80386/80286

汇编语言程序设计

[美] 威廉·默里　克里斯·帕帕斯　著
白晓笛　译　周明德　校

電子工業出版社

80386 / 80286
汇编语言程序设计

〔美〕 威廉·默里　　克里斯·帕帕斯　　著
白晓笛　译　　周明德　校

電子工業出版社

内 容 提 要

本书是为配合 IBM PC／AT，IBM PS／2 及国产 286、386 等新型微计算机的开发应用而出版的。

书中介绍了 80386／80286 微处理器及 80387／80287 协处理器的基本结构、指令集及其汇编语言程序设计技术，还介绍了如何使用汇编伪指令、宏调用、过程和库以及与高级语言的接口技术等。每一部分都有较多的应用程序举例，内容充实、实用性强。对已经接触或打算使用 16 位和 32 位微型计算机的读者，都是一本很有价值的实用手册。

80386／80286汇编语言程序设计

〔美〕 威廉·默里 克里斯·帕帕斯 著

白晓笛 译 周明德 校

责任编辑：王惠民

电子工业出版社出版（北京海淀区万寿路）

新华书店北京发行所发行 各地新华书店经售

北京密云卫新印刷厂印刷

开本：787×1092毫米 1／16 印张：31.75 字数：762千字

1988年6月第一版 1988年6月第一次印刷

印数：1—15,000册 定价：8.10元

ISBN 7-5053-0139-X／TN 69

校对者的话

从1971年美国Intel公司推出第一代微处理器4004到1985年Intel公司宣布它的全32位微处理器80386，仅仅经过短短的十五年，但片子的集成度提高了100倍，性能提高了三个数量级，进入了第四代，主振频率提高了10倍，寻址能力提高了2^{16}=64K倍，执行速度达到了3～4 MIPS（1MIPS =100万条指令／秒），从CPU的性能来说，已经达到了80年代的超级小型机的水平，功能十分强大。

从8位微处理器来说，Intel的8080、8085，Motorola的M6800和Zilog的Z80三足鼎立，各有千秋。但是从16位到32位通用微处理器来说，则是Intel的8086(8088)—80286—80386和Motorola的68000—68010—68020两雄相争了。但自从IBM的PC、PC／XT采用了8088CPU，以后的PC／AT采用了80286，今年宣布的PS／2，采用了8086—80286—80386，形成了一个新的完整的微型计算机系列以后，Intel系列片子的名声就更大了。

为了帮助广大读者学习、消化、分析和应用IBM的PC／AT、IBM的PS／2系列以及国产的GW286等机器，译者能尽快地把本书奉献给大家，相信会得到广大读者的热烈欢迎的。

译者毕业于清华大学，是一位相对来说比较年轻的作者，加上作者本人的刻苦学习和努力工作，有着较好的素质。近十年来又有较丰富的科研、开发和教学工作以及写作的实践经验，与校对者长期合作共事，工作能力较强、教学水平较高。这次奉献给广大读者的作品译文准确、严谨，文字流畅、通顺，是一部较好的译作。校对者郑重地向广大读者推荐。

周明德

1987年7月

原　　序

编写这本书是为了达到三个目的：(1)介绍使用 80386／80286 微处理器和 80387／80287 协处理器进行汇编语言程序设计的更广阔的领域；(2)教你怎样编写简单的和复杂的汇编语言程序；(3)作为程序设计指令和程序实例的一本参考书。

80386／80286 汇编语言是 8088／8086 汇编语言的一种强大的派生物。本书中出现的许多程序也能在 8088／8086 系列的计算机上运行。这主要是由于如下事实造成的：Intel 的 80xxx 系列的所有成员都是向上兼容的。基本的指令和特性对于早期的微处理器和最新的系列来说都是共同的。随着未来的 DOS 版本提供高级的程序设计特性(例如"保护方式"程序设计)，早期的微处理器和新的 CPU 之间的差距将会不断加大。

汇编语言是一种强有力的语言，它使程序员能够完全控制计算机的操作。许多高级的编译语言(例如 Pascal、Ada、FORTRAN 等等)使程序员只能依赖于编译程序作者的恩惠。如果编译程序的编写者支持某种特性(例如查询游戏适配器)，OK！但是如果他不支持，你怎么办？如果用汇编语言，你就可以编写你自己的子程序，或者利用计算机制造商提供的功能很强的 BIOS 子程序。事实上，绝大多数的汇编语言程序都是对各种高级语言的补充，加上了许多被原作者遗漏了的特性。

这本书通过例子(完整的例子)教授程序设计。本书中列出的每一个例子都是完整的，带有所有必要的程序头。换句话说，如果你完全按照原样键入一段码，这段程序就能够执行。没有简略了的例子，也没有为了节省篇幅而只列出例子的一部分。作者作出的每一点努力，都是为了用使你能够加强对各条指令和程序设计形式的理解的方式，选择、解释和提供每一个程序，不管你是一个初学者，还是一位有经验的程序员。

目　　录

第一章　汇编语言介绍

学习汇编语言程序设计，可能是软件工程师所能从事的最值得个人尝试的和技术上最具有挑战性的任务之一。如果能有一些巧妙地把高级语言和各种硬件结构结合起来的工具和知识，你就可以更好地掌握你的机器。对外部设备的直接控制和相应的命令、存储器管理、运行速度、码的效率、数据保密等等问题，都有待于汇编语言程序员去解决。掌握汇编语言程序设计需要有经验和注意研究细节。

除非你已经涉足于软件开发的特殊领域（例如致力于处理器的设计），汇编语言将是你最有用第二语言。由于汇编语言给予程序员直接存取寄存器和存储器的能力，并开辟了面向位的指令的独一无二的领域，它常常成为解决大多数高级语言所难以解决的程序设计任务的唯一工具。

汇编语言程序能够产生最快的可执行的码，因为它绕过了语言解释程序步骤（例如使用APL和BASIC所需要的那样）和语言编译程序步骤（如使用Pascal、C、PL／I和Modula—2所需要的那样）。当然，发挥这种速度的优点是需要代价的。

汇编语言程序员必须仔细注意一切细节。汇编语言程序以它自己的语言控制了微处理器而没有求助于编译程序检查的帮助。汇编程序提供了一些基本的操作，可以完成简单的数据传送、比较和逻辑操作的任务。从这些最基本的操作的结果来说，一页汇编语言码在最深奥、最好的一页高级语言码面前显得相形见绌。由于这个原因，大多数大的程序都不是用汇编语言编写的。（这一规律有一个例外，就是在由专门的公司所投资的软件开发项目中，要求付出长时间的努力编写运行得极快和效率极高的应用程序。）

运用机器语言要求掌握许多高级语言程序员所不熟悉的原理。汇编语言程序员必须考虑存储器的分段操作。当直接控制存储器的存取时，必须作出许多周密的决定。程序和数据应保存在存储器的什么地方？是否需要堆栈？如果需要，存储器的哪一部分应当保留？程序员必须作出有关每一数据的大小（字节、字、双字、四字、十字）和类型（BCD、串、静态、动态、有符号／无符号、实数、浮点）的种种决定。这些数据元素都是双下标的存储器变量吗？是否可用80287／80387数学运算协处理器？存储器变量是否要转换成80287或80387所需要的格式？应当使用哪些寄存器？哪些端口存取彩色显示器、单色显示器和打印机？这种种决定中的许多考虑都是只有在汇编语言程序设计中才有的。

汇编语言的两个主要的优点之一是它的码能以惊人的速度执行。汇编语言码能以微秒的速度执行。这些码仅仅是真正的机器码（微处理器所能懂得的语言）的一种助记符（实际的二进制机器指令的缩写的符号表示法）。

语言解释程序（例如用于BASIC和APL的解释程序）要对源码进行检查，并调用许多内部的子程序，把源码逐行翻译成机器码。这些子程序已经编写好，并且是为微处理器预先定义的。由于每次运行程序时都必须重新计算表达式和对出错条件进行检查，因此它们的执行往往是很慢的。

语言编译程序（例如用于Pascal、PL／I和FORTRAT的编译程序）阅读源码并把它转

换成微处理器能够直接执行的操作码序列。经过编译的程序比经过解释的程序执行起来快得多，因为转换只进行一次。但是编译过的程序有一个缺点：编译程序必须执行各种各样的命令。这一任务通常需要考虑码的额外开销，即使对于最小的程序也是如此。一个编译过的 BASIC 程序可能要占用 33K 字节的内存和磁盘存储器才能在显示器上显示几个字符。用汇编语言编写的同样的程序可能只占用这一开销量的三分之一，即只用大约 10K 字节。

现在让我们来看一看汇编语言程序的第二个主要的优点。没有汇编语言，用户将被限制于由已存储的程序所提供的功能。例如，假如某个人正在编写一个处理关键性数据的应用程序，而用户却只是经过一般训练的数据输入员，那么如果能有一个禁止 CTRL — ALT — DEL 这种起破坏数据作用的组合键的程序将是很有用的。在大多数情况下，这只能在汇编语言级才能完成。

汇编语言使软件工程师能够与操作系统接口，并使他们能够直接控制对显示器、打印机和重要的硬盘／软盘存储设备的操作。应用程序的程序员也常常不得不直接与操作系统接口。这样的子程序通常都是用汇编语言写成的。

在微处理器的设计过程中，计算机是以非凡的速度开发的。微处理器的"系列树"仅仅跨过了一个十年。如果继续以现在的速度进行开发，我们就要年年给这株树添枝加叶!

第一代 IC(即集成电路)是在六十年代初开发的。IC 极大地减小了电容、二极管和三极管的尺寸，而且把它们都放在一小块纯硅片上。在第一个十年的末期，Intel 公司开发出了它的第一个 8 位的微处理器芯片 8008。1974 年，第二代的微处理器 8080 问世了。8080 向用户提供了许多通用的能力。这时，众多的竞争者带着诸如 Zilog 公司的 Z — 80 一类的产品涌入了迅速扩大的市场。

第三代微处理器仅仅在四年之后就诞生了。1978 年，Intel 公司开发出了微处理器 8086。虽然这一芯片包含了一些与它的前辈 8080 向上兼容的能力，它却采用了先进得多的设计，具有许多新的特性。Intel 公司也开发出了 8088 微处理器作为 8086 的一个变种。8088 采用了稍微简单的设计，并包括了与流行的 I／O 设备兼容的能力。8088 的结构使它成为当时可用的最先进的微处理器之一。它有那么多强有力的特性，以至于 IBM 公司把它的整个第一代个人计算机都建立在 8088 的基础上。

几乎就在同时，8088／8086 的兄弟— 8087 实数数学运算协处理器芯片出现了。这个数值数据处理器专用于高速度、高精度的数学计算。

Intel 公司并没有躺在它过去的成就上休息。1984 年年中，该公司在结构设计上用 80286 微处理器完成了另一次较大的飞跃。这个新一代的 CPU 主要是为要求高性能的应用程序而设计的。80286 与 8088／8086 向上兼容。它采用了存储器管理、保护机构、任务管理和支持虚拟存储器这样一些现代技术。所有这些功能强大的特性都包含在一片 VLSI(超大规模集成电路)芯片中。这一电路向微型计算机的用户提供了许多在小型计算机上才有的计算上和结构上的特性。

由于采用了一套共同的寻址方式和基本指令，80286 与 8088／8086 向上兼容。这一基本结构支持高级语言(如 Pascal、PL／M 和 C)，因为寄存器组的设计完全适合于编译程序生成的码。

80286 支持几种非常有用的数据类型，例如串、BCD 和浮点格式。这一设计也支持对静态／动态的数组、记录以及记录中的数组这样一些复杂的数据结构的寻址。

80286 的存储器结构支持模块化程序设计技术。采用这一技术，软件工程师可以把存储器分成若干段。存储器分段提供了更短的码，因为在段内访问的路径较短。分段电路适用于有效地实现先进的存储器管理技术一例如虚拟存储器和存储器保护。

80286 还提供了很大的地址空间，以便满足今天的应用程序的需要。实际的存储器可以由大到 16 兆字节(2^{24}字节)的RAM或ROM组成。这样大的空间使微处理器在存储器中能同时保存许多大的程序以及与它们有关的数据结构，并能够进行高速存取。

对于动态改变存储器用量需求的应用程序(例如多用户系统)，80286 为每个用户提供了大到 2^{30}字节(1千兆字节)的虚拟地址空间。这样大的地址空间几乎消除了对程序的数量和大小的限制，而这些程序可以成为整个系统的一部分。

Intel 80286 是为支持多用户的、可重新编程的和实时的、多任务的应用程序而设计的。可供四个或五个用户同时使用的微型计算机系统现在可以扩展到支持该数量的三倍以上的用户，并支持响应时间占以前所需要的时间的 1／6 (或更少)的实时系统。

最后一个(至少到目前为止)进入 Intel 系列树的是 80386。80386 的外部数据总线是 32 位宽，是 80286 数据总线的两倍。它可以寻址 4 千兆字节的存储器。

在读完本书和试过本书的程序之后，你就能够全面地了解 80286 ／80386 和 80287 ／80387 的内部结构。你将非常熟悉指令集和内部寄存器结构及其使用。在学习了汇编语言之后，你就会懂得计算机是怎样用它最基本的语言进行工作的。你会明白汇编语言程序设计的优点和缺点，并知道什么时候应当连接或产生汇编语言程序，或插入汇编语言程序码。对 IBM 公司、Microsoft 公司和 Speedware 公司的三个汇编程序的深入讨论，将使你能够选择最适合你的特定的应用程序的汇编程序。

你将学习什么是伪指令和什么时候以及怎样使用伪指令。你也将学习怎样编写你自己的宏调用程序和过程。这本书的重点在于讲述包括建立、连接和修改宏调用子程序库的一些基本原理。

本书中的许多举例程序将向你说明怎样利用计算机的作图和文件管理特性。你会理解并能够使用高速和高精度的 80287 和 80387 数学运算协处理器。

本书以详细地讨论高级语言与汇编语言接口的方法为特色。本书还将向你说明怎样通过 DOS 和 BIOS 子程序与操作系统接口。

你将学会怎样采用一系列技术对程序进行测试和使用 DEBUG 进行调试，其中包括 IBM 的 DEBUG 工具和 Microsoft 的 SYMDEB 工具的使用。

本书假设你已经掌握了最流行的高级语言之一，例如 BASIC 或 Pascal。本书并不打算教你这些语言。熟知十六进制数字系统是很有帮助的(但不是必须的)，因为这是理解与、与非、或、或非、异或、左移／右移、循环、求补等等逻辑操作的基础。

第一节　数字系统

懂得计算机怎样存储信息是掌握汇编语言程序设计的关键。在计算机内部数据是怎样表示的?因为计算机是一种电设备，因此它只认识电压。特别是，大部分微处理器的操作都取决于两种可靠的电压电平的出现和消失。我们可以把这两种电压看成是接通或断开，1 或 0。这后一对量(1 或 0)是最小的单位，也叫做位。

人类最初使用十进制计数系统是因为我们有十个指头。计算机却只认识两种独特的状态，用 0 和 1 两个符号代表。当然，这就导致了人和计算机之间的通讯问题。不同符号码制的发展就是这种语言不兼容性的直接结果。

发明 ASCII 码(美国标准信息交换码)就是为了代表通常我们与打字机键盘相联系的所有的符号。几种数值代码的发明，也是为了用计算机能够懂得的形式表示数字值。2 的补码这种数字编码使计算机能够表示整个数字的正和负。对汇编语言程序员来说，最重要的数字编码是十六进制码。十六进制改进了以二进制格式表示信息的可靠性，不管信息是指令、存储器地址还是数据。

一. 二进制数

我们的十进制计数系统是因为在这个编码系统中包括了 10 个独特的符号(符号 0 到 9)而得名的。二进制码只有两个单独的符号：0 和 1。对于表示很大的数这似乎好像是一个极大的限制。很快你就会看到，二进制码的基本原理其实是你早已知道的道理!

首先，让我们看一看十进制数—比如说 1024。还在小学时，就有人教我们，这个数有 4 个 1，2 个 10，没有 100，有个 1000。这是一个有权的数字表达法。当你在数字中向左移位时，每个数值都有一个增加了的权，即价值。还有另一个方法来看数字 1024。我们的十进制就是以 10 为基数，可以用下述方法看这个数：

```
1 0 2 4
        └──→ 4 × 10^0 =      4(最低有效数位)
      └────→ 2 × 10^1 =    2 0
    └──────→ 0 × 10^2 =  0 0 0
  └────────→ 1 × 10^3 = 1 0 0 0(最高有效数位)

        1 0 2 4
```

我们用这个数字中的每一位数乘以上升到相应的幂级数的基值，指数按照从最低有效位置移到最高有效位置的顺序而增加。

同样的转换原理也可以用于任何一个二进制数。二进制数 1010 在转换成十进制码时，看起来就像这样：

```
1 0 1 0
        └──→ 0 × 2^0 =  0(最低有效位)
      └────→ 1 × 2^1 =  2
    └──────→ 0 × 2^2 =  0
  └────────→ 1 × 2^3 =  8(最高有效位)

        1 0
```

在这种情况下，我们采用了基数 2，并把新的基数值上升到相应的指数幂，从最低有效位(LSB)开始直到最高有效位(MSB)。这是很容易的。要从某一种基数转换为十进制数，可以使用和 1024 的十进制表达法或 1010 的二进制表达法转换一样的过程，唯一的差别将是使用另一个基数。

我们刚才已经看到把一个数字从二进制转换成十进制是多么容易。对于从十进制转换成二进制这样的相反的过程，可以从要转换的数字中简单地减去 2 的指数幂。例如，我们想要把十进制数 11 转换成二进制。可以从用 11 减去 2^3(8)开始，留下余数为3。没有剩下 2^2(4)，但是剩下一个2^1(2)。减去2留下余数为1。最后减去2^0(1)得到余数为0。这就使我们得到十进制 11 转换为二进制的结果 1011。

适用于较大的二进制数的一种简易方法如下。这是连续重复进行的除法运算，使用你想要转换的数字的基数为除数，而想要转换的数字为被除数。十进制数 50 转换成二进制就像这样：

被除数	除数	结果	余数	
50	2	25	0	——→答案=1 1 0 0 1 0
25	2	12	1	
12	2	6	0	
6	2	3	0	
3	2	1	1	
1	2	0	1	

你可以用这个方法把任何十进制数转换为任何基数。要把 428 变为十六进制，可以简单地代入 428 作为被除数，而 16 作为除数。

1．二进制加法和减法

二进制的加法和减法与十进制的加法和减法是一样的，只是你正以 2 的某次幂产生进位和借位，而不是以 10 的某次幂产生。让我们来看两个例子：

```
  2 5
+ 8 5
-------
1 1 0
```

在这个例子中，当我们用两个 5 相加时，产生了一个进位到下一个有效位置，即 1×10^1。因此下一列加上10。现在我们用 8 和 2 以及进位相加，产生了另一个进位到 10^2列，并产生和为 11。写下新的进位，使我们得到结果 110。

现在让我们看一看二进制加法：

```
  1 0 1 0 (1 0)
+ 0 0 1 1   (3)
-------------
1 1 0 1 (1 3)
```

在这个例子中，0 + 1 得到和为 1，没有进位。1 + 1 得到总数为 2，以二进制表示就是 10。这个 10 中的 1 是对下一个高有效位的进位并代表 1×2^1。这个进位代入2^2列，而最高有效位的 1 也记下来，得到和为 1101。

下一个例子说明当两个二进制数相加时，唯一的另外一种应当考虑的可能性。

```
  0 0 1 1 (3)
+ 0 0 1 1 (3)
-------------
  0 1 1 0 (6)
```

前两个 1 产生了一个进位到 2^1 列。这一列已经有两个 1 了。现在又包括了进位 1，使该列共有三个 1。让我们更仔细地看一看这一步。

```
    1
    1
+   1
-----
  1 1
```

1 + 1 返回二进制结果为 10，而当你用 1 与二进制的 10 相加时，你得到 11。你已经产生了一个进位到下一个有效位，并返回了结果为 1。

二进制减法使用了与十进制减法一样的基本原理，唯一的区别是当从下一个有效位置产生借位时，它是上升到某一幂指数的 2 的值，而不是 10 上升到某一指数的幂。

```
  5 3 4
- 2 5 1
-------
  2 8 3
```

从 4 中减去 1 没有什么困难。但是，当我们试着从 3 中减去 5 时，我们就需要从下一个有效位置，即 $1 * 10^2$ 产生一个借位，使我们能够从 130 中减去 50。最后，我们从 4 中减去 2，并得到结果 283。

现在让我们看一看二进制减法的例子：

```
  1 0 1 1
- 0 1 1 0
---------
  0 1 0 1
```

在最低有效位，从 1 减去 0 没什么问题。在下一个有效位位置上，从 1 减去 1 也没有任何困难。我们的问题出在从 0 减去 1。这就需要对下一个有效位借位，这发生在 $1 * 2^3$ 位上，即十进制的 8。我们现在可以减去 $1 * 2^2$，留下差为 $1 * 2^2$，这就在第三列给我们一个 1，而最后结果为 0101。

2．字节

在我们继续讨论不同的计数系统之前，我们必须记住我们所有的程序和数据都是在称为计算机的硬件设备上存储和运行的。在某种程度上，计算机的结构决定了码和数据的格式和范围。

计算机并不是随意地存储各种长度的二进制数或位。在 80286 结构中，硬件的存储器单元是一系列 8 个连续的位，称为字节。

位的位置是从 LSB(最低有效位)0 数起的，一直到 MSB(最高有效位)7。图 1—1 表

示几个连续的存储器单元，每个单元都保存着一个字节的信息。这个信息可以是机器指令、另一个存储器单元的地址、数字或字符数据。

8 个位可以产生 256 个唯一的状态。这就使一个存储器单元能够包含从 0 到 255 之间的所有正的二进制数，也可以保存一个 ASCII 字符(见 ASCII 字符表附录)。"字"是涉及一个单独的存储器单元中所保存的位的数目的技术术语。早期的计算机结构只有 4 位的存储器单元。对这种早期的结构，一个字就代表 4 位。某些计算机具有能够存储 64 位的存储器单元。在这种情况下，一个字指的就是一个 64 位的串。就 80286 的结构来说，一个字是 16 位的。一个 8 位字节可以分成为 2 个 4 位位组，称为半字节。一个半字节也可以表示一位十六进制数。

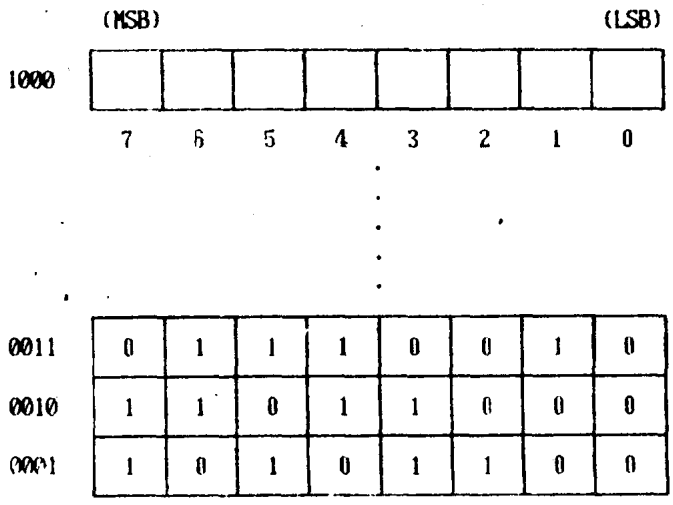

图 1—1． 连续存储器单元的一个例子

3．字符

一个字节中的 8 位并不总是必须代表数字值。ASCII 字符就是能够表示字母和数字字符的 7 位编码。它指定了一种专用的二进制格式代表每一个字母、数字、和与标准的打字机键盘有关的特殊字符。它也包括了对特殊控制码的表示法。

7 位码可以代表 128 个唯一的符号和码。第 8 位有时用作数据传送和取回的错误检测码。有些字符 ROM 芯片的制造商把这个第 8 位用于存取扩展的字符集。加上这个有效位使可以唯一表示的符号的数目有效地加倍到 256。这就允许表示特殊的外语符号、数学符号以及非常重要的图形符号。

4．带符号数

用 80286 的 8 位字节结构作为我们的硬件模型，让我们更仔细地看看内部的数据表示法。当所有的 8 位都用于表示正数时，我们可以表示从 0(00000000)到 255(11111111)的数