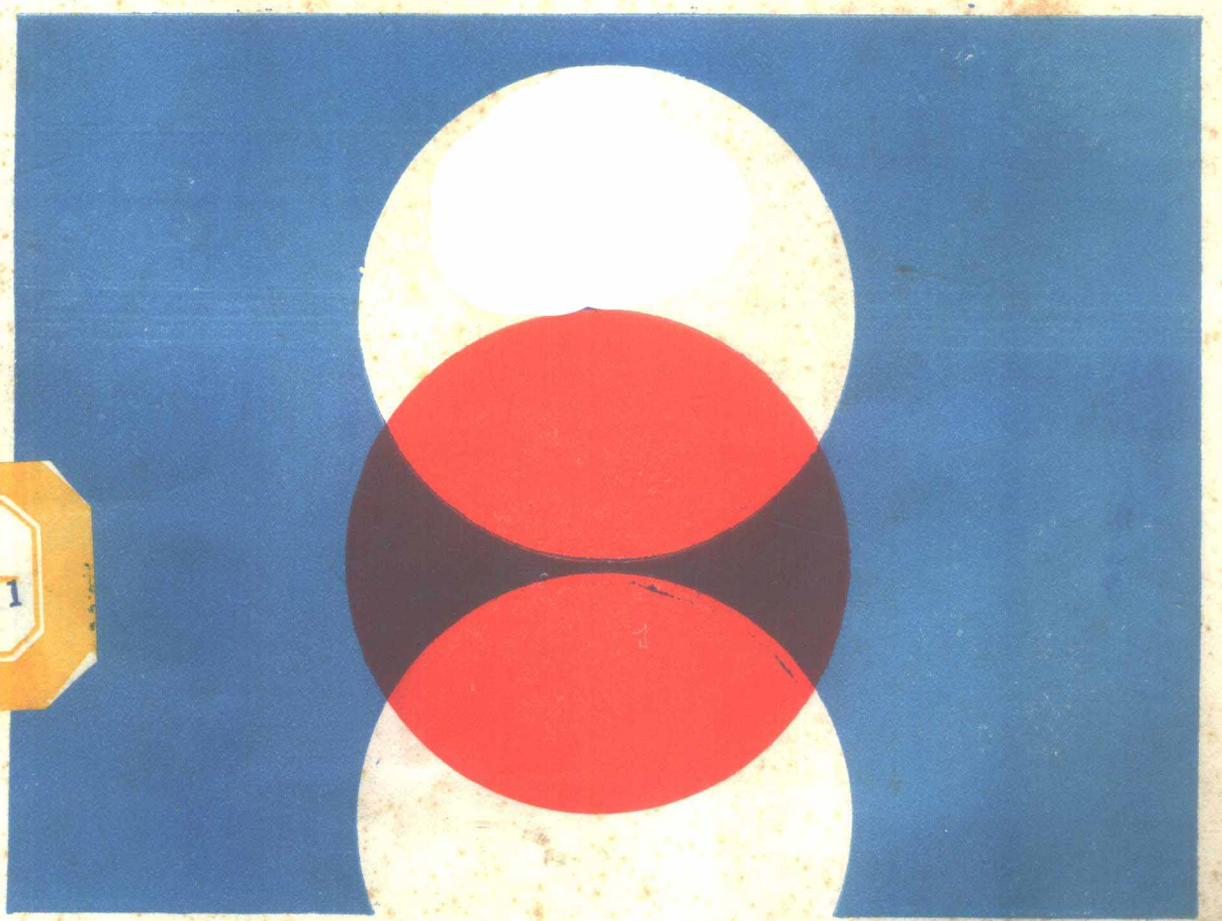
• 陈哲庆 李玉兴 译

• 上海交通大学出版社

VAX—11

结构化汇编语言编程



VAX-11结构化汇编语言编程

〔美〕Robert Sebesta 著

陈哲庆
李玉兴 译

CONVERT A... 通大学出版社

内 容 简 介

本书是美国计算机专业使用的大学课本，具有全面、通俗、内容丰富和范例众多的特点。全书共分十五章，完整地介绍了在 VAX-11 系列计算机上运行的汇编语言。整个汇编语言包括：245 条机器指令，为数众多的伪指令和 22 种寻址方式。

本书适合作为结构化汇编语言学习的教材，也是从事 VAX 机软、硬件工作的技术人员学习 VAX-11 宏汇编语言的优秀参考书。

VAX-11 结构化汇编语言编程

上海交通大学出版社出版

（淮海中路 1984 弄 19 号）

新华书店上海发行所发行

浙江上虞汤浦印刷厂排版

上海崇明永南印刷厂 印装

开本 787×1092 毫米 1/32 印张 18 字数 446 000

1987 年 7 月第 1 版 1987 年 7 月第 1 次印刷

印数：1—35 00

统一书号：ISBN7-313-00025-1/TP31 科技书目：152-295

定价：3.00元

译 者 的 话

《VAX-11 结构化汇编语言编程》一书是美国 1984 年出版的有关 VAX-11 宏汇编语言的计算机专业大学课本。VAX-11 宏汇编语言是美国数字设备公司(DEC)定义的在 VAX-11系列计算机上运行的汇编语言，DEC 公司在 VAX-11 系列机上提供了完善的汇编程序和调试程序。整个汇编语言包括了 245 条机器指令，为数众多的伪指令和 22 种寻址方式。

与一般介绍汇编语言的书籍不同，该书将庞大的指令系统根据操作数类型不同，先易后难地把它们分散在各个章节中讲解。伪指令和寻址方式也采用同样的方法，使读者在学习了几条指令以后，就能很快地进入编写程序的阶段，摆脱了先交待全部指令和寻址方式然后才能编程的文本方式。每章有大量完整的例题，这些例题不仅演示了新指令的用法，而且注意体现程序的完整性和实用性。

与讲解指令、伪指令和寻址方法并行不悖的是遵循先易后难的编程原则，由顺序、分支、循环、子程序(包括过程)到宏指令。

VAX-11 宏汇编语言属于低级语言，它反映了机器的结构特点，这一点与高级语言不同；但是它又不同于常用的八位和十六位微处理器，VAX-11 机的许多复杂的机器指令是由微码完成的，所以它的指令功能要比微处理器强得多，使用这种语言可使学生有较多的时间得到较高层次(如小的应用程序和系统程序的局部模块)的编程训练，因此对计算机专业的学生来说，这是一种进行程序设计基本训练比较理想的编程语言。此外，VAX-11 系列计算机是多用户系统，有强有力的屏幕编辑功能和调试程序支持，上机实习非常方便；每章末尾有大量习题，供读者练习；VAX 也是一种较先进的机种，所有这些条件使该书成为一本有价值的《程序设计》教科书，并已在国内部分院校采用。

由于本书内容丰富，范例众多，因而也是从事 VAX 机软、硬件工作的技术人员一部学习 VAX-11 宏汇编语言的很好的参考书。

目前 DEC 公司大量推出 MICRO VAX-Ⅱ 和 VAX-8000 系列计算机，对于 VMS 4.2 以上的操作系统版本来说，VAX-11 的宏汇编语言与上述两类机型的汇编语言是完全一致的。

本书的第十三、十四、十五章由上海交通大学李玉兴同志翻译，其余由上海机械学院陈哲庆同志翻译。上海机院黄寿根、董晓枫和机械工业部计算中心蔚强同志参加了本书部分章节的初稿翻译工作。全书由白英彩副教授审阅，在此表示衷心的感谢。

由于译者水平有限，如有翻译不妥之处，竭诚欢迎读者批评指正。

译 者

1986 年 11 月

前　言

VAX 系列超级小型计算机特别适合于汇编语言的学习。首先，VAX 机的结构代表着许多其他类型的计算机，VAX 指令系统包括了许多早期计算机，像 PDP-11 和 IBM360 等计算机指令系统的特点。熟悉 VAX 汇编程序的人会感到容易掌握那些机器中任何一种机器的汇编语言。

学习 VAX 汇编语言的第二个优点在于 VAX 机结构的复杂性。VAX 机有丰富的指令系统和寻址方式，这使其用汇编语言编程远比简单的计算机方便得多，明显的代价是增加了汇编语言学习的困难。然而 VAX 指令系统的正交性(orthogonality)对平衡它的复杂性很有帮助。例如，变址寻址可以作为一种编程概念来学习，然后可用于指定几乎任何一条指令的任何操作数。

本书的特点：

对学生来说，本书是最易读和最有用的教科书；对专业工作者来说，本书是迄今为止内容最丰富的一部参考书。

结构化程序设计：

本书始终强调结构化程序设计。建立伪代码(pseudocode)，然后将其翻译成汇编语言程序的设计方法，在书中作了详细的叙述，而且在所有的程序举例中加以运用。这种自上而下的设计方法增加了程序的可读性，而不影响其速度。

完整的程序示例：

几乎所有的程序例子都是独立的。三种整型量和字符串量在终端上的输入输出宏指令被用在几乎所有的程序例子的输入输出中。附录 C 是用到的输入输出宏指令的列表清单。它也包括了在终端上显示寄存器和内存单元的宏定义。

学习辅导：

每章有小结、新指令表和习题。几乎每章的习题都包括需要编程和不需要编程两种。

教学软件：

TOYCOM 仿真器和 VAX 输入输出宏指令包都备有磁带，收费很少。凡对上述软件有兴趣的人可与作者 (University of Colorado at Colorado Springs, College of Engineering)或当地 Benjamin/Cummings 出版公司的代理人联系。

VAX/VMS 调试程序：

整整一章是为 VAX/VMS 调试程序而写的，这一章描述了大部分有用的调试程序命令以及包括调试过程对话。

全部的指令覆盖：

本书描述了全部用户能使用的 VAX 指令(如果一门课不能讲完全部指令，可以根据要求选择一部分章节来讲)。

记录管理服务：

本书专辟一章讲记录管理服务。记录管理服务是 VMS 操作系统下的一个子系统。它是

VAX 高级语言用来处理输入输出的。

本书的使用：

这本书是根据 ACM 1978 Curriculum Recommendations 要求，作为计算机科学系三年级学生汇编语言编程标准课程的教科书。它要求读者至少已经学会一门高级语言编程的课程。

第一章描述了一台名叫 TOYCOM 的模型计算机的结构和汇编语言。TOYCOM 是一台十进制计算机，只有 12 条指令、100 个字(word)的内存和一个累加器。前面提过，有现成的 TOYCOM 交互式仿真器，所以学生可以编写和调试 TOYCOM 的汇编语言程序。第一章打算提供一些机器结构和汇编语言方面的简单介绍。对于程度较好的班级或者希望学生马上进入 VAX，这一章可以跳过，对以后章节内容的理解影响不大。

第二章包括了二进制和十六进制数，二进制和十六进制数的加减运算和它们之间的转换，以及二进制补码等所需要的材料。很明显，对于已有这方面知识的班级，这一章可以跳过。

书的重要部分是在三至十一章。在这些章节中，我们深入地讨论 VAX 的结构和指令、代码控制结构、VMS 调试程序、变址寻址、间接寻址、字符处理、子程序和宏指令等等。简言之，它们是 VAX 汇编语言编程的精髓。第十二至十五章完成全部 VAX 指令和记录管理服务 RMS 输入输出系统的讨论。

Robert Sebesta

目　录

第一章　一个模型计算机

本书旨在使读者将某种高级语言的基本编程技巧转化为对 VAX 系列计算机的结构和汇编语言的细致了解，尤其是要给学生以高超的汇编语言编程技巧。

人们有理由提出这样的问题，有什么必要花费相当多的时间去学习计算机的汇编语言编程呢？事实上，所有计算机都连同其相当好的系统软件一起出售给买主，而且几乎都包括现代的、结构化很好的高级语言的编译程序或解释程序。

回答如下：首先，虽然大多数系统软件是用高级语言写成的，但许多计算机应用程序仍需用汇编语言来写，最低层的输入输出驱动程序以及各种其他模块经常用汇编语言写成，因为写这种程序需要直接联系到机器的性能。这样的软件常常用面向具体机器的专用语言写成。为了有效地使用这类语言，就需要具有从学习该种机器的汇编语言中得到的各种经验。

为了优化程序，带有大量算术运算的数学分析程序的最里层的循环有时用汇编语言写成。在许多实时应用场合，或者对外界激励信号和精确的时间过程要快速反应的应用程序也只能用汇编语言写成。

要学生学习汇编语言的第二个理由是使得他们懂得机器的结构，否则这些知识很难获得。有了对计算机如何工作的基本了解，就为成为一个好的高级语言编程者提供了基础。

VAX-11 超级小型机是非常现代化的，它融合并提高了前一代计算机结构中大多数被肯定的特点（在这本书中我们如果仅指 VAX-11 系列中的一个，就用 VAX 描述），它们是相当复杂的机器，因此 VAX 汇编语言也相当复杂，正因为如此，我们将首先在一个很简单的计算机模型上讨论基本的机器结构和汇编语言编程。名为 TOYCOM（它是 toy computer——玩具计算机的首字母缩写词）简化了的计算机的介绍仅仅是为了模拟，它的仿真程序是用高级语言写的。通过 TOYCOM 的学习，可以学到许多关于计算机操作的特性和汇编级的编程方法。

1.1　TOYCOM 的总体结构

TOYCOM 仅作为数学上的工具，而没有其他用途。要把像 VAX 那样真实的现代化的计算机归结为 TOYCOM 需要抛弃大多数不方便的复杂细节，然而 TOYCOM 还保留了在比较复杂的计算机中使用户感到头痛的大部分风味和某些功能。

在开始讨论 TOYCOM 之前，让我们首先看一下图 1.1 的系统结构图。

TOYCOM 由内存、输入部分、中央处理单元(CPU)和输出部分所组成。在图 1.1 中，实线表示数据传送，虚线表示控制信号。CPU 中含有指令寄存器(IR)、程序计数器(PC)和累加寄存器(ACC)。寄存器是特殊的高速存贮器或存贮单元，与内存单元有相同的字长，这将在下面讨论。对许多 TOYCOM 指令而言，累加器既是操作数源又是其目的，实际上许多操作都是在累加器中进行的。

TOYCOM 的内存是由一系列称为字的存贮单元组成的，每个字能存放一个带符号的 4

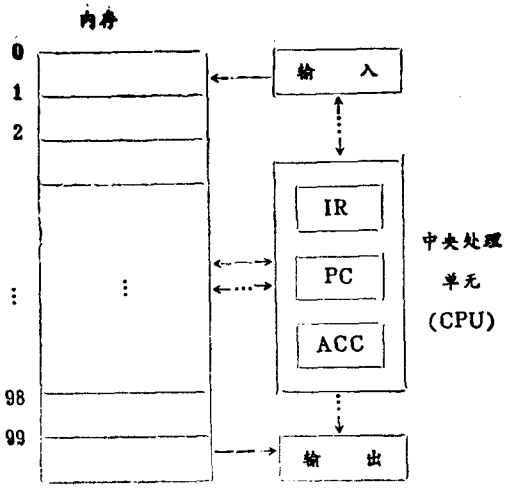


图 1.1 TOYCOM 系统结构图

位十进制数，所以TOYCOM 能存放的数的范围是－9999 到 ＋9999。因为小数点无法贮存，所以不能放小数。假若 TOYCOM 是一台真的计算机，这一点是致命的局限。 TOYCOM 的内存能贮存 100 个字，由整数 0,1,…,99 来寻址。

TOYCOM 的输入输出部件实际上被组合成一个装置，即终端。输入是通过键盘实现的，而输出是通过显示器或打印机实现的。

1.2 机器语言指令

计算机能理解的全部指令相对来说是一个很小的、相当简单的操作集。大多数高级程序语言的语句对计算机来说太复杂而不能被直接执行，所以用高级语言写成的程序，在被执行之前必须翻译成机器语言。

机器指令几乎总是由逻辑上分开的两部分组成：操作码和操作数。操作码不过是区别指令的助记名，而操作数经常指的是一个或几个内存地址。从最简单的意义上说，操作码告诉机器要作什么，操作数指出用什么去做。

尽管 TOYCOM 比现代计算机简单得多，但它们的指令结构是很相似的。不过由于 TOYCOM 设计得很简单，所以它的指令系统非常小，指令功能也很简单。

1.3 贮存式程序的概念

在看 TOYCOM 的指令之前，需要先看一下 TOYCOM 是如何处理程序的。和大多数数字计算机一样，TOYCOM 是一个存贮程序式计算机，它只能执行在内存中的指令。存贮程序式计算机的典型过程是，首先把全部程序读入内存，然后执行程序中的指令。另一种方式是计算机从输入装置中读一条指令，然后就执行这条指令，以后重复这一过程直到遇到 STOP指令为止。用贮存式程序的一个令人信服的理由是，机器执行指令的速度比输入装置能够提供指令的速度快得多。从内存中取一个字比从输入装置中去取一个字要少花很多时间，所以很自然地要将程序放到内存中执行。用贮存式程序的另一个重要理由是，几乎所有的程序都要碰到

指令循环的处理问题。如果指令是机器读一条执行一条的话，那末每循环一次，指令必须被重新读一次，50年代初期的计算机实际上就是采用一种称为可编程的卡片穿孔方法输入程序的，这是一种麻烦的方法。如果指令存放在内存中，只要机器有不按顺序执行指令的机能，循环就没有问题。

当贮存式程序的概念成为所有数字计算机基础的时候，在某些程序设计语言中引起了相当普遍的编程问题。因为指令和数据存放在同一内存中，机器不能把两者区别开来。程序能够自我操作，也就是说，机器如果能够做一个操作改变内存中的数据．它也能够做同样的操作改变内存中的指令。这是一种有用的功能，但也潜伏着一种危险性：程序可能会发生编程者不希望发生的自我修改，程序的这种无意的修改或指令的破坏一度曾是程序出错的主要原因。许多现代的程序语言，像Pascal语言，。有保护措施防止这类问题的发生低级语言，像 VAX汇编语言，往往没有保护措施，它们确实提供了程序自我修改的能力，但随意的自我修改是没有意义的。

1.4 TOYCOM的指令系统

在TOYCOM的指令中，操作码总是两个十进制数，它们贮存在内存的前半个字中，TOYCOM指令的操作数几乎总是内存地址，它们也是两个十进制数字。TOYCOM用后半个字中的两个十进制数字作操作数地址。指令的一般形式是AABB，其中AA是操作码，BB是内存地址操作数。

在高级语言中，程序由一组语句组成，而TOYCOM的机器语言程序则是一组四个十进制数字的序列。因为TOYCOM是一种存贮程序式计算机，它无法决定所给的字究竟是指令还是数据。在执行过程中，它所碰到的字，机器都认为是指令，由执行过程中，机器的顺序控制功能和机器的代码决定所遇到的是哪种字。

我们需要一些工具把数据输入TOYCOM和从TOYCOM中输出。指令IN使得TOYCOM在终端上打印或显示一个问号，然后等待用户打入一个数。它的操作码是07。当打入一个数并按回车键后，TOYCOM就把那个数存入指令操作数栏中指定的内存地址中。例如指令0742，使TOYCOM从键盘上取一个数，把它存到内存地址42号中。注意，把一个数存入内存单元的动作，不管是用IN指令或其他手段，总是破坏掉那个单元内先前的数值。

指令OUT以"OUTPUT = 数字"的格式打印或显示内存中一个数。例如，在内存地址22中存有一个数427，指令0822将打印OUTPUT = 427。

注意输入输出指令都不影响累加器，同样，打印某内存地址中的内容并不改变该内存中的值。

其他两条传送指令使数据在累加器和内存之间传送。LOAD指令（操作码是01）从一个指定的内存地址中将一个数复制到累加器中，LOAD指令并不影响那个被取数的内存地址中的内容，那个数的拷贝被传送了。

STORE指令(操作码是02)的功能与LOAD相反，它把累加器中的内容复制到指定的内存地址中去，同样，原单元(这种情况下是累加器)不受影响。LOAD和STORE两条指令的操作示于图1.2。

TOYCOM的算术指令是简单的，而且彼此很相似，它们都用累加器作第一操作数的隐含地址，第二个操作数总是指定的内存地址。操作的结果总是放在累加器中，破坏了第一个操作

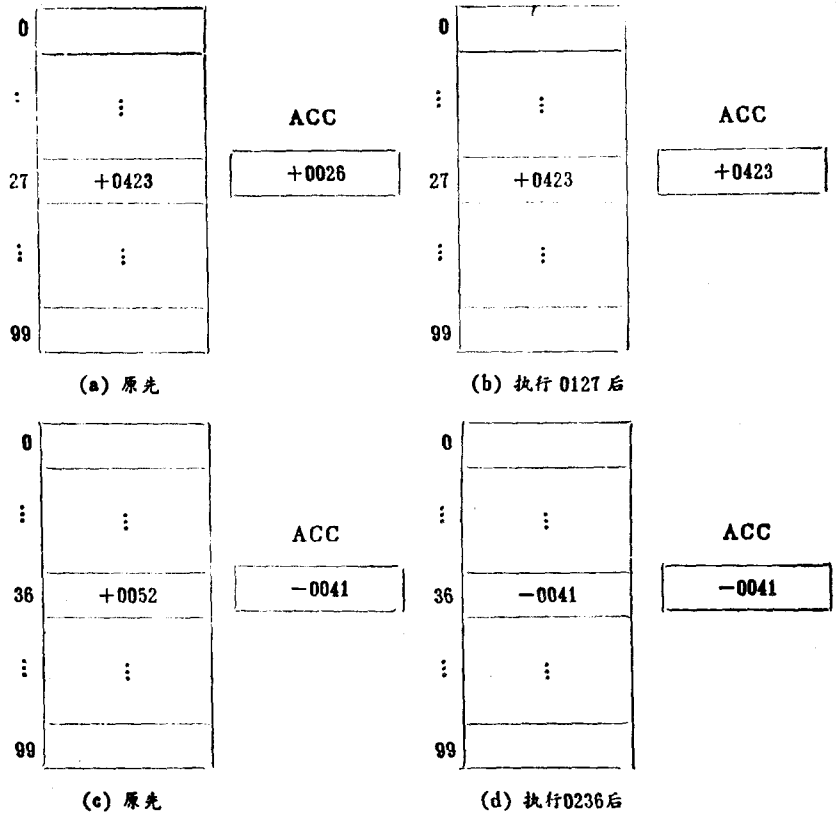


图 1.2 LOAD和STORE操作示例

数。例如看下面 ADD 指令(ADD 指令的操作码是 03)：

0327

这条指令把地址为 27 的单元中的内容加到累加器中。在高级语言中，这个操作可以写成

ACC: =ACC+C(27)

本书中，记号 C(地址)表示地址中的内容，所以C(27)表示地址为27的内存单元中的内容。

其他算术指令完全以相同的方式操作。减法指令 SUBTRACT 的操作码是 04，乘法指令 MULTLPLY 的操作码是05，除法指令 DIVIDE 的操作码是 06。TOYCOM 的 STOP 指令有点特殊，它不做什么，代码是 0，可以写成 0000，或简单地写成 0。TOYCOM 还有一些其他指令，我们打算等用那些已经学过的指令编写一个简单程序后再加以说明。

用 TOYCOM 机器语言程序解决的第一个问题是两个数相加：

例 1.1

：问题

In： 两个数。

Out： 输入的和。

创立解题的程序分两步进行。首先必须建立一个算法，然后把算法翻译成计算机所用的语言。虽然直接用程序语言解题常常是令人向往的，但是最好把它们分为两步。

用称为伪代码的一种结构化的英语使解决问题的算法得以最好的展开和表述。伪代码由控制语句(常常借助于某种现代高级语言)和非控制语句以英语的缩写形式混合组成。伪代码

不能在真实的计算机上执行，它们的唯一目的是用简化了的形式表达对编程问题的解。由于例1.1不需要代码控制语句，因此控制语句将在后面讨论。

∵伪代码解

取得输入(两个整数)

计算 SUM

打印 SUM

程序结束

伪代码解的简单也反映了问题的简单，但是这个几乎没有价值的例子仍然反映了非控制语句的概貌。

看来用已经描述过的指令系统来写 TOYCOM 的机器语言程序好像很容易，其实那些指令只是 TOYCOM 全部指令的一部分。“取得”和“打印”语句分别简单地用 IN 和 OUT 指令，“计算”操作就是处理已有的东西。一个有些微妙的操作也许是程序中变量存贮地址的分配，我们必须决定哪个机器内存地址被分配给伪代码中的变量。在上面的情况中，伪代码甚至没有指定标识或名字给需要放输入数据的变量，我们命名这两个变量为 FIRST 和 SECOND，唯一一个准备放和的变量被命名为 SUM，和伪代码中表述的一样。

TOYCOM 程序被放在从地址 0 开始的内存中，执行总是从地址 0 开始。在已经讨论过的指令范围内，指令总是顺序执行的，所以第二条要执行的指令是在地址 1，如此等等，直到遇到 STOP 指令为止。

假定把变量地址分配在内存的高端(具有最大地址数的区域)，这样它们将尽可能地远离指令区。在这个例子中，我们把 FIRST 分配为地址 99，SECOND 为地址 98，SUM 为地址 97。注意，机器不需要知道伪代码中所用的变量名。变量名在 TOYCOM 的机器语言程序中没有什么用处。

现在我们准备着手编写例 1.1 中真正的程序代码。

程序的最前面两条指令是输入指令，它们对应于伪代码解中的第一行。

```
0799
0798
```

这些指令把第一个输入数传送到地址为 99 的单元之中，把第二个输入数传送到地址为98的单元之中。

下一步是把伪代码解中“计算 SUM”行翻译成机器语言。在普通的表达式中，算术运算往往是二元的，也就是说，包括了两个操作数；此外，两个操作数都在内存中。如果把这个情况和 TOYCOM 的算术指令比较，可以看到基本的差别在于一个操作数的源。TOYCOM 的机器指令总是要求第一个操作数在累加器中，所以必须首先把数从 FIRST 移到累加器中，加完以后，余下要做的事是把结果送回 SUM 中，这个操作对任何语言都是必不可少的。那末所需的指令是：

```
0199
0398
0297
```

这三条指令序列对完成一个算术运算的任务是有代表性的。

程序中余下的操作是输入和结果的打印，后面再跟 STOP 指令。下面是完整的 TOYC-