计算机等级(水平)考试系列教材

微型计算机原理及应用

孙家启　朱武　周鸣争　编著

安徽大学出版社

高 级 顾 问

韦 穗　杨善林　陈国良

编 委 会 名 单

主　任：孙家启

委　员：（按姓氏笔划）王忠仁　石　竹　孙家启
仲　红　朱　武　朱学勤
吴国凤　张国平　周鸣争
姚合生　聂会星　谢荣传

秘书长：聂会星

编 写 说 明

为了支持计算机基础教育改革与建设，促进计算机基础课程教学与水平考试向纵深发展，我们组织编写了计算机等级（水平）考试系列教材。这套教材囊括了计算机入门知识、高级语言程序设计、软件技术基础、微型计算机原理及应用等方面内容，涵盖计算机水平考试的一、二、四级（全国等级考试的一、二、三级），因而具有广泛的适应性。这套教材所具有的突出特点是：紧扣计算机基础教育大纲（即计算机水平考试大纲），兼具普通教材与考试辅导材料的双重功能；立意创新，内容简炼，大量针对性极强的习题和典型例题分析是其它教材所少见；编写人员都是教学、科研第一线有着丰富教学与实践经验的教师，他们深谙相关知识点的张弛取舍。我们还聘请了三位知名专家担任高级顾问，这诚然为本系列教材添色增辉。

本系列教材的先期版本现已问世，第一辑各册预计在 1998 年底全部出齐。由于计算机技术的发展比人们想象的还要快，所以我们今后还将不断调整教材内容使之与当时发展相适应，以便教材以更新更好的面目呈现在读者面前。

本系列教材编写目的明确，它特别适合于作为普通高校非计算机专业的本、专科教学用教材或成教、夜大、函大计算机专业的教材，也可供各地计算机水平考试考点使用，还可供广大计算机自学者、工程技术人员参考。

编写委员会

一九九七年十月

前　言

“微型计算机原理及应用”是理工类大学生、计算机应用研究人员在解决了计算机入门问题之后，继续向深层次发展而必须研修的一门重要技术基础课。为“知其然并知其所以然”，就得对“原理”潜心研究一番。本书也就是为此而奉献给读者的。

本书以高等学校非计算机专业本科和计算机专业专科学生为教学对象，系统介绍了微型计算机的结构原理、接口方法及应用技术。兼顾全国高等学校计算机考试的需要，全书内容覆盖了四级偏硬(三级 A)考试范围的大部，所以它又是一本考试指导用书——计算机等级(水平)考试系列教材之一。

本书的编写原则是：理论与实践并重，传统与发展兼顾，教学与考试相容。它具有这样一些特点：选材上力求广泛、实用、新颖；叙述上力求简洁、准确、易懂；非计算机专业、少学时决定了小而全、少而精，但全而不滥、少而不乏；以传统的 16 位微处理器 Intel 8086/8088 为主体展开全面讨论，同时又对发展过程中的 80286～Pentium 及其组成系统作了简单介绍；书中特意安排的一些典型例题既是成功经验的总结，可供实际应用参考，又是各种考试中频繁涉及的内容，对备考极具参考价值。

这是一门实践性很强的课程，除课堂教学外，还应辅之以一定量的实验。作为教材，建议课内讲授 50～60 学时，实验 30 学时。

本书共分十章。第一章至第三章及附录由孙家启编写，第四章和第五章由周鸣争编写，第六章至第十章由朱武编写。编写过程中，得到了本系列教材高级顾问们的悉心指导，参阅了国内外有关著作，安徽大学出版社为本书出版付出了极大努力，编者在此一并表示衷心的感谢。

由于水平有限，加之时间仓促，错误与疏漏之处难免，恳请读者批评指正。

编者

一九九七年十二月

目　次

第一章　计算机概论

现在，人们提到的计算机都是指电子数字计算机。由于计算机应用的广泛性，其高超的计算能力是家喻户晓人人皆知的。然而，从科学的角度来看，它仍然只是一种计算和信息加工的机器。由于它的存储容量大，运算速度快和计算精度高等优点，使它远远胜过已有的其他计算工具。当然，作为计算和信息加工的工具，计算机与人类发明的其它计算工具相比，有了质的飞跃。它减轻并部分代替了人的脑力劳动，在特定时间内，能够完成人脑无法完成的工作量，也因此获得电脑的美称。

1.1　电子计算机的基本组成和工作过程

1.1.1　电子计算机的基本组成

1.1.1.1　电子计算机算题和人工算题的相似性

电子数字计算机是作为一种计算工具出现的，其算题过程和人用算盘算题具有很大的相似性。

现以计算 $21\times12-117\div3=?$ 为例。首先人用算盘作为运算工具。其次要有纸，用它来记录和存放原始数据、中间结果和运算结果。这些原始数据、运算结果记录到纸上是用笔来完成的。而整个运算过程的控制又是由人来完成的。其步骤可概括如下：

①人将算题和原始数据用笔记录在纸上；

②人用算盘算出 21×12 的中间结果 252，用笔记录在纸上；

③人再用算盘算出 $117\div13$ 的中间结果 9，用笔记录在纸上；

④最后，人用算盘将第一个中间结果 252 减去第二个中间结果 9，得出最后结果 243，用笔记录在纸上。

用计算机来完成这样的任务，首先需要一个起算盘作用，能完成各种运算的部件，这称为运算器。其次需要一个起纸作用，用来记录、存放原始数据、中间结果和运算结果的部件，这称为存储器。起笔作用，用来将原始数据输入到存储器的部件，这称为存储器。起笔作用，用来将原始数据输入到存储器的部件，这称为输入设备；而用来将中间结果或最后结果输出的部件，这称为输出设备。这是计算机算题和人工算题相似之处。

但计算机和人工算题本质不同之处在于：计算机算题的每一步不需要人工控制，人的任务只是编写程序(Program)和操作计算机。程序由输入设备输入到存储器中存放后，启动计算机，计算机就根据程序的步骤，在不同时间向各个部件发出控制命令，自动完成计算任务的全过程。我们称发出控制命令的部件叫控制器。

1.1.1.2　电子计算机的基本组成

综上所述，电子计算机由 5 大部件组成，它们是：运算器、控制器、存储器、输入设备和输出设备。这 5 大部件关系如图 1－1 所示。

图中，有两类信息在流动：一类是数据，用双线表示，包括原始数据、中间结果、计算结果以及程序代码；另一类是控制命令，用单线表示。不管是数据还是控制命令，在机器中都是用“0”和“1”表示的二进制数。

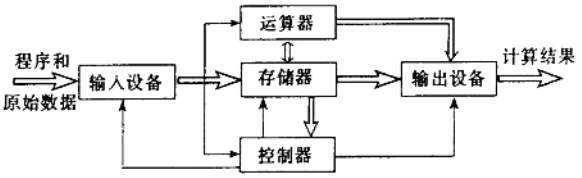


图 1-1　计算机的基本结构框图

1.1.2　电子计算机的工作过程

首先，把原始数据和程序通过输入设备存入存储器中的一个个单元里；然后，操作计算机让其启动，所存的程序一条条地被取到控制器中，控制器根据指令的不同发出不同的控制命令，自动进行全部运算；最后，通过输出设备输出计算结果。运算过程中，数据从存储器取入运算器进行运算，得到的中间结果和最后结果可存入存储器中，也可由输出设备输出。

下面仍以前面所举的运算题目为例，将计算机的工作过程归结如下：

第一步：由输入设备将事先编写好的程序（即解题步骤）和原始数据——21、12、117、13输入到存储器指定编号的单元存放起来。

第二步：命令计算机从第一条指令开始执行程序，计算机便在程序控制下自动完成解题的全过程。该过程包含以下操作：

①把第一个数据 21 从存储器中取到运算器（取数操作）；

②把第二个数 12 从存储器取到运算器，进行 21×12 的运算，并得到中间结果 252（乘法运算）；

③将运算器中的中间结果 252 送到存储器中暂时存放（存数操作）；

④把第三个数据 117 从存储器中取到运算器（取数操作）；

⑤把第四个数 13 从存储器中取到运算器，进行 $117\div13$ 的运算，并得到中间结果 9（除法运算）；

⑥将运算器中的中间结果 9 送到存储器中暂时存放（存操作数）；

⑦将暂存的两个中间结果先后取入运算器，进行 $252-9$ 的运算，得到最后结果 243（减法运算），并存入存储器中（存操作数）；

⑧将最终结果 243 直接用运算器或存储器经输出设备输出，如打印在纸上；

⑨停机。

以上就是迄今为止电子计算机所共同遵循的程序存储和程序控制的工作原理。这种原理是 1945 年由冯·诺依曼（John Von Neumann）提出的，所以又称为冯·诺依曼型计算机原理。

图 1-1 所示的 5 大基本组成部件是计算机的实体，统称为计算机的硬件（Hardware），而把包括解题步骤在内的各种各样的程序叫做计算机的软件（Software）。硬件中的运算器、控制器和存储器称为计算机的主机，其中的运算器和控制器相当于用算盘算题的人工系统中的算盘和人的作用，是计算机结构中核心部件，又称为中央处理器 CPU（Central Processing Unit）。

1.2 微型机中的数制与码制

1.2.1 进位计数制及各计数制间的转换

数制是人们对事物数量计数的一种统计规律。在日常生活中最常用的是十进制。但在微型机中，由于电气元件最易实现的是两种稳定状态，即器件的“开”与“关”，电平的“高”与“低”，因此，采用二进制数的“0”和“1”能很方便地表示机内的数据运算与存储。在编程时，为了方便阅读、书写和记忆，人们还经常用八进制数或十六进制数来表示二进制数。一个数可以不同计数制形式表示它的大小，但该数的量值则是相等的。

1.2.1.1 进位计数制

进位计数制是采用位置表示法，即同一数字在不同的数位所代表的数值是不同的。每一种进位计数均包含着两个基本的因素：

①基数 R(Radix)：它代表计数制中所用的数码个数。如：二进制计数中用到 0 和 1 两个数码；而八进制计数中用到 0～7 共 8 个数码。一般地说，基数为 R 的计数制(简称 R 进制)中，包含 0、1…、R－1 个数码，进位规律为“逢 R 进一”。

②位权 W(Weight)：进位计数制中，某个数位的值是由这一位的数码值乘以处在这一位的固定常数决定的，我们把这一固定常数称之为位权值，简称位权。各位的位权是以 R 为底的幂。如十进制基数 R＝10，则个位、十位、百位上的位权分别为 10^0，10^1，10^2。

因此，一个 R 进制数 N，可以用以下两种形式表示：

①并列表示法，或称位置计数法：

$$(N)_R=(K_{n-1}K_{n-2}\cdots K_1K_0K_{-1}K_{-2}\cdots K_{-m})_R$$

②多项式表示法，或称权展开式：

$$(N)_R=K_{n-1}R^{n-1}+K_{n-2}R^{n-2}+\cdots+K_1R^1+K_0R^0+K_{-1}R^{-1}+\cdots+K_{-m}R^{-m}$$

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

其中，m、n 为正整数，n 代表整数部分的位数；m 代表小数部分的位数；K_i 代表 R 进制中的任一个数码，$0\leqslant K_i\leqslant R-1$。

1．二进制数

二进制数，R＝2，K_i 可取 0 或 1，进位规律为“逢二进一”。任一个二进制数 N 可表示为：

$$(N)_2=K_{n-1}2^{n-1}+K_{n-2}2^{n-2}+\cdots+K_12^1+K_02^0+K_{-1}2^{-1}+\cdots+K_{-m}2^{-m}$$

例如：$(1001.101)_2=1\times2^3+0\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$

2．八进制数

八进制数，R＝8，K_i 可取 0～7 共 8 个数码中的任意一个，进位规律为“逢八进一”。任意一个八进制数 N 可以表示为：

$$(N)_8=K_{n-1}8^{n-1}+K_{n-2}8^{n-2}+\cdots+K_18^1+K_08^0+K_{-1}8^{-1}+\cdots+K_{-m}8^{-m}$$

例如：$(246.12)_8=2\times8^2+4\times8^1+6\times8^0+1\times8^{-1}+2\times8^{-2}$

3．十六进制数

十六进制数，R＝16，K_i 可取 0～15 共 16 个数码中的任一个，但 10～15 分别用 A、B、C、D、E、F 表示，进位规律为“逢十六进一”。任意一个十六进制数 N 可表示为：

$$(N)_{16}=K_{n-1}16^{n-1}+K_{n-2}16^{n-2}+\cdots+K_1 16^1+K_0 16^0+K_{-1}16^{-1}+\cdots+K_{-m}16^{-m}$$

例如：$(2D07.A)_{16}=2\times16^3+13\times16^2+0\times16^1+7\times16^0+10\times16^{-1}$

注意：在实际表示中，一个十六进制数如果最高位数字是字母(A~F)，则字母前必须加一个数字0，以便与变量名相区别。

以上三种进制数与十进制数的对应关系如表1-1所示。为避免混淆，除用$(N)_R$的方法区分不同进制数外，还常用在数字后加字母作为标注。其中：字母B(Binary)表示为二进制数；字母Q(Octave的缩写为字母O，为区别数字0写为Q)表示为八进制数；字母D(Decimal)或不加字母表示为十进制数；字母H(Hexadecimal)表示为十六进制数。

表1-1 二、八、十、十六进制数码对应表

十进制	二进制	八进制	十六进制
0	0000B	0Q	0H
1	0001B	1Q	1H
2	0010B	2Q	2H
3	0011B	3Q	3H
4	0100B	4Q	4H
5	0101B	5Q	5H
6	0110B	6Q	6H
7	0111B	7Q	7H
8	1000B	10Q	8H
9	1001B	11Q	9H
10	1010B	12Q	0AH
11	1011B	13Q	0BH
12	1100B	14Q	0CH
13	1101B	15Q	0DH
14	1110B	16Q	0EH
15	1111B	17Q	0FH
16	10000B	20Q	10H

1.2.1.2 各种进制数间的相互转换

1. 各种进制数转换成十进制数

各种进制数转换成十进制的方法是：将各进制数先按权展开成多项式，再利用十进制运算法则求和，即可得到该数对应的十进制数。

例1-1 将数1001.101B;246.12Q;2D07.AH转换为十进制数。

$$1001.101B=1\times2^3+0\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$$
$$=8+1+0.5+0.125=9.625$$

$$246.12Q=2\times8^2+4\times8^1+6\times8^0+1\times8^{-1}+2\times8^{-2}$$
$$=128+32+6+0.125+0.03125=166.15625$$

$$2D07.AH=2\times16^3+13\times16^2+0\times16^1+7\times16^0+10\times16^{-1}$$
$$=8129+3328+7+0.625=11527.625$$

2. 十进制数转换为二、八、十六进制数

任一十进制数N转换为Q进制数，须将整数部分与小数部分分别进行转换，然后再用小数点将已转换过这两部分连接起来。

(1)整数部分转换步骤：

第一步：用q去除N的整数部分，得到商和余数，记余数为q进制整数的最低位数码

K_0；

第二步：再用 q 去除得到的商，求出新的商和余数，记余数为 q 进制整数的次低位数码 K_1；

第三步：重复第二步，直至商为零，整数转换结束。此时，记余数为 q 进制整数的最高位数码 K_{n-1}。

例 1－2 将数 168 转换为二、八、十六进制数。

```
2 | 168
  2 | 84   余数 0, K0 = 0
    2 | 42   余数 0, K1 = 0
      2 | 21   余数 0, K2 = 0
        2 | 10   余数 1, K3 = 1
          2 | 5   余数 0, K4 = 0
            2 | 2   余数 1, K5 = 1
              2 | 1   余数 0, K6 = 0
                0   余数 1, K7 = 1
∴ 168 = 10101000B
```

```
8 | 168
  8 | 21   余数 0, K0 = 0
    8 | 2   余数 5, K1 = 5
      0   余数 2, K2 = 2
∴ 168 = 250Q
```

```
16 | 168
   16 | 10   余数 8, K0 = 8
      0   余数 10, K1 = A
∴ 168 = A8H
```

(2)小数部分转换步骤：

第一步：用 q 去乘 N 的纯小数部分，得到乘积的整数部分，记为 q 进制小数的第一个数码 K_{-1}；

第二步：再用 q 去乘上次乘积的纯小数部分，得到新乘积的整数部分，记为 q 进制小数的次位数码 K_{-i}；

第三步：重复第二步，直至乘积的小数部分为零，或者达到所需要的精度位数为止。此时，乘积的整数部分，记为 q 进制小数位的数码 K_{-m}。

例 1－3 将数 0.636 转换为二、八、十六进制数。

$0.686\times2=1.372$ $K_{-1}=1$	$0.686\times8=5.488$ $K_{-1}=5$	$0.686\times16=10.976$ $K_{-1}=A$
$0.372\times2=0.744$ $K_{-2}=0$	$0.488\times8=3.904$ $K_{-2}=3$	$0.976\times16=15.616$ $K_{-2}=F$
$0.744\times2=1.488$ $K_{-3}=1$	$0.904\times8=7.232$ $K_{-3}=7$	$0.616\times16=9.858$ $K_{-3}=9$
$0.488\times2=0.976$ $K_{-4}=0$	$0.232\times8=1.856$ $K_{-4}=1$	$0.856\times16=13.696$ $K_{-4}=D$
$0.976\times2=1.952$ $K_{-5}=1$	$0.856\times8=6.848$ $K_{-5}=6$	$0.696\times16=11.136$ $K_{-5}=B$
∴ $0.686\approx0.10101B$	∴ $0.686\approx0.53716Q$	∴ $0.686\approx1.AF9DBH$

例 1－4 将数 168.636 转换为二、八、十六进制数。

依据例 1－2、例 1－3 可得：

168.686＝10101000.1010B

168.686＝250.53716Q

168.686＝A8.AF9DBH

由以上例子可以看出，二进制表示的数愈精确，所需的数位就愈多，很不方便书写和记忆，且易出错。若用同样数位表示数，八、十六进制数所表示数的精度高。所以在汇编语言中常用八进制或十六进制数作为二进制数的编码，这样书写方便，且易于记忆。

3. 二进制数与八进制数之间的相互转换

因为 $2^3=8$，所以采用“合三为一”的原则，从小数点开始分别向左、右两边各以三位为一组进行二——八换算，若不足三位以 0 补足，即可将二进制数转换为八进制数。

例 1-5 将数 1111011.0101B 转换为八进制数。

∵ 001 111 011 . 010 100

1 7 3 . 2 4

∴ 1111011.0101B=173.24Q

反之，采用“一分为三”的原则，每位八进制数用三位二进制数表示，即可将八进制数转换为二进制数。

例 1-6 将数 1357.246Q 转换成二进制数。

∵ 1 3 5 7 . 2 4 6

001 011 101 111 . 010 100 110

∴ 1357.246Q=1011101111.01010011B

4. 二进制数与十六进制数之间的相互转换

因为 $2^4=16$，所以采用“合四为一”的原则，从小数点开始分别向左、右两边各以四位为一组进行二——十六换算，若不足四位以 0 补足，即可将二进制数转换为十六进制数。

例 1-7 将数 1101000101011.001111B 转换为十六进制数。

∵ 0001 1010 0010 1011 . 0011 1100

1 A 2 B . 3 C

∴ 1101000101011.001111B=1A2B.3CH

反之，采用“一分为四”的原则，每位十六进制数用四位二进制数表示，即可将十六进制数转换为二进制数。

例 1-8 将数 4D5E.6FH 转换成二进制数。

∵ 4 D 5 E . 6 F

0100 1101 0101 1110 . 0110 1111

∴ 4D5E.6FH=100110101011110.01101111B

1.2.2 二进制数的运算与带符号数的表示方法

1.2.2.1 二进制数的算术运算

二进制数不仅物理上容易实现算术运算，而且运算规则也比较简单，其加、减法遵循“逢二进一”、“借一当二”的原则，和、差、积的运算规律如下：

0+0=0	0-0=0	0×0=0
0+1=1	0-1=1 有借位	0×1=0
1+0=1	1-0=1	1×0=0
1+0=10 有进位	1-1=0	1×1=1

下面将通过四个例子来说明二进制数的加、减、乘、除运算过程。

例 1-9 求 11001010B+11101B

被加数		11001010
加数		11101
进位	+)	00110000
和		11100111

∴ 11001010B+11101B=11100111B

由此可见，两个二进制数相加时，每一位有三个数参与运算(本位被加数、加数、低位进位)，从而得到本位的和，以及向高位的进位。

例 1－10 求 10101010B－10101B

初减数	10101010
减数	10101
借位 －)	00101010
差	10010101

∴ 10101010B－10101B＝10010101B

由此可知，二进制减法与加法类似，每一位有三个数参与运算(本位被减数、减数、低位借位)，从而得到本位的差，以及向高位的借位。

例 1－11 求 110011B×1011B

初乘数	110011
乘数 ×)	1011
	110011
	110011
	000000
+)	110011
积	1000110001

∴ 110011B×1011B＝1000110001

由此可知，二进制数乘法与十进制数乘法相类似，可用乘数的每一位去乘被乘数，乘得的中间结果的最低有效位与相应的乘数位对齐，若乘数位为 1，则中间结果为被乘数；若乘数位为 0，则中间结果为 0，最后把所有中间结果同时相加即可得到乘积。这种算法对微型机实现很不方便。在没有乘法指令的微型机中，常采用被乘数左移或部分积右移的方法编程，来实现乘法运算。目前 8086 以上微型机均有专门的乘法指令来完成乘法(详见本书第三章指令系统的介绍)，给用户带来许多方便，提高了机器的运算速度。

例 1－12 求 100100B÷101B

```
      111
101 ) 100100
      101
      ---
      1000
       101
      ----
       110
       101
      ----
         1
```

∴ 100100B÷101B＝111B 余 1

由此可知，二进制数除法是二进制数乘法的逆运算，8086 以上微型机提供专门的除法指令来完成除法运算。

1.2.2.2 二进制数的逻辑运算

1.“与”运算(AND)

“与”运算又称逻辑乘，运算符为·或∧。“与”的运算规则如下：

0·0＝0　　0·1＝1·0＝0　　1·1＝1

例 1－13 若二进制数 A＝10101111，B＝01011110 求 A·B

$$\begin{array}{rr} & 10101111 \\ \wedge & 01011110 \\ \hline & 00001110 \end{array} \qquad \therefore \quad A \cdot B = 00001110$$

由此可见，若要保留 A 中的某位，就可在 B 中对应某位为 1 与其进行“与”运算；若要 A 中某位为 0，只要使 B 中对应位为 0 与其进行“与”运算即可。所以，在微型机控制或运算中可以用“与”逻辑屏蔽掉一些位（即：使一些位为 0），或保留一些位不变。

2.“或”运算（OR）

“或”运算又称逻辑加，运算符为＋或 V。“或”的运算规则如下：

$0+0=0 \qquad 0+1=1+0=1 \qquad 1+1=1$

例 1－14 若二进制数 A＝10101111，B＝01011110 求 A＋B

$$\begin{array}{rr} & 10101111 \\ V & 01011110 \\ \hline & 11111111 \end{array} \qquad \therefore \quad A+B = 11111111$$

由上可知，“或”运算与算术加运算的区别在于不产生进位。无论 A 中某位为 0 或 1，只要 B 中某位为 1，就可使该位置 1；B 中某位为 0，就可使该位结果不变，所以，在微型机控制或运算中可以用“或”运算给某位置 1。

3.“非”运算（NOT）

“非”运算又称逻辑非，变量 A 的“非”运算记作 $\bar{A}$。“非”的运算规则如下：

$\bar{1}=0 \qquad \bar{0}=1$

例 1－15 若二进制数 A＝10101111，求 $\bar{A}$

$\bar{A}=\overline{10101111}=01010000$

由此可知，逻辑“非”运算可使 A 中各位结果均发生反变化，即 0 变 1，1 变 0。

4.“异或”运算（XOR）

“异或”运算的运算符为∀或⊕。异或的运算规则如下：

$0 \oplus 0=0 \qquad 0 \oplus 1=1 \oplus 0=1 \qquad 1 \oplus 1=0$

例 1－16 若二进制数 A＝10101111，B＝01011110 求 A⊕B

$$\begin{array}{rr} & 10101111 \\ \oplus & 01011110 \\ \hline & 11110001 \end{array} \qquad \therefore \quad A \oplus B = 11110001$$

由此可知，A 与 B 中两个数值相同位“异或”的为 0，否则为 1。用 A⊕A 就可实现 A 清 0，用“异或”运算还可检验两个数是否相等，即：若 A⊕B＝0，则说明 A＝B。

以上二进制数的算术、逻辑运算规则奠定了微型机中数据运算与控制的基础。

1.2.2.3 *带符号数的表示方法——原码、反码、补码*

在 1.2.2.1，1.2.2.2 中讨论的二进制数运算均为无符号数的运算，但实际的数值是带有符号的，既可能是正数，也可能是负数，运算的结果也可能是正数，也可能是负数。那么在微型机中就存在着如何表示正、负数的问题。

由于微型机只能识别 0 和 1，因此，在微型机内把一个二进制数的最高位作为符号位，用来表示数值的正与负（若用 8 位表示一个数，则 D_7 位为符号位；若用 16 位表示一个数，则 D_{15} 位为符号位），并用 0 表示“＋”；用 1 表示“－”。

例如：$N_1=+1011$，$N_2=-1011$ 在微型机中用 8 位二进制数可分别表示为：

把原来的二进制数连同符号位一起作为一个新数，称之为机器数，原来二进制数的数值称为机器数的真值。为了运算方便（即把减法变为加法），在机器中，带符号数有 3 种表示方法——原码、反码和补码。

1. 原码

正数的符号位用 0 表示，负数的符号位用 1 表示，数值部分保持不变，这种表示方法，称之为原码，用$[X]_{原}$表示。

设用 n 位二进制数表示一个整数 X，则其原码定义为：

$$[X]_{原}=\begin{cases}0. \quad K_{n-2}K_{n-3}\cdots K_1K_0 & X\geqslant 0\\ 1. \quad K_{n-2}K_{n-3}\cdots K_1K_0 & X\leqslant 0\end{cases}$$

例如：+115 和 −115 在微型机中（设机器字长为 16 位），其原码可分别表示为：

$[+115]_{原}=0000000001110011$；　$[-115]_{原}=1000000001110011$

若用 m 位二进制数表示一个小数 X，则其原码定义为：

$$[X]_{原}=\begin{cases}0. \quad K_{-1}K_{-2}\cdots K_{-m} & X\geqslant 0\\ 1. \quad K_{-1}K_{-2}\cdots K_{-m} & X\leqslant 0\end{cases}$$

例如：0.125 与 −0.125 在微型机中（设机器字长为 8 位），其原码可分别表示为：

$[0.125]_{原}=0.0010000$；　$[-0.125]_{原}=1.0010000$

原码表示法简单易懂，与真值的转换方便，但缺点是，当两个数做加减运算时，微型机必须先判断两个数的符号是否相同。如果符号相同，则数值相加，符号不变；如果符号不同，就要做减法。首先要比较两个数绝对值的大小，然后以大减小，最后还要选择适当的符号。在微型机中，判断绝对值的大小和符号是否相同，会使运算器的逻辑复杂化，增加机器的运行时间。为了克服原码运算的缺点，简化微型机结构，提高运行速度，在微型机运算中引入补码的概念，使正、负数的加法和减法运算简化为单一的加法运算。

2. 补码与反码

(1)补码的概念

在日常生活中有许多“补”数的事例。如钟表，若标准时间为 6 点正，而钟表却指在 9 点，要把表拨准，可以有两种拨法：一种是倒拨 3 小时，即 9−3=6；另一种是顺拨 9 小时，即 9+9=18。尽管将表针倒拨或顺拨不同的时数，但却得到相同的结果，即 9−3 与 9+9 是等价的。这是因为钟表的计量范围是以 12 为一个计数循环，12 称为时钟的“模”(Mod)。

模(Mod)是一个系统的量程或此系统所能表示的最大数，它会自然丢掉，即：

9−3=9+9=12+6→6　(Mod12　自然丢掉)

称 +9 是 −3 在模为 12 时的补数。

使用补码的目的，一是使减法运算变成加法运算，二是使符号位成为数值的一部分，直接参加数值运算，从而简化运算规则。

一般情况下，任一整数 X，在模为 K 时的补数可用下式表示：

$$[X]_{补}=X+K \quad (\text{Mod } K)$$

$$=\begin{cases}X & 0\leqslant X<K\\ K-|X| & -K\leqslant X<0\end{cases}$$

在微型机运算与存储中，用二进制码表示数据，其数据的位数，即字长总是有限的。设字长为 n，则两数相加求和时，如果 n 位的最高位产生进位，就会丢掉，这正是在模的意义下相加的概念，丢掉的进位即为模 2^n。所以，以 2^n 为模的补数，称之为 2 的补码，简称补码。

由补数的概念引伸，当用 n 位二进制数表示整数 X(1 位为符号位，n－1 位为数值位)，模为 2^n 时，数 X 的补码可表示为：

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

当 X 为小数时，其补码可表示为：

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

由上式可知，正数的补码与其原码相同，只有对负数才有求补的问题。

(2)负数补码的求法

补码的求法一般有两种：

①用补码定义式 $[X]_{补}=2^n+X=2^n-|X|\qquad -2^{n-1}\leqslant X<0$ (整数)

$[X]_{补}=2+X=2-|X|\qquad -1\leqslant X<0$ (小数)

在用补码定义式求补码的过程中，要做一次减法很不方便，该法一般不用。

例如：X＝－0101111B，n＝8，则

$$\begin{aligned}[X]_{补}&=2^8+(-0101111\text{B})\\&=100000000\text{B}-0101111\text{B}\\&=11010001\text{B}\qquad(\text{Mod } 2^8)\end{aligned}$$

②用原码求反码，再在数值末位加 1 可得到补码。即：$[X]_{补}=[X]_{反}+1$

(3)反码：一个正数的反码，等于该数的原码；一个负数的反码，等于该负数的原码符号位不变(即为 1)，数值位按位求反(即 0 变 1，1 变 0)；或者是在该负数对应的正数原码上连同符号位，逐位求反。反码用$[X]_{反}$表示。

例 1－17 $X_1=+369$，$X_2=-369$，$X_3=-0.369$，当用 16 位二进制数表示一个数时，求 X_1、X_2、X_3 的原码、反码及补码。

$[X_1]_{原}=[X_1]_{反}=[X_1]_{补}=0000000101110001\text{B}$

$[X_2]_{原}=1000000101110001\text{B}$

$[X_2]_{反}=1111111010001110\text{B}$

$[X_2]_{补}=[X_2]_{反}+1=1111111010001111\text{B}$

$[X_3]_{原}=1.010111100111011\text{B}$

$[X_3]_{反}=1.101000011000100\text{B}$

$[X_3]_{补}=[X_3]_{反}+0.000000000000001\text{B}$

$=1.101000011000101\text{B}$

综上所述可得：正数的原码、反码、补码就是该数本身；负数的原码是符号位为 1，数值位不变；负数的反码是符号位为 1，数值位逐位求反；负数的补码是符号位为 1，数值位逐位求反再在末位加 1。

3. 补码的运算规则

补码的运算规则如下：

①$[X+Y]_{补}=[X]_{补}+[Y]_{补}$

即：任何两个数相加，无论其正负号如何，只要对它们各自的补码进行加法运算，就可得到正确的结果，该结果是补码形式。

②$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

即：任意两个数相减，只要对减数连同“-”号求补，则变成[被减数]补与[-减数]补相加，就可得到正确结果，该结果是补码形式。

③$[[X]_{补}]_{补}=[X]_{原}$

对于运算产生的补码结果，若要转换为原码表示，则正数的结果$[X]_{补}=[X]_{原}$；负数结果，只要对该补码结果再做一次求补码运算，就可得到负数的原码结果。

例 1-18 X=25DFH，Y=327BH，用补码运算求：X+Y，X-Y。

∵ X=0010010111011111B=$[X]_{补}$

Y=0011001001111011B=$[Y]_{补}$

$[-Y]_{补}$=1100110110000101B

$[X+Y]_{补}=[X]_{补}+[Y]_{补}$=0101100001011010B=585AH

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$=1111001101100100B

∴ X+Y=0101100001011010B=+585AH

X-Y=$[[X-Y]_{补}]_{补}$=1000110010011100B=-0C9CH

1.2.3 机器内小数的表示方法

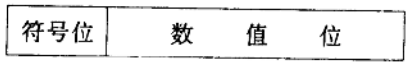
在微型机中，数据不仅有符号，而且经常含有小数，即既有整数部分又有小数部分。对小数的表示主要表现在对小数点位置的处理上，其内容包括：一是如何表示一个带小数点的数；二是如何对带小数点的数进行运算。一般有两种处理方式：定点表示法和浮点表示法。

1.2.3.1 定点表示法

在微型机中，小数点的位置固定不变，称之为定点表示法。这个固定的位置是事先约定好的，不必用符号表示。用定点法表示的实数叫做定点数。通常，定点表示也有两种方法：

1. 定点整数表示法

小数点固定在最低数值位之后，机器中能表示的所有数都是整数，称之为定点整数表示法。其格式如下：



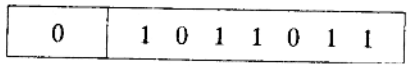
设定的小数点位置

当用 n 位表示数 N 时，一位为符号位，n-1 位为数值位，则 N 的范围是：

$-2^{n-1}\leqslant N\leqslant 2^{n-1}-1$

即：n=8，$-128\leqslant N\leqslant 127$；n=16，$-32768\leqslant N\leqslant 32767$。

例如：N=+1011011，n=8，则在微型机内用定点整数法，将 N 表示为：



2. 定点小数表示法

小数点固定在最高数值位之前，机器中能表示的所有数即为纯小数。这种方法称之为