计算机程序设计语言系列丛书

DOS 6.22

技术手册

—— 汇编语言教程

6.11

旗标
希望
学苑出版社

施威铭 著

计算机程序设计语言系列丛书

DOS 6.22 技术手册——汇编语言教程

施威铭　　编著
麻信洛　徐文军　改编
燕卫华　　审校

学苑出版社

(京)新登字 151 号

内 容 提 要

本书从最基本的原理开始，对 80x86 CPU 的结构、PC 的组成、80x86 的指令、DEBUG 的用法以及用 MASM 6.11 开发汇编语言程序一一给予详细说明。全书列出了大量的例子，供使用者逐一印证，以帮助读者克服学习汇编语言的困难。本书的重点在于为读者建立编写汇编语言程序的基本概念，因此所举的例子注重对过程的了解，而非例子本身的实用性。本书适用于计算机操作人员及开发人员。

需要本书的读者，请直接与北京海淀 8721 信箱书刊部联系，电话：2562329，邮编：100080。

在本书的改编过程中，刘杰做了大部分的录入工作，李晓中、张景生、宋涛等在校对和审核的过程中做了大量的工作，在此一并致以衷心的感谢。

版 权 声 明

计算机程序设计语言系列丛书

DOS 6.22 技术手册——汇编语言教程

编　　著：施威铭
改　　编：麻信洛　徐文军
审　　校：燕卫华
责任编辑：甄国宪
出版发行：学苑出版社　　**邮政编码**：100036
社　　址：北京市海淀区万寿路西街 11 号
印　　刷：　施园印刷厂
开　　本：787×1092　1/16
印　　张：12.875　　**字数**：295 千字
印　　数：1～5000 册
版　　次：1995 年 9 月北京第 1 版第 1 次
ISBN7 — 5077 — 0807 — 1/TP・18
本册定价：18.50 元

学苑版图书印、装错误可随时退换

目　录

第一篇　基本概念

第二篇　必备的工具

第三篇 指令介绍与练习

第四篇 高级应用

第一篇　基本概念

第一章　数字系统

计算机是一种机器，机器所使用的语言和我们人类的语言大不相同。所以，在接触计算机之初，首先要做的事就是了解计算机的语言——数字符号系统。

本章所要介绍的内容有：

□二进制、十进制、十六进制数字系统

□二进制、十进制、十六进制之间的转换及其四则运算

□正负数的表示法

□数据的基本单位：位(Bit)、字节(Byte)和字(Word)

最后我们会说明两种常用的符号系统：ASCII 码以及汉字编码。这是我们学习汇编语言的准备工作。让我们以愉快的心情来学习吧！

汇编语言是计算机的基本程序设计语言，速度快、效率高、但不易为人所了解。事实上，学习汇编语言就像学习外语一样，不但要学习其文字、语法，还要认识其生活环境、法令、制度，相对于学习汇编语言来说，就是要认识电脑的系统环境。因此，要想学好汇编语言，就必须对电脑内部的数制、符号、结构……等有相当全面的了解。所以，在本书的首章，我们将介绍与电脑系统紧密相关的数字系统。有了这些基本认识，我们才可以顺利的进入电脑世界。当然，勤于练习是学习任何语言的不二法门，这是研读本书随时要牢记在心的。

1.1　汇编语言与数字系统的关系

电脑内部的基本构造是开关装置。所谓开关装置是指以高电位(ON)来表示“1”；以低电位(OFF)来表示“0”，而电脑所能接受的命令就是一连串“0”和“1”的组合，例如：“0110110101110011”。因此在早期的电脑上，程序设计人员只能以机器语言给电脑下命令，才能使电脑进行运算(早期的程序设计甚至是通过更改电路来实现的)。以下面这行二进制的机器语言为例，我们分别将它转换成十六进制的机器码或汇编语言的格式：

```
00000101 00101010 00000000      ;二进制机器码
05 2A 00                        ;十六进制机器码
ADD AX,42                       ;汇编语言
```

如果我们对于上述三种情况都不熟悉，显然十六进制机器码比二进制机器码更容易记忆，而当我们知道了在汇编语言中 ADD 这个指令是代表相加的意思时，汇编语言又比十六进制机器码更容易令人接受，这也是程序开发者用汇编语言取代机器语言的原因。可见，我们之所以需要不同的数字系统，只是为了更方便地分析电脑工作的原理！

从上述的例子可以看出，电脑内部的数据适合以二进制来描述。但由于人们不习惯于使用 1000101011010110 这种一长串的数字表示法，因此才折衷采用了十六进制。在使用汇编语言的过程中，我们并不总是固定使用二进制、十进制或十六进制，而是根据具体的情况来选用某

种数制。这就好比一些原文的科技名词，在中文中还没找到一个比较贴切的的用语来表达它时，就直接将其原文穿插在中文中，这样做往往更容易使人理解。反复使用不同的数字系统也是这个道理！

1.2 二进制、十进制和十六进制数字系统

1.2.1 二进制数字系统

前面我们已经提到了：二进制是最适合于描述电脑内部运算的数字系统。所以我们首先必须知道二进制是怎样处理的。以我们所熟悉的十进制数字系统来说，其含义是指在这种数字系统中，计数是逢 10 进位的。同理，二进制数字系统，就是逢 2 进位的数字系统，两者之间的对应关系如表 1.1 所示。

表1.1 二进制数与十进制数对照表

二进制数	十进制数
0 = 0	0
0+1 = 1	1
1+1 = 10	2
10+1 = 11	3
11+1 = 100	4
100+1 = 101	5
101+1 = 110	6
110+1 = 111	7
111+1 = 1000	8
1000+1 = 1001	9
1001+1 = 1010	10
.	.
.	.

进位 → 1+1 = 10
进位 → 11+1 = 100
进位 → 101+1 = 110
进位 → 111+1 = 1000
进位 → 1001+1 = 1010

每加两次1就进位

加10次1才进位

1.2.2 十六进制数字系统

十六进制数字系统就是逢 16 进位的数字系统。也就是由 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 所组成的，其中 A、B、C、D、E、F 六个符号相当于十进制数中的 10、11、12、13、14、15 六个数字。两者对照见表 1.2 所示。

表1.2　十六进制数与十进制数对照表

十六进制数	十进制数
0	0
0+1= 1	1
1+1= 2	2
.	.
.	.
8+1= 9	9
9+1= A	10
A+1= B	11
B+1= C	12
C+1= D	13
D+1= E	14
E+1= F	15
进位 → F+1=10	16
10+1=11	17
.	.
.	.

表1.3　二进制数、十进制数、十六进制数对照表

二进制数	十进制数	十六进制数
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10
10001	17	11
.	.	.
.	.	.

有一点是值得一提的：人类只创造了一种数目的观念，但相同的一个数，却有好机种表示法。比如中文中的“你”这个字，英语是以“you”来表示；德语是以“du”来表示；法语中是以“vous”来表示；日语则是以“あなた”来表示。同理，“15”这个数，以二进制来表示是“1111”，以十进制来写是“15”，以十六进制来表达则是“F”，表 1.3 给出了三者之间的对照关系。

从表 1.3 中，我们可能会产生一个疑问：到底某数（如 11）是代表哪种数字系统？对于这个问题的解决方法是：我们通常会在该数字的尾部加一个英文字母（大写或小写）以示区别：

□在数尾加上 b，则表示是二进制数（取 binary 的首字母）。

□在数尾不加任何字母，则表示是十进制数。

□在数尾加上 h，则该数是十六进制数（取 hexadecimal 的首字母）。

例如：二进制数字系统的 1011b，相当于十进制数字的 11，同时也等于十六进制数字的 0Bh。

当然，如果在很容易辨别为何种数字系统的場合，数字后面的“b”或“h”是可以省略的。

1.3 二进制数、十进制数、十六进制数之间的转换关系

介绍完上节所提到的三种数字系统之后，在我们进入本小节之前，要先介绍一个“加权指数”的概念：见图 1.1。

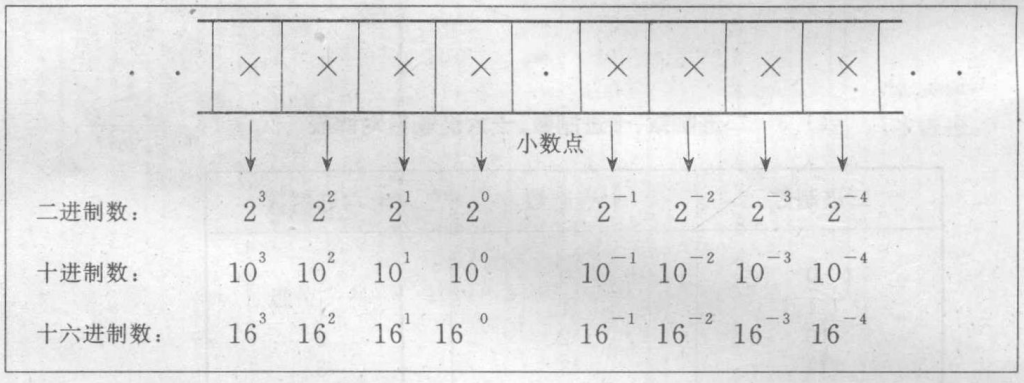


图1.1 三种不同数字系统下的加权指数大小

不同数字系统下的各个位数加权指数大小并不相同，但其原理都是一样的。以我们常用的十进制数字系统来说，个位数（第 0 位数）的加权指数就是 10 的 0 次方，而十位数就是 10 的 1 次方……第 N 位数就是 10 的 N 次方，依此类推，二进制的第 0 位数的加权指数则为 2 的 0 次方，第一位数则为 2 的 1 次方…。

实际上一般的数字都可以用加权指数的方式来书写，例如：

第3位　　第2位　　第1位　　第0位

$$1024 = 1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$$

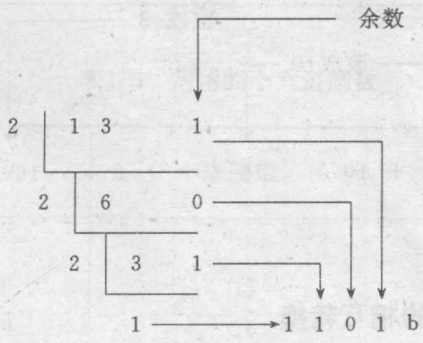
$$1010b = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$A1D1h = 10 \times 16^3 + 1 \times 16^2 + D \times 16^1 + 1 \times 16^0$$

接下来，我们就可以开始进行本节的内容：各种数字系统之间的转换。

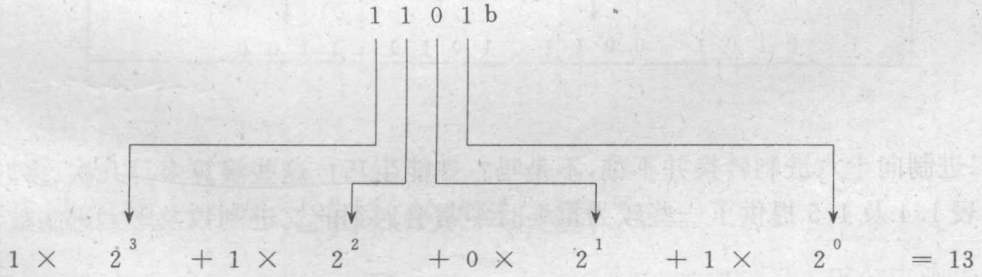
1.3.1 二进制数与十进制数之间的相互转换

例如：将 13 转换成二进制数的方法是



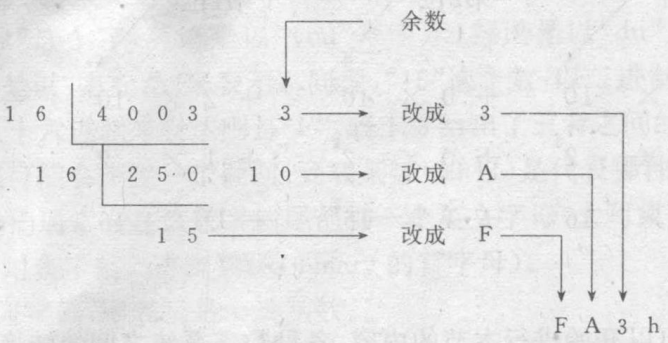
所以 13＝1101b。

同理，我们可以利用如下方法将二进制的数转换成十进制数：

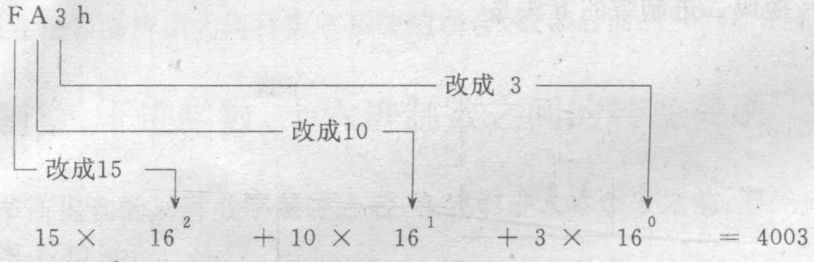


1.3.2 十六进制数与十进制数之间的相互转换

与上同理，十六进制数与十进制数之间的转换也是有规律可循的。例如：将 4003 转换成十六进制可用如下方法：

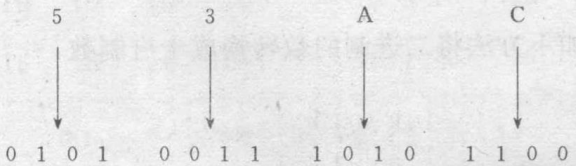


同样，以下是十六进制数转换成十进制数的方法：



1.3.3 二进制与十六进制之间的相互转换

二进制数与十六进制数相互转换时，由最右边开始将二进制数以四个为一组，如果不满四个则依次向左补 0。例如：



可见二进制向十六进制转换并不难，不是吗？熟能生巧！这些演算多算几次、多转换几次就熟悉了！表 1.4 及 1.5 提供了一些较为重要且经常会遇到的二进制数及十六进制数，请仔细的观察！

表1.4　常用的二进制数

二进制数	十进制数	2 的乘幂
1111	15	2^4-1
10000	16	2^4
111111	63	2^6-1
1000000	64	2^6
11111111	255	2^8-1
100000000	256	2^8
1111111111	1023	$2^{10}-1$

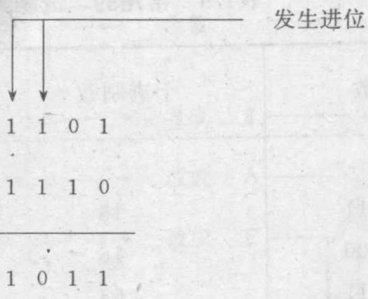
表1.5　常用的十六进制数

十六进制数	十进制数	16 的乘幂
F	15	16^1-1
10	16	16^1
FF	255	16^2-1
100	256	16^2
7FFF	32767	$8\times16^3-1$
8000	32768	8×16^3
FFFF	65535	16^4-1
10000	65536	16^4

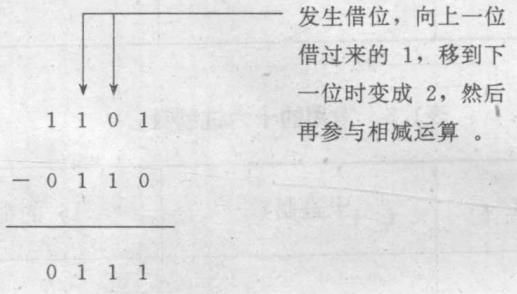
1.4　二进制数与十六进制数的四则运算

1.4.1　二进制数的运算

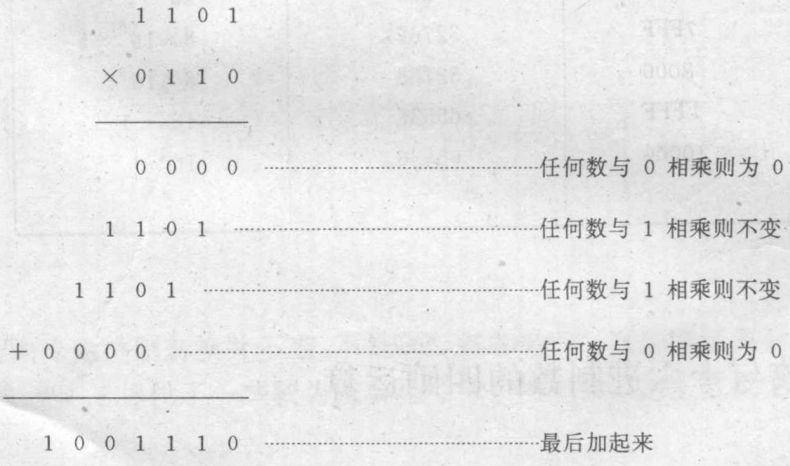
其实二进制的加减法并不难掌握，如果读者已经了解了上述的转换表，则应该已经找出了一些规律性，例如二进制数的加法如下：



请注意进位发生的时机(只有两个 1 相加才进位),及计算是由右往左算(即由低位数往高位数运算)。二进制数的减法如下:



至于乘法则是这么算:



除法则是这么算:

```
        1 0  ..... 商
    ┌──────
1 1 0 / 1 1 0 1
        1 1 0  ..... 2 进制数相减
        -------
          0 0 1
          0 0 0  ..... 2 进制数相减
          -------
              1  ..... 余数
```

都很简单吧？有兴趣的话，可自行练习本章后面的习题。习而熟之，也许你会喜欢这些算法。

1.4.2 十六进制数的运算

十六进制的数也可以加、减、乘、除。但十六进制数的乘、除法直接运算起来并不方便，通常是先转换成十进制数，算好之后再转换回来。以下仅就加法与乘法来作说明，至于减法和除法读者不妨自行练习。

□加法：

```
  0 D h
+ C 9 h
---------  (直接运算，如果不太习惯的话，也可以
  D 6 h    先换成 10 进制数运算，算完再换回来)
```

□乘法：

```
D h  ×  1 9 h =  1 4 5 h
↓                   ↑
换成十进制数运算      再将 10 进制数换回 16 进制数
↓                   ↑
1 3  ×  2 5  =  3 2 5
```

人类比较习惯十进制数的加减乘除，但电脑比较习惯二进制数的加减乘除，而十六进制数则是帮助人们去了解二进制数的一种数字。所以学习汇编语言的人要将二进制及十六进制的四则运算弄懂才行。

1.5 正负数的表示法

我们知道电脑内部充满着诸如"010101001"的信号，可以用二进制数字来表示。但前面我们所介绍的情况都是正整数的情形，如果是负数怎么办？而电脑内部并没有"+""-"等符号可以代表正负数。为了解决这个问题，有人提出以二进制数的最高位(Bit)来表示正负号，也就是说最高位为1时表示是负数，为0时则表示正数，如表1.6。

然而这种方法有个缺点，就是产生了+0和-0两个重复的数，并且原来00000000～11111111总共可表示256个数，现在只能表示255个数(因为+0和-0表示同一个数)。

表1.6 符号大小值表示法(Sign magnitude)

正 数		负 数	
00000000	+0	10000000	−0
00000001	+1	10000001	−1
00000010	+2	10000010	−2
0.....	.	1.....	.
0.....	.	1.....	.
0.....	.	1.....	.
01111111	+127	11111111	−127

所以又有人提出另一个方法，即所谓二进制补码——一个数的负数(补码)即是将该数的每一位取补数(即反相，将1变成0，0变成1)再加1。例如：

表1.7 二进制数的补码(2' s complement)**表示法**

	正数		取补数再加1		负数(补码)	
0:	00000000	⟶	11111111	+1 ⟶	00000000	:−0 (去掉进位)
1:	00000001	⟶	11111110	+1 ⟶	11111111	:−1
2:	00000010	⟶	11111101	+1 ⟶	11111110	:−2
3:	. . . . .					
.:	. . . . .					
.:	. . . . .					
127:	01111111	⟶	10000000	+1 ⟶	10000001	:−127
					10000000	:−128

这样就不会有+0和-0的问题了。并且如果把0放在正数这边，结果正数由0～127，负数

由-1～-128 各有 128 个，总共 256 个。

其实这种 2 的补码的表示法可以这样看更容易理解：就是一切由 0 开始，1、2、3…就是 0+1、1+1、2+1…，而-1、-2、-3…就是 0-1、-1-1、-2-1…

表 1.8　有符号数以不同数制表示的对照表

	二进制	十进制	十六进制
(126+1)	01111111	127	7Fh
.	0.....	.	.
.	0.....	.	.
(1+1)	00000010	2	02h
(0+1)	00000001	1	01h
由此开始 →	00000000	0	00h
(0−1，去掉借位)	11111111	−1	FFh
(−1−1)	11111110	−2	FEh
(−2−1)	11111101	−3	FDh
.	1.....	.	.
.	1.....	.	.
(−126−1)	10000001	−127	81h
(−127−1)	10000000	−128	80h

通常正整数的表示法又称为**无符号数**(Unsigned)**表示法**，而正负数的表示法则称为**有符号数**(Signed)**表示法**(因为有+、-号)。

一个负数前面可以无限制地加上 1，就如同一个正数前面可以任意加上 0 一样。例如处理 8 位(bit)数据时，-3 为 11111101b(或 FDh)。而处理 16 位数据时，-3 就要写成 1111111111111101b(或 FFFDh)。所以 11111101b 和 1111111111111101b 在用有符号数表示法时都代表-3，也就是说负数开头有几个 1 都可以任意加上去(二进制)。

1.6　电脑数据的基本单位：位(Bit)、字节(Byte)、字(Word)

位(Bit)是指二进制的一个位数，也就是指 0 或 1。但由于位的单位太小，不便于处理数据，因此在电脑内一般把 8 个位合成一组称为一个字节(Byte)，而以字节为单位来处理数据。在一长串的二进制数中，我们将最左边的位称为最高有效位(MSB 即 Most Significant Bit 的缩写)；而将最右边的位称为最低有效位(LSB 即 Least Significant Bit 的缩写)。

在电脑世界中，我们习惯于从右至左，由 0 开始计数，所以一个字节中的 8 个 Bits 是由 Bit0 开始向左计数直到 Bit7(这是源于英国哲学家罗素之后所产生的影响)。见图 1.2。