中等专业学校试用教材

微型计算机原理及其应用

福建机电学校　陈立周　主编

机械工业出版社

78/80

中等专业学校试用教材

微型计算机原理及其应用

福建机电学校　陈立周　主编

学院图书馆
藏书专用章

机械工业出版社

(京)新登字054号

本书共分十章。第一至第八章主要介绍微型计算机的基本组成和基本原理，Z80 指令系统和汇编语言程序设计，接口技术以及微型计算机硬件软件方面的有关知识。书中附有练习题、实验指导书、Z80 指令系统表以及 TP801 单板机总图。

本书适合作机械系统中专电类专业微型计算机原理及其应用课程的教材。除第九章外，其余各章也参照机械类专业教学大纲的要求进行编写，所以亦可供机械类专业选用。

微型计算机原理及其应用

福建机电学校　陈立周　主编

*

责任编辑：贡克勤　　责任校对：罗文莉

封面设计：郭景云　　版式设计：冉晓华

责任印刷：王国光

*

机械工业出版社出版（北京阜成门外百万庄南街一号）

（北京市书刊出版业营业许可证出字第117号）

北京市密云县印刷厂印刷

新华书店北京发行所发行·新华书店经售

*

开本 787×1092 1/16 · 印张 14 3/4 · 插页 1 字数367千字

1989年 11 月重庆第 1 版 · 1994年 10 月北京第 6 次印刷

印数 78 801—98 800 · 定价：8.95元

*

ISBN 7-111-01790-0/TP.101（课）

目　录

第一章　计算机中的数与代码……1
§1-1　进位计数制……1
§1-2　不同进位计数制之间的互换……3
§1-3　带符号的二进制数……4
§1-4　二进制数的运算……6
§1-5　BCD码和文字符号代码……11
练习题……13
第二章　微型计算机的基础知识……14
§2-1　微型计算机的发展概况……14
§2-2　微型计算机系统的组成……15
§2-3　微处理器……20
§2-4　微型计算机的工作过程……27
练习题……29
第三章　Z80指令系统……30
§3-1　Z80指令系统的分类、书写格式和寻址方式……30
§3-2　传送指令……35
§3-3　算术运算和逻辑运算指令……40
§3-4　通用算术指令与CPU控制指令……44
§3-5　移位和循环指令……45
§3-6　位操作指令……47
§3-7　转移指令……47
§3-8　调用、返回、输入、输出和数据块传送、搜索指令……50
练习题……54
第四章　Z80汇编语言程序设计……58
§4-1　汇编语言程序的格式……58
§4-2　伪指令……59
§4-3　汇编语言程序的编写步骤及基本结构……61
§4-4　运算程序的设计……67
§4-5　非数值操作程序……71
练习题……77
第五章　半导体存贮器……79
§5-1　存贮器的分类……79
§5-2　随机存取存贮器（RAM）……80
§5-3　只读存贮器（ROM）……83
§5-4　CPU与存贮器的连接……86
§5-5　CPU的时序和存贮器存取速度之间的配合……91
练习题……95
第六章　输入输出与中断……96
§6-1　输入输出设备与接口……96
§6-2　输入输出的传送方式……98
§6-3　中断的基本概念……100
§6-4　Z80中断系统……105
练习题……112
第七章　可编程序通用接口……114
§7-1　Z80 PIO接口芯片……114
§7-2　Z80 CTC接口芯片……123
§7-3　Z80 SIO接口芯片……133
练习题……139
第八章　A/D及D/A转换……141
§8-1　概述……141
§8-2　D/A转换……142
§8-3　A/D转换……147
练习题……152
第九章　单板机和单片机……153
§9-1　单板机的结构……153
§9-2　TP801单板机的操作与使用……160
§9-3　TP801单板机的监控程序……163
§9-4　单片微型计算机……173
第十章　微型计算机的应用……177
§10-1　概述……177
§10-2　顺序控制系统……178
§10-3　数据采集系统……183
附录一　实验……187
实验预备知识……187
实验一　基本命令键的使用……189
实验二　算术运算程序设计与调试……191
实验三　子程序设计……192
实验四　存贮器的扩充……192
实验五　信息存贮和EPROM固化操作……196
实验六　Z80　PIO基本实验……198
实验七　Z80　CTC基本实验……201
实验八　综合实验……205
附录二　Z80指令的机器码表……209
附录三　Z80指令系统的功能表……219
附录四　ASCII表……230
附录五　TP801总逻辑图……插页
参考文献

第一章　计算机中的数与代码

§1-1　进位计数制

进位计数制是按进位原则进行计数的一种方法。长期以来，人们在日常生活中形成了多种的进位计数制，例如1cm等于10mm 的十进制，英制中 1ft 等于 12in 的十二进制，我国旧制中1市斤等于16两的十六进制，1分等于60秒的六十进制等等。在计算机内部，因为只能通过电位的高低，表示数码0和1，所以计算机内只能使用二进制。但是二进制由于数码冗长，书写和阅读都不太方便，所以我们在编写程序时，习惯用十六进制来代替二进制。或者用十进制来代替。为此，在学习计算机时，需要熟悉这三种进位计数制的使用以及三种进位计数制的相互转换。

一、十进制

十进制用0～9十个数码表示数的大小，超过 9 就要进位，这就是所谓逢十进一。同样的数码放在不同的“位”上，它所代表的数值也就不同，这称之为位权。十进制各位的位权可以用底数为10的n次幂来确定。例如：第1位的位权为$10^0=1$；第2 位的位权为$10^1=10$等等。任意一个整数，如6543，按位权计数法可以表示为

$$(6\times10^3)+(5\times10^2)+(4\times10^1)+(3\times10^0)$$
$$=6543$$

对于小数，小数点以后各位的位权则可用底数为10的负 n 次幂来确定。例如：小数点后的第1位位权为$10^{-1}=0.1$；小数点后的第2位位权为 $10^{-2}=0.01$ 等等。任意一个小数，如0.654，按位权计数法，可以表示为

$$(6\times10^{-1})+(5\times10^{-2})+(4\times10^{-3})=0.654$$

可见，任意一个十进制的数，都可以按位权展开为

$$D=D_{n-1}\times10^{n-1}+D_{n-2}\times10^{n-2}+\cdots\cdots+D_1\times10^1+D_0\times10^0$$
$$+D_{-1}\times10^{-1}+D_{-2}\times10^{-2}+\cdots\cdots+D_{-m}\times10^{-m}$$

式中　n为小数点左边的位数（第1位规定为0位），m为小数右边的位数，m、n为正整数。

二、二进制

二进制用 0 和 1 两个数码表示数的大小，低位向高位进位的原则为逢二进一。整数各数位的位权用底数为2 的 n 次幂来确定，即小数点左边第 1 位的位权为 $2^0=1$，第2位的位权为 $2^1=2$，第3位的位权为 $2^2=4$等等，任意一个二进制数，如 10101B，其对应于十进制数的值为

$$10101B=(1\times2^4)+(0\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)$$
$$=21$$

式中在数的末尾加上一个符号B，表示这个数是一个二进制数，不加任何符号的数，表示这个数是一个十进制数。如式中的21，就表示它是一个十进制数。

对于二进制的小数，其小数点以后各位的位权，也可以用底数为 2 的负 n 次幂来确定，

此为试读，需要完整PDF请访问：www.ertongbook.com

例如小数点后的第1位其位权为$2^{-1}=0.5$，小数点后的第2位其位权为$2^{-2}=0.25$等等。任意一个二进制小数，如0.1101B，相当于十进制数的值为

$$0.1101B=(1\times2^{-1})+(1\times2^{-2})+(0\times2^{-3})+(1\times2^{-4})$$
$$=0.8125$$

可见，任意一个二进制数，都可以按位权展开为

$$B=B_{n-1}\times2^{n-1}+B_{n-2}\times2^{n-2}+\cdots\cdots+B_1\times2^1+B_0\times2^0$$
$$+B_{-1}\times2^{-1}+B_{-2}\times2^{-2}+\cdots\cdots+B_{-m}\times2^{-m}$$

式中 n为小数点左边的位数（第1位为0位），m为小数点右边的位数，m和n都是正整数。

三、十六进制

计算机使用的二进制数，其位数通常为8位、16位、32位等。这么长的位数，而且只有0和1两个数码，不论书写还是记忆都十分不便，且容易出错。为此在编写程序的时候，常常把4位二进制数，缩成1位十六进制，这样，在书写时就简单得多。

十六进制使用0～9A～F共16个符号作为数码，它对应于十进制0～15这16个数，一般规定在十六进制的数末尾加上符号H，或者在数的前面加上符号$，以便与十进制数相区别。例如5543是一个十进制数，而5543H或$5543则表示它是一个十六进制的数。

十六进制整数各数位的位权可用底数为16的n次幂来确定。即小数点左边第1位的位权为$16^0=1$，第2位的位权为$16^1=16$等等。任意一个十六进制数，如8A71H对应于十进制的数值为

$$8A71H=(8\times16^3)+(10\times16^2)+(7\times16^1)+(1\times16^0)$$
$$=35441$$

对于十六进制的小数，其小数点以后各位的位权，同样可以用底数为16的负n次幂来确定。例如小数点后第1位的位权为$16^{-1}=0.0625$，第2位的位权为$16^{-2}=0.0039063$，第3位的位权为$16^{-3}=0.0002441$等等，任意一个十六进制的小数例如0.4AC9H其对应于十进制的数值则为

$$0.4AC9H=(4\times16^{-1})+(10\times16^{-2})+(12\times16^{-3})+(9\times16^{-4})$$
$$=0.2921295$$

可见任意一个十六进制的数，都可以按位权展开为

表1-1 三种数制的整数对照表

十进制	十六进制	二进制	十进制	十六进制	二进制
0	0H	0B	12	0CH	1100B
1	1H	1B	13	0DH	1101B
2	2H	10B	14	0EH	1110B
3	3H	11B	15	0FH	1111B
4	4H	100B	16	10H	10000B
5	5H	101B	17	11H	10001B
6	6H	110B	18	12H	10010B
7	7H	111B	19	13H	10011B
8	8H	1000B	20	14H	10100B
9	9H	1001B	⋮	⋮	⋮
10	0AH	1010B			
11	0BH	1011B			

$$H=H_{n-1}\times16^{n-1}+H_{n-2}\times16^{n-2}+\cdots\cdots+H_1\times16^1+H_0\times16^0$$
$$+H_{-1}\times16^{-1}+H_{-2}\times16^{-2}+\cdots\cdots+H_{-m}\times16^{-m}$$

式中 n表示小数点左边的位数（其中第1位定为0位），m表示小数点右边的位数。m和n都是正整数。

表 1-2 三种数制的小数对照表

十进制	十六进制	二进制	十进制	十六进制	二进制
0.00390625	0.01H	0.00000001B	0.046875	0.0CH	0.00001100B
0.0078125	0.02H	0.00000010B	0.05078125	0.0DH	0.00001101B
0.01171875	0.03H	0.00000011B	0.0546875	0.0EH	0.00001110B
0.015625	0.04H	0.00000100B	0.05859375	0.0FH	0.00001111B
0.01953125	0.05H	0.00000101B	0.0625	0.1H	0.0001B
0.0234375	0.06H	0.00000110B	0.125	0.2H	0.0010B
0.02734375	0.07H	0.00000111B	0.1875	0.3H	0.0011B
0.03125	0.08H	0.00001000B	0.25	0.4H	0.0100B
0.03515625	0.09H	0.00001001B	⋮	⋮	⋮
0.0390625	0.0AH	0.00001010B			
0.04296875	0.0BH	0.00001011B			

§1-2 不同进位计数制之间的互换

要把二进制或者十六进制数转换为十进制数，可以利用上一节介绍的位权展开式。下面主要介绍如何把十进制数转换为二进制或者十六进制数。

一、十进制数转换为二进制数

要把十进制数转换为二进制数，可以采用逐次除 2 法。记下每一次相除后的余数，并按反序排列（指第一个余数作为最低位的排列方法），就可以得出相应的二进制数。以十进制数25为例，逐次除2法列式如下：

25÷2=12……余1（最低位）

12÷2=6……余0

6÷2=3……余0

3÷2=1……余1

1÷2=0……余1

转换为二进制等于11001B。

要把十进制小数转换为二进制小数，则用逐次乘2法，将小数部分一次又一次地乘2，取其乘积的整数个位（即所谓溢出数）按正序排列（指第一个溢出数作为最高位的排列方法），就可以得相应的二进制小数。以十进制小数0.6875为例，逐次乘2法列式如下：

0.6875×2=1.375……溢出数1（最高位）

0.375×2=0.75……溢出数0

0.75×2=1.5……溢出数1

0.5×2=1……溢出数1

转换为二进制小数等于0.1011B

二、十进制数转换为十六进制数

要把十进制数转换为十六进制数，与十进制转换为二进制相类似，即整数部分采用逐次除16法，再将余数按反序排列。小数部分采用逐次乘16法，再将溢出数按正序排列。例如十进制数13562需转换为十六进制，逐次除16法列式如下：

13562÷16＝847……余数10（或写成0AH）

847÷16＝52……余数15（或写成0FH）

52÷16＝3 ……余数4（或写成4H）

3÷16＝0 ……余数3（或写成3H）

转换为十六进制数等于34FAH。

要把十进制小数0.359375转换为十六进制，可用逐次乘16法

0.359375×16＝5.75……溢出数5（或5H）

0.75×16＝12.0 ……溢出数12（或0CH）

转换为十六进制小数等于0.5CH。

三、二进制数与十六进制数的互换

二进制数与十六进制数之间的互换比较简单，1位十六进制数对应于4位二进制数，互换时只要以4位二进制数为一个组，每一个组作相应变换即可。但要注意，分组时要以小数点为准，整数从小数点开始从右向左每4位划为一组，而小数则从小数点开始从左向右，也是4位划为一组。如果最后一个组不足4位，可以加零补齐，以 1100110101.11B 为例，其分组方法如下：

$$\underbrace{0110}_{6}\ \underbrace{0110}_{6}\ \underbrace{1101}_{D}\ .\ \underbrace{1100}_{C}$$

转换为十六进制数等于66D.CH。

要把十六进制数 0AA3.3H 化为二进制数，可将每一个位的十六进制数。转换为对应二进制数即

$$\underbrace{A}_{1010}\ \underbrace{A}_{1010}\ \underbrace{3}_{0011}\ .\ \underbrace{3}_{0011}$$

转换为二进制数等于101010100011.0011B。

§1-3 带符号的二进制数

在数学运算中，表示一个数的正负，可以在数的前面冠以正负号。但在计算机内部，正负号要用0和1两个数码代替。例如在字长为8位的二进制数中，把它的最高位定为符号位，最高位为0表示该数为正，最高位为1表示该数为负。例如数 01101111B 等于 +1101111B，而数10101111B则等于－0101111B，这种把最高位定为符号位的二进制数称之为带符号的二进制数。

同样一个二进制数，例如数11001100B，如果是一个无符号数，它等于十进制的 204；如果是一个带符号的数，则等于十进制的－76。可见，由于二进制有无符号与带符号的区别，故使用前要加以说明。对于带符号的二进制数，表示方法有三种。

一、原码

用原码表示一个带符号的二进制数时，将最高位定为符号位，其余各位为数值，例如

$$x=+1010101B \qquad [x]_{原}=01010101B$$

$$x=-1010101B \qquad [x]_{原}=11010101B$$

式中 $[x]_{原}$表示x的原码，对于数0，其原码可以写成两种形式

$$[+0]_{原}=00000000B$$

$$[-0]_{原}=10000000B$$

原码表示方法简单，容易阅读，但在运算时要把符号和数值分开，例如做加法运算，首先要判断两个数的符号，如果符号相同，可以直接相加，如果符号相异，就不能相加，相反要做减法，将绝对值较大的数减去绝对值较小的数，其差值就是两数之和。和的符号与绝对值较大的数的符号一致。可见原码本身虽然比较易写，但运算过程却比较复杂，所以计算机内部不用原码。

二、反码

反码也是带符号二进制数的一种表示方法，它同样要规定最高位为符号位。对于正数，其余各位表示数值。对于负数，其余各位应将1换成0，0换成1，通常叫做取反。例如

$$x=+1010101B \qquad [x]_{反}=01010101B$$

$$x=-1010101B \qquad [x]_{反}=10101010B$$

式中 $[x]_{反}$表示为数x的反码，对于数0，其反码同样也有两种形式。

$$[+0]_{反}=00000000B$$

$$[-0]_{反}=11111111B$$

三、补码

补码仍然把最高位规定为符号位。对于正数，其余各位表示数值。因此，一个正数的原码、反码和补码实际上完全相同。对于负数，则其余各位等于数值取反后再加以1，或者称之为取反加1，例如：

$$x=+1010101B \qquad [x]_{补}=01010101B$$

$$x=-1010101B \qquad [x]_{补}=10101011B$$

式中 $[x]_{补}$表示为数x的补码。

补码的概念与某个具体计数器的最大记数容量有关。例如一个7位的二进制计数器，其最大容量为128，当计数器计至128时，最高位溢出，这是因为7位计数器最多只能记到128的缘故。如下式方框中的数就是被丢弃的数。

$$0=0000000B$$

$$128=\boxed{1}0000000B$$

我们将这个被丢弃的数（7位计数器等于128）称为模，用符号mod表示。在一个7位计数器中，由于最高位进位被丢弃，从留下来的示数来看，0和128相同，1和129相同…或者说两者等价，推而广之，对于模等于M的计数器来讲，a与$a+M$等价，即

$$a+M=a \pmod{M}$$

再将这个概念推广到负数领域，假如$a=-2$，M为128，代入上式

$$a+M=(-2)+128=126 \pmod{M}$$

可见－2和126等价，我们把这两个数互称为补码。

为了进一步理解这个概念，可以时钟为例，钟面最大示数为12，时针走到12点等于0点，数12被自动丢弃，可见钟面的模为12，如果把时针向前拨3点，即$a=3$，跟时针向前拨$a+M=3+12=15$点，时针将停在同一位置上，或者说钟面上3与15等价。

若将时针向后倒拨定为负数，那么向后倒拨3点，即$a=-3$，按式$a+M=(-3)+12=9$，即时针向前拨9点与向后拨3点等价。或者说9是－3的补码（条件为mod $M=12$）。

因此要求一个数的补码，可以利用公式$a=a+M$。也可以用上面说的用取反加1的办法（不包括符号位）。表1-3是8位带符号二进制数的原码、反码和补码对照表。

但要注意不要把求一个负数的补码跟以后介绍的变补概念相混淆。求负数补码用取反加1(不包括符号位)。变补虽然也是取反加1，但包括符号位。

变补是指已知一个数y的补码为$[y]_补$，求其负数补码$[-y]_补$的过程，例如

已知 $y=+2$ $[y]_补=[+2]_补=00000010B$

$[-y]_补=[-(+2)]_补=11111110B$

可见要求$[-y]_补$可以将$[y]_补$连同符号位取反加1，同样

已知 $y=-2$ $[y]_补=[-2]_补=11111110B$

$[-y]_补=[-(-2)]_补=00000010B$

式中00000001是11111110变补的结果，00000001不是11111110的补码。

表1-3 二进制数原码、反码和补码对照表

十进制数	二进制数	原 码	反 码	补 码
+0	+0000000	00000000	00000000	00000000
+1	+0000001	00000001	00000001	00000001
+2	+0000010	00000010	00000010	00000010
⋮	⋮	⋮	⋮	⋮
+126	+1111110	01111110	01111110	01111110
+127	+1111111	01111111	01111111	01111111
−0	−0000000	10000000	11111111	00000000
−1	−0000001	10000001	11111110	11111111
−2	−0000010	10000010	11111101	11111110
⋮	⋮	⋮	⋮	⋮
−126	−1111110	11111110	10000001	10000010
−127	−1111111	11111111	10000000	10000001
−128	−10000000	无法表示	无法表示	10000000

§1-4 二进制数的运算

一、加法运算

设有x、y两个数需要相加，若用原码运算，首先要判断x、y两个数的正负号，若是异号，还要判断两个数哪一个大，这样就增加了运算过程的复杂性。所以，计算机对二进制数进行加法运算时，一律采用补码。

采用补码运算时，先将两个数的补码输入到计算机，计算机无需判断符号即可直接相

加。但是因为$[x]_补+[y]_补=[x+y]_补$，所以相加结果是x、y和的补码，如要求其真值，还要把补码换算成真值（正数补码与真值相同）。在相加过程中，虽然符号位参加运算，但并不影响结果的准确性，这可以从下面的例子中看出：

例1-1 求下列各组中x、y之和。

$x=+0011001B$	$[x]_补=00011001B$	$[+25]_补$
$y=+0010101B$	$+[y]_补=00010101B$	$+[+21]_补$
	$[x+y]_补=00101110B$	$[+46]_补$
$x=-0011001B$	$[x]_补=11100111B$	$[-25]_补$
$y=-0010101B$	$+[y]_补=11101011B$	$+[-21]_补$
	$[x+y]_补=\boxed{1}11010010B$	$[-46]_补$
$x=+0011001B$	$[x]_补=00011001B$	$[+25]_补$
$y=-0010101B$	$+[y]_补=11101011B$	$+[-21]_补$
	$[x+y]_补=\boxed{1}00000100B$	$[+4]_补$
$x=-0011001B$	$[x]_补=11100111B$	$[-25]_补$
$y=+0010101B$	$[y]_补=00010101B$	$+[+21]_补$
	$[x+y]_补=11111100B$	$[-4]_补$

在上例第2组和第3组中，最高位均发生进位（加方框数字表示是进位数字），只是因为计算机字长只允许8位，进位无处可进，就产生了自动丢弃的现象，计算机在运算中这种自动丢弃是允许的，它并不影响结果的准确性。

但要注意，有的进位是不允许的，例如和数超过7位二进制数所允许的表示范围（即超出$-128\sim+127$）而产生的进位，这种进位就不允许，我们称这种进位为溢出。溢出与进位的自然丢弃是两个不同的概念，如果第7位和第8位同时产生进位，即所谓双进位，则这种进位允许自然丢弃，且不影响结果的准确性，如果只有第7位或者只有第8位产生进位，则是溢出，溢出表示超出计算机字长所允许的表示范围，运算结果必然错误。

例1-2 求下列两组数x、y之和。

$x=+1100100B$	$[x]_补=01100100B$	$[+100]_补$
$y=+1000011B$	$+[y]_补=01000011B$	$+[+67]_补$
	$[x+y]_补=10100111B$	$[-89]_补$
$x=-1111000B$	$[x]_补=10001100B$	$[-120]_补$
$y=-1011000B$	$+[y]_补=11101000B$	$+[-24]_补$
	$[x+y]_补=\boxed{1}01110000B$	$[+112]_补$

例中第1组的第7位产生进位，但第8位并没有进位。第2组中第8位产生进位，但第7位没有进位，所以都属于溢出，其运算结果为-89和$+112$显然也是错误的。

还应注意，溢出主要是指带符号二进制数进行加法运算时可能产生的一种结果，对于无符号数，不论是单进位还是双进位，进位都是和的一个部分，不采用溢出这个概念。

二、减法运算

设 x、y 为两个带符号的二进制数，并用补码表示。计算机对 x、y 进行减法运算时，总是把减法运算转换为加法运算，因为

$$[x]_补-[y]_补=[x]_补+[-y]_补=[x-y]_补$$

可见两个补码相减，也可以把减数 y 转换为 $-y$ 补码然后相加，其结果恰等于差的补码。如何把 y 的补码 $[y]_补$ 转换为 $-y$ 的补码 $[-y]_补$ 呢？这个转换称为变补，转换方法是将 $[y]_补$ 连同符号位一起取反加 1 。

从下面例子可以看出，两个带符号数的补码相减，跟对减数变补后进行相加，其结果完全一样。

例1-3　求下列各组 x、y 之差。

1)　$x=+0000111B$　　$y=+0000011B$

$[x]_补=00000111B$　　$[x]_补=00000111B$

$-[y]_补=00000011B$　　$+[-y]_补=11111101B$

$[x-y]_补=00000100B$　　$[x-y]_补=\boxed{1}00000100B$

检验

$[+7]_补$　　$[+7]_补$

$-[+3]_补$　　$+[-3]_补$

$[+4]_补$　　$[+4]_补$

2)　$x=-0000111B$　　$y=-0000011B$

$[x]_补=11111001B$　　$[x]_补=11111001B$

$-[y]_补=11111101B$　　$+[-y]_补=00000011B$

$[x-y]_补=\boxed{1}11111100B$　　$[x-y]_补=11111100B$

检验

$[-7]_补$　　$[-7]_补$

$-[-3]_补$　　$+[+3]_补$

$[-4]_补$　　$[-4]_补$

3)　$x=+0000111B$　　$y=-0000011B$

$[x]_补=00000111B$　　$[x]_补=00000111B$

$-[y]_补=11111101B$　　$+[-y]_补=00000011B$

$[x-y]_补=\boxed{1}00001010B$　　$[x-y]_补=00001010B$

检验

$[+7]_补$　　$[+7]_补$

$-[-3]_补$　　$+[+3]_补$

$[+10]_补$　　$[+10]_补$

4)　$x=-0000111B$　　$y=+0000011B$

$[x]_补=11111001B$　　$[x]_补=11111001B$

$-[y]_补=00000011B$　　$+[-y]_补=11111101B$

$[x-y]_补=11110110B$　　$[x-y]_补=\boxed{1}11110110B$

检验

$$\begin{array}{r} [-7]_{补} \\ -[+3]_{补} \\ \hline [-10]_{补} \end{array} \qquad \begin{array}{r} [-7]_{补} \\ +[-3]_{补} \\ \hline [-10]_{补} \end{array}$$

如果要将两个无符号数进行相减，并且把两个无符号数直接输入到计算机，这时计算机本身无法判别所输入的数到底是无符号数还是带符号数的补码，总是将它们当成补码运算，但运算结果并不影响得数的准确性。

例如向计算机输入两个无符号数

x=10011111B

y=00001000B

要求计算机求出 $x-y$ 的值，计算机将 x、y 两数看成是带符号数的补码，其结果为

s=10010111B

如果把 x、y 两数理解为带符号数的补码，则 x 的真值为 −97，y 的真值为 +8，运算结果为$[x-y]_{补}=s$=10010111B，其真值为$x-y$=11101001B=−105，结果正确。如果把x、y 两数理解为无符号数，则 x=158，y=8 运算结果 s=10010111=150，结果仍然正确。

当然，对于无符号数来讲，不允许将一个较小的数减去一个较大的数，因为较小的数减去较大的数结果是一个负数，而无符号数的前提是没有符号，显然也就没有负数，如果这样做，就会得出错误的结果。

三、乘法运算

若有 x、y 两个数，求其乘积，通常是从低位开始，逐个与被乘数相乘，得出每次相乘的部分积，然后再将部分积相加。

以 x=10111101B，y=10101010B 相乘为例运算过程如下：

$$\begin{array}{r} 10111101 \\ \times\ 10101010 \\ \hline 00000000 \\ 10111101 \\ 00000000 \\ 10111101 \\ 00000000 \\ 10111101 \\ 00000000 \\ 10111101 \\ \hline 1111011000010 \end{array}$$

从式中可以看出，若乘数为 1，部分积等于将被乘数重抄一遍，只是其最低位要与相应乘数的位对齐。或者说，被乘数与第 n 位乘数相乘时，其部分积等于被乘数左移 $n-1$ 位。若乘数为 0，则部分积为 0，所有乘数位都乘过之后，将部分积相加或逐个累加，便得到最

后乘积。

由于有的计算机指令系统中没有乘法指令，求乘积可根据上面乘法步骤，把乘法运算分解为移位和累加运算。

例如上述例子，如分解为移位和累加其过程如下：

被乘数左移 1 位（乘第 2 位）　　101111010

被乘数左移 3 位（乘第 4 位）＋10111101000

相加得部分积　　11101100010

被乘数左移 5 位　　＋1011110100000

相加得部分积　　1111100000010

被乘数左移 7 位　　＋101111010000000

积　　111110110000010

也可以采用右移法，右移法是从乘数最高位开始计算，每乘 1 位，部分积右移一位，至全部乘完，将部分积相加，即可得出最后积，运算过程如下：

10111101

×10101010

10111101

00000000

10111101

00000000

10111101

00000000

10111101

00000000

111110110000010

右移法同样可以把乘法运算分解成右移和累加两种运算。

四、除法运算

二进制数的除法运算与十进制相类似，即从被除数的最高位开始，先取出一位作为余数，检查它的值是否超过除数。如小于除数，则记商为 0，并将被除数下一位移到余数上，继续与除数相比较。如大于除数，则记商为 1，从余数中减去除数，减后继续作为余数，并移入被除数下一位，继续比较，直至全部除尽。以下式为例。

00011101

00000101 | 10010001

0 ……取第 1 位小于除数记商为 0

10……取第 2 位仍小于除数记商为 0

00

```
100 ·····取第3位仍小于除数记商为0
000
---------
1000·····取第4位大于除数记商为1
0101      减去除数
---------
01000 ····取第5位
00101     减去除数
---------
000110····取第6位
000101
---------
0000010   取第7位
0000000
---------
00000101  取第8位
00000101
---------
0
```

如果计算机内部没有除法指令，遇到两数相除，就可以按以上步骤，分解为移位和相减。

§1-5 BCD码和文字符号代码

一、BCD码

BCD码（Binary Coded Decimal）又称为二—十进制码，它是用二进制数的形式表示十进制数的一种编码。从外表看来，它貌似二进制数，实际上，它不是二进制数，只是把十进制数写成计算机所能接受的形式而已。

BCD码以4位二进制为一组，每组代表一个十进制数。当BCD码与十进制数互换时，就可以按4位为1组的原则，逐组进行互换。

例1-4 将84.7转换为BCD码。

$$\underbrace{8}_{1000}\ \underbrace{4}_{0100}\cdot\underbrace{7}_{0111}$$

例1-5 将BCD码10010100.01110010转换为十进制数。

$$\underbrace{1001}_{9}\ \underbrace{0100}_{4}\ .\ \underbrace{0111}_{7}\ \underbrace{0010}_{2}$$

要把一个二进制数转换为BCD码，通常先将其转换为十进制，然后再转换为BCD码。同样，从BCD码反转为二进制数，也是先转换为十进制，再转换为二进制数。

例1-6 将二进制数1101100100000 1B转换为BCD码。

1101100100000 1B=13889

1　3　8　8　9

0001 0011 1000 1000 1001

例1-7　将 BCD 码 1000010001100101 转换为二进制数。

1000 0100 0110 0101

8　4　6　5

8465＝0010000100010001B

BCD 码的低位与高位之间是根据逢10进 1 的原则进位。这一点与二进制显然不同，二进制第 4 位向第 5 位进位则为逢16进 1。因此两个 BCD 码相加时，可以当成二进制相加，但必须将其 4 个位划成 1 组，逐组相加，若相加后其和大于 9，则应进行加 6 修正。

例1-8　将 BCD 码 01011000 与 01101001 相加。

```
0101     1000……被加数
0110     1001……加数
-----------------
 100     0001
    1 ←
    进位
```

由于第 1 组相加和为17，第 2 组相加和为12，都超过 9，必须进行加 6 修正，修正过程为：

```
1100     0001……上式求得的和
0110     0110……加 6 修正
-----------------
0001 0010     0111
   1 ←
   进位
```

检验

```
 58
+69
----
127
```

两个 BCD 码相减时，若有借位就要进行减 6 修正，无借位时可以不要修正。

例1-9　将 BCD 码10000101 与 00101000 相减。

```
1000     0101……被减数
-0010    1000……减数
-----------------
0101     1101
   |→
   借位
```

由于第 1 组相减时有借位，应进行减 6 修正，第二组由于无借位，不必修正，即

0101　　1101……上式求得之差

－　　0110……减 6 修正

0101　　0111

二、文字符号代码

计算机只能辨认 0 和 1 两个数码，所以在计算机内部只能使用二进制数。但是在信息处理中，除了使用大量的数码外。还要处理许多文字符号，这就要把文字符号用一串二进制数码来代表，以便能在计算机内部进行存贮与传送，这就是所谓文字符号数码化问题。

目前，计算机使用的文字符号都采用美国信息交换标准代码，简称为 ASCII 码（American Standard Code for Information Interchange）。

ASCII码包括英文大小写字母52个，0～1数字符10个，常用书写符号，如：$ % 和常用的数学运算符号，如＋－<>共31个。以及一些控制符号共计128个。具体可参看附录。

常用的 ASCII 码为 7 位二进制数码，余下最高位作为奇偶校验。例如字符 A 的编码为01000001或写成十六进制为 41H。数码 5 的编码为00110101或写成十六进制的35H。

练 习 题

1. 将下列十进制数转换为十六进制和二进制数。

① 14.375　　② 132.96875

③ 127.65625　　④ 256.875

2. 将下列十六进制数转换为二进制和十进制数。

① 0A85H　　② 2FF6H

③ 4200H　　④ 2FFFH

3. 将下列二进制带符号数，分别用原码、反码和补码表示。

① ＋1111000　　② ＋1111111

③ －1000000　　④ －1111111

4. 下列各数为带符号数的补码，试求出它们的真值

① 03H　　② 74H

③ F4H　　④ CDH

5. 将下列 BCD 码转换为十进制数。

① 0010000110101001　　② 1000100100110001

6. 将下列十进制数转换为 BCD 码

① 97.45　　② 1145.45

7. 有两个数 A，B，已知 A=10010011B，B=00001111B，求它们的和，如果 A、B 是无符号数，和为多少；如果是带符号的数，和又为多少。