微型计算机接口设计

（原理和实践）

（上）

谢启江
宗儒 译校

苏州电子计算机厂
一九八四年三月

Microcomputer Interfacing

Principles and Practtces

G. Jack Lipovski
University of Texas

Lexington Books
D. C. Heath and Company
Lexington, Massachusetts
Toronto
1980

目　录

上　册

作者序 ……（1）
关于作者 ……（3）
译者序 ……（4）

第一章　微处理

1-1　微计算机引言……（5）
1-2　6809 和 6800 微处理器指令系统 ……（15）
1-3　几种程序设计技术……（43）
1-4　结束语……（71）

第二章　总线和信号

2-1　数字硬件……（78）
2-2　微处理机的控制信号和时序……（85）
2-3　总线接口……（95）
2-4　结束语……（104）
习题 ……（105）

第三章　并行输入和并行输出

3-1　输入/输出结构和简单的输入/输出接口……（110）
3-2　输入/输出软件 ……（116）
3-3　可程序的並行输入/输出…… ……（124）
3-4　结束语……（134）
习题 ……（135）

第四章　中断和交换

4-1　输入/输出的同步原理 ……（140）
4-2　同步交换……（145）
4-3　在 M6809 的中断机构 ……（158）
4-4　直接存贮器访问和文本开关……（163）
4-5　结束语……（171）
习题 ……（172）

第五章　模拟接口

5-1　输入和输出传感器……（177）
5-2　基本的模拟过程部件……（180）
5-3　用运算放大器和模拟开关的信号条件……（189）

5-4 键盘和 LEO 显示模件 ……(195)
5-5 转换器……(203)
5-6 数据采集系统……(212)
5-7 结束语……(216)
习题 ……(218)

下　册

第六章 计数器和定时器
6-1 M6840 模块 ……(224)
6-2 信号发生器……(226)
6-3 频率模拟量的测量……(240)
6-4 M6840 结构……(246)
6-5 结束语……(252)
习题 ……(254)
第七章 通信系统
7-1 通信原理……(260)
7-2 信号传输……(262)
7-3 UART 链路规程 ……(268)
7-4 其他规程……(281)
7-5 结束语……(286)
习题 ……(288)
附录 A ASCII 字符集……(292)
附录 B IM6402/6403 UART……(293)
附录 C MC14469 可寻址 UART ……(305)
附录 D M6821 (PIA) ……(314)
附录 E M6840(PMT) ……(333)
附录 F M6844(DMAC) ……(344)
附录 G M6850 (ACIA) ……(360)
附录 H M6852(SSDA) ……(377)
附录 I M6800(MPU)……(393)
附录 J M6809 ……(436)

作 者 序

微计算机的出现在许多方面改变了以前的工程设计，以至“微计算机革命”的说法广泛地出现在许多报刊文章和学术论文中。微计算机是一种工具，然而至今微计算机系统的设计却仍然不得不以微型计算机产品说明书作为依据。原因是至今还没有详细阐述微型计算机设计基本原理的教科书或参考书。本书旨在讲授微计算机系统设计的基本原理和实践，着重讲授接口的设计原理。

本书是作者在为大学高年级学生讲授“微计算机系统设计”的讲稿发展而来的。本书特别强调了微计算机系统设计软硬相结合的方法，着重讲述了设计原理。因为正如任何工程设计都需要有理论支持一样，计算机的理论对于为系统设计打好坚实基础来说是完全必要的。不仅如此，本书还同时强调了实践，即如何实现系统运行的具体方法。这是因为微计算机系统设计需要有直接经验。如果一个学生只读过几本有关微计算机的书，而另一个学生却曾搞过微计算机，这两者之间的差别可以清楚地证明理论必须结合实践来进行教学。实践经验在几乎所有的工程学课程中都是需要的，但并不总是可能的。但是微计算机系统却价格低廉得使学校或学生们能够得到这种直接实践的机会。学生们亲眼见到所学的原理能指导具体实践，他们所产生的喜悦是可以想象的。但至今学生们却得不到足够的资料。为此该课程和本书在着重讲述了微计算机设计的原理的同时，又强调了它的实践。

这些原理和实践包括了软件和硬件两方面的内容，纯粹的硬件的设计方法对于数字系统设计师或学电机工程的学生来说或许会更有吸引力。但是这种纯粹的硬件设计方法由于没有体现软件的功能或者没有强大软件的支持，使整个设计过分拘束，致使系统造价大大提高。与此相反，用纯粹软件的设计方法，由于不了解“如何”和“为什么”产生一些现象，因此也就不知道怎样进一步发挥系统的潜在功能。

使用软硬结合的方法确实需要更多的背景知识。作为本书基础的教材前两章是基本汇编语言和逻辑设计基础。本书将基本汇编语言和逻辑设计原理核心部分作为予备知识介绍给读者。如果读者没有学过上述两门课，也没有读过有关教材，那么这两章可以作为简明教材来学习。因为这两章是作为以后各章的准备知识，所以写得精炼、简明。

本书所说的实践是指对 Motorola M6800 微计算机系列和 M6809 微处理器指令系统的详细讨论而具体化的。这样的一些讨论，目的是为了讲授微型计算机接口的设计原理和设计实践，而不是用来推销、出售或使用 Motorola 的产品。具体而详尽的讨论能鼓励和帮助读者通过实际操作来进行学习。而且还能极大地激发读者对进一步理解原理的热情。我们之所以使用 M6800 系列主要是因为 M6809 被确认为最容易教的微计算机。M6809 的指令系统和其他任何机器一样完全。支持它的寻址方式和变址寄存器多得足以用来教会读者灵巧地使用数据结构，然而对 M6809 和其它一些设计得很好的微计算机进行详细比较以后，我们可以清楚地看出，对不同的应用，有些机器可能比 M6809 好。我们要对读者强调的是，我们选择 M6809 并不意味着 6809 对所有的应用都是一种最好的微计算机，不过，对于不同微计算机和不同应用的比较研究已超出了本书的范围。而 M6800 系列，特别是 M6809微计算机是讲授微计算机设计原理的一个极好的工具。

第一、二两章概括了本书后几章学习所需的背景知识。第一章讲述M6809的指令系统，这一章是为那些对其它微计算机或大型机的汇编语言有所了解的读者所作的概括。第二章涉及计算机的基本组织，不过仅仅局限于与微计算机关系特别密切的几方面。例如，在基本的计算机组织中，传统的是包括浮点运算。但是在这一章中并不介绍浮点运算部件。此外，这一章还论述了三态总线，但是一般在讨论微计算机组织时这个内容常常不包括进去。本书的其余部分涉及到三个基本内容：第三第四章并行输入、输出(I/O)口；第五第六章模拟部件和第七章的通信设备。简单的并行输入输出口在第三章中讨论。而且采用软件和硬件互相渗透的方法。这样，读者不至对基本接口的设计望而生畏。第四章讨论中断和中断交互，分析软件和硬件的交互作用；并且举例说明了不同的技术。第五章对微计算机输入输出系统中常用的传统(电压)的模拟量作了全面的论述。这一章还为读者提供了充分资料，使读者进一步了解模拟量的用途，局限性和优点，并且可以以这个为起点对模拟设计有更全面的了解。第六章把计数器定时器作为频率——模拟信号的接口介绍给读者。讨论不同频率信号的产生和测量技术，即模拟—频率转换技术。此外，还讨论和评价了用一些很有趣的LSI微处理支持芯片实现软硬件的交替。第七章描述了通信设备；全面论述了UART，SDLC和IEEE-488的通信技术

本书力求完整，假如读者希望略去每章结尾提出的问题和课文中建议做的实验的话，本书也已足够了。尽管如此，我们还是推荐读者认真考虑章末的习题和做一些实验。这样做，读者就会需要一本1980年6月1日以后出版的“Motorola微计算机数据大全”(The Complete Motorola Microcomputer Data Library)早先版本对M6809的叙述都不完整。大多数读者会得到这本书，或许他们还会要求得到一本进一步讲述与M6809接口的LSI芯片的数据手册。为此我们增加了两个附录，一个关于Intersil UART，另一个关于Motorola可寻址UART。这两个附录包含了数据大全中所没有的内容。读者还可做所列出的全部习题。也许读者还要参考得克萨斯仪表公司出版的TTL数据手册(The TTL Data Book for Design Engineers)，为此我们正在编辑一本实验手册(可望在1980年12月前出版)习题和解答，只要与作者联系就可得到。

此外我们还有必要对本书的风格作些说明。本书对于正式介绍的术语和术语的使用是尽可能地谨慎。书后还有一个索引以帮助读者找到这些术语的解释。至于那些概念上模糊的术语和一些未经定义的行话，书中一概就不列入了。微计算机系统设计和接口设计发展迅速，因而在专业杂志和产品使用说明中所引用的术语很不一致，甚至常常互相矛盾；但作为教科书中所用的术语应该是确切的，并前后一致。但是一本充斥了定义的书是难以阅读的。作为本书脚本的讲稿本来显得有些冗长，加上现在的学生习惯于电视对话语言，不习惯于权威教科书所使用的那种用第三人称的一本正经的说教，所以我们迎合他们使用第一人称的讲话方式，倒也觉得颇为生动。书中所说的“我们”不仅指作者也指作者的同事们，他的老师和学生。他们对他教育甚多，并在一起发展了书中所讨论到的原理和实践。如果允许我们使用“韦氏大学辞典”(Webster's collegiate)名称中“大学”一词的话，本书倒真正是“大学”集体合作的成果。在不影响内容叙述的前提下，本书用了些轻松的俏皮话和双关语。我们不能将它们拒之门外，因为微计算机设计本身就是一种乐趣。我们在谈论它时感到快乐轻松。假如本书行文体现出了这种轻快的话，请读者见谅。

致　谢

作者愿对所有为这书本的出版作出贡献的人表示深切的感谢。除一些学生外，作者的同事，工业界的同行们都作出了很大帮助。特别要感谢查尔士·罗斯(Charles Roth)教授，他采用这本讲义为大学高年级学生开设了"微计算机设计"，并且改正了其中许多错误和为每章结尾提供了一些很好的习题。同时对M6809设计师之一，Motorola公司的特里·里塔(Terry Ritter)深致感谢。他详细地审阅了初稿，对不少错误作了更正，并提出许多建议，得以使这本书比原来好得多。我还要感谢莱恩斯·莱文(Lanece Levnthal)和哈罗德·斯通(Harold Stone)，他们审阅初稿后提出了很宝贵的意见。还要感谢特里·瓦格纳(Terry Wagner教授和学生埃德·厄普丘奇(Ed Upchurch)和基富·奥利弗(Jeff Oliver,)他们校对了本书的部分章节。最后我要感谢我的秘书Janelle Dolby和我的技师Par Horne，他们分别改正了教科书上的错误和制作了插图草稿。如果我还漏掉了什么人的话，那么我要说本书自始至终用的"我们"，它指我，也指所有在形成这本书的思想和出版这本书的过程中提供帮助的人们。

关于作者

作者G Jack Lipovski 1976年以来执教于美国TEXAS大学电子工程和计算机科学系。由于他的具有重大开拓性的数据库计算机CASSM的设计以及重组阵列计算机TRAC的设计，使他成为在计算机系统结构领域中国际上公认的权威。他在微计算机方面的建树，同样亦为国际上所公认。他是欧洲计算机学会ENROMICRO的主席，现任IEEE计算机协会计算机系统结构专业委员会主席和美国计算机协会(ACM)的计算机体系结构专业组(SIGARCH)主席、计算机委员会理事和委员会及会议主持人。

G Jack Lipovski 1969年在伊利诺德大学获博士学位。1969年至1976年在美佛罗里达大学执教，并为美HAVLIS半导体公司设计出一个微计算机。他的兴趣包括阵列处理机、数据库计算机、微计算机、硬件描述语言、光学计算机和计算机结构方面几乎所有的热门课题。

译 者 序

本书作者 G.Jac Liporvski 是美国著名的计算机科学家。由于他领导设计了世界上第一台数据库计算机(CASSM)和第一台重组阵列计算机(TRAC)，使他成为国际上公认的计算机结构方面的权威。他在微型计算机方面的重大贡献，也使他在国际上享有盛誉。他现任欧洲微型计算机学会(EUROMICΓO)主席、IEEE 计算机协会系统结构事业组主席、美国计算机委员会理事和委员会及会议主持人。

本书是以作者在为美国 Texas 大学高年级学生讲授"微型计算机系统设计"时所用的讲稿为基础发展而来。是第一本以软硬结合的设计体系，系统地论述了微型计算机系统的设计原理和方向。接口软件的讨论则以 Motorola 准十六位微处理器 6809 的指令为背景。本书论述的原理和方法同样适用于以 Intel 8088/8086 和 MC68000为主处理器的十六位微型计算机系统的接口设计。本书在叙述上力求避免一般教科书习惯采用的说教方式，而是用作者和读者谈话时的比较活泼和轻松的语言风格。对此风格，译者在初读本书时，确有耳目一新的感觉。

我们在 82 年末开始学习本书，这使我们有机会第一次比较系统地自学了微型计算机及其接口的设计原理和方法。我们也尝试用书中的一些原理和方法，指导并解决了工作中遇到的问题，收益颇大。更重要的是，使我们意识到一个成熟的微型计算机设计人员，应该是"软硬兼施"的。

从 83 年初我们开始本书的翻译，旨在希望由于我们的努力，能把这本书推荐给更多的读者，也使他们从中得到启发和借鉴。

由于翻译工作是在八小时之余进行的，加上我们水平所限，译文中会有许多技术及语言上的错误，这有待同志们的指正。

我们感谢张文国同志参加了本书第七章的初译；更要感谢上海交大白英彩老师给我们推荐了这本书，提供了本书的复印本；并借此机会我们向所有曾帮助本书出版的同志，表示深切的谢意。

译 者

第一章 微 处 理

微计算机，微处理器和微处理对于绝大多数工程师和计算机科学家来说既相当熟悉，又有些模糊不清。当我们教授微计算机接口课时，如果我们问一个简单的问题“微计算机是什么？”我们会得到各种各样的回答。不同的读者，各自的学历和经历各不相同，但是有一点是十分清楚的，这就是他们必须透彻理解微计算机，微处理器和微处理这些概念之间的差别才能讨论和设计接口。本章论述了以后各章讨论接口所需要的微计算机概念和微处理机的程序设计技术。

其他论述接口设计的书都要花半本书的篇幅来讨论本章所论述的内容。这里所讨论的内容是很重要的。接口设计师必须对计算机基本结构，程序和系统组织相当了解。我们的目的是让读者在读完这本书后能为微计算机接口设计出良好的软件和硬件。为了达到这个目的，我们将不得不精简有关计算机系统结构、程序语言和计算机组织等方面的内容。由于需要讲述的内容如此之多，而时间和版面又太少，我们只能给读者讲述一些要点；提提供基本的设计方法。不少读者可以在类似的讲义，或者从工作以及阅读有关微计算机的专业期刊和业余爱好者杂志中学到有关知识。本章就将这些背景知识综合起来。因此有些读者在阅读本章以后就可以进入其余各章的学习。而有一些读者还需进一步学习有关计算机科学方面的基础教材。本章最后部分介绍了有关的参考资料，便于这样的一些读者查阅。

讲述本章内容时，我们假设读者都相当熟悉大型机或小型机的某种汇编语言，或者能很快学会它。在这一章里读者将学会一般微型机的软件要点，特别是应用于 M6809 微处理机的程序设计。读者应能写出大约有一百条指令码的程序并能顺利地调试。读者还应熟悉算术运算，数据结构和子程序处理的基本思想。不仅如此，他们还应充分具备弄懂本书其余部分所讨论的微计算机接口程序的条件。

1-1 引言

微计算机和微处理器到底什么东西？常常与微计算机混为一谈的微程序的涵义又是什么？本节对这些问题和在数字系统设计中常常被误解的一些术语进行必要的阐述，并将描述数字计算机的结构。可惜，计算机研究中充塞各种各样的概念和术语。不过定理和公式并不多。本节将提供大量的概念及其定义，为以后的讨论作些准备。但是在给一些概念下定义的时候，我们面临着一个“先有鸡还是先有蛋”的老问题，因而我们不得不用一些尚待定义的术语。好在本节旨在给读者品尝一些重要概念的“风味”。所以，使用尚待定义的术语不会有什么影响。当以后解释这些术语时读者再可重读本节，以便对微计算机的概念打下更坚实的基础。本书索引可以帮助读者找到关于这些术语的解释。

我们注意到微计算机除了比其它计算机小些，便宜些外，它与其它计算机十分相象。因此本书首先介绍计算机理论，然后细述指令的设计依据。这两者对大型机和微计算机都适用。最后再描述微计算机的特征。

1-1.1 计算机结构

事实上，在计算机结构方面第一篇也是最好的一篇论文是 A.W.Burks, H.H.Goldstein 和 J.von Neumann 三人早在**计算机结构**这术语产生前的十五年所写的论文"电子计算机装置逻辑设计的初步讨论"(Preliminary Discussion of the Logical Design of an Electronic Computing Instrument")中提出。 我们发现把其所述的设计与迄今为止所生产的计算机作一番比较是很引人入胜的。本意用于解决非线性微分方程的这种设计不仅用于解决数值问题，还成功地用于商业数据处理、信息处理和工业控制。这确实是对冯·诺依曼天才的最好推崇。他的设计简明，慎密，以至现今人们所使用的绝大多数计算机从大型机到微计算机都以它为依据，这些计算机被称为**冯·诺依曼计算机**。

结构这个说法是包括 Fred Brooks 在内的 IBM 公司中一些计算机设计师们于六十年代初提出并在 IBM 公司内部使用。他们最初用这个词来描绘 IBM 360 系列机的"兰图"。后来设计出几种不同成本，不同速度的计算机(例如 IBM360/50)都属于 IBM360 系列。一台计算机的结构就是它的指令系统和输入输出通道能力。有相同结构的计算机可以执行相同的程序并能连接相同的输入输出装置。不少计算机制造厂已成功地仿照了这种用相同的"兰图"设计系列计算机的设计思想。本书所讨论的计算机就是 Motorola 公司运用这种设计思想制造出来的。它们都是从 MC6800，MC 6802 和MC6808 引伸出来，并有相同的结构。计算机结构和这种设计方法间的关系已给"计算机结构"下了明确的定义。

"计算机结构"这一术语已经非常流行，它甚至还被引伸出来指整个计算机系统，包括下面所要讨论到的执行技术和组织。事实上，要使两个计算机结构设计师对如何定义计算机结构持相同意见並非易事。我们也对这种含糊不清感到棘手。这也许是因为计算机本身尚处于运动和迅速发展中的结果。

和计算机一样，数字系统"组织"通常是用方块图来表示的。方块中注明了计算机寄存器，总线和数据操作器。以 MC6800 和 MC6802 为例，它们结构相同，因为它们有相同的指令系统并能使用相同的输入输出装置。但是，因为它们内部(结构)略有不同，它们就有不同的组织。一台计算机组织也可以称为计算机的实现(Implementation)。最后要说的是，计算机的"实现"，实际上就是计算机的硬件连接和系统组织。例如，MC6800 和 MC68A00 方块图和指令系统都相同，但是制造 68A00 时可用更快速的半导体器件，以使它的运行速度比 6800 快一倍。(事实上，MC6800，MC68A00 和 MC68B00是按相同的工艺生产的。测试后，那些碰巧速度最快的就作为 68B00 出售。MC68A00 次之，MC6800 最慢。)MC68A00 的实现方法与 MC68B00 不同。一个计算机制造公司用更新型的硬件来替换方块图中某一个方块，以此改变实现方法，或者设计师用更快速的微计算机实现，这都是合情合理的。实现方法不同，但执行过程和组织仍然相同。在本书中，凡在讨论实际实现方法时，我们将用全称来标明系列机中的各机种，如 MC6800 或 MC68A00。但是我们通常只对其结构或组织感兴趣。这样，我们涉及结构时就用 6800 (不加 MC)来代表整个系列，在涉及组织时用 M6800。这样做比较明确，读起来也方便。

随着新技术的不断涌现，在实际应用中又暴露出原有结构的弱点，新的结构就应运而生了。这种新结构总是包括了原有的指令系统，再加上一些新的指令。它要求原计算机编制的程序稍加改变或不加改变后就能在新的计算机上运行。而新编制的效率更高的程序又能发

挥新结构的特长。有这种性质的结构就对原来的计算机有向上的兼容性。

本书将以与6800向上兼容的6809结构为中心进行阐述。为了使本书对只了解6800系统的读者也有用，我们还将对6800作些必要的阐述。这些阐述对运用6800结构设计的诸如M 6802，M6808系统，而现在又在使用6809的读者也是适用的。为适用于6800，本书不少章结尾的部分习题要求读者用M 6800指令系统重新编制书中M6809的程序实例。

冯·诺依曼计算机的结构简单得不致使人对计算机望而生畏，这可以从下列模拟图中看出。　（图 1-1　缺）

假设一个人拿着一个加法器和一扇向外部世界打开的窗户站在邮筒前面。邮筒上的每个邮箱都编上号，我们可以把这个邮筒模拟为主存贮器。加法器模拟为数据操作器（即算术逻辑单元），人便是控制器，窗户为输入输出(I/O)。人的两只手对存贮器进行存取操作，主存贮器称为随机存取存贮器(RAM)。因为人可以根据邮箱号码随意存取。每个邮箱的投邮狭缝中都有一张写有8个1和8个0（位）一串数字的纸条。我们把由八个信息位组成的代码串称是一个字节。在本书中字位计数是从字的右端0位开始数起。

当人用左手从第n个邮箱中取出一个字，把它读成指令，并把它重新放回邮箱。把一个字从邮筒(主存贮器)中送给人(控制器)叫取指令。取字的那只手模拟为程序计数器。当取出第n号邮箱的指令后，手已指向下一个第n+1号邮箱了。

在6809中的指令是象01001100这样的一个二进制码。为了和Motorola所用的标记一样，本书中凡是二进制码都用一个百分符号"%"表示。在"%"后面跟有二进制代码的"0"或者"1"(为了与此相区别，本书中凡是十进制数不用任何指定的符号)。因为所存这些"0"和"1"很难记忆，经常用的是便于记忆的习惯格式。即十六进制数。十六进制的标记是一个美元符号"$"。每四位一组，从0000到1111的二进制码除相应的十进制数中的0到9外，相应十进制的10，11，12，13，14，15则分别用字母A，B，C，D，E，F来表示。二进制码%0100；是十进制数4的二进制码。二进制码%1100是十进制数12的二进制代码，对十六进制记数，则用字母C表示。因此上面两个四位二进制码结合在一起就可以用十六进制数$4C表示。无论是用二进制代码还是用十六进制代码来表示的指令，我们都称为指令的机器码。因为它是从计算机的主存贮器中取得的代码字。因此如果用机器码来记忆指令的话，那实在是太困难了。因此我们用所谓的记忆符来表示机器的指令。如在6809或在6800中，指令$4C它的真正意思是将累加器A的内容加"1"。因此可以把$4C这样的指令写成

INCA

(对M6809的累加器和其他寄存器在第1-2.1节中介绍)。

汇编语言是一种程序，它是将指令的记忆符翻译成相应的机器码。因此程序员可以用规定的指令记忆符书写程序，经汇编语言翻译后输出机器码放到计算机的主存贮器中。因此可以把指令记忆符称为**汇编语言指令**。

许多接口软件都是用汇编语言来写的，而且在本书中讨论的大多数的程序实例用的也是汇编语言。由于与输入/输出设备接口的一部分软件通常很短，因此可以直接用机器码来写。此外，对于一种很固定的程序也可以用机器码来写。因此，在这一节中我们可以看

到某些重要的汇编语言指令相应的机器码。

在6809中的许多指令是用8位为一个字来描述的。然而，6809中也有一些指令需要用16位或24位或更多一些位才能指定。它们是以字的方式存放在连续几个主存单元中，这样，对指令的存取就象对字的存取一样，是一个接着一个地进行。

现在我们已经具备了指令的一些基本概念。我们可以把指令看作要求机器完成某一件事的一个命令。例如，一条指令给控制器发送这样的命令，用它的右手从邮箱的第m个盒子中取出一个字，把它放到加法器中以替代加法器中原来老的一个字，然后把这个字再放回原来的盒子中。这个例子说明了称为装载指令LDA的功能。在6809中，这条指令是将在主存贮器中十进制数第256个单元或者十六进制数$100单元的字装载到累加器A中。这条指令以三个字从主存贮器中取出。即

$136
$01
$00

在这里第二个字是地址的高位字节，第三字是地址的低位字节。用汇编语言中的指令记忆符表示，可以记作

LDA $100

把一个字从邮筒(主存贮器)中取出送到加法器(数据操作器)这种的主操作，称为**数据调用**。右手用来取出一个字，用它比喻为**有效地址**。

和指令一样，汇编语言中用一个速记符号来表示存贮器中的某一单元。**符号地址**是一个名字，它对程序员说来还是有价值的：它实际上是内存中的某一个地址。例如，地址$100，可以把它称作地址ALPHA。这样，上述的汇编语言指令，就可以写成

LDA ALPHA

(在这一章的许多例子中，我们用符号地址ALPHA代表内存中的$100单元，当然不同的地址单元，就要用不同的符号地址来表示。我们在1-3.1节讨论数据结构时，要说明汇编语言可以根据符号地址代真为主存贮器中的实际地址)。要记住符号地址仅仅是主存贮器实际地址的表达式，这个表达式正好表示由符号地址所指向的主存中的实际地址单元。正因为符号地址代表的是一个数，因此可以对符号地址进行诸如加法、乘法等数值运算。如指令

LDA ALPHA+1

表示将地址为$101(ALPHA+1=$100+1)单元的内容加载到累加器A。

通常，在执行这条指令之后，同时用左手(程序计数器)在地址n+1的盒子中取出下一条指令。例如，下一条指令给出一个命令，使加法器中的数复制到邮筒中的一个邮盒中，这样原来在盒子中的数被删去。这是存贮指令的例子。在6800或6809中，存贮指令

是把累加器 A 中的数存到 $100 单元，可写成

```
STA  ALPHA
```

这条存贮指令中的主操作，是把从加法器(数据操作器)中取来的数放到主存贮器中的某一指定存贮单元。这称为**存贮数据**。右手(**有效地址**)用来把数据放到指定的盒子中。

在继续讨论前，我们要指出至今我们还没有论及到冯、诺依曼结构的一个重要特点，而这些特点却是冯·诺依曼对计算机结构的最伟大的贡献，并且也是计算机中一个十分重要的问题。在冯·诺依曼结构中指令和数据是以同样的方式存贮在主存贮器中，而且没有什么办法可以区分出指令和数据，除了用手分别检出指令和数据。如果没有很多指令要存贮的话，可以简单地不把指令存贮在主存贮器中，而主存贮器只用来存贮数据。反过来当然也可以在取指令前，把指令当作数据那样来修改原则上是可能的，但这对于一个计算机科学家来说，一想到这个就会"不寒而栗"。事实上，由于程序中的错误，把指令作为一个数据来取走也是可能的，这样当程序执行时就会出现一种莫名其妙的结果。

程序执行的次序就是连续地从主存取指令的次序。为了在地址 $100 单元的内容加"1"，可以用 LDA 指令将该单元的内容取出放入累加器 A，并用 INCA 指令将累加器 A 的内容加"1"，然后用 STA 指令将累加器 A 的内容放回主存单元 $100。这个程序顺序可以写成

```
LDA  ALPHA
INCA
STA  ALPHA
```

如果不改变左手(程序计算器)的位置，在邻近盒子上的一串数据就会被作为指令取走，并执行之。例如，可以取出并执行一系列的装载和存贮指令，把一些数据从邮筒中的一个地方装载到另一个地方。

当然，在读一条指令后，把他的左手指向到一个新的地址(把一个新的数装载到程序计数器)。这样的指令，称为转移指令JMP。类似的指令是用来执行一个称为子程序的程序段，这个子程序段是放在主存贮器中另一个存贮区。并且在执行完该子程序段的最后一条指令后，返回原来转移指令的下一条程序地址，继续执行原来的程序。这样一条跳转执行子程序的指令不仅象 JMP 指令那样仅仅只改变程序计数器的内容，并且需要把程序计数器中原来的程序地址保存起来，使子程序完成后，恢复原来程序执行。子程序中的最后一条指令，即子程序返回指令，可以恢复程序计数器原来的内容。在小计算机中子程序是特别有用的，它可以使具有共性的操作由子程序完成，而不需要再增加程序段。因为在小计算机中指令是最基本的。跳转指令也可以模仿人的条件判断功能，例如只有在加法器中的数为正数时，程序才跳转执行子程序。如果加法器中的数不是正数时，则程序继续执行下一条指令。这好象人的左手不移到新的一个邮盒的情况那样。这就是所谓的条件转移。转移指令和条件转移指令允许某些程序段可以反复地取出並反复地执行之。用这种方法可以写出一个有上百条指令组成的程序段，这个程序段可以传送或修改成千个数据。

转移指令和条件转移指令本质上是反复执行同一个程序段。计算机的优点就在于可以用存入主存贮器中的控制程序管理大量的数据。

(硬件或输入/输出)中断是一个计算机结构的重要特点，它对于输入/输出接口来说是非常重要的。当输入/输出设备请求服务时，如将数据传送给输入/输出设备或者由设备输出数据或者当设备检测错误时，由中断信号"唤起"计算机处理。中断是打断处理机现行的程序执行，并转入执行另一个程序，以服务于中断请求。在离开中断服务后，可以准确地返回执行主程序。服务于中断的程序称为**中断处理程序**或**设备处理程序**，它非常象一个子程序，而且中断可以认为由输入/输出设备"诱使"计算机转入执行一个子程序。然而这并不意味中断服务程序会以任何方式打扰了正在运行的程序，只要一有中断产生，被中断的程序无论在什么时候都可以保持原有执行结果。满足于这个要求其中的一个问题是，中断服务子程序可以调出正在运行中的程序所调用的子程序。如果子程序现时正被执行，那末正被运行的程序所得到的数据和在中断服务程序所用的数据混淆起来的情况是完全起来可能的。这样就会产生不正确的结果。如果这个情况是可避免的话，那末可以认为子程序是可再入的。因为它可以再一次地进入。即使这样，它已经进入但尚没有完成。这个性质在接口软件的设计中是很重要的。与此相关，我们引入**递归**的概念(recursion)，它是一种处理过程的，在递归过程中某一步要用到它自身上一步(或几步)的结果。因此只要需要，子程序本身想要执行多少次就执行多少次。递归是一种很抽象的概念，递归子程序通常是可再入的。

大多数现代计算机把对中断的控制条件和操作的结果保存在**条件寄存器**(Condition Code register)中。在得到一个结果时，如果结果为零，将标志位 Z 置"1"。当结果为负值时，将 N 位置"1"，如果加法操作产生进位的话，则进位位 C 置"1"。如果加法操作产生一个在二进制补码制被认为无效的结果时，由于溢出，溢出位 V 置"1"。这些位可以用下面所要介绍的条件转移指令来测试。**中断屏蔽位** I 也是保存在条件码寄存器中。当 I = 1 时，中断被屏蔽。条件码也用以产生计算机的条件转移能力和中断控制能力。在控制器和数据操作器的条件码寄存器、累加器，程序计数器和其他的寄存器总的称为**机器状态**(machine state)。并且只要有中断产生，所有这些内容都必须保存起来。中断解除返回主程序前，有这些内容必须被恢复。为了存贮数据或调用数据，可以用不同方法计算有效地址，这称为**寻址方式**(addressing modes)。6809 的寻址方式特别丰富。关于寻址方式，我们在 1-2.1 节中详细介绍。

1-1.2 指令：

在这一节中，我们从另一个角度来描述指令的概念。首先讨论的是取指令，指令的译码和每条指令对应的微指令执行顺序。然后讨论有关软件—硬件的协调。

因为指令可以由单字节、双字节或多字节组成。因此当控制器从主存贮器中取出每一条指令后，向所有的模块发送一个命令以执行这条指令。从本质上来说，一条指令是一个相当复杂的命令，它在从主存贮器中取出的不可分的单字节、双字节或多字节指令控制下，完成给定的操作。

一条指令分成几个"域"，它分别表示指令的操作码，主存中的地址或者是数据操作器

或输入/输出设备中寄存器的地址。例如指令 LDA ALPHA 的二进制代码为 1011011000 0000100000000，当把指令码取到控制器后，从最左的一位开始，从左到右它的第5位到第8位 1，0110 是指令的操作码，告诉计算机执行装载累加器操作。每一条指令必须有不同的指令操作码，这样使控制器准确地知道所执行的指令功能。操作码中从左边数起的第二位"0"表示装载累加器的地址，"0"表示装载累加器 A。从左数起的第3位和第4位"11"表示访存贮器的寻址方式。最后的 16 位二进制数 0000000100000000 表示主存地址 $100(ALPHA)。一般说来，指令操作码、寄存器地址，寻址方式和主存贮器地址对不同的指令来说是不同的，因此必须对指令码进行译码。例如本例中 1，0110，从左数起的第二位"0"表示寄存器地址，第三和第四位则是访问主存的寻址方式。

控制器对每条指令的执行是以执行一定顺序的**微指令**(microinstruction)完成的，与指令不同，指令是存贮在主存贮中，而微指令是存贮在一个被称为**控制存贮器**(control men—ry)内的小的快速存贮器中。微指令是一个转换为**微命令**(micro-order)的数据集合。这些微命令是同时执行的。通常，这些微指令是由计算机制造厂把它写在一个用户不可以修改的只读存贮器的快速控制存贮器中。但是在有些计算机中，控制存贮器也可以由用户重新写入。写在控制存贮器中的程序称为**微程序**。微程序是把一条指令的具体动作转化为数据传送的控制。

一条完整指令的执行称为**取指—执行周期**(fetch-execut cycle)，它是由微指令系列所组成。由于对主存贮器的存取速度相当慢，一条指令所对应的全部微指令的执行是聚集在存贮器周期内完成的。存贮器周期是从主存贮器中取一条单字节指令，存贮或调用一个数据字或者是一次空操作所占用的时间。存贮器时钟是基本的时钟信号，每一个存贮器周期有一个时钟脉冲。取指—执行周期是由几个存贮器周期组成。如果一条多字节的指令，一般说来需要有几个存贮器周期。在有些计算机中，存贮器周期后的下一个周期是用来对指令进行译码，分析指令码后决定下一个周期做什么。6800 和 6809 没有把取指周期和译码周期分开，下一个周期可以用来作**地址计算**。数据可以在一个或几个**调用周期**周期(recall cycles)中从存贮器中读出。而指令的主要功能是在**执行周期** (execute eycle)中完成。在最后一个周期—**存贮周期**(memorize eycle)将数据存入。

输入/输出设备可以在任何一个存贮器周期中向处理机申请中断。然而，总是有某些信息位或信息片散布在数据操作器中，因此不能停止在现行指令的执行，只有在现行指令执行结束后识别中断源。在累加器和其他寄存器中的数据，即机器状态被安全地保存起来或恢复后，从输入/输出设备请求中断开始，直到所要传送的数据传结束或者报告出错的这段时间称为**中断等待**时间(latency)。快速的输入/输出设备所需要的中断服务时间很短。最短的中断等待时间受一条最长指令执行时间和保存机器状态所需要的时间限制。这是因为输入/输出设备可能在这样一条指令的开始向处理机请求中断。

设计一条指令，使它在一条指令中可以完成一个很复杂的操作，可以想象这完全是可能的。也就是说，有这样一条指令，它可以用一定的寻址方式完成需要用许多指令才能完成的操作。另一方面，取出并执行一系列很简单的指令用以完成由一条指令所完成的复杂操作，通常来说也是完全可能的。实际上，我们早已指出可以用一条 6809 中的指令 INC ALPHA 来完成上面介绍的一个程序所要完成的工作。它从 $100 单元取一个数，加"1"后，把结果重新存贮到 $100 单元，而不需要改变累加器的内容。如果一个有用的操作不

是由一条像INC ALPHA指令来完成，而宁可用一系列最简单的指令组成的程序来完成，那么这样一系列的指令组称为**宏指令**(macroinstruction)，即**宏命令**(macro)或**子程序**。如果每次需要完成以前需要由一个程序段所完成的工作时，可以直接调用它的宏命令完成。如果这样的程序段只写一次的话，它就是一个子程序。每次在需要完成由这子程序段指定的操作时，程序就可跳到该子程序的入口执行该子程序。宏指令和子程序这两个概念非常相似，它们都是顺序执行一组指令以完成给定的操作。

对于一台计算机结构设计来说，一个中心问题是确定什么工作由指令(包括作为寻址方式)所完成，什么工作可以用宏命令或子程序所替代？以前已经证明，一台只有一条指令的计算机同样可以完成现有的所有计算机所能完成的工作。这种只有一条指令的计算机执行某种操作需要很长的时间，而且它的程序太长太复杂了。尽管如此，现今计算机所能做的任何工作，都可以由这台只有一条指令的计算机所完成。与此相反的另一个极端是，从大多数程序员的角度来看，他们或许希望他们所用的计算机只要用一条指令就可以完成象FORTRAN这样的高级语言的一条语句。这样复杂的指令会带来许多问题。如发现中断时，需要有非常长的等待时间，这仍然是一个效率问题。指令选择的依据，是使由计算机所完成的操作可以转换为用这些指令所写的程序来完成。并且要求指令执行速度尽可能快，程序所占的存贮空间尽可能小(换句话说即存贮密度问题)，而且要求程序等待时间也要尽可能地短。

指令的选择由于两方面的要求而使指令设计复杂化。有些使用，需要计算机有最好速度；而另有一些使用却要求优化程序密度。例如，用象台式计算机这类计算机，完成一次操作的速度为0.1秒，这种使用对于提高机器的运算速度倒是无关紧要的，因为使用者并不会从提高速度中得到好处，而关心的却是提高程序密度。因为对这样的台式计算机来说，存贮器占有机器成本中的主要部分，因此由于程序密度的增加，可以大幅度地降低机器的成本。另一方面，对用在计算中心的具有大存贮容量的计算机来说，要求它在运算速度上有大幅度提高，这样就可以在相同的时间内完成更多的工作，然而对程序密度的提高却并不是很在乎的。因为它本身已经有大容量后援存贮器的支持。此外，不同使用场合的计算机对速度和程序密度的要求也各有不同。

直至今日，我们还不能说有某一种计算机可以适应各种不同的使用要求。因此针对各种不同的要求产生了许多具有不同特点的计算机。由此就存在着如何根据实际的要求选用最适合的计算机的问题。为了从许多计算机中选择一个能很好满足使用要求的计算机，一般说来，可以用一个称为**基准测试程序**(bench maks)对计算机进行测试。这些基准测试程序是，用户根据预期的使用情况而编制的简单易学，目的明确的各种程序的集合。作为基准测试程序的一些实例可以有二个无符号16位数的乘法，把内存某一单元的数搬到内存的另一个单元，从序列数据字中搜索出其中的一个等等。为了达到基准要求，对每个计算机都要写这些程序，并且记下每一计算机的执行速度和程序密度。用这一些数据的加权和，得到每一个机器的**性能指数**(figure of merit)。如果要研究的是存贮密度，那么所用加权数值是与预期要存在主存中的基准试验程序(或类似于基准试验程序的测试程序)的次数成比例，并把它们的性能因素称为**静态效率**。如果所研究的是速度，那么加权值是与期望执行的基准试验程序(或与此类似的子程序)的次数成比例，并把它们的性能因素称为**动态效率**。这些性能因素与计算机的租用费或购买费用，可用的软件，厂家的服务信誉

和其他的一些因素通盘考虑，用来作为选择机型的参考。

在这一章和全书中，对接口软件的设计，我们自始至终强调了这样的观点：对于一个好的程序，在指令的选择上必须强调指令的效率。

6809 具有特别丰富的指令系统，它对许多重要的操作都有几种方式可选择。读者要十分重视程序设计的技巧，应该在你的程序中尽可能使用效率最高的指令。如果你珍视的是动态效率的话，那么程序中要选择可以以最短的时间完成的指令，如果希望提高程序的静态效率，那么要选择占用存贮空间最少的指令。

1-1.3 微计算机

微计算机是上面讨论的计算机中一种价格最便宜的计算机。小计算机可以按字长，内存容量以及数据操作器、控制器和存贮器的价格进行分类。1970 年贝尔(Bell) 将小计算机(Small Comput)分为微型机(Microcompute)、小型机(Minicomputer)和中型计算机(Midicomputer)三类。其分类表见表 1-1。

表 1-1 计算机分类

类型	内存容量	价格($1)(1970)	位/字
微型机	8,000(8K)	5,000	8至 12 位
小型机	32,000(32K)	10,000	12至 16 位
中型机	65,000(64K)	20,000	16至 24 位

尽管技术的迅速发展使微型机、小型机以及中型机之间的差别已不象过去那样明显，而且机器的价格已经大幅度地下降，但上述分类目前还是流行的。半导体集成技术已经可以把成千上万的三极管、寄存器或电容等集成在一个 0.04×0.04 英吋的薄薄硅片上，这就是目前所称的大规模集成电路 LSI。因此就有可能把计算机中的数据操作/控制器集成在一块或几块大规模集成电路上。这样的数据操作/控制器称为**微处理器**。微处理器与输入/输出模块和存贮器模块相连就组成**微计算机**。模块间用印制电路板连接，各个印制板通过公用母板连接。目前微计算机的硬件系统已经非常便宜，然而所配置的系统软件还比较缺乏。例如，单板机在 1978 年大约 100 美元，一个功能更强的微处理机的价格也只有 300 美元。作为微计算机的核心微处理器的价格就更加便宜，一般只有 10 美元左右 。而小型机的特点在于它的系统软件相当丰富，一般都配有装入程序、查错程序汇编语言、高级语言和操作系统、数据库等功能相当强的系统软件。现在微计算机与小型计算机之间的差别变得愈来愈模糊，这是因为某些系统结构既用在小型机上，也用在微型计算机上。如小型机 PDP11-45 和微型机 PDP-11/LSI就是这样。由于 16 位微型计算机(如 68000)的出现，使小型机和微型机间的差别变得更小，它的内存容量已经达到 64K 字，这样有可能将小型机的软件直接移植到微型机上。因此本书中将微型机和小型机统称为小计算机，主要是用来说明计算机之小。

有讽刺意义的是，微型机这个七十年的超级明星却是一个“破裂婚姻”的产儿。七十年代初期，我们已经可以把一个相当复杂的计算器集成在一块 LSI 片上，人们就想能不能把一个计算机集成在一块大规模集成电路片上呢？美国仙童半导体公司 (Fairchild) 和