IBM PC[0520]

汇编语言程序设计

沈美明 叶乃菶 温冬婵

尹祥明 丁士元

清华大学出版社

IBM PC(0520)
汇编语言程序设计

沈美明 叶乃奉 温冬婵
尹祚明 丁士元 编译

苏伯珙 审校

清华大学出版社

内 容 提 要

本书主要根据美国 Waite Group出版社 1984 年出版的著名个人计算机丛书之一——Robert Lafore 编写的“Assembly Language Primer for the IBM PC&XT”一书，并增加了有关宏汇编技术、磁盘操作系统及 IBM PC/AT 等资料编译而成。书中内容也适用于 IBM PC 兼容机及国产0520微型计算机系列。全书共十三章，由浅入深地结合实例阐明汇编语言程序设计方法，使读者能边自学边上机实践，能较为顺利地掌握汇编语言程序设计技术。全书的叙述结合 IBM PC的磁盘操作系统进行，特别着重说明 DOS 功能调用的使用方法，并介绍了调用 ROM 例行程序、磁盘文件的存取及汇编语言与高级语言连接的方法。附录中介绍了宏汇编程序、磁盘操作系统、IBM PC/AT 机的指令系统及寻址方式，并提供了程序实例供读者参考。因此，本书既可作为高等院校“汇编语言程序设计”课程的教材，也可供使用汇编语言的工程技术人员参考。

IBM PC(0520)汇编语言程序设计

沈美明等 编译

苏伯珙 审校

☆

清华大学出版社出版

（北京 清华园）

北京朝阳关西庄印刷厂印刷

新华书店北京发行所发行

☆

开本：787×1092 1/16 印张：28 字数：696千字

1987年7月第1版 1987年7月第1次印刷

印数：00001—15000

统一书号：15235·292 定价：5.70元

前　言

汇编语言是计算机能提供的最快而又最有效的语言，也是能够利用计算机所有硬件特性的唯一语言，因而在许多对运行速度要求很高的场合，汇编语言是必不可少的，如操作系统、编译程序等系统程序多数是用汇编语言编写的。近年来，微型计算机的迅速发展，特别是它在管理、控制、通信等方面的广泛应用，使汇编语言程序设计技术不仅要为系统程序设计而且要为广大微型计算机用户所普遍使用。

“汇编语言程序设计”是高等院校电子计算机硬、软件专业学生必修核心课程之一。它不仅是计算机原理、操作系统等其他核心课程的必要的先修课，而且对于训练学生掌握程序设计技术、熟练上机操作和程序调试技术有重要作用。由于汇编语言本身的特点，使本课程必须结合一台具体的计算机来组织教学。我们认为，在目前情况下汇编语言程序设计课程应着重结合微型计算机进行。考虑到目前国内使用的微型机以 IBM PC机（包括XT、AT、IBM PC 兼容机及国产 0520 微型机系列）最为广泛，而且 IBM PC机的硬件特征及软件配置也比较适于作为汇编语言程序设计课程的基础机型。也就是说，掌握了 IBM PC机汇编语言程序设计的方法，再编写其他计算机的汇编语言程序也就不会有多大困难了。

本书主要依据美国 Waite Group 出版社 1984 年出版的著名的个人计算机丛书之一——Robert Lafore 编写的“Assembly Language Primer for the IBM PC&XT”一书进行编译的。该书具有下列主要特点：

1. 全书完全不用一般汇编语言书籍所用的从指令到程序设计方法的叙述过程，而是由浅入深、循序渐进地结合实例阐明汇编语言程序设计方法，以便读者能边自学边上机实践，较为顺利地掌握汇编语言程序设计技术。

2. 全书的叙述过程密切结合 IBM PC 的磁盘操作系统(PC-DOS)，特别着重说明 DOS 功能调用的使用方法。这样不但可简化程序设计，而且使编制的程序比较实用。

3. 书中给出很多程序实例，着重说明 IBM PC 机在声音、图形等方面的程序编制技术。介绍了调用 ROM 例行程序、磁盘文件的存取及汇编语言与高级语言连接的方法。这些程序实例既很有趣又很实用。

我们以该书为基础并考虑到我国的具体情况，除编译该书的主要内容外，还增加了宏汇编技术、IBM PC DOS 及IBM PC/AT 等有关资料。我们还准备编写“IBM PC 汇编语言程序设计习题集”以配合本书使用。书中程序的注释部分都译成中文以便于读者阅读，但程序清单中的注释字段仍保留原文，供读者上机时装入程序用。

本书适于初学者使用。读者只要有一种高级语言程序设计的基础，就可以比较顺利地通过学习本书掌握汇编语言程序设计技术。此外，本书也适用于有一定经验的程序设计人员熟悉 IBM PC 的汇编语言程序设计技术。因此，本书既可作为高等院校“汇编语言程序设计”课程的教材，也可供使用汇编语言的工程技术人员参考。

在学习本书时，最好能有一台配有适当软件的 IBM PC 计算机(或 IBM PC 兼容机或国

产 0520 机）提供上机实践条件。下面介绍一下使用本书所需要的设备。

硬件：

• IBM PC 或 IBM PC/XT 机（或国产 0520 微型计算机），既可以是带有一个或两个软盘驱动器的 IBM PC 机，也可以是带有固定磁盘的 IBM PC/XT 机。

• 主存储器至少应有 64K 字节的容量，能有 96K 或 128K 字节的容量则更好。

• 一台终端显示器。可以是单色显示器，或是 RGB 彩色显示器。最好使用 80 列的，否则可能会产生图象模糊等问题。

以上这些就是所需要的基本设备。此外，如果能有一台打印机在调试程序时会更加方便，标准的 80 列打印机就已够用，有 132 列的打印机则更好。

如果你所使用的机器属于 IBM 兼容计算机，则只要它能运行 MS-DOS 操作系统，本书的主要内容也都是适用的，只是在发声、图象等问题的处理上可能会有些差别。

软件：

• PC-DOS。可使用版本 1.00、1.10 或 2.00 中的任一种。有版本 2.00 最好。这是由于它的 DEBUG 程序功能较强，而且还引入了一套全新的“文件代号式磁盘存取”方式。但是，版本 2.00 要求存储器有较大的存储空间。如果存储容量只有 64K，那就只能使用版本 1.10 了。

• DOS 实用程序。它包括用来对汇编语言程序进行监控，调试和编辑的 DEBUG 程序；用来把汇编语言程序中的中间形式（OBJ 文件）转换为可执行程序（EXE 文件）的 LINK 程序，以及用来把 EXE 文件转换为 COM 文件（一种较简单的可执行程序）的 EXE2BIM 程序。

• 汇编程序。IBM 公司可为用户提供作为选件的 IBM PC MACRO 汇编程序软件包，其中包括两个不同的汇编程序：ASM 和 MASM。我们可以使用其中的任一种。MASM 功能更强，ASM 则短小因而装入迅速。如果存储器容量只有 64K，则只能使用 ASM。存储器容量大于 96K 时，就可以使用 MASM。

• 文本编辑或字处理程序。它可用来建立汇编语言程序的源文件（ASM 文件），可选用面向行编辑的 EDLIN 或面向屏幕编辑的 Word Star 或其他用于编辑的程序。

本书的编译工作由清华大学计算机科学与技术系的下列同志分别承担：叶乃華负责第 1、2、3章及附录 C，沈美明负责第 4、5、6 章，温冬婵负责第 7、8 章，尹祚明负责第 9、10、11 章及附录 D，丁士元负责第 12、13 章及附录 A、B，全书由苏伯珙审校。由于我们水平有限，书中倘有不当和错误之处，敬请读者批评指正。

目　录

第一章　汇编语言和 DEBUG 程序

概念

汇编语言与高级语言的比较
使用DEBUG
存储器
存储器寻址
ASCII 码

调试命令

D = Dump
F = Fill

本章首先对汇编语言进行概述，解释汇编语言与 BASIC 或 Pascal 等高级语言的差别，并一般性的讨论汇编程序的运行原理及其与用于高级语言的解释程序或编译程序的差别。

本章后半部分对汇编语言程序设计入门必需的 DEBUG 实用程序进行介绍。

1.1　汇编语言与高级语言

1.1.1　汇编语言与高级语言的差别

如果读者已熟悉 BASIC 或 Pascal 等高级语言，就会知道用这些语言编写的程序语句中具有一定程度的抽象性，一个 BASIC 语句，如 LET A = 3 或一个 Pascal 语句 A: = 3 都是在抽象级上运行的，在这一级上我们通常并不知道也无需知道计算机中“A”在哪儿，或当 A 赋值为 3 时，计算机中发生了什么变化。这是因为高级语言是面向对带有“代数公式”的数进行处理的。例如，最早的高级语言之一 FORTRAN，代表 FORmula TRANslator 这是一种很容易表示公式的语言，BASIC 是由FORTRAN 发展而来的，它基本上也是面向处理抽象代数中的数字数据，Pascal 也同样如此。使用这些语言的程序员就与计算机中真正进行的工作脱开，以便集中精力于处理公式。用高级语言书写的计算机程序涉及到一些抽象事物：代表数（numbers）和字符（characters）的变量。

与此相比，汇编语言则在具体级上运行，它要对位、字节、字（两个字节组合）、寄存器及存储单元进行处理。寄存器是微处理器中储存字节和字的物理设备，而存储单元有特定数字地址并在 PC 机的存储器件中有特定物理位置。

另一方面高级语言可以在各种不同的计算机上运行，比如在 IBM PC 机上的 BASIC 程序只作少量修改即可在 Apple 机上运行。而用汇编语言编写的程序只适用于某一特定的计算机，在多数情况下可适用于使用同一种微处理机芯片的某些计算机。

1.1.2 汇编程序与解释程序或编译程序的差别

读者如果在 PC 机上用 BASIC 语言编写过程序，就会对所进行的两步过程比较熟悉：首先写出一组 BASIC 语句构成程序，然后当执行程序时，这些语句被“解释”，或者说变为 8088 微处理机可执行的机器语言指令。（关于机器语言以后要详细讨论），读者可能对 BASIC 语言的这一解释过程并不很注意，因为它对用户似乎是“看不见的”。每次解释一行程序，而且在解释下一行之前对解释的每一行所产生的机器语言指令由 8088 加以执行，这过程的简况可参考图1-1。

在 Pascal 等编译性语言中，处理的办法有所差别。用户首先建立源文件，这是整个程序的文本文件。然后由编译程序把它变为机器语言指令。（实际上还使用了连接程序，现在暂时不予考虑）在象 Pascal 这样的编译性语言中，一次性地把整个程序都转换为机器语言。

汇编语言类似于编译性语言，而不象 BASIC 那样的解释性语言。首先建立由程序的文本组成的汇编语言源文件。然后由汇编程序将其汇编为机器语言指令。汇编程序处理这一过程与编译程序很相象，只是在汇编语言指令和机器语言指令之间的对应关系比 Pascal 语句与其所产生的一组机器语言指令之间的对应关系更为密切，这一点我们将在下一章中看到。

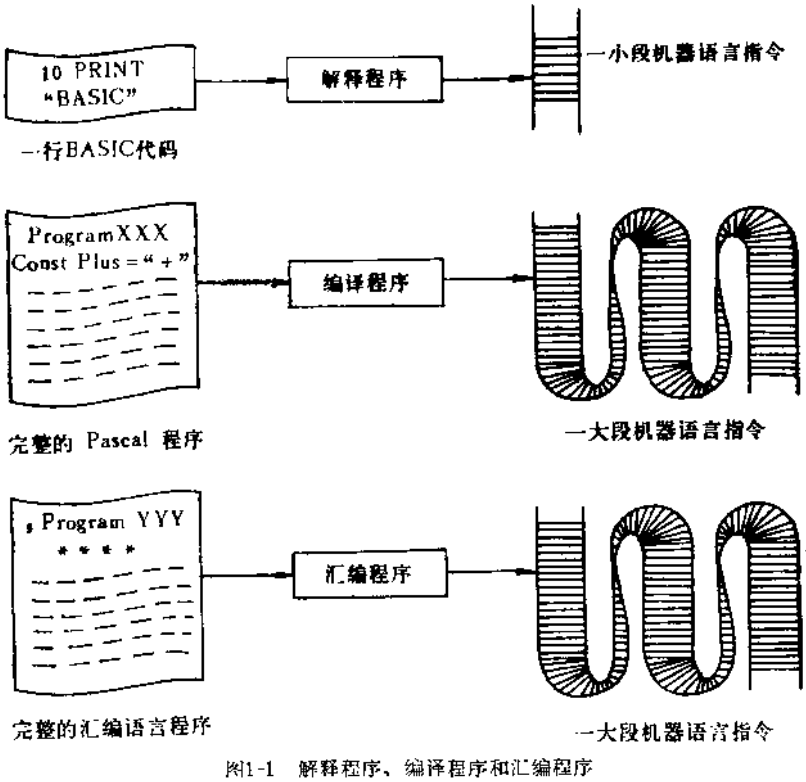


图1-1 解释程序、编译程序和汇编程序

以上所述是把汇编语言程序转换为机器语言指令的传统方法。但是在本书的前面几章中我们将采用另一种方法：利用 DEBUG 程序中具有的叫做“小型汇编程序”的功能。使用 DEBUG 来建立并运行一个短的汇编语言程序几乎象建立并运行 BASIC 这样的解释语言程序一样容易，本章后面部分将介绍 DEBUG，在第二章中要讲到如何使用 DEBUG 来对程序进行汇编。

1.2 微处理机

我们已经好几次提到过微处理机，一个微处理机就是可完成计算机中所有基本功能的一块器件。因为汇编语言不可避免地要围绕一个特定的微处理机器件，在IBM PC机中用Intel8088，因此在这里将对8088略加叙述，图1-2表示了8088发展历史。

最早的微处理机是Intel公司于1970年制造的4004在4004之前，制造计算机经历了好几种方法。最早的半导体计算机有几千个分离的晶体管安装在几百块印刷线路板上，在空调房间中占据了大量的机柜，价值好几十万美元，后来，集成电路——在一个小封装中放入十几个或更多的晶体管，把计算机的体积减小为几个较小的机柜，放在不需空调的房间里，但计算机的价格仍是六位数甚至七位数。

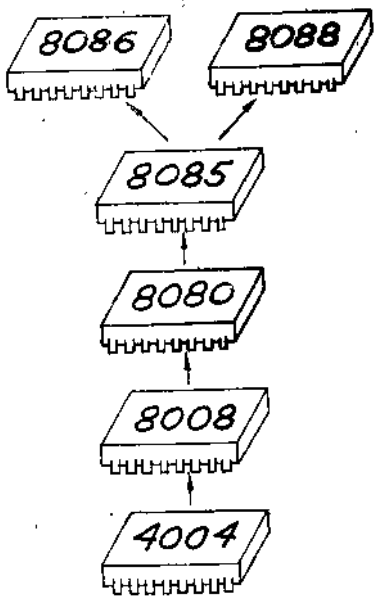


图1-2 8088系列树

4004作为我们年代里最惊人成就之一，它的体积小，重量轻而且价格便宜。4004并不是一台真正十分高效的微处理机，它只能处理4位字长的数据，仅具有最基本的指令系统。但是不久就有了第一台8位微处理机8008。8008发展成为8080，然后又进一步改进为8085，但8080在很多计算机中仍在使用。

下一步的主要进展是从8位发展到16位，因为16位微处理机比8位机提供了更多的功能及使用更大存储器的能力。达到这一突破的微处理机是Intel 8086，8086处理16位数据，它需要16位存储器，16位数据总线(连接计算机各部分)及其它16位外围设备，然而由于8位计算机的使用时间已很长，现存的价格合理的很多外围设备仍然是8位形式，于是Intel公司生产了8086的另一种形式叫做8088，8088具有与8086相象的内部结构，相同的16位寄存器，但当它与外界连系时采用8位数据：一次一个字节。所以8088使用的存储器和外围设备可以是真正的8位模式，这样也比较便宜，因而可以降低计算机系统的成本，这就是IBM PC机所采用的方法。

1.3 DEBUG与汇编程序

如上所述，要在IBM PC机上运行简短的汇编语言程序有两种方法。第一种方法是用汇编程序ASM或与之类似但更先进的MASM，人们通常用这两个汇编程序中某一个来对汇编语言程序进行汇编(转换为机器语言程序)。汇编汇编语言程序的另一方法是使用另一个不同的程序，称之为DEBUG。DEBUG不是真正的汇编程序，它的主要用途是用于“调试”(即修改程序中的错误)汇编语言程序。但是，也可以用DEBUG来汇编简单的汇编语言程序。

本书前面几章中都选用DEBUG来运行程序。其原因如下：

(1) DEBUG比ASM(或MASM)汇编程序容易操作得多。为了用DEBUG来键入及运行一个程序，只需调用DEBUG本身即可，这是很简单的。与此相比若用汇编程序来进行，

则要用文本编辑程序、汇编程序本身和一个叫做 LINK 的程序，还经常需要另一个叫做 EXE2BIN 的程序。其中每一个程序都需要相当复杂的一系列命令才能运行。因此对初学者来说使用 DEBUG 比较合适。

(3) 使用 DEBUG 所要求的程序比使用汇编程序所要求程序的“额外开销”要少得多。这些额外开销必须以程序语句的形式出现在 ASM 的“源文件”中，但这在 DEBUG 中是不必要的。

(3) DEBUG 可使用户更紧密地与计算机中真正进行的工作相联系。读者不久将看到，DEBUG 有可能深入到计算机运行的最基本级上。

当然，汇编程序具有在对长程序进行汇编时所必不可少的各种有效特性，但目前用 DEBUG 更为合适。下列表格概括了 DEBUG 和汇编程序的优缺点。

DEBUG与汇编程序比较

DEBUG	汇编程序
运行容易	运行较难
程序的额外开销少	程序的额外开销多
与机器联系紧密	与机器隔离
不太通用	非常通用
适用于短程序	适用于长程序

DEBUG 除了可以对汇编语言程序进行汇编外，DEBUG 同样可以用来检查和修改存储单元，装入、存储及启动运行程序，检查及修改寄存器（以后会学到什么是“寄存器”）。换句话说，设计 DEBUG 是用来使人与 IBM PC 机的各种物理特性相连系的，

1.4 DEBUG 的运行

假如把一个具有 DEBUG 程序的磁盘插入到 A 驱动器，终端上就出现提示符 A> 等待用户的下一步动作。(如果使用固定磁盘，必须先保证 DEBUG 程序已复制到固定磁盘片上，在本书正文中凡是见到 A> 的地方都用 C> 来代替。) DEBUG 是 PC-DOS 的系统盘所提供的程序之一。

在 DOS 提示符之后，键入程序名“DEBUG”。(在本书中让用户“键入”某物，意思就是在键盘上敲入“某物”，然后按位于数字键组左边的回车键↵。)

```
A>debug←键入此命令
-          ←DEBUG 的提示符
```

屏幕上出现的短划线是 DEBUG 的“提示符”，这符号通知用户机器已作好准备来接受下一个命令。

1.4.1 “D” 命令

通常键入单个字母命令，后跟一个或多个数字即可告诉 DEBUG 做什么事。在文本中提到这些单个字母命令时，通常用大写字母以示醒目（如“D”）。但当键入时，也可用小写字母，它的作用与大写字母相同。例如，键入字母“d”，后跟数字“1”，“0”和“0”。

```
-d100        ←键入
08F1:0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00……
08F1:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00……
08F1:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00……
08F1:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00……
08F1:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00……
08F1:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00……
08F1:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00……
08F1:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00……
```

这些显示内容表示什么意思呢？首先在显示屏幕上所看到的可能并非都是零。“D”命令所做的就是在屏幕上“转储”或者说显示出机器存储器的一部分。每一对数字代表存在存储器某单元中数据的一个字节即 8 位。如果在装入 DEBUG 之前，存储器中恰好有其它 数据，当键入“D”时，就会显示出来，所以可能会看到各种看起来无意义的数字，如

```
-d100
08F1:0100  03  EB  42  90  75  03  EB  41-90  2C  30  72  38  3C  0A  73   .KB.u.KA..Or8<. S
08F1:0110  34  52  8B  D3  9F  03  DB  03-DB  03  DA  03  DB  DI  DE  9E   4R. S. . I. I. Z. lQ
08F1:0120  D1  D6  8A  D0  B6  00  9F  03-DA  DI  DE  9E  DI  D6  5A  E8   QV. P6...ZQ. QVZh
08F1:0130  21  00  72  OA  74  OA  2C  30-72  04  3C  OA  72  D3  4I  4A   1. r. t..or. <. rSAJ
08F1:0140  8A  C7  OA  CO  C3  9F  4I  4A-9E  F9  C3  E8  05  00  75  01   .G. @CAJ. ych.. u.
08F1:0150  C3  EB  F8  8A  C5  0A  C1  75-01  C3  49  42  8B  F2  AC  24   Ckx. E. Au.ClB.r.S
08F1:0160  7F  0A  C0  F9  75  01  C3  3C-0C  F9  75  01  C3  3C  0A  F9   ..@yu.C<.yu.C<.y
08F1:0170  75  01  C3  3C  IA  F9  75  01-C3  72  01  C3  F5  C3  A9  46   u.C<.yu.Cr. CuClF
```

显示屏幕上的所有数字都是十六进制的，事实上，十六进制是 DEBUG 唯一认识的数字系统，如果读者还不熟悉这种表示方法，可先看书后的附录A。

我们采用这一惯例：除了在程序清单中，或在上下文清楚的场合外，十六进制数后面都跟一个小写字母“h”以区别于十进制数。当然，因为 DEBUG 只用十六进制，所以输出时不用“h”，在作为 DEBUG 命令键入时，也不必在十六进制数后面加“h”。

对于十进制数，除非上下文清楚，在数字后面跟一个小写字母“d”。数字 0 到 9 在两种系统中是相同的，所以不必在它们后面跟一个区别字母，有时为了一致起见还是这么做。

一个 8 位字节的数需要用两个十六进制数来表示。这两个十六进制数的值可以从 00h 到 FFh（即是从 0 到255d）。所以以上打印出来的所有两位数字都包括在这个范围中。在显示屏幕的每一行有16d个这类数字。打印输出的中间有一短划线是为了区分一行中 左 边 8 个字节与右边 8 个字节。

1.4.2 地址

最左边那一列数据（如 08F1:0120）是数据字节的存储器地址，转储中显示的每一个字节都具有一个特定的地址，如图 1-3 所示。

图 1-3 中左边的竖列代表存储器的实际部分。注意在 DEBUG 转储中，每个存储单元，也就是字节是如何与一个特定的数相对应的。

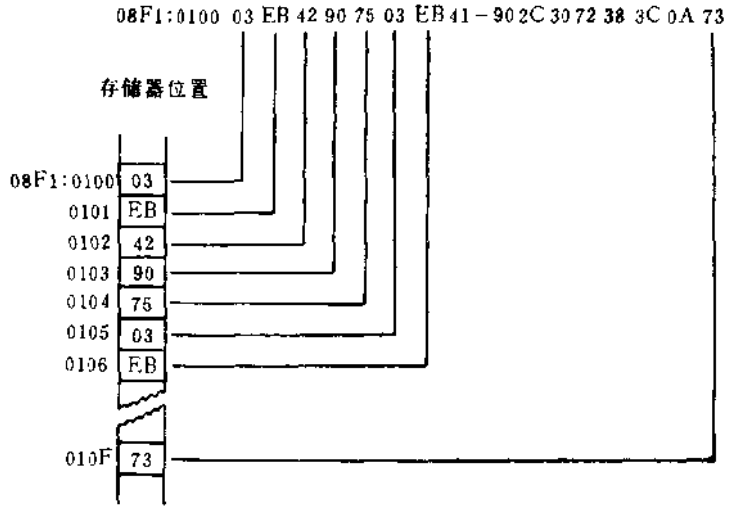


图1-3 转储中每个字节是存储器中的一个字节

每个地址由两个数组成，中间用冒号隔开，这个数所表示的意思如下：

偏移地址

冒号右边的数0100叫做偏移地址（offset address），在下面几章中这将是地址中所要讨论的唯一部分。所以，如果读者想跳过下面几段实际上不会有什么问题。

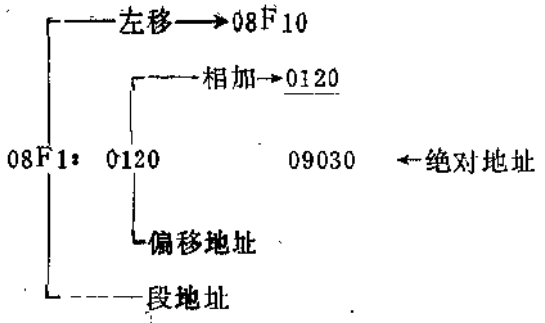
段地址

冒号左边的数08F1叫做段地址（segment address）。（用户系统可能会出现不是08F1的数，这也无碍），段地址部分比较复杂，所以我们推迟到第八章再作深入讨论。但是在这里用极普通的术语来说明其意思，以便读者容易理解以后六章中的有关问题。

为了得到真正的地址，先取出段地址，将其左移一个数位（十六进制），再加上偏移地址

简单地说，这就是地址两个部分的概念。8088大多是处理四位十六进制数，如FFFFh或1234h。但是在8088中有较多的存储器地址，以至于需要用五位十六进制数来指定地址，如FFFFFh或12345h。Intel公司用两个四位十六进制数来表示每个存储器地址：第一个数是偏移地址，第二个数是段地址。这两个数组合成实际地址或称之为绝对地址（absolute address）。段地址（左边的数）被左移一个数位，就象用10h与之相乘一样，然后把它加到偏移地址上（右边的数）。

例如，假定在DEBUG转储中，指示的数为08F1:0120，把08F1左移一位得到08F10，然后再加上120，所得到的五位和就是代表这一特定存储单元绝对地址的十六进制数，如下所示：



从现在开始直到第八章，可以避开段地址而仅注意偏移地址的原因是：我们只在存储器中某一特定部分来运行程序，这部分称之为段，段的长度为64K字节，也就是65536d字节，即FFFFh字节；其地址可用一个四位十六进制数来表示，所以在段中所有需要用来表示地址的就是四位数字的偏移地址。对此在第八章中讨论存储器分段时还要作进一步解释。

偏移地址和DEBUG

请注意。DEBUG转储中左边那列的偏移地址都以零结尾。如果读者熟悉十六进制数，就可理解其原因。每行中有16d或10h个字节，所以当从0h到Fh计数时，就准备在表示10的那列增加1，因为在十六进制中10h是紧跟着Fh以后的数。所以显示16d（10h）个字节，然后移到下一行，把地址增加10h，再显示10h个字节。

如果在每一列数值的顶上打印出列地址的值，那么显示出的内容就容易阅读和理解了，如下所示：

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
08F1:0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

应该清楚，在顶行的第一个字节是在存储单元0100，下一个是在0101，再下一个在0102等等。类似地，第二行的第一个字节是在0110，第二个是在0111等等。

1.4.3 “F”命令

如果要改变所显示内容，一个简单的办法是使用DEBUG的“F”或“fill”命令。这一命令用一个指定的十六进制数填入一部分存储单元。使用“fill”命令时，用户键入“f”，再键入三个数，每个数之间用一个空格分开，其中第一个数是开始填入的地址，第二个是填入中止的地址，第三个数是打算填入到第一和第二地址之间的常量（从00h到FFh）。注意虽然要填入的数是由两位十六进制数（字节）组成的，但地址仍是十六进制数。当然不必键入开头的零，所以有时可以键入少于四位数的地址，如下所示：

键入：

```
-f120     14f     ff
 |         |       |
 |         |       |--- 要填入的常量
 |         |
 |         |————————结束地址
 |
 |——————————————————— 起始地址
```

为了看到所发生的变化，必须再对存储器的同一部分进行转储。

```
-d100
08F1:0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0120 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF ................
08F1:0130 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF ................
08F1:0140 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF ................
08F1:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
08F1:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

现在可以看到，正象用"F"命令指定的那样，在120和14F之间的所有存储单元都填入了FF，（当然，如果在填入前是其它的数而不是零，在转储后它们仍为那些数，而不显示零）。

1.4.4 ASCII码

对转储显示出来的那些小点和字符就是左边的数字所代表的字符（如"A"，"B"等等）。表示一个特殊字符的数叫做"ASCII码"（ASCII代表"Americon Standard Code for Information Interchange"），读者可能已经知道，ASCII码是计算机存储器中表示字符的标准方法（在IBM PC机技术手册中有一张代码表）。

因为00和FF都不代表可打印出来的ASCII字符，所以在ASCII码显示对应这些数字的位置中充满了小点，表示是"非打印字符"（如果计算机存储器开始放的是些无用的内容，并非都为零，那么有些数可能是可打印字符）。为了看到字符显示随数字而变化的情况，我们用表示可打印ASCII码字符的数填入一部分存储器。

键入以下的DEBUG命令：

```
-f100 117 61
-f178 175 24
-d100
08F1:0100 61 61 61 61 61 61 61 61-61 61 61 61 61 61 61 61   aaaaaaaaaaaaaaaa
08F1:0110 61 61 61 61 61 61 61 61-00 00 00 00 00 00 00 00   aaaaaaaa........
08F1:0120 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
08F1:0130 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
08F1:0140 FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF   ................
08F1:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
08F1:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
08F1:0170 00 00 00 00 00 00 00 00-24 24 24 24 24 24 24 24   ........$$$$$$$$
```

十六进制的61是小写字母“a”的ASCII码，十六进制的24是美元符号($)的代表，我们可以把它们既看作数同时在右边又可以看作字符。

小结

本章讨论了汇编语言与高级语言的差别，也解释了一些关于DEBUG实用程序的操作。读者可发现在现阶段用DEBUG来作一些试验是很有用的。建议读者填入不同的常数，再进行“转储”，同时观察其显示结果及检查存储器的各个部分。在以后几章中，读者会更多地使用DEBUG，并且将体会其方便性。