

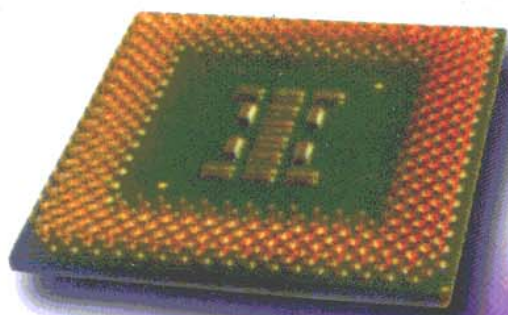


中等专业学校 教材
电子信息类

汇编语言

程序设计教程

翟社平 编著



3-43



西安电子科技大学出版社

<http://www.xduph.com>

中等专业学校
电子信息类教材

汇编语言程序设计教程

翟社平 编著

西安电子科技大学出版社

内 容 简 介

本书以 8086/8088 汇编语言为基础,详细介绍了 8086/8088 的指令系统,程序设计的概念、方法和技巧,模块化程序设计及中断程序设计的方法。另外,对 80286/80386/80486/Pentium CPU 的结构和指令系统做了一些介绍,使读者对于计算机的发展及汇编语言的使用有一个全面系统的了解。

本书在编写过程中力求做到体系完整,结构合理,内容丰富,通俗易懂,由浅入深。为便于读者学习,每章章首均有本章内容提要,每章章末均附有一定数量的习题。

本书可作为中等专业学校教材,也可供大专院校选用,亦可作为全国计算机等级考试三 A 的复习参考书。

图书在版编目(CIP)数据

汇编语言程序设计教程/翟社平编著. —西安:西安电子科技大学出版社,2000.7

中等专业学校电子信息类教材

ISBN 7-5606-0888-4

I. 汇… II. 翟… III. 汇编语言-程序设计-专业学校-教材 IV. TP312

中国版本图书馆 CIP 数据核字(2000)第 33996 号

责任编辑 戚文艳

出版发行 西安电子科技大学出版社(西安市太白南路 2 号)

电 话 (029)8227828 邮 编 710071

<http://www.xduph.com> E-mail: xdupfxb@pub.xaonline.com

经 销 新华书店

印 刷 陕西画报社印刷厂

版 次 2000 年 7 月第 1 版 2001 年 1 月第 2 次印刷

开 本 787 毫米×1092 毫米 1/16 印张 17

字 数 405 千字

印 数 4 001~10 000 册

定 价 17.00 元

ISBN 7-5606-0888-4/TP·0471

*** 如有印装问题可调换 ***

本书封面贴有西安电子科技大学出版社的激光防伪标志,无标志者不得销售。

前 言

汇编语言面向机器硬件,使用户能够对寄存器、标志位和存储单元进行直接处理。利用汇编语言,可以编写出在“时间”和“空间”方面都具有很高效率的程序。如果掌握了汇编语言,不仅有助于理解计算机的基本工作原理,而且能够充分发挥计算机的潜力。因此,对于计算机专业技术人员及高水平的程序设计人员来说,汇编语言始终是必须掌握的计算机基础知识。

“汇编语言程序设计”是大中专院校计算机专业学生必修的核心专业课程之一,是计算机原理、操作系统等其他核心课程的先期必修课。该课程也是国家计算机等级考试三 A 主考课程之一。

本书以 8086/8088 汇编语言为基础,详细介绍了 8086/8088 的指令系统,程序设计的概念、方法和技巧,模块化程序设计及中断程序设计的方法。另外,对于 80286/80386/80486/Pentium CPU 的结构和指令系统做了一些介绍,使读者对于计算机的发展及汇编语言的使用有一个全面系统的了解。

本书共分 8 章。第 1 章讲解了一些汇编语言使用的计算机基础知识,对此已有了解的读者可以跳过此章。第 2 章介绍了汇编语言的寻址方式、8086/8088 指令系统及指令的编码方法,并给出指令的使用示例。第 3 章介绍了汇编语言的伪指令、汇编程序的基本格式结构以及汇编程序的上机过程。第 4 章介绍了汇编语言程序设计基本方法、子程序设计、DOS 系统功能调用方法。第 5 章介绍了输入/输出和中断技术以及 ROM BIOS 的中断调用。第 6 章通过大量的实例介绍了应用程序设计的基本方法及技巧。第 7 章介绍了汇编语言程序与其他高级语言程序的连接问题。第 8 章介绍了 80286/80386/80486/Pentium CPU 的指令系统及编程结构。

在具体教学安排上,编者建议使用 68~74 学时讲授基本内容(1~7 章),如有更多学时,可以选讲或学生自学第 8 章。

本书的第 1、2、7 章由魏娟丽编写,其余各章由翟社平编写。由于编者水平有限,错误和不妥之处在所难免,恳请读者批评指正。

编 者

2000 年 4 月

第 1 章

基 础 知 识

本章主要介绍学习汇编语言的重要性及计算机的一些基础知识,包括计算机中数和字符的表示、8086/8088 CPU 的编程结构、存储器及堆栈等内容。

1.1 汇编语言概述

1.1.1 为什么要学习和使用汇编语言

计算机程序设计语言大致可分为高级语言、低级语言、机器语言。汇编语言属于低级语言。高级语言(如 BASIC、PASCAL、C 等)具有“易用好学”的特点。因为即使你不懂计算机内部的组成,采用高级语言仍然能够设计出很好的程序来。相对来说,用汇编语言来编写程序要困难一些,因为你不但要熟悉机器的指令系统,还要熟悉其内部结构,特别是中央处理器(CPU)和存储器的结构,既然如此,为什么还要学习汇编语言呢?其原因有以下几点:

(1) 采用高级语言编写的程序,机器是不能直接执行的,需要编译程序或解释程序将它翻译成相应的机器语言程序,机器才能接受。而通过编译或者解释得到的目标代码程序比较冗长,占有存储空间大,执行速度慢,效率低,而汇编语言程序无论是在目标代码长短和程序的执行速度,还是在节省存储空间方面,都要优于高级语言,尤其在实时控制中,常常用汇编语言来编制程序。

(2) 学习和使用汇编语言可以从根本上认识和理解计算机的工作过程。学习汇编语言要熟悉计算机的内部结构,通过汇编程序的编制和运行,可以更清楚地了解计算机是怎样完成各种复杂工作的,以及如何利用计算机的所有硬件特征来直接控制硬件的。

(3) 现在计算机系统某些功能只能采用汇编语言来完成。例如,机器的自检、系统的初始化等。

1.1.2 汇编语言的基本概念

1. 机器语言

机器语言是用二进制编码的机器指令的集合及一组使用机器指令的规则。它是 CPU 能直接识别的唯一语言。只有用机器语言书写的程序, CPU 才能直接执行。用机器语言描述的程序称为目的程序或目标程序。用机器语言编写的程序不易为人们理解、记忆和交

流,而且容易出错。一旦出错,也很难发现和纠错。

2. 汇编语言

为了克服机器语言的缺点,人们采用便于记忆并能描述指令功能的符号来表示指令的操作码。这些符号被称为指令助记符。助记符一般是说明指令功能的英语词汇或词汇的缩写,同时也用符号表示操作数,如 CPU 的寄存器、存储单元等。

汇编指令由指令助记符及操作数构成。汇编语言是汇编指令、伪指令的集合及表示和使用这些指令的一组规则。用汇编语言书写的程序称为汇编语言程序或汇编语言源程序。用汇编语言编写的程序要比用机器语言编写的程序容易理解、调试和维护。

3. 汇编程序

由于 CPU 能直接识别的唯一语言是机器语言,所以用汇编语言编写的源程序必须被翻译成用机器语言表示的目标程序后才能由 CPU 执行。把汇编语言源程序翻译成目标程序的过程称为汇编。完成汇编任务的程序叫做汇编程序。汇编过程如图 1-1 所示。

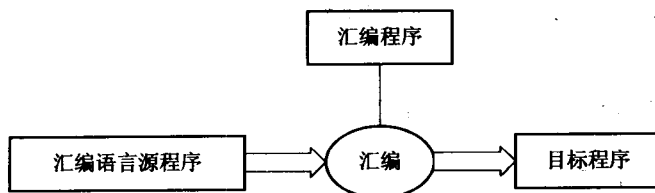


图 1-1 汇编过程示意图

1.2 计算机中数和字符的表示

1.2.1 数制及数制间的转换

1. 二进制计数

- (1) 二进制计数只有两个数码,即 0 和 1。
- (2) 其进位原则是“逢二进一”。
- (3) 二进制基数为 2,位权为 2^k (k 为整数)。

例如:

$$(11011.101)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ = (27.625)_{10}$$

任意一个二进制数 B,可以表示成如下形式:

$$(B)_2 = B_{n-1} \times 2^{n-1} + B_{n-2} \times 2^{n-2} + \cdots + B_1 \times 2^1 + B_0 \times 2^0 + B_{-1} \times 2^{-1} + \cdots + B_{-m+1} \times 2^{-m+1} \\ + B_{-m} \times 2^{-m}$$

式中, B 为数位上的数码,其取值为 0 和 1; n 为整数位个数; m 为小数位个数; 2 为基数, $2^{n-1}, 2^{n-2}, \dots, 2^{-m}$ 为二进制数的位权。

从上面可以看出,把二进制数转化为十进制数非常简单,只要按权展开相加即可。为了与其他进制相区别,常在二进制数后面写一个字母 B,如 10010101B。

因为二进制数只有两个数码 0 和 1，所以二进制数的每一位都可以用任何两个不同稳定状态的物理元件来表示。如电灯的亮和灭，晶体管的截止和导通，脉冲的有无，电位的高低等，只要规定其中一种状态表示“1”，另一种状态表示为“0”，就可以用来表示二进制数了。在计算机内部，采用的就是二进制计数。

2. 八进制计数

(1) 八进制计数有 8 个不同的数码。即 0, 1, 2, 3, 4, 5, 6, 7。

(2) 进位原则为“逢八进一”。

(3) 八进制数基数为 8，位权为 8^k (k 为整数)。

例如：

$$\begin{aligned}(123.24)_8 &= 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 + 2 \times 8^{-1} + 4 \times 8^{-2} \\ &= (83.3125)_{10}\end{aligned}$$

任意一个八进制数 Q ，可以表示成如下形式：

$$\begin{aligned}(Q)_8 &= Q_{n-1} \times 8^{n-1} + Q_{n-2} \times 8^{n-2} + \cdots + Q_1 \times 8^1 + Q_0 \times 8^0 + Q_{-1} \times 8^{-1} + \cdots \\ &\quad + Q_{-m+1} \times 8^{-m+1} + Q_{-m} \times 8^{-m}\end{aligned}$$

式中， Q 为数位上的数码，其取值范围为 $0 \sim 7$ ； n 为整数位个数； m 为小数位个数；8 为基数， 8^{n-1} , 8^{n-2} , $\cdots$, 8^{-m} 是八进制数的位权。

为了与其他进制数值区别。一般在八进制数后面加上 Q ，例如 723.24 Q 。

八进制数也是计算机中常用的一种计数方法，它可以弥补二进制数书写位数过长的不足。

3. 十六进制计数

(1) 十六进制计数有 16 个不同的数码，即 $0 \sim 9$, A, B, C, D, E, F。

(2) 进位原则是“逢十六进一”。

(3) 十六进制基数为 16，位权为 16^k (k 为整数)。

例如：

$$\begin{aligned}(3AB.48)_{16} &= 3 \times 16^2 + A \times 16^1 + B \times 16^0 + 4 \times 16^{-1} + 8 \times 16^{-2} \\ &= (939.28125)_{10}\end{aligned}$$

任意一个十六进制数 H ，可以表示成如下形式：

$$\begin{aligned}(H)_{16} &= H_{n-1} \times 16^{n-1} + H_{n-2} \times 16^{n-2} + \cdots + H_1 \times 16^1 + H_0 \times 16^0 + H_{-1} \times 16^{-1} + \cdots \\ &\quad + H_{-m} \times 16^{-m}\end{aligned}$$

其中， H 为数位上的数码，其取值范围为 $0 \sim 9$, A~F； n 为整数位个数； m 为小数位个数；16 为基数， 16^{n-1} , 16^{n-2} , $\cdots$, 16^{-m} 为十六进制的位权。

为了与其他进制数值区别。一般在十六进制数后面加上 H ，例如 7A.CDH。在汇编语言中，凡是以字母 A~F 打头的十六进制数，都要以“0”开头进行书写，以避免与标识符相混淆，例如十六进制数 B7H 应写成 0B7H。

十六进制数是计算机中常用的一种计数方法，它可以弥补二进制数书写位数过长的不足。

总结以上几种计数制，可将它们的特点概括为：

(1) 每一种计数制都有一个固定的基数 J (J 为大于 1 的整数)，它的每一位可取 J 个不

同的数值。

(2) 每一种计数制都有自己的位权, 并且遵循“逢 J 进一”的原则。

表 1-1 中列出了几种常用进位计数制的表示方法。

表 1-1 常用进位计数制的表示方法

十进制	十六进制	二进制	十进制	十六进制	二进制
0	0	0000	9	9	1001
1	1	0001	10	A	1010
2	2	0010	11	B	1011
3	3	0011	12	C	1100
4	4	0100	13	D	1101
5	5	0101	14	E	1110
6	6	0110	15	F	1111
7	7	0111	16	10	10000
8	8	1000			

4. 不同进位计数制间的转换

计算机内部处理的数据都是二进制数, 但编程时往往会用到其他一些进制数, 这就必然存在各种进制数之间的转换。不同进制数之间的转换, 实质上是基数之间的转换, 一般转换原则是: 如果两个有理数相等, 则两数的整数部分和小数部分一定分别相等。因此, 各数制之间进行转换时, 通常对整数部分和小数部分分别进行转换。

1) 非十进制数转换为十进制数

通过前面的例子可以看出, 非十进制数转换为十进制数的方法为, 把各个二进制数 (或八进制数, 或十六进制数) 写成 2 (或 8, 或 16) 的各次幂之和的形式, 然后计算其结果。

【例 1-1】 把下列二进制数转换成十进制数:

$$(110101)_2 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ = (53)_{10}$$

$$(1101.101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ = (13.625)_{10}$$

【例 1-2】 把下列八进制数转换成十进制数:

$$(305)_8 = 3 \times 8^2 + 0 \times 8^1 + 5 \times 8^0 = (197)_{10}$$

$$(456.124)_8 = 4 \times 8^2 + 5 \times 8^1 + 6 \times 8^0 + 1 \times 8^{-1} + 2 \times 8^{-2} + 4 \times 8^{-3} \\ = (302.1640625)_{10}$$

2) 十进制数转换成非十进制数

(1) 十进制数转换成二进制数。把十进制数转换为二进制数的方法是: 整数部分转换用“除 2 取余法”, 小数部分转换用“乘 2 取整法”。

【例 1-3】 将十进制数 $(97.625)_{10}$ 转换成二进制数。

整数部分 97 转换如下:

2	97	余数	
2	48	1	二进制数低位
2	24	0	
2	12	0	
2	6	0	
2	3	0	
2	1	1	
	0	1	二进制数高位

小数部分 0.625 转换如下

		0.625
	×	2
二进制小数首位	1.....	1.250
		0.250
	×	2
	0.....	0.500
		0.500
	×	2
二进制小数末位	1.....	1.000
		0.000.....为零转换结束

即

$$(97.625)_{10} = (1100001.101)_2$$

在以上例子中, 小数部分经过有限次乘 2 取整过程即告结束。但有些情况下可能转换是无限的, 这就要根据转换的精度要求截止。对于八进制和十六进制也有同样的情况。

(2) 十进制数转换为八进制数。十进制数转换为八进制数的方法是: 整数部分采用“除 8 取余法”, 小数部分采用“乘 8 取整法”。

【例 1-4】 将十进制数 $(1725.6875)_{10}$ 转换成八进制数。

整数部分 1725 转换如下:

8	1725	余数	
8	215	5	八进制数低位
8	26	7	
8	3	2	
	0	3	八进制数高位

小数部分 0.6875 转换如下:

	0.6875
	× 8
八进制小数首位	5..... 5.5000
	0.5000
	× 8
八进制小数末位	4..... 4.0000
	0.0000..... 为零转换结束

即 $(1725.6875)_{10} = (3275.54)_8$

(3) 十进制数转换成十六进制数。将十进制数转换成十六进制数的方法是: 整数部分转换采用“除 16 取余法”, 小数部分转换采用“乘 16 取整法”。

【例 1-5】 将十进制数 $(12\ 345.671\ 875)_{10}$ 转换成十六进制数。

整数部分 12 345 转换过程如下:

16	12345	余数	十六进制数低位
16	771.....	9	
16	48.....	3	
16	3.....	0	
	0.....	3	十六进制数高位

小数部分 0.671875 转换过程如下:

	0.671875
	× 16
十六进制小数首位	A..... 10.750000
	0.750000
	× 16
十六进制小数末位	C..... 12.000000
	0.000000..... 为零转换结束

即 $(12\ 345.671\ 875)_{10} = (3039.AC)_{16}$

3) 非十进制数之间的相互转换

由于 1 位八进制数相当于 3 位二进制数, 只需以小数点为界, 向左或向右每 1 位八进制数用相应 3 位二进制数取代即可; 反之, 二进制数转换成八进制数, 只是上述方法的逆过程, 即以小数点为界, 向左或向右每 3 位二进制数用相应 1 位八进制数取代即可, 如果不足 3 位, 可用零补足。

【例 1-6】 将八进制数 $(714.413)_8$ 转换成二进制数。

7	1	4	.	4	1	3
111	001	100	.	100	001	011

即 $(714.413)_8 = (111001100.100001011)_2$

【例 1-7】将二进制数 $(1110011101.01)_2$ 转换成八进制数。

$$\begin{array}{ccccccc} 001 & 110 & 011 & 101 & . & 010 \\ 1 & 6 & 3 & 5 & . & 2 \end{array}$$

即

$$(1110011101.01)_2 = (1635.2)_8$$

由于 1 位十六进制数相当于 4 位二进制数，因此要将十六进制数转换成相应二进制数，只需以小数点为界，向右每 1 位十六进制数用相应的 4 位二进制数取代即可。反之，要将一个二进制数转换成相应的十六进制数，只是上述过程的逆过程，即将二进制数以小数点为界分成左右两部分，向左或向右每 4 位二进制数用相应 1 位十六进制数取代即可，如果不足 4 位，则以零补足。

【例 1-8】将十六进制数 $(9ACD.6B)_{16}$ 转换成相应的二进制数。

$$\begin{array}{ccccccc} 9 & A & C & D & . & 6 & B \\ 1001 & 1010 & 1100 & 1101 & . & 0110 & 1011 \end{array}$$

即

$$(9ACD.6B)_{16} = (1001101011001101.01101011)_2$$

【例 1-9】将二进制数 $(101011001.111)_2$ 转换成相应的十六进制数。

$$\begin{array}{ccccccc} 0001 & 0101 & 1001 & . & 1110 \\ 1 & 5 & 9 & . & E \end{array}$$

即

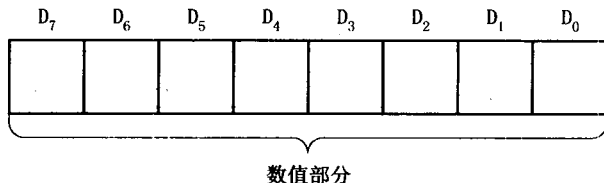
$$(101011001.111)_2 = (159.E)_{16}$$

1.2.2 数和字符在计算机中的表示

1. 计算机中数的表示

1) 真值与机器数

计算机中的数分为有符号数和无符号数。对于无符号数来说，所有二进制数位都是该数的一部分。例如，8 位无符号数的形式如下：



它能表示的最小无符号数是 00000000B，即 0；最大无符号数是 11111111B，即 255D。

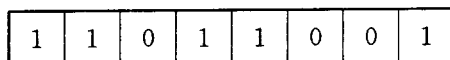
对于有符号数，在计算机中只能用数字化的信息表示数的正负。人们规定用“0”表示正号，用“1”表示负号。例如，在机器中用 8 位二进制数表示 +91，其格式如下：

0	1	0	1	1	0	1	1
---	---	---	---	---	---	---	---



符号位为 0，表示正

用 8 位二进制数表示 -89，其格式如下：



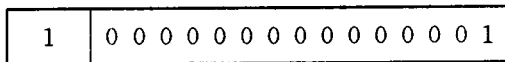
符号位为 1，表示负

在计算机内部，数字和符号都用二进制代码表示，两者合在一起构成数的机内表示形式，称为机器数。而它真正表示的数值称为这个机器数的真值。

2) 定点数和浮点数

(1) 机器数的表示受设备限制。计算机是以字为单位进行数据的处理、存储和传递的。所以运算器中的加法器，累加器以及其他一些寄存器，都选择与字长相等的位数。字长一定，则计算机所能表示数的范围也就确定了。例如，使用 8 位字长的计算机，它可以表示无符号整数的范围在 0~255。如果运算数值超出机器数所能表示的范围，机器就会停止运算进行处理。这种现象称为溢出。

(2) 定点数。计算机中的数，既有整数，也有小数。如何确定小数点的位置呢？通常有两种约定：一种是规定小数点位置固定不变，这时的机器数称为定点数；另一种是小数点位置可以浮动的，这样的机器数称为浮点数。对于定点数，小数点位置可以固定在符号位之后，这样的机器表示的全是定点小数。例如，假定机器字长为 16 位，符号位占 1 位，数值占有 15 位，于是机器数表示为：



•

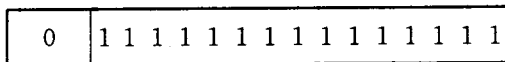


符号位 小数点

数值部分

其相当于十进制数为 -2^{-15} 。

小数点位置固定在数的最后，则该机器表示的全是定点整数。例如，假设机器字长为 16 位，符号位占有 1 位，数值部分占 15 位，下面表示的机器数相当于十进制数为 +32 767：



•



符号位

数值部分

小数点

定点表示法表示的数值范围有限，为了扩大定点数的表示范围，可以采用多个字节来表示一个定点数，例如，采用 4 字节或 8 字节来表示。

(3) 浮点数。浮点数表示法就是小数点在数中的位置是浮动的。由于定点数表示的范围较窄，不能满足实际问题的需要，因此要采用浮点表示法。在同样字长情况下，浮点表示法能表示数的范围扩大了。

浮点表示法包括两部分：一部分是阶码，另一部分是尾数。浮点数在机器中的表示方法如下所示：

阶符	阶码	数符	尾数
----	----	----	----

↑ 小数点

由尾数部分隐含的小数点位置可知,尾数总是小于1的数,它给出该浮点数的有效数字。尾数部分的数符确定该浮点数的正负。阶码总是整数,它确定小数点浮动的位数。若阶符为正,小数点向右移动;若阶符为负,则向左移动。

当浮点数的尾数为零或者阶码为最小值时,机器通常规定,把该数看作0,称为“机器零”。在浮点数的表示和运算中,当一个数的阶码大于机器所能表示的最大阶码时,产生“上溢”;当一个数的阶码小于机器所能表示的最小阶码时,产生“下溢”。

3) 原码,反码和补码

计算机中数的编码通常有原码、反码和补码表示法。

(1) 原码表示法。原码表示法是机器数的一种简单表示法,其符号位用0表示正号,用1表示负号,数值部分用二进制数表示。

【例 1-10】 $X_1 = +11011011$

$X_2 = -11011011$

则

$[X_1]_{\text{原}} = [+11011011]_{\text{原}} = 011011011$

$[X_2]_{\text{原}} = [-11011011]_{\text{原}} = 111011011$

原码表示法对0有两种表示形式:

$[+0]_{\text{原}} = 00000000$

$[-0]_{\text{原}} = 10000000$

(2) 反码表示法。机器数的反码可由原码得到。对于一个正数,该数的反码与其原码相同,对于一个负数,该数的反码是其原码(符号位除外)各位取反得到的。

【例 1-11】 $X_1 = +1010110$

$X_2 = -1001010$

则

$[X_1]_{\text{原}} = 01010110$

$[X_1]_{\text{反}} = [X_1]_{\text{原}} = 01010110$

$[X_2]_{\text{原}} = 11001010$

$[X_2]_{\text{反}} = 10110101$

在反码表示中,对0有两种表示形式:

$[+0]_{\text{反}} = 00000000$

$[-0]_{\text{反}} = 11111111$

(3) 补码表示法。机器数的补码可由原码得到。对于一个正数,则该数的补码与其原码相同;对于一个负数,则该数的补码等于它的原码(除符号位外)各位取反并在末尾加1而得到。

【例 1-12】 $X_1 = +1010110$

$X_2 = -1001010$

则

$$[X_1]_{\text{原}} = 01010110$$

$$[X_1]_{\text{补}} = [X_1]_{\text{原}} = 01010110$$

$$[X_2]_{\text{原}} = 11001010$$

$$[X_2]_{\text{补}} = 10110101 + 1 = 10110110$$

在补码表示方法中, 0 只有一种表示形式: $[+0]_{\text{补}} = [-0]_{\text{补}} = 00000000$ 。

在机器内部, 负数是以其补码的形式存储和参加运算的, 例如, -1 、 -5 和 -23 分别是以其补码即 $11111111(0FFH)$ 、 $11111011B(0FBH)$ 和 $11101001B(0E9H)$ 的形式进行存储的。而运算是以 $8 - 2 = 8 + (-2) = 00001000B + 11111110B = 00000110B = 6$ 的方式进行的。

2. 二进制数的运算

1) 二进制数的算术运算

(1) 二进制数的加法运算。加法运算有如下三条法则:

$$0+0=0$$

$$0+1=1+0=1$$

$$1+1=10$$

↑

向高位进位

(2) 二进制数的减法运算。减法运算有如下三条法则:

$$0-0=1-1=0$$

$$1-0=1$$

$$0-1=1$$

↑

向高位借 1

(3) 二进制数的乘法运算。乘法运算有如下三条法则:

$$0 \times 0 = 0$$

$$0 \times 1 = 1 \times 0 = 0$$

$$1 \times 1 = 1$$

(4) 二进制数的除法运算。二进制运算有如下三条法则:

$$0 \div 0 = 0$$

$$0 \div 1 = 0 (1 \div 0 \text{ 无意义})$$

$$1 \div 1 = 1$$

【例 1-13】 计算 $(110111)_2 \div (1001)_2$ 。

$$\begin{array}{r}
 \text{除数} \rightarrow 1001 \overline{) 110111} \\
 \underline{1001} \\
 1001 \\
 \underline{1001} \\
 1
 \end{array}
 \begin{array}{l}
 110 \leftarrow \text{商} \\
 110111 \leftarrow \text{被除数} \\
 1 \leftarrow \text{余数}
 \end{array}$$

即 $(110111)_2$ 除以 $(1001)_2$, 商为 $(110)_2$, 余数为 $(1)_2$ 。

2) 二进制数的逻辑运算

(1) 逻辑加法(逻辑或运算)。逻辑加法运算用符号+或 $\vee$ 表示, 有如下运算法则:

$$0 \vee 0 = 0 \quad \text{读成“0 或 0 等于 0”}$$

$$0 \vee 1 = 1 \quad \text{读成“0 或 1 等于 1”}$$

$$1 \vee 0 = 1 \quad \text{读成“1 或 0 等于 1”}$$

$$1 \vee 1 = 1 \quad \text{读成“1 或 1 等于 1”}$$

逻辑加法的特点是: 只要参加运算的逻辑变量中有一个为 1, 那么逻辑运算结果就为 1。

【例 1-14】 求二进制数 $(10011011)_2$ 和 $(10100101)_2$ 的逻辑或运算结果。

$$\begin{array}{r}
 10011011 \\
 \vee \quad 10100101 \\
 \hline
 10111111
 \end{array}$$

即

$$10011011 \vee 10100101 = 10111111$$

(2) 逻辑乘法(逻辑与运算)。逻辑乘法运算符用 $\cdot$ 或 $\wedge$ 表示, 有如下运算法则:

$$0 \wedge 0 = 0 \quad \text{读成“0 与 0 等于 0”}$$

$$0 \wedge 1 = 0 \quad \text{读成“0 与 1 等于 0”}$$

$$1 \wedge 0 = 0 \quad \text{读成“1 与 0 等于 0”}$$

$$1 \wedge 1 = 1 \quad \text{读成“1 与 1 等于 1”}$$

逻辑乘法的特点是: 只要参加运算的逻辑变量中有一个为 0, 则运算结果为 0。

【例 1-15】 求二进制数 $(10101101)_2$ 和 $(00101011)_2$ 的逻辑与运算结果。

$$\begin{array}{r}
 10101101 \\
 \wedge \quad 00101011 \\
 \hline
 00101001
 \end{array}$$

即

$$10101101 \wedge 00101011 = 00101001$$

(3) 逻辑否定(逻辑非运算)。逻辑否定的运算符是在逻辑变量的上方加一横线, 例如 $\bar{A}$ 表示对 A 的否定运算。其运算法则如下:

$$\bar{0} = 1 \quad \text{读成“非 0 等于 1”}$$

$$\bar{1} = 0 \quad \text{读成“非 1 等于 0”}$$

(4) 逻辑异或。逻辑异或的运算符通常用“ $\oplus$ ”表示。其运算法则如下:

$$0 \oplus 0 = 0 \quad \text{读成“0 异或 0 等于 0”}$$

$$0 \oplus 1 = 1 \quad \text{读成“0 异或 1 等于 1”}$$

$$1 \oplus 0 = 1 \quad \text{读成“1 异或 0 等于 1”}$$

$1 \oplus 1 = 0$ 读成“1 异或 1 等于 0”

逻辑异或的特点是：只要两个参加运算的逻辑变量的值相同，则异或运算结果为 0。两个逻辑变量值不同时，异或运算结果为 1。

【例 1 - 16】 求二进制数 $(10101101)_2$ 和 $(00101011)_2$ 的逻辑异或运算结果。

$$\begin{array}{r} 10101101 \\ \oplus \quad 00101011 \\ \hline 10000110 \end{array}$$

即 $10101101 \oplus 00101011 = 10000110$

3. 字符编码

在计算机中，数值数据是以原码、反码、补码的形式存放的，而计算机不仅要处理数值数据，还要处理大量的非数值数据，即要对文字和符号进行数字化编码。为了进行信息的交换、表示、处理、存储，人们制定了许多国家标准和国际标准的字符编码。以下是几种常见的编码方式。

1) ASCII 码

ASCII 码即美国信息交换标准码 (American Standard Code for Information Interchange)。这种代码用一个字节 (8 位二进制代码) 来表示一个字符，其中低 7 位为字符的 ASCII 值，最高位一般用作校验位。表 1 - 2 列出了用十六进制数表示的字符的 ASCII 值。

表 1 - 2 十六进制表示的 ASCII 码表

字符	ASCII	字符	ASCII	字符	ASCII	字符	ASCII
NULL	00	4	34	M	4D	f	66
BEL	07	5	35	N	4E	g	67
LF	0A	6	36	O	4F	h	68
FF	0C	7	37	P	50	i	69
CR	0D	8	38	Q	51	j	6A
SP	20	9	39	R	52	k	6B
!	21	:	3A	S	53	l	6C
"	22	;	3B	T	54	m	6D
#	23	<	3C	U	55	n	6E
\$	24	=	3D	V	56	o	6F
%	25	>	3E	W	57	p	70
&	26	?	3F	X	58	q	71
'	27	@	40	Y	59	r	72
(	28	A	41	Z	5A	s	73
)	29	B	42	[	5B	t	74
*	2A	C	43	\	5C	u	75
+	2B	D	44	]	5D	v	76
,	2C	E	45	↑	5E	w	77
-	2D	F	46	←	5F	x	78
.	2E	G	47	'	60	y	79
/	2F	H	48	a	61	z	7A
0	30	I	49	b	62	{	7B
1	31	J	4A	c	63		7C
2	32	K	4B	d	64	}	7D
3	33	L	4C	e	65	~	7E

2) BCD 码

BCD 码是用二进制编码来表示十进制数的，而每位十进制数用 4 位二进制数来表示（即 0~9 用 0000~1001 表示）。BCD 码在存储器中有两种存储方式，即压缩 BCD 码和非压缩 BCD 码。

(1) 非压缩 BCD 码。其存储方式是一个字节只存一个 BCD 码，如图 1-2 所示。

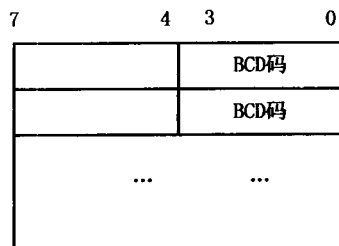


图 1-2 非压缩的 BCD 码

(2) 压缩的 BCD 码。其存储方式是一个字节存放两个 BCD 码，如图 1-3 所示。

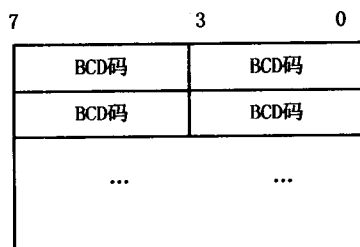


图 1-3 压缩的 BCD 码

例如，十进制数 98 用压缩 BCD 码表示为 1001 1000，用非压缩 BCD 码表示为 00001001 00001000。

4. 数值及字符的存储

数值及字符编码在计算机内部可以以位、字节、字、双字等形式存在。

1) 位(bit)

计算机中最小的一个数据单位是二进制的数位，简称位(bit)。位是计算机内部最基本的数据存储单位，一个二进制位可表示两种状态(0 或 1)。两个二进制位可表示四种状态(00, 01, 10, 11)。位越多，所表示的状态就越多。

2) 字节(Byte)

一个字节由八个二进制位所组成。字节用 B 表示，是计算机中用来表示存储空间大小的最基本的容量单位。一个字节即一个存储单元，每个存储单元都有地址，每个存储单元中存放的数据称为该存储单元的内容。如果用 $\times\times\times\times\text{H}$ 表示某存储单元的地址，则用 $(\times\times\times\times\text{H})$ 表示该存储单元的内容。例如， $(2000\text{H})=45\text{H}$ ，表示地址为 2000H 的单元内存放的内容是 45H。