高等学校计算机科学与技术应用型教材

HUIBIAN
YUYAN CHENGXU
SHEJI JIANMING
JIAOCHENG

汇编语言程序设计简明教程

主　编◎赵　梅　杨永生　夏　娟
副主编◎郭新宇

北京邮电大学出版社
www.buptpress.com

高等学校计算机科学与技术应用型教材

汇编语言程序设计简明教程

主　编　赵　梅　杨永生　夏　娟
副主编　郭新宇

北京邮电大学出版社
www. buptpress. com

内 容 简 介

本书以80X86系列微处理器为背景，系统介绍了汇编语言程序设计的基础知识和程序设计的基本方法。全书内容共7章。可以分为三部分；第1章为第一部分，是汇编语言程序设计的基础部分，介绍了数在计算机中的表示、计算机的基本结构和80X86寄存器组。第2章～第5章为第二部分，是本书的核心部分，详细介绍80X86的基本指令和汇编语言程序设计的基本方法和技巧。第6章、第7章为第三部分，分别介绍了中断程序设计、BIOS和DOS中断以及汇编语言和高级语言的混合编程等内容。

全书内容简明，每章后面配有大量的习题和参考答案，有选择题、判断题、填空题、简答题、编程题及程序分析等多种题型，是集教材与习题集于一体的适合学生学习与应试的教材。

图书在版编目(CIP)数据

汇编语言程序设计简明教程/赵梅等主编．--北京：北京邮电大学出版社，2012.11
ISBN 978-7-5635-3256-8

Ⅰ．①汇… Ⅱ．①赵… Ⅲ．①汇编语言—程序设计—教材 Ⅳ．①TP313

中国版本图书馆CIP数据核字(2012)第258538号

书　　名：汇编语言程序设计简明教程
主　　编：赵　梅　杨永生　夏　娟
责任编辑：王丹丹
出版发行：北京邮电大学出版社
社　　址：北京市海淀区西土城路10号(邮编：100876)
发 行 部：电话：010-62282185　传真：010-62283578
E-mail：publish@bupt.edu.cn
经　　销：各地新华书店
印　　厂：北京源海印刷有限责任公司
开　　本：787 mm×1 092 mm　1/16
印　　张：12.25
字　　数：288千字
印　　数：1—3 000册
版　　次：2012年11月第1版　2012年11月第1次印刷

ISBN 978-7-5635-3256-8　　定　价：26.00元

前　言

汇编语言程序设计是计算机科学与技术等专业的必修课程。近几年人们经常认为汇编语言的应用范围很小，而忽视它的重要性。其实汇编语言对每一个希望学习计算机科学与技术的人来说都是非常重要的，是不能不学习的语言。通过学习和使用汇编语言，能够感知、体会、理解机器的逻辑功能，向上可以理解各种软件系统，向下能够感知硬件，是我们理解整个计算机系统的最佳起点和最有效途径。然而，“汇编语言程序设计”课程在应用型本科院校的学时数却在逐年减少。在这种情况下我们编写了《汇编语言程序设计简明教程》和《汇编语言程序设计案例式实验指导》两本教材，目的是使学生能在短时内掌握汇编语言程序设计的基本方法和技巧。

本书是在多年教学积累的基础上，精心编写而成的，是为计算机及相关专业本、专科的“汇编语言程序设计”课程而编写的，它也特别适合于用做计算机工作者学习汇编语言程序设计的自学教材。在书中每章后面配有大量的习题和参考答案，有选择题、判断题、填空题、简答题、编程及程序分析题等多种题型，是集教材与习题集于一体的适合学生学习，同时也适合学生复习应试的教材。

本书以80X86系列微处理器为背景，系统介绍了汇编语言程序设计的基础知识和程序设计的基本方法。全书内容共7章。可以分为三部分：第1章为第一部分，是汇编语言程序设计的基础部分，介绍了数在计算机中的表示、计算机的基本结构和80X86寄存器组。第2章～第5章为第二部分，是本书的核心部分，详细介绍80X86的基本指令和汇编语言程序设计的基本方法和技巧。第6章、第7章为第三部分，分别介绍了中断程序设计、BIOS和DOS中断以及汇编语言和高级语言的混合编程等内容。

本书特点是力求简明实用，介绍的都是最基本的，并由“汇编语言程序设计案例式实验指导教材”来配套，进行程序设计能力的培养，围绕着程序设计的需要学习相应的知识点，做到“学一点、用一点、巩固一点”，不断进行程序设计训练的过程。出于本书的基本思想，没有对庞大的80X86指令系统做全面的介绍，而是选择了8086的16位指令自然扩展得到的32位指令，这些指令完全能够满足培养在校学生汇编语言程序设计能力的需要。

本书由赵梅、杨永生、夏娟任主编，郭新宇任副主编。书中的全部程序都经过了调试和运行。书中如有错误和不当之处，欢迎读者批评指正。

编　者

目　录

第1章　汇编语言基础知识

汇编语言是一种面向机器的“低级”语言，是电子数字计算机的基础语言，通过学习和使用汇编语言，能够感知、体会、理解机器的逻辑功能，向上可以理解各种软件系统，向下能够感知硬件，是我们理解整个计算机系统的最佳起点和最有效途径。

本章主要介绍学习汇编语言必备的基础知识，包括计算机内部数据的表示和计算机的逻辑结构。

1.1　计算机内部数据的表示

现代计算机可以处理各种各样的信息，如数值数据、文字、声音、图形等。这些信息在计算机内部都是用一组二进制代码来表示的，统称为数据。本节重点介绍计算机如何利用0/1表示现实中的各种数值、字符等内容。

1.1.1　数制

按进位的方法进行计数，称为进位计数制，简称数制。数制可以有很多种，如二进制、十进制、十二进制、十六进制、八进制等。人们每天都在使用着十进制数；但是由于物理器件的原因计算机内部采用二进制数，人们要与计算机交流，就需要了解不同进制之间的相互转换。下面介绍数制的基数与权。

一个 R 进制的数 N_R 可表示为

$$N_R = \sum_{i=-m}^{n-1} a_i R^i$$

式中，n 和 m 分别为 N_R 小数点左边和右边的数据位数；a_i 和 R^i 分别是 N_R 第 i 位的数符和位权，R 称为进位计数制的基数，即该数制中允许使用的数符个数。在 R 进制中允许使用的数符有 R 个，计算原则是“逢 R 进一，借一为 R”。

十进制数(decimal system)有0、1、2、3、4、5、6、7、8、9十个数符，因此十进制数的基数为“10”。十进制数各位的权是以10为底的指数幂。如45892d(d是十进制数的说明符，可以省略不写；也可以用大写字母D表示)：

4	5	8	9	2
10^4	10^3	10^2	10^1	10^0
万	千	百	十	个

其各位的权从低到高为个、十、百、千、万，即以 10 为底的 0 次幂、1 次幂、2 次幂、3 次幂、4 次幂等。

二进制数(binary system)有 0、1 两个数符，因此二进制数的基数为“2”，二进制数各位的权是以 2 为底的指数幂。如 1011b(b 是二进制数的说明符)：

1	0	1	1
2^3	2^2	2^1	2^0
8	4	2	1

其各位的权从低到高是 1、2、4、8，即以 2 为底的 0 次幂、1 次幂、2 次幂、3 次幂等

十六进制数(hexadecimal system)有 0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f 十六个数符，因此十六进制数的基数为“16”，十六进制数各位的权是以 16 为底的指数幂，如 8a9ch(h 是十六进制数说明符)：

8	A	9	C
16^3	16^2	16^1	16^0
4096	256	16	1

其各位的权从低到高是 1、16、256、4096，即以 16 为底的 0 次幂、1 次幂、2 次幂、3 次幂等。

为了区别数的不同进制，常用两种标记方法。第一种是加下标，即在数的右下角加上表示该数进制的数字，如$(12)_8$、$(10110)_2$、$(58AD)_{16}$和$(123)_{10}$等；第二种方法是加后缀，即在数后面加上表示该数进制的英文字母，如 12q、10110b、58adh 和 123d 等(q 表示八进制，b 表示二进制，h 表示十六进制，d 表示十进制)。十进制数后的标记常常省略，为了统一，在书中我们都采用第二种标记方法。

1.1.2 数制的转换方法

在日常生活中人们习惯了用十进制数，在研究问题或探讨问题的过程时，总是用十进制数来考虑和书写。当需要用计算机来处理这些数据时，首先就要把数据转换成计算机能够“看得懂”的形式，即把问题中的所有十进制数转换成二进制代码。在计算机运算完毕后，要将二进制数的结果转换成十进制数输出，以利于人的理解和使用习惯。

虽然在计算机中采用二进制数进行各种运算和操作有许多优点，但书写冗长，读起来麻烦，不便记忆。为此，人们在书写程序和算式时常采用十六进制。因此，计算机中常常需要进行不同进制之间的相互转换。

1. 非十进制数转换成十进制数的方法

非十进制数转换成十进制数的方法是“按权展开求和”。

【例 1.1】 将数 123h、101101b 转换成十进制数。

$123h=1\times16^2+2\times16^1+3\times16^0=291$

$101101b=1\times2^5+0\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0=32+0+8+4+0+1=45$

2. 十进制数转换成非十进制数

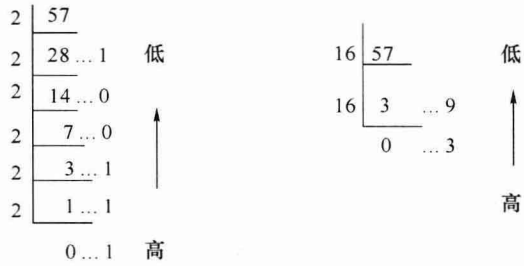
十进制数转换成非十进制数需要把整数小数分开进行。

(1) 整数部分的转换方法

用十进制数除以基数取余数，一直除到商为 0 为止，先得到的余数是低位，后得到的余数是高位，简称为“除基取余法”。

【例 1.2】 将 57 转换成二进制数和十六进制数。

转换成二进制数时，将十进制数整数除以二进制数的基数 2。转换成十六进制数时，将十进制数整数除以十六进制数的基数 16。



即 57＝111001b＝39h。

(2) 小数部分的转换方法

用十进制数乘以基数取整数，先得到的整数是高位，后得到的整数是低位。一直乘到需要保留的小数位数，简称为“乘基取整法”。

【例 1.3】 将 0.628 转换成二进制数和十六进制数，小数点后保留 4 位。

转换成二进制数时，将十进制小数乘以二进制数的基数 2。转换十六进制数时，将十进制小数乘以十六进制数的基数 16。

	整数部分			整数部分
$0.628\times2=1.256$ ……	1		$0.628\times16=10.048$ ……	a
$0.256\times2=0.512$ ……	0		$0.048\times16=0.768$ ……	0
$0.512\times2=1.024$ ……	1		$0.768\times16=12.288$ ……	c
$0.024\times2=0.048$ ……	0		$0.288\times16=4.608$ ……	4

即 0.628＝0.1010b＝0.a0c4h。

【例 1.4】 将 57.628 转换成二进制数和十六进制数。

将例 1.2 和例 1.3 的结果合并，即

$$57.628=111001.1010b=39.a0c4h$$

3. 十六进制数和二进制数之间的转换

由于十六进制数的基数是 2 的整数幂，所以这两种数制之间的转换十分容易。一个二进制数，整数部分只要把它从低位到高位每 4 位组成一组，不够 4 位高位补 0；小数部分从高位到低位每 4 位组成一组，不够 4 位低位补 0，直接用十六进制数来表示就可以了。

【例 1.5】 将二进制数 11010110111111.11b 转换为十六进制数。

```
00 11     01 01     10 11     11 11 . 1100b
  3         5         b         f  .c  h
```

即 0011010110111111b=35bfh。

反之，把十六进制数中的每一位用 4 位二进制数表示，就转换成相应的二进制数了。

【例 1.6】 将十六进制数 a19ch 转换为二进制数。

```
  a        1        9        c        h
1010     0001     1001     1100     b
```

即 a19ch=1010000110011100b。

显然，为把一个十进制数转换为二进制数，可以先把该数转换为十六进制数，然后再转换为二进制数，这样可以减少计算次数。反之，要把一个二进制数转换为十进制数，也可以采用同样的方法。

十进制数、二进制数及十六进制数之间的相互转换方法可归纳为图 1-1。

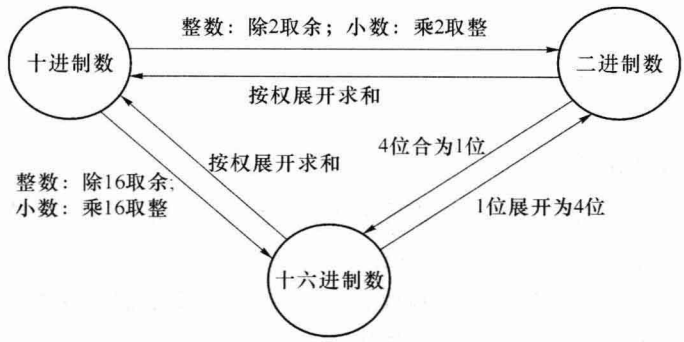


图 1-1 十进制数、二进制数及十六进制数之间的相互转换方法

1.1.3 数据组织

计算机内的信息按一定的规则组织存放。

1. 位(bit)

二进制数的一位包含的信息称为一比特(bit)，是表示信息的最小单位，用小写字母“b”表示。1 个二进制位可以表示一个开关的状态(称为开关量)，例如，用“1”表示“接通”，用“0”表示“断开”。

大多数的数据无法用一位二进制数表示，不能从计算机内单独取出 1b 的信息处理。

2. 字节(byte)

字节(byte)是计算机内信息读写、处理的基本单位，由 8 位二进制数组成，用大写字母“B”表示。1 个字节可以表示 256(2^8)个不同的值，可以用来存放一个范围较小的整数、一个西文字符或者 8 个开关量。

一个字节内的 8 个“位”自右(低位)向左(高位)从 0 开始编号，依次为 b_0，b_1，…，b_7，如图 1-2(a)所示。其中，b_0 称为最低有效位(Least Significant Bit，LSB)，b_7 称为最高有

效位(Most Significant Bit,MSB)。

3. 字(word)和双字(double word)

1个字(word)由16位二进制(2个字节)组成,可以存放一个范围较大的整数或者一个汉字的编码。它的16个二进制位仍然自右(低位)向左(高位)从0开始编号,依次为b_0,b_1,…,b_{15},如图1-2(b)所示。其中,b_0～b_7称为低位字节,b_8～b_{15}称为高位字节。

1个双字(double word)由32位二进制(4个字节)组成,可以存放一个范围更大的整数或者一个浮点格式表示的单精度实数。它的32个二进制位中,b_0～b_7,b_8～b_{15},b_{16}～b_{23},b_{24}～b_{31},分别称为低位字节、次低位字节、次高位字节、高位字节,如图1-2(c)所示。

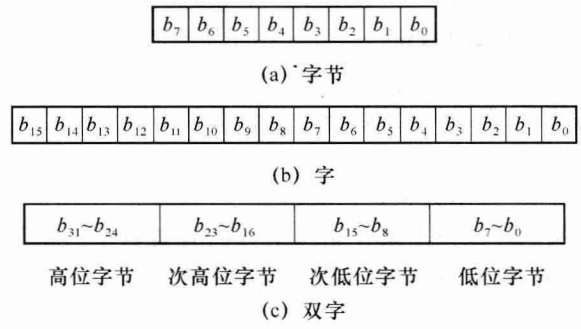


图1-2 数据组织

1.1.4 无符号数的表示

所谓无符号数是正数和零的集合。存储一个正数或零时,所有的位都用来存放这个数各位数字,无须考虑它的符号,“无符号数”因此得名。

可以用字节、字、双字或者更多的字节存储和表示一个无符号数。

用N位二进制表示一个无符号数时,最小数是0,最大数是2^N-1(二进制数111…111)。一个字节、字、双字无符号数的表示范围分别是0～255、0～65 535、0～4 294 967 295。

一个无符号数需要增加它的位数时,只需要在它的左侧添加若干个“0”,称为零扩展。例如,用一个字存储8位无符号数1011 0011时,低位字节置入这个无符号数,高位字节填“0”,结果为0000 0000 1011 0011(插入空格是为了阅读和区分,书写时没有这个要求)。

两个N位无符号数相加时,如果最高位产生了“进位”,表示它们的和已经超过了N位二进制所能表示的范围,需要向高位进位。同样,两个N位无符号数相减,如果最高位产生了借位,表示数据“不够减”,需要向更高位借位。

计算机内用进位标志(Carry out Flag,CF)表示两个无符号数运行结果的特征。如果CF=1,表示它们的加法有进位,或者它们的减法有借位;CF=0,则没有产生进位或借位。

1.1.5 有符号数的表示

可以用字节、字、双字或者更多字节来存储和表示一个有符号数。

表示一个有符号数有多种不同的方法，如原码、反码、补码表示法。同一个有符号数在不同的表示法中可能有不同的形式。

1. 原码

表示有符号数最简单的方法是采用原码。用原码表述一个有符号数时，最左边一位(最高有效位)二进制表示这个数的符号，“0”代表正，“1”代表负，后面是它的“有效数字”。例如，用字节存储一个有符号数时，$[-3]_{原}=10000011$。为了和计算机内的数据组织协调，通常用 8 位、16 位、32 位二进制表示一个数的原码。

用一个字节存储有符号数原码时，可以表示 127 个正数(1～127)、127 个负数(－1～－127)和 2 个“0”，“正 0”：0 000 0000，“负 0”：1 000 0000。

原码的表示规则简单，但运算规则比较复杂，不利于计算机高速运算的实现。

2. 反码

反码仍然用最高位“0”表示为正，“1”表示符号为负。

符号位之后的其他二进制位用来存储这个数的有效数字。正数的有效数字不变，负数的有效数字取反。例如，用字节存储一个有符号数时，$[+11011]_{反}=0\ 001\ 1011$，$[-11011]_{反}=1\ 110\ 0100$。

对于正数 $X=d_{n-2}d_{n-3}\cdots d_2d_1d_0$，$[X]_{反}=X=0d_{n-2}d_{n-3}\cdots d_2d_1d_0$。

对于一位二进制，$\overline{b}=1-b$。所以，对于负数 $Y=-d_{n-2}d_{n-3}\cdots d_2d_1d_0$，$[Y]_{反}=1\ \overline{d_{n-2}d_{n-3}\cdots d_2d_1d_0}=11\cdots111-|Y|=2^n-1-|Y|=2^n-1+Y$。

用一个字节存储有符号数反码时，可以表示 127 个正数(1～127)，127 个负数(－1～－127)和 2 个“0”，“正 0”：0 000 0000，“负 0”：1 111 1111。

反码的运算规则仍然比较复杂，可以用作原码和常用的补码之间的一个过渡。

3. 补码

补码表示法仍然用最高有效位表示一个有符号数的符号，“1”表示负，“0”表示正。

符号位之后的其他二进制位用来存储这个数的有效数字。正数的有效数字不变，负数的有效数字取反后最低位加 1。用字节存储一个有符号数时，$[+11011]_{补}=[+001\ 1011]_{补}=0\ 001\ 1011$，$[-11011]_{补}=[-001\ 1011]_{补}=1110\ 0100+1=1\ 110\ 0101$。

对于正数 $X=d_{n-2}d_{n-3}\cdots d_2d_1d_0$，$[X]_{补}=X=0d_{n-2}d_{n-3}\cdots\cdots d_2d_1d_0$。

对于负数 $Y=-d_{n-2}d_{n-3}\cdots d_2d_1d_0$，$[Y]_{补}=1\ \overline{d_{n-2}d_{n-3}\cdots\cdots d_2d_1d_0}+1=1111\cdots111-|Y|+1=2^n-|Y|=2^n+Y$。表 1-1 列出了用 8 位二进制代码表示的部分数值的原码、反码和补码。

表 1-1 部分数据的 8 位原码、反码和补码

真值(十进制)	二进制表示	原码	反码	补码
+127	+111 111	0111 111	0111 111	0111 111
+1	+000 0001	0000 0001	0000 0001	0000 0001
+0	+000 0000	0000 0000	0000 0000	0000 0000

续表

−0	−000 0000	0000 0000	1111 1111	0000 0000
−1	−000 0001	0000 0001	1111 1110	1111 1111
−2	−000 0010	0000 0010	1111 1101	1111 1110
−127	−111 1111	0111 1111	1000 0000	1000 0001
−128	−100 0000	无	无	1000 0000

用一个字节存储有符号数补码时，可以表示127个正数(1～127)，128个负数(−1～−128)，1个“0”(0000 0000)。其中，$[-1]_{补}=1\ 111\ 1111$，$[-128]_{补}=1\ 000\ 0000$。

如果把一个数补码的所有位(包括符号位)“取反加1”，将得到这个数相反数的补码。称为“求补”，$[[X]_{补}]_{求补}=[-X]_{补}$。例如，$[5]_{补}=0000\ 0101$，$[[5]_{补}]_{求补}=[0000\ 0101]_{求补}=1111\ 1011=[-5]_{补}$。

已知一个负数的补码，求这个数自身时，可以先求出这个数相反数的补码。例如，已知$[X]_{补}=1\ 010\ 1110$，求X的值(真值)可以遵循以下步骤：

$$[-X]_{补}=[[X]_{补}]_{求补}=[1\ 010\ 1110]_{求补}=0\ 101\ 0001+1=0\ 101\ 0010$$

于是，$$-X=+101\ 0010\text{b}=+82\text{d}$$

即 $$X=-82$$

4. 补码的扩展

一个补码表示的有符号数需要增加它的位数时，对于正数，需要在它的左侧添加若干个“0”，对于负数，则需要在它的左侧添加若干个“1”。上述操作实质上是用它的符号位来填充增加的“高位”，称为“符号扩展”。例如，$[-5]_{补}=1\ 111\ 1011$(8位)$=1\ 111\ 1111\ 1111\ 1011$(16位)，$[+5]_{补}=0\ 000\ 0101$(8位)$=0\ 000\ 0000\ 0000\ 0101$(16位)。

5. 补码的运算

补码的运算遵循以下规则：

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

$$[X-Y]_{补}=[X]_{补}-[Y]_{补}$$

或者 $$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=[X]_{补}+[[Y]_{补}]_{求补}$$

例如：

$$[15+27]_{补}=[15]_{补}+[27]_{补}=0000\ 1111+0001\ 1011$$
$$=0010\ 1010=[42]_{补}$$

$$[15-27]_{补}=[15]_{补}-[27]_{补}=0000\ 1111-0001\ 1011$$
$$=1111\ 0100=[-12]_{补}\text{(舍去借位)}$$

进行补码加法时，最高位如果有进位/借位，将其自然抛弃，不会影响结果的正确性。例如，$[(-3)+(-5)]_{补}=[-3]_{补}+[-5]_{补}=1111\ 1101+1111\ 1011=1111\ 1000=[-8]_{补}$。

计算机内用溢出标志(Overflow Flag，OF)表示2个有符号数运算结果特征。如果补码表示的有符号数的运算结果超过了该数据的表示范围，称为溢出，OF=1，反之OF=0，则没有产生溢出。

我们可以用下面的方法来进行判断。

两个同符号数相加，结果符号与原来相反，则产生了溢出。异号数相加不会产生溢出。

两个异号数相减，结果符号与被减数符号位不同，则产生了溢出。同符号数相减不会产生溢出。

从上面的叙述可以看出，补码的运算规则具有突出的优点：

(1) 同号数和异号数相加使用相同的规则；

(2) 有符号数和无符号数加法使用相同的规则；

(3) 减法可以用加法实现(对于电子计算机内的开关电路，求补是十分容易实现的)。

上述特性可以用来简化运算器电路，简化指令系统(如果有符号数和无符号数的运算规则不同，则两者运算需使用不同指令，用不同的电路来实现)。由于这个原因计算机内的有符号数一般都用补码表示，除非特别说明。

1.1.6 字符编码

计算机处理的对象除了数值数据之外，还有大量的文字信息。文字信息以字符为基本单元，每个字符用若干位二进制表示。

计算机内常用的字符编码是美国信息交换标准编码(American Standard Code for Information Interchange，ASCII)。它规定 7 位二进制表示一个字母、数字或符号，包含 128 个不同的编码。由于计算机用 8 位二进制组成的字节作为基本存储单位，一个字符的 ASCII 码一般占用一个字节，低 7 位是它的 ASCII 码，最高位置“0”，或者用作“校验位”。

ASCII 编码的前 32 个(编码 00～1fh)用来表示控制字符，例如 CR(“回车”，编码为 0dh)，LF(“换行”，编码 0ah)。

ASCII 编码 30h～39h 用来表示数字字符 0 ～9。它们的高 3 位为 011，低 4 位就是这个数字字符对应的二进制表示。例如：5＝011 0101＝35h。

ASCII 编码 41h～5ah 用来表示大写字母 A ～Z。它们的高 2 位为 10。

ASCII 编码 61h～7ah 用来表示小写字母 a～z。它们的高 2 位为 11。小写字母的编码比对应的大写字母大 20h。例如：A＝41h，a＝61h，a－A＝61h－41h＝20h。

1.1.7 BCD 码

十进制小数和二进制小数相互转换时可能产生误差，这对于某些应用会带来不便。计算机内部允许用一组 4 位二进制来表述 1 位十进制数，组间仍然按照“逢十进一”的规则进行，这种用二进制表示十进制数编码称为 BCD(Binary Coded Decimal)码。BCD 码分以下两种。

1. 压缩的 BCD 码

压缩的 BCD 码用一个字节存储 2 位十进制数，高 4 位二进制表示高位十进制数，低 4 位二进制表示低位十进制数。例如$[25]_{压缩BCD}$＝0010 0101b。需要使用压缩 BCD 数时，可以用相同数字的十六进制数表述。

2. 非压缩的 BCD 码

非压缩 BCD 码用一个字节存储 1 位十进制数，低 4 位二进制表示该位十进制数，对高 4 位的内容不作规定。例如，数字字符 7 的 ASCII 码 37h 就是数 7 的非压缩 BCD 码。

从上面的叙述可以看出，计算机内的一组二进制编码和它们的“原型”之间存在着“一对多”的关系。如有符号数+65 的补码、无符号数 65、大写字母“A”的 ASCII 码在计算机内的表示都是 41h，它甚至还可以是十进制数 41d 的压缩 BCD 码。所以，面对计算机内的一组二进制编码，它究竟代表什么？知道它的其实就是汇编语言的程序员。

1.2 计算机的基本结构

使用汇编语言进行程序设计时，不仅需要考虑求解问题的过程或者算法，安排数据在计算机内的存储格式，同时还要根据实际需求对计算机内的资源进行调度和分配。因此，作为一名汇编语言程序员，必须了解计算机的基本结构，了解有哪些可供使用的资源，以及不同资源在使用上的区别。不同的计算机具有不同的结构，本节主要结合 80X86 系列微型计算机来介绍程序员需要掌握的计算机“逻辑结构”。

1.2.1 计算机组成

迄今为止，电子计算机的基本结构仍然属于冯·诺依曼体系结构。这种结构的特点可以概要归结如下。

1. 存储程序原理

把程序事先存储在计算机内部，计算机通过执行程序实现高速数据处理。

2. 五大功能模块

电子数字计算机由运算器、控制器、存储器、输入设备、输出设备组成。

图 1-3 列出了各功能模块在系统中的位置，以及和其他模块的相互作用，图中实线表示数据/指令代码的流动，虚线表示控制信号的流动。各模块的功能简述如下。

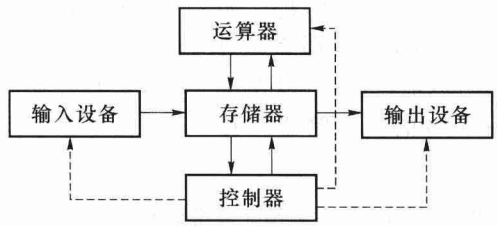


图 1-3 计算机的基本组成

(1) 存储器：存储程序和数据。

(2) 运算器：执行算术、逻辑运算。

(3) 控制器：分析和执行指令，向其他功能模块发出控制命令，协调一致地完成指令规定的操作。

(4) 输入设备：接收外界输入，送入计算机。

(5) 输出设备：将计算机内部的信息向外部输出。