高等院校计算机基础
教育系列课程教材

微型计算机原理及应用

孙家启　张毅坤　主编
陈金华　主审

中国科学技术出版社

高等院校计算机基础教育系列课程教材

微型计算机原理及应用

孙家启 主编
张毅坤
陈金华 主审

中国科学技术出版社
·北 京·

图书在版书目（CIP）数据

微型计算机原理及应用/孙家启等主编．－北京：中国科学技术出版社，1997.8
高等院校计算机基础教育系列课程教材
ISBN 7-5046-2198-6

Ⅰ.微… Ⅱ.孙… Ⅲ.微型计算机-基本知识-高等学校-教材 Ⅳ.TP36

中国版本图书馆 CIP 数据核字（97）第 08495 号

中国科学技术出版社出版
北京海淀区白石桥路 32 号 邮政编码 100081
新华书店北京发行所发行 各地新华书店经售
中国文联印刷厂印刷
*
开本：787×1092 毫米 1/16 印张：16.625 字数：400 千字
1997 年 8 月第 1 版 1997 年 10 月第 2 次印刷
印数：5001—13000 册 定价：20.00 元

内 容 提 要

本书是根据国家教委计算机基础课程教学指导委员会制定的非计算机专业“微型计算机原理及应用”课程的教学要求，为高等院校理、工类非计算机专业而编写的教材。

本书以 Intel 8086/8088 CPU 为目标机型，介绍了微型机系统的基本组成、结构特点和工作原理，主要内容包括：微型机基础知识，指令系统和汇编语言程序设计，存储器、中断系统、输入输出设备及其接口技术等，最后还简要介绍了高性能的 80386、80486 和 Pentinum 微处理器。

本书除适用于作高等院校非计算机专业本科生教材外，也可以作为高等专科学校、职工大学、成人教育计算机应用专业的教学用书，还可供从事微型机应用的广大工程技术人员参考。

责任编辑　张秀智　胡　萍
封面设计　炎　尘
正文设计　郑爱华
责任校对　刘红岩
责任印制　王　沛

机械工业部部属高校计算机基础教育系列课程教材出版说明

《中国教育改革和发展纲要》指出："要按照现代科学技术文化发展的新成果和社会主义现代化建设实际需要，更新教学内容，调整课程结构。"人类即将进入的21世纪，是高度信息化的社会。作为信息社会重要基础的计算机，将渗透到社会各个角落。计算机不仅作为一种工具来使用，而且作为一种文化来普及；计算机科学技术不仅是一门独立的学科，而且是所有学科知识结构中的重要组成部分。

在我国多年来，数、理、化及外语等基础教育在高等技术人才的素质培养中发挥了十分重要的作用。面对新的形势，加强高校非计算机专业计算机基础教育，将是促进高校教育质量提高的必要措施。在这方面，除了设备配置、师资培养、计划安排和课程设置等工作外，教材是加强计算机基础教育的基本建设。为此，机械工业部部属高校在总结多年来实施非计算机专业计算机基础教育的教学经验基础上，发挥知识群体优势，组织编写本系列教材，提供有关专业选择使用。

根据目前各专业课程设置，本系列教材包括3个层次：即计算机基础及高级程序设计语言，微型计算机原理及应用，软件技术基础及计算机网络概论。考虑到不同专业对计算机知识要求的差异，部分教材将按不同学时编写。由于计算机技术发展迅速，教材内容将不断更新，因此第一轮教材安排在1997年秋季全部出齐，以便在使用过程中修改完善，使第二轮教材以更新的和更好的面目问世。

本系列教材除适合于高校非计算机专业本科生、大专生使用外，也可用于成人教育及自学教育教学，同时也可作为工程技术人员自修计算机技术的参考书。

机械工业部部属高校计算机基础
教育系列课程教材编审委员会
1995年6月5日

前　言

计算机技术的发展日新月异。在32位、64位微型机成为主流机型的今天，仍然按8位机的教材组织教学，介绍微型机工作原理，已与实际相差甚远，内容也显得过于陈旧。有鉴于此，我们组织编写了本教材。它以Intel 8086/8088 CPU为目标，介绍了16位微型机的系统结构和工作原理。虽然16位机也已成为微型机发展史上的过去，但32位、64位微型机的向上兼容性及Intel系列产品的社会拥有量却保证了本教材的存在价值与积极意义。

本课程参考学时数为50～60学时（不含上机实验）。

本教材以高等学校非计算机专业的本科生为教学对象，同时兼顾高等专科学校、职工大学、成人教育的计算机专业教学需要。因此在编写上作了以下几点考虑。

1. 安排了微型机基础知识一章，以向读者提供必备的专门知识，减小跨度，扫除学习上的障碍；

2. 在基本概念及工作原理的叙述上，力求逻辑性、通俗性、易学性的统一；

3. 书中给出的适量例题，许多都是调试通过的或经实际运行证明可行的，它有助于读者对所学知识的理解、掌握甚至运用，各章所附习题可供课后复习和训练使用；

4. 重点讨论16位微型机系统，兼顾当前发展，还专门介绍了高性能的微处理器。

本书共分八章。第一、二章和第八章的8.2节由张毅坤编写；第三章、第八章的8.4节和附录由孙家启编写；第四章和第八章的8.1及8.3节由朱武编写；第五、六章由张正武编写；第七章由宣善立编写。全书由孙家启担任主编，张毅坤担任副主编。

江苏理工大学陈金华教授对全书进行了审阅，并提出了许多有价值的修改意见；本书在编写过程中得到了机械工业部部属高校计算机基础教育系列课程教材编审委员会的大力支持和帮助；中国科学技术出版社做了大量工作，使本书得以尽早与读者见面，在此一并表示衷心的感谢。

由于编者水平有限，书中难免存在错误和不当之处，恳请读者及专家批评指正。

编　者

1996年10月

目　录

第一章　微型计算机基础知识

1.1　微型机中的数制与码制

1.1.1　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，器件的“开”与“关”，电平的“高”与“低”，因此，采用二进制数的“0”和“1”来很方便地表示机内的数据运算与存储。在编程时，为了方便阅读和书写，人们还经常用八进制数或十六进制来表示二进制数。一个数可以用不同计数制形式表示它的大小，但该数的量值则是相等的。

1.1.1.1　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到的数码个数。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，1 个 R 进制数 N，可以用以下两种形式表示：

ⓐ 并列表示法，或称位置计数法：

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

ⓑ 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 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，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 可以代表为：

$(N)_8=K_{n-1}8^{n-1}+K_{n-2}8^{n-2}+\cdots+K_1 8^1+K_0 8^0+K_{-1}8^{-1}+\cdots+K_{-m}8^{-m}$

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，1 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 1 个 0，以便与变量名相区别。

以上 3 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，还常用数字后加字母作为标注。其中：字母 B(Binary)表示二进制数；字母 Q(Octal 的缩写为字母 O，为区别数字 0 写为 Q)表示八进制数；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

1.1.1.2　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。

$$1001.101B=1\times2^3+0\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$$
$$=8+1+0.5+0.125=9.625$$
$$246.12Q=2\times8^2+4\times8^1+6\times8^0+1\times8^{-1}+2\times8^{-2}$$
$$=128+32+6+0.125+0.03125=166.15625$$

$$2D07.AH = 2\times16^3 + 13\times16^2 + 0\times16^1 + 7\times16^0 + 10\times16^{-1}$$
$$= 8192 + 3328 + 7 + 0.625 = 11527.625$$

(2) 十进制数转换为二、八、十六进制数

任一十进制数 N 转换 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，余数又作为 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，余数作为转换后 Q 进制整数的最高位数码 K_{n-1}。

例 1-2 将 168 转换成二、八、十六进制数。

2	168	
2	84	余数 0，$K_0=0$
2	42	余数 0，$K_1=0$
2	21	余数 0，$K_2=0$
2	10	余数 1，$K_3=1$
2	3	余数 0，$K_4=0$
2	2	余数 1，$K_5=1$
2	1	余数 0，$K_6=0$
	0	余数 1，$K_7=1$

168 = 10101000B

8	168	
8	21	余数 0，$K_0=0$
8	2	余数 5，$K_1=5$
	0	余数 2，$K_2=2$

168 = 250Q

16	168	
16	10	余数 8，$K_0=8$
	0	余数 10，$K_1=A$

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，记下乘积的整数部分，作为 Q 进制小数的第一个数码 K_{-1}；

第二步：再用 Q 去乘上次积的纯小数部分，得到新乘积的整数部分，记为 Q 进制小数的次位数码 K_{-2}；

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。此时乘积的整数位，作为 Q 进制小数位的数码 K_{-m}。

例 1-3 将 0.686 转换成二、八、十六进制数。

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

可得：0.686 = 0.10101B　　0.686 = 0.53716Q　　0.686 = 0.AF9DBH

例 1-4 将 168.636 转换为二、八、十六进制数。

依据例 1-2、例 1-3 可得：

168.686=10101000.10101B

168.686=250.53716Q

168.686=A8.AF9DBH

由以上例子可以看出,二进制表示数愈精确,所需数位就愈多,很不方便书写和记忆,且易出错。另外,若用同样数位表示数,八、十六进制数所表示数的精度高。所以在汇编语言编程中常用八进制或十六进制数作为二进制数的缩码,来书写和记忆二进制数,这一点在学习第四章汇编语言程序设计时体会将更深一些。

(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,若不足三位的以0补足,即可将二进制数转换为八进制数。

例1-5 将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1101000101011.001111B转换成十六进制数。

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

1.1.2 二进制数的运算与带符号数的表示方法

1.1.2.1 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:

$0+0=0$　$0-0=0$　$0\times0=0$　$1+0=1$　$1-0=1$　$1\times0=0$

$0+1=1$　$0-1=1$ 有借位　$0\times1=0$　$1+1=10$ 有进位　$1-1=0$　$1\times1=1$

下面将通过4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3个数参与运算（本位被加数、加数、低位进位），从而得到本位和，以及向高位的进位。

例 1-10　求 10101010B-10101B

被减数		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3个数参与运算（本位被减数、减数、低位借位），从而得到本位的差和向高位的借位。

例 1-11　求 110011B×1011B

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

∴　110011B×1011B=1000110001B

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。这种算法对计算机实现很不方便。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（见本书第三章指令系统介绍），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。

1.1.2.2 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：

$0\cdot0=0$　　$0\cdot1=1\cdot0=0$　　$1\cdot1=1$

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

$$\begin{array}{rr} & 10101111 \\ \wedge & 01011110 \\ \hline & 00001110 \end{array}$$

∴ A·B=00001110

由此可见，若要保留 A 中的某位，就可在 B 中对应某位用 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 ∨。“或”的运算规则如下：

$0+0=0$　　$0+1=1+0=1$　　$1+1=1$

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

$$\begin{array}{rr} & 10101111 \\ \vee & 01011110 \\ \hline & 11111111 \end{array}$$

∴ A+B=11111111

由上可知，“或”运算与算术加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，因此，微型机中可利用“或”逻辑给某位置 1。

(3) “非”运算(NOT)

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

$\overline{1}=0$　　$\overline{0}=1$

例 1-15 若二进制数 A=10101111，求 $\overline{A}$

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

由此可知，逻辑“非”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：

$0\oplus0=0$　　$0\oplus1=1\oplus0=1$　　$1\oplus1=0$

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

$$\begin{array}{rr} & 10101111 \\ \oplus & 01011110 \\ \hline & 11110001 \end{array}$$

∴ A⊕B=11110001

由此可知，A 与 B 中两个数值相同位异或的结果为 0，否则为 1。利用 A⊕A 就可实现 A 清 0，利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。

1.1.2.3 常用符号的表示方法——原码、反码、补码

在 1.1.2.1 和 1.1.2.2 中讨论的二进制数运算均为无符号数运算，但实际的数值是带有

符号的，既可能是正数，符号用“+”号表示，也可能是负数，符号用“-”号表示，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，来表示数值的正与负（若用8位表示1个数，则D_7位为符号位；若用16位表示1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：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	1	0	1	1
符号	数值部分						

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1
符号	数值部分						

把原来的二进制数连同符号位一起作为1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（把减法变为加法），在机器中，有符号数有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，数值部分保持不变的二进制数表示方法，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：

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

若用m位二进制数表示一个小数X，则原码定义为：

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

例如：0.125与-0.125在微型机中（设机器字长为8位），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，微型机必须先要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，为了能直接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，为此，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点整，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。这是因为钟表是以12小时进位，超过12就从头算起，即：9+9=12+6，该12称之为模(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　(Mod 12　自然丢掉)

称+9是-3在模为12时的补数。因此,引入补数后就可使减法运算变为加法运算。

例如:$11-7=11+5\rightarrow 4 \quad (\text{Mod } 12)$

+5是-7在模为12时的补数。

一般情况下,任一整数X,在模为K时的补数可用下式表示:

$$[X]_{补数}=X+K \quad (\text{Mod } K)$$

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

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:

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

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:

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

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

$$[X]_{补}=2^8+(-0101111B)$$
$$=100000000B-0101111B$$
$$=11010001B \quad (\text{Mod } 2^8)$$

b. 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。

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

$[X_2]_{原}=1000000101110001B$

$[X_2]_{反}=1111111010001110B$

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

$[X_3]_{原}=1.010111100111011B$

$[X_3]_{反}=1.101000011000100B$

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