高职高专计算机系列教材

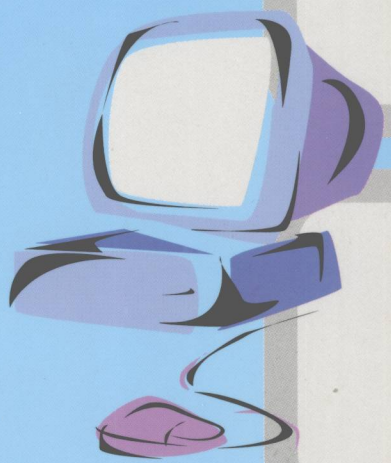
JISUANJI

# 微机原理及应用

Weiji Yuanli ji Yingyong

主　编　张开成

副主编　曹烈斌　胡继宽



重庆大学出版社

# 微机原理及应用

主　编　张开成
副主编　曹烈斌　胡继宽



重庆大学出版社

## 内容简介

本书以80X86微处理器为对象，介绍了微型计算机的基本结构、基本工作原理及其在工程实践中的应用。全书共8章，前2章主要介绍计算机的基础知识和微处理器的结构、特性、功能和发展概况；后面6章则主要介绍8086指令系统及汇编语言程序设计、存储器、输入输出系统、典型微机接口芯片、模拟量接口等内容。为了便于大家学习，每章后面还附有习题和思考题。

本书内容简明扼要、深入浅出，既可作为高职/高专各专业的教学用书，也可作为一般工程技术人员从事技术工作的参考用书。

**图书在版编目(CIP)数据**

微机原理及应用/张开成主编. —重庆：重庆大学出版社，2004.7
(高职高专计算机系列教材)
ISBN 7-5624-3138-8

Ⅰ.微… Ⅱ.张… Ⅲ.微型计算机—高等学校：技术学校—教材 Ⅳ.TP36

中国版本图书馆CIP数据核字(2004)第062134号

**微机原理及应用**

主　编　张开成
副主编　曹烈斌　胡继宽
责任编辑：彭　宁　姚正坤　　版式设计：彭　宁
责任校对：何建云　　责任印制：张立全

*

重庆大学出版社出版发行
出版人：张鸽盛
社址：重庆市沙坪坝正街174号重庆大学(A区)内
邮编：400030
电话：(023) 65102378　65105781
传真：(023) 65103686　65105565
网址：http://www.cqup.com.cn
邮箱：fxk@cqup.com.cn (市场营销部)
全国新华书店经销
重庆升光电力印务有限公司印刷

*

开本：787×1092　1/16　印张：17.75　字数：443千
2004年7月第1版　2004年7月第1次印刷
印数：1—5 000
ISBN 7-5624-3138-8/TP·481　　定价：25.00元

# 前言

随着计算机科学技术的进步和发展，微型计算机的应用已渗透到各个领域。学习和掌握微型计算机的基本工作原理、基本结构及其性能是使用微型计算机的必备基础，也是设计和开发各类微机应用系统的关键。本书正是为适应高职/高专各专业师生及一般科技人员学习微型计算机的需要而编写的。

本书以当前应用极为广泛的 PC 系列微机为背景，从系统角度出发，在讲清原理的基础上，强调实际应用。在内容的安排上，够用为度，难度适中。既注重基础知识的掌握和实用技能的培养，又体现新技术的发展；硬件部分着重说明 MPU 与接口芯片的功能及应用，软件部分强调与硬件结合；内容系统、循序渐进、由浅入深。

本书共分八章，主要内容如下：

第 1 章　讲述微型计算机的发展史、计算机中的数和编码以及微型计算机系统的组成和工作原理。

第 2 章　以 8086CPU 为核心讲述微处理器的编程结构、外部特性和操作时序，同时介绍高档微处理器特点和功能。

第 3 章　全面介绍 8086 指令系统。

第 4 章　讲述汇编语言程序设计方法及相关知识。

第 5 章　主要讲述内存储器的基本结构和功能，同时也简要介绍外存储器的组成和作用。

第 6 章　重点讲述微机输入输出系统中的中断技术、DMA 技术，同时也简要介绍了两个芯片（8259A 和 8237A）的基本结构和功能以及微机系统中常用的几种总线标准。

第 7 章　讲述 8255A 并行接口芯片、8251A 串行接口芯片的结构、功能和应用及 8253A 定时/计数芯片。

第 8 章　讲述 A/D 和 D/A 转换的基本原理及微机与模拟量接口的方法。

本书是一本实用性较强的专业基础课教材，适合于作高职/高专计算机专业教材，或非计算机专业本科教材。

本书由张开成主编。第1章由张开成编写；第2,3,4章由胡继宽编写；第5,6章由曹烈斌编写；第7章由陈美湘编写；第8章由王海云编写；全书由张开成统稿、定稿。

限于编者的水平，且时间仓促，书中难免有错误和不妥之处，恳请读者不吝赐教、指正。

编　者

2004年2月

# 目录


第1章　微型计算机的基础知识 …… 1
1.1　微型计算机的发展史 …… 1
1.2　计算机中的数和编码 …… 2
1.3　微型计算机系统概述 …… 16
习题与思考题 …… 22

第2章　X86系列微处理器 …… 24
2.1　8086微处理器 …… 24
2.2　80286微处理器 …… 37
2.3　80386微处理器 …… 43
2.4　80486微处理器 …… 49
2.5　Pentium系列微处理器 …… 51
2.6　P6系列微处理器 …… 53
2.7　基于NetBurst微体系结构的处理器 …… 55
2.8　IA—64系列微处理器 …… 58
2.9　我国微处理器的发展和龙芯 …… 59
习题与思考题 …… 60

第3章　8086指令系统 …… 61
3.1　指令及指令系统 …… 61
3.2　8086寻址方式 …… 62
3.3　8086指令系统 …… 66
习题与思考题 …… 83

第4章　汇编语言程序设计 …… 85
4.1　汇编语言(Assembly Language)程序 …… 85
4.2　符号指令中的表达式 …… 87
4.3　常用伪指令和宏指令 …… 91

4.4 常用系统功能调用和 BIOS …… 98
4.5 汇编语言程序的上机过程及调试…… 101
4.6 汇编语言程序设计…… 103
习题与思考题 …… 113

**第 5 章 存储器** …… 114
5.1 存储器概述…… 114
5.2 随机访问存储器 RAM …… 118
5.3 只读存储器 ROM …… 125
5.4 存储器与 CPU 的接口 …… 126
5.5 外存储器简介…… 133
5.6 高速缓冲存储器…… 135
习题与思考题 …… 140

**第 6 章 输入输出系统** …… 141
6.1 概述…… 141
6.2 总线技术…… 145
6.3 中断处理技术…… 151
6.4 DMA 技术 …… 165
习题与思考题 …… 180

**第 7 章 典型接口芯片及应用** …… 181
7.1 可编程并行接口 8255A …… 181
7.2 可编程串行通信接口 8251A …… 199
7.3 定时/计数技术 …… 215
习题与思考题 …… 234

**第 8 章 模拟量接口** …… 235
8.1 模拟量接口的基本概念 …… 235
8.2 数字到模拟(D/A)转换器…… 235
8.3 模拟到数字(A/D)转换器…… 245
习题与思考题 …… 255

**附录** …… 256
附录 A 8086 指令表 …… 256
附录 B DOS 系统功能调用表 …… 264
附录 C BIOS 中断功能调用表 …… 270

**参考文献** …… 275

# 第1章 微型计算机的基础知识

## 1.1 微型计算机的发展史

以半导体集成电路为中心的微电子技术的进步，使计算机向着微型化、高性能、低成本的方向迅猛发展。自1946年第一台电子计算机问世以来，计算机已经历了从电子管、晶体管、集成电路到大规模和超大规模集成电路的发展演变。计算机通常按其体积、性能和价格可分为巨型机、大型机、中型机、小型机和微型机五类。由于微型机具有体积小、重量轻、功耗低、价格便宜、结构灵活等特点，从而得到了广泛的普及和应用。

微型计算机的发展史也就是微处理器的发展史，自从20世纪70年代初第一个微处理器诞生以来，微处理器的性能和集成度几乎每两年提高一倍，而价格却降低了一个数量级。

1971年10月，美国Intel公司首先推出4004微处理器，采用P-MOS技术，在4.2 mm × 3.2 mm的硅片上集成了2 250个晶体管，可进行4位二进制的并行处理。这是实现4位并行运算的单片处理器，所有元件都集成在一片MOS大规模集成电路芯片上。以4004微处理器为基础，再配以相应RAM、ROM和I/O接口芯片等，就构成了MCS—4微型计算机。

从20世纪70年代初至今仅三十多年的时间，微处理器的发展经历了以下时期：

第一代为20世纪70年代初，是4位微处理器和低档8位微处理器的时期。典型产品有Intel的4004(4位)、4040(8位)和8008(8位)，其集成度为2 000管/片，采用P-MOS工艺，10 μm光刻技术，时钟频率仅1 MHz，平均执行指令时间约为20 μs。

第二代包括1974—1978年Intel公司推出的8080/8085、Zilog公司推出的Z80、Motorola公司的MC6800/6802等。这一代8位高档的微处理器的特点是采用N-MOS工艺，集成度达8 400管/片以上，时钟频率为2～4 MHz，平均执行指令时间约为1～2 μs。这时的微处理器的设计和生产技术相当成熟，配套的各种器件也很齐备。以后的微处理器在提高集成度，提高功能和速度，增加外围电路的功能和种类上发展很快。

第三代以16位微处理器的出现为代表，时间为1978年。典型产品有Intel的8086、Zilog的Z8000和Motorola的MC68000。它们采用了H-MOS工艺，集成度为20 000～60 000管/片，时钟频率为4～8 MHz，平均执行指令时间为0.5 μs。

第四代微型计算机(1980—1992 年)以 32 位微型机为代表。典型的微处理器产品有 80386/80486,MC68020,Z80000 等。时钟频率为 16 ~ 100 MHz,集成度为 $10^5$ 管/片。

1993 年 Intel 公司推出的 Pentium 微处理器,宣布了第五代微处理器的诞生。这种 Pentium 微处理器芯片采用了全新的体系结构。芯片的集成度高达 $5.0 \times 10^6 \sim 9.3 \times 10^6$ 管/片。时钟频率达 150 ~ 300 MHz。

1995 年 Pentium Ⅱ 问世,1999 年 Pentium Ⅲ 诞生。目前市场上的主流产品已由 Pentium Ⅳ 取代了 Pentium Ⅲ。随着集成电路工艺和计算机科学技术的迅猛发展,更高性能的微处理器将不断推出,微型计算机的发展速度是不可估量的,微型计算机的应用越来越广泛。

## 1.2 计算机中的数和编码

### 1.2.1 计算机中的数制

**(1)常用进位计数制**

1)十进制

在十进制中,每个数位规定使用的数码为 0,1,2,…,9,共 10 个,故其进位基数 R 为 10。其计数规则是“逢十进一”,各位的权值为 $10^i$,i 是各数位的序号。

十进制数用下标“D”表示,也可省略。例如:

$$(256.408)_D = 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 4 \times 10^{-1} + 0 \times 10^{-2} + 8 \times 10^{-3}$$

十进制数人们是最熟悉的,但机器实现起来却很困难。

2)二进制

在二进制中,每个数位规定使用的数码为 0,1,共 2 个数码,故其进位基数 R 为 2。其计数规则是“逢二进一”,各位的权值为 $2^i$,i 是各数位的序号。

二进制数用下标“B”表示。例如:

$$(1101.101)_B = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

二进制数由于每位只需两个状态“0”或“1”,机器实现容易,因而二进制是数字系统惟一认识的代码,但二进制书写太长。

3)八进制

在八进制中,每个数位上规定使用的数码为 0,1,2,…,7,共 8 个,故其进位基数 R 为 8。其计数规则为“逢八进一”。各位的权值为 $8^i$,i 是各数位的序号。

八进制数用下标“O”表示。例如:

$$(375.46)_O = 3 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 + 4 \times 8^{-1} + 6 \times 8^{-2}$$

因为 $2^3 = 8$,因而 3 位二进制数可用一位八进制数表示。

4)十六进制

在十六进制中,每个数位上规定使用的数码符号为 0,1,2,…,9,A,B,C,D,E,F,共 16 个,故其进位基数 R 为 16,其计数规则是“逢十六进一”。各位的权值为 $16^i$,i 是各个数位的序号。

十六进制数用下标“H”表示。例如:

$$(FC2.3D)_H = F \times 16^2 + C \times 16^1 + 2 \times 16^0 + 3 \times 16^{-1} + D \times 16^{-2}$$

$$=15\times16^{2}+12\times16^{1}+2\times16^{0}+3\times16^{-1}+13\times16^{-2}$$

因为 $2^{4}=16$，所以 4 位二进制数可用一位十六进制数表示。

在计算机应用系统中，二进制主要用于机器内部的数据处理；八进制和十六进制主要用于书写程序；十进制主要用于运算最终结果的输出。

**(2)数制转换**

1)非十进制数转换成十进制数

不同数制之间的转换方法有若干种，把非十进制数转换成十进制数采用按权展开相加法。具体步骤是，首先把非十进制数写成按权展开的多项式，然后按十进制数的计数规则求和。

[例 1.1]　$(3B.8)_{H}=(?)_{D}$

解：$(3B.8)_{H}=3\times16^{1}+11\times16^{0}+8\times16^{-1}=(59.5)_{D}$

[例 1.2]　$(135.4)_{O}=(?)_{D}$

解：$(135.4)_{O}=1\times8^{2}+3\times8^{1}+5\times8^{0}+4\times8^{-1}=(93.5)_{D}$

[例 1.3]　$(1011.11)_{B}=(?)_{D}$

解：$(1011.11)_{B}=1\times2^{3}+1\times2^{1}+1\times2^{0}+1\times2^{-1}+1\times2^{-2}=(11.75)_{D}$

2)十进制数转换成二进制数

对于既有整数部分又有小数部分的十进制数转换成二进制数，首先要把整数部分和小数部分分开转换，再把两者的转换结果相加。下面介绍具体的方法。

①整数转换

方法：连续除以 2 取余数，直到商为 0，并按照和运算过程相反的顺序把各个余数排列起来，即为二进制整数。

[例 1.4]　$(13)_{D}=(?)_{B}$

解：

| | | | | |
|---|---|---|---|---|
| 2 | 13 | | 余数 | |
| 2 | 6 | …… | 1 | 最低位 |
| 2 | 3 | …… | 0 | |
| 2 | 1 | …… | 1 | |
| | 0 | …… | 1 | 最高位 |

故　　$(13)_{D}=(1101)_{B}$

②小数转换

方法：连续乘 2 取整数，并按照和运算过程相同的顺序把各个整数排列起来，即为二进制小数。

[例 1.5]　$(0.625)_{D}=(?)_{B}$

解：　　$0.625\times2=1.25\cdots\cdots1$　　最高位

$0.25\times2=0.5\cdots\cdots0$

$0.5\times2=1.0\cdots\cdots1$　　最低位

故　　$(0.625)_{D}=(0.101)_{B}$

[例 1.6]　$(13.625)_{D}=(?)_{B}$

解：$(13.625)_{D}=(1101.101)_{B}$

3)二进制数转换成八进制数或十六进制数

二进制数转换成八进制数(或十六进制数)时，其整数部分和小数部分可以同时进行转

换。其方法是:以二进制数的小数点为起点,分别向左、向右,每3位(或4位)分一组。对于小数部分,最低位一组不足3位(或4位)时,必须在有效位右边补0,使其足位。然后把每一组二进制数转换成八进制(或十六进制)数,并保持原顺序。对于整数部分,最高位一组不足位时,可在有效位的左边补0,也可以不补。

[例1.7] $(1010101100.10011)_B=(?)_O=(?)_H$

解:
$$\begin{array}{l} 1010101100.100110 \\ \llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner \\ 1\ \ 2\ \ 5\ \ 4\ .\ 4\ \ 6 \end{array}$$

故 $(1010101100.10011)_B=(1254.46)_O$

$$\begin{array}{l} 1010101100.10011000 \\ \llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner \\ 2\ \ A\ \ C\ .\ 9\ \ 8 \end{array}$$

故 $(1010101100.10011)_B=(2AC.98)_H$

4)八进制数或十六进制数转换成二进制数

八进制(或十六进制)数转换成二进制数时,只要把八进制(或十六进制)数的每一位数码分别转换成3位(或4位)的二进制数,并保持原排序即可。整数最高位一组左边的0,及小数最低位一组右边的0,可以省略。

[例1.8] $(27.34)_O=(?)_B$

解: $(27.34)_O=($ $\begin{array}{l} 010111.011100 \\ \llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner \\ 2\ \ 7\ .\ 3\ \ 4 \end{array}$ $)_B=(10111.0111)_B$

[例1.9] $(3AC.78)_H=(?)_B$

解: $(3AC.78)_H=($ $\begin{array}{l} 001110101100.01111000 \\ \llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner\llcorner\!\lrcorner \\ 3\ \ A\ \ C\ .\ 7\ \ 8 \end{array}$ $)_B$

$=(1110101100.01111)_B$

### 1.2.2 符号数的表示法

**(1)机器数与真值**

二进制数与十进制数一样有正有负。在计算机中,常把数的符号和数值部分一起编码后表示带符号数。常用的带符号数编码方法有原码、反码和补码表示法。这几种表示法都将数的符号数码化。通常正号用"0"表示,负号用"1"表示。为了区分一般书写时表示的数和机器中编码表示的数,称前者为真值,后者为机器数,即数值连同符号数码"0"或"1"一起作为一个数称为机器数,而它的数值连同符号"+"或"-"称为机器数的真值。

为了表示方便,常把8位二进制数称为字节,把16位二进制数称为字,把32位二进制数称为双字。对于机器数应将其用字节、字或双字表示,因此只有8位、16位或32位机器数的最高位才是符号位。

1)原码

如上所述,数值部分用该数的绝对值;符号部分,用"0"表示正数,用"1"表示负数。这样表示数的方法称带符号数的原码表示法,所表示的数称原码。如:

$y_1=+127=+1111111B$ $\quad [y_1]_{原码}=01111111B$

$$y_2 = -127 = -1111111B \qquad [y_2]_{原码} = 11111111B$$

其中最高位为符号，后面 7 位是数值。用原码表示时，+127 和 -127 的数值部分相同而符号位相反。

8 位整数原码表示的数值范围为 FFH ~ 7FH，即 -127 ~ +127。原码数 00H 和 80H 的数值部分相同，符号位相反，它们分别为 +0 和 -0；16 位整数原码表示的数值范围为 FFFFH ~ 7FFFH，即 -32767 ~ +32767。原码数 0000H 和 8000H 的数值部分相同，符号位相反，它们分别为 +0 和 -0。

原码表示简单易懂，而且与真值的转换方便。但若是两个异号数相加，或两个同号数相减，就要做减法。为简化计算机的结构，需把减法运算转换为加法运算，因而引入了反码和补码。

2）反码

正数的反码与原码相同；负数的反码为它的原码的数值部分按位取反，而符号位不变，即仍为“1”。如：

$$y_1 = +127 = +1111111B \qquad [y_1]_{反码} = 01111111B$$
$$y_2 = -127 = -1111111B \qquad [y_2]_{反码} = 10000000B$$

3）补码

正数的补码与原码相同；负数的补码为它的原码的数值部分按位取反，最末位再加 1，而符号位不变，即仍为“1”。如：

$$y_1 = +127 = +1111111B \qquad [y_1]_{补码} = 01111111B$$
$$y_2 = -127 = -1111111B \qquad [y_2]_{补码} = 10000001B$$

把一个数连同符号位按位取反，最末位加 1，可以得到该数的补数。求补数还可以直接求，方法是从最低位向最高位扫描，保留直至第一个“1”的所有位，以后各位按位取反。负数的补码可以由其正数求补得到。根据两数互为补数的原理，对补码表示的负数求补就可以得到其正数，即可求得该负数的绝对值。如：

$$[-127]_{补} = 10000001B = 81H$$

对其求补，从右向左扫描，第一位就是 1，故只保留该位，对其左面的 7 位均求反得：01111111，即补码表示的机器数 81H 的绝对值是 127。

由此可以知道，一个用补码表示的机器数，若最高位为 0，则其余位表示为此数的绝对值；若最高位为 1，则此数其余位表示的不是此数的绝对值，若把该数求补，才得到它的绝对值。

8 位补码数表示的数值范围为 80H ~ 7FH，即 -128 ~ +127；16 位补码表示的数值范围为 8000H ~ 7FFFH，即 -32768 ~ +32767。

一个二进制补码数的符号位向左扩展若干位后，所得到的补码数的真值不变。对于用补码表示的数，正数的扩展应在其前面补 0，而负数的扩展，则应在其前面补 1，相当于将其符号位向左扩展。例如：48 用 8 位补码表示为 30H，用 16 位表示为 0030H；-48 用 8 位补码表示为 D0H，用 16 位表示为 FFD0H。

当数采用补码表示时，就可以把减法转换为加法。例如：

$$48 - 8 = 48 + (-8)$$
$$[48]_{补} = 30H = 00110000B$$
$$[8]_{补} = 08H = 00001000B$$

$$[-8]_{补} = F8H = 11111000B$$

做减法运算过程如下：

$$\begin{array}{r} 00110000 \\ -\ 00001000 \\ \hline 00101000\cdots\cdots 28H \end{array}$$

做补码加法运算过程如下：

$$\begin{array}{r} 00110000 \\ +\ 11111000 \\ \hline 100101000\cdots\cdots 28H \end{array}$$

↑ 进位自然丢失

两种方法运算结果相同，其真值为28H(32+8=40)。这就说明在微型计算机中，带符号数的减法运算转换为加法运算的正确性。

**(2)原码、反码、补码和真值的相互转换关系**

前面我们已经讨论了符号数的真值表示和在机器中的表示，下面我们总结一下机器数和真值的相互转换关系。

正数的原码、反码、补码相同，即正号以“0”加上绝对值的数值位表示。

负数的原码、反码、补码和真值的相互转换关系，如图1.1所示。

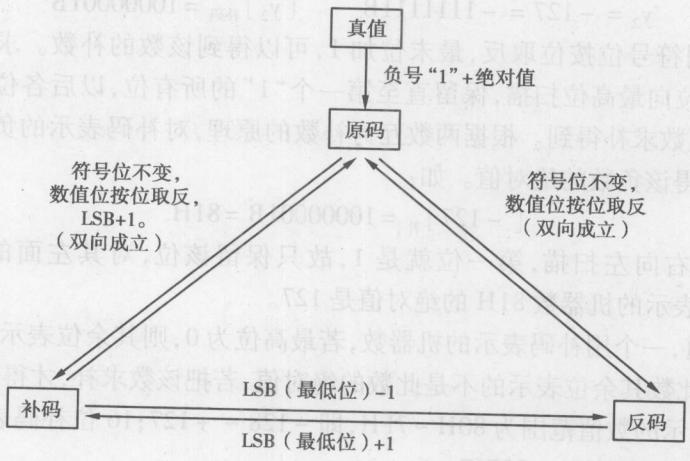



图1.1　负数的原码、反码、补码和真值的相互转换关系

下面我们举例说明符号数的机器表示和真值的相互转换关系。以负数情况为例进行分析。

1)已知原码，求补码

[例1.10]　已知某数X的原码为10110100B，试求X的补码。

解：　由$[X]_{原}$=10110100B，X为负数。求其补码表示时，符号位不变，数值部分按位取反，再在末位加1。

```
    1  0  1  1  0  1  0  0     原码
    ↓  ↓  ↓  ↓  ↓  ↓  ↓  ↓
    1  1  0  0  1  0  1  1     符号位不变,数值位取反
+                   1         +1
-------------------------------
    1  1  0  0  1  1  0  0     补码
```

故　　$[X]_{补}=11001100B$

2)已知补码,求原码

[例 1.11]　已知某数 X 的补码为 11101110B,试求其原码。

解：由$[X]_{补}=11101110B$,知 X 为负数。求其原码表示时,符号位不变,数值部分按位求反后,再在末位加 1。

```
    1  1  1  0  1  1  1  0     补码
    ↓  ↓  ↓  ↓  ↓  ↓  ↓  ↓
    1  0  0  1  0  0  0  1     符号位不变,数值位取反
+                   1         +1
-------------------------------
    1  0  0  1  0  0  1  0     原码
```

故　　$[X]_{原码}=10010010B$

3)求补

已知$[X]_{补}$,求$[-X]_{补}$的过程称为求补。所谓求补,就是将$[X]_{补}$的所有位(包括符号位)一起逐位取反,然后在末位加 1,即可得到 -X 的补码,亦即$[-X]_{补}$。不管 X 是正数还是负数,都应按此方法操作。

[例 1.12]　试求 +97, -97 的补码。

解：+97 = 1100001,于是$[+97]_{补}=01100001B$

求$[-97]_{补}$的方法是:

```
[97]补 = 0  1  1  0  0  0  0  1     [+97]补
         ↓  ↓  ↓  ↓  ↓  ↓  ↓  ↓
         1  0  0  1  1  1  1  0     逐位取反
+                      1           +1
----------------------------------
         1  0  0  1  1  1  1  1     [-97]补
```

故　　$[+97]_{补}=01100001B$,$[-97]_{补}=10011111B$

4)已知补码,求对应的十进制数。

[例 1.13]　已知某数 X 的补码为 10101011B,试求其对应的十进制数。

解：由$[X]_{补}=10101011B$,知 X 为负数,故符号位不变,数值部分按“求反加 1”法得到$[X]_{原}$:

$$[X]_{原}=11010101B,X=-1010101B,X=-85$$

**(3)定点数和浮点数**

依照小数点的不同表示方法,计算机中的数可分为两类:定点数和浮点数。下面说明这两种数的表示方法。

1)定点数

所谓定点数是指小数点的位置固定不变的数,定点数又分为定点整数和定点小数。

- 定点整数:其小数点的位置固定在数据位的最低位之后,见图 1.2(a)。
- 定点小数:其小数点的位置固定于符号位与数值位之间,见图 1.2(b)。

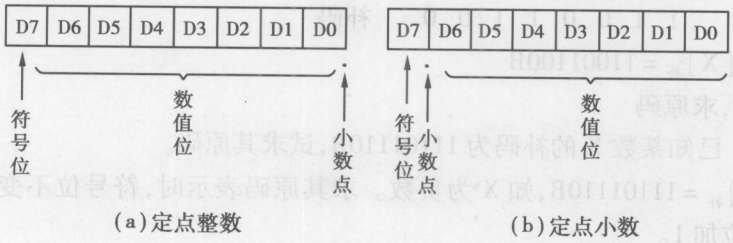



图 1.2 定点数的格式

2)浮点数

与定点数相反,若小数点的位置不固定,是浮动可变的,称这类数为浮点数。

浮点数的引入克服了定点数所能表示的数的范围太小这一缺点。

浮点数一般用 $\pm K \times a^b$ 的形式来表示,其中:K 称为尾数,一般取纯小数,即 $0 \leqslant K < 1$;b 是指数,又称阶码,它是一个整数;a 称为浮点数的基,通常取 a=2。

计算机中的浮点数由尾数和阶码两部分组成:尾数是带符号的定点纯小数,它的符号称为数符,表示这个浮点数的正负;阶码是一个带符号的整数,其符号称为阶符。不难看出,阶码实际上是尾数中的小数点向左或向右移动的位数。

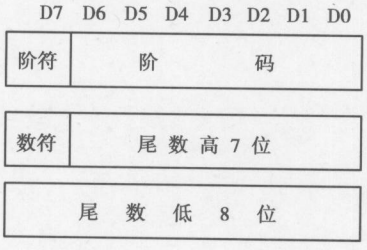



图 1.3 三字节浮点数

图 1.3 给出了一种浮点数的格式。

需要说明的是,为了提高运算精度,应尽量增加尾数中有效值的位数。由于一个数的有效数值位从该数左边第一个非零数值位开始,因此尾数的最高数值位不等于零时,其有效数值位最多,这种称为规格化浮点数。如下面的浮点数:

$$y_1 = 2^{101} \times 0.110101$$

$$y_2 = 2^{011} \times 0.101101$$

而尾数的第一位不是 1 的浮点数,就是非规格化的数。如:$y_3 = 2^{111} \times 0.01011$。

要使浮点数规格化,只要移动小数点并同时调整阶码即可。具体做法是:尾数小数点右移一位,同时将阶码减 1,直到尾数的第一位是 1 为止。例如:

$$y_4 = 2^{100} \times 0.00101 = 2^{010} \times 0.10100$$

[例 1.14] 将十进制数 24.09375 化为二进制形式的规格化浮点数,用图 1.3 的格式表示。

解:

①先将该数转换为二进制数:

$$24.09375D = 11000.00011B(\times 2^0)$$

②将此数规格化,使它的尾数变成最高位为 1 的纯小数。不难看出,要做到这一点,须将尾数中的小数点向左移动 5 位,每左移一位,阶码加 1,移位的次数便是该浮点数阶码的大小。

于是，所得规格化浮点数的尾数和阶码分别为：

k = +0.1100000011B　　b = +5 = +101B

③将该数表示为如图 1.4 所示的浮点数的格式。

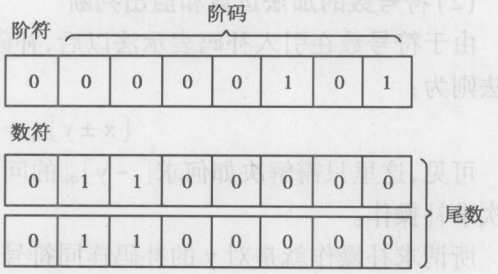



图 1.4　例题 1.4 的浮点数格式

### 1.2.3　二进制数的加减运算

计算机把机器数均当作无符号数进行运算，即符号位也参与运算。运算结果的符号要根据运算有无进位（借位）和溢出等来判别。计算机中设置有这些标志位，标志位的值由运算结果自动设定。

**（1）无符号数的运算**

无符号数实际上是指参加运算的数均为正数，且全部数位都用于表示数值。n 位无符号二进制数能表示的数值范围为 0 ~ $(2^n - 1)$

1）两个无符号数相加，由于两个加数均为正数，因此其和也是正数。当和超过其位数所允许表示的数值范围时，就向更高位进位。如：

127 + 150 = 7FH + 96H

```
  01111111
+ 10010110
----------
1 00010101 = 115H = 256 + 16 + 5 = 227
↑
└── 进位
```

2）两个无符号数相减时，被减数大于或等于减数，无借位，结果为正；被减数小于减数，有借位，结果为负。如：

128 − 10 = 80H − 0AH

```
  10000000
- 00001010
----------
  01110110 = 76H = 112 + 6 = 118
```

反过来相减，即 10 − 128，运算过程如下：

```
  00001010
- 10000000
----------
1 10001010 = −01110110 = −118
↑
└── 借位
```

由此可见，对无符号数进行减法运算，其结果的符号用借位标志位来判别：若 CF = 0，则无借位，结果一定为正；若 CF = 1，则有借位，结果一定为负。对 8 位数值位求补便可求得它的绝对值。

(2)符号数的加法运算和溢出判断

由于符号数在引入补码表示法以后,补码的减法运算可以转换为加法运算来进行。其运算法则为:

$$[x \pm y]_{补} = [x]_{补} + [\pm y]_{补}$$

可见,这里只需解决如何求$[-y]_{补}$的问题,而这个问题的实质就是对 y 的补码再次进行一次求补操作。

所谓求补操作就是对 y 的补码连同符号位再次求补,即

$$[-y]_{补} = [[y]_{补}]求补$$

如上所述,我们只需讨论补码的加法运算和溢出判断问题。两个补码加数相加时,在什么情况下才有可能发生溢出呢?很显然,只有当它们都是正数或当它们都是负数补码相加时才有可能发生溢出。例如,两个带符号数$(01000001)_B$(+65)和$(01000011)_B$(+67)相加,即

$$\begin{array}{r} 01000001 \\ +\ 01000011 \\ \hline 10000100 \end{array}$$

它是两个正数相加,但相加结果却是一个负数,显然这个结果是错误的,出现这种情况的原因就在于这两个加数相加结果超过了 8 位二进制带符号数补码所能表示的范围(-128 ~ +127)。

现在,再来看两个负数$(10001000)_B$(-120)和$(11101110)_B$(-18)的相加情况。

$$\begin{array}{r} 10001000 \\ +\ 11101110 \\ \hline \boxed{1}\,01110110 \end{array}$$

由于规定用 8 位二进制数来表示带符号数,故忽略作为进位位的第 9 位。按 8 位二进制数来解释这两个符号数的相加,其结果却为一个正数,很明显,结果是错误的。错误的原因仍如上所述。

以上两种情况叫做补码运算的溢出。当两个同符号的数据相加时,如果相加的结果超过了微处理器所能表示的数值范围,就将发生溢出,其结果就是错误的。因此,在微处理机中设有专门的电路用以判断运算结果是否产生溢出,并以某种标志告诉人们这次运算的结果是否存在溢出。只要溢出没有发生,运算的结果总是正确的。这个标志在 8086 微处理机中是用 OF 来标志的。若运算结果发生溢出,则置 OF 为 1;否则 OF 为 0。

在机器中如何判别溢出呢?又如何置 OF 标志位呢?下面就来讨论这个问题。

设符号位向进位位的进位为 $C_Y$,数值部分向符号位的进位为 $C_S$,则有无溢出的判别式如下:

$$OF = C_Y \oplus C_S$$

式中,OF = 1 表示有溢出;OF = 0 表示无溢出。

现以 105 + 50, -105 - 50 和 -50 - 5 为例,来判别其运算结果有无溢出:

$$\begin{array}{r} 01101001 \\ +\ 00110010 \\ \hline 10011011 \end{array}$$