制	基	数　码	尾　标
十六进制	16	0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F	H
十进制	10	0，1，2，3，4，5，6，7，8，9	D（或默认）
八进制	8	0，1，2，3，4，5，6，7	Q（或O）
二进制	2	0，1	B

1.2.1 常用计数制及其数的算术运算

1. 十进制（Decimal）

首先观察以下十进制数的组成：

$$361.905D=3\times10^{2}+6\times10^{1}+1\times10^{0}+9\times10^{-1}+0\times10^{-2}+5\times10^{-3}$$

不难看出每一个数码根据它在这个数中所处的位置（数位）来决定实际数值。事实上，任意一个正的十进制数$S=K_{n-1}K_{n-2}\cdots K_0K_{-1}K_{-2}\cdots K_{-m+1}K_{-m}$都可以表示成以下形式：

$$S=K_{n-1}\times10^{n-1}+K_{n-2}\times10^{n-2}+\cdots+K_1\times10^{1}+K_0\times10^{0}+K_{-1}\times10^{-1}+\cdots+K_{-m+1}\times10^{-m+1}+K_{-m}\times10^{-m}$$

$$=\sum_{i=-m}^{n-1}k_i\times10^{i}$$

其中K_i可以是0至9这十个数码中的任意一个，m和n均为自然数，10^i称为权。一般而言，一个正的N进制数$S=K_{n-1}K_{n-2}\cdots K_0K_{-1}K_{-2}\cdots K_{-m+1}K_{-m}$可以表示为以下通用形式：

$$S=\sum_{i=-m}^{n-1}k_i\times N^{i}$$

其中K_i可以是0至N−1中任意一个数字，m和n均为自然数，N^i为各数位相应的权。

2. 二进制（Binary）

在日常生活中，存在着大量的两种对立的现象，例如：是和非，开和关，通和断。电子元件、物理器件中两种状态易于实现，如电压、电流的有和无，晶体管的导通和截止，这两种状态是非常稳定的，而找到8种、10种或16种稳定的状态则要复杂得多，所以计算机中采用二进制作为进位数制，以便于存储及计算的物理实现。

二进制数与人们常用的十进制数的对照关系如表1.2所示。

一个二进制数也可以用上述通用形式来表示。例如：

$$11101.011B=1\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}+1\times2^{-3}$$

二进制数的算术运算与十进制数的算术运算类似，区别仅在于该运算遵循逢二进一的规则。

【例 1.1】（1）1101B+1011B=11000B

$$\begin{array}{r} 1101 \\ +1011 \\ \hline 11000 \end{array}$$

（2）1101B−1011B=0010B

$$\begin{array}{r} 1101 \\ -1011 \\ \hline 0010 \end{array}$$

（3）1101B×1011B=10001111B

$$\begin{array}{r} 1101 \\ \times1011 \\ \hline 1101 \\ 1101 \\ 0000 \\ 1101 \\ \hline 10001111 \end{array}$$

（4）1111B÷101B=11B

$$\begin{array}{r|l} & 11 \\ \hline 101 & 1111 \\ & 101 \\ \hline & 101 \\ & 101 \\ \hline & 0 \end{array}$$

表 1.2　常用进位计数制对照表

十 进 制	二 进 制	八 进 制	十 六 进 制
0	000	0	0
1	001	1	1
2	010	2	2
3	011	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

3. 十六进制（Hexadecimal）

十六进制数采用 0～9，A～F 十六个数码，这里 A 表示 10D，B 表示 11D，C 表示 12D，D 表示 13D，E 表示 14D，F 表示 15D。

十六进制数与人们常用的十进制数的对照关系如表 1.2 所示。

一个十六进制数也可以用上述通用形式来表示。例如

$$6B.0CH=6\times16^1+B\times16^0+0\times16^{-1}+C\times16^{-2}$$

十六进制数的算术运算与十进制数的算术运算类似，区别仅在于该运算遵循逢十六进一的规则。

【例 1.2】（1）8A04H+110CH=9B10H　　（2）8A04H−110CH=78F8H

$$\begin{array}{r} 8A04 \\ +\ 110C \\ \hline 9B10 \end{array} \qquad \begin{array}{r} 8A04 \\ -\ 110C \\ \hline 78F8 \end{array}$$

在汇编语言程序中，数据通常采用十六进制形式，汇编语言的调试、列表文件中显示的数据也都是使用十六进制形式，所以有必要熟练掌握这种数制及其数的运算。

4. 八进制（Octal）

八进制数采用 0～7 八个数码，与十进制数的前八个数码一一对应，如表 1.2 所示。

一个八进制数也可以用上述通用形式来表示。例如

$$36.53Q=3\times8^{1}+6\times8^{0}+5\times8^{-1}+3\times8^{-2}$$

1.2.2 数制转换

1. 非十进制数→十进制数

非十进制数转换为十进制数的方法是：将非十进制数用上述通用形式表示，即按权展开，计算的结果即为十进制数。不管非十进制数是否带有小数部分，均可用这种方法转换。

【例 1.3】$1010110B=1\times2^{6}+0\times2^{5}+1\times2^{4}+0\times2^{3}+1\times2^{2}+1\times2^{1}+0\times2^{0}=86D$

$4D.8H=4\times16^{1}+13\times16^{0}+8\times16^{-1}=77.5D$

$1362Q=1\times8^{3}+3\times8^{2}+6\times8^{1}+2\times8^{0}=754D$

2. 十进制数→非十进制数

十进制数转换为二进制、八进制及十六进制等非十进制数的工作可分为整数的转换和小数的转换两种情况。一个十进制数与其对应的非十进制数相比，两者的整数部分和小数部分分别相等。于是，将一个十进制数转换为非十进制数时，可以对其整数部分和小数部分分别作转换，将两个转换结果结合起来就可以得到对应的非十进制数。

（1）十进制整数→非十进制整数

十进制整数转换为二进制、八进制及十六进制等非十进制整数可以采用除基取余法或减权记位法。前一方法较易掌握，但使用时较烦琐；后一方法使用时较方便，但需要熟悉非十进制各数位对应的权值。

① 除基取余法。将十进制整数及此间产生的商不断除以非十进制数的基，直至商为 0 为止，并记下每一次相除所得到的余数，按照从后往前的次序，将各余数作为 $K_{n-1}K_{n-2}\cdots K_0$，从而构成对应的非十进制数。

【例 1.4】将 233D 转换为十六进制数。

按照题目要求，基应取 16，具体转换过程如下：

	除以基 2	余数	K_i
16	233	9	K_0
16	14	14（即 E）	K_1
	0		

转换结果为：233D=E9H。

【例 1.5】将 233D 转换为二进制数。

按照题目要求，基应取 2，具体转换过程如下：

转换结果为：233D=11101001B。

	除以基 2	余数	K_I
2	233	1	K_0
2	116	0	K_1
2	58	0	K_2
2	29	1	K_3
2	14	0	K_4
2	7	1	K_5
2	3	1	K_6
2	1	1	K_7
	0		

② 减权记位法。减权记位法常用于十进制数到二进制数的转换，对于十进制数到其他进制数的转换使用这种方法则不够方便。这里仅介绍十进制数到二进制数转换时，减权记位法的具体内容。

将十进制整数与其最相近的权值 2^{n-1} 做比较，前者不小于后者则减去 2^{n-1}，并在 $n-1$ 位记 1；否则在 $n-1$ 位记 0。然后再与 2^{n-2} 做比较并做相同的工作，直至最低位被记为 1 或 0。从 $n-1$ 位、$n-2$ 位直至最低位所记的 1 或 0 就构成了二进制数 $K_{n-1}K_{n-2}\cdots K_0$。通常第 1 次比较所用的 2^{n-1} 取做小于等于十进制整数的最大二进位权值。

【例 1.6】将 233D 转换为二进制数。

由于 $2^7<233D<2^8$，故开始用做比较的数值取 2^7。

具体转换过程如下：

比较，减权	记位	K_i
$233-2^7=105$	1	K_7
$105-2^6=41$	1	K_6
$41-2^5=9$	1	K_5
$9<2^4$	0	K_4
$9-2^3=1$	1	K_3
$1<2^2$	0	K_2
$1<2^1$	0	K_1
$1-2^0=0$	1	K_0

转换结果为：233D=11101001B。

【例 1.7】将 130D 转换为二进制数。

考虑到 130D 为 128D 与 2D 之和，即为 2^7 与 2^1 之和，于是对应的二进制数 $K_7K_6\cdots K_0$ 中 K_7=1，K_1=1，其余位均为 0，也即转换结果为 130D=10000010B。

由该例可见，在熟悉二进制各数位对应的权值的基础上，参照减权记位法可以方便地进行十进制整数到二进制整数的转换。

（2）十进制小数→非十进制小数

十进制小数转换为二进制、八进制及十六进制等非十进制小数可以采用乘基取整法或减权记位法。前一方法较易掌握，但使用时较烦琐；后一方法使用时较方便，但需要熟悉非十进制数各数位对应的权值。

① 乘基取整法。将十进制小数以及此间产生的小数部分不断乘以非十进制数的基，并记下每次相乘所得到的整数部分，直至积的小数部分为 0 为止，按照从前往后的次序，将各整数部分作为 K_{-1}，K_{-2}，… K_{-m}，就构成了对应的非十进制数 $K_{-1}K_{-2}\cdots K_{-m}$。

【例 1.8】将 0.6875D 转换为二进制数。

按照题目要求，基应取 2，具体转换过程如下：

$$
\begin{array}{rl}
 & 0.6875 \\
 & \times\quad 2 \\
\hline
K_{-1} \leftarrow & \boxed{1}.3750 \\
 & \times\quad 2 \\
\hline
K_{-2} \leftarrow & \boxed{0}.7500 \\
 & \times\quad 2 \\
\hline
K_{-3} \leftarrow & \boxed{1}.5000 \\
 & \times\quad 2 \\
\hline
K_{-4} \leftarrow & \boxed{1}.0000
\end{array}
$$

转换结果为：0.6875D=0.1011B。

【例 1.9】将 0.6875D 转换为八进制数。

按照题目要求，基应取 8，具体转换过程如下：

$$
\begin{array}{rl}
 & 0.6875 \\
 & \times\quad 8 \\
\hline
K_{-1} \leftarrow & \boxed{5}.5000 \\
 & \times\quad 8 \\
\hline
K_{-2} \leftarrow & \boxed{4}.0000
\end{array}
$$

转换结果为：0.6875D=0.54Q。

当一个十进制小数对应的非十进制小数的位数过多时，可根据需要取前若干位作为近似结果。

② 减权计位法。与上述十进制整数转换为二进制整数时使用的减权计位法类似，但有两点区别：其一，十进制小数首先应与 2^{-1} 比较，然后与 2^{-2} 比较，依次类推；其二，有时要根据需要，取部分小数位作为近似转换结果。

【例 1.10】将 0.6875D 转换为二进制数。

具体转换过程如下：

比较、减权	记　位	K_I
$0.6875-2^{-1}=0.1875$	1	K_{-1}
$0.1875<2^{-2}$	0	K_{-2}
$0.1875-2^{-3}=0.0625$	1	K_{-3}
$0.0625-2^{-4}=0$	1	K_{-4}

转换结果为：0.6875D=0.1011B。

如前所述，将一个十进制数转换为非十进制数时，可以对其整数部分和小数部分分别作转换，然后将整数部分和小数部分的转换结果结合成为最终结果。

【例 1.11】将 233.6875D 转换为二进制数。

由前面的例题可知：

233D=1110 1001B

0.6875D=0.1011B

于是转换结果为：233.6875D=11101001.1011B。

3. 二进制数↔八进制数、十六进制数

由于一位八进制数对应三位二进制数，一位十六进制数对应四位二进制数，于是二进制数与八进制数、十六进制数之间的转换比较简单。

（1）二进制数→八进制数、十六进制数

将二进制数由小数点向左右每三位分为一组（不足三位则用 0 补充），每一组用对应的八进制数码表示，即可得到对应的八进制数。类似地，将二进制数由小数点向左右每四位分为一组（不足四位则用 0 补充），每一组用对应的十六进制数码表示，即可得到对应的十六进制数。

【例 1.12】将 0101 1101.01B 分别转换为八进制数和十六进制数。

01 011 101.01B = 001 011 101 .010B = 135.2Q

0101 1101.01B = 0101 1101.0100B = 5D.4H

说明：在分组中若不足三位或不足四位时，一定要用 0 补充，否则极易出错。就该例而言，若不注意这一点，就极易将结果错写为 135.1Q 和 5D.1H。

（2）八进制数、十六进制数→二进制数

将八进制数的每一位数用三位二进制数码表示，即可得到对应的二进制数。类似地，将十六进制数的每一位数用四位二进制数码表示，即可得到对应的二进制数。必要时可以去掉转换结果中的前 0 和尾 0。

【例 1.13】分别将 45.4Q 和 27CH 转换为二进制数。

45.4Q=100 101 .100B=100101.1B

27CH=0010 0111 1100B=100111 1100B

从以上介绍可以看出，在将十进制数转换为非十进制数时，转换为二进制数较为简单，转换为八进制数、十六进制则较为复杂。考虑到二进制数转换为八进制数、十六进制数比较简单，故在需要将十进制数转换为八进制数、十六进制数时，可以先将其转换为二进制数，然后再由二进制数转换为八进制数、十六进制数。

【例 1.14】将 233.6875D 转换为十六进制数。

首先将 233.6875D 转换为二进制数，由【例 1.11】可知：

233.6875D = 1110 1001.1011B

然后将 11101001.1011B 转换为十六进制数。

1110 1001.1011B = E9.BH

于是十进制数到十六进制数的转换结果为：233.6875D = E9.BH

1.3 计算机中数和字符的表示

人们在使用计算机时，需要计算机不仅能处理数，即数值数据，而且能处理非数值数据，如字符、声音和图像等。虽然非数值数据在计算机中也是以二进制代码的形式存储，但它们的实际意义与数值数据不同。

1.3.1 数的表示

1. 机器数与真值

在本书前面的介绍中，均将数视为无符号数。例如，1010 0010B 的每一位二进制数码均被视为数值，其对应的十进制数为 162。有符号数在计算机中如何表示？首先需要解决数符如何表示的问题。在计算机中实际采用数值化的方法来表示数符，通常用 0 表示正，用 1 表示负。这样表示的数称为机器数，而人们所习惯的用+、−分别表示正、负的数称为真值。机器数的位数与计算机的字长有关，例如

真值	机器数（假设使用 8 位表示）
+10110	0 0010110
−11010	1 0011010

这里机器数采用原码表示。实际上，机器数还可以采用补码、反码表示。

2. 原码

原码的最高位表示真值的数符，其余位为数值位，且与真值的数值位相同，必要时在数值位前加上前 0。

数的原码表示具有直观、与真值的转换方法简单等优点，但是原码有着算术运算复杂的缺陷。与手算相同，做加法运算时，首先要判断两数的数符，同号则相加，异号则相减。做减法运算时，还必须比较两数的绝对值，用较大的绝对值减去较小的绝对值，差的数符则采用绝对值较大者的数符。以上的算法对于硬件实现来说比较困难。

3. 补码

补码的引入，是为了简化减法运算。补码的概念在日常生活中经常用到，例如手表的校时。假定手表停在 11 时，而标准时间为 9 时，可以使用两种校时方法：一种方法是逆时针调 2 小时，即 11−2=9；另一种方法是顺时针调 10 小时，在调至 12 时后则为 0 时，即到 12 时时归零，因此有 11+10=9（称为模加）。于是，就手表校时而言有：

$$11-2 = 11+10 \qquad (\text{MOD } 12)$$

这里 12 称为模，10 称为 2 的补数。由此看出，减去一个数等价于加上该数的补数。下面介绍计算机中使用的补码的含义、求取方法及其运算。

（1）补码定义

$$[X]_{补} = \begin{cases} X & 0 \leqslant X < 2^{n-1} \\ 2^n - |X| & -2^{n-1} \leqslant X < 0 \end{cases} \qquad (\text{MOD } 2^n)$$

当一个数为正数，则其补码就是该数本身；为负数，则其补码等于模值与该数绝对值之差。式中的 n 为补码的位数。

（2）根据真值求补码

根据补码定义可以求取一个数的补码，然而还可以采用更为简便的方法。求取一个数的 n 位补码的简便方法是：对于正数，通过补前 0，将其数值部分补至 n 位即可；对于负数通过补前 0，将其数值部分补至 n 位，然后按位取反并在末位加 1 即可。

【例 1.15】求出以下各数的 8 位补码：

+1000011B，−111000B，+1111111B，−10000000B，0

① 将+1000011B 的数值部分通过补前 0 达到 8 位，即可得到其补码：

$[+1000011B]_{补}$=01000011B

② 将−111000B 的数值部分通过补前 0 达到 8 位，即 00111000B。

按位取反后得到：11000111B

末位加 1 后得到：11001000B

$[-111000B]_{补}$=11001000B

③ 将+1111111B 的数值部分通过补前 0 达到 8 位，即可得到其补码：

$[1111111B]_{补}$=01111111B

④ −10000000B 的数值部分已达 8 位，即 10000000B。

按位取反后得到：01111111B

末位加 1 后得到：10000000B

$[-10000000]_{补}$=10000000B

⑤ 用同样方法不难求得：

$[0]_{补}$=0

（3）根据补码求真值

补码最高位为 0，则真值数符为“+”，真值数值位与补码其余位相同；补码最高位为 1，则真值数符为“−”，将补码所有位按位取反且末位加 1 后就可得到真值的数值位。

用这种方法可以方便的将【例 1.15】中求得的补码还原为对应的真值。

（4）补码的表示范围及补码的扩展

① 补码的表示范围。若要求给出+10000000B 和−10000100B 的 8 位补码，结果如何？显然，将求不出正确的结果。原因在于，这两个数超出了 8 位补码所能表示的范围−128～+127。一般而言，n 位补码所能表示的范围为-2^{n-1}～$+2^{n-1}-1$。在计算机中，n 常取 8，16，32 等。以上两个数可以表示为 16 位补码：

$[+10000000B]_{补}$=0000 0000 1000 0000B

$[-10000100B]_{补}$=1111 1111 0111 1100B

② 补码的扩展。为了满足进行算术运算等方面的需要，有时要求将一个补码扩展成双倍位数，比如由 8 位补码扩展为 16 位补码，由 16 位补码扩展为 32 位补码等。扩展方法是：将扩展的各位都置为原来补码的最高位。

例如：$[+100\ 0011B]_{补}$=0100 0011B=0000 0000 0100 0011B

$[-11\ 1000B]_{补}$=1100 1000B=1111 1111 1100 1000B

（5）补码的加减法运算

补码加法规则：$[X+Y]_{补}=[X]_{补}+[Y]_{补}$