68000汇编语言程序设计

格里·凯恩

〔美〕 杜戈·霍金斯 著

兰斯·利文撒尔

张洪 万秀芝 张俊荣 黄晓明 刘绍富 译

王 军 校

国防工业出版社

68000汇编语言程序设计

格里·凯恩
〔美〕 杜戈·霍金斯 著
兰斯·利文撒尔
张 洪 万秀芝 张俊荣 黄晓明 刘绍富 译
王 军 校

国防工业出版社

内容简介

68000是一种先进的、功能很强的新一代16位微型机。本书系统地阐述了68000汇编语言程序设计基本知识。全书正文五篇附录一篇。第一篇论述了汇编语言程序设计基本概念，包括特点、基本规则、格式等，并与高级语言进行了比较。第二篇及第三篇以微型机的实际典型任务为例，详细说明了如何编写汇编程序。第四篇论述如何把具体计算机任务变成计算机可执行的程序。第五篇详细说明了68000微型机的指令系统、寻址方式和指令执行图。第六篇为附录，是指令系统小结。

本书的主要特点是理论密切联系实际，并且程序实例都是选择典型的经常使用的任务。同时附有大量习题。因此，本书的基本思想和设计技巧，对其它微型机也是有参考价值的。

本书可供从事微型计算机科学技术的科技人员和大专院校有关专业的师生阅读参考。

68000 ASSEMBLY LANGUAGE PROGRAMMING

Gerry Kane

Doug Hawkins

Lance Leventhal

OSBORNE/McGraw-Hill, 1981年

*

68000 汇编语言程序设计

格里·凯恩

[美] [illegible] 著

张 洪 万 [illegible] 译

国防工业出版社出版 发行

(北京市海淀区紫竹院南路23号)

新华书店经售

国防工业出版社印刷厂印装

*

787×1092 1/16 印张$33^1/_2$ 784千字

1990年8月第一版 1990年8月第一次印刷 印数：0,001—2,565册

ISBN-7-118-00311-5/TP·35 定价：17.00元

译 者 序

本书译自美国 Osborne/McGraw 出版公司 1981 年出版的 68000 Assembly Language Programming 一书。作者为 Gerry Kane，Doug Hawkins 和 Lance Leventhal。

本书清楚地阐明了 68000 微处理机的指令系统和它的汇编语言程序设计。通过一些简单的程序介绍了汇编语言程序设计技巧和方法。借助于精心选用的例题，阐明其发展。这些例题包括了日益精巧的程序设计技术，例如循环结构、各种表格和使用、各种循环程序的使用和各种输入输出方法。对每一阶段的解释都用许多恰当的例子加以阐明。

全书正文五篇，共二十二章，另外还有附录。第一篇主要讨论汇编语言程序的主要特征，汇编语言格式和汇编程序的一般规则，并着重讨论 68000 汇编语言指令功能。第二篇和第三篇以大量的微型计算机典型任务为实例，详细说明编写汇编语言程序的基本方法。第四篇着重叙述怎样把具体的微型机任务，例如控制任务、数据处理任务、数字计算任务等变成可执行的程序。从问题的定义开始，包括程序设计的一般方法、查错和调试以及最后形成文件。第五篇详细讨论了 68000 的指令、寻址方式和指令执行图。第六篇为本书附录，是 68000 指令系统小结，包括 68000 指令码，所需的存储单元、指令执行时间以及 68000 指令目标码。

MC68000 是一种先进的、功能很强的十六位微型机，号称十六位微型机的工业标准。这本书对于从事微型机使用和教学的工作者是一本必备的参考书。

参加本书翻译的有张洪（第一篇、第六篇）、黄晓明（第二篇）、张俊荣（第三篇）、刘绍富（第四篇）和万秀芝（第五篇）。任元兰、卓小越、易新春三位同志对本书的译文提出了许多宝贵意见。全部译文经王军同志进行了仔细校阅。由于我们水平有限，译文中难免有不妥和疏漏之处，敬请读者批评指正。

译 者

作者介绍

格里·凯恩（Gerry Kane）是奥斯鲍恩/麦克劳（Osborne/McGraw）出版公司的技术人员，是著名的多卷丛书“微型计算机引论”的作者之一。最近，他为新的Osborne手册丛书写了其中“CRT控制器手册”和“68000微处理器手册”两本书。格里·凯恩在国立海岸警卫学院（United States Coast Guard Academy）获得了理科学士学位。

杜戈·霍金斯（Doug Hawkins）是亚利桑那州凤凰城（费尼克斯）的凤凰（Phoenix）数字公司工程部副总经理，负责设计和实现以微处理机为基础的分布式工厂监视和过程控制系统。在此之前，霍金斯先生在莫托洛拉（Motorola）微系统公司工作，是语言系统经理，也是MC68000的主要创始人。杜戈·霍金斯先生在密执安州州立大学获得电子工程理科学士，并在亚利桑那州州立大学获得电子工程科学硕士和工商管理硕士学位。

兰斯·利文撒尔（Lance Leventhal）是设在圣地亚哥的仿真（Emulative）系统公司的一个股东。该公司是微处理器和微程序设计方面的咨询服务公司。兰斯·利文撒尔本人是IEEE全国性微处理器方面的主讲人，也是象“仿真和微型计算”这样一些出版物的经常投稿人。利文撒尔先生在微处理器方面有十本专著和六十多篇文章。他也是“计算仿真”的技术编辑，“数字设计”的特约编辑。

利文撒尔先生曾写了有关汇编语言程序方面的许多书，并着手写这方面的新丛书。他在圣路易市华盛顿大学获得了文科学士学位，在圣地亚哥的加利福尼亚大学获得了理科学士和哲学博士等学位。他也是计算机仿真学会（SCS）、美国计算机学会（ACM）、电气和电子工程师协会（IEEE）、电气和电子工程师协会计算机学会（IEEE Computer Society）的会员。

目　录

第一篇　基本概念

第二篇　问题的提出

第三篇　深入的论题

第四篇　软件开发

第五篇　MC68000指令系统

第六篇　附　录

第一篇　基本概念

本书讲述汇编语言的程序设计。作者假定你已通晓《微型计算机引论》一书的第一卷——基本概念(An Introduction to Microcomputers, Berkeley, Osborne/McGraw-Hill, 1980)。特别是其中第6章和第7章与本书关系密切。本书不再讨论计算机和微型计算机及其寻址方式和指令系统等一般特点。有关这方面的材料可参阅《微型计算机引论》第一卷。

本篇内的各章概括地介绍了汇编语言基本知识，并具体地介绍了MC68000汇编语言。第一章讨论汇编语言的用途并把汇编语言同计算机高级语言进行比较。第二章讨论汇编程序，并简单介绍装入程序。第三章讨论MC68000微处理机的结构，并把它与类似的处理机进行比较，进而讨论了莫托洛拉（Motorola）公司MC68000汇编程序的主要特征。

第一章　汇编语言程序设计概论

计算机的程序基本上是一系列的数，因而它很少为人们所理解。本编我们将讨论把计算机程序表示成为类似人们语言的那些计算机语言，我们也将讨论为什么要引入汇编程序以及汇编程序的用法。这些是本书的主题。

指令的意义

微处理机的指令系统是使微处理机在指令周期期间产生规定动作的一组二进制输入量。指令系统与微处理机的关系就好象功能表与门电路、加法器或移位寄存器这些逻辑器件的关系。不过，微处理机响应它的指令输入值时所执行的动作远比逻辑设备响应它的输入值时所执行的动作要复杂的多。

二进制指令

指令是一些二进制数的组合，它必须以数据的形式在适当的时间输入到微处理器中，以便将其解释成为一条指令。例如，在指令存取操作期间，当MC68000微处理器接收到16位二进制形式的数字1101001100000000作为输入时，该模式的意思是："把数据寄存器$D0$的内容加到数据寄存器$D1$中去。"

类似地，模式0001000000111010000000011111111的意思是："传送11111111到数据寄存器$D0$中去。"

象其它计算机一样，微处理机只能识别二进制形式的指令或数据，它并不认识字符、八进制和十六进制的数。

计算机程序

程序是使计算机执行特定任务的一串指令。

事实上，计算机程序不仅包括指令，它亦包括数据和存储器地址，这些是微处理机完成指令规定任务所必需的部分。显然，如果微处理机要作加法运算，就必须有要相加的两个数及存放相加结果的地方。计算机程序必须确定数据源和结果的去向，从及规定要进行的运算。

除某条指令改变了执行顺序或使计算机停机外，所有的微处理机都按顺序执行指令。这就是说，只要现行指令没有特别指定处理机要执行其它操作，处理机就按顺序从下一个较高的存储器地址内得到下一条指令。

归根结底，每个程序都是一组二进制数。例如，有一个 MC68000 程序，把存储器单元 6000_{16} 和 6002_{16} 的内容相加并把结果放置在单元 $(6004)_{16}$：

```
0011000000111000
0110000000000000
1101000001111000
0110000000000010
0011000111000000
0110000000000100
```

这是机器语言程序或叫目标程序。若将该程序送入以 MC68000 为基本组成部分的计算机的存储器中，该微型计算机便能直接执行这个程序。

二进制程序设计问题

要编制目标程序或二进制机器语言程序有许多困难，问题是：

- 这种程序难于看懂和调试（你会把它们全看作二进制数，特别在最初几小时更是如此）。
- 因为对每一位要拨动前面板开关，并且每次只能存一个字到存储器中，故程序输入很慢。
- 程序没有采用人们可读的格式来描述计算机执行的任务。
- 程序太长，写起来很费力。
- 程序员经常会由于粗心而出错，这种错误难以查明和纠正。

例如下面加法目标程序有一位是错误的，请试图找出这个错误：

```
0011000000111000
0110000000000000
1100000001111000
0110000000000010
0011000111000000
0110000000000100
```

虽然计算机容易处理二进制的数，但是人却不易处理。人们感到，二进制程序太长、麻烦、容易混淆和没有意义。长期下去，程序员可能记住一些二进制码，但却是事倍功半的事。

采用八进制或十六进制

用八进制或十六进制而不用二进制数来写指令，则情况会有一些改进。在本书中我们将采用十六进制数，原因是十六进制数较短，而且它们也是微处理机工业的标准。表 1-1 定义了十六进

表1-1　十六进制数的变换表

十六进制数字	相应二进制数字	相应十进制数字	十六进制数字	相应二进制数字	相应十进制数字
0	0 0 0 0	0	A	1 0 1 0	10
1	0 0 0 1	1	B	1 0 1 1	11
2	0 0 1 0	2	C	1 1 0 0	12
3	0 0 1 1	3	D	1 1 0 1	13
4	0 1 0 0	4	E	1 1 1 0	14
5	0 1 0 1	5	F	1 1 1 1	15
6	0 1 1 0	6			
7	0 1 1 1	7			
8	1 0 0 0	8			
9	1 0 0 1	9			

制数字及与它们对应的二进制数。现在两个数相加的 MC68000 程序就变为：

```
3038
6000
D078
6002
31C0
6004
```

至少用十六进制数字写的程序较短，检查起来也不太麻烦。

用十六进制数字写的程序其中错误比较容易发现，这时有错误的加法程序其十六进制形式为：

```
3038
6000
C078
6002
31C0
6004
```

错误就更明显了。

微处理机只识别二进制的指令码。那末我们如何处理这种十六进制的程序呢？如果机器的前面板有十六进制键而不是二进制的位式开关，你就能够用按键把十六进制程序直接输入到存储器中去——键盘逻辑把十六进制数翻译成二进制数。但是，如果前面板只有位式开关那将怎么办呢？你自己可以把十六进制数转换成二进制数，但这是一种重复而麻烦的工作。试图这么做的人们会犯各种各样的细小的错误，例如看错了行、漏掉了一位或者将数字或二进制数位的顺序弄颠倒了。此外，只要我们自己转换十六进制程序，就必须不断地通过前面板开关把这些二进制位放置到存储器中去。

十六进制装入程序

然而，这些重复的、极度紧张的工作用计算机来完成则是十分理想的。计算机永远不会感到疲劳或厌烦并且几乎不犯错误。人们希望写出这样一种程序，它接受十六进制数，将其转换成二进制数并放在存储器中。这是许多微型计算机所具备的一种标准程序，称之为十六进制装入程序。

象任何其它的程序一样，十六进制装入程序也是一种程序，它占有存储空间。在一些计算机系统中，它驻留在存储器中，其保存时间是足以装入其它程序的；在另一些系统中，它存在存储器中为只读存储器保留的区域。在微型计算机的前面板上也许没有位式开关，甚至连前面板也没有。这反映出机器设计者认为二进制程序设计不仅令人讨厌而且完全不必要了。你的计算机系统的十六进制装入程序可能是很大的监控程序的一部分。监控程序还为程序调试和程序分析提供若干工具。

当然，十六进制装入程序并不能解决程序设计中的一切问题。用十六进制数写的程序仍然难读和难于理解。例如，我们既不能区分出地址操作和数据操作，又不能从程序清单上看到程序要干什么的提示。3038 或 31C0 是什么意思？要记住卡片上的全部编码几乎是一件枯燥无味的事情。而且，不同的微处理机又有完全不同的编码。此外程序将要求编制大量的说明文件。

指令码的助记符

程序设计的明显改进是赋给每一个指令码一个名字。人们把指令码的名字称作“助记符”或记忆唤起符。指令助记符应该以最少的字符来描述指令是干什么的。

助记符设计

事实上，微处理机制造厂家对其微处理机的指令系统（它们不能用十六进制来记忆）提供一组助记符。你不一定要用制造厂家所提供的助记符，这种助记符不是一点不能改动的。但是制造厂所提供的助记符对于某一给定的微处理机是标准的，因而可以被所有的用户所理解。这些是你能够从手册、卡片、书籍、论文和程序中找到的指令代码。选择指令助记符的问题是，并非所有的指令都有“一看就明白”的名字。有些指令的确如此（如 ADD，AND 和 OR）。另一些指令采用明显的缩写（如减法（subtraction）用 SUB，异或（exclusive-OR）用 XOR）。可是仍有一些指令例外，如 WMP，PCHL 以及 SOB 就属于这种情况。绝大多数厂家提供的名字有些是合理的但也有些是令人失望的。但是，若用户创建自己的助记符，也不见得比制造厂起的名字好多少。

标准助记符

有一个汇编语言助记符集的推荐标准[1]。所能采用的助记符总量不作规定，但能作为对指令系统进行比较的基础，和作为未来的微处理机选择助记符的基础。

与指令助记符一道，制造厂也常常赋给 CPU 的寄存器一些名字。正如指令名字一样，有些寄存器的名字一看就懂（例如用 A 表示累加器（Accumulator）），而另一些名字则是习惯叫法。我们将仍然采用制造厂家起的名字，以期促进命名的标准化。

汇编语言程序

如果我们使用莫托洛拉公司制定的 MC68000 指令和寄存器的标准助记符，则 MC68000 的加法程序为：

```
MOVE    $6000,D0
ADD     $6002,D0
MOVE    D0,$6004
```

这个程序还不能一看就懂，但至少某些部分是容易理解的。ADD 比 D078 容易理解，MOVE 助记符规定把数据传送到一个寄存器或存储器单元中。可以看出，程序的一些部分是操作，另一些部分是地址。这种程序就是汇编语言程序。

汇编程序

我们如何将汇编语言程序放到计算机中去呢？必须将汇编语言程序翻译成为十六进制或二进制数。你可以用手工方法翻译汇编语言程序，此时将指令逐条地进行翻译，这称作手工汇编。

下面列出加法程序的手工汇编：

指令助记符	寄存器/存储单元	相应十六进制数
MOVE	$6000,D0	30386000
ADD	$6002,D0	D0786002
MOVE	D0,$6004	31C06004

就象把十六进制数转换成二进制数一样，手工汇编也是一种枯燥的、重复的并且会引起许多小错误的工作。例如，取错了行、颠倒了数字、遗漏了指令及读错了代码等是你可能犯的一些错误。绝大多数微处理机都用不同长度的指令使得上述工作更趋复杂。一些指令长度为一个字，而另一些指令却为两个字或三个字。有些指令要求数据在第二个或第三个字里，而另外一些指令却要求存储器地址、寄存器号或其它什么。

汇编是一件枯燥的工作，我们可以让微型计算机来做。翻译代码时，微型计算机从来不会犯任何错误，它总是知道每条指令需要多少字和要求怎样的格式。从事这个工作的程序叫作汇编程序。汇编程序把用助记符写的用户程序或“源”程序翻译为微型计算机能够执行的机器语言程序或“目标”程序。汇编程序的输入是源程序，而输出是目标程序。

正如十六进制装入程序一样，汇编程序也是一种程序。不过，汇编程序写起来更困难，占据的存储器空间更大，需要更多的外部设备和更长的执行时间。用户常常可以写自己的装入程序，但很少考虑编写自己的汇编程序。

汇编程序有它们自己的规则，读者必须掌握它。这些规则包括在适当的位置使用某种记号(如空格、逗号、分号和冒号)、正确地拼读、信息的适当控制以及正确地放置名字和数字。这些规则通常很简单并且很快就能掌握。

汇编程序的其它特性

早期的汇编程序只不过是把各种指令和寄存器的助记符翻译为相应的二进制码。然而，现在的绝大多数汇编程序都有如下特性：

- 允许用户给存储单元、输入和输出设备甚至指令序列起名字。
- 把各种数制系统的数据和地址（例如十进制数或十六进制数）转换为二进制数并且把字符转换为它们的 ASCII 码或 EBCDIC 码。
- 作为汇编过程的一部分而进行的某些算术运算。
- 告诉装入程序应把程序和数据存放在存储器中的哪个部位。

· 允许用户在规定的存储区存放暂存数据以及固定数据。
· 提供必要的信息给现行程序中所包含的程序，如程序库的标准程序或以前写好的程序等。
· 允许用户控制程序清单格式和所使用的输入输出设备。

选择汇编程序

当然，所有这些特性将造成成本和存储量的增加。微型计算机的汇编程序通常比大型计算机的汇编程序简单得多，但总的趋势是汇编程序的规模在增加。你常常要对各汇编程序进行选择，重要的准则不是汇编程序有多少特性而是在实际中该汇编程序的使用是否方便。

汇编语言的缺点

就象十六进制装入程序那样，汇编程序并不能解决程序设计中的所有问题。问题之一是微型计算机的指令系统与微型计算机要执行的任务之间有很大的差距。计算机指令能做的事情是：把两个寄存器的内容相加，把累加器中的内容移一位或者把新的值放入程序计数器中，等等。而用户通常希望微型计算机做的事情是：打印一个数，对电传打字机来的特殊命令进行搜索并作出相应的反应或在适当的时间启动继电器等。汇编语言程序员必须把后一种任务转化为一系列简单的计算机指令序列。这种转化是一件困难的、花费时间的工作。

问题之二，如果要用汇编语言进行程序设计，你就必须详细了解所用的那台微型计算机的性能。必须知道该微型计算机有什么样寄存器和指令，并精确地知道指令如何影响各种寄存器，及该计算机采用什么寻址方式等其它很多知识。这些知识都与微型计算机最后必须完成的任务没有直接关系。

不能移植

此外，汇编语言程序不能移植。每一种微型计算机都有与它的结构相适应的汇编语言。为MC68000编写的汇编语言程序就不能移植到6809，8080或Z8000等微处理机上。例如，为Z8000编写的加法程序是：

```
LD  R0,%6000
ADD R0,%6002
LD  %6004,R0
```

不能移植，不仅意味着你不能在不同的微型机上使用你的汇编语言程序，而且也意味着不能使用那些不是专门为所使用的微型机而写的程序。这对于象MC68000这样的十六位微型计算机是一个典型的缺陷。因为这种机器是新问世的，可供它们使用的汇编语言程序很少。这就要取决于你自己了。如果你需要一个程序来完成某一具体任务，你不太可能在绝大多数厂家所提供的小程序库中找到它；也不可能在档案、期刊文章或其它人的老的程序文件中找到它，那你就不得不自己动手编写程序。

高级语言

解决与汇编语言程序有关的许多困难，其办法是用“高级语言”或“面向过程”的语言来代替汇编语言。高级语言允许你用面向问题的形式而不是面向计算机的形式来描述任务。高级语言的

每一个语句都执行一种可识别的功能，它通常对应着许多条汇编语言指令。由所谓编译程序把高级语言源程序转变为目标码或机器语言指令。

FORTRAN——一种高级语言

对不同类型的任务有许多不同的高级语言。例如，如果能用代数符号来表示你想要计算机完成的任务，你就可以用 FORTRAN（公式翻译语言）来编写你的程序。在高级语言中，它是最老的和使用最广泛的高级语言。现在，如果你想把两个数相加，你只要告诉计算机

SUM＝NUMB1＋NUMB2

就可以了。这比相应的机器语言程序或相应的汇编程序都简单的多（程序短了许多）。其它的高级语言有 COBOL（商务语言）、PASCAL（用于结构程序设计）、PL/I(FORTRAN和 COBOL 的综合)、APL（用于编写很紧凑的程序）、BASIC（用于较小的微型计算机的大众语言）和C（由贝尔电话实验室开发的系统程序设计语言）语言等。

高级语言的优点

很明显，使用高级语言可以更容易、更快速地编写出程序。通常估计，程序员用高级语言比用汇编语言编写程序快10倍[2]~[4]。这仅仅是指编写程序一项，尚不包括问题定义、程序设计、程序调试、查错或文件编制等项目。用高级语言时，这些都变得比较简单和迅速。例如，高级语言程序在一定程度上是自我说明的文件。即使你不懂 FORTRAN 语言，你也可以看出上述语句的含意。

与机器无关

高级语言解决了汇编语言程序设计中其它许多问题。高级语言有自己的语法（通常由国家或国际标准定义）。高级语言并不涉及指令系统、寄存器或特定计算机的其它性能。编译程序对于这一切都仔细地进行了考虑。程序员可以把精力集中到自己的任务上，他们不需要了解作为基础的CPU本身结构，也不必了解计算机的知识就可以进行程序设计。

可移植性

用高级语言写的程序是可以移植的，至少在理论上是这样。高级语言可以移植到配有该语言编译程序的任何计算机上。

同时，当你对一台新的计算机进行程序设计时，你可以利用先前用高级语言为其它计算机编写的程序。这就意味着，象 FORTRAN 或 BASIC这样的通用语言有成千上万个程序可供使用。

高级语言的缺点

既然高级语言有上述那么多的优点——即可以较快地编写程序，又便于移植——那么为什么还要用汇编语言来编写程序呢？有谁愿意不厌其烦地去操心有关寄存器、指令码、助记符及所有这些无用的东西呢！象任何事物那样，高级语言有优点必然也有缺点。

语法

象汇编语言一样，一个明显的问题是你必须学习任何一种你想使用的高级语言的“规则”或“语法”。每一个高级语言都有颇为复杂的一套规则。你会发现，想要获得一个符合正确语法的

程序（即使这样，也未必满足你的要求）你得花费大量的时间。一个高级计算机语言如同一门外语一样。如果你有能力，你就会逐渐习惯于这些语法规则，而且能设计出编译程序可以接受的程序。然而，学习这些语法规则和设法得到编译程序可接受的程序对你正从事的工作并没有直接的帮助。

例如，下面是有关 FORTRAN 编译程序的几条规则：

- 标号必须是放在卡片前 5 列的数字。
- 语句必须从第 7 列开始。
- 整型变量必须以字母 I、J、K、L、M或N开始。

编译程序的成本

另一个明显的问题是，需要一种编译程序来把用高级语言编写的程序转变成为机器语言程序。编译程序的价格昂贵而且要占用大量的存储器。编译程序在存储器中占用 4 K～64K字节，而大多数汇编程序在存储器中占用 2 K～16K字节的存储器（1 K＝1024）。所以使用编译程序的总开销是相当大的。

任务与语言相适应

而且，仅有为数不多的编译程序能使得你的任务简化。例如，FORTRAN 最适用于那些用代数公式表示的问题。但你的任务如果是控制显示终端、编辑一串字符或是监视一个报警系统，这样的一些问题则很不容易用代数符号来表示。事实上，用 FORTRAN 列公式求解这类问题比用汇编语言列公式求解更困难也更棘手。答案是采用更适当的高级语言。为上面所述任务专门设计的高级语言确实存在——称为系统工具语言（system implementation languages）。然而，这种语言在使用广泛和标准化方面远不如 FORTRAN 语言。

效率低

高级语言不能产生很有效的机器语言程序。其基本原因是，编译程序是自动处理的，它要综合考虑各种可能性从而找出折衷的方案。编译程序的工作很类似计算机化的语言翻译器——有时词是对的，但句子结构难以理解。简单的编译程序并不知道，什么时候某一变量不再使用并可删去，什么时候应该使用寄存器而不使用存储器单元或者什么时候各变量间有简单的关系。有经验的程序员可操捷径以缩短程序执行时间或减少所使用的存储单元。有少数编译程序（称之优化的编译程序）能够做到这一点，但这种编译程序比常规编译程序大得多。

优缺点摘要

高级语言的优点是：

- 便于教学
- 更便于描述任务
- 编写程序所花的时间较少
- 易于编制文件
- 标准的语法
- 与特定计算机结构无关

· 可以移植
· 可利用程序库和其它程序

高级语言的缺点是：

· 有专用的规则
· 要较多的硬件和软件支持
· 通用语言有一个面向数学或是商务的问题
· 程序的效率低
· 最佳编码难以满足时间和存储量两方面要求
· 不能方便地利用计算机的特殊性能

微处理机的高级语言

当使用高级语言时，微处理机用户将遇到几个特殊的困难。这些困难是：

· 微处理机几乎不存在高级语言。这对于新型的、比较不大众化的或简单控制应用的微处理机尤为如此。

· 没有几种标准语言是广泛使用的。

· 编译程序通常需要大容量存储器甚至完全不同的计算机。

· 许多微处理机的应用不宜采用高级语言。

· 许多微处理机的语言不产生目标程序。即微处理机将程序转换一行就执行一行——这叫做解释而不叫做编译——或者产生一个输出，但需要专门的系统软件（一种运行——时间程序包）来执行。这两种方法都将使所执行的程序慢而又需大量的存储器。BASIC 和 PASCAL，这种普遍应用的高级语言，通常就采用这两种方法之一。

· 在微处理机应用中存储器成本常常是关键问题。

相对说来，微型计算机高级语言数量较少，这是因为微处理机的历史短，它们发源于半导体工业而不是计算机工业。在高级语言中，最经常使用的是 BASIC[5]、PASCAL[6]、[7]、FORTRAN、C[8]和PL/1 型语言如 PL/M[9]等。

现有的许多高级语言与所承认的标准文本不一致。这样，微处理机用户不能期待获得很多可移植的程序，不能访问程序库或使用以前的经验或程序。剩下的主要优点是：设计程序的工作量减少了、比较容易编制文件、不太需要详细了解你所使用计算机的体系结构。

高级语言的总开销

微处理机使用高级语言的总开销是很可观的。直到最近，微处理机较适用于控制和慢速交互系统而不太适用于编译过程中的字符管理和语言分析。因此，绝大多数微处理机编译程序不在以微处理机为基础的系统上运行。相反，它们需要一个更大型的计算机。也就是说，它们是交叉编译程序而不是自编译程序。用户不仅必须承担大型计算机的费用，而且还必须把大型计算机的程序转变成微处理机的程序。

已有几种自编译程序。这些编译程序能运行在微处理机上生产本机的目标代码。遗憾的是，它们通常需要大量的存储器以及专门支持的硬件和软件。

高级语言的适应性

高级语言一般不能很满意地适合微处理机的应用。所发明的大多数通用语言是为了帮助解决科学问题或者去处理大量的商务数据。许多微处理机的应用并不属于这两种领域。而是向输出设备发送数据和控制信息，或从输入设备接收数据和状态信息。控制和状态信息常用含义非常精确并与硬件相关的几种二进制数字组成。如果你试图用高级语言编写一个典型的控制程序，你就会感到象一个人用筷子吃汤一样为难。高级语言用于试验设备、终端设备、导航系统、信号处理和商业设备这类任务时，比其应用于仪表、通讯、外围设备和自动装置方面情况好得多。

高级语言的应用范围

高级语言比较适用于具备大量存储器的设备。如果在阀门控制器、电子游戏、设备控制器或小型仪表等中单片存储器的价格是重要的因素，那末高级语言在容量不足的存储器中使用是不能令人容忍的。另一方面，如象在终端或试验设备中，如果该系统中有成千上万个字节的存储器，则高级语言的容量不足问题就显得不那么严重了。很明显，程序的规模和容量也是重要的因素。大的程序将大大增强高级语言的优点。而应用大容量的存储器将意味着有限的软件开发成本不象作为系统一部分的存储器的成本那么重要了。

你应选择哪一级语言？

究竟选用哪一级语言取决于你的实际应用。让我们简略地评述一下也许有助于你选择哪一级语言。

机器语言的应用

事实上，没有一个人用机器语言来编写程序，因为机器语言既花费人力也难于编辑文件。汇编程序成本低并能大大减少程序设计时间。

汇编语言的应用

- 短至中等长度的程序
- 存储器成本是应用中一个重要因素
- 实时控制应用
- 有限数据处理
- 大批量应用
- 更多的用于输入/输出或控制而不是用于计算

高级语言的应用

- 长程序
- 小批量应用
- 需要大容量的存储器的应用场合
- 应用中涉及的计算任务比输入/输出或控制任务多