普通高等教育“十五”国家级规划教材

微型计算机原理

（第二版）

宋汉珍　主编

高等教育出版社

内容提要

本书是普通高等教育“十五”国家级规划教材。根据高职高专教育特点，本书将计算机组成原理的主干内容和微型计算机原理的内容有机结合，统筹安排，形成独具特色的一本教材。本书具有内容充实、结构严谨、深入浅出、通俗易懂的特点。

本书的内容包括：计算机系统概述、计算机中数据的表示法、运算器与控制器、Intel 80x86微处理器、存储系统、8086指令系统与汇编基础、输入输出系统及接口、中断系统串行和并行通信及常用接口电路、总线。

本书各章后面均附有习题。

本书适合于高等职业学校、高等专科学校、成人高校、本科院校举办的二级职业技术学院，也可供示范性软件职业技术学院、继续教育学院、民办高校、技能型紧缺人才培养使用，还可供本科院校、计算机专业人员和爱好者参考使用。

图书在版编目（CIP）数据

微型计算机原理/宋汉珍主编. —2版. —北京：高等教育出版社，2004.11

ISBN 7-04-015740-3

Ⅰ.微… Ⅱ.宋… Ⅲ.微型计算机—高等学校—教材 Ⅳ.TP36

中国版本图书馆CIP数据核字（2004）第101873号

策划编辑 冯 英 责任编辑 关 旭 封面设计 王凌波 责任绘图 朱 静
版式设计 胡志萍 责任校对 康晓燕 责任印制

出版发行 高等教育出版社 购书热线 010-64054588
社 址 北京市西城区德外大街4号 免费咨询 800-810-0598
邮政编码 100011 网 址 http://www.hep.edu.cn
总 机 010-58581000 http://www.hep.com.cn

经 销 新华书店北京发行所
印 刷
开 本 787×1092 1/16 版 次 2001年9月第1版
印 张 20 年 月第2版
字 数 480 000 印 次 年 月第 次印刷
定 价 25.10元

物料号：15740-00

出 版 说 明

为加强高职高专教育的教材建设工作，2000年教育部高等教育司颁发了《关于加强高职高专教育教材建设的若干意见》（教高司〔2000〕19号），提出了“力争经过5年的努力，编写、出版500本左右高职高专教育规划教材”的目标，并将高职高专教育规划教材的建设工作分为两步实施：先用2至3年时间，在继承原有教材建设成果的基础上，充分汲取近年来高职高专院校在探索培养高等技术应用性专门人才和教材建设方面取得的成功经验，解决好高职高专教育教材的有无问题；然后，再用2至3年的时间，在实施《新世纪高职高专教育人才培养模式和教学内容体系改革与建设项目计划》立项研究的基础上，推出一批特色鲜明的高质量的高职高专教育教材。根据这一精神，有关院校和出版社从2000年秋季开始，积极组织编写和出版了一批“教育部高职高专规划教材”。这些高职高专规划教材是依据1999年教育部组织制定的《高职高专教育基础课程教学基本要求》（草案）和《高职高专教育专业人才培养目标及规格》（草案）编写的，随着这些教材的陆续出版，基本上解决了高职高专教材的有无问题，完成了教育部高职高专规划教材建设工作的第一步。

2002年教育部确定了普通高等教育“十五”国家级教材规划选题，将高职高专教育规划教材纳入其中。“十五”国家级规划教材的建设将以“实施精品战略，抓好重点规划”为指导方针，重点抓好公共基础课、专业基础课和专业主干课教材的建设，特别要注意选择一部分原来基础较好的优秀教材进行修订使其逐步形成精品教材；同时还要扩大教材品种，实现教材系列配套，并处理好教材的统一性与多样化、基本教材与辅助教材、文字教材与软件教材的关系，在此基础上形成特色鲜明、一纲多本、优化配套的高职高专教育教材体系。

普通高等教育“十五”国家级规划教材（高职高专教育）适用于高等职业学校、高等专科学校、成人高校及本科院校举办的二级职业技术学院、继续教育学院和民办高校使用。

教育部高等教育司

2002年11月30日

第二版前言

本书是普通高等教育“十五”国家级规划教材。

“微型计算机原理”一书自2001年出版以来，已连续4次印刷，被几十所院校选用，受到广大读者的好评。随着高等职业教育的发展和教育改革的不断深入，对原理性课程的教学提出了新的要求，教学内容更加注重应用性、实用性，更进一步突出实践能力的培养。为更好地适应我国高等教育大众化发展趋势和高职高专教育培养应用性人才的需要，也考虑到很多读者的意见和建议，我们按教育部对“十五”国家级规划教材的要求，对本书进行了修订。与第一版相比，本书主要进行了以下几方面的修订：

1. 丰富计算机及微型机发展的新技术、新知识，扩充第1章计算机系统概论的内容。

2. 微处理器一章中高级处理器的内容只介绍典型高级处理器，不面面俱到。

3. 对存储系统一章，调整内容编排，使结构更合理。增加存储器扩展方面的应用实例，并增加闪速存储器的介绍。

4. 原书考虑到在高职高专计算机类专业中，一般不单独开设“计算机组成原理”课，将计算机组成原理的基本核心内容融于书中，与微型机结构原理有机地结合，旨在使学生在较全面地了解计算机基本原理的基础上，掌握微型机的应用技术，也使计算机类专业的学生对计算机的组成原理有一个系统的概念。现考虑高职高专基本不再开设组成原理课，所以在第二版中，将运算器和控制器两章的内容简化、压缩为一章。

5. 原书考虑到计算机相关专业单独开设“汇编语言程序设计”课程，所以书中从指令格式，指令构成和指令分类方面介绍了8086指令系统，不详细介绍指令系统和汇编语言的内容。现考虑有些院校不单独开设汇编语言课程，所以在第二版中较大幅度增加指令系统和汇编语言的内容。

6. 对输入/输出系统及中断系统部分，增加串行接口芯片的介绍和实例以及8259的介绍和实例。

7. 增加一章有关总线的内容，介绍有关总线的基本知识，包括总线概念、总线分类、总线标准、总线仲裁、总线通信协议等，并分类介绍几种常用的系统总线和外部总线。

8. 在附录中增加常用集成芯片的引脚号和功能表，便于实验中查询。

9. 因高职、高专学生在理论基础的学习上，不过高追求系统性和理论论证，所以本书在阐述上注重深入浅出，问题的叙述尽可能详尽、通俗，便于自学。

本书由宋汉珍担任主编，马秋菊、董国增、王立萍担任副主编。第1、2、3、4、5章由宋汉珍编写，第6章由王立萍编写，第7、10章及附录C由董国增编写，第8、9章及附录A、B由马秋菊编写，全书由宋汉珍统稿，孔小利教授担任主审。由于编者水平有限，书中难免存在错误和不妥之处，敬请广大读者多提宝贵意见。

编　者

2004年7月

第一版前言

本书是教育部高职高专规划教材。

考虑到高职高专教育以技术应用为目的，注重教学内容的应用性、针对性和实用性，并考虑到高职高专学制短、内容多的特点，本教材将“计算机组成原理”的主要内容和“微型计算机原理”的整体内容融于一体，有机结合，使得学生不必单独学习“计算机组成原理”课程，却能较系统地掌握其主要内容，既能节省学时，提高教学效率，又能减轻学生负担，提高教学效果。

本书结合 Intel80x86系列微型计算机，有针对性地介绍微型计算机的基本原理和应用技术。该系列微型机是世界上处于主流地位的机型，以其为例机具有普遍的应用意义。

全书共分为 10章，主要内容为：第 1章计算机系统概述，从计算机的发展、应用、基本构成和工作过程等方面介绍计算机的总体概念；第 2章计算机中数据的表示，介绍计算机中数值数据和非数值数据的表示方法；第 3章运算方法与运算器，介绍计算机中各种运算的基本实现方法和运算器的基本结构；第 4章控制器，主要以微程序控制器为例介绍计算机控制器的基本原理和基本构成；第 5章 Intel80x86微处理器，介绍 80x86微处理器的基本结构、引脚功能和时序；第 6章存储系统，主要介绍微型机主存储器的工作原理、结构及扩展方法；第 7章 80x86的寻址方式与指令系统，从计算机指令格式、寻址方式和指令系统的构成角度概括介绍微型机指令系统的总体内容，其详细内容在“汇编语言程序设计”课程中介绍；第 8章输入输出系统及接口，从计算机端口、接口和数据传输角度介绍输入/输出基本内容；第 9章中断系统及 DMA系统，先介绍计算机中断的普通概念，然后具体介绍 80x86系列微型机的基本中断系统；第 10章串、并行通信及接口电路，在计算机基本串、并行通信概念的基础上，介绍典型的可编程并行接口芯片 Intel8255和可编程计数/定时器 Intel8253。

本教材第 1、2、3、4、5、6章由宋汉珍编写，第 7、8章由董国增编写，第 9、10章由马秋菊编写，全书由宋汉珍统稿。本书由刘永华担任主审。由于编者水平有限，书中难免存在错误和不妥之处，敬请广大读者多提宝贵意见。

编　者

2000年 11月

目　录

第 1章　计算机系统概论

电子计算机是现代社会最有价值的工具之一，它的出现极大地推动了人类社会的发展。计算机的发展水平，已经成为衡量一个国家现代文明的重要标志。在现代社会中，计算机已深入到人类工作、学习与生活的各个方面，计算机的使用，已成为各行各业的技术人员、管理人员必备的基本技能和基本素质。

由于计算机具有计算、模拟、分析问题、操纵机器、处理问题等能力，被看作是人类大脑的延伸，是一种有“思维”能力的机器，尤其是微型机，由于具有体积小重量轻的特点，可作为各种系统、设备的控制中枢，所以常被人们俗称为“电脑”。

1.1　计算机的发展

在公元 10世纪，中国人就发明了早期的计算工具——算盘，它采用十进制运算，是纯数字计算工具，至今仍流传于世界各地。17世纪出现了计算尺，随后，各种机械式、机电式、电动式计算仪器不断出现。1642年，法国科学家巴斯卡（B. Pascal）发明了能实现十进制加减运算的机械式计算机；20世纪 40年代初，德国工程师楚译（Konrad Zaye）采用继电器制造了机电式程控计算机。这些计算机的出现，为电子数字计算机的发展奠定了基础。

1.1.1　电子数字计算机的发展

电子数字计算机简称电子计算机或计算机。世界上第一台电子数字计算机于 1946年问世，它是美国宾夕法尼亚大学研制的，被命名为 ENIAC（Electronic Numerical Integrator And Computer，电子数字积分计算机）。当时正处在第二次世界大战期间，为了解决许多复杂的弹道计算问题，在美国陆军部的资助下开始了这项研究工作，领导研制的是埃克特（J. P. Eckert）和莫克利（J. W. Mauchly）。ENIAC于 1945年底完成，1946年 2月正式交付使用，因为它是最早问世的一台电子数字计算机，所以一般认为它是现代计算机的鼻祖。

ENIAC共用 18 000多个电子管，1 500个继电器，重达 30 000 kg，占地 170 m^2，功率 140 kW，每分钟能计算 5 000次加法。ENIAC存在两个主要缺点，一是存储容量太小，只能存 20个字长为 10位的十进制数；二是用线路连接的方法来编排程序，因此，每次解题都要依靠人工改接线路，使用不方便。

与 ENIAC计算机研制的同时，冯·诺依曼（Von Neumanu）与莫尔小组合作研制 EDVAC计算机，在这台机中，确定了计算机的 5个基本部件，采用了存储程序方案，这种结构的计算机称为冯·诺依曼结构。

1. 冯·诺依曼计算机的基本特点

① 由运算器、控制器、存储器、输入设备、输出设备五大部件构成。

② 采用存储程序的方式，将程序和数据放在同一存储器中。

③ 采用二进制码表示数据和指令。

④ 指令由操作码和地址码组成。

⑤ 以运算器为中心,输入输出设备与存储器间的数据传送都通过运算器。

冯·诺依曼思想被看作是计算机发展史上的里程碑。随着技术的发展,计算机系统结构有了很大发展,对冯·诺依曼机作了很多改革,但原则变化不大,基本组成仍属冯·诺依曼结构。

2. 电子数字计算机发展的 4个阶段

近几十年来,根据电子计算机所采用的物理器件的发展,一般把电子计算机的发展分成四代。

第一代——电子管计算机时代 (约 1946年—1958年),其主要特点是采用电子管作为基本器件,用磁鼓、延时线存储信息,编制程序主要使用机器语言,符号语言开始使用。这一代计算机主要用于科学计算,如 1954年由美国 IBM公司推出的 IBM 650小型机是第一代计算机中行销最广的计算机,销售量超过 1 000台。1958年 11月问世的 IBM 709大型机,是 IBM公司性能最高的最后一个电子管计算机产品。

第二代——晶体管计算机时代 (约 1958年—1964年),其主要特点是采用晶体管作为基本器件,所以缩小了体积,降低了功耗,提高了速度和可靠性。用磁心存储信息。软件方面出现了高级语言,如 ALGOL、FORTRAN。这一代计算机除进行科学计算外,在数据处理方面得到了广泛应用,而且开始用于过程控制,如 1960年控制数据公司 (CDC)研制的高速大型计算机系统 CDC 6600,于 1964年完成。该公司当时在生产用于科学计算的高速大型机方面处于领先地位。1969年 1月大型机 CDC 7600研制成功,平均速度达到每秒千万次浮点运算,成为 20世纪 60年代末性能最高的计算机。

第三代——集成电路计算机时代 (约 1964年—1971年),这一时代的计算机采用中小规模集成电路作为基本器件,因此功耗、体积、价格进一步下降,速度和可靠性相应提高。仍采用磁心存储器。软件方面,操作系统得到进一步发展与普及,使计算机的使用更方便了。IBM360系统是最早采用集成电路的通用计算机,也是影响最大的第三代计算机,其平均运算速度达每秒钟百万次,且走向通用化、系列化、标准化。

第四代——大规模集成电路计算机时代 (约 1972年至今),这一时代的计算机采用大规模集成电路和超大规模集成电路作为基本器件。20世纪 70年代初,半导体存储器问世,迅速取代了磁心存储器,并不断向大容量、高速度方向发展。此后存储芯片集成度大体上每三年翻两番,价格平均每年下降 30%。软件方面出现了与硬件相结合的趋势。

3. 第五代计算机的构想

目前用器件划分计算机时代的方法已遇到问题,新一代计算机涉及系统结构、材料、人工智能、神经网络众多领域,很难再以器件作为划分时代的标准了。

1981年日本政府提出了发展第五代计算机的十年计划,突破了冯·诺依曼结构原理,其目标是实现智能计算机,但没有取得预期的结果。美国也有多家公司推出智能计算机。一般要求智能计算机具有下列功能:

① 智能接口功能。能自动识别自然语言、图形、图像,即有语音识别、视觉、感知、理解功能。

② 解题推理功能。根据自身存储的知识进行推理,具有问题求解和学习的功能。

③ 知识库管理功能。要求能完成知识获取、检索和更新等功能。

随着大规模集成电路的迅速发展,计算机进入了大发展时期,通用机、巨型机、小型机、微型机、工作站都得到了不同程度的发展。

1.1.2 微型计算机的发展

微型计算机(简称微型机)是电子计算机技术和大规模集成电路技术的结晶,它的出现和发展是和大规模集成电路技术的迅速发展分不开的。微型计算机指采用超大规模集成电路,体积小、重量轻、功能强、耗电少的计算机系统。

微型机的发展是以微处理器的发展为表征的,以微处理器为中心的微型机是电子计算机的第四代产品。微处理器自1971年诞生以来,发展迅猛,每2~3年就更换一代,根据微处理器的发展可把微型机的发展分为五代。

1971年,Intel公司研制成功世界上第一片微处理器芯片4004,并推出由它组成的MCS-4微型计算机。4004是4位微处理器芯片,采用PMOS工艺,在一块0.297×0.404 cm^2硅片上集成了2 250个晶体管,指令执行速度为0.06 MIPS(Million's Instruction Per Second,每秒执行百万条指令),工作时钟不到1 MHz。1972年Intel公司推出了8位微处理器8008及MCS-8微型机,8008是第一只通用的8位微处理器。4004和8008是这个时期的代表产品,称为第一代微处理器。第一代微处理器的特点是采用PMOS工艺,速度较低,指令系统简单,运算功能差。

1973年,Intel公司研制成功了性能更好的8位微处理器8080。加速了微处理器和微型机的发展。这一时期,具有代表性的8位微处理器还有Zilog公司生产的Z80,Motorola公司生产的M6800,MCS公司生产的6501和6502。这些高性能的8位微处理器是第二代微处理器的代表产品。第二代微处理器采用NMOS工艺,除了集成度有了提高外,性能也有明显改进,运算速度约提高了一个数量级,指令寻址方式增至10种以上,基本指令可达一百多条。1976年,Intel公司又推出了与8080兼容的8085微处理器,在当时的世界微处理器市场上,由Intel 8080和8085、Zilog的Z80以及Motorola的M6800形成了三足鼎立的局面。

1978年,Intel公司推出了新一代16位微处理器Intel 8086,成为80x86的第一个成员,这标志着微处理器和微型机的发展进入了第三代。该微处理器集成了29 000多个晶体管,指令执行速度达0.75 MIPS,工作时钟频率为4 MHz~8 MHz。随后,Zilog公司生产了16位微处理器Z8000,Motorola公司生产了M68000。16位微处理器比8位微处理器有更大的寻址空间、更强的运算能力、更快的处理速度。1982年,增强型16位微处理器Intel 80286出现,该芯片集成13.4万个晶体管,工作时钟为8 MHz~10 MHz,指令平均执行速度为1.5 MIPS。同年Motorola公司推出了M68010。第三代微处理器采用HMOS高密度集成半导体工艺技术。这类微处理器具有丰富的指令系统,采用多级中断系统,具有多种寻址方式。

1985年,Intel公司推出了第四代微处理器80386,它是80x86系列的第一个32位微处理器,该芯片集成27.5万个晶体管,工作时钟频率达16 MHz~40 MHz,指令平均执行速度5 MIPS。同期的32位微处理器还有Motorola的M68020和NEC的V70等。1989年高档的32位微处理器Intel 80486推出,该芯片集成120万个晶体管,工作时钟频率达50 MHz~100 MHz,指令平均执行速度40 MIPS,同期Motorola推出M68030、M68040,NEC推出V80等。第四代微处理器采用流水线控制,具有面向高级语言的系统结构,有支持高级调度和调试以及开发系统用的专用指令。

1993年,Intel公司推出了第五代64位微处理器Pentium(奔腾),即80586,简称P5。该芯片

集成了 315万个晶体管,工作时钟频率达 66 MHz~200 MHz,指令平均执行速度 112 MIPS。1995年,Intel公司推出性能更高的新产品 Pentium Pro,同期还有 IBM、Apple和 Motorola三家联合推出的 PowerPC、AMD公司的 K5和 Cyrix公司的 6X86 (M1)等。1997年,Intel公司推出了 Pentium II (奔腾二代),AMD公司推出 K6,Cyrix公司推出 6X86MX (M2)微处理器。这一阶段微处理器市场上形成了 Intel、AMD、Cyrix三足鼎立的状态,微处理器工作时钟频率达 266 MHz~450 MHz。1999年,Intel公司又推出性能更高的微处理器 Pentium III (奔腾三代),工作时钟频率达 500 MHz~750 MHz。第五代微处理器采用一些最新设计技术,如双执行部件、超标量体系结构、集成的浮点部件、高速缓存、多媒体增强指令集 (MMX)等,采用先进的 0.25 μm~0.13 μm生产工艺。2001年,Intel公司推出 Pentium 4处理器主频达 1.2 GHz以上。2002年,Intel公司 3.06 GHz的 Pentium 4处理器在全球发布,它是新式 0.09 μm制造工艺的 Prescott核心 Pentium 4处理器。

微型计算机的发展历程,实际上是微处理器从低级到高级、从简单到复杂的发展过程。通过体系结构和制造工艺的改进,微处理器的集成度不断提高,运算速度迅速提高,功能也越来越复杂,成本越来越低。计算机技术的迅速发展,极大地推动了计算机的普及应用。

1.1.3 我国计算机的发展概况

1956年国家制定 12年科学规划时,把发展计算机、半导体等技术学科作为重点,相继筹建了中国科学院计算机研究所、中国科学院半导体研究所等机构。1958年我国第一台电子计算机 (103机)在北京诞生,1959年研制成大型通用电子管计算机 (104机),1960年研制成我国第一台自己设计的通用电子管计算机 (107机)。其中 104机运算速度为 10 000次/秒,主存为 2 048 B (2 KB)。

1964年我国开始推出第一批晶体管计算机,其运算速度为 10万次/秒 ~20万次/秒。

1971年研制成第三代集成电路计算机。1974年后 DJS-130晶体管计算机开始小批量生产。1982年采用大、中规模集成电路研制成 16位的 DJS-150机。

1983年国防科技大学推出向量运算速度达一亿次/秒的银河 I 巨型计算机。1992年向量运算速度达 10亿次/秒的银河 II 投入运行。1997年银河III 投入运行,速度为 130亿次/秒,内存容量为 9.15 GB。目前只有少数国家能生产巨型机。

1999年具有世界水平的大规模并行计算机系统神威 I 号研制成功,其最高运算速度达每秒 3 840亿次浮点运算。

2002年,国内第一台万亿次超级计算机——联想深腾诞生。

20世纪 90年代以来,我国微型机形成大批量、高性能的生产局面并且发展迅速,而且还生产了许多我国自己的知名微型机品牌,如联想、方正、浪潮、海信等,这些微型机厂家无论在生产规模,还是在质量水平上已与国际 PC厂商 IBM、Compaq等相当。

1.2 计算机的分类及应用

1.2.1 计算机的分类

计算机的种类繁多,根据不同的分类方式,有多种分类方法。下面从几个方面进行分类。

1. 按信息的表示形式和处理方式分类

按信息的表示形式和处理方式分为数字计算机、模拟计算机、数字模拟混合计算机。

数字计算机所处理的信息是离散的、数字化的,其特点是解题精度高、信息便于存储、通用性强、具有逻辑功能。通常所说的计算机都指的是数字计算机。模拟计算机所处理的信息是连续变化的模拟量,如电压、电流,其运算部件是一些电子线路。模拟计算机运算速度快,但精度不高,难于存储信息,使用也不方便,主要用于实时控制等专用场合。混合计算机处理的信息既有数字量又有模拟量,取数字计算机和模拟计算机之长,既运算速度快又便于存储,但这种计算机结构复杂,设计难度大,造价高,一般只用在专用场合。

2. 按用途分类

按计算机的用途可分为通用计算机和专用计算机。

通用计算机按一定标准配置存储容量、外围设备、系统软件、通用接口等,并形成系列。这种计算机功能齐全、通用性强。一般所讲的计算机都是通用计算机。专用计算机是为某些特定的问题专门设计的计算机,其功能单一、可靠性高,一般用于军事、工业控制等方面。

3. 按计算机的规模分类

按计算机的规模可分为巨型机、大型机、中型机、小型机和微型机。

这种划分综合了计算机的运算速度、字长、存储容量、输入输出能力、价格等多方面指标。一般巨型机和大型机结构复杂、运算速度快、系统功能强、有丰富的外部设备和通信接口、价格昂贵。但随着计算机性能的发展变化,这种划分标准也逐渐失去了意义,例如现在的高档微型机,在性能、速度等多方面已远远超出了过去的大、中型机。

1.2.2 微型机的分类

在微型机中,又可根据其他指标进行分类。

1. 按微处理器的位数分类

按微处理器的位数分为 8位机、16位机、32位机、64位机,即分别以 8位、16位、32位、64位处理器为核心的微型计算机。

2. 按组装形式和系统规模分类

按组装形式和系统规模可分为单片机、单板机、个人计算机 (PC机)。

单片机是将微型机的主要部件集成在一片大规模集成电路芯片上形成的计算机。如将微处理器、存储器、输入输出接口等集成在一个芯片上,它具有完整的微型机功能。单片机具有体积小、可靠性高、成本低等特点,广泛应用于智能仪器、仪表、家用电器、工业控制等领域。

单板机是将微处理器、存储器、输入输出接口、简单外设等部件安装在一块印刷电路板上构成的计算机。单板机具有结构紧凑、使用简单、成本低等特点,常应用于工业控制和实验教学等领域。

PC机是将一块主机板 (包括微处理器、内存储器、输入输出接口等芯片) 和若干接口卡、外部存储器、电源等部件组装在一个机箱内,并配置显示器、键盘等外部设备和系统软件构成的计算机系统。PC机具有功能强、配置灵活、软件丰富、使用方便等特点,是最普及的微型计算机。

1.2.3 计算机的应用

计算机的应用非常广泛,已经深入到生产、科研、生活、管理等各个领域。下面从几个方面进行概括介绍。

1. 科学计算

科学计算一直是电子计算机的重要应用领域之一,人们在天文学、空气动力学、核物理学等领域中,都需要依靠计算机进行复杂的运算。在军事方面,导弹的发射及飞行轨道的计算、先进防空系统等现代化军事设施通常都是由计算机控制的大系统,如雷达、地面设施、海上装备等。现代航空、航天技术的发展,如超音速飞行器的设计、人造卫星与运载火箭轨道计算更是离不开计算机。过去人工需要几个月、几年,甚至根本无法计算的问题,使用计算机只需几天、几小时甚至几分钟便可完成。

除了国防及尖端科技以外,计算机在其他学科和工程设计方面,诸如数学、力学、晶体结构分析、石油勘探、桥梁设计、建筑、土木工程设计等领域内也得到广泛的应用,促进了这些学科的发展。

2. 数据处理

利用计算机对数据进行分析加工的过程就是数据处理的过程。当前大部分计算机都用于数据处理。在银行系统、财会系统、档案管理系统、经营管理系统等管理系统及文字处理、办公自动化等方面都大量使用计算机进行数据处理。如现代企业的生产计划、统计报表、成本核算、销售分析、市场预测、利润预估、采购订货、库存管理、工资管理等,都可通过计算机来实现。计算机的应用程度,已经成为衡量一个企事业单位现代化管理水平的重要方面。

3. 过程控制

在现代化工厂里,计算机普遍用于生产过程的自动控制。例如在化工厂中用计算机来控制配料、温度、阀门的开闭等;在炼钢车间用计算机控制加料、炉温、冶炼时间等;在制造业,企业用计算机实现程控机床的精确制造等。采用计算机进行过程控制,可大大提高生产过程的自动化水平,提高产品质量,提高劳动生产率,降低成本,提高经济效益。

用于生产过程自动控制的计算机,一般对计算机的速度要求不高,但对实时性和可靠性要求很高,否则将生产出不合格的产品,甚至造成重大设备事故或人身事故。

单片机通常用于智能仪器、仪表,它给现代日常生活带来极大方便,如可用单片机控制电冰箱、电视机、空调、洗衣机等。

4. 计算机辅助设计/计算机辅助制造(CAD/CAM)

由于计算机有快速的数值计算、较强的数据处理及模拟能力,因而目前在飞机、船舶、光学仪器、超大规模集成电路等的设计制造过程中,CAD/CAM 占据着越来越重要的地位。在超大规模集成电路等的设计制造过程中,要经过设计制图、照相制版、光刻、扩散、内部连接等多道复杂工序,是人工难以胜任的。

使用已有的计算机辅助设计新的计算机,达到设计自动化或半自动化程度,从而减轻人的劳动强度,并提高设计质量,这也是计算机辅助设计的一项重要内容。由于设计工作与图形分不开,一般供辅助设计用的计算机配备有图形显示、绘图仪等设备以及图形语言、图形处理软件等。

除用计算机进行辅助设计(CAD)、辅助制造(CAM)外,还进行辅助测试(CAT)、辅助工艺

(CAPP)、辅助教学 (CAI)等。

5. 人工智能

人类的许多脑力劳动,诸如证明数学定理、进行常识性推理、理解自然语言、诊断疾病、破译密码等都需要"智能"。

人工智能是将人脑进行演绎推理的思维过程、所采用的规则和策略、技巧等编成计算机程序,在计算机中存储一些公理和推理规则,然后让计算机去自动探索解题的方法,所以这种程序不同于一般计算机程序。

当前人工智能领域比较注重自然语言理解、机器视觉和听觉等的研究。智能机器人是人工智能各种研究课题的综合产物,它应有感知和理解周围环境,进行推理和操纵工具的能力,并有通过学习适应周围环境,完成某种动作的能力。专家系统也是人工智能研究的一个方面。

6. 信息通信

计算机网络是计算机在通信方面的重要应用,它是计算机技术和通信技术结合的产物。通过全球的计算机网络,可实现全球性情报检索、信息查询、电子商务、电子邮件等。企业网、城域网、校园网改变着人们的管理和经营方式。银行系统可通过全国性网络实现联机取、存款业务;民航、铁路系统可通过全国性网络实现异地订、售票业务;旅游系统可通过网络进行客房预订等业务。

信息高速公路是一个高速信息网络体系,可大量地、并行地、高速地传送多种信息。采用异步传输方式实现信息的高速交换,在网上同时实现数据、语音、视频通信及高清晰度的电视广播和图像传输。

1.3 计算机的基本构成

1.3.1 计算机的基本硬件结构

计算机硬件 (Hardware)和软件 (Software)是经常遇到的计算机术语。计算机硬件是指构成计算机的所有物理部件的集合。这些部件是"看得见,摸得着"的"硬"设备,故称之为"硬件"。

一般,数字计算机由五大部分构成,其硬件结构框图如图 1-3-1所示。

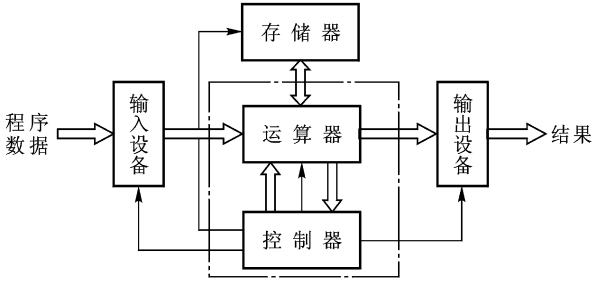


图 1-3-1 计算机硬件结构框图

⑴ 控制器

控制器是计算机的控制中枢,发布各种操作命令和控制信息,控制各部件协调工作。控制器用来实现计算机本身运行过程的自动化。控制器由时序电路和逻辑电路组成。

⑵ 运算器

运算器是对信息或数据进行处理和运算的部件,经常进行的运算是算术运算和逻辑运算。它由算术逻辑运算单元 (ALU)、寄存器、移位器和一些控制电路组成。

⑶ 存储器

存储器用来存储程序和数据,是计算机各种信息存储和交换的场所。存储器可以与运算器、控制器、输入输出设备交换信息,起存储、缓冲、传递信息的作用。

程序和原始数据以二进制形式存放在存储器中,存储器有很多存储单元,每个存储单元存放一个数据。存取信息时,应首先知道要对哪一个存储单元进行操作,为区分出不同的存储单元,就为每个存储单元进行编号,这个编号就称为存储单元的地址。

⑷ 输入设备

输入设备用于输入原始数据和程序等信息。常用的输入设备有键盘、鼠标、光电输入机等。输入的信息都是以二进制码的形式存储的,目前有语音和图像输入设备。

⑸ 输出设备

输出设备用于输出计算结果和各种有用信息。常用的输出设备有显示器、打印机、绘图仪等。磁盘既是输入设备又是输出设备。

输入设备和输出设备常合称为输入/输出设备,简称 I/O (Input/Output)设备。

运算器和控制器合在一起称为中央处理单元 (CPU, Central Processing Unit)。

1.3.2 计算机软件系统

软件是相对于硬件而言的,计算机软件指各类程序和文档资料的总和。

计算机硬件系统又称为“裸机”,计算机只有硬件是不能工作的,必须配置软件才能够使用。软件的完善和丰富程度,在很大程度上决定了计算机硬件系统能否充分发挥其应有的作用。

软件系统包括系统软件、应用软件两大类,如图 1-3-2所示。

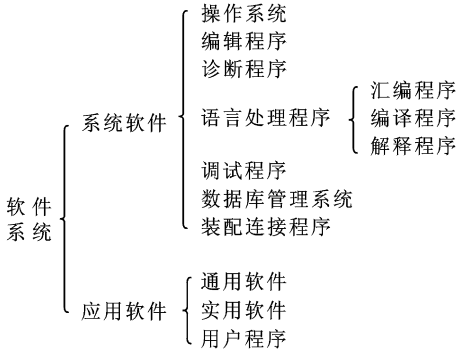


图 1-3-2 计算机软件系统

1. 系统软件

系统软件的作用是管理、调度、监控、维护计算机。包括操作系统、各种程序设计语言处理程序、监控程序、调试程序、诊断程序等。

操作系统是系统软件的核心,是计算机必须配置的软件。操作系统的任务是管理计算机的硬件和软件资源,组织、协调计算机的运行,增强系统的处理能力,提供人机接口,为用户提供方便。操作系统可分为单用户操作系统、分时操作系统、实时操作系统、网络操作系统、分布式操作系统等。

程序设计语言处理程序包括汇编、解释、编译程序,其功能是将用各种高级语言编写的程序翻译成机器能识别的二进制代码,这样计算机才能执行。

监控、调试、诊断程序是计算机的支持软件,用于维护计算机系统。

数据库管理系统,在一些资料中将其归于应用软件,它也是一个通用软件,有系统软件和应用软件的特点。计算机在信息处理、情报检索、各种管理系统中都要大量地处理某些数据,为使得检索更迅速,处理更方便,将这些数据按一定的规律组织起来,就形成了数据库。为了便于用户根据需要建立自己的数据库,查询、显示、修改数据库内容,就要建立数据库管理系统。任何应用程序要使用数据时,都必须通过数据库管理系统,这样可保证数据的安全性。

2. 应用软件

应用软件是为解决一些具体问题而编制的程序。一类是由软件公司和计算机公司开发的通用软件、实用软件,如文字处理软件、各种程序设计语言环境、各种工具软件等;另一类是用户为解决各种实际问题而开发的用户程序,如工资管理程序、档案管理程序等。

应用软件也可逐步标准化、模块化,形成解决各种典型问题的应用程序组合——软件包(Package)。

1.3.3 计算机系统的层次结构

1. 计算机语言

计算机能识别并直接执行的是一系列由二进制代码编写的指令,这些指令叫机器语言,也叫机器码。早期的计算机只提供机器语言。

使用机器语言编写程序、阅读程序、调试程序都非常困难,且必须由专业人员来做。为提高编程、读程效率,产生了汇编语言。汇编语言用助记符表示机器指令和编写程序。但机器不能识别用汇编语言编写的程序(称为源程序),必须将它转换为机器语言(称为目标程序)才能运行,转换工作是由一个叫做汇编程序的软件来实现的,其转化过程如图 1-3-3所示。

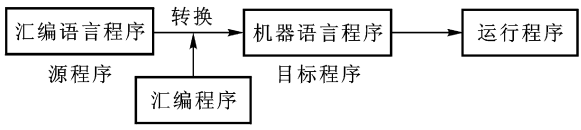


图 1-3-3 汇编语言程序转换成机器语言程序的过程

汇编语言是面向机器的,即同一个程序在不同种类的计算机中不能通用,需重新编写程序,

这样非常不方便,因此,出现了高级语言,如 BASIC 语言、FORTRAN语言、C语言等。高级语言是一种面向问题的语言,与自然语言相近,与计算机的种类无关,用高级语言编写程序,容易阅读和理解,可提高编程效率,且有较好的通用性和可移植性。

高级语言程序 源程序)同样必须转换为机器语言程序 目标程序)才能执行。实现这种转换的程序是编译程序或解释程序,图 1-3-4为转换过程。

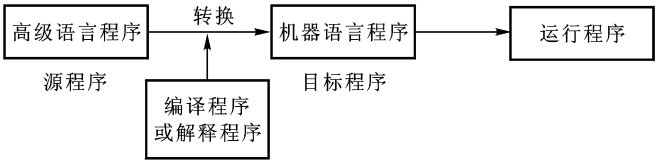


图 1-3-4　高级语言程序的转换过程

编译方式是将用户编写的源程序中全部语句转化 翻译)成机器语言程序后,再执行机器语言程序。只要源程序不变,每次运行不需再翻译,可直接运行,但源程序若有任何修改,就要重新编译。

解释方式是将用户编写的源程序的一条语句翻译成机器语言后,立即执行它,且不保留机器语言,然后,再翻译下一条语句,如此重复,直到程序结束。它的特点是翻译一次只能执行一次,当第二次重复执行时,又要重新翻译,因而效率较低。

2. 计算机系统的层次结构

使用计算机时,人们总是把用汇编语言或高级语言编写的源程序,输入给计算机,依靠计算机中的转换 翻译)程序将其翻译成机器语言程序,然后再由计算机执行。由于翻译工作是计算机做的,对用户来说,就像计算机能够直接运行汇编语言或高级语言程序,此时的计算机可以看作是在实际机器上出现的一台虚拟机。之所以称为虚拟机,是因为它安装了软件。

高级语言程序翻译成汇编语言程序或中间语言程序,而后再翻译成机器语言程序才能最终被执行。有的计算机则将高级语言直接翻译成机器语言。从这个过程可以看到计算机系统的多级层次结构 如图 1-3-5所示)。

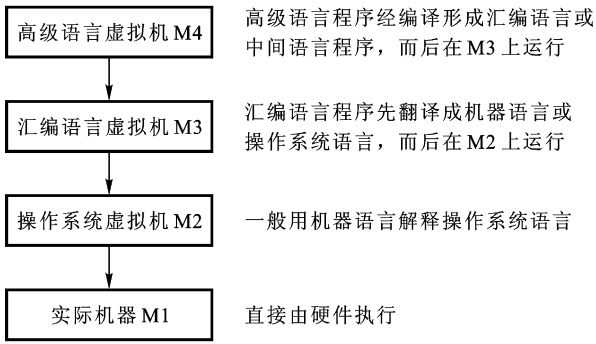


图 1-3-5　计算机系统层次结构

图 1-3-5中,在虚拟机 M2与实际机器 M1之间,还存在一种称为操作系统的软件。它提供了实际机器所没有的,但在汇编语言和高级语言的使用和实现过程中所需的基本操作和数据结构。操作系统的功能是通过操作系统的指令系统实现的,它可以看作是实际机器的扩充。

1.4 微型计算机的基本构成

1.4.1 微型计算机系统组成

微型计算机系统与任何其他电子计算机系统一样由硬件系统和软件系统两大部分组成。硬件是机器部分,即所有硬设备的集合。软件是控制系统完成操作任务的程序系统。如图 1-4-1所示为微型机系统的组成关系。

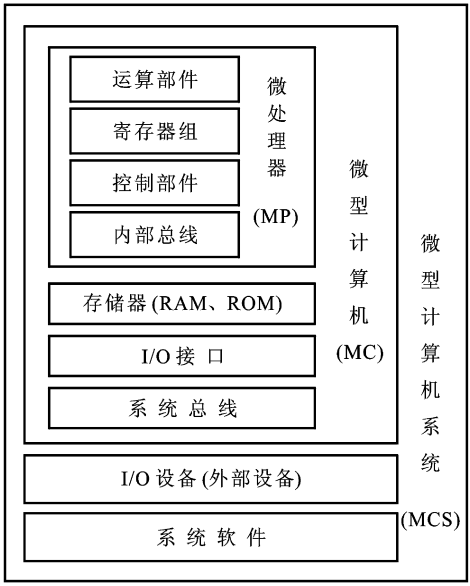


图 1-4-1 微型计算机系统组成

⑴ 微处理器 (MicroProcessor,MP)

微处理器本身不是计算机,它是将运算部件、控制部件、寄存器组和内部总线集成在一块硅片上形成的一个独立的器件。微处理器一般也称为 CPU。

微处理器的运算部件是专门用来处理各种数据信息的,可进行加、减、乘、除算术运算和与、或、非等逻辑运算。寄存器组用来保存参加运算的数据和中间结果等信息。控制器件对所执行的指令进行分析,发出各种时序控制信号,控制各部件完成相应的操作。内部总线是微处理器内部各部件之间进行信息传送的一组信息线。

⑵ 微型计算机 (MicroComputer,MC)

以微处理器为中心配上存储器、外设接口电路和系统总线构成的整体称为微型计算机。

微型机的存储器包括随机存储器（RAM）和只读存储器（ROM），用于存放程序和数据。输入/输出接口（I/O接口）是外部设备与微型机的连接电路。系统总线为 CPU和其他各部件之间提供数据、地址和控制信号的传输通道。

(3) 微型计算机系统（MicroComputer System，MCS）

以微型计算机为中心配上外部设备、软件系统和电源等，构成能独立工作的完整的计算机系统，即为微型计算机系统。

外部设备通过 I/O接口与微型计算机连接，用来实现数据的输入和输出，微型计算机的外部设备常包括显示器、键盘、鼠标、磁盘、光驱、打印机、绘图仪等，此外，还必须配上系统软件微型计算机系统才能工作，微型机常用的系统软件（操作系统）有 DOS、Windows等。

上面阐述了微型计算机系统的基本组成，同时也介绍了微处理器、微型计算机、微型计算机系统三者之间的关系。实际上通常所讲的微型机指的就是微型计算机系统。

1.4.2　微型计算机的典型结构

微型机在工作中，各功能部件之间有大量的信息相互传送，要完成这些信息的相互传送，需要有一组公共的传输线把各部件连接起来，这组公共传输线称为系统总线（Bus）。系统总线按传输的信息类别又可分为地址总线 AB（Address Bus）、数据总线 DB（Data Bus）、控制总线 CB（Control Bus）。各部件之间通过这三组总线连接起来。典型硬件结构如图 1－4－2所示。

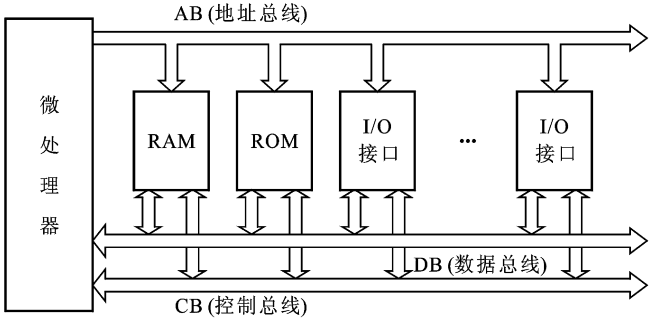


图 1－4－2　微型计算机的典型结构

地址总线 AB是传送地址信息的一组单向总线，它把 CPU要访问的外部单元地址送到存储器或 I/O口。

数据总线 DB是传送数据信息的双向总线。数据在 CPU与存储器及 CPU与 I/O接口之间的数据都通过数据总线传送，即可取出又可存入，故数据总线是双向的。通常有 8位、16位、32位、64位等。

控制总线 CB用来传送控制和状态信息，如读信号、写信号、中断信号等。有的是 CPU到存储器和外设接口的控制信号，有的是外设到 CPU的信号。控制总线既有输入线又有输出线，但每条线一般是单向的。

微型机的这种结构为单总线结构，即用一组总线（数据线、地址线、控制线）将所有部件连接起来。这种结构的缺点是每一时刻只能传输一个数据，不能多路并行传输，操作速度慢。为解决

这个问题可采用双总线结构,即部件之间使用两组相互独立的总线进行连接,这样两组总线可并行工作,提高了工作效率。还有多总线结构,但实现起来比较复杂。

1.4.3 微型计算机的典型配置

微型机硬件系统由主机和外部设备两大部分构成。主机包括主板、I/O接口卡(又称适配卡)和电源机箱等部件。微型机的外部设备(简称外设)很丰富,配置也较灵活,常用的外设有键盘、鼠标器、显示器、软盘驱动器、硬盘驱动器、光盘驱动器及打印机等。

1. 主板

微型机的主板又称系统板或母板,它是微型机硬件系统的主要部件,微型机的大部分功能芯片都装在这块印制电路板上,其组成如图 1-4-3所示。

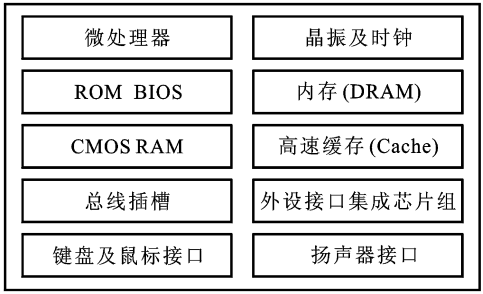


图 1-4-3 典型主机板组成框图

(1) 微处理器及时钟电路

微处理器是主板的核心部件。不同类型的微处理器构成不同性能的主板,如 486主板、Pentium主板等。一般来讲微处理器的档次越高其主板的性能就越高。

早期的微处理器芯片直接焊接在主板上,后来是插在主板的脚座上,现在一般都单独封装,插在主板的微处理器插槽上。

处理器的时钟,一般由片外的时钟电路提供,由时钟发生器对一定频率的石英晶体经过分频后得到。

(2) 内存

微型机系统的内存又叫主存,要求容量大、成本低、速度快。目前主要采用动态随机存储器(DRAM)作为内存。一般 DRAM芯片并不是直接安装在主板上,而是由若干 DRAM芯片构成直插式内存条(SIMM),插在主板的内存插槽上。内存条规格有多种,如 1 MB/条、256 MB/条等。主板上的内存条插槽一般有 4~8个,内存容量可根据用户需要任意配置,微型机的典型配置为 4 MB~1 GB。

(3) 高速缓存

大容量的动态随机存储器(DRAM)相对微处理器来说,其速度较慢,为加快微处理器访问 DRAM的速度,通常在微处理器和 DRAM之间加入一层速度接近 CPU、容量较小的随机存储器,作为高速缓存(Cache)。当 Cache位于微处理器芯片外部时,称为外部高速缓存。当 Cache位于

微处理器芯片内部时,称为内部高速缓存。一般 Cache的容量不大,在 64 KB～256 KB范围内,但速度很快,可实现 CPU访问内存的零等待。

⑷ ROM BIOS

主板上配置了一片称为固件的只读存储器 (ROM)芯片,内部固化了自检程序、基本输入/输出控制程序 (基本输入输出系统 BIOS)、系统配置程序等,称为 ROM BIOS。这些程序作为操作系统的低层程序,供启动和操作系统调用。

⑸ CMOS RAM

CMOS RAM是一种低功耗的半导体存储器。它由微型机电池供电,可长期保存信息。主要用来存储微型机系统的各种配置信息,如时钟与日期、系统口令、主存容量、磁盘参数等各种硬件参数信息。

⑹ 外围接口集成芯片组

在高档微型机中,很少再采用大量的小规模接口芯片来构成微处理器的外围接口电路,而是采用少量几片超大规模的集成 I/O芯片来实现接口电路功能。将原多种芯片的功能集成在一个组合芯片内,这样微型机主板更加简洁,系统可靠性与性能也得到增强。

⑺ 总线插槽

总线插槽是主板上用于插接 I/O接口卡的插槽,通过这些插槽可将外设 I/O卡连接到系统总线上,并通过 I/O接口卡将外设连接到主机上。主板上的总线插槽一般支持某种总线标准,如 IBM PC/XT主板总线插槽支持 8位数据传送的 PC总线,IBM PC/AT主板总线插槽支持 16位数据传送的 ISA总线,Pentium主板总线插槽支持 32位数据传送的 EISA总线或 PCI总线等。

⑻ 键盘、鼠标、扬声器接口

键盘、鼠标、扬声器接口电路一般直接集成在主板上,由单片机来控制,它负责将键盘按键产生的扫描码转换成能表示字符的 ASCII码,将鼠标器送来的电脉冲信号转换成光标的移动数据,并将数据送给 CPU,它也能将 CPU给出的声音频率数据转换成脉冲频率信号驱动扬声器发出声音。

2. I/O接口卡

一个微型机系统可配置多种输入与输出设备,它们是通过 I/O接口卡与主机连接的。常用的 I/O接口卡有多功能卡、显示卡等。

多功能 I/O接口卡集成了多种常规外设的接口驱动电路。通常一块多功能接口卡可连接两个硬盘驱动器、两个软盘驱动器、两个串行口设备、一个并行口设备。目前,高档微型机的多功能 I/O接口卡的所有功能已直接集成在主板上了。

显示卡是显示器设备的接口电路。不同类型和标准的显示器有不同的显示卡支持。

1.4.4 微型计算机的特点

微型计算机与大、中型计算机相比有很多特点,也正是由于这些特点,使其获得普及和大力发展。下面从几个方面说明。

① 体积小、重量轻、功耗低。微型计算机,由于采用大规模集成电路,使很多部件集成在一个芯片上,使整机的体积减小,重量减轻,相应的功耗也大幅度降低。

② 价格低廉。微型计算机的功能不断提高,价格不断降低,以至一般的家庭可购置微型机

使用。微型机价格低廉是其获得广泛应用的最重要的原因之一。

③ 可靠性高、使用环境要求低。由于使用了大规模集成电路，大大简化了外部接线和外加逻辑，提高了可靠性。其安装容易，可在日常的环境中使用。

④ 结构灵活、使用方便。现在的微处理器芯片及其相应支持逻辑，都有标准化、系列化产品，用户可根据不同要求构成不同规模的系统，而且组装系统较简单。微型机系统操作方便，通用软件、工具软件丰富，使用方便。

1.5 微型计算机的工作过程

下面从存储器的组织和工作过程的角度来说明微型机的工作过程。

1.5.1 存储器的组织及工作过程

存储器是用来存放数据和程序的。在计算机内部，数据和程序都是用二进制码的形式表示。一般 8位二进制码称为一个字节 (Byte)，一个或多个字节组成一个字 (Word)。存储器中每个存储单元存放一个字节或一个字，这样存储器需要很多单元来存放数据和程序，为能识别不同的单元，赋予每个单元一个编号——地址。

下面以 256个单元的存储器为例，说明存储器的组织。256个单元，每个存储单元一个编号，编号范围为 0～255，用 8位二进制码表示编号即为 00000000～11111111 (00H～FFH)，如图 1－5－1所示。

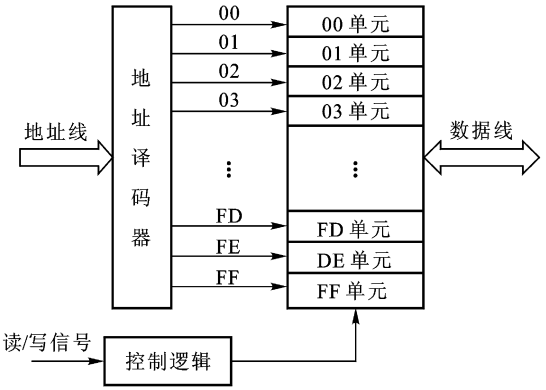


图 1－5－1 存储器组织示意图

来自地址线的地址信号，经过地址译码器的译码，选中相应的存储单元，以便从中读出信息或写入信息。控制部件控制存储器的读写过程。

存储器在进行读写工作时，先由 CPU通过地址线给出要读写信息存放的单元地址，经过地址译码器的译码，选中相应的存储单元，再由读写控制信号，经过控制逻辑来控制读出或写入。要读出信息时，选中单元的数据经数据线送往 CPU进行处理。要写入信息时，由 CPU将数据通过数据线，写入到选中的单元。

1.5.2 微型计算机的工作过程

微型机的工作过程就是执行程序的过程。程序是指令的序列,执行程序就是逐条取出程序的指令,对指令进行分析,然后完成该指令规定的操作。所以微型机的工作过程可概括为:取指令→分析指令→执行指令,如图 1-5-2所示。

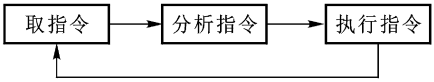


图 1-5-2 执行程序的过程

下面用示意图 1-5-3说明微型机的工作过程。

存储器通过三总线与微处理器(CPU)进行连接。程序按顺序存放在连续的存储单元中。

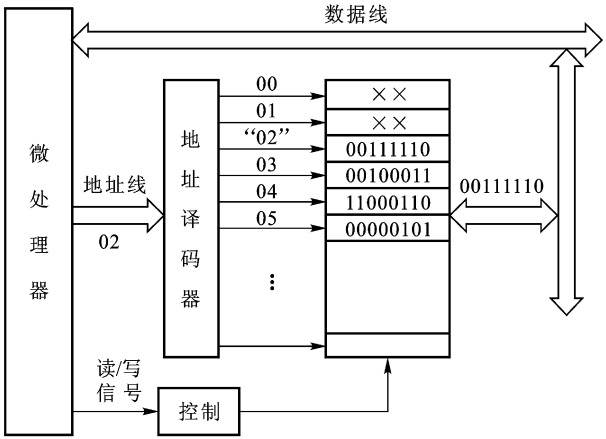


图 1-5-3 微型机工作示意图

首先,CPU给出第一条指令的地址,如为 02,通过地址总线送到地址译码器。经译码后找到存放第一条指令的 02单元。CPU发出读命令,在读命令控制下,将这条指令 00111110读出,经数据总线送 CPU。CPU对该指令进行译码分析,发一系列的控制信号,完成该指令指定的操作。

该指令执行完成后,CPU再给出下条指令的地址,继续按上述过程执行,直到整个程序运行完毕为止。

关于微处理器内部的工作过程,在第 4章中再详细介绍。

1.6 计算机的性能指标

一台计算机系统的技术性能好坏,是由它的系统结构、指令系统、外部设备配备以及软件是否丰富等多方面因素决定的。不同用途的计算机,侧重点不同。计算机的主要性能指标有如下

几个方面。

⑴ 基本字长

字长是指参与运算的数据的基本二进制位数。字长标志着计算精度，它决定了内部寄存器、运算器和数据总线的位数。一般为 8位、16位、32位、64位等。当计算机的字长确定后，为提高精度可采用双倍或多倍字长运算。

⑵ 内存容量

内存容量代表着内存储器的存储单元个数。通常计算机存储器容量以字节（Byte）为单位。可直接访问的内存容量大小，受地址总线条数的限制，如 16位地址线，可寻址的内存单元为 2^{16} ＝64 KB，24位地址线，可寻址的内存单元为 2^{24} ＝16 MB。一般微型机的内存容量在 4 MB～1 GB。

表示容量的单位有 KB（KiloByte）、MB（MegaByte）、GB（GigaByte）、TB（TeraByte）。1 KB＝2^{10} B（1024），1 MB＝1 024 KB，1 GB＝1 024 MB，1 TB＝1 024 GB。

⑶ 运算速度

运算速度以每秒执行的指令条数来表示，一般用 MIPS表示。计算机对不同指令的执行时间不同，一般用执行指令的平均时间来衡量，也可以用 CPU时钟频率来比较它们的速度，如 266 MHz，350 MHz，550 MHz和 750 MHz等。

⑷ 性能价格比

这是衡量计算机的一项综合性能指标，除考虑计算机的性能外，还要考虑其价格。性能价格比大，表明计算机性能好、价格低。

⑸ 外部设备的配置

指结构上允许配置外设的最大数量和种类。要考虑常规外设和扩充外设的情况。

⑹ 系统软件的配置

系统软件配置是否齐全，软件功能强弱，是否支持多任务、多用户操作系统等是计算机硬件性能能否充分发挥的重要因素。

习　题

1. 如何划分计算机发展的 4个阶段（第一代到第四代），当前广泛应用的计算机主要采取哪一代的技术？
2. 微型机的发展是以什么为表征的。
3. 什么是计算机硬件，什么是计算机软件？
4. 计算机硬件由哪几部分组成，各部分的作用是什么？
5. 计算机的软件是如何分类的？
6. 冯·诺依曼结构的特点是什么？
7. 微型机是如何分类的，计算机是如何分类的？
8. 计算机有哪几方面的应用？
9. 微处理器、微型计算机和微型计算机系统三者之间有什么不同？
10. 计算机能直接识别的是什么语言，汇编语言和高级语言程序如何在计算机上运行？
11. 计算机系统可分几个层次？说明各层次的特点及其相互联系。
12. 什么是总线？微型机的总线分为哪几类？
13. 画出微型机的典型结构。

14. 微型机系统的典型配置有哪几方面？
15. 概括说明计算机的工作过程。
16. 计算机有哪些主要性能指标？

第 2章　计算机中数据的表示法

输入计算机中的数据、字母、符号、汉字等信息，必须转换成“0”、“1”组合的代码形式，才能被计算机识别、处理。也就是说计算机中所有信息都是用二进制代码形式表示的。本章介绍计算机中数据信息的表示方法。

2.1　计数制及其相互转换

2.1.1　计数制

计数制是计数的方式。按进位原则进行计数的方法称为进位计数制，简称进位制。

在日常生活中，最常用的是十进制数，即“逢十进一”。除了十进制外还有六十进制（如钟表的时、分、秒）、十二进制（如 12个月为一年）等。

数据无论采用哪种进位制，都包含两个要素：基数（Radix）和位权（Weight）。

基数是某种计数制中允许的数字符号的个数。如十进制数，有 0，1，2，…，8，9这 10个数字符号，其基数为 10。在 R进制数中，基数为 R，即包含 R个不同的数字符号，每个数位计满 R就向高位进 1，即“逢 R进 1”。

一个数字符号（数码）所在的位置不同，代表数值的大小也不同。如十进制数 3456，数码“6”所在的位置是“个位”，代表 6个，“5”所在的位置是“十位”，代表 50……每个数字符号所表示的数值等于该数字符号的值乘以一个与数码位置有关的常数，这个常数叫位权，简称“权”。十进制中的“个、十、百、千……”就是各位的位权。所以每一位上的数码值与该位权的乘积表示了该位数值的大小。

位权的大小是以基数为底，数字符号所在的位置序号为指数的整数次幂。如十进制 888.8，百位的 8表示 800，即 8×10^2，10^2 为百位的位权。可见十进制数十分位、个位、十位、百位……的位权分别为 10^{-1}、10^0、10^1、10^2……注意个位的位置序号为 0。

【例 2.1】　在十进制计数制中，123.45可以表示为按权展开的多项式和的形式。

解：　　$123.45=1\times10^2+2\times10^1+3\times10^0+4\times10^{-1}+5\times10^{-2}$

一般 R进制数表示为

$$N_R=K_{n-1}\times R^{n-1}+K_{n-2}\times R^{n-2}+\cdots+K_1\times R^1+K_0\times R^0+K_{-1}\times R^{-1}+K_{-2}\times R^{-2}+\cdots+K_{-m}\times R^{-m}$$

$$=\sum_{i=n-1}^{-m}K_i\times R^i$$

其中，R——基数，表示进制。

K_i——数字符号，是 0，1，…，R－1中的任何一个。

i——位序号。

R^i——位权。

m, n——正整数。

2.1.2 计算机中常用的进位计数制

计算机中常用十进制、二进制、八进制、十六进制。实际上计算机能直接识别处理的只是二进制码,这里的十进制、八进制和十六进制是指在应用计算机的汇编语言、高级语言等情况下使用,即在虚拟机上使用的进位计数制。

为了区分各进制数,在数字后面加一个字母来标识。二进制用 B (Binary)、八进制用 Q (Octal,为防止字母 O与数字 0混淆,使用 Q表示)、十六进制用 H (Hexdecimal)、十进制用 D (Decimal) (通常省略)。

1. 十进制

十进制数是人们日常生活中使用最多的进位制,人们对其表示、运算等都非常熟悉,所以经常使用十进制数。

对任意一个十进制数 N可表示为

$$\begin{aligned}N_{10} &= K_{n-1}\times 10^{n-1}+K_{n-2}\times 10^{n-2}+\cdots+K_1\times 10^1+K_0\times 10^0+\\ &\quad K_{-1}\times 10^{-1}+K_{-2}\times 10^{-2}+\cdots+K_{-m}\times 10^{-m}\\ &=\sum_{i=n-1}^{-m} K_i\times 10^i\end{aligned}$$

基数 R=10,数字符号 K_i 为 0, 1, 2, …, 9。采用“逢十进一”计数。

2. 二进制

计算机内部对各种各样的数据、操作命令、存储地址等都使用二进制码表示。这是因为二进制数字系统有几方面优点:

① 二进制码物理上容易实现。可用电位高、低,脉冲有、无,正、负极性,开关的开、合,器件的两个稳态等来表示二进制数位上的“0”和“1”。

② 二进制数运算规则简单,可用开关电路实现。

③ 二进制码的 0和 1正好与逻辑代数的 0, 1吻合,可方便进行逻辑运算。

④ 二进制与十进制、八进制、十六进制的转换关系简单。

对任意一个二进制数 N可表示为

$$\begin{aligned}N_2 &= K_{n-1}\times 2^{n-1}+K_{n-2}\times 2^{n-2}+\cdots+K_1\times 2^1+K_0\times 2^0+\\ &\quad K_{-1}\times 2^{-1}+K_{-2}\times 2^{-2}+\cdots+K_{-m}\times 2^{-m}\\ &=\sum_{i=n-1}^{-m} K_i\times 2^i\end{aligned}$$

基数 R=2,数字符号 K_i 为 0, 1。采用“逢二进一”计数。

例 2.2 写出二进制数 1001.11B的多项式形式。

解: $(1001.11)_2 = 1\times 2^3+0\times 2^2+0\times 2^1+1\times 2^0+1\times 2^{-1}+1\times 2^{-2}$

3. 八进制

八进制与二进制有一种特殊关系,即 3位二进制码表示 1位八进制码,这样常用八进制作为二进制的书写形式。

对任意一个八进制数 N可表示为

$$N_8 = K_{n-1} \times 8^{n-1} + K_{n-2} \times 8^{n-2} + \cdots + K_1 \times 8^1 + K_0 \times 8^0 + K_{-1} \times 8^{-1} + K_{-2} \times 8^{-2} + \cdots + K_{-m} \times 8^{-m} = \sum_{i=n-1}^{-m} K_i \times 8^i$$

基数 R=8,数字符号 K_i 为 0,1,2,3,4,5,6,7。采用“逢八进一”计数。

[例 2.3] 写出八进制数 3525.76Q的多项式形式。

解：$(3525.76)_8 = 3 \times 8^3 + 5 \times 8^2 + 2 \times 8^1 + 5 \times 8^0 + 7 \times 8^{-1} + 6 \times 8^{-2}$

4. 十六进制

十六进制是计算机一种常用书写形式。采用‘逢十六进一”计数。

对任意一个十六进制数 N可表示为

$$N_{16} = K_{n-1} \times 16^{n-1} + K_{n-2} \times 16^{n-2} + \cdots + K_1 \times 16^1 + K_0 \times 16^0 + K_{-1} \times 16^{-1} + K_{-2} \times 16^{-2} + \cdots + K_{-m} \times 16^{-m} = \sum_{i=n-1}^{-m} K_i \times 16^i$$

基数 R=16,数字符号 K_i 为 0,1,2,…,9,A,B,C,D,E,F。

[例 2.4] 写出十六进制数 9A5D.7EH 的多项式形式。

解：$(9A5D.7E)_{16} = 9 \times 16^3 + 10 \times 16^2 + 5 \times 16^1 + 13 \times 16^0 + 7 \times 16^{-1} + 14 \times 16^{-2}$

常用的几种进位计数制从 0～16的表示方法列于表 2-1。

表 2-1 常用进位计数制从 0～16的表示

十进制数	二进制数	八进制数	十六进制数
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C

续表

十进制数	二进制数	八进制数	十六进制数
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

2.1.3　不同进制数之间的转换

1. 十进制数转换为二进制数

十进制数转换为二进制数应该将整数部分和小数部分分别进行转换。

⑴ 十进制整数转换成二进制整数

通常采用除 2取余法。所谓除 2取余法,就是将已知的十进制数反复除以 2,每次取其余数,若得到的余数为 1,则对应二进制数的相应位为 1,若得到的余数为 0,则对应二进制数相应位为 0,第一次得到的余数是二进制数的最低位,最后一次余数是二进制数的最高位,从低位到高位逐次进行,直至商为 0为止。

[例 2.5]　将 215D转换成二进制数。

解:

2 | 215　……余 1, K_0 =1
2 | 107　……余 1, K_1 =1
2 | 53　……余 1, K_2 =1
2 | 26　……余 0, K_3 =0
2 | 13　……余 1, K_4 =1
2 | 6　……余 0, K_5 =0
2 | 3　……余 1, K_6 =1
2 | 1　……余 1, K_7 =1
0

所以有:215D=11010111B。

⑵ 十进制纯小数转换成二进制纯小数

通常采用乘 2取整法。所谓乘 2取整法,就是将已知的十进制纯小数部分反复乘以 2,每次取其整数,若得到的整数为 1,则对应二进制数的相应位为 1,若得到的整数为 0,则对应二进制数相应位为 0,第一次乘 2得到的整数是二进制数的最高位,从高位到低位逐次进行,直至满足精度要求或乘 2后的小数部分为 0为止。设最后一次乘 2所得的整数为 K_{-m},转换后,所得的纯

二进制小数为 $0.K_{-1}K_{-2}\cdots K_{-m}$。

[例 2.6] 将 0.725D 转换成纯二进制小数。

解：

0.725		
× 2		
1.450	取 1	K_{-1}
0.450		
× 2		
0.900	取 0	K_{-2}
× 2		
1.800	取 1	K_{-3}
0.800		
× 2		
1.600	取 1	K_{-4}

如果取 4位小数能满足精度要求，则有 0.725D≈0.1011B。

(3) 十进制混合小数转换为二进制数

混合小数由整数和小数复合而成，需要将整数部分和小数部分分别进行转换，然后将转换结果组合起来即可。

[例 2.7] 将 215.725D 转换为二进制数。

解：如上求得 215D＝11010111B，0.725D≈0.1011B

则 215.725D≈11010111.1011B

2. 二进制数转换为十进制数

将二进制数转换为十进制数，只需要将二进制数按位权展开求和，便得到相应的十进制数。

[例 2.8] 将二进制数 11011.1001B转换为十进制数。

解：$11011.1001B = 1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+0\times2^{-3}+1\times2^{-4}$

$=16+8+2+1+0.5+0.0625=27.5625$

3. R进制数与十进制数转换

由如上转换可知，R进制数转换为十进制数的方法即是将所给的 R进制数用按位权展开式展开并求和即可。

[例 2.9] 将十六进制数 4E.CH 转换为十进制数。

解：$4E.CH=4\times16^1+14\times16^0+12\times16^{-1}=64+14+0.75=78.75$

十进制数转换为 R进制数仍然是将整数部分和小数部分分别进行转换。整数部分采用除 R取余法；小数部分采用乘 R取整法。转换的过程和十进制整数转换成二进制整数相同。

4. R进制数转换为 K进制数

借助于任意进制数与十进制数的转换，可以进行 R进制数与 K进制数的转换。即先将 R进制数转换成十进制数，然后再将十进制数转换成 K进制数。

5. 二进制与八进制、十六进制间的转换

二进制与八进制、十六进制间的转换可以用上述办法进行，另外还有更简捷的转换方法。

⑴ 二进制与八进制间的转换

由于有 $2^3=8$这个关系，即每 3位二进制数对应一位八进制数，所以二进制数转换成八进制数的方法是：以小数点为界，分别向左、右将二进制数每 3位分为一组，若不够 3位时，可在最高位的左边，或在小数的最右边添 0，补足 3位（不影响原数值的大小），然后将每 3位二进制数用一位八进制数表示即可完成转换。

【例 2.10】　把二进制数 1101110.0100111B转换为八进制数。

解：

001　101　110.　010　011　100

1　5　6　2　3　4

即　1101110.0100111B＝156.234Q

八进制数转换成二进制数的方法是：将一位八进制数用相应的 3位二进制数代替即可完成转换。

【例 2.11】　把八进制数 643.57Q转换成二进制数。

解：

6　4　3.　5　7

110　100　011.　101　111

即　643.57Q＝110100011.101111B

⑵ 二进制与十六进制间的转换

根据 $2^4=16$这个关系，即每 4位二进制数对应一位十六进制数。仿照二进制与八进制间的转换方法，可以进行二进制数与十六进制数的转换。

【例 2.12】　将 1110110111.1101001B转换成十六进制数。

解：

0011　1011　0111.　1101　0010

3　B　7.　D　2

即　1110110111.1101001B＝3B7.D2H

2.1.4　二进制数的运算规则

二进制数的运算包括算术运算与逻辑运算两类。算术运算有：加法运算、减法运算、乘法运算和除法运算；逻辑运算有：逻辑“与”运算、逻辑“或”运算、逻辑“异或”运算和逻辑“非”运算。下面分别介绍它们的运算规则。

⑴ 加法运算规则

0＋0＝0　0＋1＝1　1＋0＝0　1＋1＝0（进位）

【例 2.13】　计算 011011B＋010001B。

解：

$$\begin{array}{r} 011011 \\ +\ 010001 \\ \hline 101100 \end{array}$$

验算

$$\begin{array}{r} 27 \\ +17 \\ \hline 44 \end{array}$$

所以 011011B＋010001B＝101100B。

⑵ 减法运算规则

0－0＝0　0－1＝1（借位）　1－0＝1　1－1＝0

例 2.14　计算 10011B－00101B。

解：

```
  10011
－ 00101
-------
  01110
```

验证

```
  19
－ 5
----
  14
```

所以 10011B－00101B＝01110B。

⑶ 乘法运算规则

0×0＝0　0×1＝0　1×0＝0　1×1＝1

例 2.15　计算 1011B×1001B。

解：

```
  1011
× 1001
------
  1011
 0000
0000
1011
-------
1100011
```

验算

```
  11
×  9
----
  99
```

所以 1011B×1001B＝1100011B。

⑷ 除法运算

可以用乘法和减法规则完成。

⑸ 与运算规则

0∧0＝0　0∧1＝0　1∧0＝0　1∧1＝1

例 2.16　设 A＝100111，B＝101010，求 Y＝A∧B。

解：

```
  100111
∧ 101010
--------
  100010
```

所以 A∧B＝100111∧101010＝100010。

⑹ 或运算规则

0∨0＝0　0∨1＝1　1∨0＝1　1∨1＝1

例 2.17　设 A＝100111，B＝101010，求 Y＝A∨B。

解：

```
  100111
∨ 101010
--------
  101111
```

所以 A∨B＝100111∨101010＝101111。

⑺ 异或运算

0⊕0＝0　0⊕1＝1　1⊕0＝1　1⊕1＝0

例 2.18　设 A＝100111，B＝101010，求 Y＝A⊕B。

解：

$$\begin{array}{rr} & 100111 \\ P & 101010 \\ \hline & 001101 \end{array}$$

所以 A P B＝100111P 101010＝001101。

(3) 非运算

$\overline{0}=1$　$\overline{1}=0$

逻辑运算是按位进行运算，即任何一位的运算结果与前后位无关。

2.2 计算机中数值数据的表示

计算机中的数据信息分成数值数据和非数值数据（也称符号数据）两大类。数值数据包括定点数、浮点数、无符号数、数串等。非数值数据包括字母、数字、通用符号、控制符号、汉字等字符信息，还有逻辑信息，图形、图像、语音等信息。

本节先介绍数值数据在计算机中的表示方法。

2.2.1 机器数和真值

1. 机器数

数值数据在计算机中的二进制表示形式称为机器数，也就是一个数值数据的机内编码。

数有正数和负数，其正号“＋”或负号“－”在计算机中也要进行二进制编码。最简单的方法是用一位二进制的“0”和“1”来表示，用“0”代表“＋”，用“1”代表“－”。通常这个符号放在二进制数的最高位，称为符号位。这样机器中数的符号就被数值化了，符号数值化是机器数的一个特征。

小数点的位置在机器数中隐含在一个固定位置，不再占用一个数位。如何约定小数点的位置，将在后面介绍。

【例 2.19】　设 N_1 ＝＋1010 10，N_2 ＝－1001010 其机器数可表示为：N_1 ＝0101010，N_2 ＝11001010。

这是机器数的一种表示形式，一个数的机器数可有多种表示形式，不但可进行符号的数值化，数值位也可有多种编码方法。这将在后面进行介绍。

2. 真值

机器数所对应的实际数值称为机器数的真值。因机器数将符号进行了数值化，作为机器数的一位，数值部分又有不同的编码方法，这样机器数的形式值就不等于真正的数值了，为区别起见，将机器数代表的数值称为真值。

【例 2.20】　按上述编码方法，机器数 N_2 的形式值为：11001010B＝202，真值为 －1001010＝－74。

2.2.2 无符号数的表示方法

当计算机字长的所有位都用来表示数值，而不设置符号位时，称为无符号数。一个无符号二进制数表示整数时，称为无符号整数。表示纯小数时称为无符号小数。无符号整数的小数点默

认在最低位之后,无符号小数的小数点默认在最高位之前。

例 2.21 二进制数 10010000B代表无符号整数和无符号小数时,其十进制值分别是多少?

解:代表无符号整数时 $10010000B=1\times2^{7}+1\times2^{4}=144$

代表无符号小数时 $10010000B=1\times2^{-1}+1\times2^{-4}=0.5625$

对于 n位二进制数值 N,表示范围为:$0\leq N\leq 2^{n}-1$。

8位无符号整数能表示的数据范围是 00000000B～11111111B,即 0～255。

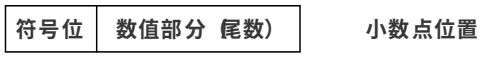
在计算机中无符号数常用来表示地址。

2.2.3 数的定点表示方法

小数点在数中的位置固定不变的数称为定点数。小数点是隐含约定的不占数据位。根据小数点的约定可分为定点整数和定点小数两种。

1. 定点整数

当约定小数点位置在机器数的最低位之后时,称定点整数。定点整数是纯整数,在计算机中的格式如下:



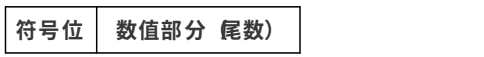
n位字长(其中一位是符号位)的定点整数(原码)所能表示的数值范围为

$$-(2^{n-1}-1)\leq N\leq 2^{n-1}-1$$

例 2.22 8位字长的定点整数表示的最小值是 $11111111=-(2^{7}-1)=-127$,最大值是 $01111111=2^{7}-1=127$。

2. 定点小数

当约定小数点位置在机器数的符号位之后,数值部分最高位之前时,称定点小数。定点小数是绝对值小于 1的纯小数,在计算机中的格式如下:



n位字长(其中一位是符号位)的定点小数(原码)所能表示的数值范围为

$$-(1-2^{-(n-1)})\leq N\leq 1-2^{-(n-1)}$$

例 2.23 8位字长的定点小数表示的最小值是 $11111111=-(1-2^{-7})=-0.9921875$,最大值是 $01111111=1-2^{-7}=0.9921875$。

无论是定点整数还是定点小数,都可以是无符号数,对有符号定点数又可有原码、反码、补码等表方法。在下一节介绍。

2.2.4 数的浮点表示方法

小数点在数中的位置是浮动变化的数称为浮点数。采用浮点数是为了扩大数的表示范围。浮点数是将一个 R进制数表示为纯小数 M和一个 R的 E次幂的乘积形式。

$$N=\pm M\cdot R^{E}$$

其中 M(Mantissa):尾数,一般为定点小数。

R (Radix)：底数，称作基数。机器数中通常取为 2。

E (Exponent)：指数，又称作阶码。

例 2.24　1100.11可以写成浮点表示方式为 0.110011×2^4。

在计算机中，一般浮点数的基数固定不变为 2，其表示形式由阶码和尾数两部分组成，如下所示。

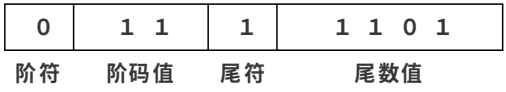
也就是说，计算机中一个浮点数的阶码和尾数要分别表示，且都有自己的符号位。阶码的符号位叫阶符，尾数的符号位叫尾符或数符。阶码反映了小数点的位置，当基数为 2时，小数点每右移一位，阶码减少 1，反之，阶码加 1。尾数的位数决定了数的精度，位数越长精度越高。

例 2.25　设一浮点数阶码部分为 3位，其中阶符占 1位，阶码占 2位；尾数部分为 5位，其中尾符占 1位，尾数占 4位，表示为：

$$N=-2^3\times 13D$$

则其二进制表示形式为 $N=2^{11}\times(-1101)$。

在机器中相应的表示形式为：



2.2.5　二一十进制数字编码

通常人们习惯于十进制数，将十进制数输入给计算机，计算机运算的结果以十进制数的形式输出而机器中需以二进制表示数，所以需要用二进制为十进制数进行编码。一般用四位二进制数表示一位十进制数，称为二进制编码的十进制数，又称二一十进制编码，即 BCD码 (Binary Coded Decimal)。它有二进制形式，又有十进制特点。

4位二进制有 16个状态，可用其中任意 10个状态表示 BCD码。所以可有多种 BCD码。常用的有 8421码、2421码、余三码、格雷码等，如表 2-2所示。

表 2-2　几种 BCD码

十进制数	8421码	2421码	余 3码	格雷码	余 3循环码
0	0000	0000	0011	0000	0010
1	0001	0001	0100	0001	0110
2	0010	0010	0101	0011	0111
3	0011	0011	0110	0010	0101
4	0100	0100	0111	0110	0100
5	0101	1011	1000	1110	1100

续表

十进制数	8421码	2421码	余 3码	格雷码	余 3循环码
6	0110	1100	1001	1010	1101
7	0111	1101	1010	1000	1111
8	1000	1110	1011	1100	1110
9	1001	1111	1100	0100	1010

下面主要介绍 8421 BCD 码。

8421 BCD 码是计算机中使用最广泛的一种 BCD 码。8421是指 4位二进制码各位的权。知道了各位的权,就能方便地得到相应的十进制数。

例 2.26 给出 8421 BCD 码 001010000110表示的十进制数。

解:

0010 1000 0110

2 8 6

所以,8421BCD码 001010000110所表示的十进制数为 286。

注意,十进制数的 BCD 码表示,与十进制所对应的二进制数是不同的。

例 2.27 十进制 286,其 8421 BCD 码为 001010000110,对应的二进制数为 100011110。

2.3 计算机中带符号数的表示

为了方便地对机器数进行运算,提高运算速度和效率,人们对带符号数设计了多种编码方法,常用的编码方法有原码、反码和补码。

2.3.1 原码

机器数的最高位为符号位,“0”代表正数,“1”代表负数,其余位给出数值的绝对值,这样的表示方法即为原码表示。

(1) 定点小数的原码表示

$$[X]_{原}=\begin{cases}X, & 0\le X<1\\ 1-X, & -1<X\le 0\end{cases}$$

例 2.28 $X_1=+0.1011$ $[X_1]_{原}=0.1011$

$X_2=-0.1011$ $[X_2]_{原}=1.1011$

(2) 定点整数的原码表示

$$[X]_{原}=\begin{cases}X, & 0\le X<2^{n-1}\\ 2^{n-1}-X, & -2^{n-1}<X\le 0\end{cases}$$

例 2.29 $X_1=+1011$ $[X_1]_{原}=01011$

$X_2=-1011$ $[X_2]_{原}=11011$

当 n=8时，$-127 \leq [X]_{原} \leq 127$。

原码的性质：

性质 1：$[X]_{原}$ =符号位 + |X|。

性质 2:原码的零有两种表示法：

$[+0]_{原} = 00\cdots0$

$[-0]_{原} = 10\cdots0$

即在原码机器数中，遇到这两种情况都做“0”处理。

原码表示直观、简单，与真值转换很方便，易实现乘除运算。但是原码进行加减运算时，符号位不能同数值一样参加运算，需通过判断参加运算数的符号和绝对值的情况，来决定如何作运算，这样的运算不方便，且速度慢。

2.3.2 反码

机器数的最高位为符号位，“0”代表正数，“1”代表负数，数值部分若为负数按位取反，正数不变，这样的表示方法即为反码表示。

(1) 定点小数的反码表示

$$[X]_{反} = \begin{cases} X, & 0 \leq X < 1 \\ (2-2^{-n}) + X, & -1 < X \leq 0 \end{cases}$$

例 2.30] $X_1 = +0.1011$ $[X_1]_{反} = 0.1011$

$X_2 = -0.1011$ $[X_2]_{反} = 1.0100$

(2) 定点整数的反码表示

$$[X]_{反} = \begin{cases} X, & 0 \leq X < 2^{n-1} \\ (2^n-1) + X, & -2^{n-1} < X \leq 0 \end{cases}$$

例 2.31] $X = +1011$ $[X]_{反} = 01011$

$X = -1011$ $[X]_{反} = 10100$

当 n=8时，$-127 \leq [X]_{反} \leq 127$。

反码的性质：

性质 1:零的反码有两种表示法：

$[+0]_{反} = 00\cdots0$

$[-0]_{反} = 11\cdots1$

性质 2:当 X为正数时 $[X]_{反} = [X]_{原}$;当 X为负数时符号为 1,其他位按位取反，即得反码。

2.3.3 补码

1. “模”和“补数”的概念

计算机的补码来源于数学上的“模”和“补数”。为了介绍补码，先介绍“模”和补数的概念。

通常把计数器的容量称为“模”或“模数”，记为模 M或 mod M。

例如，一个钟表，模 M=12,将表针顺时针拨 5点和逆时针拨 7点，表针的位置相同。即在模 12的情况下，$+5 = -7 \pmod{12}$

一个 4位二进制计数器,它的量程为 2^4,所以它的模为 $M=2^4=16$,计数范围为 0000B～1111B。当计数计到 1111B (15) 时,再加 1,则应为 10000B (2^4),但此时计数器中的 4位数变成 0000B,也就是说又重新开始计数;加 111B (7),则计数器中的 4位数变成 0111B,这与 2^4 减 1001B (9) 所得数相同。即 $+7=-9 \pmod{16}$。

我们将关系 $+7=-9 \pmod{16}$,称作 +7是 −9对于模 16的补数,或说 +7与 −9关于模 16同余。同理将关系 $+5=-7 \pmod{12}$,称作 +5是 −7对于模 12的补数,或说 +5与 −7关于模 12同余。

一般地,若两个数 a、b用模 M除,所得的余数相等时,称 a和 b对模 M是同余的,当 a、b对模 M同余时,就称 a、b在以 M为模时是相等的,记为

$$a=b \pmod{M}$$

对钟表来说, $14=2 \pmod{12}$,即 14点就是 2点。

由同余概念不难得出 $M+a=a \pmod{M}$

当 a为负时,如 $a=-7 \bmod 12$

则 $12+(-7)=-7 \pmod{12}$,即 $5=-7$。

这样以 12为模时,减 7可转化为加 5运算,对钟表来说 10点减 7点与 10点加 5点都是 3点。

变减法运算为加法运算,这是计算机中采用补码的原因。

由上述可得 $[X]_{补}=模+X$

2. 补码

机器数的最高位为符号位,“0”为正,“1”为负,其余位按 2 (2^n) 取模的结果给出数值,这样的表示方法即为补码表示。

在定点机中,如果字长为 n位,表示整数时模为 2^n,表示纯小数时模为 2。

(1) 小数的补码表示

$$[X]_{补}=\begin{cases} X, & 0\le X<1 \\ 2+X, & -1\le X\le 0 \end{cases}$$

例 2.32 $X=+0.1011$ $[X]_{补}=0.1011$

$X=-0.1011$ $[X]_{补}=1.0101$

(2) 整数的补码表示

$$[X]_{补}=\begin{cases} X, & 0\le X<2^{n-1} \\ 2^n+X, & -2^{n-1}\le X\le 0 \end{cases}$$

实际可用一个式子表示

$$[X]_{补}=2+X \pmod{2}$$

或

$$[X]_{补}=2^n+X \pmod{2^n}$$

当 $n=8$时, $-128\le [X]_{补}\le 127$。

求一个数的补码,可根据定义来求。综合求补码的过程可以发现,可用下面简便办法求补码:最高位是符号位,“0”代表正数,“1”代表负数;对于正数,补码的数值部分为其真值的数值部分不变;对于负数,其真值的数值部分每位按位求反,再将最低位加 1即得补码的数值部分。

例 2.33　已知　$X_1=01110011, X_2=-0110011$

则有　　$[X_1]_{补}=01110011$，$[X_2]_{补}=10011101$。

3. 补码的性质

性质 1：当 X为正数时，$[X]_{补}=[X]_{原}$；

当 X为负数时，符号位为 1，数值部分各位按位取反后末位加 1。

性质 2：在补码表示中，0有唯一的编码，即 $[+0]_{补}=[-0]_{补}=00\cdots0$。

这可由定义直接求出。$[+0]_{补}=0$，$[-0]_{补}=2^n+(-0)=2^n=0$。

性质 3：当 $X=-2^{n-1}$时，$[X]_{补}=2^n+(-2^{n-1})=2^{n-1}$。

这表明，-2^{n-1}的补码存在，且等于 2^{n-1}。

例 2.34　当 n=8时　若 $X=-2^7=-10000000B=-128$

则　　$[X]_{补}=2^7=10000000B$。

性质 4：补码加法运算时，可连同符号位一起运算，只要不溢出，结果为和的补码，即

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}。$$

例 2.35　$[X]_{补}=11011001$，$[Y]_{补}=00111000$

则　$[X+Y]_{补}=[X]_{补}+[Y]_{补}=11011001+00111000=00010001$

性质 5：$[[X]_{补}]_{补}=[X]_{原}$

例 2.36　已知 $X=-1110011$

则　$[X]_{补}=10001101$，$[[X]_{补}]_{补}=11110011=[X]_{原}$

性质 6：补码的算术移位，将 $[X]_{补}$的符号位与数值位一起右移一位，相当于是求 $[X/2]_{补}$，求得的结果应保持原符号位不变，即最高位补符号位。

例 2.37　已知　$[X_1]_{补}=01110011$，$[X_2]_{补}=10001111$

则　　$[X_1/2]_{补}=00111001$，$[X_2/2]_{补}=11000111$。

在移位过程中最低位被丢掉。当无溢出时 $[2X]_{补}$为 $[X]_{补}$左移一位，符号位不变。

性质 7：对 $[X]_{补}$连同符号位一起按位取反，末位加 1，即可得 $[-X]_{补}$。这称为变补。

例 2.38　已知　$[X_1]_{补}=01110011$，$[X_2]_{补}=10011101$

则有　　$[-X_1]_{补}=10001101$，$[-X_2]_{补}=01100011$

注意，无论正数还是负数，变补的方法相同，读者可自行举例验证。

2.3.4　变形补码

变形补码即双符号位补码或称模 4补码。

变形补码的定义：对于任意二进制数 X，有

$$[X]_{补}=\begin{cases}X, & 0\le X<1\\ 4+X, & -1\le X\le 0\end{cases}$$

例 2.39　$X=+0.1011$，$[X]_{补}=00.1011$。

$X=-0.1011$，$[X]_{补}=11.0101$。

即在补码的基础上增加一位符号位。

变形补码的性质：

性质 1:在运算时,若没有溢出,两个符号位相同。

符号是“00”表示正;符号是“11”表示负。

性质 2:运算时符号位不同,表示溢出。

符号是“01”表示上溢出,符号是“10”表示下溢出。上溢出实际为两个正数之和大于 1(纯小数);下溢出实际为两个负数之和小于 −1(纯小数)。若为整数,则超出 n位二进制数的表示范围。

变形补码常用于判断溢出。

例 2.40 $X_1 = +0.1010$, $[X_1]_{补} = 00.1010$

$Y_1 = -0.1011$, $[Y_1]_{补} = 11.0101$

则有 $[X+Y]_{补} = [X]_{补} + [Y]_{补}$

$= 00.1010 + 11.0101 = 11.1111$ (没有溢出)

例 2.41 $[X_2]_{补} = 11.1001$, $[Y_2]_{补} = 11.0101$

则有 $[X_2 + Y_2]_{补} = [X_2]_{补} + [Y_2]_{补}$

$= 11.1001 + 11.0101 = 10.1110$ (溢出,下溢出)

例 2.42 $[X_3]_{补} = 00.1011$ $[Y_3]_{补} = 00.1101$

则有 $[X_3 + Y_3]_{补} = [X_3]_{补} + [Y_3]_{补}$

$= 00.1011 + 00.1101 = 01.1000$ (溢出,上溢出)

小结:带符号的数在计算机中可以表示为原码、反码和补码,对于正数,它的原码、反码和补码是相同的,对于负数,它的原码、反码和补码的符号位都为 1,其数值部分是:原码就是其原数值;反码是将原数值按位求反即可;补码是将原数值按位求反后,最低位加 1。

8位二进制数的原码、反码、补码的对照如表 2−3所示。

表 2−3 几种编码的对照

真 值	原 码	反 码	补 码
+127	0111 1111	0111 1111	0111 1111
+126	0111 1110	0111 1110	0111 1110
…	…	…	…
+1	0000 0001	0000 0001	0000 0001
+0	0000 0000	0000 0000	0000 0000
−0	1000 0000	1111 1111	0000 0000
−1	1000 0001	1111 1110	1111 111
…	…	…	…
−126	1111 1110	1000 0001	1000 0010
−127	1111 1111	1000 0000	1000 0001
−128	—	—	1000 0000

2.4 计算机非数值数据的编码

计算机除了要进行数值计算外,还要进行大量字符、文字等非数值信息的处理。无论是什么样的信息,在计算机内部都必须以二进制码的形式进行存储、处理。如何用二进制表示非数值数据,即为非数值数据的编码。本节介绍非数值数据的编码方法。

2.4.1 字符的编码

字符包括字母、数字、通用符号等。用一组二进制码的不同组合可分别表示这些字符。如何进行表示,其编码方法很多,常用的编码有 ASCII码和 EBCDIC码。

ASCII是 American Standard Code for Information Interchange的英文缩写,即美国标准信息交换代码,它用 7位二进制进行编码。7位二进制共有 128种不同的组合(状态),可表示 128个字符。ASCII码选择了国际上最通用的字符(共 128个)进行编码。这些字符包括:

① 字母:A, B, …, Z, a, b, …, z。

② 数字:0, 1, 2, …, 9。

③ 通用符号:+、−、*、$、各种括号、空格等。

④ 通用控制符。

ASCII码编码表如表 2-4所示。

表 2-4 常用字符的 ASCII字符编码

$b_6\ b_5\ b_4$ / $b_3\ b_2\ b_1\ b_0$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	、	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	−	o	DEL

非显示字符的控制功能如表 2-5所示。

表 2-5 非显示字符及其控制功能

非显示字符	控制功能	非显示字符	控制功能
NUL (NULL)	空	SOH (Start of Heading)	标题开始
STX (Start of Text)	正文开始	ETX (End of Text)	正文结束
EOT (End of Transmission)	传送结束	ENQ (Enquiry)	询问
ACK (Affirmative Acknowledgement)	承认	BEL (Bell)	响铃
BS (Back Space)	退格	HT (Horizontal Table)	横向列表
LF (Line Feed)	换行	VT (Vertical Table)	垂直列表
FF (Form Feed)	换页	CR (Carriage Return)	回车
SO (Shift Out)	移出	SI (Shift In)	移入
DC0 (Device Control 0)	设备控制 0	DC1 (Device Control 1)	设备控制 1
DC2 (Device Control 2)	设备控制 2	DC3 (Device Control 3)	设备控制 3
DC4 (Device Control 4)	设备控制 4	SYN (Synchronous)	同步
NAK (Negative Acknowledge character)	否认	CAN (Cancel)	作废
ETB (End of Transimission Block)	块传送结束	EM (End of Medium)	纸尽
SUB	减	ESC (Escape)	换码
FS (File Separator)	文件分隔符	GS (Group Separator)	组分隔符
RS (Recod Separator)	记录分隔符	US (Unit Separator)	单元分隔符
		DEL (Delete)	作废,删除

128个字符分为两大类,一类是可显示字符,另一类是非显示字符。

可显示字符的特点是,可从键盘上输入,可在屏幕上显示,可在打印机上打印出来。其编码范围为 0100000～1111110,共 95个。

非显示字符主要用来控制输入输出设备,是不可显示的。其编码范围为 0000000～0011111 的 32个和 1111111共 33个。

一般用 8位二进制编码表示字符,低 7位表示字符,最高 1位一般用作校验位。

常用的编码方法还有 EBCDIC码 (Extended Binary-Code-Decimal Interchange Code,扩充的二－十进制交换码)。EBCDIC码编码方式与 ASCII码类似,只是扩充使用 8位二进制进行编码。8位 256个状态,可为 256个字符进行编码。EBCDIC码无论是显示字符还是非显示字符比 ASCII码都有扩充。

2.4.2 汉字的编码

汉字也是一种字符,也需用“0”、“1”组合进行编码,才能被计算机接受。汉字是象形文字,有近 60 000个左右的汉字,常用汉字就有 7 000个左右。汉字的编码处理与西文的拼音文字有较大区别,汉字信息处理较复杂,它涉及输入码、内码、字型码等多种编码。

1. 汉字输入码

为了能直接在键盘上输入汉字,就需要为汉字进行相应的输入编码。采用输入码,就是通过

键盘的字母、数字等实现汉字的输入。

常用的输入编码方法有数字编码、字音编码、字型编码。

数字编码常用的是国标区位码，电报码等。国标区位码字符集共收录 6 763个汉字，分两级，一级字库和二级字库，用 4位十进制数字串代表一个汉字。该编码方法是将汉字按行（区）、列（位）排列，共设 94行，每行为一个区，每区 94个汉字。每个汉字对应两位十进制区号和两位十进制位号。该编码无重码，但难记忆。

字音编码是以汉语拼音为基础的编码。可输入 6 700多个汉字及一万多个词汇。是普通使用者常使用的编码。该编码方法重码多。

字型编码是用汉字的形状来进行编码。常用的有五笔字型输入码。字型编码重码少。

2. 汉字内码

汉字内码是汉字在计算机内部存储、交换、检索等处理的信息代码。

无论采用何种输入码进行汉字的输入，为存储、处理方便，都要转换成长度一致的汉字内码。常使用的内码是以 GB2312—80码为基础的编码。该编码用两个连续的字节表示一个汉字，且这两个字节最高位均为 1，与西文字符区别。它最多可表示汉字数为：128×128＝16 384个。

3. 汉字字型码

汉字字型码是计算机中用于输出（显示、打印等）汉字的一种编码，它是用汉字点阵表示的汉字字型代码。在字型点阵中，笔画经过的点为 1，其他点为 0。由于汉字有多种字体，字型不同，其字型点阵也不同。所有汉字字符集的字型点阵构成字型库，需要显示、打印时，根据汉字内码向字型库检索出该汉字的字型信息后，进行输出。

汉字编码处理过程如图 2－4－1所示。

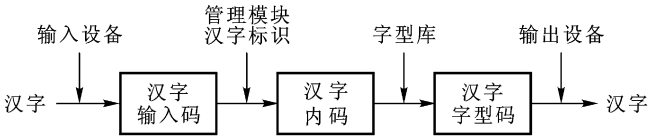


图 2－4－1　汉字编码处理过程

2.5　数据校验码

数据在计算机系统内的存取、传送过程中，可能产生错误，诸如“1”误变为“0”或“0”误变为“1”等。为提高数据传输的正确性，一方面要提高硬件电路的可靠性和抗干扰能力，另一方面要在数据的编码上采取检错纠错的措施。数据校验是通过少量电路和软件的方法发现某些错误，甚至确定错误的性质和出错的位置，进而改错的一种措施。

数据校验码（Check Code）是一种常用的带有发现某些错误或带有自动改错能力的数据编码方法。实现原理是在合法的数据之间，加进一些不允许出现的编码（非法码），使得当合法数据出现错误时，就成为非法码，通过检测码的合法性而发现错误。

这里要用到码距的概念。码距是指任意两个合法码之间，不相同二进制位数的最小值。当

仅有一位不相同时，称其码距为 1，例如用 4位二进制表示 16种状态，则 16种编码都用到了，每个状态都合法，码距为 1，此时无查错能力。因为任何一位出错，都变成另一个合法码，无法检错。

合理安排编码数量和规则，就可提高查错能力和纠错能力。如果采用如下编码来表示 0～7，即 0000，0011，0101，0110，1001，1010，1100，1111，则码距变成 2，即任意两个码之间，至少有 2个二进制位不同。这样若有一位出错，则变成非法码，可查出一位错。可见增加码距可增加查错能力。

校验码的种类很多，这里只介绍奇偶校验码、交叉校验和循环冗余校验码。

2.5.1 奇偶校验码

奇偶校验码是一种开销小，能发现数据代码中一位出错情况的一种数据校验编码，常用于存储器读/写检查或 ASCII字符传送过程中的检查。

实现原理是在 n位数据上增加一位二进制位作为校验位（Parity Bit），形成 n+1位校验码，使码距由 1增加到 2。若合法码中有一个二进制位的值出错了，由 1变成 0或由 0变成 1，这个码都将成为非法编码。

实现方法是，将校验位放在码的最高位或最低位，校验位的取值，使整个校验码（数据位和校验位）含有 1的个数为奇数或偶数个（奇校验或偶校验），当某一位出错时，则整个码中 1的个数奇偶性发生变化，从而发现了某位有错。在使用奇数个 1的方案进行校验时，称为奇校验（Odd Parity），反之，则称为偶校验（Even Parity）。

这种方法只能发现 1位错，不能定位和纠错。

下面是对几个字节数值的奇校验和偶校验的例子。

数据	奇校验的编码	偶校验的编码
00000000	100000000	000000000
01010101	101010101	001010101
01111100	001111100	101111100
11111111	111111111	011111111

该例中，最高位为校验位，其余低 8位为数据位，校验位的值为 1还是为 0，是由数据位中 1的个数的奇偶性决定的。

计算机中有专门的奇偶检测电路负责对校验码进行检测。当检测无错时，自动去掉校验位，取出有效数据。

2.5.2 交叉校验

当一次传送一个数据块时，可采用交叉校验的方法，即不仅每个字设置 1个奇偶校验位（称为水平校验位），而且数据块的同一位也设置奇偶校验位（称为垂直校验位），对数据块水平、垂直方向同时进行校验，这种校验方法叫交叉校验。

设有 4个字节的信息组成数据块，约定水平、垂直都采用偶校验，校验位的位置和取值见表 2－6。

校验位的取值使得每行代码每列代码含 1的个数均为偶数。

传输过程中,若只有某字节的 1位发生错误,则可通过两个方向的奇偶检验,查出有错误,且能确定出错的位置。若两位出错,用此方法可查出有错误,但不能定位是哪一位出错。

假设传输过程中,使第 2字节的 a_3 位发生错误,由 1变成 0,则第 2字节的水平校验码 1的个数变成 3个,不符合偶校验规则,同时 a_3 所在的列也发生了 3个 1的校验错,行列交叉点可确定出错的位置是第 2字节的 a_3 位出错。

表 2-6　交叉校验校验位的取值

取值位 / 字节	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	水平校验位
第 1字节	1	0	0	1	1	0	0	1	0
第 2字节	0	0	1	0	1	1	0	0	1
第 3字节	1	1	1	0	0	1	1	1	0
第 4字节	0	1	1	1	1	1	1	0	0
垂直校验位	0	0	1	0	1	1	0	0	

若第 3字节的 a_0、a_3 位同时发生错误,由 1变成 0,则第 3字节的水平校验码 1的个数变成 4个,仍是偶数个,符合偶校验规则,不能发现错误;但 a_0、a_3 所在的列同时发生了奇数个 1的偶校验错,说明有 2位发生错误,但不能定位出错的位置。

2.5.3　循环冗余校验码

循环冗余校验码 (Cyclic Redundancy Check, CRC),是一种具有很强纠错能力的校验码。该校验码在计算机信息通信中被广泛采用。

1. CRC码的检错方法

CRC码由两部分组成,左边是信息码,右边是校验码。对 n位信息位,校验位占 k位,CRC码共计为 n+k位,如图 2-5-1所示。

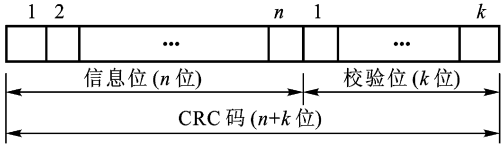


图 2-5-1　CRC码格式

n位信息代码左移 k位后,被一个约定好的生成表达式 (k+1位)相除,相除后得到 k位余数,该余数即为 k位校验位,将校验位拼接在 n位信息位的后面,形成 n+k位 CRC码。

CRC码有能被生成表达式整除的特征,当校验时,用 CRC码与生成表达式相除,若正好除尽,表明无信息错,否则说明有信息错。

2. 校验位的计算——模 2运算

校验位是通过被校验的信息代码左移 k位后,被一个约定好的生成表达式相除后得到的,这种除法是基于‘模 2运算’的多项式除法。

模 2运算时不考虑加减的进借位,即按位运算,规则是

$$0\pm0=0 \quad 0\pm1=1 \quad 1\pm0=1 \quad 1\pm1=0$$

作模 2除法时,上商的原则是,当部分余数首位是 1时商为 1,为 0时商为 0,然后按模 2相减求得余数,部分余数不计高位。最后余数比除数少 1位,此余数就是校验位。

例 2.43 设信息码为 1101,选择生成表达式为 $x^3+x^0=1001$,试计算校验位并写出 CRC码。

解:因生成表达式是 4位 =k+1,所以 k=3位,将信息码左移 3位为 1101000,除以 1001,如下计算式所示。

```
          1100
1001 ) 1101000
       1001
       ----
        1000
        1001
        ----
         0010
         0000
         ----
          0100
          0000
          ----
           100   <-------- 校验位
```

校验位是 100,CRC码为 1101100。

若想校验数据信息是否出错,可把 CRC码除以同一表达式,如果信息无错,则余数为 0,如果有错,则余数不为 0。下面两式分别为无错和 CRC码有一位错变成 1100100时的校验过程。

CRC码无错时

```
          1100
1001 ) 1101100
       1001
       ----
        1001
        1001
        ----
         000      余数为 0
```

CRC码出错变成 1100100时

```
          1101
1001 ) 1100100
       1001
       ----
        1011
        1001
        ----
         0100
         0000
         ----
          1000
          1001
          ----
           001      余数不为 0
```

因当余数不为 0时,继续模 2相除,余数值将出现反复循环,这就是‘循环码’的由来。又因校验位扩充了数据码的位数,所以是一种基于“冗余校验”思想的校验方法。

3. CRC码的纠错原理

选择适当的生成表达式,在信息码长度确定时,余数与 CRC码出错位位置的对应关系不变,由此可以用余数作为判断出错位置的依据,而达到纠错的目的。

4. 生成表达式

生成表达式的位数越多，校验能力越强。并不是任一个 k+1位的二进制数都可以作为生成表达式。要作为生成表达式它应满足一些要求，比如，任一位发生错误时都能使检错时的余数不为 0，不同位发生错误时应当使余数不同等。

常用的生成表达式有：

$x^{12}+x^{11}+x^{3}+x^{2}+x^{1}+1$

=1100000000111

$x^{16}+x^{15}+x^{2}+1$

=1100000000000101

$x^{16}+x^{12}+x^{5}+1$

=1000100000001000001

$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$

=100000100110000010001110111110111

习　　题

1. 将下列十进制数转化成二进制数和八进制数（二进制数取 3位小数，八进制数取 1位小数）。

73.5　　64　　725.9375　　25.34

2. 把下列各数化成十进制数。

101.10011B　　22.2Q　　1AD.4H

3. 完成下列二进制数的运算。

(1) 101.111+11.011

(2) 1001.10−110.01

(3) 101.01×11.01

(4) 101110111÷1101

4. 写出下列二进制数的原码、反码和补码。

(1) 0.1010　(2) −0　(3) −0.1010　(4) 0.1111　(5) −0.0100

5. 已知 X的原码为下述各值，求 X的补码。

(1) 0.10100　(2) 1.10111　(3) 1.10110

6. 已知 X的补码为下述各值，求 X的真值。

(1) 0.1110　(2) 1.1100　(3) 0.0001　(4) 1.1111　(5) 1.0001

7. 试给出 8位二进制数表示无符号数、原码、反码、补码时的最小值。

8. 已知 X=0.0110，Y=−0.0101，试求：$[X]_{补}$，$[Y]_{补}$，$[X/2]_{补}$，$[2X]_{补}$，$[Y/2]_{补}$，$[2Y]_{补}$。

9. 已知 $[X]_{补}$值如下，用变补方法求出 $[-X]_{补}$。

(1) 01010110　　(2) 10111001

(3) 11100011　　(4) 00010111

10. 按要求填空。

(1) 17D=（　）BCD　　(2) 10011001BCD=（　）D

(3) 01001101B=（　）BCD　　(4) 01101000BCD=（　）B

11. 用变形补码进行下列 $[X]_{补}+[Y]_{补}$运算，并判断有无溢出。

(1) X=+0101　　Y=+0111

(2) X=+1101　　Y=−0010

(3) X＝－0110　　Y＝＋1011

(4) X＝－1101　　Y＝－0111

12. 设机器字长 16位,定点表示时,数值 15位,符号位 1位;浮点表示时,阶码 6位 (其中阶符 1位),尾数 10位 (其中数符 1位),阶码底为 2。试求：

(1) 定点原码整数表示时,最大正数、最小负数各是多少？

(2) 定点原码小数表示时,最大正数、最小负数各是多少？

(3) 浮点原码表示时,最大浮点数和最小浮点数各是多少？

13. 写出 D、H、3、9、a 的 ASCII码。

14. 说明汉字输入码、汉字内码、汉字字形码的作用。

15. 如果采用偶校验,下述两个数据的校验位各是什么？分别写出校验码。

(1) 01010110　　(2) 00110011

16. 设有信息码 16位,如果采用循环冗余校验,采用 13位的生成表达式,需要多少个校验位,应放在什么位置上？

17. 已知信息码为 100101,生成表达式为 x^3+x+1,试计算 CRC校验的校验位,写出 CRC码。说明 CRC码校验出错的方法。

第 3章　运算器与控制器

运算器是计算机加工处理数据的功能部件。运算器对数据的加工处理包括:数值数据的算术运算,主要是加、减、乘、除等;逻辑数据的逻辑操作,主要有与、或、非、异或等运算。实现算术运算和逻辑运算是运算器的核心功能,运算器中完成这些功能的核心部件是算术逻辑单元 (ArithmeticLogicalUnitALU),ALU也是运算器内部传送数据的重要通路。

运算器除进行算术、逻辑运算外,还暂存参加运算的数据和中间结果,选择参加运算的数据,所以运算器内包含一定数目的通用寄存器和多路选择器、译码电路等。

目前应用最普遍的运算器可分为定点运算器和浮点运算器两类。定点运算器用硬件直接实现,是必备的运算器。浮点运算器又称为数学协处理器,主要用硬件完成,是可选件。现在的高档微型机已经把定点运算器和浮点运算器集成在一块芯片中。

控制器是整个计算机系统的指挥中心,协调并控制计算机的各个部件执行程序的指令序列。早期计算机的各部件由多个机柜的若干插板构成,随着大规模集成电路技术的发展,出现了微处理器,微处理器将运算器与控制器集成在一个芯片上,通常称为中央处理单元 (CentralProcessing Unit,CPU)。

根据工作原理,控制器可分为组合逻辑控制器 (又称硬布线逻辑控制器)和微程序控制器。

本章主要介绍运算器和控制器的基本组成及工作原理。

3.1　算术逻辑运算的基本电路

在介绍运算器之前,先介绍构成算术逻辑运算单元的基本电路,加法单元和加法器。

3.1.1　加法单元

加法单元是构成加法器的单元电路,是基本运算部件,包括半加器和全加器。

1. 半加器

不考虑进位时的两个二进制位 X_i、Y_i 相加称为半加,实现半加的电路称为半加器。其真值表如表 3-1所示,逻辑表达式如下:

$$H_i = X_i \cdot \overline{Y}_i + \overline{X}_i \cdot Y_i$$
$$= X_i \oplus Y_i$$

表 3-1　半加器真值表

X_i	Y_i	H_i
0	0	0
0	1	1
1	0	1
1	1	0

如图 3-1-1所示为实现电路。

2. 全加器

考虑低位进位时,两个二进制位相加为全加,即 X_i、Y_i 和进位 C_{i-1} 相加。实现全加的电路为全

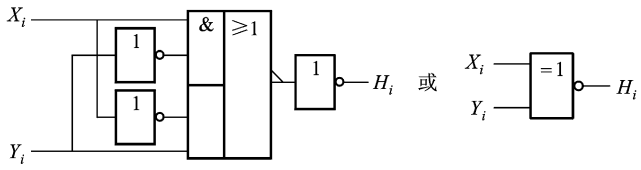


图 3-1-1 半加器逻辑图

加器,真值表如表 3-2所示。

由真值表有下列逻辑表达式

$$F_i=\overline{X_i}\,\overline{Y_i}C_{i-1}+\overline{X_i}Y_i\overline{C_{i-1}}+X_i\overline{Y_i}\,\overline{C_{i-1}}+X_iY_iC_{i-1}$$

$$C_i=\overline{X_i}Y_iC_{i-1}+X_i\overline{Y_i}C_{i-1}+X_iY_i\overline{C_{i-1}}+X_iY_iC_{i-1}$$

此式可直接由组合逻辑电路实现,也可化简为

$$F_i=X_i\bar{\imath}\ Y_i\bar{\imath}\ C_{i-1}$$

$$C_i=X_iY_i+X_iC_{i-1}+Y_iC_{i-1}=X_iY_i+(X_i\bar{\imath}\ Y_i)\,C_{i-1}$$

逻辑电路实现如图 3-1-2所示,表示符号如图 3-1-3所示。

表 3-2 全加器真值表

X_i	Y_i	C_{i-1}	F_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

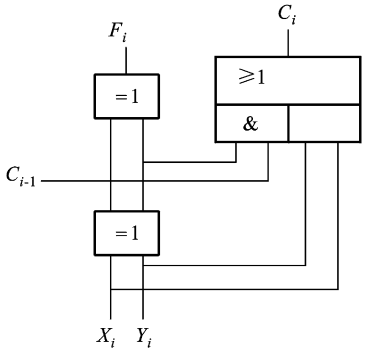


图 3-1-2 全加器逻辑电路

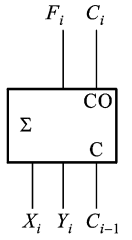


图 3-1-3 全加器表示符号

3.1.2 加法器

半加器和全加器只能进行 1位二进制的加法运算,全加器作为组成加法器的元件,可构成能进行 n位数运算的加法器。

根据运算方法不同,加法器可分为串行加法器和并行加法器。

1. 串行加法器

串行加法器只有一位全加器,每次实现一位二进制数的运算,n位数据需要通过移位的办法,一位一位地串行移入全加器,分时进行运算。

串行加法器的逻辑图如图 3－1－4所示。

图中，A、B分别为两个右移寄存器，存放参加运算的两个数。在同步脉冲 CP的控制下，A、B两个数从低位开始逐位移入全加器进行运算。本位产生的和 F_i 移入寄存器 A的高位，本位产生的进位 C_i 经触发器同下一个送来的数相加。这样经 n次移位计算后，便计算出两数的和，结果存放在寄存器 A中。

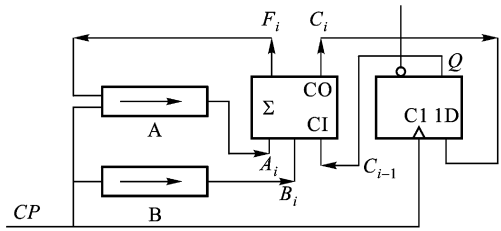


图 3－1－4　串行加法器逻辑结构框图

可见串行加法器，进行两个 n位数的运算，需要 n次运算才能完成，运算速度慢，且与数据字长有关。

2. 并行加法器

并行加法器由 n位全加器组成，数据的各位同时进行运算，其逻辑结构如图 3－1－5所示。

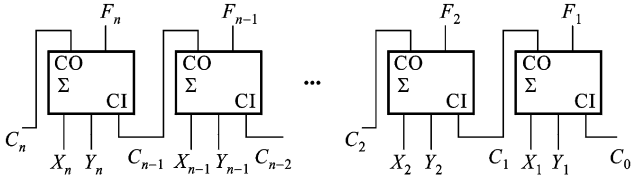


图 3－1－5　串行进位的并行加法器

图中各位运算同时完成，但每一位的运算都要加上低位的进位，即当低位的进位到来时，才能完成本位的运算。而低位的进位又和再低一位的进位有关，这样一级级的推下去，每一位的进位都与比该位低的所有位有关。时间最长的运算情况是最低位的进位从最低位一位一位地传到最高位。由此可见，进位是串行的。这样的加法器结构虽然简单，但由于串行进位使得运算速度降低，同样加法运算的时间与计算位数有关。只有改进逐位传送的进位方式，才能提高速度。

改进串行进位的办法之一，是采用“超前进位产生电路”，来同时形成各位进位，即使串行进位变成并行进位，从而实现快速加法运算，这样的加法器称为超前进位加法器。关于超前进位加法器本书不再详述，如要进一步了解具体内容，可查阅计算机组成原理方面的书籍。

3.2　定点加减运算的实现

如前一章所述，在计算机中带符号数的表示形式有原码、反码、补码等。各种编码都可以实现加法运算。但一般机器不直接设计原码加法器，因为原码表示的符号位不能直接参加运算，必须单独处理。在进行加减运算时，要首先判断参加运算的两个数的符号，来决定加或减运算，最后还要根据结果的情况来决定符号。这样实现起来很复杂。而补码运算时，符号位直接参加运算，不用单独处理，更重要的是，减法运算可以转换成加法运算，这样只设计一个加法器就可实现加法和减法

运算。所以现代计算机一般都采用补码加法运算器。这里只介绍补码加减运算的实现。

通过第2章的学习可知：

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

由此可推出：$[X-Y]_{补}=[X+(-Y)]_{补}=[X]_{补}+[-Y]_{补}$

所以，已知 $[Y]_{补}$，可用变补的方法求出 $[-Y]_{补}$，即连同符号一起按位求反再在最低位加1。这样采用补码方案时，可用加法器实现减法运算。

例 3.1 $X=+0.1011, Y=-0.1010$，求：$X+Y$。

解：因为 $[X]_{补}=00.1011$，$[Y]_{补}=11.0110$

$$\begin{array}{rl} & [X]_{补}\quad 00.1011 \\ + & [Y]_{补}\quad 11.0110 \\ \hline & [X+Y]_{补}\quad 00.0001 \end{array}$$

符号位最高位前自动丢掉一个1。

所以 $X+Y=+0.0001$。

例 3.2 $X=-0.1010, Y=-0.0101$，求：$X+Y$。

解：因为 $[X]_{补}=11.0110$，$[Y]_{补}=11.1011$

$$\begin{array}{rl} & [X]_{补}\quad 11.0110 \\ + & [Y]_{补}\quad 11.1011 \\ \hline & [X+Y]_{补}\quad 11.0001 \end{array}$$

符号位最高位前自动丢掉一个1。

所以 $X+Y=-0.1111$。

例 3.3 $X=+0.1100, Y=+0.0111$，求：$X-Y$。

解：因为 $[X]_{补}=00.1100$，$[Y]_{补}=00.0111$

则 $[-Y]_{补}=11.1001$

$$\begin{array}{rl} & [X]_{补}\quad 00.1100 \\ + & [-Y]_{补}\quad 11.1001 \\ \hline & [X-Y]_{补}\quad 00.0101 \end{array}$$

符号位最高位前自动丢掉一个1。

所以 $X-Y=+0.0101$。

例 3.4 $X=-0.1100, Y=-0.0110$，求：$X-Y$。

解：因 $[X]_{补}=11.0100$，

$[Y]_{补}=11.1010$

则 $[-Y]_{补}=00.0110$

$$\begin{array}{rl} & [X]_{补}\quad 11.0100 \\ + & [-Y]_{补}\quad 00.0110 \\ \hline & [X-Y]_{补}\quad 11.1010 \end{array}$$

所以 $X-Y=-0.0110$。

如图3-2-1所示为补码加法器逻辑电路。

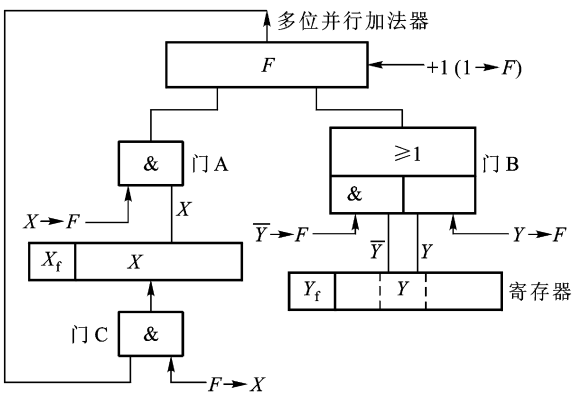


图 3-2-1 实现补码加减运算的逻辑电路

图3-2-1为实现 $[X\pm Y]_{补}\rightarrow X$ 的补码加减运算的逻辑电路。图中的F代表多位并行加法器，X、Y代表两个寄存器，临时

保存参加运算的数据 X、Y的补码，X还用来保存计算结果。“与”门 A控制把寄存器 X的输出内容送到加法器 F的左输入端，由控制信号 X→F完成。“与”门 C控制把加法器 F的运算结果送寄存器 X中，由控制信号 F→X完成。“与或”门 B控制把寄存器 Y的输出内容“原样”或“取反”送到加法器 F的右输入端，由控制信号 Y→F和 非→F完成。加法器的最低位还可接收加 1信号 1→F。加法器和寄存器由多少位组成，即为定点运算器的字长，一般可为 16位、32位、64位等。门 A、B、C每位有一个，图中只画出一位。

下面介绍运算操作过程。

当参加运算的数之补码 $[X]_{补}$和 $[Y]_{补}$被送到寄存器 X、Y中，若要实现 $[X+Y]_{补}\rightarrow X$操作，则给出 X→F，Y→F两个命令，将 $[X]_{补}$和 $[Y]_{补}$送入加法器的两个输入端，F则完成 $[X]_{补}+[Y]_{补}$的加法过程，通过 F→X命令把运算结果写入 X寄存器，整个加法过程结束。

若要实现 $[X-Y]_{补}\rightarrow X$操作，则给出 X→F命令，将 $[X]_{补}$送入加法器的左输入端，给出非→F命令，将 $[Y]_{补}$按位求反送入加法器的右输入端，并给出 1→F命令，在加法器的最末尾加 1，完成将 $[-Y]_{补}$送加法器的操作，实现 $[X]_{补}+[-Y]_{补}$，结果同样用 F→X命令送入寄存器 X中。

可见，用同一套加法电路，可以完成 $[X+Y]_{补}$和 $[X-Y]_{补}$的运算。

图 3-2-1电路是逻辑图，实际电路中需进行多方面的完善。最主要的要进行如下几方面考虑。

① 需有判溢出的功能。实现上述运算的前提是运算结果不发生溢出，若发生溢出则结果是错误的。所以要找出判断溢出的条件。判断溢出的方法很多，常用的方法有：

两个符号相同的补码相加，结果符号相反，或异号相减结果符号与减数相同，则溢出。这样判断较复杂。

另一种方法是若最高数值位向符号位的进位值与符号位产生的进位不同则溢出，即 $OVR=C_{n-1}\oplus C_n$。

再有一种方法是采用双符号位（变形补码），当两个符号位的值不同时，即结果为 01或 10时则发生溢出，如前所述。

② 还应有多个寄存器及移位功能，以提高运算效率和方便程序设计人员的程序设计。

③ 运算器不仅要给出运算结果，还要记录本次运算结果的某些状态和特征，如结果是否为 0，结果是正还是负，是否产生溢出，等等，以便程序中判断使用。为此需设置标志寄存器，并给出相应的读写电路。

总结上述补码加减的运算过程可得补码加减运算的规则如下：

参加运算的操作数用补码表示。

符号位一起参加运算。

若加则直接相加，若减则减数变补后相加。

结果以补码表示。

对原码加减运算的实现，其电路较复杂，这里不再赘述。

3.3　定点乘法运算的实现

在计算机中实现乘除运算，一般有下面几种办法。

① 用软件的办法实现乘除法运算。这种方法中,运算器只要能实现加、变补、移位等基本功能就可以,不需要专门的硬件电路,但运算速度慢。

② 用专用硬件来实现,即设置专用乘除法器。这种方法速度快,但电路复杂。一般适用于大、中型机。

③ 在原有的运算器基础上,增加一些硬件设备来实现乘除法运算。适合中、小型机和微型机。

下面主要通过原码 1位乘运算来介绍乘法运算的实现方法。

用原码实现乘法运算十分方便。原码表示的两个数相乘,其乘积数值为两数绝对值之积,积的符号为两数符号的异或值,即符号单独处理。

设 $[X]_{原}=X_sX_1X_2\cdots X_n$

$[Y]_{原}=Y_sY_1Y_2\cdots Y_n$

则 $[X\times Y]_{原}=S_pS$

$S=(X_1X_2\cdots X_n)\ (Y_1Y_2\cdots Y_n)$

$S_p=X_s\bar{\ }Y_s$

式中,X_s、Y_s 分别是被乘数和乘数的符号。S_p 是乘积的符号,因同号相乘为正,异号相乘为负,即是异或关系。

首先看手工计算乘法的过程。

[例 3.5] $X=0.1101$, $Y=0.1011$, 则

$$\begin{array}{r} 0.1101 \\ \times\ 0.1011 \\ \hline 1101 \\ 1101 \\ 0000 \\ 1101 \\ \hline 0.10001111 \end{array}$$

即 $X\times Y=0.1000111$ 符号为正。

手工计算时,依乘数每一位的取值是 1还是 0,决定加被乘数的值还是加零值,并且是从乘数的最低位求起,逐渐向高位进行,每次相加逐个左移一位,最后一并求和。

在计算机中实现原码乘法,不能简单照搬上述的方法,主要存在下面几方面问题。

首先,由于运算器内很难实现多个数据同时相加,通常只能实现两个数求和操作。在计算机中,解决该问题的办法是,每求得一个相加数,就同时完成与上次部分积相加的操作,这样将多个数同时相加分解成两个数的相加。

其次,在手工计算中,各加数逐位左移,最终相加数的位数是相乘两数位数的两倍,而在计算机中,加法器的位数一般与寄存器的位数相同。解决办法是,用部分积右移来代替相加数左移,因为在求每次部分积之和时,前一次部分积的最低位不再进行相加,若采用部分积右移,则可只用 n位加法器就能实现两个 n位数相乘。

最后,在计算机中判断乘数的每一位是 0还是 1,是采用判断存放乘数的寄存器的最低一位来实现的,只要每求一次部分积就使存放乘数的寄存器执行一次右移操作即可。若乘数寄存器每移一次位时,用空出的高位接收加法器最低位的输出,则完成乘法运算后,该寄存器中保存的

是低位积。

计算机内求乘积的符号，用相乘两数符号的半加和（或异或）值实现。

由上所述，可得到下面的计算机内实现原码相乘的具体方法。另外，在计算时用计数器控制运算次数。

【例 3.6】　$X=0.1101$，$Y=0.1011$，求 $X\times Y$。

解：运算过程如下：

	高位部分积 A	乘数/低位部分积 C		说明
	00000	$Y_0$1011		起始
+	01101			乘数最低位为 1，加被乘数 X
	01101			
→	00110	1$Y_0$101	1	右移部分积
+	01101			加 X
	10011			
→	01001	11$Y_0$10	1	右移
→	00100	111$Y_0$1	0	乘数末位为 0，不加 X，右移
+	01101			加 X
	10001			
→	01000	1111Y_0	1	右移

结果为：0.10001111（符号 =0⊕0=0为正。

开始　　A：0（部分积初值）　　B：被乘数　　C：乘数

结束　　A：积的高位　　B：被乘数　　C：积的低位

原码一位乘的逻辑结构如图 3－3－1所示。

乘法开始时，A寄存器被清 0，作为初始部分积。被乘数放在 B寄存器中，乘数放在 C寄存器中。实现部分积与被乘数相加是通过给出 A→F命令和 B→F命令实现的。实现部分积右移，是在 F/2→BUS控制下，把经过 F求得的结果右移一位送 BUS，并用 BUS→A命令将结果送 A寄存器实现的。

C寄存器是用移位寄存器实现的，在命令 C/2→C控制下自行进行移位操作。C的最低位可用作 B→F的控制命令，其值为 1时，发 B→F命令，送被乘数，否则，不发 B→F命令，直接移位。

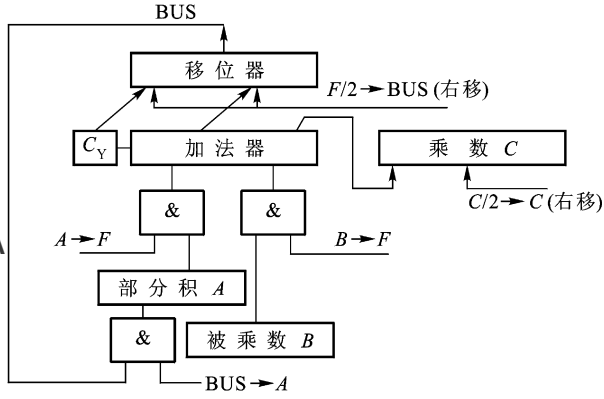


图 3－3－1　实现原码一位乘的逻辑结构

加法器最低位移位时，移入 C寄存器的最高位，使低位部分积存入 C寄存器中。原来的乘数在逐位移位过程中丢失。

另设一个计数器，对计算的次数进行计数，当计数次数与参加运算的数据位数相同时，控制停止运算。

乘法运算还有补码一位乘、两位乘、多位乘实现方案，这里不再详述。

3.4 定点除法运算的实现

定点除法的实现方法有很多，有一位除、两位除、多位除。一位除又有原码一位除和补码一位除等实现办法。

用原码实现定点除法运算是比较方便的，在原码除法中，商的符号为相除二数符号的异或值，数值为两数的绝对值之商。

定点原码除法常用的实现方法有：恢复余数法和不恢复余数法（加减交替法）。下面只通过原码一位除恢复余数法介绍除法的实现方法。

由于定点小数的绝对值小于 1，如果被除数大于等于除数，则商就大于或等于 1，因而会产生溢出，这是不允许的。因此，在执行除法之前，先要判断是否有溢出，无溢出时才执行除法运算，否则不执行。

先通过手工除法计算过程，来分析原码除法的实现方法。

【例 3.7】 $X=0.1011$，$Y=0.1101$，求 X/Y。

$$
\begin{array}{r}
0.1101 \\
0.1101\overline{)\,0.10110} \\
\underline{\;\;1101\;\;} \\
10010 \\
\underline{\;\;1101\;\;} \\
010100 \\
\underline{\;\;1101\;\;} \\
0111
\end{array}
$$

所以 $X/Y=0.1101$，余数为 0.0111×2^{-4}，商的符号为 0。

手工计算除法的规则是，判断被除数 X与除数 Y的大小，若 X小则商为 0，并把被除数的下一位移下来或补 0，再与右移的 Y比较，够除则商为 1，用部分余数减除数，否则商为 0。重复上述步骤，直到除尽或满足要求为止。

上述计算中，要求加法器的位数必须为除数位数的两倍，实现起来不合理。计算机中，用被除数（余数）左移代替除数右移，这样左移出界的被除数（余数）的高位都是无用的零，不再参加下面的运算，对运算不会产生任何影响。

另外，在计算机中，判断是否够除，用先做减法，再判断结果正负的方法来判断。若为负则不够减，商为 0并把除数再加到差上去，恢复余数为原来的正值之后，再将其左移一位进行下一步运算（这就是恢复余数法的由来）。若为正就不进行恢复余数的操作，商为 1，余数左移一位，进

行下一步操作。

再者,上商是通过把求得的每一位商,放到放商值的寄存器的最低位,并把原来的部分商左移一位实现的。在除法开始时,商寄存器还可以存放双倍字长的被除数的低位部分,在实现部分余数左移时,将它的高位数值位移入部分余数的最低位。

下面是恢复余数除法的一个例子。

例 3.8　设 $X=+0.10110$, $Y=+0.11111$,求 X/Y。

解:在运算中,减 Y用加 $[-Y]_{补}$ 来实现,所以应先求出 $[-Y]_{补}=11.00001$。操作步骤如下:

操 作 说 明	被除数/部分余数 A		商 C
	00.10110		00000
$-B$ (Y)	+) 11.00001		
$S_A=1$ $A<B$	11.10111		
恢复余数	+) 00.11111		
	00.10110	r_0	
←	01.01100	$2r_0$	0000_
$-B(Y)$	+) 11.00001		
$2A>B$ 商 1	00.01101	r_1	00001
←	00.11010	$2r_1$	0001_
$-B(Y)$	+) 11.00001		
$2r_1<B$ 商 0	11.11011		00010
恢复余数	+) 00.11111		
	00.11010	r_2	
←	01.10100	$2r_2$	0010_
$-B(Y)$	+) 11.00001		
$2r_2>B$ 商 1	00.10101	r_3	00101
←	01.01010	$2r_3$	0101_
$-B(Y)$	+) 11.00001		
$2r_3>B$ 商 1	00.01011	r_4	01011
←	00.10110	$2r_4$	1011_
$-B(Y)$	+) 11.00001		
$2r_4<B$ 商 0	11.10111		10110
恢复余数	+) 00.11111		
	00.10110		

结果:商为 0.10110　余数为:0.10110×2^{-5}

A:存放部分余数 (初值为被除数)。

B:存放除数。

C:存放商 (初值为 0)。

这种方法每恢复一次余数多一次 +Y运算,降低了运算速度,线路也复杂。计算机中常采用不恢复余数法。

图 3-4-1为原码一位除的逻辑电路框图。

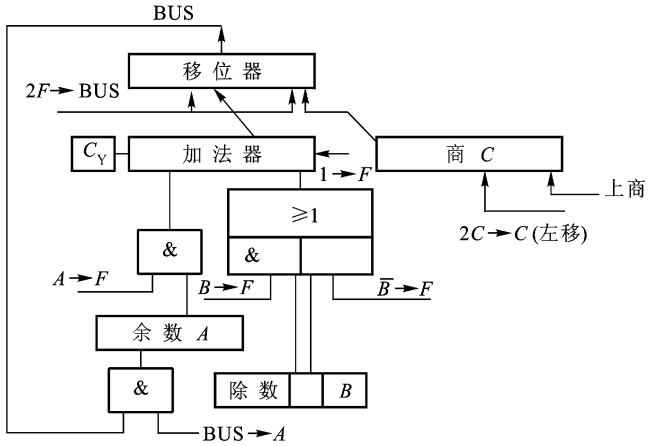


图 3-4-1 实现原码一位除的逻辑电路框图

A寄存器存放部分余数,初始值为被除数,终止值为余数。B寄存器存放除数,加 Y时直接传送,用加 $[-Y]_{补}$ 时反向传送,且末位加 1的方法实现。移位寄存器 C存放商,每次将商放到最低位,然后通过左移,实现下次上商。其他方面的操作类同于乘法运算。

关于补码一位除、多位除等,这里不作详细介绍。

这里所讲的一位乘或一位除,不是每次只进行数的一位运算,是指对 n位数来说要进行 n步运算,每次完成一步。对乘法,每次判断乘数的一位,求得一步部分积。对除法来说,每次只求得一位商。而两位乘或两位除,则每次完成两步操作。显然这样操作所用的步骤少,运算速度快,但增加了运算的复杂度,其硬件实现电路也复杂得多。

3.5 浮点运算

定点数的表示范围小、精度低。为解决这个问题,引入了浮点运算。浮点运算精度高,更适合于科学与工程计算。但浮点运算的处理较复杂,硬件代价高,运算速度也较慢。

因浮点数分阶码和尾数两个部分,阶码是定点整数形式,尾数是定点小数形式。所以在运算中,阶码和尾数是分别进行的。其实这两部分的运算仍然是定点运算规则,只不过增加一些规格化的步骤。下面介绍浮点数的四则运算方法。

3.5.1 浮点加减运算

设有两个浮点数 X和 Y,分别表示为:

$X=M_X\times 2^{E_X}$，$Y=M_Y\times 2^{E_Y}$，其中 E_X、E_Y 是阶码，M_X、M_Y 是尾数。

进行两浮点数的加减运算时，要经过下面步骤。

1. 对阶

在进行浮点加减运算时，首先要看两数的阶码是否相同，即小数点位置是否对齐。只有阶码相等时才能进行尾数的加减运算。使参加运算的两个数阶码相等的操作为对阶。

实现方法是，比较两个浮点数的阶码值的大小，即求阶差

$$\Delta E=E_X-E_Y$$

若 $\Delta E=0$，表示阶码相等，不需要对阶。

若 $\Delta E\neq 0$，表示阶码不等，需要对阶。

要通过尾数的移位来改变 E_X 或 E_Y，使它们相等。为使误差减小，一般总是使小阶向大阶对齐，即使小阶的尾数向右移位（相当于小数点左移），每右移一位，阶码加 1，直到两数的阶码相等为止。也就是阶码小的数尾数右移 ΔE 位，其阶码加 ΔE。

【例 3.9】 $X=2^{01}\times 0.1101$，$Y=2^{11}\times(-0.1010)$，求 $X+Y$。

设两数在计算机中以补码表示

$[X]_{补}=00\ 01,\ 00\ 1101$

$[Y]_{补}=00\ 11,\ 11\ 0110$

阶差 $\Delta E=E_X-E_Y=0001+1101=1110$

即 $\Delta E=-2$ 说明 E_X 比 E_Y 小，将 X尾数右移两位，阶码加 2。

则 $[X]_{补}=00\ 11,\ 00\ 0011$，使得 X、Y的阶码相等。

2. 尾数加（减）

两个完成对阶后的浮点数的求和（差）过程，使用定点数求和的实现方法即可。

【例 3.10】 对例 3.9尾数求和：00 0011+11 0110=11 1001，所以

$$[X+Y]_{补}=0011,\ 11\ 1001$$

3. 结果规格化

如果得到的结果不满足规格化规则，就必须把它变成规格化的数。这样可增加有效数字的位数，提高运算精度。

规格化数要求小数点后必须是有效数字。当数位用二进制表示时，规格化的定义是尾数应满足：$1/2\leq|M|<1$。

对正数而言，有 $M=0.1\times\times\cdots\times$，即小数点后第 1位必须是 1。

对负数而言，有 $M=1.0\times\times\cdots\times$，即小数点后第 1位必须是 0。

如果结果不是规格化数，需要尾数向左移位，以实现规格化过程，称为向左规格化。

【例 3.11】 如对例 3.10结果进行规格化处理：

因例 3.10结果为 $[X+Y]_{补}=0011,\ 11\ 1001$

由于是负数，小数点后应该是 0，尾数左移 1位，阶码减 1，即

$$[X+Y]_{补}=0010,\ 11\ 0010$$

4. 舍入

在执行对阶和右移操作时，会使尾数低位上的一位或几位的数值被移掉，使数值的精度受到影响，可以把移掉的几个高位值保存起来供舍入使用。

常用的舍入处理方法有两种：一种是“0舍 1入”法，即如果右移时被移掉数位的最高位为 0则舍掉，为 1则将尾数的末位加 1。另一种是“恒置 1法”，即只要有数位被移掉，就在尾数末位恒置 1。

5. 判断结果的正确性

即检查结果是否有溢出，对浮点数检查溢出是以阶码的溢出表现出来的，所以主要是检查阶码是否有溢出。

3.5.2 浮点乘除运算

1. 浮点乘法运算

浮点乘法运算，即两个操作数的尾数相乘得到积的尾数，阶码相加得到积的阶码。相乘和相加运算与定点运算相同。

若 $X=M_X\times 2^{E_X}$，$Y=M_Y\times 2^{E_Y}$

则 $X\times Y=(M_X\times M_Y)\times 2^{E_X+E_Y}$。

运算分下面几步进行：

① 判断操作数是否为 0，如果为 0，其结果直接为 0，不必再进行运算。

② 阶码相加，以求得结果的阶码。

③ 尾数相乘，以求得结果的尾数。

④ 结果尾数和阶码进行规格化处理，并判断有无溢出。

2. 浮点除法运算

浮点除法运算，即两个操作数的尾数相除得到商和余数，阶码相减得到商的阶码。相除和相减运算与定点运算相同。

若 $X=M_X\times 2^{E_X}$，$Y=M_Y\times 2^{E_Y}$

则 $X/Y=(M_X/M_Y)\times 2^{E_X-E_Y}$

运算分下面几步进行：

① 判断操作数是否为 0，如果被除数为 0，则结果为 0；如除数为 0，为非法除，另作处理。

② 阶码相减，以求得结果的阶码。

③ 尾数相除，求出商和余数。

④ 结果尾数和阶码进行规格化处理，并判断有无溢出。

3.6 定点运算器

运算器主要由算术逻辑运算单元 ALU、锁存器、寄存器、内部总线和控制电路等构成，其核心部件是 ALU。下面介绍运算器的基本结构和组成。

3.6.1 运算器的基本结构

现代计算机的运算器一般有 3种结构形式：单总线结构、双总线结构、三总线结构。

1. 单总线结构

单总线结构的运算器如图 3-6-1所示。

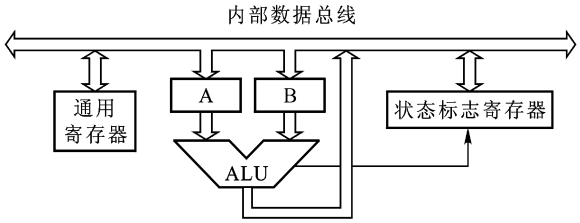


图 3-6-1 单总线结构运算器

在单总线结构的运算器中,所有部件都挂在同一总线上,各部件之间的数据传送都通过同一总线进行。由于同一时刻只能有一个操作数放在总线上,各部件对总线是分时使用的。这样,进行一次运算需 3步完成,第一步通过总线把第一个操作数送锁存器 A;第二步将第二个操作数送入锁存器 B;这时 ALU的两个输入端数据有效,才能进行运算;第三步将运算结果通过总线送入目的寄存器中。单总线结构的运算器结构简单,但速度较慢。

2. 双总线结构

双总线结构的运算器如图 3-6-2所示。

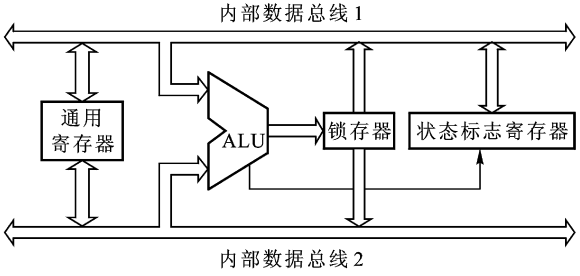


图 3-6-2 双总线结构运算器

在双总线结构的运算器中,用两组内部数据总线连接运算器的所有部件。这种结构中,两个操作数可同时加到 ALU的输入端,且可立刻得到运算结果。ALU的输出不能直接连到总线上,因为当形成操作结果时,两条总线都被输入数据占据着,所以必须在 ALU输出端设置锁存器。这样,进行一次运算需两步完成,第一步把两个操作数分别通过数据总线 1、数据总线 2送 ALU的输入端,并形成运算结果送锁存器;第二步把锁存器的运算结果通过任一总线送入目的寄存器中。显然双总线结构的运算器提高了运算速度。

3. 三总线结构

三总线结构的运算器如图 3-6-3所示。

在三单总线结构的运算器中,用三条总线连接运算器的所有部件。这种结构中,ALU的两个输入端,分别由内部数据总线 1、内部数据总线 2供给数据,而 ALU的输出与内部数据总

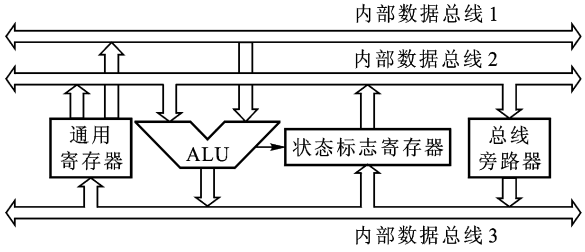


图 3-6-3 三总线结构运算器

线 3相连。这样,进行一次运算只需一步就可完成,把两个操作数分别通过内部数据总线 1、内部数据总线 2送 ALU的输入端,并形成运算结果直接通过内部数据总线 3送入目的寄存器中。

图 3-6-3中设置了总线旁路器,如果有的数据要在总线间直接传送,可通过总线旁路器将内部数据总线 2的数据直接传给内部数据总线 3。所以如数据要运算或改变,则通过 ALU传送,如不需要改变可通过旁路器直传。三总线结构的运算器运算速度很快,但内部线路复杂。

3.6.2 运算器的组成

下面以一个模型机的运算器为例,说明运算器的组成。如图 3-6-4所示。

该运算器由如下几部分构成:

⑴ 算术/逻辑运算单元 ALU

算术/逻辑运算单元 ALU可由 4片 SN74181和 SN74182构成。SN74181是 4位并行加法器,SN74182是并行进位部件。这样可构成 16位的 ALU。

⑵ 锁存器

锁存器 A、B用来暂存来自通用寄存器、存储器或外部设备的数据。给 ALU提供参加运算的数据。一旦数据进入锁存器,不管外部数据怎样变化,都不能改变锁存器的内容。ALU将依据锁存器 A、B的数据进行处理。

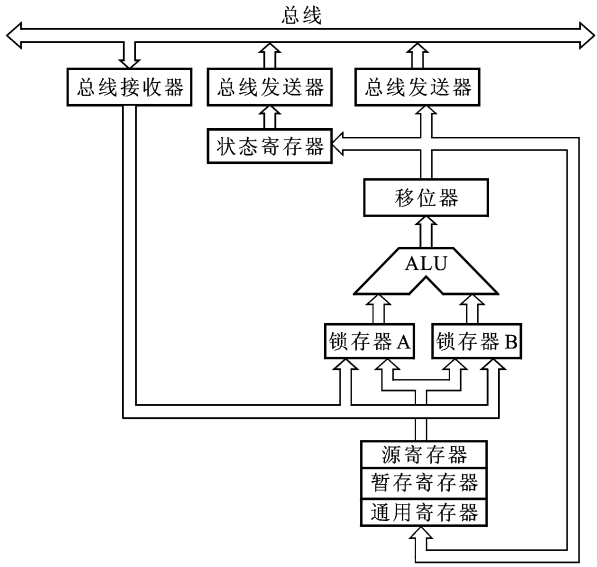


图 3-6-4 模型机运算器组成框图

⑶ 通用寄存器组

该运算器中,设置了 8个通用寄存器 R_0 ~ R_7。通用寄存器可供程序员访问,用来作为累加器、变址寄存器、操作数寄存器等来使用。源寄存器用来存放源操作数,暂存寄存器用来暂存中间结果。

⑷ 移位器

移位器用来将 ALU的数据进行左移、右移、直传、半字交换等操作。供不同运算要求使用。

⑸ 状态寄存器

状态寄存器用来存放在运算过程中得到的状态标志（如零标志、进位标志、符号标志等），以便供程序判断使用。

3.7 控制器的功能和基本组成

3.7.1 控制器的功能

计算机对信息的处理是通过程序的执行而实现的，程序是完成某个确定算法的指令序列，要预先存放在存储器中。控制器的作用是控制程序的执行，它应具有下列功能。

⑴ 取指令

根据程序入口地址，从存储器中取出一条指令，并指出下条指令的地址。取出的指令送到指令寄存器，以便分析运行该指令。

⑵ 分析指令

分析指令又叫解释指令或指令译码。是对当前取得的指令进行分析，指出它要求完成什么操作，并产生相应的操作控制命令。如果参与操作的数据在存储器中，还要形成操作数地址。

⑶ 执行指令

根据分析指令产生的控制命令和操作数地址，形成相应的操作控制信号序列，通过运算器、存储器、输入/输出设备的执行，实现每条指令的功能。

计算机不断重复上述 3种操作：取指、分析、执行；再取指、再分析、再执行……如此循环，直到遇到停机指令或外来干预为止。

⑷ 控制程序和数据的输入与结果的输出

根据程序的安排并通过人的干预，在适当的时候向输入/输出设备发出一些相应的命令来完成输入/输出功能，这实际上也是通过执行程序来完成的。

⑸ 对异常情况和某些请求的处理

当机器出现某些异常情况时（如溢出、校验错）或某些外来请求（如中断、DMA等）时，要进行相应的处理。

3.7.2 控制器的组成

控制器的基本组成如图 3－7－1所示。

1. 程序计数器（Program Counter, PC）

程序计数器又称为指令计数器（Instruction Pointer, IP），是用来存放下条指令地址的。当取出指令后，应确定下条指令的地址，这样才能保证程序的连续执行。在程序开始执行时，必须将程序的入口地址（第一条指令地址）送入 PC。程序运行中，CPU自动修改 PC的值。PC一直指向下次要取的指令。

一般有两种途径来形成指令地址：一是顺序执行时，通过程序计数器自动增量形成下条指令

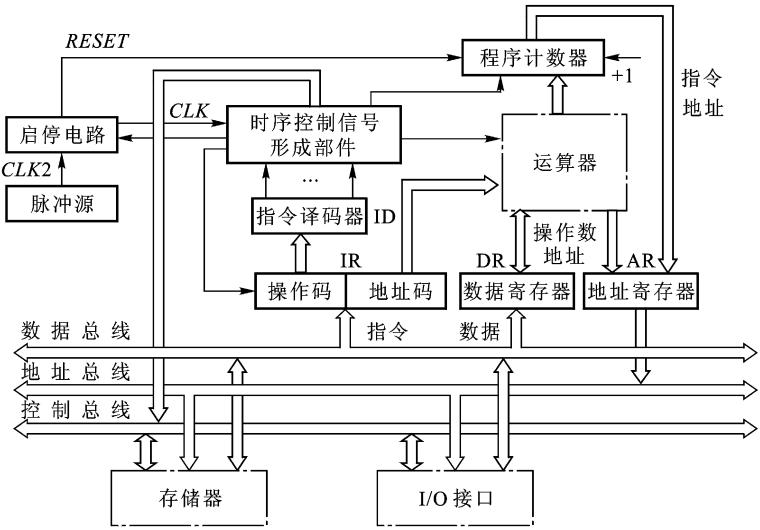


图 3-7-1 控制器基本组成框图

地址,即每取一条指令,PC自动加 1;二是遇到要改变程序执行顺序时,一般由转移指令形成转移地址送程序计数器,形成下条指令地址。

PC在某些计算机中用来存放当前正在执行的指令地址,而在另一些计算机中用来存放下条指令地址,在有预取指令功能的计算机中用来存放下条要取出的指令地址。

2. 指令寄存器 (Instruction Register, IR)

指令寄存器用来存放当前执行的指令。要执行一条指令时,先将指令从存储器中取出,取出的指令首先存放在指令寄存器当中,以便下一步送指令译码器译码执行。

3. 指令译码器 (Instruction Decoder, ID)

一条指令一般包括操作码部分和操作数地址部分。操作码指示指令要完成什么操作。指令译码器的作用就是对指令寄存器的操作码部分进行译码,产生相应的控制信号。

4. 时序控制信号形成部件

时序控制信号形成部件,对译码器送来的控制信号,按一定的时序关系产生控制信号序列,控制 CPU内部各部件完成指令功能。同时也发出各种外部控制信号。

时序控制信号的形成,是在时钟脉冲的作用下进行的。时序控制信号还要根据被控制部件的反馈信号进行调整。

5. 脉冲源和启停线路

脉冲源主要是指时钟发生器,它产生一定频率的脉冲信号,作为整个机器的时钟脉冲,该脉冲信号是机器的工作脉冲的基准信号。在机器加电时,应产生一总清 (复位)信号 (RESET)。启停电路保证送出或封锁时钟脉冲,控制时序信号的发生或停止,从而启动机器或使之停机。

图 3-7-1给出的控制器是最基本的组成框图,假设操作数地址以及转移地址的计算在运算器中进行。事实上有不少计算机专门设有地址加法器,并假设运算器和控制器之间有内部总

线连接,而运算器、控制器与存储器、输入/输出设备之间均通过外部总线 DB、AB、CD相连。

3.7.3 指令的执行过程

下面以图 3-7-1为例,说明指令 MOV [2000],AL的执行过程。这是一条 8086指令,但执行过程以模型机为例,主要是通过此过程进一步说明计算机的工作原理。

指令的功能是将寄存器 AL中的内容送入地址为 2000H的内存单元中。该指令是一个三字节指令,假设存放在 1200H开始的内存单元中,如图 3-7-2所示。

该指令的执行需要 3步来完成,取指令操作码并译码、取操作数地址、存储操作数。下面分别进行介绍。

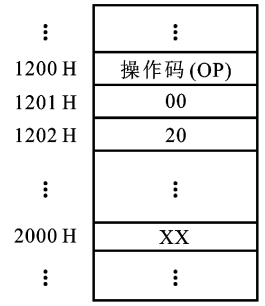


图 3-7-2 指令的内存存放示意图

1. 取指令操作码

上条指令形成的指令地址 1200H已存放在指令计数器 PC中,取本条指令的过程如下:

① 将程序计数器 (PC)的内容 (1200H)送存储器的地址寄存器 (AR)。

② 程序计数器 (PC)内容加 1,变成 1201H,为取下面内容作准备。

③ AR内容送地址总线,发读信号给存储器,读出指令操作码送指令寄存器 IR。

④ 译码器对取出指令的操作码进行译码,识别出是直接寻址的传送指令,发出完成下一步操作的控制信号。

2. 取操作数地址

因 1201H和 1202H单元存放的是操作数的地址 (低位在前,高位在后),首先将其取出,再将寄存器的内容存入该地址所指的内存单元。这里假设一次从存储器中取出一个字 (两个字节),操作过程如下:

① 将 PC的地址送 AR。

② PC内容增量,指向下条指令。

③ 将 AR的地址送上地址线,发读一个字的读命令,读出数据 (2000H)送数据寄存器 DR。

④ 将 DR的内容送 AR。

3. 存储操作数

将寄存器 AL的内容存入上一步取出的地址 (2000H)单元。操作过程如下:

① 将寄存器 AL的内容送入 DR。

② 将 AR的地址 (2000H)送到地址线上,指向要写入的单元。

③ 将 DR的内容送到数据线上。

④ 发出存储器写信号,将数据总线上的数据写入所选存储单元。

到此整个指令执行完毕,继续执行下条指令,直到程序结束。

3.7.4 控制器的控制方式

由上述指令的执行可知,一条指令的功能是通过按一定次序执行一系列基本操作来完成的,

这些基本操作称为微操作。指令的微操作序列都是严格按照一定的时间顺序进行的，即在时钟脉冲的作用下进行。微操作序列每进行一步叫一个节拍，控制器的控制方式就是控制器控制微操作序列的执行方式。常用的控制方式有同步控制方式、异步控制方式和联合控制方式。

1. 同步控制方式

同步控制方式是指每条指令在执行时所需的节拍数是确定的、固定不变的，由统一的同步脉冲控制微操作的执行。

在同步方式下，为保证最长时间的微操作能正确完成，需要以微操作序列最长的指令为标准，确定控制微操作运行的节拍数。这种方式节拍脉冲和节拍电位是统一的、固定的，所以对微操作序列短的指令，将造成时间的浪费，使运行速度变慢。但此方式控制简单、易于实现。

2. 异步控制方式

在异步方式下，没有统一的时间标准，每条指令需要多少节拍，就产生多少节拍。所以微操作的进行采用应答方式工作。当控制器发出进行某一微操作的控制信号后，等待执行部件完成该操作后发回“回答”信号或“结束”信号，再开始新的微操作。该控制方式运行速度快，但控制电路较复杂。

3. 联合控制方式

联合控制方式是同步控制方式和异步控制方式相结合的方式。对不同指令的各个微操作实行大部分统一、小部分区别对待的方式。即大部分指令采用同步控制方式，称为中央控制；小部分特殊指令或微操作序列过长或过短的指令，采用异步控制方式执行，称之为局部控制。

3.8 微程序控制器

前面已经介绍过，控制器分为组合逻辑控制器和微程序控制器。两者的主要差别是“控制信号形成部件”不同，它反映了不同的设计方法和设计原理，而控制器的其他部分不因控制方式而异。

微程序设计思想是英国剑桥大学的威尔克斯（M. V. Wilkes）教授于 1951年提出来的。本节就来介绍微程序控制器的基本工作原理和设计方法。

3.8.1 微程序控制器的基本概念

1. 微程序的基本概念和术语

① 微命令：是构成控制信号序列的最小单位，即能直接作用于某部件控制门的命令。如控制门的选通信号、触发器的打入脉冲、置位脉冲等。

② 微操作：由微命令控制实现的最基本的操作。如取指操作中的基本操作，即 (PC)→AR，(PC)+1→PC，DB→IR 等，就是微操作。

③ 微指令：由同时发出的控制信号所执行的一组微操作命令构成微指令。微指令是把同时发出的控制信号的有关信息汇集起来而形成的。将一条指令分成若干条微指令，按次序执行这些微指令就可以实现指令的功能。

④ 微程序：微指令序列的集合叫做微程序。计算机中每条指令的功能均由微程序完成的。

⑤ 控制存储器：存放微程序的存储器。由于该存储器主要存放控制命令与下条执行的微指

令地址,所以被叫做控制存储器。

2. 微程序设计的基本思想

将每一条机器指令的功能,用一段微程序来实现。每段微程序由若干微指令组成,每个微指令含有若干微命令,每个微命令完成一个微操作。所以执行相应的微程序就完成了对应机器指令的功能。

一般计算机的指令系统是固定的,所以实现指令的微程序也是固定的,于是控制存储器可以用只读存储器实现。又由于机器内控制信号数量比较多,再加上决定下条微指令的地址码有一定的宽度,所以控制存储器的字长比机器字要长得多,一般为一百多位。

3. 机器指令与微指令,程序与微程序,主存储器与控制存储器的关系

机器指令用于完成机器的一项基本功能,是提供给用户编程序的基本单位。微指令是用于实现机器指令某步操作的一系列微命令的组合,它作为机器的内部信息,对用户来说是看不到的。

程序是程序员为完成某项任务而编制的,由若干语句 (每个语句对应若干机器指令)组成的指令集合。微程序由若干微指令构成,用于实现机器指令,它是在设计计算机时就事先设计好的。

主存储器是存放程序和数据的,是构成计算机的一大部件。控制存储器是存放微程序的,是构成控制器的主要组成部分。

3.8.2　微程序控制器的组成及工作原理

1. 微程序控制器的组成

微程序控制器的组成框图如图 3-8-1所示。

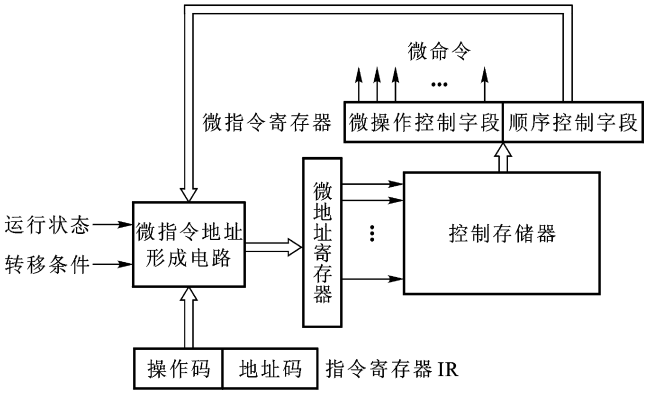


图 3-8-1　微程序控制器组成框图

微程序控制器由微指令地址形成电路、微地址寄存器、控制存储器、微指令寄存器及状态条件等部分组成。

微指令地址形成电路用于形成微程序入口地址和下条微指令地址。机器指令操作码部分送入微指令地址形成电路,形成该条指令的微程序入口地址,从此地址开始执行相应的微程序段。

在微程序运行过程中,该电路形成下条微指令地址。

微地址寄存器用来存放微指令地址。由微地址形成电路形成的微指令地址,寄存在微地址寄存器中。

控制存储器存放所有机器指令所对应的微程序。控制存储器包括微指令地址译码及驱动电路。来自微地址寄存器的微地址经译码驱动,选中相应的微指令单元,读出所选微指令,送微指令寄存器。

微指令寄存器存放从控制存储器中读出来的微指令。微指令包括两大部分:微操作控制字段和顺序控制字段(下址字段),微操作控制字段提供微命令,顺序控制字段指示下条微指令地址的形成方式或直接提供下条指令地址。

2. 微程序控制器的工作原理

微程序控制是将每一条机器指令的执行,分割成若干微操作序列并对应编制一小段微程序。执行指令的过程就是执行对应微程序的过程。运行过程有如下几个步骤:

① 从控制存储器中取出一条"取机器指令"用的微指令,送微指令寄存器。这是一条公用的微指令,因所有机器指令的取指操作都是一样的,对应的微指令也是一样的。该微指令一般存放在控制存储器的0号或1号单元。执行该微指令,从主存储器中读出一条机器指令,送指令寄存器IR。

② 机器指令操作码经微地址形成电路形成该指令对应的微程序入口地址,送微地址寄存器。

③ 从控制存储器中逐条取出微指令,送微指令寄存器。每条微指令提供一个微命令序列,控制有关操作。每执行完一条微指令,再由地址字段指出下条微指令的地址,再继续取出下条微指令执行。

④ 执行完对应一条机器指令的一段微程序后,返回0号(1号)微地址单元,读取"取机器指令"的微指令,以便取下条机器指令继续执行。

3.9 微程序设计技术

前面已经介绍了微程序控制器的基本原理,但如何确定微指令的结构,是微程序设计的关键。在实际进行微程序设计时要考虑如下因素:

① 如何缩短微指令字长。

② 如何减少控制存储器容量。

③ 如何提高微程序的执行速度。

这要从微指令的编码方法,微指令地址的形成方法,微指令的格式等多方面进行考虑。

3.9.1 微指令的编码方法

微指令的编码,就是解决对微指令的微操作控制字段如何进行表示的问题。下面主要介绍常用的几种表示方法。

1. 直接控制法

在微指令的微操作控制字段中,每一位代表一个微命令。在设计微指令时,是否发出某个微

命令,只要将控制字段中相应位置 1或置 0即可,这样就可打开或关闭某个控制门,该方法就是直接控制法。

该方法在某些复杂计算机中,当微命令太多时,则微指令太长,且每条微指令只要求一小部分微命令,即少数控制位有效,所以控制存储器的利用率太低,可用下面介绍的方法进行改进。

2. 字段直接编译法

字段直接编译法是将微命令中互斥的微命令编成一组,成为微指令的一个字段,通过字段译码器对字段进行译码,译码输出作为微操作控制信号。这种方法可大幅度缩短微指令长度。

在一个微周期 (一条微指令的执行时间)时间内,只有一部分微命令起作用。如果有若干微命令,在每次选择使用它们的微周期内,只能有一个微命令起作用,那么这若干微命令是互斥的,即不同时有效的微命令。例如,向存储器发的读命令和写命令是互斥的,ALU同一输入端的不同输入源控制信号是互斥的,等等。

如 7个互斥的微命令编成一组,可用 3位二进制编码表示每个微命令。3位编码有 8个状态,可表示 7个互斥的微命令,另外全“0”表示不发命令。同理 4位二进制数可表示 15个微命令。如图 3-9-1所示。

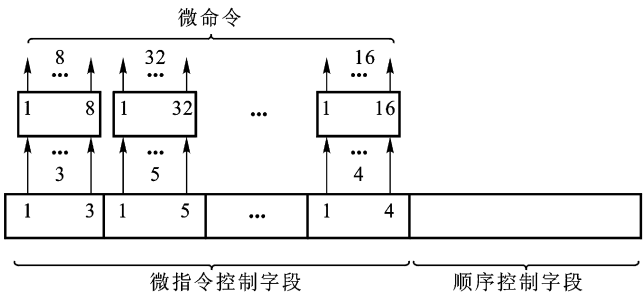


图 3-9-1 字段直接编译法

该方法由于增加了译码电路,使微指令的执行速度有所降低。

3. 字段间接编译法

在字段直接编译法中,规定一个字段的某些微命令,要兼由另一字段中的某些微命令来解释的方法称为间接编译法,相当于两级译码。

该法可进一步减少微指令长度,但削弱了微指令并行控制能力,一般只作直接编译法的辅助方法。

3.9.2 微指令地址的形成

怎样形成后继微指令的微地址,这实际上是微程序流的控制问题。

初始微指令地址,即为机器指令的微程序入口,可由机器指令的操作码译码形成。找到微程序入口后,就可以开始执行微程序,每条微指令执行完毕后,要由微指令的顺序控制字段控制形成后继微指令地址。

后继微指令地址的形成方法对微程序编制的灵活性影响很大。另外,在微程序中,也有微循环和微子程序,这将影响后继微地址的形成。下面介绍常用后继微地址形成方法。

1. 以增量方式产生后继微地址

以增量方式产生后继微地址与一般程序控制类似,顺序执行微指令时,由现行微地址加 1形成后继微地址,非顺序执行时,产生转移微地址。该方式中使用了微程序计数器 (MPC),所以又称为 "计数器 "方式,如图 3-9-2所示。

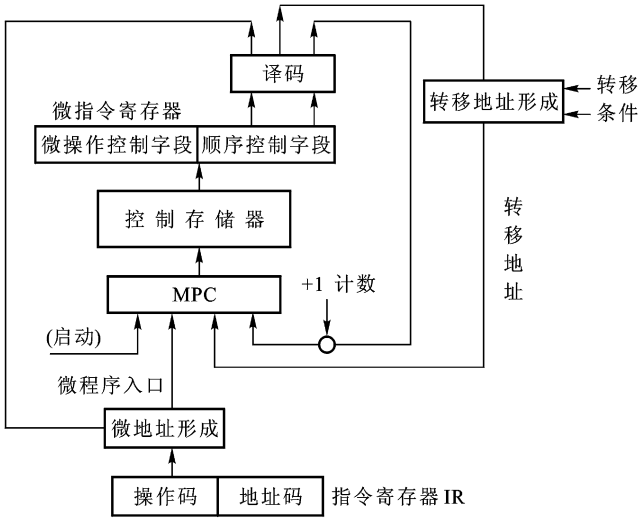


图 3-9-2 以增量方式进行微程序控制的原理图

机器加电后执行的第一条微指令地址即启动入口,来自专门的硬件电路,控制实现取指操作,然后由机器指令操作码产生相应微程序入口地址。

在微程序执行过程中,若顺序执行微指令,则将现行微指令地址加 1 (MPC +1) 产生后继地址,即微程序计数器加 1;若遇到转移类微指令,则由微程序计数器 (MPC) 与形成转移地址的逻辑电路组合形成后继微地址。

这种方式可使微指令的顺序控制字段很短,是一种常用的产生后继微地址的方法。

2. 增量与顺序控制字段结合产生后继微地址

顺序控制字段分两个部分 转移控制字段 BCF,转移地址字段 BAF。当转移时 BAF→MPC,否则 MPC +1→MPC。该方式不再详述。另外还有多路转移方式、微中断方式等。

3.9.3 微指令格式

微指令编码方法是决定微指令格式的主要因素,一般微指令分水平型和垂直型两种格式。

1. 水平型微指令

即在一条微指令中定义并执行多个并行操作的微命令。其编码基础是直接控制方式,通常将采用字段编译法的微指令也归为水平型微指令。

2. 垂直型微指令

微指令中设置微操作码,规定微指令的功能,采用微操作码编译法,即得到垂直型微指令,通常一条垂直型微指令只要求能控制一、二种操作。这种垂直型微指令采用了完全编码的方式。

3. 两种格式微指令的比较

水平型:并行性强,效率高,灵活,执行时间短;但微指令字长,控制存储器利用率低。

垂直型:并行性差,效率低,执行时间长;但指令字短,比较易掌握。

3.9.4 微程序控制存储器及操作

1. 微程序控制存储器

微程序控制存储器一般用只读存储器 ROM构成,因一般机器的指令系统是固定的,实现各机器指令的微程序也是固定的,所以在使用中只需读出,采用只读存储器即可。

也可用随机存储器 RAM构成,使用 RAM时,每次开机时需装入微程序。其优点是可修改和扩充指令系统,但降低了可靠性。

2. 控制存储器的操作

执行微指令的过程基本上分为两步,第一步将微指令从控制存储器中取出,称为取微指令。第二步执行微指令所规定的各个操作。根据这两步执行的控制方式可分为串行控制方式和并行控制方式。

⑴ 串行方式

执行一条微指令所需的时间称为微周期。在串行方式下微周期安排如图 3-9-3所示。

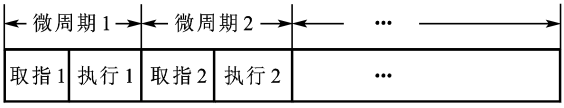


图 3-9-3 串行方式微周期

微周期包括取微指令和执行微指令两部分。可见这种方式实现控制简单,但微程序的执行速度较慢。一条指令的功能需要执行若干条微指令才能实现,所以控制存储器的取微指令时间对机器的速度影响很大。一般控制存储器的速度要比主存储器快得多。

⑵ 并行方式

并行方式中,将执行本条微指令的功能与取下一条微指令的操作在时间上重叠起来,如图 3-9-4所示。

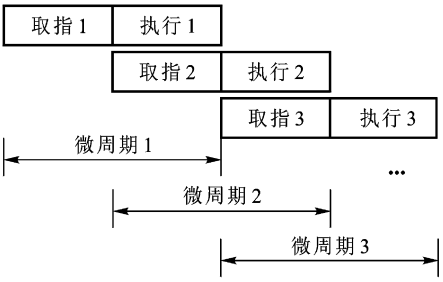


图 3-9-4 并行方式微周期

并行方式可以提高机器的速度，但由于执行本条微指令与取下条微指令是同时进行的，因此对于某些需要根据处理结果进行条件转移的微指令就不能及时取到下条微指令，此时需延迟一个微周期再取微指令。

习　题

1. 用补码运算计算下列各组数的和，并判断结果有无溢出。

⑴ X＝0.11001, Y＝ －0.10111

⑵ X＝0.10010, Y＝0.11000

⑶ X＝ －0.10010, Y＝ －0.01010

2. 用补码运算计算下列各组数的差，并判断结果有无溢出。

⑴ X＝ －0.01111, Y＝0.00101

⑵ X＝0.11011, Y＝ －0.10010

⑶ X＝ －0.01011, Y＝ －0.10110

3. 根据图 3－2－1补码加减电路的实现逻辑，说明实现补码减法的过程。

4. 用原码一位乘法计算积 X×Y（写出计算过程）。

⑴ X＝0.1101, Y＝ －0.1011

⑵ X＝ －0.1001, Y＝ －0.1010

5. 用原码一位除恢复余数法计算 X／Y，求出商及余数（写出计算过程）。

⑴ X＝ －0.10110, Y＝0.11111

⑵ X＝ －0.01110, Y＝ －0.10101

6. 运算器由哪些部件组成，核心部件是什么？

7. 控制器的基本功能是什么？简述控制器的基本组成。

8. 简单说明指令的执行过程。

9. 程序计数器 PC的作用是什么？

10. 控制器有哪几种组成方式？

11. 名词解释

⑴ 微操作　⑵ 微命令　⑶ 微指令　⑷ 微程序

12. 简述控制存储器的作用，说明与主存储器的区别。

13. 说明机器指令与微指令，程序与微程序的区别。

14. 简述微程序控制器的工作原理。

15. 微指令的编码有哪几种常用的方法，它们各有什么特点？

16. 微程序控制计算机中，下条微指令地址有哪些来源，各发生在什么场合？

第 4章　Intel 80x86微处理器

前面已经介绍了运算器和控制器,这两大部件一起构成了中央处理单元（CPU)。对微型计算机来讲,将运算器部件和控制器部件集成在一个集成电路芯片上,又称为微处理器。本章以Intel 8086为主,介绍典型的 Intel 80x86系列微处理器。

4.1　中央处理器的功能和组成

4.1.1　中央处理器的功能

计算机对信息进行处理是通过程序的执行而实现的。CPU要控制整个程序的执行,应具有以下基本功能：

1. 程序控制

程序的顺序控制称为程序控制。由于程序是一个指令序列,这些指令的前后顺序关系不能任意颠倒,必须严格按程序规定的顺序进行。因此保证计算机按一定的顺序执行程序是 CPU的首要任务。

2. 操作控制

如前所述,一条指令的功能是由若干操作信号的组合来实现的。执行指令的过程就是完成相应的微操作序列。CPU管理并产生每条指令的操作信号,把操作信号送到相应的部件,从而控制这些部件按指令的要求完成相应的任务。

3. 时间控制

对各种操作实施时间上的控制称为时间控制。在计算机中,各种操作信号均受到时间的严格控制,每条指令的整个执行过程也受到时间的严格控制。只有这样,计算机才能有条不紊地自动工作。实施时间控制也是 CPU的一项任务。

4. 数据加工

数据加工就是对数据进行各种运算和处理。数据的加工处理是 CPU的根本任务。

4.1.2　中央处理器的组成

CPU的组成包括两大部分 :运算器和控制器。在前面的章节中已经对运算器和控制器的结构及工作原理做了介绍。CPU的组成逻辑框图如图 4-1-1所示。

运算器由算术逻辑单元 ALU、累加器 AC、数据寄存器 DR、状态寄存器等组成。运算器在控制器的控制下进行数据的加工处理,它是一个执行部件。

控制器由程序计数器 PC、指令寄存器 IR、指令译码器 ID、地址寄存器 AR、时序产生电路和操作控制电路构成。

一般 CPU都配有一组通用寄存器,数量的多少与指令系统的设计有关。通用寄存器是在数据处理过程中可以指定不同用途的一组寄存器。为了高速处理数据,CPU用这组寄存器来临时存放地址和数据。

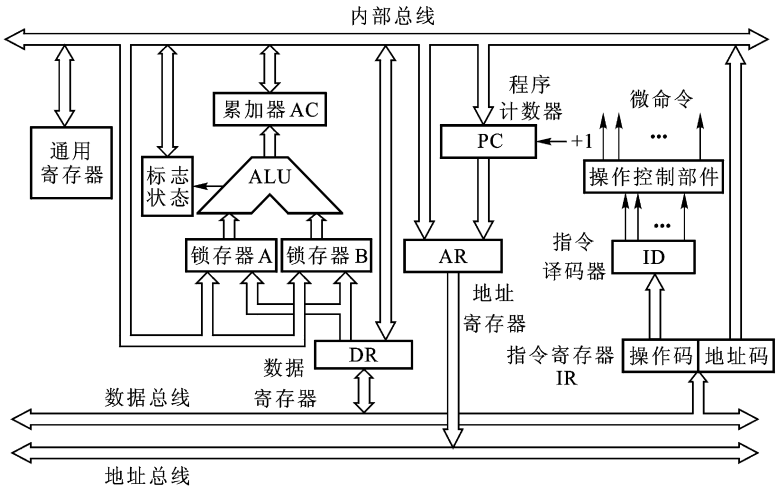


图 4-1-1 典型 CPU组成框图

累加寄存器 AC简称累加器,它除兼有通用寄存器的所有功能外,还有一些特殊功能,如在算术逻辑运算中,作为指令的隐含操作数寄存器、作为目标操作数寄存器等。其余各部分的功能已在第 3章中进行了介绍,这里不再赘述。

4.2 8086的内部结构

8086是 Intel 80x86系列的 16位微处理器,它是采用 HMOS工艺技术制造的,内部包含约 29 000个晶体管。

8086有 16根数据线,20根地址线,可寻址的地址空间达 1 MB。

8086工作时,只要一个 5 V电源和单相时钟,时钟频率为 5 MHz~10 MHz。

在推出 8086微处理器的同时,Intel公司为与当时已有的一整套 Intel外围设备接口芯片直接兼容使用,还推出了一种准 16位的微处理器 8088。8088的内部寄存器、内部运算器部件以及内部操作都是按 16位设计的,但对外的数据总线只有 8条。

要掌握一个 CPU的工作性能和使用方法,首先应该了解它的编程结构。所谓编程结构,就是指从程序员和使用者角度看到的结构,当然,这种结构与 CPU内部的物理结构和实际布局是有较大不同的。

图 4-2-1为 8086微处理器的内部结构,该结构是按编程结构给出的。从功能上 8086分为两大部分,总线接口部件 (Bus Interface Unit, BIU) 和执行部件 (Execution Unit, EU)。

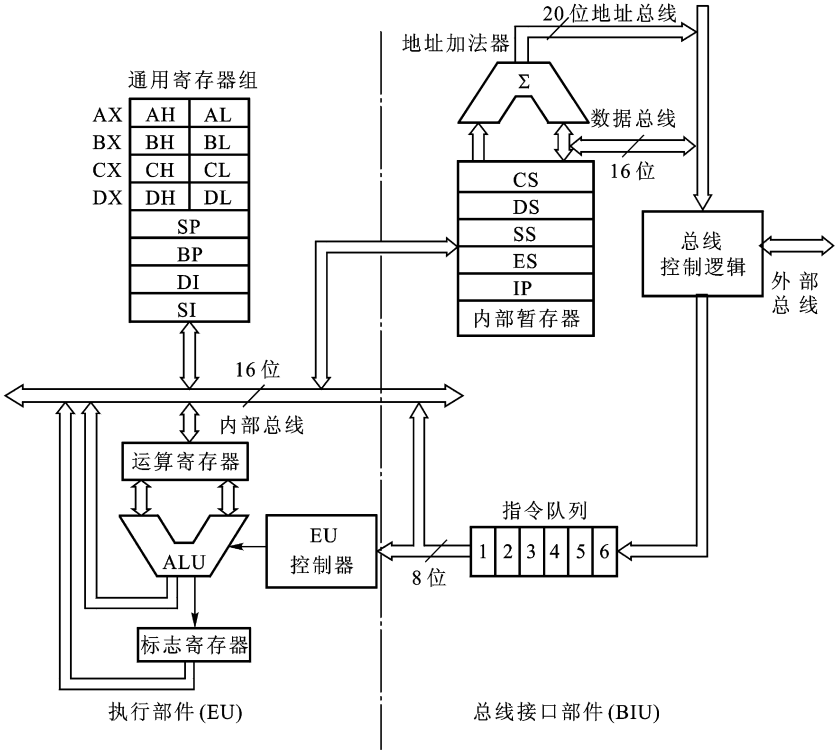


图 4-2-1 8086的内部结构

4.2.1 总线接口部件 BIU

1. 总线接口部件的功能

总线接口部件的功能是负责 CPU与存储器、I/O接口之间的数据传送。具体讲就是,从内存单元或者外设端口中取数据,传送给执行部件,或者把执行部件的操作结果传送到指定的内存单元或外设端口中。

2. 总线接口部件的组成

总线接口部件由下列各部分组成:

(1) 4个段地址寄存器

CS:代码段寄存器 (16位)

DS:数据段寄存器 (16位)

ES:附加数据段寄存器 (16位)

SS:堆栈段寄存器 (16位)

8086的存储器是分段使用的,在编程时,程序通常可分为代码段、数据段、堆栈段、附加数据段,程序和数据在内存中分段存储。由 CS存放代码段地址,DS存放数据段地址,ES存放附加数据段地址,SS存放堆栈段地址。关于分段使用的详细内容在后面介绍。

⑵ 指令指针寄存器（16位）

指令指针寄存器（Instruction Pointer，IP）类似于前面介绍的程序计数器 PC。程序运行时，由CS指定段地址，IP指定段内偏移量，即程序的运行地址由 CS和 IP共同给出。IP在程序运行时有自动增量的功能，即每运行一条指令时，IP自动指向下条指令地址。

⑶ 地址加法器（20位）

用于形成 20位访问地址。具体形成办法后面介绍。

⑷ 指令队列（6个字节）

8086的指令队列为 6个字节（8088为 4个字节），在执行指令的同时，从存储器中取下面一条或几条指令放入指令队列，这样 CPU执行完一条指令即可立即执行下条指令，不用等待执行完再取指令，提高了 CPU的效率。

⑸ 总线控制逻辑

对数据总线、地址总线、控制总线进行管理控制。

3. 8086 CPU地址形成

地址加法器用来产生 20位地址。因 8086有 20位地址线，可直接寻址的最大内存空间为 2^{20} =1 MB，即 00000H～FFFFFH。而 8086的内部所有寄存器都是 16位，不能直接计算和给出20位地址，需由地址加法器根据 16位信息计算出 20位的物理地址。

也就是在 8086的内存管理中，引入分段的概念。因寄存器均为 16位，所以每段最多为 2^{16} =64 KB空间，段与段之间是可重叠的。如图 4－2－2所示。

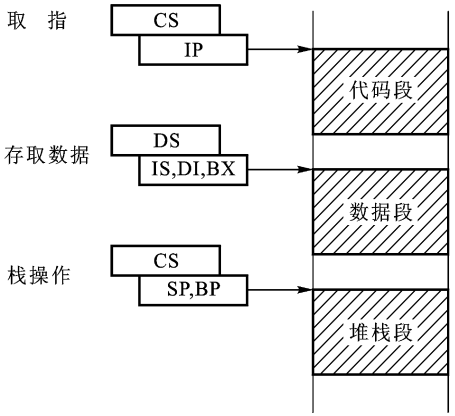


图 4－2－2 8086存储器的分段管理

程序设计中，一个程序可以有代码段、数据段、堆栈段和附加数据段，分别由段寄存器 CS、DS、SS、ES指向相应段的起始地址。8086 CPU一次只能访问某一个段中的一个单元，究竟访问段中的哪一个单元，由段内的偏移量指出，段内偏移量都是从零开始编址的。一般代码段的偏移量在 IP中；数据段偏移量根据不同寻址方式由多种形式给出，如寄存器间址寻址时偏移量由BX、SI等寄存器给出；栈段偏移量一般由 SP或 BP给出；附加段偏移量一般可由 DI给出。由地址加法器根据 16位信息计算出 20位的物理地址时，按如下方法计算：

物理地址 =段地址 ×16+偏移量

即物理地址是由段地址左移 4位加上段内偏移量得到。二进制数每左移 1位相当于乘 2,所以左移 4位相当于乘 16。如图 4-2-3所示。

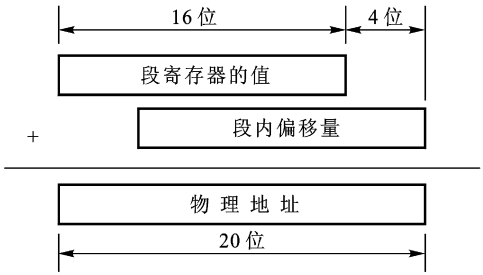


图 4-2-3　物理地址的构成

同一物理地址对应的段地址和偏移量是不唯一的。因同一物理存储单元可在不同的逻辑分段中。

[例 4.1]　设 CS=D200H,IP=2E00H,则

物理地址 =D2000H+2E00H=D4E00H

反过来,如果物理地址 =D4E00H

则 CS:IP可为 D200:2E00H,也可为 D140:3A00H等。

可见,在 8086中出现两种地址:逻辑地址和物理地址。

程序中使用的存储单元地址称为逻辑地址,其表现形式为:

段基址:段内偏移地址

[例 4.2]　设 CS=2200H,IP=4000H,则逻辑地址为 2200H:4000H。

在地址总线上提供的访问存储器单元的实际地址称为物理地址,由地址加法器形成。

[例 4.3]　设 CS=2200H,IP=4000H,其物理地址为 26000H。

4.2.2　执行部件 EU

1. 执行部件的功能

执行部件的功能是负责指令的执行。执行指令时,执行部件从总线接口部件的指令队列取出指令,由控制器单元内部的指令译码器进行译码,并给各部件发出相应的控制信号,完成指令的功能。即对数据进行运算和处理,并控制 BIU部件进行数据交换等。

2. 执行部件的组成

(1) 4个 16位通用寄存器

8086有 AX、BX、CX、DX共 4个 16位通用寄存器,一般用于暂存中间运行的结果和参加运算的数据。这 4个通用寄存器既可作为 16位寄存器使用,也可分为 8个 8位寄存器使用,即 AH、AL、BH、BL、CH、CL、DH、DL。AX也称为累加器,指令系统中有很多指令是利用累加器来执行的。

(2) 4个 16位专用寄存器

这4个专用寄存器是基址指针寄存器BP、堆栈指针寄存器SP、源变址寄存器SI和目的变址寄存器DI。

BP、SI、DI用于在寄存器间接寻址方式下，存放基地址和变址；SP用于在堆栈操作时，确定堆栈在内存中的位置，即指出段内偏移量。这几个寄存器也兼有通用寄存器的功能。

(3) 1个16位标志寄存器

标志寄存器存放CPU运算结果的特征状态和控制状态。

(4) 算术逻辑单元ALU

ALU是运算器的核心部件，用于完成数据的算术运算和逻辑运算等。

(5) EU控制电路

EU控制电路是控制器的核心部件，主要是对指令操作码进行译码，产生各种微操作控制信号，控制各部件完成指令功能。

3. 8086的标志字(Program Status Word, PSW)

8086标志寄存器16位，使用了其中的9位，设置了9个标志。这9个标志按功能分为两大类：状态标志(SF、ZF、PF、CF、AF、OF)和控制标志(DF、IF、TF)。各标志的安排如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

状态标志反映当前操作后算术逻辑单元ALU的状态。可通过测试这些标志来确定后面的操作。一般用于条件转移、条件调用等指令的判断条件。控制标志是人为设置的，对某一种特定的功能起控制作用。在指令系统中，有专门的指令用于设置和清除控制标志。下面分别介绍各个标志的具体作用。

① 符号标志SF(Sign Flag)：指出前面的运算执行后，其运算结果是正还是负。它和运算结果的最高位相同。SF=0，结果为正；SF=1，结果为负。

② 零标志ZF(Zero Flag)：指示当前运算结果是否为0。ZF=0，运算结果非零，ZF=1运算结果为零。

③ 奇偶标志PF(Parity Flag)：指示运算结果低8位1的个数的奇偶性。PF=0，运算结果1的个数为奇数；PF=1，运算结果1的个数为偶数。

④ 进位标志CF(Carry Flag)：指示运算结果最高位有无进(借)位。CF=0，无进(借)位；CF=1，有进(借)位。

⑤ 辅助进位标志AF(Auxiliary carry Flag)：反映运算时半字节有无进借位。半进位即一个字节中，低4位向高4位的进位。AF=0，无半进位；AF=1，有半进位。辅助进位标志一般在BCD码运算中作为是否进行十进制调整的判断依据。

⑥ 溢出标志OF(Overflow Flag)：反映运算结果有无溢出。OF=0，无溢出；OF=1，有溢出。

⑦ 方向标志DF(Direction Flag)：控制串操作的地址增减性。DF=0，串操作过程中地址不断增值；DF=1，串操作过程中地址不断减值。

⑧ 中断标志IF(Interrupter Flag)：控制是否允许响应可屏蔽中断。IF=0，禁止中断；IF=1，允许中断。

⑨ 跟踪标志 TF (Trap Flag):控制单步运行。 TF=0,非单步运行;TF=1,单步运行。

状态标志中,SF、ZF、PF、CF、OF是可测试的标志,AF是内部测试的,不能由用户通过软件的方法进行测试。

4. 堆栈的概念

堆栈是一段特殊组织的存储区域,即在普通随机存取存储器 RAM中,规定一段存储区域,这段存储区域在对存储单元进行操作时,其存取数据的顺序不是任意的,而是按“后进先出(Last In First Out)”原则进行存取。后存入的数据必须先取出,先存入的数据必须后取出。就像堆放的货栈,先放入的东西被压在下面,只能将后放入的东西取走才能将其取出。

堆栈用于子程序嵌套调用和嵌套中断,其作用是保护返回地址和保护现场(它符合后调用先返回的原则)。

8086的堆栈由高地址向低地址延伸。由 SS指定堆栈段地址,SP指定堆栈段内偏移量,SS:SP始终指向栈顶单元。在 8086指令系统中有专门的指令对堆栈进行操作。

4.2.3 BIU和 EU的动作管理

总线接口部件 BIU和执行部件 EU相互配合完成指令操作。二者不是同步工作的,是按一定的原则进行协同动作管理的,体现在如下几个方面:

① 每当指令队列中有 2个空字节,BIU就自动从内存中取出后继指令放入指令队列。

② 每当 EU执行完一条指令时,就会从指令队列前部取出指令代码去执行,当需访问内存或 I/O时,EU请求 BIU进入总线周期,完成访问操作。

③ 当指令队列满,又无内存或 I/O访问时,BIU空闲。

④ 执行转移、调用、返回指令时,下面要执行的指令就不是在程序中紧跟着排列的那条指令了,这时指令队列中已装入的指令就没用了,所以将指令队列原有的内容清除。

EU执行指令的操作与 BIU取指操作是可以并行进行的,这样可减少 CPU取指的等待时间,加快了 CPU执行指令的速度,提高了系统总线的利用率。

4.3 8086的引脚信号和工作模式

4.3.1 最大模式和最小模式的概念

8086 CPU可以工作在两种模式下,即最小模式和最大模式。

最小模式:在系统中只有 8086一个微处理器,所有的总线控制信号都直接由 8086产生,系统中的总线控制逻辑减到最小。

最大模式:在系统中包含两个或多个处理器,其中一个是主处理器,其他为协处理器,总线的部分控制信号由总线控制器 8288产生。

和 8086配合的协处理器有数值运算协处理器 8087和输入/输出协处理器 8089。

系统工作在最小模式还是最大模式,完全由硬件决定。最小模式和最大模式下部分控制总线的功能不同,下面介绍之。

4.3.2 8086的引脚信号和功能

8086芯片采用40引脚双列直插式(DIP)封装,引脚图如图4-3-1所示。

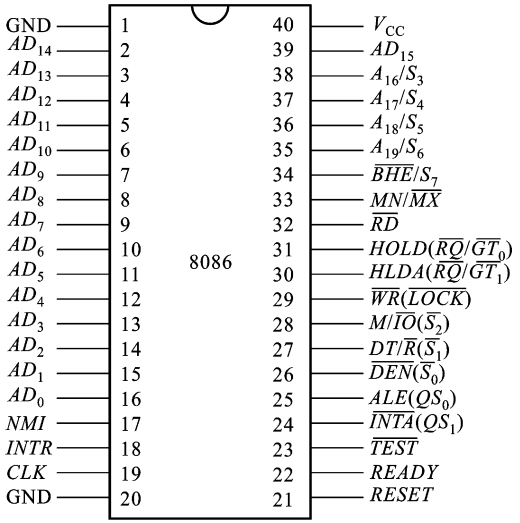


图4-3-1 8086引脚图

8086的引脚在不同模式下,除24～31引脚的功能不同之外,其他引脚功能相同。下面先介绍功能相同的引脚。

GND:地线。

V_{cc}:电源引线。8086采用单一的+5 V电源供电。

AD_{15}～AD_0:地址/数据线(Address Data Bus),复用引脚,双向三态。8086的数据线和地址线是复用的,分时传送地址信号和数据信号,即某一时刻总线上出现的是地址信号,另一时刻,总线上出现的是数据信号。

A_{19}/S_6～A_{16}/S_3:地址/状态线(Address/Status),复用引脚,输出。分时输出地址信号高4位和状态信号S_6～S_3。

S_6为0,指示8086当前与总线相连。

S_5反映中断标志的当前值,如为1,表示允许中断请求,如为0表示禁止中断请求。

S_4、S_3用来指示当前正在使用哪个段寄存器,用于多处理器协调。表4-1给出S_4、S_3的组合含义。

表4-1 S_4、S_3的组合及含义

S_4	S_3	使用的段寄存器
0	0	ES
0	1	SS
1	0	CS(或不用)
1	1	DS

$\overline{BHE}/S_7$:高位数据总线允许/状态信号(Bus High Enable/Status),复用输出引脚。

$\overline{BHE}$低电平有效,表示高8位数据线D_{15}～D_8上数据有效。S_7未使用。

$\overline{BHE}$与 A_0合起来,给出几种组合操作,通知内存或外设,当前的数据在总线上将以何种格式出现,如表 4-2所示。

表 4-2 $\overline{BHE}$与 A_0的组合所对应的操作

$\overline{BHE}$	A_0	操 作	数据引脚
0	0	从偶地址开始读/写一个字	AD_{15}～AD_0
1	0	从偶地址开始读/写一个字节	AD_7～AD_0
0	1	从奇地址开始读/写一个字节	AD_{15}～AD_8
0	1	从奇地址开始读/写一个字	AD_{15}～AD_8
1	0	(两个周期完成)	AD_7～AD_0

在 8086系统中,如果从奇地址单元开始读/写一个字,需要两个总线周期,而从偶地址单元开始读/写一个字,只需要一个总线周期即可完成。

NMI:非屏蔽中断请求 (Non-Maskable Interrupter),输入引脚,上升沿有效。用于向 CPU发出非屏蔽中断请求信号。CPU在完成当前指令后,将响应非屏蔽中断请求。该中断请求不受中断允许标志 IF的影响。

INTR:可屏蔽中断请求 (Interrupt Request request),输入引脚,高电平有效。用于向 CPU发出可屏蔽中断请求信号。CPU在每条指令最后一个时钟采样该引脚信号,若为高电平,并且中断标志 IF=1,则 CPU在完成当前指令后,响应可屏蔽中断请求,否则继续执行下条指令。

$\overline{RD}$:读信号 (Read),输出引脚,低电平有效。当 $\overline{RD}$输出低电平时,指示 CPU正执行一个对内存或外设端口的读操作。

CLK:时钟输入 (Clock)。为 CPU和总线控制逻辑提供基准时钟。8086的时钟频率是 5MHz～10MHz。

RESET:复位信号 (Reset),输入引脚,高电平有效。要求至少维持 4个时钟周期。RESET信号有效后,CPU立即停止当前操作,进行处理器初始化,并使系统重新启动。CPU内部的复位操作,使指令队列置空,CS置为 FFFFH,其余寄存器全部清零。

READY:"准备好"信号,输入引脚,高电平有效。是由所访问的存储器或外设发出的响应信号。有效时,表示内存或 I/O设备准备就绪,可进行数据传输;无效时,表示内存或 I/O设备数据未准备好,CPU自动插入等待时钟周期,直到 READY信号有效后,才脱离等待状态,进行数据的传输。

$\overline{TEST}$:等待测试信号,输入引脚,低电平有效。

该信号与指令 WAIT结合使用,执行 WAIT使 CPU处于等待状态,每隔 5个时钟周期,CPU测试 $\overline{TEST}$信号一次,若为低电平 (有效),则结束等待状态,否则继续保持等待状态。

MN/$\overline{MX}$:最大/最小模式选择信号 (Minimum/Maximum Mode control),输入引脚。

MN/$\overline{MX}$=0时 8086工作在最大模式,MN/$\overline{MX}$=1时 8086工作在最小模式。

上述 32个信号在最大、最小模式下意义相同,另外 8个信号,即 24～31引脚,在最大、最小模式下意义不同,下面分别进行介绍。

4.3.3 最小模式

1. 最小模式的引脚功能

MN/$\overline{MX}$=1时为最小模式,这时 24～31引脚的定义如下：

$\overline{INTA}$:中断响应信号 (Interrupt Acknowledge),输出引脚,低电平有效。

用于对外设的中断请求作出响应。当 CPU响应中断时,输出两个连续的负脉冲,通知外设中断已经允许,外设可将中断类型码送数据总线 DB。

ALE:地址锁存允许信号 (Address Latch Enable),输出引脚,高电平有效。控制地址锁存器锁存地址信息,即用下降沿将地址信号打入地址锁存器,实现地址总线与数据总线的分时复用。

$\overline{DEN}$:数据允许信号 (Data Enable),输出引脚,低电平有效。

表示 CPU准备发送或接收一个数据。可作为数据总线收发器的输出允许信号。

DT/$\overline{R}$:数据发送/接收方向控制信号 (Data Transmit/Receive),输出引脚。作为数据总线收发器的数据传送方向的控制信号,高电平时进行数据的发送,低电平时进行数据的接收。

M/$\overline{IO}$:存储器/输入输出控制信号 (Memory/Input and Output),输出引脚。区分 CPU进行存储器访问还是进行输入/输出端口访问的控制信号。

M/$\overline{IO}$=1为存储器访问,M/$\overline{IO}$=0为 I/O访问。

$\overline{WR}$:写控制信号 (Write),输出引脚,低电平有效。当该信号有效时,表示 CPU当前正进行存储器或 I/O写操作。

HOLD:总线保持请求信号 (Hold Request),输入引脚,高电平有效。

CPU之外的主模块请求占用总线的请求信号。当为高电平时,表示外部总线设备向 CPU申请总线使用权。

HLDA:总线保持响应信号 (Hold Acknowledge),输出引脚,高电平有效。该信号有效表示 CPU对其他主模块的总线请求作出响应,让出了总线控制权。

2. 最小模式下的典型配置

最小模式下的典型配置,如图 4－3－2所示。

在最小模式下,硬件连接上有如下特点：

① MN/$\overline{MX}$=1决定了 8086工作在最小模式。

② 8284作为时钟发生器,供给系统时钟脉冲。

③ 3片 8282 (或 74LS373) 作为地址锁存器。

④ 当需增加数据总线驱动能力时,可用 2片 8286/8287作总线收发器。

8282是 8位锁存器,用来锁存地址信号。因为地址信号线与数据信号线是分时复用的,当地址信号有效时,将其锁存在地址锁存器中,以便下一步用此复用线进行数据传送。ALE为地址锁存控制,$\overline{BHE}$同地址线一起被锁存。3片 8282共 24位,锁存 20位地址,1位$\overline{BHE}$,3位空闲,选通信号输入端 STB与 ALE相连,$\overline{OE}$为输出允许。这里$\overline{OE}$接低电平,使输出一直有效。

8286为数据收发器 (总线驱动器),8位双向驱动,用来在总线驱动能力不够时,进行数据总线的双向驱动。用 T来控制数据的传送方向,T=1发送数据,T=0接收数据,CPU用 DT/$\overline{R}$控制

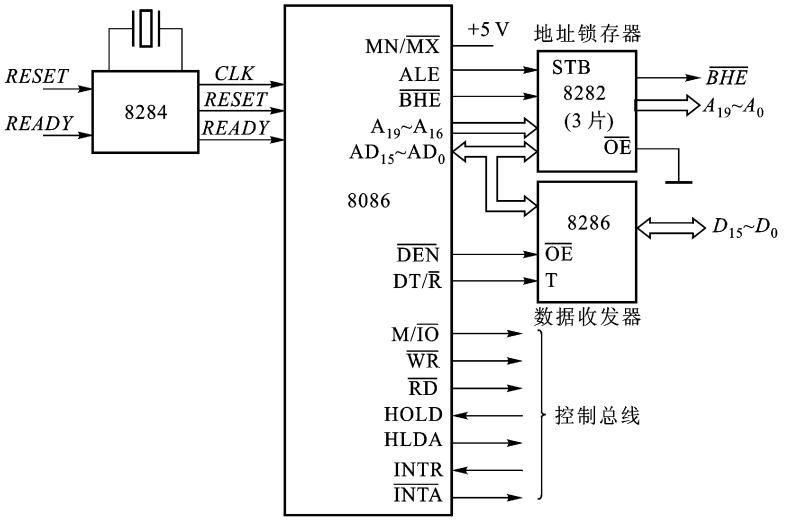


图 4－3－2　8086在最小模式下的典型配置

T端。$\overline{OE}$是输出允许,为 0时允许数据传输,为 1时不能传输,由$\overline{DEN}$控制。

在最小系统中,信号 M/$\overline{IO}$与$\overline{WR}$和$\overline{RD}$组合起来决定了系统中数据传输的方式,包括存储器读、存储器写、I/O读、I/O写 4种操作。如表 4－3所示。

表 4－3　M/$\overline{IO}$与$\overline{WR}$和$\overline{RD}$的配合功能

M/$\overline{IO}$	$\overline{WR}$	$\overline{RD}$	功　能
0	0	1	I/O写
0	1	0	I/O读
1	0	1	存储器写
1	1	0	存储器读

注意:$\overline{WR}$信号和$\overline{RD}$信号不能同时有效,即不能同时为低电平,如都为高电平表示不进行读写操作。

3. 8284A时钟发生器简介

8284A是时钟发生器,为 8086提供基准脉冲源。8284A和 CPU的连接如图 4－3－3所示。

8284A不但提供恒定的时钟 CLK,且对 READY和 RESET信号进行同步。外界的准备好信号输入给 8284A的 RDY,已被同步的 READY信号从 8284A输出,供给 8086。同样

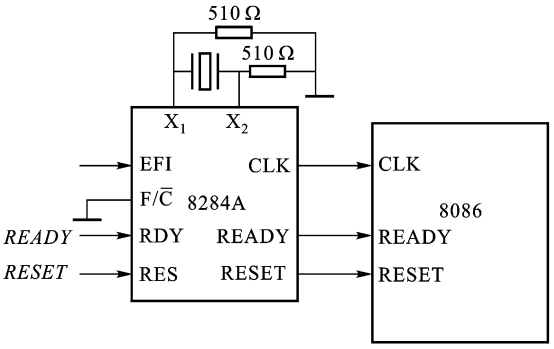


图 4－3－3　8284与 8086的连接

外界的复位信号输入给 8284A 的 RES，已被同步的复位信号 RESET从 8284A 输出，供给 8086。在 8284A 内部由时钟的下降沿同步。

8284A 的时钟输出频率为振荡源频率的 1/3。

根据不同的振荡源，8284A 有两种连接的方法：

① 用外接脉冲发生器作为振荡源。这时将脉冲发生器的输出端接 8284A 的 EFI，F/$\overline{C}$=1即可。

② 利用晶体振荡器作为振荡源。这时将晶振接于 8284A 的 X1、X2间，F/$\overline{C}$=0即可。

4.3.4 最大模式

1. 最大模式下的引脚功能

MN/$\overline{MX}$=0时，CPU工作在最大模式。最大模式下 24～31引脚的功能如下：

QS_0、QS_1：指令队列状态信号（Instruction Queue Status）。

输出引脚，提供前一个时钟周期中指令队列的状态，以便外部对 8086内部指令队列的动作进行跟踪。其组合含义如表 4－4所示。

表 4－4 QS_0、QS_1 组合及其含义

QS_1	QS_0	含 义
0	0	无操作
0	1	从指令队列的第一个字节中取走代码
1	0	队列为空
1	1	除第一个字节外，还取走了后继字节的代码

$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$：总线周期状态输出信号（Bus Cycle Status）。

3个信号组合指示当前总线周期中所进行的数据传输过程的类型。

最大模式下，总线控制器 8288用 $\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$ 作输入，产生存储器和 I/O的控制信号。控制状态如表 4－5所示。

表 4－5 $\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$ 的组合对应的操作

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	操 作 过 程
0	0	0	发中断响应信号
0	0	1	读 I/O
0	1	0	写 I/O
0	1	1	暂停
1	0	0	取指令
1	0	1	读内存
1	1	0	写内存
1	1	1	无源状态①

① 无源状态一个总线操作过程就要结束，另一个新的总线周期还未开始的状态称为无源状态。

$\overline{LOCK}$:总线封锁信号。输出引脚,低电平有效。

有效时,系统中其他总线主部件就不能占有总线。$\overline{LOCK}$由指令前缀 LOCK产生,在一条指令内有效。

$\overline{RQ}/\overline{GT_1}$、$\overline{RQ}/\overline{GT_0}$:总线请求/总线允许信号 (Request/Grant)。输入/输出引脚。作为输入时,是总线请求信号;作为输出时,为总线允许信号。允许两个总线部件 (或协处理器)同时向主处理器请求使用总线。$\overline{RQ}/\overline{GT_0}$的优先级高于$\overline{RQ}/\overline{GT_1}$。

2. 最大模式下的典型配置

最大模式下的典型配置如图 4-3-4所示。

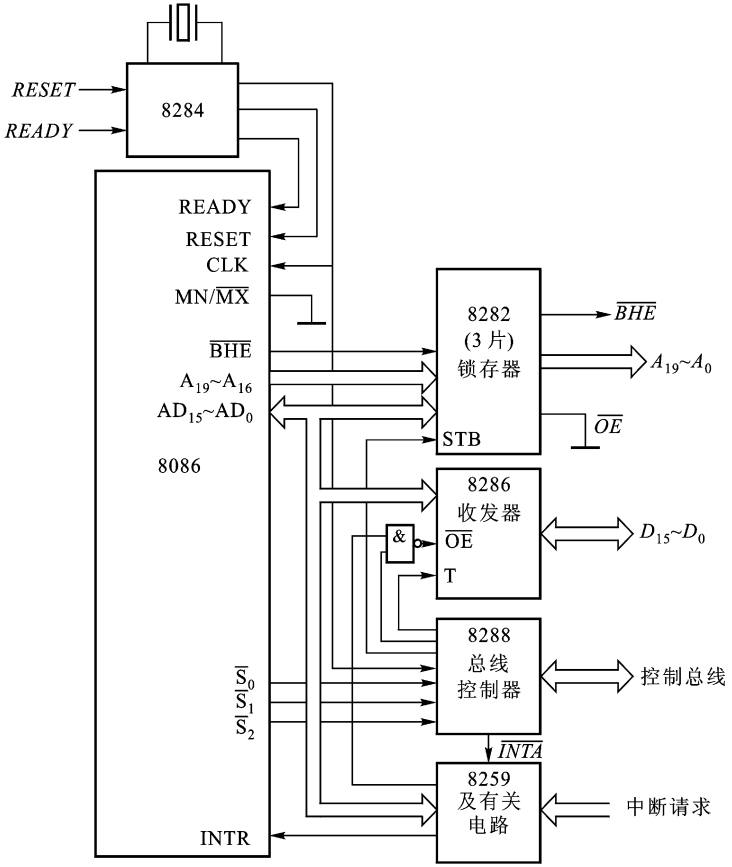


图 4-3-4　8086在最大模式下的典型配置

与最小模式的主要区别是,用 8288总线控制器对 CPU发出的控制信号进行变换和组合,产生新的控制信号。

在最大模式下,一般包含 2个或多个处理器,要解决主处理器和协处理器之间的协调工作问题和对总线的共享问题,需要有更多的控制信号,用总线控制器,产生更多的控制信号,来满足多处理器的需要。

在最大模式下,一般还有中断管理部件,本系统中,采用中断控制器 8259作为中断管理部件,关于 8259将在后面章节详述。

3. 8288总线控制器简介

8288接收时钟发生器的 CLK信号和来自 CPU的 $\overline{S_0}$、$\overline{S_1}$、$\overline{S_2}$ 信号,产生相应的各种控制信号和时序,并提高总线驱动能力。

CLK使 8288与 CPU同步。

$\overline{S_0}$、$\overline{S_1}$、$\overline{S_2}$ 相当于 M/$\overline{IO}$、$\overline{S_1}$相当于 DT/$\overline{R}$。

8288采用 20脚封装,引脚如图 4-3-5所示。

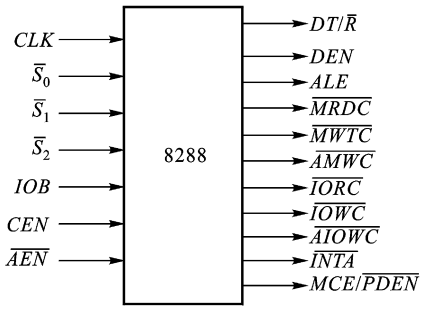


图 4-3-5 总线控制器 8288的引脚图

8288产生的 ALE、$\overline{INTA}$、DEN、DT/$\overline{R}$信号与最小模式下 CPU相应的引脚信号相同,只有 DEN反相,即高电平有效。其他信号介绍如下:

IOB:I/O总线方式。IOB=0时为单处理器方式,IOB=1为多处理器方式。

CEN:命令允许。有效时输出命令有效。

$\overline{AEN}$:地址允许。多总线系统中与总线裁决器 8289的 $\overline{AEN}$连接,有效时,使 I/O读写及存储器读写之一有效。

MCE/$\overline{PDEN}$:总线主模块允许/外设数据允许信号 (Master Cascade Enable/Peripheral Data Enable)。

IOB=0时为 MCE,用于多片 8259的 CAS_0、CAS_1、CAS_2 驱动器控制。IOB=1时为 $\overline{PDEN}$,作为收发器开启信号。

$\overline{MRDC}$:存储器读控制 (Memory Read Command)。

$\overline{MWTC}$:存储器写控制 (Memory Write Command)。

$\overline{IORC}$:I/O读控制 (I/O Read Command)。

$\overline{IOWC}$:I/O写控制 (I/O Write Command)。

$\overline{AMWC}$:存储器提前写控制 (Advanced MWTC)。

$\overline{AIOWC}$:IO提前写控制 (Advanced IOWC)。

$\overline{AMWC}$和 $\overline{AIOWC}$与 $\overline{MWTC}$和 $\overline{IOWC}$功能相同,只是提前一个时钟周期有效,适用于较慢的存储器芯片或 I/O设备。

4.3.5 系统的复位和启动操作

8086复位和启动操作是由 RESET引脚的触发信号来执行的。要求 RESET维持 4个时钟周期的高电平,如果是初次加电,要求 RESET维持不小于 50 μs的高电平。

当 RESET信号一进入高电平,8086就进入复位状态,在复位状态下 CPU的状态如下:

① CS=FFFFH,其他寄存器清零。

② 指令队列空。

③ 标志位寄存器清零。

④ 总线置成无效状态。

因复位后 CS被初始化为 FFFFH,IP被初始化为 0000H,所以启动地址为:

$$CS\times16+IP=FFFF0H+0000H=FFFF0H$$

一般在 FFFF0H处放一条无条件转移指令,转到系统程序入口。这样启动后就直接去执行系统程序。

复位时,因标志状态寄存器被清零,所以中断控制标志 IF=0,这时 CPU处于禁止中断状态,所以程序运行时,应在适当的时候开中断。

4.4　8086 CPU的操作时序

4.4.1　时钟周期、指令周期和总线周期

1. 时钟周期

CPU的任何操作都是在时钟脉冲信号 CLK的统一控制下,一个节拍一个节拍地工作。时钟脉冲的间隔时间称为时钟周期,通常称为 T周期 (或称 T状态)。时钟周期是 CPU的基本时间计量单位。

时钟周期即为微处理器的时钟主频 f的倒数,即 T=1/f。

[例 4.4]　若 8086的主频 f=5 MHz,则时钟周期 T=1/f=0.2 μs=200 ns。

2. 指令周期

指令周期即为执行一条指令所用的时间。指令周期的长度,以时钟周期 T为单位来计量。在微型机系统中,一般不同指令的执行时间不同,所以不同指令的指令周期长度不等。例如 8086 CPU的最短指令为 2T,而最长的指令周期达到 200T左右。

3. 总线周期

CPU与存储器或 I/O端口进行信息交换都是通过总线来完成的。完成一次总线操作所用的时间称为总线周期。取指令和传送数据时,需要 BIU执行一个总线周期。8086的一个基本总线周期由 4个时钟周期完成,即一般一个典型总线周期由 4个 T状态组成,分别以 T_1、T_2、T_3、T_4 表示,如图 4-4-1所示。

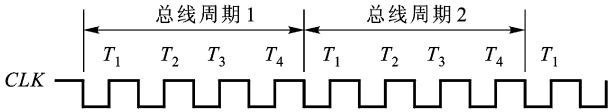


图 4-4-1　总线周期示意图

每个基本时钟周期完成一些基本操作。在 8086系统中,一般完成基本操作情况如下:

① T_1 状态:总线周期开始,地址/数据总线上输出要访问存储单元或 I/O端口的地址信号。

② T_2 状态:CPU从地址/数据总线上撤销地址信号,使地址高 4位输出状态信息,低 16位变成高阻态。以便下面 CPU使用该地址/数据线进行数据的传送。

③ T_3、T_4 状态:数据在 CPU与存储器或 I/O端口之间进行传送。这时低 16位地址/数据复用总线作为数据线使用。T_4 状态还标志着总线周期结束。

当存储器或外设速度慢时，可通过 READY线，在 T_3 和 T_4 之间插入一个或多个等待周期 T_W 来延长总线周期时间，以便与慢速设备同步操作。

下面介绍 8086 CPU的典型总线操作时序。

4.4.2 最小模式下的总线读周期

当 CPU进行存储器或 I/O端口读操作时，进入总线读周期，如图 4－4－2所示为总线读周期时序图。

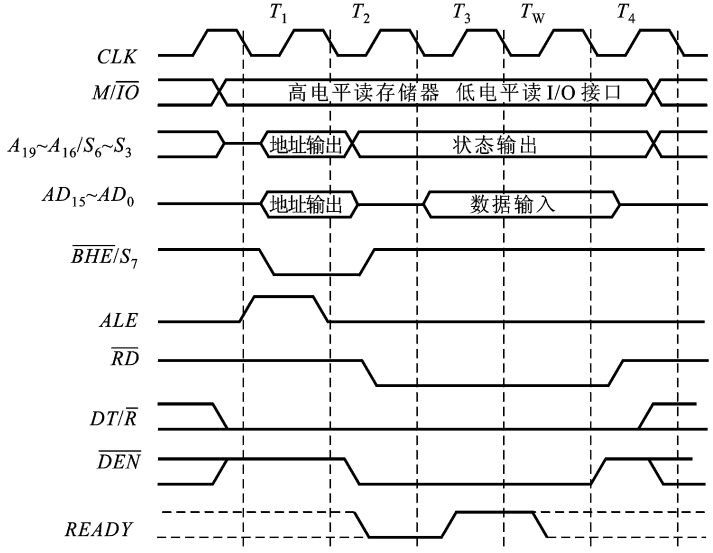


图 4－4－2 8086最小模式下总线读周期时序

⑴ T_1 状态

首先，用 M/IO信号指示是存储器读还是 I/O端口读，所以 M/IO信号在 T_1 状态有效，并一直保持到 T_4，即整个总线周期结束。如果是存储器读，则 M/IO为高；如果是 I/O端口读，则 M/IO为低。

此外，CPU要指出所读的存储单元地址或 I/O端口地址。8086的 20位地址信号 A_{19}～A_0 通过地址/数据复用引线给出，其中，高 4位 A_{19}～A_{16}通过 A_{19}/S_6～A_{16}/S_3 给出，低 16位 A_{15}～A_0 通过 AD_{15}～AD_0给出。

地址信号必须锁存起来，才能在其他 T状态下，使用这些复用引线给出状态信号和进行数据的传输。为了实现地址锁存，CPU在 T_1 状态下从 ALE引脚输出一个正脉冲，作为地址锁存信号。ALE到来之前地址信号已有效，地址锁存器用 ALE的下降沿对地址锁存。

BHE也在 T_1 状态送出，它用来表示高 8位数据线上的信号有效。同样由 ALE的正脉冲锁存。

当系统中接有数据收发器时，使用 DT/R和 DEN作为控制信号。控制数据的传送方向和数据选通。在 T_1 状态时 DT/R＝0表示数据接收，即读周期。

(2) T_2 状态

在 T_2 状态，地址信号消失，AD_{15} ～AD_0 进入高阻态，为读入数据作准备。此时 A_{19}/S_6 ～A_{16}/S_3 及 $\overline{BHE}/S_7$ 引脚上输出状态信号 S_7 ～S_3。$\overline{DEN}$在 T_2 状态有效（低电平），提供数据收发器允许信号。同时$\overline{RD}$有效（低电平），输出读控制信号，读入数据。

(3) T_3 状态

在 T_3 状态时，内存单元或 I/O端口将数据送上数据总线，CPU通过 AD_{15} ～AD_0 接收数据。

若内存或外设工作速度慢，不能在 T_3 状态送出数据，就要从 READY线上给出低电平，通知CPU数据没有准备好，需要等待，CPU在 T_3 的前沿（下降沿）采样 READY，若为低电平（无效），则在 T_3 和 T_4 之间插入等待周期 T_W。以后在每个 T_W 前沿再采样 READY线，若为低电平，则继续插入 T_W，直到为高电平，则停止插入 T_W 进入 T_4。

(4) T_4 状态

CPU在 T_4 状态前沿（下降沿）处，对数据线进行采样，获得数据。

4.4.3 最小模式下的总线写周期

当 CPU进行存储器或 I/O端口写操作时，进入总线写周期，如图 4－4－3所示为总线写周期时序图。

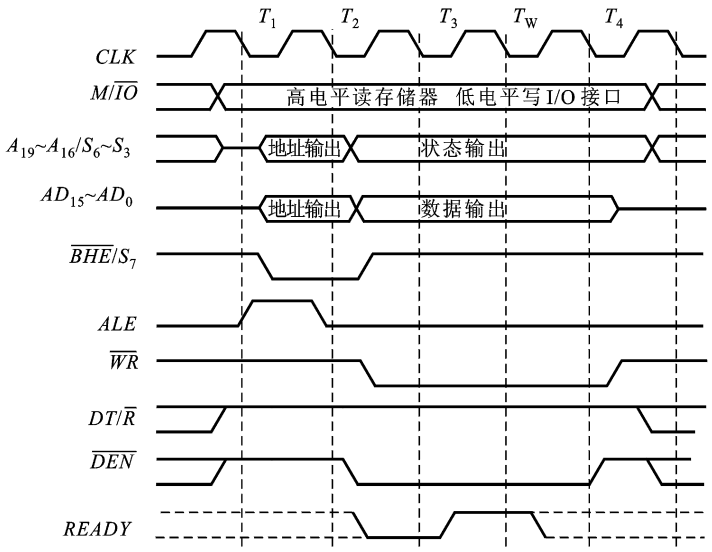


图 4－4－3　8086最小模式下总线写周期时序

与总线读周期类似，基本写周期也包含 4个 T状态。当存储器或外设工作速度较慢时，在 T_3 和 T_4 之间插入 1个或多个等待周期 T_W。

(1) T_1 状态

用 M/$\overline{IO}$信号指示是存储器写还是 I/O端口写，如果是存储器写，则 M/$\overline{IO}$为高电平；如果是

I/O端口写，则 $M/\overline{IO}$为低电平。

CPU通过 $A_{19}/S_6 \sim A_{16}/S_3$ 给出所写存储单元高 4位地址，通过 $AD_{15} \sim AD_0$ 给出所写存储单元低 16位地址或 I/O端口地址。用 ALE引脚输出一个正脉冲（下降沿）对地址进行锁存。

$DT/\overline{R}=1$表示数据发送。

(2) T_2 状态

在 T_2 状态，地址信号消失后，立即从 $AD_{15} \sim AD_0$ 发出写数据且一直保持到 T_4 状态。此时 $A_{19}/S_6 \sim A_{16}/S_3$ 及 $\overline{BHE}/S_7$ 引脚上输出状态信号 $S_6 \sim S_3$。$\overline{DEN}$在 T_2 状态有效（低电平），提供数据收发器允许信号。

在 T_2 状态时$\overline{WR}$有效（低电平），输出写控制信号，一直维持到 T_4 状态。

(3) T_3、T_4 状态

在 T_3 状态时，CPU继续维持各信号有效，直到 T_4 状态，完成了写入数据操作。

若内存或外设工作速度慢，就通过 READY信号在 T_3 和 T_4 之间插入等待周期 T_W。

4.4.4 最大模式下的总线读周期

在最大模式下，总线读周期与最小模式的总线读周期类似，但要考虑 CPU和总线控制器两者产生的控制信号。

在最大模式下，每个总线周期开始之前，$\overline{S_2}\,\overline{S_1}\,\overline{S_0}$必置为高电平，表示处于无源状态。总线控制器只要检测到$\overline{S_2}\,\overline{S_1}\,\overline{S_0}$中的任何 1个或几个从高电平开始变为低电平，便立即开始一个新的总线周期。

图 4-4-4为最大模式下的读操作时序。图中带星号的信号，为总线控制器发出的信号。

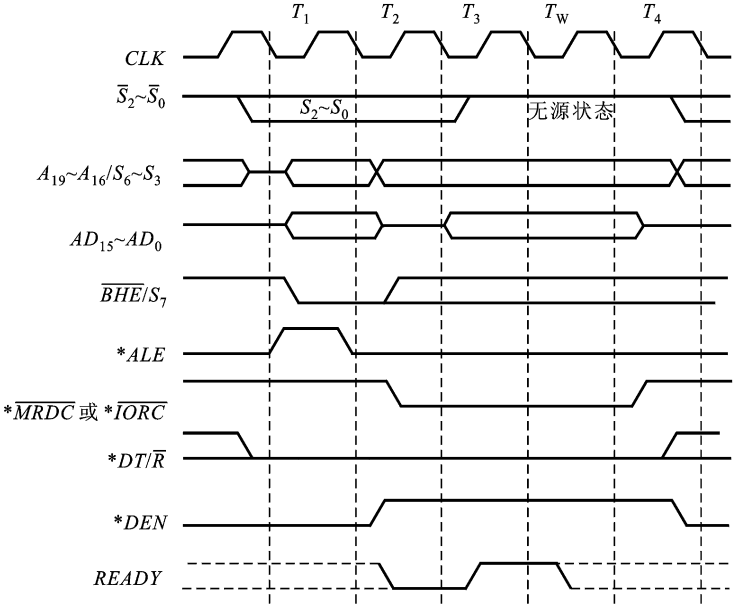


图 4-4-4 最大模式下总线读周期时序

在最大模式下,地址信号的输出和锁存、数据信号的输入及插入等待周期等操作与最小模式相同。这里不再重复,下面只介绍不同点。

① 在总线周期开始时,总线控制器根据 $\overline{S_2}\ \overline{S_1}\ \overline{S_0}$信号的电平组合产生一系列控制信号,完成总线周期的控制操作。

② 最大模式下$\overline{RD}$也有效,但一般使用总线控制器产生的存储器读信号$\overline{MRDC}$或 I/O读信号$\overline{IORC}$,而不使用$\overline{RD}$。因为$\overline{MRDC}$或$\overline{IORC}$信号交流特性好,且直接指出了具体读目标是存储器还是外设,所以在最大模式下无 M/$\overline{IO}$信号。

③ 在 T_3 状态下,CPU可以获得数据。使$\overline{S_2}\ \overline{S_1}\ \overline{S_0}$完全进入高电平,可很快启动一个新的总线周期。

④ T_4 状态下,$\overline{S_2}\ \overline{S_1}\ \overline{S_0}$按照下一个总线周期的操作类型产生电平变化。

4.4.5　最大模式下的总线写周期

8086在最大模式下工作时,CPU通过总线控制器提供两组写信号,一组是普通存储器写信号$\overline{MWTC}$和普通 I/O端口写信号$\overline{IOWC}$;另一组是提前存储器写信号$\overline{AMWC}$和提前 I/O端口写信号$\overline{AIOWC}$。提前写信号比普通写信号提前一个时钟周期开始起作用,给一些速度较慢的设备提供较长的写有效时间。

图 4-4-5是最大模式下总线写周期时序图。

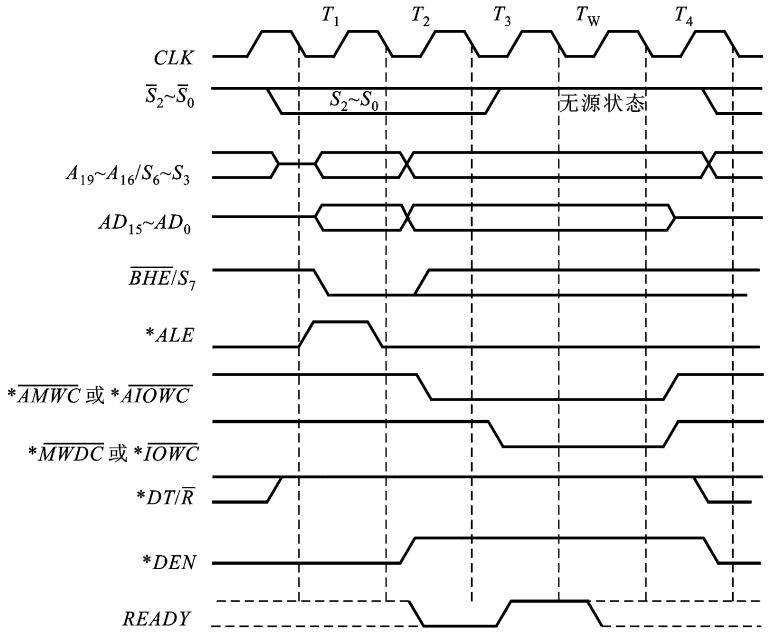


图 4-4-5　最大模式下总线写周期时序

在最大模式下总线写周期时序中,除了以下几点需要说明之外,其他方面与最大模式总线读

周期相同 图中带星号的信号为总线控制器发出的信号)。

① 在 T_1 状态下,DT/$\overline{R}$为高电平表示数据发送,控制数据收发器处于数据发送状态。

② CPU从 T_2 开始就把输出数据送到数据总线上。

③ 提前写信号$\overline{AMWC}$或$\overline{AIOWC}$在 T_2 前沿开始有效,$\overline{MWTC}$和$\overline{IOWC}$在 T_3 的前沿开始有效。

4.4.6 总线空操作

CPU在不执行总线周期时,进入总线空闲周期 T_I,T_I中 S_3 ~S_5 和前一个总线周期相同,若前一个周期为写,则 AD_{15} ~AD_0 上会继续驱动前一个总线周期数据,若前一个周期为读周期,则 AD_{15} ~AD_0 在 T_I 时为高阻态。

总线空操作是总线接口部件对执行部件的等待。

4.4.7 最小模式下的总线保持

当系统中有多个总线主模块时,CPU以外的其他总线主模块为获得总线控制权,需要向CPU发出使用总线的请求信号,CPU得到请求之后,如果同意让出总线,就发出响应信号。

在最小模式下,外部总线主模块通过 HOLD 引脚向 CPU发出总线保持请求信号,CPU通过HLDA引脚发出回答信号。

在每个时钟脉冲的上升沿处,CPU对 HOLD信号进行检测,若检测到高电平,且允许让出总线时,则在总线周期的 T_4 状态或空闲周期 T_I 之后的下一个时钟周期,CPU发出 HLDA信号,并释放总线控制权,直到 HOLD无效为止。

图 4-4-6为最小模式下的总线保持请求/响应时序图。

8086 CPU一旦让出总线控制权,便使地址/数据引脚、地址状态引脚及大部分控制引脚处于浮空状态,与总线脱离。

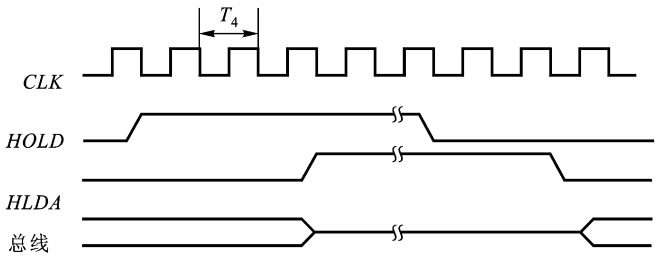


图 4-4-6 最小模式下的总线保持请求/响应时序图

4.4.8 最大模式下的总线请求/允许

在最大模式下,总线控制是通过两个总线请求/总线允许信号$\overline{RQ}/\overline{GT_0}$和$\overline{RQ}/\overline{GT_1}$来完成的。

如图 4-4-7为最大模式下总线控制时序。

当其他总线主模块需要使用总线时,在$\overline{RQ}/\overline{GT}$上向 CPU发一个负脉冲,宽度为一个时钟

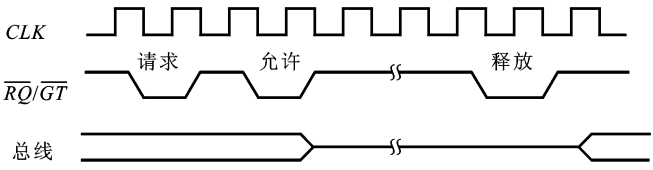


图 4-4-7　最大模式下的总线控制时序图

周期。

CPU在每个周期的上升沿对$\overline{RQ}/\overline{GT}$进行检测（采样），若测得一个负脉冲，则在下一个 T_4 状态或 T_I 状态从同一个引脚$\overline{RQ}/\overline{GT}$上发一个允许负脉冲，且让出总线控制权。

其他总线主模块收到允许脉冲后，便得到了总线控制权，进行总线操作。当外部总线主模块准备释放总线时，便在$\overline{RQ}/\overline{GT}$上发一个释放脉冲，CPU检测到释放脉冲后，在下一个时钟周期便收回总线控制权。

4.5　80386微处理器

80386微处理器是 Intel公司 1985年推出的全 32位微处理器。在内存管理和处理器速度上比 80286有很大突破，是一种功能完善、高可靠性的 CPU。但在目标代码上保持与 8086、80286的向上兼容。

80386 CPU的特点：

① 数据总线和地址总线都是 32位，指令系统提供 8位、16位、32位和 64位数据类型，可直接寻址的物理地址空间达 4 GB。

② 除了保留存储器分段管理外，还增加了内存分页管理。每个任务具有 64 TB的虚拟存储空间。

③ 采用了 6级并行流水线结构，引入片上地址转换高速缓存。

④ 有 3种工作模式：实模式、虚模式和虚拟 8086模式。

⑤ 增加了可测试特性和调试功能。

4.5.1　80386的组成

1. 80386的 6个组成单元

80386由 6个单元组成，这 6个单元是：总线接口单元、指令预取单元、指令译码单元、执行单元、分段单元和分页单元。

⑴ 总线接口单元

总线接口单元负责 80386与外部部件的高速接口，即负责取指令和数据传送，产生执行当前总线周期的地址、数据和控制信号，还控制与其他总线主控设备及协处理器的接口。

⑵ 指令预取单元

指令预取单元在总线空闲时，使用总线接口单元预取指令，填入 16 B的指令队列中，等待指令译码单元进行译码。

⑶ 指令译码单元

指令译码单元从预取指令队列取出指令流字节进行译码,将译得的微码存入译码指令队列,等待执行部件处理。

⑷ 执行单元

执行单元负责执行译码指令队列的指令,并与有关部件进行通信。它包括 8个 32位的通用寄存器和 1个 64位桶型移位器,提高了乘除法的运算速度。

⑸ 分段单元

分段单元负责将指令的逻辑地址变换为线性地址,同时由保护机构完成分段合法性检查,将转换后的线性地址送分页单元。

⑹ 分页单元

分页单元负责将线性地址换算成物理地址。

2. 寄存器组

80386的寄存器组共有 32个寄存器,分为 7类。如图 4-5-1所示。

⑴ 通用寄存器

80386共有 8个 32位的通用寄存器,EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP。这些寄存器的低 16位可作为 16位寄存器单独访问,分别为 AX、BX、CX、DX、SI、DI、BP、SP。其中 AX、BX、CX、DX这 4个 16位寄存器又可分为 8个 8位寄存器使用,分别为 AH、BH、CH、DH和 AL、BL、CL、DL。这样保证了与 8086和 80286的兼容。

⑵ 状态寄存器

80386状态寄存器包括 2个 32位的寄存器,指令指针寄存器 EIP和标志寄存器 EFR。

EIP用于保存下一条待取指令的偏移地址,其低 16位称为 IP,和 8086一样。若运行 16位指令时,代码段使用 16位偏移地址,即为 IP寄存器,其最大段大小为 64 KB。若运行 32位指令时,代码段使用 32位偏移地址,即为 EIP寄存器,其最大段大小为 4 GB。

EFR的低 16位为 FR,包含 9个标志,其定义与 8086相同,高 16位使用了 3位,定义了 3个标志,恢复标志 RF、虚拟 86模式标志 VM和对界标志 AC。

EFR的格式如图 4-5-2所示。

恢复标志 RF,是与指令断点异常相关的标志。RF=1不产生异常中断,RF=0产生异常中断。

虚拟 8086模式标志 VM,指示 80386是否工作在模拟 8086模式。VM=1,CPU工作在虚拟 8086模式下;VM=0,CPU工作在非虚拟 8086模式下。

对界标志 AC在 80486中定义。

⑶ 段寄存器

80386段寄存器是在 80286基础上扩充的,除 CS、DS、DS和 SS外,又增加了 FS和 GS,这两个寄存器是支持当前数据段的。另外与段寄存器相关联的段高速缓存器扩充到 64位。支持 32位段基址和 32位段界限。

⑷ 系统表寄存器

80386系统表寄存器与 80286一样,只是 GDTR和 IDTR扩充到 48位,LDTR和 TR的关联段高速缓存器扩充到 64位。

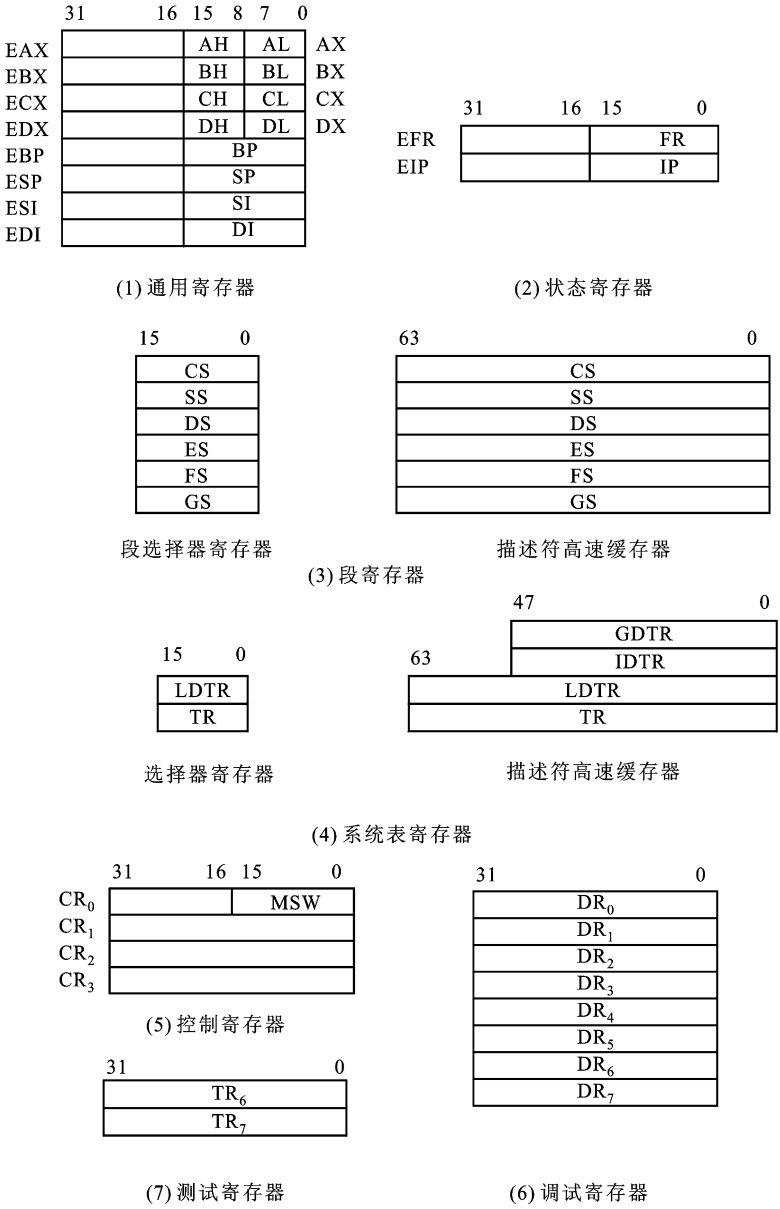


图 4-5-1 80386的寄存器组

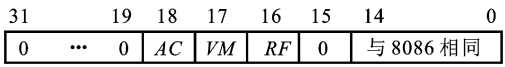


图 4-5-2 EFR格式

⑸ 控制寄存器

80386设置了 4个控制寄存器，CR_0、CR_1、CR_2 和 CR_3。

CR_0 中设置了 6个控制标志，PE、MP、EM 和 TS与 80286相同，另加分页允许标志 PG和处理器类型标志 ET。

PG＝1时，允许使用分页部件；PG＝0时，禁止使用分页部件。ET＝1时，系统中使用 80387协处理器；ET＝0时，系统中使用 80287协处理器。

CR_1 是保留后继使用的。

CR_2 是用于提供页故障线性地址的。

CR_3 是用于提供页目录基地址的。

⑹ 调试寄存器

80386设置 6个 32位的调试寄存器 DR_0、DR_1、DR_2、DR_3、DR_6 和 DR_7，用于程序员调试程序时设置断点。其中 DR_0、DR_1、DR_2 和 DR_3 用于指定 4个线性断点，DR_4、DR_5 保留，DR_6 为调试状态寄存器，用于指示断点的当前状态，DR_7 为调试控制寄存器，用于设置断点。

⑺ 测试寄存器

80386设置 2个 32位的测试寄存器 TR_6 和 TR_7。

4.5.2 80386的引脚功能

80386CPU芯片具有 132个引脚，采用 PGA (Pin Grid Array) 技术封装。引脚配置示意图如图 4－5－3所示。引脚信号功能图如图 4－5－4所示。各引脚的说明见表 4－6。

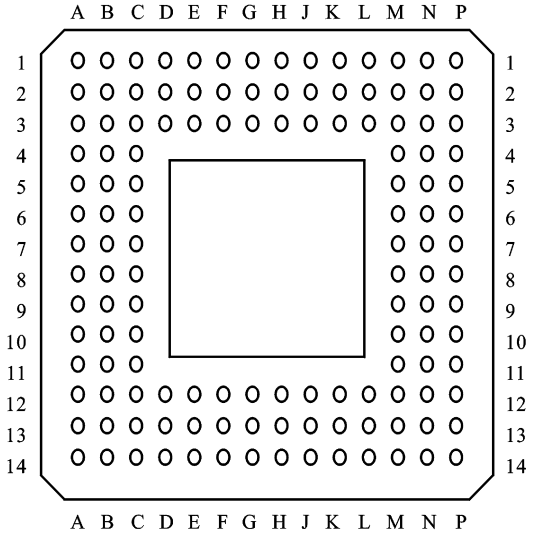


图 4－5－3 80386CPU引脚封装示意图

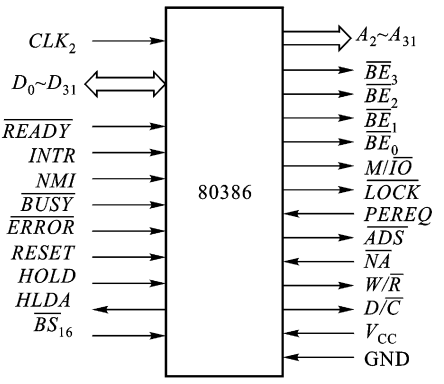


图 4－5－4 80386的引脚信号

表 4-6 80386引脚功能

引脚信号	输入/输出	功 能
CLK2	I	系统时钟输入线
D_{31} ~D_0	I/O	数据总线 (32位)
A_{31} ~A_2	O	地址总线,表示 4 B数据的有效位置
$\overline{BE_3}$ ~$\overline{BE_0}$	O	字节允许信号,表示 4B数据的有效字节
W/$\overline{R}$	O	读写信号,高电平为写,低电平为读
D/$\overline{C}$	O	数据/指令代码传送信号,高电平数据,低电平代码
M/$\overline{IO}$	O	存储器/IO访问选择信号
$\overline{ADS}$	O	地址选通信号
$\overline{LOCK}$	O	总线封锁信号,低电平有效
$\overline{READY}$	I	准备好信号,低电平有效
$\overline{NA}$	I	下一地址请求信号,即流水线方式请求信号
$\overline{BS_{16}}$	I	总线宽度控制信号,指定 16位数据总线
HOLD	I	总线请求信号,高电平有效
HLDA	O	总线响应信号,高电平有效
INTR	I	可屏蔽中断请求信号
NMI	I	非屏蔽中断请求信号
PEREQ	I	协处理器存储操作数请求信号,高电平有效
$\overline{BUSY}$	I	协处理器“忙”信号,低电平有效
$\overline{ERROR}$	I	协处理器“出错”信号,低电平有效
RESET	I	系统复位信号
GND	I	系统地
V_{CC}	I	系统电源

4.6 Pentium微处理器

Pentium CPU是 Intel公司的第五代产品。Pentium是采用 RISC技术的 CISC处理器。Pentium的特点如下。

① 数据总线 64位,地址总线 36位,可寻址的物理地址空间达 64 GB。

② 片内集成了 16 KB的高速缓存和浮点协处理器。

③ 采用超标量体系结构和多条流水线技术。

④ 增设了动态转移预测技术。

4.6.1 Pentium的结构

1. 超标量体系结构

Pentium微处理器具有3条指令执行流水线:两条独立的整数指令流水线,称为U流水线和V流水线,一条浮点指令流水线。两条独立的整数指令流水线都拥有独立的算术逻辑运算部件、地址生成逻辑和高速数据缓存接口。每个时钟周期可同时执行两条指令,这种能一次同时执行多条指令的处理器结构称为超标量体系结构。

Pentium微处理器的整数指令流水线也具有指令预取、指令译码、生成地址和取操作数、指令执行、写操作数5级。每一级处理需要一个时钟周期。

2. 浮点指令流水线与浮点指令部件

浮点指令流水线具有8级,实际上它是U流水线的扩充。U流水线的前4级用来准备一条浮点指令,浮点部件的后4级执行特定的浮点运算操作并报告执行错误。在浮点部件中,对常用浮点指令采用专用硬件来执行,不是采用微码执行。这样大幅度提高了浮点处理性能。

3. 指令转移预测部件

程序指令的执行大多数是顺序执行的,指令流水线利用这一特点形成流水线,提高了指令执行的吞吐量。但程序中也有转移执行的情况,转移指令会冲掉流水线已有的内容,重新装载指令流水线,这会降低流水线效率和指令执行速度。

Pentium处理器提供了一个小型的1KB高速缓存来预测指令转移。它用来记录正在执行的程序最近所发生的几次转移,预测部件将进入流水线的新指令与所存储的有关转移的信息进行比较,确定是否将再次执行转移。若是就产生一个目标地址,提前指出要发生的转移。如果预测正确,就立即执行程序转移,不需要计算下条指令地址,可防止指令流水线停顿。由于程序局部性原理,使得转移预测部件在大多数情况下预测正确,这就有效地提高了处理器的性能。

4. 数据和指令高速缓存

Pentium芯片内部有两个超高速缓冲存储器(Cache)。一个是8KB的数据Cache,另一个是8KB的指令Cache,它们可以并行操作。这种分离的Cache结构可减少预取指令和数据操作的冲突,提高处理器的信息存取速度。

4.6.2 Pentium的内部寄存器

Pentium CPU的寄存器组与80486基本相同,作了如下扩充。

EFR标志位寄存器增加了两位:VIF和VIP,用于控制Pentium虚拟8086方式扩充部分的虚拟中断。控制寄存器CR_0的CD和NW被重新定义,控制Pentium的片内高速缓存,并增加了CR_4控制寄存器对80486模式进行扩充。

此外还增加了几个模式专用寄存器,用于控制可测试性、执行跟踪、性能检测和机器检查错误等功能。

Pentium的内部寄存器都是32位的,但它的外部总线是64位的。64位数据总线并不直接与CPU寄存器相连。内部数据处理是32位的,内部总线为128位~256位,因此可以一次传送大量数据。

4.6.3 Pentium 的工作模式

Pentium与 80386/80486一样,可以在实模式、保护模式和虚拟 8086模式下工作。开机后自动进入实模式下工作。

⑴ Pentium实地址模式

Pentium实地址模式与 8086完全兼容,但实际寻址范围扩大为 0H～10FFEFH。

⑵ Pentium的保护模式

Pentium的保护模式与 80386/80486完全兼容。该方式可支持存储器管理、特权保护和多任务切换。

⑶ Pentium的虚拟 8086方式

Pentium的虚拟 8086方式是 80386/80486的虚拟 8086方式的扩展。主要使 EFR标志位寄存器增加了 VIF和 VIP位,用于控制 Pentium虚拟 8086方式扩充部分的虚拟中断。

习 题

1. 8086 CPU由哪两个单元构成,每个单元有哪些主要组成部分?
2. 8086 CPU的总线接口部件有哪些功能?
3. 8086 CPU执行部件有什么功能?
4. 8086标志寄存器设置了哪些标志,分为哪两类,各类标志的作用是什么?
5. 8086 CPU为什么要设置段寄存器,设置了哪些段寄存器?
6. 8086段寄存器 CS=1200H,指令指针寄存器 IP=0FF00H,此时指令的物理地址是多少?指向这一物理地址的 CS值和 IP值是唯一的吗?
7. 8086 CPU启动时有哪些特征,对 8086系统的启动程序应如何去寻找?
8. 8086的存储器空间最大为多少,怎样用 16位寄存器实现对 20位地址的寻址?
9. 简述 8086 CPU控制信号 $\overline{RD}$、$\overline{WR}$、M/$\overline{IO}$、ALE的功能。
10. 8086系统为什么需要地址锁存,需锁存哪些地址信息?
11. 8086是怎样解决地址总线和数据总线的分时复用问题的,ALE信号何时处于有效电平?
12. 简述时钟发生器 8284的功能。
13. 简述总线控制器 8288的作用。
14. 什么是时钟周期,什么是总线周期,什么是指令周期?
15. 8086的基本总线周期由几个时钟组成,如果 CPU的时钟频率为 8MHz,那么它的时钟周期为多少,一个基本总线周期为多少?
16. 在总线周期中,什么情况下需要插入等待状态,在什么地方插入?
17. 最小模式中总线读周期和总线写周期的区别是什么?
18. 画出 8086最小模式时的总线读周期时序。
19. 8086在最小模式下,总线保持过程是怎样产生和结束的?
20. 分别解释实地址模式、保护地址模式和虚拟 8086模式。
21. 80386的内部结构由哪些单元构成,寄存器组有哪几类寄存器?
22. 简述 Pentium微处理器的特点。

第 5章　存 储 系 统

存储系统是计算机中的重要组成部分,用来存放计算机工作时使用的信息:程序和数据。由于有了存储系统,计算机才有了记忆功能,存储系统的性能如何,直接影响着计算机的性能。

本章主要介绍微型机系统的主存储系统。

5.1　存储系统概述

5.1.1　存储器的分类

随着计算机系统结构和器件的发展,存储器的种类日益繁多,分类方法也有很多种。可按存储器的存储介质划分、按存取方式划分、按存储器在计算机中的作用划分等。

1. 按存储器在计算机中的作用和位置分类

按存储器在计算机中的作用和位置可分为主存储器(内存)、辅助存储器(外存)、缓冲存储器等。

(1) 主存储器(内存)

主存储器又叫内部存储器简称主存或内存,它是主机的组成部分,CPU可通过系统总线直接访问它,用来存放正在使用或经常使用的程序和数据。其特点是:可直接存取、容量小、速度快。其容量受地址线条数限制,如20位地址线可直接访问的主存空间最大为1MB(2^{20}B)。

微型机的主存储器又分为随机存取存储器、只读存储器等。

(2) 辅助存储器(外存)

辅助存储器又称为外部存储器,简称辅存或外存。它在主机外部,属于外部设备,CPU需通过I/O接口才能进行访问。用于存放不常使用,且需要长期保存的信息。需要时将其存储的信息传送到内存中方可使用。其特点是可长期保存数据、存储容量大、但速度慢。

常用的外存有软磁盘、硬磁盘、磁盘组、磁带、光盘等。

(3) 缓冲存储器

缓冲存储器设置在两个访问速度不同的存储部件之间,用以加快部件间的信息交换。例如在CPU和主存之间的高速缓存(Cache)等。

2. 按工作方式分类

按存储器的工作方式可分为可读/写存储器和只读存储器。

(1) 可读/写存储器

可读/写存储器是一种既可读出信息又可写入信息的存储器。如主存储器、磁盘和磁带等。

(2) 只读存储器

只读存储器(Read Only Memory, ROM)对于存储在其内部的信息只能读出使用,不能进行写入,即在使用中不能改变内部的内容。如半导体只读存储器、CD－ROM光盘等。

半导体只读存储器又可分为掩模型只读存储器（Read Only Memory, ROM）、可编程只读存储器（Programmable ROM, PROM）、可擦除可编程只读存储器（Erasable PROM, EPROM）、电可擦除可编程只读存储器（Electrically EPROM, E^2PROM）等。

3. 按存取方式分类

按存储器的存取方式，可以将存储器分为随机存取存储器、顺序存取存储器和直接存取存储器。

⑴ 随机存取存储器

随机存取存储器（Random Access Memory, RAM），可随机从任何位置进行信息的存取，其存取数据的时间与信息在存储器中的位置无关。如半导体随机存储器、磁芯存储器等。

⑵ 顺序存取存储器

顺序存取存储器（Sequential Access Memory, SAM）的存储内容，只能按某种顺序进行存取，即存取时间与存取单元的物理位置有关。当对存储单元进行访问时，往往等待的时间较长，如磁带等。

⑶ 直接存取存储器

直接存取存储器（Direct Access Memory, DAM）是不必经过顺序定位就可以直接定位存取位置的存储器。其特点是存取等待时间短，访问速度较快。如软盘、硬磁盘和光盘等。

4. 按存储介质分类

按存储信息的介质，可将存储器分为磁存储器、半导体存储器、光存储器等。

⑴ 磁存储器

磁存储器是采用磁性记录材料制造的存储器。如磁芯、磁带、磁盘等存储器。磁芯存储器是早期使用的存储器，现在很少见到。现在常用的是磁盘、磁带等磁表面存储器。

⑵ 半导体存储器

采用半导体器件和技术制造的存储器为半导体存储器。又可分为双极型和 MOS型两种。

双极型半导体随机存储器为电流驱动型存储器，速度快、集成度低、功耗大、价格高，常用于高速缓存等。MOS型半导体随机存储器是电压控制型存储器，具有集成度高、功耗低、价格低的特点，常用作为主存储器。

⑶ 光存储器

采用激光技术控制访问的存储器。如只读光盘（CD－ROM）、可读写光盘（MO）等。光盘的容量很大，使用广泛。

存储器的分类如图 5－1－1所示。

5.1.2 存储系统的层次结构

衡量存储器有 3个重要指标：容量、速度、价格/位。一般速度快的存储器，其价格就高，所以容量就不可能太大。为解决 3个指标之间的矛盾，实际中采用以下几个层次来构成存储系统。

1. 主存－辅存结构

早期，从辅存到主存的程序和数据由程序员自己安排，现在由专用软、硬件辅助设备来完成，即形成了存储系统。

如图 5－1－2所示为主存－辅存层次的存储系统。使用速度快、容量较小的半导体存储器

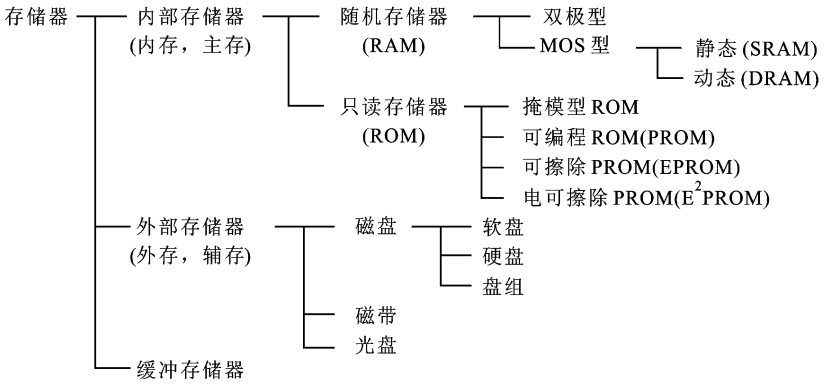


图 5-1-1 存储器分类

作为主存，而容量大、价格便宜的磁盘、磁带等作为辅存。把 CPU 当前运行的程序和数据放在主存中，暂时不运行的程序和数据存放在辅助存储器中。在运行时，不断将辅存的程序和数据装入主存储器中，处理过的信息不断存入辅存。这些管理调度工作由辅助软、硬件设备来完成。对 CPU 来讲，相当于存在一个这样的存储器：其速度与主存的速度相同，容量和辅存的容量相同，即虚拟存储系统。

计算机中基本都采用虚拟存储系统。

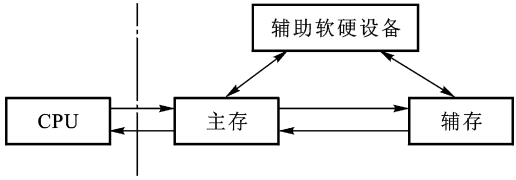


图 5-1-2 主—辅存储结构

2. Cache—主存层次

主存的速度虽然比辅存快得多，但与 CPU 相比还差大约一个数量级，影响 CPU 效率的发挥。为解决主存速度问题，设置了高速缓冲存储器——Cache，形成 Cache—主存层次，如图 5-1-3 所示。

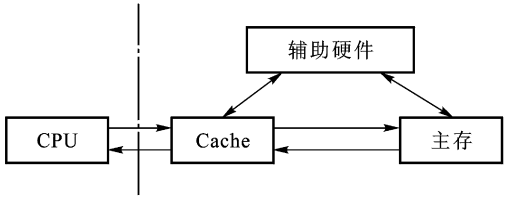


图 5-1-3 Cache—主存结构

Cache的速度与CPU的速度相近，Cache与主存之间的信息交换完全由硬件来完成。主存中要运行的程序，按一定的算法和规则，将要运行的一小部分预取入Cache中进行执行，这样可大幅度提高运行速度。

整体虚拟系统如图5－1－4所示。

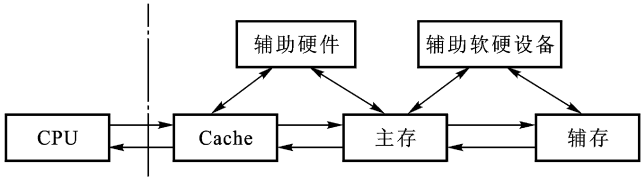


图5－1－4 Cache–主存—辅存结构

现代计算机大都采用Cache–主存—辅存三级存储层次结构的存储系统。

5.1.3 存储器的基本组成

主存储器一般采用半导体存储器，它由存储体、地址寄存器、地址译码器、读写驱动电路、数据寄存器以及时序控制电路等部件组成。如图5－1－5所示。

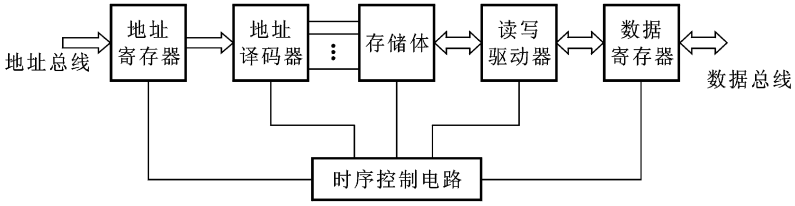


图5－1－5 主存储器基本组成

存储体是存储单元的集合，是存放信息的地方。地址总线用于寻找要存取的单元地址，数据线用于进行存储单元与CPU之间的数据传送。

当CPU要进行主存的访问时，通过地址总线把地址码送入地址寄存器锁存，再传给地址译码器进行地址译码，译码输出选定存储体相应的存储单元。当CPU发出读/写命令时，时序电路产生读/写操作控制信号。当对主存进行读操作时，将选中单元的数据读入数据驱动器，再通过数据总线送给数据寄存器供CPU读取使用。当对主存进行写操作时，CPU在发出地址码后，并将要写入的数据送数据寄存器。当选中单元后，将数据寄存器的数据通过数据总线送入选中的存储单元，完成写入操作。

5.2 半导体静态随机存储器（SRAM）

5.2.1 SRAM的工作原理

存储器存放信息，其实质是存放一位位的二进制码，每一位二进制码只有两个状态：0和1。任何有两个状态的器件都可作为存放二进制码的基本存储单元，SRAM（半导体静态存储器）采

用双稳态电路来作为存放 1位二进制码的基本存储电路。

如图 5－2－1所示为由 5个 MOS管组成的 RAM的基本存储电路，即 5管静态基本存储单元。

T_1、T_2 交叉耦合组成双稳态触发器，T_3、T_4 为负载管，T_5、T_6 为控制门，控制双稳触发器单元与外界的联系。T_7、T_8 为数据输入/输出的控制管，控制整个列的所有基本存储电路，不包括在 6管基本存储电路之内。

当 T_1 截止时，A点为高电平，该高电平接到 T_2 的栅极，使 T_2 饱和导通，于是使 B为低电平，B点接到 T_1 的栅极，所以，V_B ＝0又保证了 T_1 截止，这样就达到了一个稳态，与此类似 T_1 饱和，T_2 截止是另一个稳态。这个电路的两个稳态可存储二进制信息的“0”和“1”。

假设当 T_1 截止 T_2 饱和时，V_A ＝1，V_B ＝0表示存储“1”，当 T_1 饱和 T_2 截止时，V_A ＝0，V_B ＝1表示存储“0”。

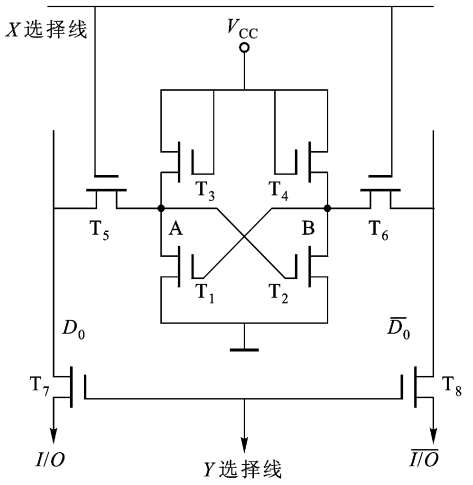


图 5－2－1 静态存储电路

下面介绍基本存储电路在读操作、写操作和保持信息时的工作原理。

1. 写操作

写入信息时，给 X选择线提供高电平，使控制管 T_5、T_6 导通；再给 Y选择线提供高电平，使 T_7、T_8 导通，这样 A、B点与 I/O线接通，要写入的信息从 I/O线引入。

写 1时：I/O＝1，$\overline{I/O}$＝0，使 A点为高电平，B点为低电平，T_1 截止 T_2 饱和，即写入 1。

写 0时：I/O＝0，$\overline{I/O}$＝1，使 B点为高电平，A点为低电平，T_2 截止 T_1 饱和，即写入 0。

2. 保存信息

当撤销选择信号后，所写入的信息便保持在基本存储电路中，即进入保存信息状态。如图 5－2－1所示，当 X、Y选择线信号撤销后，T_5、T_6、T_7、T_8 都不通，触发器与外界断开，触发器的稳态保持不变，写入的信息保存在存储电路中。

3. 读出操作

读出时，使 X选择线为高电平，T_5、T_6 导通，再使 Y选择线为高电平，T_7、T_8 导通，从 I/O线读出 A点的电平，$\overline{I/O}$线读出 B点的电平，即为读出信息。

因 6管电路使用管子多，所以位容量低，耗电量大，但不需要动态刷新，外围电路也较简单。

5.2.2 SRAM 结构

在存储器中将 8个二进制位称为一个字节，2个字节为一个字，内存中以一个字节为一个单元。为区分不同的存储单元，需对它们分别进行编号，单元的编号称为单元的存储地址，每个存储单元对应一个存储地址。将上述存放 1位二进制信息的基本存储电路，组织在一起构成能存放大量信息的存储器即构成存储器的结构。

一般的存储器结构如图 5－2－2所示。图中假设为 1K ×1位容量的存储器。

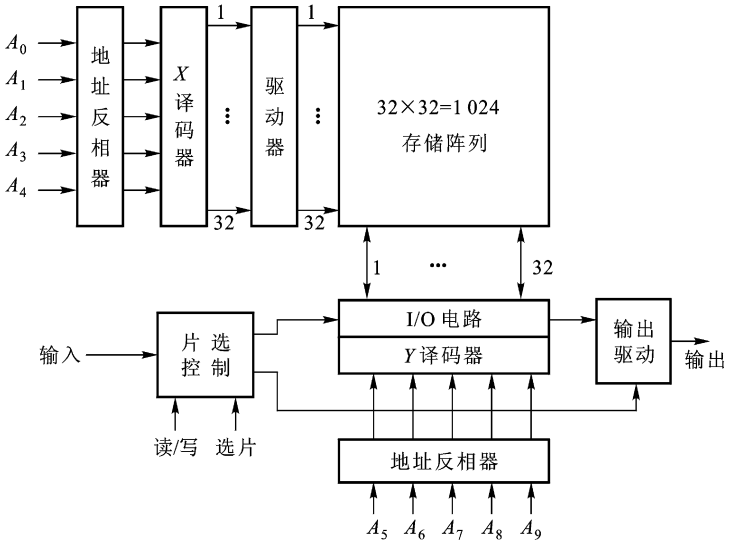


图 5－2－2 静态存储器框图

1. 存储体

一个基本存储电路表示 1位二进制位,而一个静态存储芯片一般为几千字节到几十千字节,要构成一个存储器需要大量的存储电路。这些存储电路有规则地组合起来,就构成存储体,又称存储阵列。在存储器中,通常将各个字的同一位组织在一个芯片中,如图 5－2－2中为 1 024×1位的,它是 1 024个字的同一位。由这样的 8个芯片可构成 1 024B的存储器。1 024个存储电路通常排成矩阵的形式,图 5－2－2为 32×32＝1 024的排列形式。由 X选择线 (行选择线) 和 Y选择线 (列选择线) 译码交叉来选择所需单元。

2. 外围电路

外围电路通常由地址译码器、I/O电路、片选控制、输出驱动电路等组成。

(1) 地址译码器

存储单元是按地址来选择的,CPU要选择某一单元就要在地址总线上输出此单元的地址信号给存储器,存储器对地址信号进行译码,译码输出来选择需要访问的单元。对地址信号的译码是通过地址译码器实现的。另外,在此结构中,X方向的负载大,译码输出需经过驱动器。

(2) I/O电路

它处于数据总线和被选中的单元之间,用以控制被选中的单元读出或写入,并具有放大信息的作用。

(3) 片选控制

一个存储器系统往往需要由一定数量的存储芯片构成。在进行访问时,首先要选择哪些芯片工作,这就需要进行芯片选择控制。一般用高位地址译码输出和一些控制信号形成片选信号,只有片选信号有效时,才能对所连芯片的存储单元进行读写。

⑷ 输出缓冲器（输出驱动）

输出缓冲器对从存储体中读出的数据进行缓冲，通过输出缓冲器将芯片内部数据线与外部双向数据总线连接。

3. 地址译码方式

在存储器中，通过地址译码器对地址总线的地址进行译码，来选择要访问的存储单元。地址译码通常有两种方式：单译码方式（或称字结构）和双译码方式（或称复合译码结构）。

⑴ 单译码方式

在单译码结构中，将每个字的所有位排成一行，只有行方向的译码器，译码输出的每条字选线选择某个字的所有位，如图 5－2－3所示。

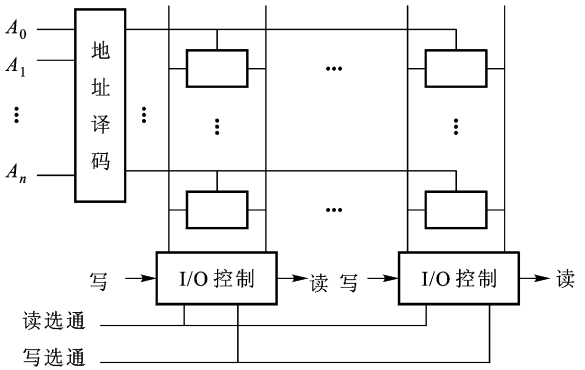


图 5－2－3 单译码结构存储器

该译码方式中每一行对应一个字，每一列对应每个字的其中一位。有多少字，就要有多少条字选择线，所以使用的选择线太多，只能用于容量小的存储器中。在此结构中，当地址译码使某个字的选择线有效时，该字的所有位同时被选中，一块进行输入或输出操作。

数据线通过读写控制电路与数据输入端或数据输出端相连，根据读写控制信号对被选中的单元进行读出或写入。

⑵ 双译码方式

双译码方式中，地址译码分成两部分：行译码（X译码）和列译码（Y译码），如图 5－2－4所示。

该译码结构中，当 X地址译码输出选中某一行时，并不是该行所有位都进行输入/输出，而是再由列地址译码输出选中被选中行的某一位，使行、列交叉点的一位进行输入/输出操作。

采用 X、Y交叉译码选择，可大幅度减少选择线，这样可减少译码器的译码输出，简化译码器的结构。图 5－2－4中假设为 1 024个存储电路，排成 32行 32列，需 32条行选择线和 32条列选择线，各需 5条地址线进行译码即可。若采用单译码结构，则需 1 024条选择线，而双译码时，只需 32＋32＝64条选择线，所以大幅度减少了选择线的条数。当位容量很大时，该问题更加明显。

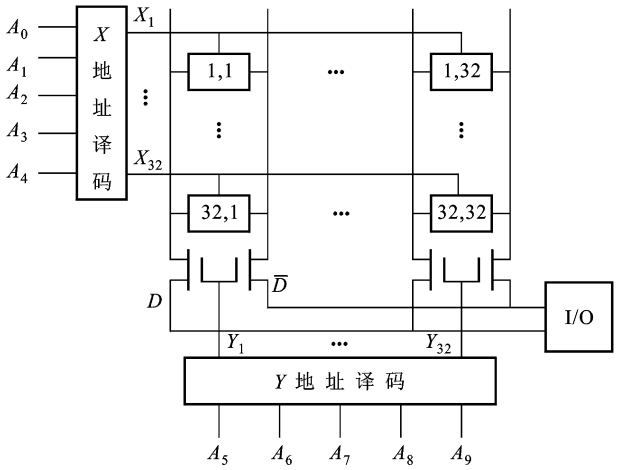


图 5-2-4 双译码结构存储器

5.2.3 SRAM 实例

静态 RAM存储芯片主要有 2114 (1K ×4位)、4118 (1K ×8位)、6116 (2K ×8位)、6264 (8K ×8位)、62256 (32K ×8位)等几种。下面以静态存储芯片 Intel2114为例,说明存储器的构成和工作原理。

Intel2114是 1K ×4位的静态 RAM芯片,采用 18引脚双列直插式封装,其逻辑引脚图如图 5-2-5所示。

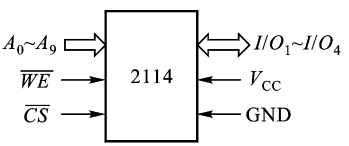


图 5-2-5 2114引脚逻辑图

引脚功能如下:

A_0 ~A_9:地址线,10条,寻址 2^{10} =1K。

$\overline{WE}$:写允许。低电平时,控制写操作;高电平时,控制读操作。

$\overline{CS}$:芯片选择。只有该信号有效 (低电平)时,芯片才被选中工作。

I/O_1 ~I/O_4:数据输入输出线,4条。芯片每次被选中可同时读写 4位数据。

Intel2114的内部结构框图如图 5-2-6所示。

要排成 64×64的阵列共需 4 096个 1K×4位基本存储电路。因 1K容量需要 10条地址线进行译码选择,其中 6条地址线进行行译码,输出 64条行选择线,进行 64行的选择;4条地址线进行列译码,输出 16条列选择线,每条列选择线选择 4列,16条列选择线共选 64列,这样每次选择 4位进行输入输出。

存储器内部数据通过 I/O电路以及输入输出三态门与数据总线相连。由选片信号$\overline{CS}$和写允许信号$\overline{WE}$一起控制这些三态门。只有$\overline{CS}$有效 (低电平)时,才能进行数据的读写,当$\overline{WE}$有效 (低电平)时,使输入三态门导通,数据由数据总线写入存储器;当$\overline{WE}$无效 (高电平)时,使输出三态门打开,数据由存储器读出送至数据总线。

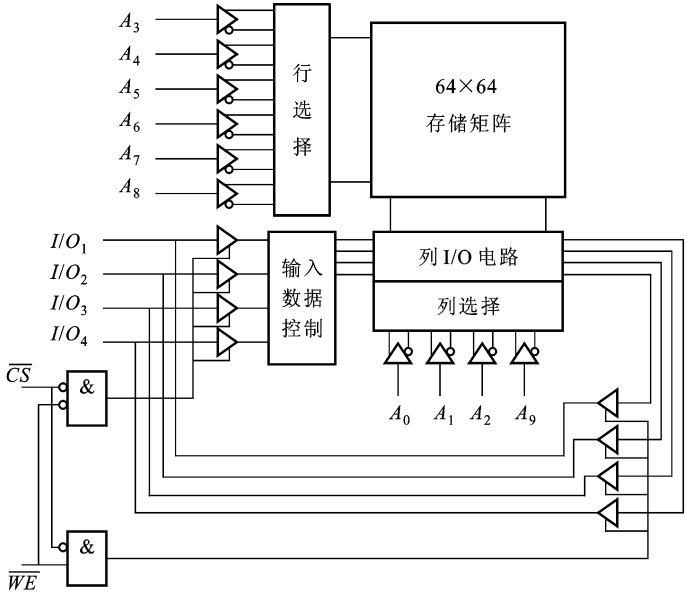


图 5－2－6 2114的结构框图

5.3 半导体动态随机存储器（DRAM）

5.3.1 DRAM 的工作原理

半导体动态随机存储器利用 MOS管的栅极电容存储电荷的原理来存储信息。因电容的充电、放电、泄漏、补充是一个动态的过程，所以称为动态随机存储器（DRAM）。由于电容有泄漏，必须不断地补充电荷，这种补充电荷的过程即为动态 RAM的刷新。

一般动态存储器要求 2ms之内刷新一次。

下面先介绍动态基本存储电路的工作原理。

1. 四管动态基本存储电路

图 5－2－1所示六管静态单元中，依靠 T_1、T_2 栅极上的电荷来存储信息，V_{CC} 通过 T_3、T_4 往 T_1、T_2 补充电荷，使电荷保持不变，实际上由于 MOS的栅极是绝缘的，电荷泄放很小，所以即使去掉 T_3、T_4 后，T_1、T_2 的栅极电荷也能维持一定的时间，这就构成了四管动态单元。

如图 5－3－1所示即为四管动态基本存储电路。

从图中可以看出，其电路结构与六管电路相比，只是去掉了 T_3、T_4，存储原理也类似六管电路。

⑴ 写入操作

给 X选择线加高电平，T_5、T_6 导通，使 A、B点与 I/O线接通，要写入的信息从 I/O线引入：

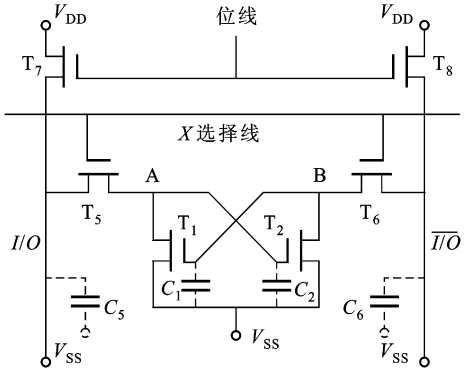


图 5-3-1 四管动态单元

写 1时,使 I/O=1,$\overline{I/O}$=0,则 A点为高电平,B点为低电平,T_1 截止 T_2 饱和,即写入 1。

写 0时 :使 I/O=0,$\overline{I/O}$=1,则 B点为高电平,A点为低电平,T_2 截止 T_1 饱和,即写入 0。

⑵ 保存信息

当选择信号撤销后,T_5、T_6、不通,触发器与外界断开,靠 T_1、T_2 的栅极电容 C_1、C_2 的存储电荷作用保持信息,即将写入的信息保存在存储电路中。

⑶ 读出操作

位线为高电平,使 T_7、T_8 导通,分布电容 C_5、C_6 预充电到 V_{DD},当 X选择线为高电平时,T_5、T_6 导通。若原存 "1",即 T_1 截止、T_2 饱和,则 C_6 通过 T_6、C_1 放电,同时 C_5 通过 T_5 往 C_2 上充电,使 I/O=1,$\overline{I/O}$=0,即读出 "1"。若原存 "0"则 C_5 通过 T_5、C_2 放电,同时 C_6 通过 T_6 往 C_1 上充电,I/O=0,$\overline{I/O}$=1,即读出 "0"。

可见,读出的同时即为原信息的补充过程,所以读出过程即为动态 RAM的刷新过程。

2. 三管动态基本存储电路

一个基本存储电路所用的管子越少,芯片的位密度越高,所以出现了三管和单管动态存储电路。

如图 5-3-2所示,在此电路中,电路将两个耦合管变成一个即构成三管电路,且读写选择线分开,读写数据线分开。

⑴ 写入操作

写入时,写选线 =1使 T_1 导通,写数据通过 T_1 送到 T_2 的栅极。

⑵ 读出操作

给出预充电信号,T_4 导通,线分布电容 C_D 充电至 V_{DD},使读选线 =1,T_3 导通,若原存 "1",则 T_2 导通,C_D 通过 T_3、T_2 放电,读得 "0";若原存 "0",则 T_2 截止,C_D 上得电压不变,读得 "1"。读得的信息和原来存入的信

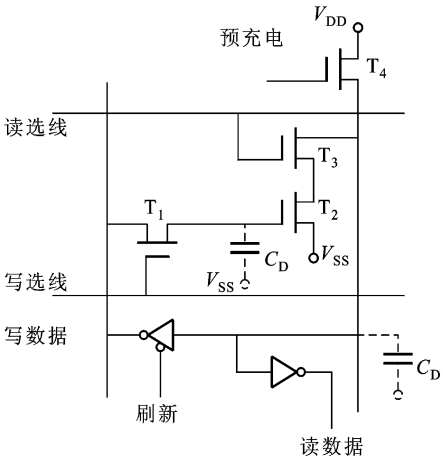


图 5-3-2 三管动态单元

息相反,输出时应反相。

刷新:读出的信息,反相后再写入,即可达到刷新的目的。

3. 单管动态基本存储电路

单管动态基本存储电路只有1个管子和1个电容,如图5-3-3所示为单管存储电路。在该单管电路中,存放信息是"1"还是"0",决定于电容中有没有电荷。

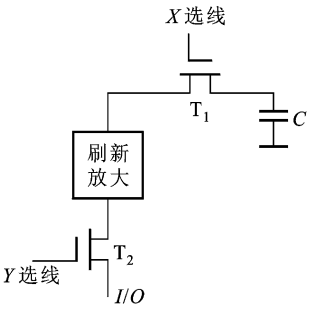


图5-3-3 单管动态单元

(1) 写入操作

X、Y选线为高电平,T_1、T_2导通,I/O线上的信息通过T_1、T_2存到C上。

(2) 读出操作

X、Y选线为高电平,T_1、T_2导通,C上的信息通过T_1、T_2并放大输出。

(3) 刷新

读出时原电容上的电荷被泄放掉,破坏了原信息,所以刷新放大在读出后立即重新写入,实现刷新。

对上述3种动态存储电路来说,四管单元读出过程就是刷新过程,三管和单管单元需另加刷新电路。

5.3.2 DRAM实例

动态存储芯片主要有2164(64K×1位)、4164(64K×1位)等几种。下面以动态存储芯片Intel2164为例,说明动态存储器芯片的构成和工作原理。

Intel2164是64K×1的动态RAM芯片,即片内共有65536个地址单元,每个地址单元一位数据。用8片Intel2164就可构成64KB的存储器。片内要寻址64K,则需16条地址线,为减少封装引脚,地址线分为两部分,行地址和列地址,分时送入存储器,所以只有8条地址线。

如图5-3-4所示为Intel2164引脚逻辑图。

A_0~A_7:地址输入线。

$\overline{CAS}$:列地址选通信号。

$\overline{RAS}$:行地址选通信号。

$\overline{WE}$:写信号。

D_{IN}、D_{OUT}:数据输入/输出线。

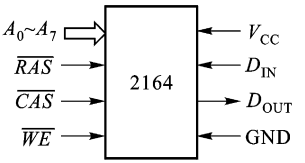


图5-3-4 Intel2164引脚逻辑图

芯片内部设有地址锁存器,利用多路开关,由行地址选通信号(Row Address Strobe)$\overline{RAS}$先把8位行地址送入芯片内的行锁存器,再由列地址选通信号(Column Address Strobe)$\overline{CAS}$把列地址送入列地址锁存器,该8条地址线也用于刷新地址输入。所以连接时,需要通过多路开关,分时将行列地址送入地址输入线。

图5-3-5为Intel2164的内部结构示意图。

图5-3-5所示64K存储体由4个128×128的存储矩阵构成。每个128×128的存储矩阵由7条行地址线和7条列地址线进行选择。7条行地址线经过译码产生128条选择线,分别选

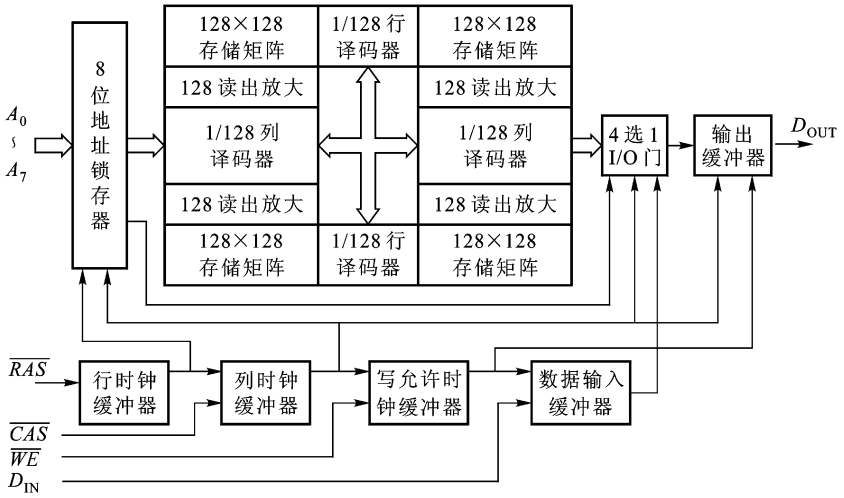


图 5-3-5 Intel2164内部结构框图

择128行；7条列地址线经过译码产生128条选择线，分别选择128列。

锁存在行地址锁存器的7位行地址 RA_0～RA_6 同时加到4个存储矩阵上，在每个矩阵中都选中一行，则共有512个存储电路被选中，它们存放的信息被选通至512个读出放大器，进行锁存和重写，从而实现刷新。

锁存在列地址锁存器的7位列地址 CA_0～CA_6 在每个矩阵中都选中一列，则共有4个存储电路被选中，最后经过1/4 I/O电路（由 RA_7 和 CA_7 控制）选中一个单元，进行读写。

芯片的输入/输出数据线是分开的。

只设一个写控制信号，当 $\overline{WE}=0$ 时为写入，$\overline{WE}=1$ 时为读出。没有单设片选信号，可用 $\overline{RAS}$ 兼作片选信号。

5.4 只读存储器（ROM）

只读存储器（Read Only Memory，ROM）：一旦写入信息，就不能轻易改变，所存信息在掉电时也不会丢失，正常使用时，只能读出。一般用于存放固定不变的程序和数据。

特点：结构简单，位密度高，非易失性，可靠性高。

只读存储器可分为如下4类：

⑴ 掩模型ROM

信息在芯片制造时由厂家写入，用户无法进行任何修改。

⑵ 可编程ROM（PROM）

用户采用专用编程设备，可以一次性写入信息，一旦写入，就不能再进行修改。

⑶ 可擦除可编程ROM（EPROM）

可用特殊手段对已存内容进行擦除，然后再重新进行写入，即可进行多次写入操作。

⑷ 电可擦除可编程只读存储器（E^2PROM）

可用电方式进行在线擦除。

5.4.1 掩模型只读存储器

掩模型 ROM中的信息是厂家根据用户给定的程序或数据对芯片图形（掩模）进行两次光刻而写入的。

这种类型存储器的基本存储电路可由二极管、晶体管、MOS管构成。如图 5－4－1所示。X选择线与 I/O线之间有存储管 T则存“0”,无存储管 T则存“1”或反之。

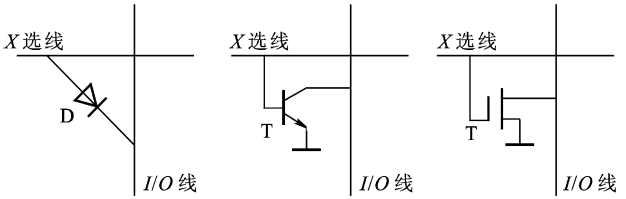


图 5－4－1 掩模型 ROM基本存储电路

一种典型的复合译码的 MOS型 ROM结构如图 5－4－2所示。

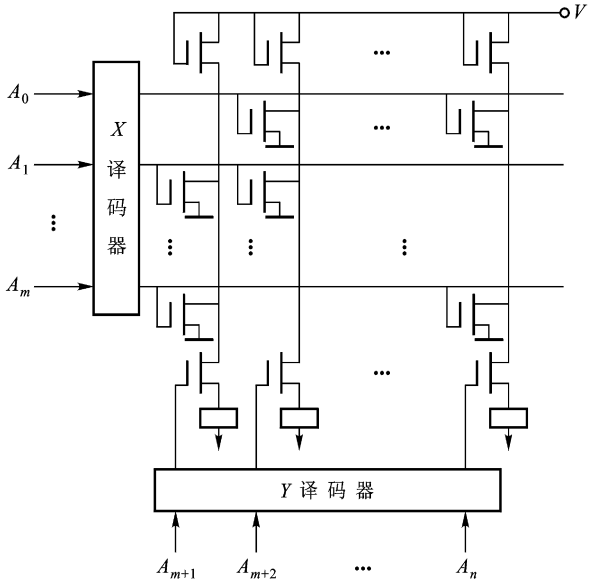


图 5－4－2 复合译码的 MOS型 ROM结构

假设图中为一 1 024×1的 ROM,则需 10条地址线,分两组分别经 X、Y译码器译码,各产生 32条选择线。X译码输出选中某一行,但这一行中哪一个输出与 I/O电路相连,还取决于列译码输出,列译码输出选中某列的控制管,将行列交叉处的单元数据输出,故每次只选中 1位。8

个这样的电路,将它们的地址并联,则可得到8位信号输出。

5.4.2 可编程只读存储器(PROM)

可编程ROM(Programmable ROM,PROM)一般由二极管矩阵组成,也可由MOS管或晶体管矩阵组成。如图5-4-3所示为存储电路结构。

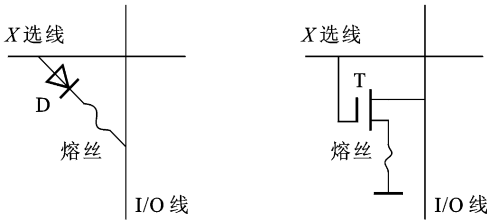


图5-4-3 PROM的基本存储电路

出厂时芯片的每个单元均由二极管或MOS管通过熔丝连通。用户写入信息时,由特殊电路将存放"0"的单元通以大电流,使熔丝熔断,存"1"的单元保持通态或相反。这样就实现了用户一次性编程。因熔丝熔断后,再不能恢复,所以信息写入后不能再更改。

5.4.3 可擦除可编程只读存储器(EPROM)

1. 基本存储电路

可擦除可编程ROM(Erasable PROM,EPROM)是一种可多次擦除和重复写入的ROM。

采用浮栅极场效晶体管代替前面讲过的PROM存储单元中的熔丝,就形成了EPROM的基本存储单元。如图5-4-4所示。

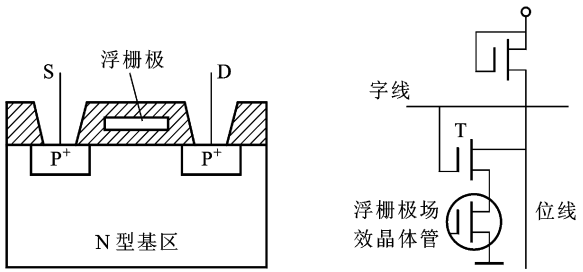


图5-4-4 EPROM的基本存储电路

浮栅极场效晶体管与普通的P沟道增强型MOS电路相似,在N型的基片上产生两个高浓度的P型区,分别引出源极(S)和漏极(D),在S和D之间有一个由多晶硅做的栅极,但它是浮空的,被绝缘层SiO_2所包围。出厂时,浮栅上无电荷,D、S间无导电沟道,处于不通状态,这时存储矩阵为全"1"。要写入信息时,在D、S间加高电压,另外加上编程脉冲,写"0"的单元被选通后,在D、S的电场作用下,使D、S间瞬间击穿,通过绝缘层将电子注入到浮栅上,形成栅极电场,在

该电场的作用下，使 D、S间形成导电沟道，D、S间导通，即存储“0”，写“1”的单元不变，这样即将信息写入。

写入电压撤销后，由于浮栅无泄放回路，电荷保持不变，即保持信息。

当要擦除时，用紫外光照射，所有电路中的栅极电荷形成光电流泄放，使电路恢复初始状态，实现了擦除，可重新再写入。

2. EPROM实例

EPROM芯片有 Intel 2716（2K×8位）、2732（4K×8位）、2764（8K×8位）、27128（16K×8位）、27256（32K×8位）、27512（64K×8位）等。

如2764为 8K×8（64K）位的 EPROM芯片。图 5－4－5为其引脚逻辑图。

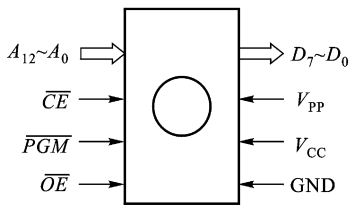


图 5－4－5 Intel 2764引脚逻辑图

A_{12}～A_0：地址输入线，13条地址线可寻址 8 K。

D_7～D_0：数据线，8位。

$\overline{CE}$：片选使能，低电平有效。

$\overline{PGM}$：编程脉冲。

$\overline{OE}$：输出允许，低电平有效。

V_{PP}、V_{CC}：电压输入。

该芯片有 4种工作方式：读方式、编程方式、检验方式和备用方式。如表 5－1所示。

表 5－1 Intel 2764的工作方式

	V_{CC}	V_{PP}	$\overline{CE}$	$\overline{PGM}$	D_7～D_0
读方式	+5V	+5V	0	0	输出
编程方式	+5V	+25V	1	正脉冲	输入
检验方式	+5V	+25V	0	0	输出
备用方式	+5V	+5V	—	1	高阻

⑴ 读方式

$V_{CC}=V_{PP}=+5V$，地址输入端用来输入存储单元地址，$\overline{CE}$、$\overline{PGM}$有效（低电平）时，数据线上便出现所寻址单元的数据，CPU可取走该数据。注意，$\overline{CE}$必须在地址稳定以后有效，才能保证读的是所选单元的数据。如图 5－4－6所示为读出方式的时序图。

⑵ 编程方式

在编程方式下，$V_{CC}=+5V$、$V_{PP}=+25V$，$\overline{CE}$无效，地址线上输入要编程的单元地址，在数据线上输入数据，$\overline{PGM}$加 +5V的编程脉冲，便可实现编程。必须在地址和数据稳定后，才能加编程

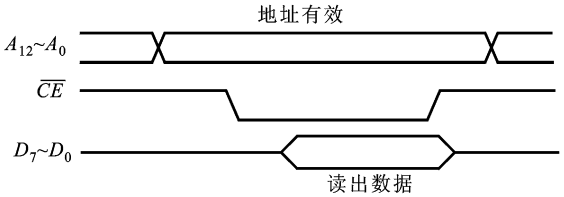


图 5-4-6 2764读方式时序

脉冲。如图5-4-7所示编程时序。注意在编程方式下，V_{PP}要加 +25V 的编程高电压。

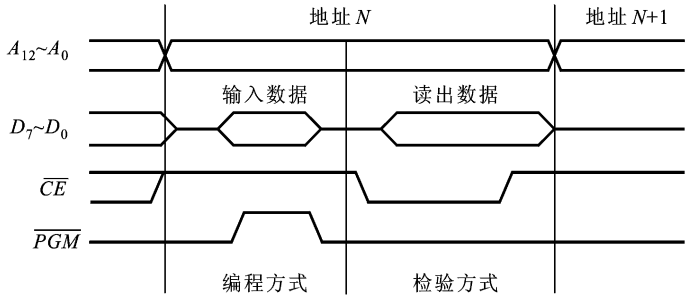


图 5-4-7 2764编程和检验时序

⑶ 检验方式

检验方式总是与编程方式配合使用的，在每次写入 1个数据后，紧接着读出写入的数据，进行检查，看写入的是否正确。如图 5-4-7所示，V_{CC}和 V_{PP}与编程方式一样，$\overline{CE}$有效，$\overline{PGM}$为低电平即可对刚编程的单元进行检验。

⑷ 备用方式

只要使$\overline{PGM}$=1则 EPROM 工作在备用方式，此时芯片功耗下降为读方式的 25%，该方式下数据输出为高组态。使用时，将$\overline{PGM}$与$\overline{CE}$接在一起，只要不读，则$\overline{PGM}$=1即进入备用方式，可大大减少功耗。

不同的 EPROM芯片，信号要求不同，如 Intel27128有 8种工作方式。这里不在一一介绍。

5.4.4 电可擦除可编程只读存储器（E^2PROM）

尽管可以对 EPROM擦除后重新编程，但擦除时需要紫外线光源，使用起来仍然不太方便。近年来出现了一种可用电擦除的可编程 ROM（Electrically Erasable Programmable ROM，E^2PROM）。E^2PROM基本存储单元是采用两极浮栅，形成一个隧道二极管，可在第二栅极与漏极之间电压 V_G 的作用下使电荷流向第一栅极，起编程作用，也可反向加 V_G 使电荷从浮栅极上泄漏，起擦除作用。

E^2PROM通常有 4种工作方式：读方式、写方式、字节擦除方式和整体擦除方式。表 5-2列出了 E^2PROM芯片 Intel2815各种工作方式下的信号电平。

从表中可以看出,根据$\overline{CE}$、$\overline{OE}$和 V_{PP}的不同,就可以选择其中某种工作方式。

表 5-2 Intel2815的工作方式

工作方式	V_{CC}	V_{PP}	$\overline{CE}$	$\overline{OE}$	$D_7 \sim D_0$
读方式	+5V	+5V	0	0	输出
写方式	+5V	+21V	1	1	输入
字节擦除方式	+5V	+21V	0	1	高阻
整体擦除方式	+5V	+21V	0	+9V～15V	高阻

读方式是 E^2PROM最经常使用的工作方式,此时地址输入端为所要读取的存储单元的地址,$\overline{CE}$、$\overline{OE}$均为低电平,V_{PP}加 4V～6V电压,输出端便会出现读到的数据。

写方式时,从地址输入端加上所要写入单元的地址,数据输入端为要写入的数据,$\overline{CE}$、$\overline{OE}$均为高电平,V_{PP}加 21V电压,即可实现写入。

在字节擦除方式下,由地址输入端输入要擦除的字节单元地址,$\overline{CE}$为低电平,$\overline{OE}$为高电平,V_{PP}加 21V电压,数据输入端加高电平,便可对 E^2PROM进行字节擦除操作。

整体擦除方式可以使整个 E^2PROM 回到初始状态。整体擦除方式下,$\overline{CE}$为低电平,$\overline{OE}$加 +9V～+15V高电平,V_{PP}加 21V电压,数据输入端加高电平,便可对整个 E^2PROM进行擦除。

5.4.5 闪速存储器 (Flash Memory)

闪速存储器 (Flash Memory),简称 Flash存储器或闪存,它是一种新型半导体存储器,是 1983年由 Inter公司首先推出的, 1988年开始商品化使用。就其本质而言 Flash存储器属于 E^2PROM类型,在不加电的情况下,能长期保存所存信息。 Flash存储器之所以被称为闪速存储器,是因为可快速进行电擦除,它能通过公共源极或公共衬底加高压实现擦除整个存储矩阵或部分存储矩阵,擦除整个存储矩阵的时间,与擦除 E^2PROM一个地址单元的时间相同。

Flash存储器既有掩模型 ROM和 RAM那样结构简单的性能,又有与掩模型 ROM和 DRAM一样的高密度、成本低、体积小特性。它是目前唯一具有大容量、非易失性、低价格、可在线改写和较高速度几个特性共存的存储器。它可擦写几十万次,但同 DRAM比较,Flash存储器的可擦写次数还是有限,且速度较慢。它是一种理想的文件存储介质,特别适用于在线编程的大容量、高密度存储领域。

由于 Flash存储器独特的优点,一些较新的主板上采用了 Flash ROM BIOS,使得 BIOS升级非常方便,Pentium微型机中已经把 BIOS系统驻留在 Flash存储器中。由于 Flash存储器集成度不断提高,价格不断降低,使其在便携机上取代小容量硬盘成为可能。

5.5 存储器与 CPU的连接

5.5.1 存储器与 CPU连接中要考虑的问题

(1) CPU的总线负载能力

CPU输出线的直流负载能力一般为带一个 TTL负载。现在存储器大部分为 MOS电路,直

流负载很小,故在小型系统中,CPU可以直接与存储器连接,而较大系统中,就要考虑CPU能否带得动,必要时就要加上总线驱动器或缓冲器。

⑵ CPU的时序和存储器存储时序之间的配合

因CPU的总线操作有固定的时序,应由此来确定对存储器存取速度的要求,选择与总线时序相适应的存储器芯片构成存储器系统,或在存储器已经确定的情况下,考虑其存取速度是否符合CPU时序的要求,如果速度慢,需要通过电路实现加入等待周期,以便进行可靠的读写。

⑶ 存储器地址分配和片选控制

确定所构成的存储器占整个存储空间的哪一部分,如何分配RAM区和ROM区,系统区、用户区、数据区、程序区等。另外,目前生产的存储器单片容量仍然有限,要用多片才能组成一个存储器,这就要解决如何产生片选信号的问题。

⑷ 控制信号的连接

CPU在与存储器交换信息时,有以下几个控制信号(以8086系统最小模式为例):M/$\overline{IO}$、$\overline{RD}$、$\overline{WR}$等。这些信号如何与存储器要求的控制信号连接,以实现所需的控制作用,也是连接中要解决的问题。

5.5.2 RAM与CPU的连接

下面通过用一定容量的存储芯片构成较大容量的存储器来介绍RAM与CPU的连接。

⑴ 用1K×1位的静态RAM芯片组成4K×8位的RAM

接线方框图如图5-5-1所示。在用多片存储芯片构成较大容量存储器时,采用位并联(位扩展)和地址串联(字扩展)的办法进行连接。如图5-5-1中使用1K×1位的存储芯片构成4K×8位的存储器。因为是1K×1位的芯片,所以每片即为1K字的同一位,只有一位数据I/O线连到某一位数据线上,需8片构成1组,连接到8条数据线上,即位扩展,每组的地址线接法相同。这样每组的容量为1K×8位。构成4K×8位需4组,即字扩展。由此可见共需存储芯片32片。

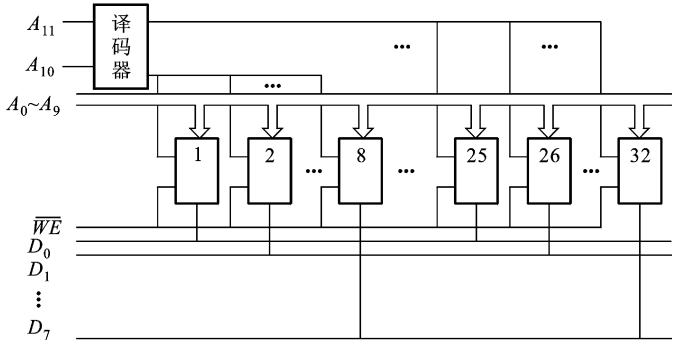


图5-5-1 用1 024×1的存储芯片组成4KB RAM框图

因每个芯片为1K容量,所以片内寻址需10条地址线(A_0~A_9),寻址片内的1K单元。4组芯片共需4条片选择线,分别选择各组进行工作。由A_{10}、A_{11}进行译码,输出4条片选线选择各

组。每次工作时,各组分别选通,所以各组的地址空间不同,即地址是串联的。

(2) 用 2114构成 4K×8的 RAM

2114为 1K×4位的静态 RAM。同理在此构成中每组需 2片,因为每片 4位,所以两片构成 8位。共需 4组,每组 1K容量,4组构成 4K容量,则共需 8片 2114。片内寻址需 10条地址线,直接接至 CPU的 A_0~A_9。要区分每一组,就要利用片选信号,片选译码需 2条地址线通过译码产生 4条选择线控制片选端。

如图 5-5-2所示构成框图。

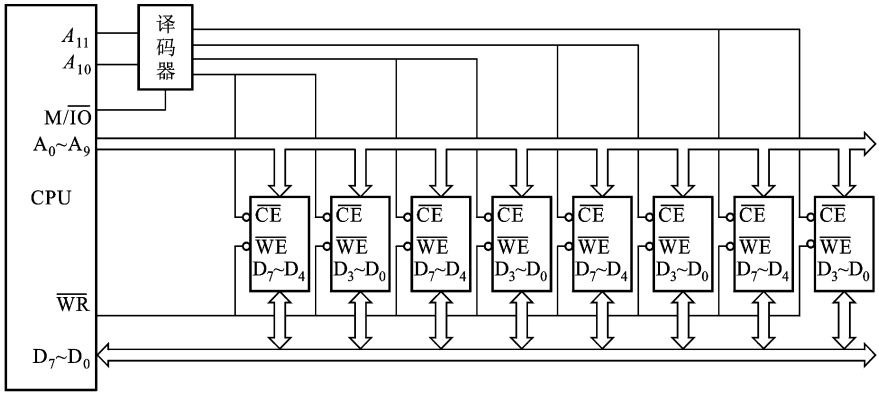


图 5-5-2 用 2114构成 4KB RAM方框图

可见用多个存储芯片构成较大容量存储器时,一般要分成若干组,每组有多个芯片。具体考虑方法如下:

设用 k×j位的存储芯片,构成 m×n位的存储器,则分组数 =m/k,称为字方向扩展。每组芯片数 =n/j称为位方向扩展。需用总芯片数 =分组数 ×每组芯片数。

[例 5.1] 用 2K×4位的存储芯片,构成 16K×8位的存储器。

则分组数 =16K/2K=8组,每组芯片数 =8/4=2,需用总芯片数 =8×2=16。

[例 5.2] 用 16K×1位的存储芯片,构成 64K×8位的存储器。

则分组数 =64K/16K=4组,每组芯片数 =8/1=8,需用总芯片数 =4×8=32。

5.5.3 地址空间分配与片选译码

1. 地址空间分配

如上所述,用多组芯片构成存储器时,各组分别由不同的地址译码信号进行选择,所以各组所占的地址空间不同。如图 5-5-2构成的 4K×8的 RAM,其地址空间为 0000H~0FFFH。共 4组,每组的地址空间不同,各占 1K地址空间。具体空间分配由片选信号决定,图 5-5-2中各组地址空间分配如下:

	A_{11}	A_{10}	A_9	…	A_0	
第一组:	0	0	0	…	0	0000H
	0	0	1	…	1	03FFH

第二组：　0　1　0 … 0　0400H
　　0　1　1 … 1　07FFH
第三组：　1　0　0 … 0　0800H
　　1　0　1 … 1　0BFFH
第四组：　1　1　0 … 0　0C00H
　　1　1　1 … 1　0FFFH

同样是 4K地址空间,接线不同则地址空间则不同。

2. 地址译码

用多片存储芯片构成大容量存储器时,每片的片内地址线连接到低位地址线上,片选信号由高位地址线通过译码电路产生。

地址译码电路可根据具体情况选用各种门电路构成,也可以使用现成的译码器。

常用的译码器有 2线 -4线译码器 74LS139, 3线 -8线译码器 74LS138, 4线 -16线译码器 74LS154等。各种型号译码器的工作原理和控制过程基本相同。下面简单介绍 74LS138,其引脚图如图 5-5-3所示。

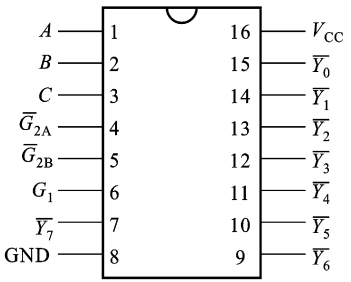


图 5-5-3　74LS138引脚图

G_1、$\overline{G}_{2A}$、$\overline{G}_{2B}$为控制端,只有当 $G_1=1$, $\overline{G}_{2A}=0$, $\overline{G}_{2B}=0$这 3个条件都满足时,3线 -8线译码器才能工作。只要有一个条件不满足,译码器功能则被禁止。

C、B、A是 3个输入端,$\overline{Y}_0 \sim \overline{Y}_7$分别为 8个输出端,输出低电平有效。3个输入端的 8种组合分别控制一个输出端有效 (为 0),其余输出为 1。74LS138译码器的真值表如表 5-3所示。

表 5-3　74LS138译码器的真值表

G_1	$\overline{G}_{2A}$	$\overline{G}_{2B}$	C	B	A	译码器输出有效
1	0	0	0	0	0	$\overline{Y}_0$
1	0	0	0	0	1	$\overline{Y}_1$
1	0	0	0	1	0	$\overline{Y}_2$
1	0	0	0	1	1	$\overline{Y}_3$
1	0	0	1	0	0	$\overline{Y}_4$
1	0	0	1	0	1	$\overline{Y}_5$
1	0	0	1	1	0	$\overline{Y}_6$
1	0	0	1	1	1	$\overline{Y}_7$
非上述情况			×	×	×	输出全为 1,无效

2线 -4线译码器,输入变量为 2个,输出变量为 4个。4线 -16线译码器,输入变量为 4个,输出变量为 16个。工作原理与 3线 -8线译码器相同。

有两种片选译码方式:全译码方式和部分译码方式。

(1) 全译码方式

除了接到片内地址引脚上的地址线外,其余所有的高位地址线都参加译码,产生片选信号。这种方式为全译码方式。

如片内用 10条地址线 A_0～A_9,若地址总线共有 16条,则剩下的 6条地址线 A_{10}～A_{15}用于译码,产生 64条片选信号。片内 10条线,寻址空间为 1K,此状态下最多可有 64组,最大容量为 64K。

全译码方式的译码电路复杂,但因所有地址线都参加了译码,译码输出是唯一的,所以每一单元地址是唯一确定的。

例 5.3 用 4K×4位 RAM组成 12K×8位存储系统,采用全译码方式。

解 本题需要 2片 4K×4位芯片构成一组,每组为 4K×8位,共 3组形成 12K×8位存储系统。片内寻址需要 12根地址线,另外 8根地址线进行全译码形成选片信号。图 5-5-4为连接图。

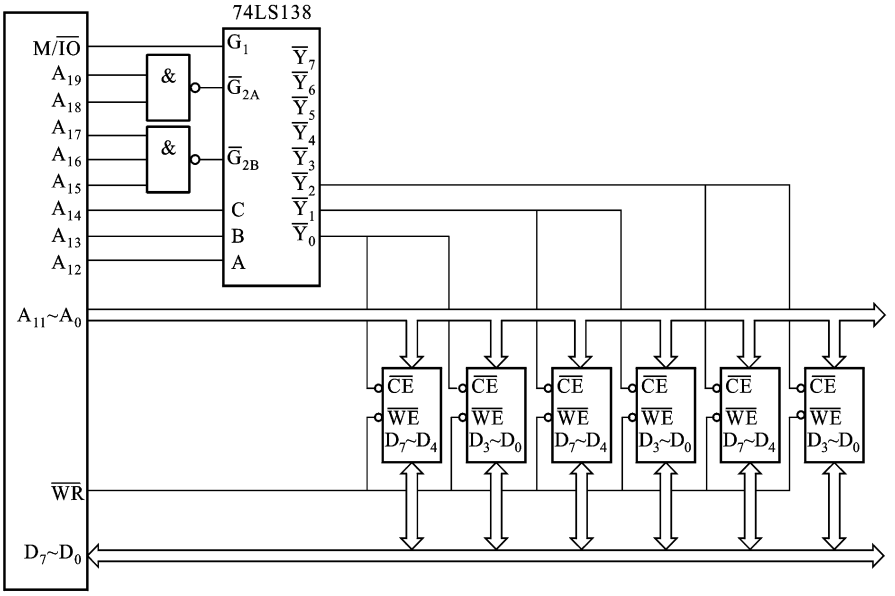


图 5-5-4 用 4K×4芯片构成 12KB RAM的连线图

12条地址线 A_{11}～A_0 寻址片内 4K单元。A_{19}～A_{12}通过 74LS138译码器进行片选译码,译码输出的 $\overline{Y}_0$～$\overline{Y}_2$ 分别作为 3组芯片的片选信号。只有当 A_{19}～A_{15}为高电平时才能使 74LS138译码器 G_1、$\overline{G}_{2A}$、$\overline{G}_{2B}$有效,通过 A_{14}、A_{13}、A_{12}来控制译码器的输出。

地址空间分配如下:

	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10} …	A_0	
第一组	1	1	1	1	1	0	0	0	0	0 …	0	F8000H
									1	1 …	1	F8FFFH
第二组	1	1	1	1	1	0	0	1	0	0 …	0	F9000H
									1	1 …	1	F9FFFH
第三组	1	1	1	1	1	0	1	0	0	0 …	0	FA000H
									1	1 …	1	FAFFFH

该例为地址全译码方式，即所有的地址信号都参加译码，来选择芯片。

⑵ 部分译码方式

除了片内地址线外，需要几条片选信号就译码产生几条，剩下的地址线不参加译码。这种方式为部分译码方式。

如图 5－5－2为用 1K×4位的芯片构成的 4K×8位 RAM，片内用 10条地址线占用 A_0～A_9，4组所需 4个片选信号用两条地址线 A_{10}、A_{11}译码产生。若共 16条地址线，则 A_{12}～A_{15}这 4条线不参与译码。

这样就存在存储单元地址不唯一的问题，即一个物理单元对应多个地址值，这种现象称为地址重叠。造成地址重叠的原因是不参加译码的地址线取值任意所引起的。因不参加译码的地址线的任何取值都不影响选择某个单元，当片内地址值和参加译码的片选地址值确定后，就选中一个存储单元，这时不参加译码的地址无论取何值都不影响该地址单元的选择，造成一个地址单元对应多个地址值的现象。

如图 5－5－2所示的连接，若设地址线为 16条，各组地址空间的分配如下：

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9 … A_0	
第一组：	0	0	0	0	0	0	0 … 0	0000H
					0	0	1 … 1	03FFH
第二组：					0	1	0 … 0	0400H
					0	1	1 … 1	07FFH
第三组：	…	…	…	…	1	0	0 … 0	0800H
					1	0	1 … 1	0BFFH
第四组：					1	1	0 … 0	0C00H
	1	1	1	1	1	1	1 … 1	0FFFH

可见 A_9～A_0的取值范围为 0000000000～1111111111，用来选择片内 1K单元，A_{11}、A_{10}为 00 01 10 11时各选择一组，但 A_{15}～A_{12}没有连接，可为任意值，这样 A_{15}～A_{12}的取值组合有 16种，使每个单元对应 16个地址值。如：

当 A_{15}～A_{12}＝0000时，地址空间为 0000H～0FFFH

当 A_{15}～A_{12}＝0001时，地址空间为 1000H～1FFFH

… … …

当 A_{15}～A_{12}＝1111时，地址空间为 F000H～FFFFH

这 16组地址空间对应同一个 4K单元，所以地址是重叠的。

5.5.4 动态存储器与 CPU的连接

如图 5－5－5所示为用 4K×4位的动态芯片构成 16K×8位的动态 RAM的连接电路。其连接与静态存储器的连接类似，只是需考虑刷新的问题。

$\overline{CE}$是芯片允许控制端，$\overline{OE}$是数据输出允许控制端。

刷新时序发生器产生刷新周期信号，启动刷新周期，使多路转换器接通刷新（地址）计数器一边，由刷新（地址）计数器提供行地址。正常读写时，多路器切换到总线一边，由地址线提供行

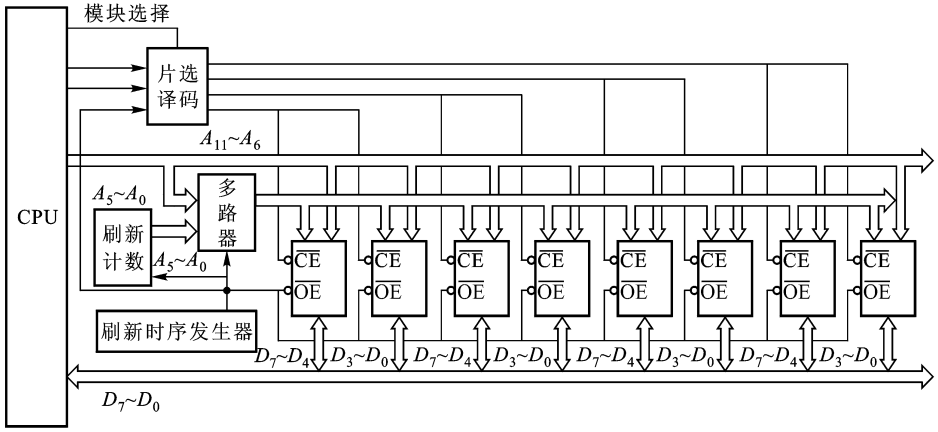


图 5-5-5 4K×4位的动态 RAM构成 16 KB模块

地址。刷新时序发生器还通过多路转换开关和各存储芯片的$\overline{OE}$端相连,在刷新周期中,使$\overline{OE}$端处于无效状态。此外,刷新时序信号通过译码电路使 4组芯片允许信号$\overline{CE}$都处于有效状态,即只进行内部刷新,而不读出数据。

4组存储芯片同时进行刷新,对 1个存储组来说,在 1个刷新周期中只刷新一行,在每个刷新周期的末尾,刷新地址计数器加 1,为下次刷新作准备。动态存储器在刷新期间不允许进行正常读写。

动态存储器有位密度高、功耗低、价格低廉的优点。

5.5.5 综合举例

由 6116 (2K×8位)组成 4K×8位的 RAM子系统,由 2732 (4K×8位)组成 8K×8位的 ROM子系统,共组成 12 KB的存储系统,采用全译码方式。

要构成要求的存储系统,需要 2片 6116组成 4K×8位 RAM子系统以及 2片 2732组成 8K×8位 ROM子系统。1片 6116为 2K×8位,所以片内寻址需要 11根地址线 (A_0~A_{10}),1片 2732为 4K×8位,所以片内寻址需要 12根地址线 (A_0~A_{11})。因为片内所需地址线数不同,所以需要在译码后再经过二级译码产生 6116的片选信号。连接电路如图 5-5-6所示。

用 74LS138译码器进行片选译码,当 $\overline{Y}_2$ 输出有效 (为 0) 时,A_{11}为低电平,第一片 6116片选信号$\overline{CE}$有效;A_{11}为高电平,第二片 6116片选信号$\overline{CE}$有效。

地址空间分配如下:

	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10} …	A_0	
第一组 2732	1	1	1	1	1	0	0	0	0	0 …	0	F8000H
									1	1 …	1	F8FFFH
第二组 2732	1	1	1	1	1	0	0	1	0	0 …	0	F9000H
									1	1 …	1	F9FFFH

第一组 6116	1	1	1	1	1	0	1	0	0 0 … 0	FA000H
									0 1 … 1	FA7FFH
第二组 6116	1	1	1	1	1	0	1	0	1 0 … 0	FA800H
									1 1 … 1	FAFFFH

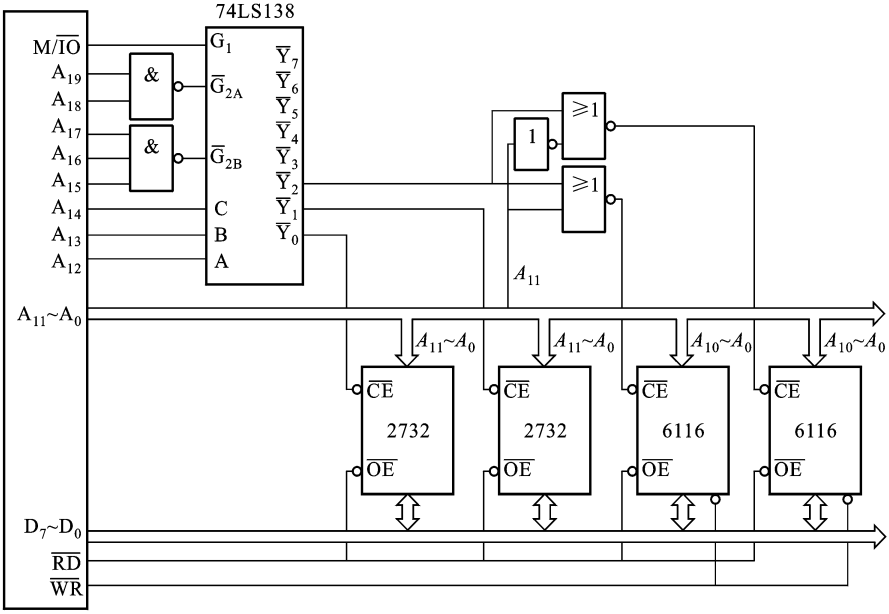


图 5-5-6 存储系统连接图

5.6 存储器的工作时序

存储器对输入信号的时序要求是很严格的,且不同规格存储器的时序要求是不相同的。

在设计存储器时,一方面要根据要求选择合适的存储器芯片,另一方面要考虑 CPU 的读/写时序和存储器的时序配合问题。

5.6.1 存储器对读/写周期的时序要求

在选择存储器时,最重要的参数是存取时间,即存储器的读取时间和写入时间。

1. 存储器的读周期

在考虑存储器读周期时,要满足图 5-6-1所示的时序要求。

图中 t_{RC}为读周期时间,即两次连续读所需的最小时间间隔。地址有效到数据输出有效的时间为 t_a,称为读取时间。片选有效到数据输出有效的时间为 t_{co}。

读周期从 CPU送出存储单元地址开始。一般有 $t_a > t_{co}$,要求片选在地址有效后的 $t_a \sim t_{co}$时间内有效,在整个读周期中,写信号一直为高电平。

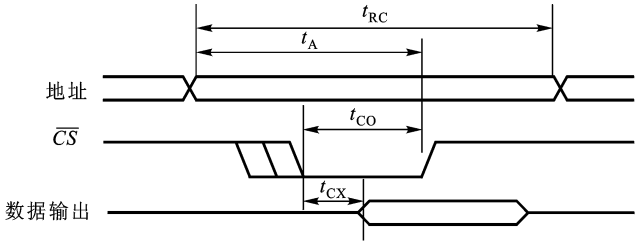


图 5-6-1 存储器读周期时序要求

2. 存储器写周期

存储器写周期时序要满足图 5-6-2所示要求。

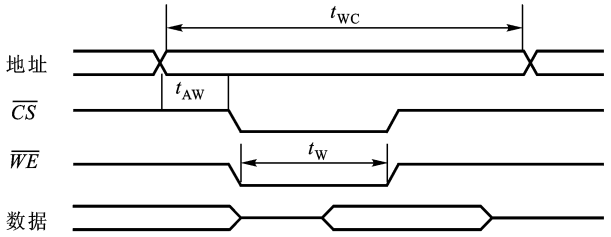


图 5-6-2 存储器写周期时序要求

图中 t_{WC}为存储器写周期，t_{AW}为地址建立时间。

t_W为写脉冲时间，为$\overline{CS}$和$\overline{WE}$同时有效的时间。

在写周期开始前要求有一段地址建立时间 t_{AW}，要求在写脉冲到来之前地址稳定，另外要保证写脉冲的宽度不小于 t_W。

5.6.2 8086对存储器的读/写时序

前面已经介绍过 8086的总线读写时序，要求与上述介绍的存储器读写时序进行配合。图5-6-3为存储器读周期时序。

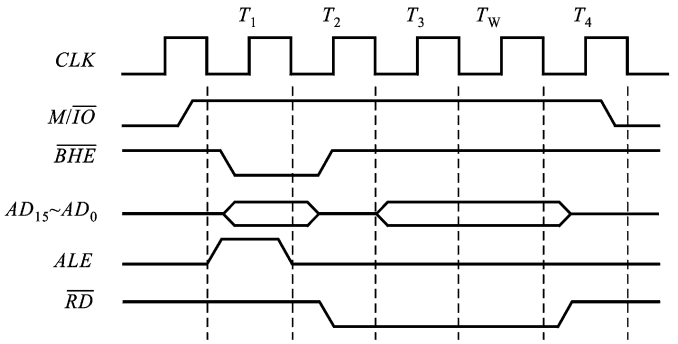


图 5-6-3 8086系统存储器读周期时序

设 8086主频为 5 MHz，则一个周期 T=200 ns。

要求 T_1～T_4的时间应大于存储器的读周期时间 t_{RC}，若时间不够可插入等待周期 T_W。一般 $M/\overline{IO}$和高位地址同时有效作为存储芯片的片选信号 $\overline{CS}$，要求 $\overline{CS}$在地址信号有效后不超过 t_R～t_O的时间内有效。

图 5－6－4为存储器写周期时序。要求地址有效时间大于存储器的写周期时间 t_{WC}，若时间不够可插入等待周期 T_W。一般 $M/\overline{IO}$和高位地址同时有效作为存储芯片的片选信号 $\overline{CS}$，要求在地址信号有效后 t_{WD}时间 $\overline{CS}$有效。且 $\overline{WR}$与 $\overline{CS}$同时有效时间应大于 t_W。

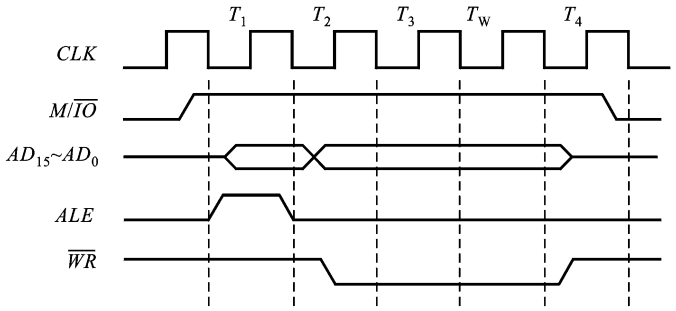


图 5－6－4　8086系统存储器写周期时序

习　　题

1. 计算机的内存有什么特点，内存有哪些类，外存一般指哪些设备，外存有什么特点？

2. 用基本存储电路组成内存时，为什么总是采用矩阵形式？试用一个具体的例子说明。

3. 为了节省存储器的地址译码电路，一般采用哪些方法？

4. RAM与 CPU进行连接时，需要考虑哪些因素？

5. 什么是地址全译码，什么是地址部分译码，各有什么特点？

6. 由存储芯片构成存储器时，怎样确定需要多少芯片，如何进行分组，地址空间如何确定？

7. 静态 RAM芯片上为什么往往只有写信号而没有读信号，怎样才能发出读控制信号？

8. 写出下列容量的芯片片内的地址线和数据线的条数。

(1) 512K×4位　　(2) 64K×1位

9. 用下列芯片构成存储系统，各需要多少个 RAM芯片，分别采用部分译码方式和全译码方式，问片内地址线条数和片外地址线条数各为多少？

(1) 用 512×4位 RAM构成 8K×8位的存储系统。

(2) 用 1024×1位 RAM构成 32K×8位的存储系统。

10. 用 2114芯片构成 2K×8位的 RAM，采用部分译码方式，画出连接图，并确定地址空间。

11. 用 2K×1位的存储芯片构成 2K×8位 RAM，采用部分译码方式，画出连接图，并确定地址空间。

12. 用 4K×8位的存储芯片构成 16K×8位 RAM，采用全译码方式，画出连接图，并确定地址空间。

13. 一个存储系统有 4K×8位的 RAM和 2K×8位的 ROM，假设采用 1K×8位的 RAM和 ROM芯片，构成该存储系统。采用全译码方式，画出连接图，并确定地址空间。

14. 动态 RAM工作时有什么特点，和静态 RAM相比，动态 RAM有什么长处，有什么不足处，动态 RAM一般用在什么场合？

15. 动态 RAM为什么要进行刷新，刷新过程和读操作进程有什么差别？

16. ROM、PROM、EPROM各有什么不同，分别用在什么场合？

第 6章　8086指令系统与汇编基础

计算机是通过执行指令来完成各种工作的，不同类型的计算机所具有的指令系统是不同的，本章主要介绍了微型计算机 8086的寻址方式、指令系统及汇编基础。

6.1 概　　述

6.1.1 指令及指令系统概念

在第一章概述中已经指出，计算机系统由硬件系统（Hardware System）和软件系统（Software System）两部分组成。计算机硬件必须配置相应软件才能进行工作。软件存储在存储器中，硬件系统完全按照存储器中的软件进行工作，也就是说，计算机的工作过程就是执行软件的过程。不论是系统软件还是应用软件，都是由计算机硬件所能识别的指令组成的程序。

① 指令（Instruction）：是要求计算机执行特定操作的命令，通常一条指令对应一种特定操作。指令的执行是在计算机的 CPU中完成的，每条指令规定的运算及基本操作都是简单的、基本的，它和计算机硬件所具备的能力相对应。

② 指令系统（Instruction set）：计算机所能执行的全部指令的集合组成该计算机的指令系统。不同类型的计算机具有不同的指令系统，如 8086的指令系统就不同于 80286、80386、80486、Pentium等微型计算机及 MCS－51单片机的指令系统。

6.1.2 机器指令和汇编指令格式

1. 机器指令

计算机编程语言有机器语言、汇编语言及高级语言等。机器语言与计算机的核心——CPU相对应，不同类型的计算机有其独特的机器语言指令系统；汇编语言仅是机器语言的英文助记符表示形式，也与相应的计算机系统相对应；高级语言脱离了具体的计算机，具有通用性。

计算机只能识别二进制代码，因此计算机能执行的指令必须以二进制代码的形式表示，这种以二进制代码形式表示的指令称为指令的机器码（Machine Code）。既使用汇编语言编写的程序输入计算机后，若要在计算机上执行，也必须由机器提供的“汇编程序”（Assembler）将它翻译成由机器指令组成的机器语言程序，才能最终被计算机识别并执行。

2. 汇编指令格式

计算机是通过执行指令来处理数据的，为了指出数据的来源、操作结果的去向及所执行的操作，一条指令一般包含操作码和操作数两部分。操作码是指令的重要组成部分，用来表示该指令所要完成的操作，不同的指令用不同的操作码表示；操作数用来描述指令的操作对象，操作数可以是立即数、寄存器和存储器，不同的指令可以有 1个、2个、3个或无操作数，根据操作数个数的不同指令格式分为以下几种：

⑴ 零操作数指令

格式：

操作码

指令中只有操作码，没有操作数，也称为无操作数指令。有两种情况使用这种指令：一是指令中不需要任何操作数，如空操作指令、停机指令等；二是指令的操作数是默认的，如加法的ASCII码调整、十进制调整指令等。

⑵ 一操作数指令

格式：

操作码	A

其中 A为存储器地址或寄存器名

指令中只给出一个地址，该地址既是操作数的地址，又是操作结果的存储地址，如增量、减量指令等。

⑶ 二操作数指令

格式：

操作码	A1	A2

这是最常见的指令格式。A1、A2指出两个源操作数的地址，其中一个还指出存放结果的目的地址。对两个操作数完成所规定的操作后，将结果存入目的地址。

⑷ 三操作数指令

格式：

操作码	A1	A2	A3

A1、A2指出两个源操作数的地址，A3指出存放结果的目的地址。

⑸ 多操作数指令

在某些性能较好的大、中型甚至高档微小型计算机中，往往设置一些功能很强的、用于处理成批数据的指令。为了描述一批数据，指令中需要多个操作数来指出数据存放的首地址、长度和下标等信息。

6.2 8086的寻址方式

计算机通过执行指令进行信息处理，但参与处理的信息并不都是直接出现在指令中，大多数时候存放在存储器和外部设备中。所谓指令的寻址方式（Addressing Mode）就是指令中操作数的表示方式。由于程序编写的需要，大多数情况下，指令中并不直接给出操作数的数值，而是给出操作数存放的地址。并且许多情况下操作数的地址也不直接给出，而是给出计算操作数地址的方法。计算机执行程序时，根据指令给出的寻址方式，计算出操作数的地址，然后从该地址中取出数据处理，处理后再把处理结果送入某操作数地址中去。一般说来，计算机的寻址方式越丰富，指令系统的功能就越强，工作的灵活性就越大。

8086的寻址方式有：立即寻址、寄存器寻址、直接寻址、寄存器间接寻址、寄存器相对寻址、基址变址寻址、相对基址变址寻址（以上寻址与数据有关）、段内直接寻址、段内间接寻址、段间直接寻址及段间间接寻址（与程序转移有关）。

6.2.1 立即寻址

立即寻址（Immediate Addressing）方式所提供的操作数直接存放在指令中，紧跟在操作码之后，操作数作为指令的一部分存放在代码段里，这种操作数称为立即数。立即数可以是8位的或16位的，若是16位数，则高位字节存放在高地址中，低位字节存放在低地址中。立即寻址方式常用于为寄存器赋初值，并只能用做源操作数，不能用做目的操作数。

例 6.1] MOV AL, 12H

则指令执行后，(AL) =12H

例 6.2] MOV DS, 3456H

则指令执行后，(DS) =3456H

6.2.2 直接寻址

直接寻址（Direct Addressing）方式指令，操作数在存储器中，指令中直接提供操作数的16位偏移地址（Effective Address, EA），EA紧跟在指令操作码之后。由于操作数一般存放在数据段中，所以必须先计算出操作数的物理地址，再访问存储器才能取得数据。

例 6.3] MOV AX, [2000H]

如 (DS) =3000H，则指令执行情况如图 6－2－1所示。执行结果为 (AX) =8060H。

如果数据在数据段以外的其他段中，应在指令中指定段跨越前缀，如

MOV AX, ES：[2000H]

在汇编语言中，可用符号地址代替数值地址，可以写为：

MOV AX, BLOCK

或

MOV AX, [BLOCK]

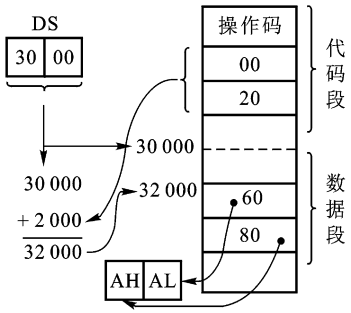


图 6－2－1 例 6.3的执行过程

6.2.3 寄存器寻址

寄存器寻址（Register Addressing）的指令，操作数在寄存器中，即寄存器的内容就是操作数的数值。由于这种寻址方式操作数就在寄存器中，不需访问存储器即可取得操作数，因而速度快。可使用的寄存器有通用寄存器和段寄存器，寄存器可作为源操作数或目的操作数。

例 6.4] MOV AX, BX

如执行前 (AX) =4567H, (BX) =1234H，则执行后，(AX) =1234H, (BX) 保持不变。

6.2.4 寄存器间接寻址

寄存器间接寻址（Register Indirect Addressing）方式中，操作数存放在存储器中，但操作数的有效地址 EA在基址寄存器 BX、BP或变址寄存器 SI、DI中。操作数的物理地址为：

物理地址 =16×（段寄存器）+（寄存器）

如果寄存器是 BX、SI、DI，则段寄存器用 DS如果寄存器是 BP，则段寄存器用 SS。

例 6.5】　MOV　AX, [BX]

如果 (DS) =3000H, (BX) =2000H,则物理地址 =30000+2000=32000H,执行情况如图 6-2-2所示。执行结果为：(AX) =8060H

指令中也可指定段跨越前缀以取得其他段中的数据,如 MOV　AX, ES: [BX]指令,从附加段中取得数据。

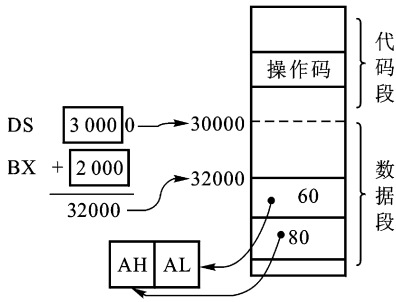


图 6-2-2　例 6.5的执行过程

6.2.5　寄存器相对寻址

采用寄存器相对寻址 (Register Relative Addressing)时,操作数的有效地址 EA由一个基址或变址寄存器的内容和指令中给出的 8位或 16位的位移量 (Displacement)相加得到,即：

物理地址 =16×(段寄存器) + (寄存器) +位移量

例 6.6】　MOV　AX, COUNT [SI]或写为 MOV AX, [COUNT+SI]

若 (DS) =3000H, (SI) =2000H, COUNT=3000H

则物理地址 =30000H+2000H+3000H=35000H

6.2.6　基址变址寻址

采用基址变址寻址 (Based Indexed Addressing)时,操作数的有效地址为基址寄存器 (BX或 BP)和变址寄存器 (SI或 DI)的内容之和,即：

物理地址 =16×(段寄存器) + (基址寄存器) + (变址寄存器)

例 6.7】　MOV　AX, [BX] [DI]或写为 :MOV　AX, [BX+DI]

若 (DS) =3000H, (BX) =1000H, (DI) =2000H

则有效地址 =1000H+2000H=3000H,物理地址 =30000H+3000H=33000H

6.2.7　相对基址变址寻址

相对基址变址寻址 (Relative Based Indexed Addressing)操作数的有效地址为基址寄存器 (BX或 BP)和变址寄存器 (SI或 DI)的内容及 8位或 16位位移量之和,即：

物理地址 =16×(段寄存器) + (基址寄存器) + (变址寄存器) +位移量

例 6.8】　MOV　AX, MASK [BX] [SI]

若 (DS) =3000H, (BX) =1000H, (SI) =2000H, MASK=250H

则物理地址 =30000H+1000H+2000H+250H=33250H

6.2.8　程序转移寻址

指令在顺序执行时,下一条指令的地址总是由指令指针 IP自动递增得到。若程序非顺序执行,要得到将要执行的指令的地址,需要分为本段内转移和非本段内 (段间)转移两种情况考虑。若在本段内转移,需要给出即将执行的那条指令的偏移地址,并用它取代 IP的原有内容;若在段间转移,除了要给出偏移地址取代 IP外,还要给出新的代码段的段基址取代 CS中原有的内容。

此时操作数作为转移地址使用,分别赋予 IP和 CS。

1. 段内直接寻址 (IntrasegmentDirectAddressing)

转移地址为当前 IP内容加上指令中紧跟操作码之后的相对位移量 (8位或 16位),即转移的有效地址以相对于当前 IP值的位移量来表示,因此也被称为相对寻址。该方式适用于条件转移和无条件转移,用于条件转移时,位移量只能为 8位;用于无条件转移时,位移量可为 8位或16位。有效地址为:EA=(IP)+位移量。

指令的汇编语言格式表示为:

```
JMP  NEAR PTR VALUE ;位移量为 16位在符号地址前加 NEAR PTR
JMP  SHORT TABLE ;位移量为 8位在符号地址前加 SHORT
```

2. 段内间接寻址 (IntrasegmentIndirectAddressing)

转移的有效地址 EA是一个寄存器或一个存储器单元的内容,其内容可用寄存器寻址、寄存器间接寻址、寄存器相对寻址、基址变址寻址、相对基址变址寻址等寻址方式获得,用所取得的内容取代 IP寄存器的原有内容。该寻址方式只能用于段内无条件转移。

指令的汇编语言格式表示为:

```
JMP  BX
JMP  WORD PTR [BX+COUNT] ;WORD PTR用以指定所取得的转向地址是一个字的有效地址。
```

3. 段间直接寻址 (IntersegmentDirectAddressing)

该寻址方式在指令中直接给出了转移到的段地址和偏移地址,第一个地址为偏移地址,第二个地址为段地址,这两个地址都是 16位的地址,用于取代 IP和 CS,从而实现段间转移。

指令的汇编语言格式表示为:

```
JMP  FAR PTR VALUE ;FAR PTR是表示段间转移的操作符
```

4. 段间间接寻址 (IntersegmentIndirectAddressing)

该寻址方式用存储器中的两个连续字单元的内容作为转移到的偏移地址和段地址,来取代 IP和 CS寄存器中的原有内容,从而达到段间转移的目的。这里存储器单元内容的取得,可以采用寄存器寻址、寄存器间接寻址、寄存器相对寻址、基址变址寻址、相对基址变址寻址中的任何一种。

指令的汇编语言格式表示为:

```
JMP  DWORD PTR [BX+D]  ;DWORD PTR说明转向地址需取双字。
```

6.3 8086指令系统

8086汇编语言指令丰富、格式灵活,能处理多种类型的数据,具有较强的寻址能力。8086的指令系统从功能上可以分为数据传送指令、算术运算指令、逻辑运算指令、串处理指令、控制转移指令及处理机控制指令六大类,下面分别加以说明。

6.3.1 数据传送指令

数据传送指令是程序中使用最多的一类指令,在程序中占据很大的比例,负责把数据、地址或立即数传送到寄存器或存储器单元中,数据传送指令又分为四小类:通用数据传送指令、累加

器专用传送指令、地址传送指令、标志寄存器传送指令。

1. 通用数据传送指令

通用数据传送指令中包括最基本的传送指令 MOV,入栈指令 PUSH和出栈指令 POP以及交换指令 XCHG。特别需要说明的是,只有在通用数据传送指令中段寄存器（CS除外）才可以作为操作数使用,而其他的指令是绝对不能将段寄存器作为操作数使用的。

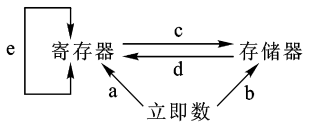
(1) 最基本的传送指令 MOV (Move)

指令格式 :MOV　　目的操作数,源操作数

指令功能 :将源操作数内容传送到目的操作数,源操作数内容不变。

注意事项：

① 目的操作数和源操作数的搭配规则,如图 6-3-1所示。



a: MOV　AX, 1234H
b: MOV　[BX], 1234H
c: MOV　[BX], AX
d: MOV　AX, [BX]
e: MOV　AX, BX

图 6-3-1　目的操作数和源操作数的搭配规则

由搭配规则可知,还需注意以下几点：

立即数只能作为源操作数,而不能作为目的操作数，如：

MOV　　12H, AL　　(×)

MOV　　AL, 12H　　(√)

存储器单元之间不能直接传送数据,如：

MOV　　[DI], [SI]　　(×)

正确写法如下：

MOV　　AX, [SI]

MOV　　[DI], AX　　(√)

CS不能作为目的操作数,即 CS的值不能随意改变,如：

MOV　　CS, AX　　(×)

不允许两个段寄存器之间直接传送信息,如：

MOV　　DS, ES　　(×)

立即数不能直接送给段寄存器,要通过其他寄存器,如：

MOV　　DS, 1234H　　(×)

正确写法如下：

MOV　　AX, 1234H

MOV　　DS, AX　　(√)

② 目的操作数和源操作数的类型要匹配,如：

MOV　　AX, BL　　(×)

MOV　　AL, DX　　(×)

MOV AL,BL （√）

MOV AX,DX （√）

③ 在给 SS用 MOV指令赋值时,要紧接着给 SP赋值,不能在修改 SS和 SP指令之间插入其他指令,系统在执行这两条语句时,自动禁止外部中断,以防止堆栈空间变动过程中出现中断。

④ 不影响标志寄存器的值。

[例 6.9] 执行下面程序段：

```
MOV     AX,5000H        ;(AX)=5000H
MOV     DS,AX           ;(DS)=5000H
MOV     [5000H],AL      ;(55000H)=00H
MOV     BX,AX           ;(BX)=5000H
MOV     [BX],56H        ;(55000H)=56H
MOV     [BX+1],1234H    ;(55001H)=34H、(55002H)=12H
MOV     CX,[BX]         ;(CX)=3456H
```

(2) 入栈指令 PUSH (Push Word Onto Stack)和出栈指令 POP (Pop Word Off Stack)

指令格式：USH 源操作数

POP 目的操作数

指令功能：USH指令是把源操作数送入堆栈的顶部

POP指令是把栈顶内容弹出到目的操作数

[例 6.10] 若 (AX)=1234H, (BX)=5678H, (SP)=1056H,执行下面 4条指令：

```
PUSH    AX
PUSH    BX
POP     AX
POP     BX
```

指令执行情况如图 6-3-2所示。

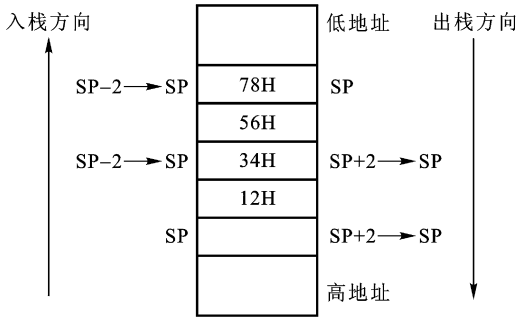


图 6-3-2 例 6.10的执行过程

通过如上执行过程可知,执行后, (AX)=5678H, (BX)=1234H, (SP)=1056H,所以这 4条指令完成的功能是交换寄存器 AX和 BX的内容。

注意事项：

① 入栈操作是先改变指针 SP再入栈;出栈操作是先出栈再改变指针 SP。

② 入栈是 SP逐渐靠近基地址的过程,SP始终指向最后入栈的地址单元;出栈是 SP逐渐远离基地址的过程,SP始终指向即将出栈的地址单元。

③ 对栈操作时低字节放在低地址单元,高字节放在高地址单元。

④ 堆栈操作符合后进先出(或先进后出)的原则。

⑤ 堆栈位置由 SS决定,堆栈容量由 SP决定,堆栈容量即为 SP的初值与 SS之间的距离,8086堆栈容量为 64 KB。

⑥ 堆栈指令只能对字操作而不能对字节进行操作,如:

PUSH　BL　(×)

POP　DH　(×)

PUSH　SI　(√)

POP　ES　(√)

⑦ 堆栈指令的操作数可以是寄存器和存储器,但 CS只能作为源操作数入栈,而不能作为目的操作数从堆栈中弹出一个值到 CS寄存器。

(3) 交换指令 XCHG (Exchange)

指令格式:XCHG　目的操作数,源操作数

指令功能:将目的操作数内容和源操作数内容相互交换。

例 6.11　若 (AX)=1234H, (BX)=5678H,执行下面指令:

XCHG　AX, BX

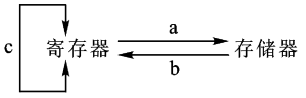
则执行后, (AX)=5678H, (BX)=1234H

例 6.12　若 (BX)=6F30H, (BP)=0200H, (SI)=0046H, (SS)=2F00H, (2F246H)=4154H,执行指令:XCHG　BX, [BP+SI]

则执行后, (BX)=4154H, (2F246H)=6F30H

注意事项:

① 目的操作数和源操作数的搭配规则,如图 6-3-3所示。



a: XCHG　[BX], AX
b: XCHG　AX, [BX]
c: XCHG　BX, AX

图 6-3-3　目的操作数和源操作数的搭配规则

② 段寄存器不能相互交换。

③ 目的操作数和源操作数位数要统一。

2. 累加器专用传送指令

8086和其他微处理器一样,将累加器作为数据传送的核心。在 8086指令系统中,有两类指令是专门通过累加器来执行的,即输入/输出指令和换码指令。

(1) 输入指令 IN和输出指令 OUT

指令格式:N　累加器,端口地址

OUT 端口地址,累加器

指令功能:IN指令是从 I/O端口读入信息到累加器

OUT指令是从累加器中输出信息到 I/O端口

指令用途:所有 I/O端口与 CPU之间的通信都由 IN和 OUT指令来完成。

注意事项:

① 累加器可以是 16位的 AX或 8位的 AL

② 分为直接输入输出指令和间接输入输出指令。直接输入输出指令在指令中直接指定端口号,寻址范围为 0～255,共 256个端口;间接输入输出指令是先把端口号放到 DX寄存器中,即在指令中用 DX代替端口号,寻址范围为 0～65 535,共 65 536个端口。

例 6.13] 指令功能注释:

IN AX,70H ;将 70H、71H两个端口的值读入到 AX

IN AX,DX ;将 DX、DX+1所指两个端口的一个字读入到 AX

OUT 70H,AL ;将 AL中的一个字节输出到 70H端口

OUT DX,AL ;将 AL中的一个字节输出到 DX所指的端口

例 6.14] 若 (90H)=12H,(91H)=34H,执行下面指令:

IN AX,90H

则执行后,(AX)=3412H

例 6.15] 若 (AL)=10H,(DX)=2000H,执行下面指令:

MOV DX,2000H

OUT DX,AL

则执行后,(2000H)=10H

(2) 换码指令 XLAT (Translate)

指令格式: LAT 符号地址

或

XLAT

指令功能:将一种代码转换成另一种代码。

执行操作:[BX+AL] → AL,即将表格的首地址预先存到 BX中,要查的表中数据距表首地址的位移量要预先存到 AL寄存器中,根据 BX和 AL的内容将找到的数送到 AL寄存器中。

例 6.16] 若 (BX)=0050H,(AL)=0BH,

(DS)=8000H,(8005BH)=30H

执行指令:XLAT

则执行后,(AL)=30H

执行情况如图 6-3-4所示。

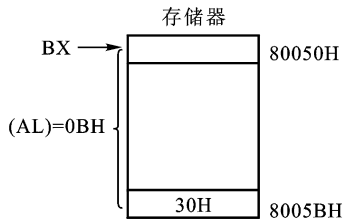


图 6-3-4 例 6.16的执行过程

3. 地址传送指令

地址传送指令完成把地址传送到指定寄存器的功能,地址传送指令处理的是变量的地址,而不是变量的值或变量的内容。在 8086指令系统中,有 3条专用于传送地址的指令:取有效地址指令 LEA,将地址指针送寄存器和 DS指令 LDS,将地址指针送寄存器和 ES指令 LES,

(1) 取有效地址指令 LEA (Load Effective Address)

指令格式：LEA 目的操作数，源操作数

指令功能：将源操作数的有效地址送到目的操作数中。

注意事项：源操作数必为内存单元地址或符号地址，目的操作数必为一个 16位的通用寄存器。

例 6.17] 若 (BX) =1200H, (SI) =0300H，执行下面指令：

LEA DI, [BX+SI+0100H]

则执行后，(DI) =1600H

(2) 地址指针送寄存器和 DS指令 LDS (Load Data Segment Register With Pointer)

指令格式：LDS 目的操作数，源操作数

指令功能：将源操作数指定的 4个字节的地址指针 (其中包括一个段地址和一个偏移量) 传送到指令指定的寄存器及 DS寄存器中，该指令常指定的寄存器一般为 SI。

操作过程：源操作数] → 指定的寄存器 SI

[源操作数 +2] → DS

例 6.18] 若 (DS) =2000H, (20060H) =3000H, (20062H) =4000H，执行下面指令：

LDS SI, [60H]

则执行后，(SI) =3000H, (DS) =4000H

(3) 地址指针送寄存器和 ES指令 LES (Load Extra Segment Register With Pointer)

指令格式：LES 目的操作数，源操作数

指令功能：将源操作数指定的 4个字节的地址指针 (其中包括一个段地址和一个偏移量) 传送到指令指定的寄存器及 ES寄存器中，该指令常指定的寄存器一般为 DI。

操作过程：源操作数] →指定的寄存器 DI

[源操作数 +2] → ES

例 6.19] 若 (DS) =5000H, (BX) =0200H, (50200H) =6000H, (50202H) =7000H，执行下面指令：

LES DI, [BX]

则执行后，(DI) =6000H, (ES) =7000H

4. 标志传送指令

在程序执行过程中，有时需要将标志寄存器内容保护、恢复或判断，这就需要存取处理机的状态标志。有 4条状态标志传送指令：标志送 AH指令 LAHF，AH送标志寄存器指令 SAHF，标志进栈指令 PUSHF，标志出栈指令 POPF。

(1) 标志送 AH指令 LAHF (Load AH With Flags)

指令格式：LAHF

指令功能：将标志寄存器的低 8位传送到 AH中。传送后，AH寄存器的 D_1、D_3、D_5 位没有意义，如图 6－3－5所示。

(2) AH送标志寄存器指令 SAHF (Store AH Into Flags)

指令格式：SAHF

指令功能：与 LAHF相反，将 AH寄存器的内容传送到标志寄存器的相应位。

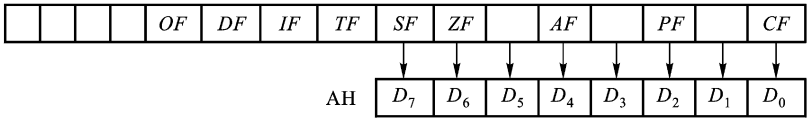


图 6-3-5 LAHF指令的功能

(3) 标志进栈指令 PUSHF (Push The Flags)

指令格式:PUSHF

指令功能:将标志寄存器的值推入堆栈顶部,但标志寄存器的值不变,且使栈指针 SP的值减 2。

注意事项:PUSHF一般用在子程序和中断处理程序之首,用来保存主程序标志。

(4) 标志出栈指令 POPF (Pop The Flags)

指令格式:POPF

指令功能:从堆栈中弹出一个字到标志寄存器,即标志寄存器的值改变,且使栈指针 SP的值加 2。

注意事项:POPF一般用在子程序和中断处理程序之尾,用来恢复主程序标志。

6.3.2 算术运算类指令

8086的算术运算指令可对数据进行加、减、乘、除等基本运算。大部分算术运算类指令影响标志位。

1. 加法指令

主要包括:不带进位加法指令 ADD、带进位加法指令 ADC和加 1指令 INC。

(1) 不带进位加法指令 ADD (Addition)

指令格式:ADD 目的操作数,源操作数

指令功能:源操作数内容 +目的操作数内容→目的操作数

注意事项:

① 目的操作数和源操作数的搭配规则与 MOV指令相同。

② 对 6个状态标志均有影响。

[例 6.20] 若 (AL)=8EH, (BL)=0D6H,执行下面指令:

ADD AL, BL

则执行后, (AL)=64H,且 CF=1, AF=1, ZF=0, SF=0, PF=0, OF=1

(2) 带进位加法指令 ADC (Add With Carry)

指令格式:ADC 目的操作数,源操作数

指令功能:源操作数内容 +目的操作数内容 +CF内容→目的操作数

注意事项:这条指令一般用在多字节加法中,从第二字节以后的加法使用本条指令。

[例 6.21] 若 (AL)=0C8H, (BL)=5FH, CF=1,执行下面指令:

ADC AL, BL

则执行后, (AL)=28H,且 CF=1, AF=1, ZF=0, SF=0, PF=1, OF=0

例 6.22】　若有 2个 4字节的数,分别存放在自 FIRST和 SECOND开始的存储区中,存放时高字节在高地址中,低字节在低地址中,实现这两个数相加,并将结果保存在 THIRD中。程序段如下:

```
MOV     AX, FIRST
ADD     AX, SECOND
MOV     THIRD, AX
MOV     AX, FIRST+2
ADC     AX, SECOND+2
MOV     THIRD+2, AX
```

(3) 加 1指令 INC (Increment Destination By 1)

指令格式:INC　　操作数

指令功能:操作数内容 +1→操作数。其中,操作数可以是寄存器和存储器。

注意事项:

① INC指令不影响 CF标志。

② INC指令主要用于修改地址指针和循环中的计数次数。

例 6.23】　若 (CX)=6789H,执行下面指令:

```
INC     CX
```

则执行后,(CX)=678AH

2. 减法指令

主要包括:不带借位减法指令 SUB,带借位减法指令 SBB,减 1指令 DEC,求补指令 NEG,比较指令 CMP。

(1) 不带借位减法指令 SUB (Subtract)

指令格式:SUB　　目的操作数,源操作数

指令功能:目的操作数内容 -源操作数内容→目的操作数

注意事项:

① 目的操作数和源操作数的搭配规则与 MOV指令相同。

② 对 6个状态标志均有影响。

例 6.24】　若 (AL)=7CH, (BL)=0E5H,执行下面指令:

```
SUB   AL, BL
```

则执行后,(AL)=97H,且 CF=1, AF=0, ZF=0, SF=1, PF=0, OF=1

(2) 带借位减法指令 SBB (Subtract With Borrow)

指令格式:SBB　　目的操作数,源操作数

指令功能:目的操作数内容 -源操作数内容 -CF内容→目的操作数

注意事项:这条指令一般用在多字节减法中,从第二字节以后的减法使用本条指令。

例 6.25】　有 2个 4字节的数,分别存放在数据段中偏移地址为 1000H与 2000H开始的存储单元中,存放时高字节在高地址中,低字节在低地址中,实现这两个数相减,并将结果保存在 3000H开始的单元中。程序段如下:

```
MOV     AX, [1000H]
```

```
SUB     AX, [2000H]
MOV     [3000H], AX
MOV     AX, [1002H]
SBB     AX, [2002H]
MOV     [3002H], AX
```

(3) 减1指令 DEC (Decrement Destination By 1)

指令格式：DEC 操作数

指令功能：操作数内容 −1→操作数。其中，操作数可以是寄存器和存储器。

注意事项：

① DEC指令不影响 CF标志。

② DEC指令主要用于修改地址指针和循环中的计数次数。

(4) 求补指令 NEG (Negate)

指令格式：NEG 操作数

指令功能：0−操作数→操作数或将操作数按位取反再加 1。

注意事项：

① NEG指令影响标志位 CF、AF、ZF、SF、PF、OF

② 如果操作数的值为 −128 (即 80H) 或 −32768 (即 8000H)，则执行 NEG指令后，结果不变，但使 OF置 1。

③ NEG指令通常使 CF为 1，只有当操作数为 0时，才使 CF为 0。

[例 6.26] 若 (AL) =34H，执行下面指令：

```
NEG     AL
```

则执行后，(AL) =0CCH，且 CF=1、AF=0、ZF=0、SF=1、PF=1、OF=0

(5) 比较指令 CMP (Compare Two Operands)

指令格式：CMP 目的操作数，源操作数

指令功能：目的操作数内容 −源操作数内容，但结果不回送，只是使结果影响标志位，用以比较两数大小。

注意事项：

① 通过 ZF标志来判断两数是否相等。若 ZF=1则相等；ZF=0则不等。

② 对于无符号数，通过 CF标志来判断两数大小。若 CF=0，则被减数大于减数；若 CF=1，则被减数小于减数。

③ 对于有符号数，通过 OF和 SF两个标志来判断两数的大小。若 OF和 SF状态相同，则被减数大于减数；若 OF和 SF状态不同，则被减数小于减数。

[例 6.27] 判断 AX与 BX的内容是否相等，若相等，则 (CX) =1，否则 (CX) =0，编写程序段如下：

```
L1: MP      AX, BX
    JZ      L1
    MOV     CX, 0
    JMP     L2
```

```
L1:MOV    CX,1
L2:HLT
```

3. 乘法指令

乘法指令包括无符号数乘法指令 MUL和带符号数乘法指令 IMUL。乘法指令中,有一个操作数总是放在 AL(8位)或 AX(16位)中,乘得结果总是放在 AX(8位)或 DX、AX(16位)中,其中 DX存放高位字,AX存放低位字。

(1) 无符号数乘法指令 MUL(Unsigned Multiply)

指令格式:MUL 源操作数

指令功能:字节操作数为(AL)×源操作数内容→(AX)

字操作数为(AX)×源操作数内容→(DX、AX)

注意事项:

① MUL指令影响 CF、OF标志,而对 AF、PF、SF、ZF是不确定的,因此这 4个标志位无意义。

② 如果乘积的高一半为 0,即字节操作的(AH)或字操作的(DX)为 0,则 CF、OF均为 0;否则 CF、OF均为 1。

(2) 带符号数乘法指令 IMUL(Signed Multiply)

指令格式:IMUL 源操作数

指令功能:字节操作数为(AL)×源操作数内容→(AX)

字操作数为(AX)×源操作数内容→(DX、AX)

注意事项:

① MUL指令影响 CF、OF标志,而对 AF、PF、SF、ZF是不确定的,因此这 4个标志位无意义。

② 如果乘积的高一半是低一半的符号扩展,则 CF、OF均为 0;否则 CF、OF均为 1。

例 6.28 若(AL)=0B4H,(BL)=11H,执行下面指令:

```
MUL    BL
```

则执行时,只需将(AL)和(BL)直接相乘即可,执行后得(AX)=0BF4H,且 CF=OF=1

```
IMUL   BL
```

则执行时,要将(AL)求补后再和(BL)相乘,乘得结果为 050CH,再将其求补得(AX)=0FAF4H,且 CF=OF=1

例 6.29 实现两个字相乘的程序段如下:

```
MOV    AX,FIRST
MUL    SECOND
MOV    THIRD,AX
MOV    FOURTH,DX
```

4. 除法指令

除法指令包括无符号数除法指令 DIV和带符号数除法指令 IDIV以及符号扩展指令 CBW和 CWD。

(1) 无符号数除法指令 DIV(Unsigned Division)

指令格式：DIV 源操作数

指令功能：字节操作数：(AL) ← (AX)／(源操作数的)商
(AH) ← (AX)／(源操作数的)余数
字操作数：(AX) ← (DX,AX)／(源操作数的)商
(DX) ← (DX,AX)／(源操作数的)余数

注意事项：

① DIV指令要求除数只能是被除数的一半字长。当被除数为16位时，除数应为8位；当被除数为32位时，除数应为16位。

② 当被除数为16位时，应存放在AX中，除数为8位，可存放在寄存器或存储器中(不能为立即数)，得到的8位商放在AL中，8位余数放在AH中；当被除数为32位时，应存放在DX(高位)和AX(低位)中，除数为16位，可存放在寄存器或存储器中(不能为立即数)，得到的16位商放在AX中，16位余数放在DX中。

③ DIV指令对标志位CF、AF、ZF、SF、PF、OF都是不确定的，即没有意义。

④ 被除数位数和除数位数相同时，要对被除数进行扩展，对于无符号数来说，只需使AH或DX内容为0即可。

(2) 带符号数除法指令IDIV (Signed Division)

指令格式：IDIV 源操作数

指令功能：与DIV指令相同。

注意事项：

① 与DIV指令类似，但操作数必须是带符号数，商和余数也都是带符号数，且余数的符号和被除数的符号相同。

② 当为字节操作时，被除数高8位的绝对值大于除数的绝对值(即商超过了8位)；或当为字操作时，被除数高16位的绝对值大于除数的绝对值(即商超过了16位)；或当除数为0时，就产生0号中断进行处理。

③ 被除数位数和除数位数相同时，要对被除数进行扩展，对于有符号数来说，AH和DX的扩展就是低位字节或低位字的符号扩展，即把AL中的最高位扩展到AH的8位中，或把AX中的最高位扩展到DX的16位中。在8086中，有专门用于有符号数扩展的指令CBW和CWD。

(3) 字节转换为字指令CBW (Convert Byte To Word)

指令格式：CBW

指令功能：将AL中的内容进行符号扩展。若(AL)的最高位为0，则(AH)←0；若(AL)的最高位为1，则(AH)←0FFH。

注意事项：

① 当遇到两个字节相除时，要先执行CBW指令，以便产生一个16位的被除数。

② 该指令不影响标志位。

(4) 字转换为双字指令CWD (Convert Word To Double Word)

指令格式：CWD

指令功能：将AX中的内容进行符号扩展。若(AX)的最高位为0，则(DX)←0；若(AX)的最高位为1，则(DX)←0FFFFH。

注意事项：

① 当遇到两个字相除时，要先执行 CWD指令，以便产生一个长为 32位的被除数。

② 该指令不影响标志位。

例 6.30] 以 BUFFER开始的缓冲区中，前两个字节是一个 16位带符号的被除数，接着两个字节是一个 16位带符号的除数，再接着的 4个字节分别存放商和余数。程序段如下：

```
LEA     BX, BUFFER
MOV     AX, [BX]
CWD
IDIV    [BX+2]
MOV     [BX+4], AX
MOV     [BX+6], DX
```

5. 十进制调整指令

前面介绍过，在计算机中，可用 4位二进制码表示 1个十进制码，这种代码叫 BCD码。BCD码有两类：一类叫压缩的 BCD码，所谓压缩，就是用 1个字节表示 2位 BCD码；另一类叫非压缩的 BCD码，就是 1个字节只用低 4位来表示 BCD码，高 4位为 0。如 2468用压缩的 BCD码表示为：0010 0100 0110 1000。而用非压缩的 BCD码表示为：00000010 00000100 00000110 00001000。根据这两种表示法，相应地将十进制调整指令分为两组：压缩的 BCD码调整指令和非压缩的 BCD码调整指令。

(1) 压缩的 BCD码调整指令

指令一——加法的十进制调整指令 DAA (DecimalAdjustForAddition)

指令格式：DAA

指令功能：若 AF＝1或 AL寄存器的低 4位是十六进制数的 A～F，则 AL寄存器内容加 06H，且将 AF置 1；若 CF＝1或 AL寄存器的高 4位是十六进制数的 A～F，则 AL寄存器内容加 60H，且将 CF置 1。

注意事项：对 OF标志无定义，但影响其他标志。

例 6.31] 若 (AL)＝34H，(BL)＝88H执行下面指令：

```
ADD     AL, BL
DAA
```

则执行 ADD指令后，(AL)＝0BCH，且 CF＝0、AF＝0

而执行 DAA指令后，(AL)＝22H，且 CF＝1、AF＝1

例 6.32] 设有两个多字节数（每一个是 16位十进制数），分别存放在以 BLOCK1和 BLOCK2开始的内存单元中，存放时都是高字节在高地址单元中，要求实现将 BLOCK1和 BLOCK2内容相加，加得结果送 BLOCK3开始的单元中。

```
L1: LEA     BX, BLOCK1
    LEA     SI, BLOCK2
    LEA     DI, BLOCK3
    MOV     CX, 8 ;一个字节可表两位十进制数，16位十进制数用 8个字节表示
    CLC           ;在最低字节相加前，要使进位标志 C清 0
```

```
L1:MOV   AL, [BX]
   ADC   AL, [SI]
   DAA
   MOV   [DI], AL
   INC   BX
   INC   SI
   INC   DI
   DEC   CX
   JNZ   L1
   HLT
```

指令二——减法的十进制调整指令 DAS (Decimal Adjust For Subtraction)

指令格式：DAS

指令功能：若 AF＝1或 AL寄存器的低 4位是十六进制的 A～F，则 AL寄存器内容减去 06H，且将 AF置 1；若 CF＝1或 AL寄存器的高 4位是十六进制的 A～F，则 AL寄存器内容减去 60H，且将 CF置 1。

注意事项：对 OF标志无定义，但影响其他标志。

例 6.33　若 (AL)＝13H，(BL)＝45H执行下面指令：

```
SUB  AL, BL
DAS
```

则执行 SUB指令后，(AL)＝0CEH，且 CF＝1、AF＝1

而执行 DAS指令后，(AL)＝68H，且 CF＝1、AF＝1

(2) 非压缩的 BCD码调整指令

指令一——加法的 ASCII调整指令 AAA (ASCII Adjust For Addition)

指令格式：AAA

指令功能：若 AL寄存器的低 4位在 0～9之间，且 AF＝0，则将 AF值送 CF即可；

若 AL寄存器的低 4位在十六进制数 A～F之间或 AF＝1，则将 AL寄存器内容加 6，AH寄存器内容加 1，并将 AF和 CF位置 1，然后将 AL的高 4位清 0，从而将 AL的内容调整为 0～9之间的数。

注意事项：影响 AF和 CF标志，对其他标志位无定义。

例 6.34　若 (AX)＝0645H，(BL)＝48H执行下面指令：

```
ADD   AL, BL
AAA
```

则执行 ADD指令后，(AL)＝8DH，AF＝0

而执行 AAA指令后，(AX)＝0703H，且 AF＝1、CF＝1

指令二——减法的 ASCII调整指令 AAS (ASCII Adjust For Subtraction)

指令格式：AAS

指令功能：若 AL寄存器的低 4位在 0～9之间，且 AF＝0，则将 AF值送 CF即可；

若 AL寄存器的低 4位在十六进制数 A～F之间，或 AF＝1，则将 AL寄存器内容减去 6，AH

寄存器内容减去 1,并将 AF和 CF位置 1,然后将 AL的高四位清 0,从而将 AL的内容调整为 0～9之间的数。

注意事项 影响 AF和 CF标志,对其他标志位无定义。

指令三——乘法的 ASCII调整指令 AAM (ASCII Adjust For Multiply)

指令格式 :AAM

指令功能 :把 AL的内容除以 10,商存放到 AH中,余数存放到 AL中。

注意事项 根据 AL寄存器的内容设置 SF、ZF和 PF,对其他标志位无定义。

例 6.35] 若 (AL)=06H,(BL)=08H执行下面指令:

```
MUL     BL
AAM
```

则执行 MUL指令后,(AL)=30H

而执行 AAM指令后,(AH)=04H,(AL)=08H

指令四——除法的 ASCII调整指令 AAD (ASCII Adjust For Division)

指令格式 :AAD

指令功能 :(AL)←(AH)*10+(AL),(AH)←0

注意事项 :

① 该指令与其他调整指令在使用方法上是不同的,加减乘法调整在运算后进行,而除法调整应在除法运算之前。

② 根据 AL寄存器的内容设置 SF、ZF和 PF,对其他标志位无定义。

例 6.36] 若 (AX)=0906H执行下面指令:

```
AAD
```

则执行 AAD指令后,(AL)=09*10+06=60H,(AH)=0

所以 (AX)=0060H

6.3.3 逻辑指令

主要包括逻辑运算指令和移位指令两大类。

1. 逻辑运算指令

8086的逻辑运算指令包括逻辑与指令 AND、逻辑或指令 OR、逻辑非指令 NOT、异或指令 XOR和测试指令 TEST。这 5条指令除 NOT指令不影响标志位外,其他 4条指令都使 CF=OF=0,对 AF无定义,而 SF、ZF和 PF则根据结果而定。

(1) 逻辑与指令 AND

指令格式 :AND 目的操作数,源操作数

指令功能 :目的操作数内容∧源操作数内容→目的操作数

注意事项 :

① 操作规则是全 1为 1,有 0为 0。

② 自身相与,清进位标志,但结果不变。

③ AND指令可使操作数的某些位清 0,其他位不变。只需将清 0的位和 0相与,不变的位和 1相与即可。

例 6.37】 把 AX中的第 1、4、8、14位内容清 0,其他位不变。指令为：

AND AX,0BEEDH

(2) 逻辑或指令 OR

指令格式：OR 目的操作数,源操作数

指令功能：目的操作数内容∨源操作数内容→目的操作数

注意事项：

① 操作规则是全 0为 0,有 1为 1。

② 自身相或,清进位标志,但结果不变。

③ OR指令可使操作数的某些位置 1,其他位不变。只需将置 1的位和 1相或,不变的位和 0相或即可。

例 6.38】 把 AX中的第 2、5、7、12位内容置位 1,其他位不变。指令为：

OR AX,10A4H

(3) 逻辑非指令 NOT

指令格式：NOT 操作数

指令功能：将操作数的内容按位取反。

注意事项：操作规则是 0变为 1,1变为 0。

(4) 逻辑异或指令 XOR

指令格式：XOR 目的操作数,源操作数

指令功能：目的操作数内容P 源操作数内容→目的操作数

注意事项：

① 操作规则：相同为 0,不同为 1。

② 自身相异或,结果为 0,进位标志为 0。

③ XOR指令可使操作数的某些位取反,其他位不变。只需将取反的位和 1相异或,不变的位和 0相异或即可。

例 6.39】 把 AX中的第 3、6、9、15位内容取反,其他位不变。指令为：

XOR AX,8248H

(5) 测试指令 TEST

指令格式：TEST 目的操作数,源操作数

指令功能：目的操作数内容∧源操作数内容,但结果不回送。

注意事项：TEST指令的源操作数一般设置为立即数,其中要测试目的操作数的哪一位,就相应地令源操作数的哪一位为 1,其他位为 0。

例 6.40】 测试 AL中第 0位是否为 1,如为 1则转到 L1。指令为：

TEST AL,00000001B

JNZ L1

2. 移位指令

包括逻辑左移指令 SHL(Shift Logical Left)、算术左移指令 SAL(Shift Arithmetic Left)、逻辑右移指令 SHR(Shift Logical Right)、算术右移指令 SAR(Shift Arithmetic Right);循环左移指令 ROL(Rotate Left)、循环右移 ROR指令(Rotate Right)、带进位循环左移指令 RCL(Rotate Left

Through Carry)、带进位循环右移指令 RCR (Rotate Right Through Carry)。

指令格式 操作码　操作数,移位次数

指令功能 对操作数按不同的操作码进行移位,见图 6-3-6。

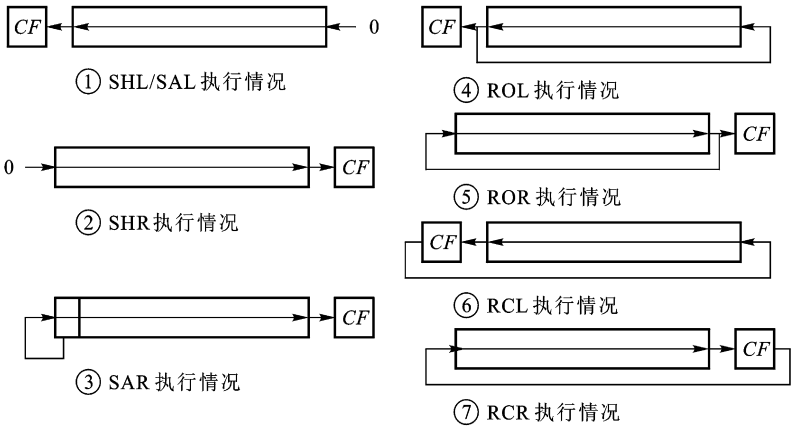


图 6-3-6　移位指令执行情况

注意事项：

① 移位次数为 1时直接写在指令中,大于 1时把移位次数放在 CL寄存器中。

② 逻辑左移一位相当于无符号数乘 2,逻辑右移一位相当于无符号数除 2;算术左移一位相当于有符号数乘 2,算术右移一位相当于有符号数除 2。

例 6.41]　若 (AL) =9CH, CF=1,执行下面指令：

```
MOV     CL, 2
SHL     AL, CL ; (AL) =70H
SHR     AL, CL ; (AL) =27H
SAL     AL, CL ; (AL) =70H
SAR     AL, CL ; (AL) =0E7H
ROL     AL, CL ; (AL) =72H
ROR     AL, CL ; (AL) =27H
RCL     AL, CL ; (AL) =73H
RCR     AL, CL ; (AL) =67H
```

例 6.42]　将 AL寄存器中的有符号数乘以 10,程序段如下：

```
SAL     AL, 1
MOV     BL, AL
SAL     AL, 1
SAL     AL, 1
ADD     AL, BL
```

6.3.4 串处理指令

串处理指令就是用一条指令实现对一串字符或数据的操作。主要包括：串传送指令 MOVS、串比较指令 CMPS、串扫描指令 SCAS、从串中取指令 LODS以及存入串指令 STOS。

串处理指令有如下特点：

① 串操作指令为双操作数指令，两个操作数分别为源串和目的串。

② 源串的地址由源变址寄存器 SI指示，且假定在数据段 DS中存放；目的串的地址由目的变址寄存器 DI指示，且假定在附加数据段 ES中存放。

③ 每次串操作后，自动修改指针。若标志位 DF＝0（执行 CLD指令），则每次操作后 SI、DI增量；若 DF＝1（执行 STD指令），则每次操作后 SI、DI减量。

④ 可以处理字节串，也可以处理字串。若处理字节串，则 SI、DI增减量为 1；若处理字串，则 SI、DI增减量为 2。

⑤ 串处理指令可以在前面加一个重复前缀，重复前缀有：REP、REPZ/REPE和 REPNZ/REPNE。一般来说，REP常与 MOVS、LODS、STOS连用，表示连续传送、存取；REPE/REPZ及 REPNE/REPNZ常与 CMPS、SCAS连用，表示连续查找、扫描。

REP（Repeat）重复，重复串操作直到（CX）＝0为止。

REPE/REPZ（RepeatWhile Equal/Zero）相等/为零则重复，重复串操作直到（CX）＝0或 ZF＝0（即某次比较的结果不等）时退出。

REPNE/REPNZ（RepeatWhile Not Equal/Not Zero）不相等/不为零则重复，重复串操作直到（CX）＝0或 ZF＝1（即某次比较的结果相等）时退出。

1. 串传送指令 MOVS（Move Byte orWord String）

指令格式：OVS 目的操作数，源操作数

```
MOVSB                          ;字节串传送
MOVSW                          ;字串传送
```

指令功能：[DS（0）＋SI] → [ES（0）＋DI]

SI＝SI±1，DI＝DI±1（字节操作）；SI＝SI±2，DI＝DI±2（字操作）

若 DF＝0时，使用“＋”号；若 DF＝1时，使用“－”号

例 6.43 实现以 FIRST单元开始的 200个字节数据传送到 SECOND开始的单元中，程序段如下：

方法一：

```
    OV    SI,OFFSET FIRST
    MOV   DI,OFFSET SECOND
    MOV   CX,200
L1: MOV   AL,[SI]
    MOV   [DI],AL
    INC   SI
    INC   DI
    DEC   CX
    JNZ   L1
```

```
        HLT
方法二：MOV     SI,OFFSET FIRST
        MOV     DI,OFFSET SECOND
        MOV     CX,200
        CLD
     L1：MOVSB
        DEC     CX
        JNZ     L1
        HLT
方法三：MOV     SI,OFFSET FIRST
        MOV     DI,OFFSET SECOND
        MOV     CX,200
        CLD
        REP     MOVSB
        HLT
```

2. 串比较指令 CMPS (Compare Byte or Word String)

指令格式：MPS 目的操作数,源操作数

```
        CMPSB                          ;字节串传送
        CMPSW                          ;字串传送
```

指令功能：[DS (0) ＋SI] － [ES (0) ＋DI],但结果不回送

SI＝SI±1,DI＝DI±1 (字节操作)；SI＝SI±2,DI＝DI±2 (字操作)

若 DF＝0时,使用“＋”号；若 DF＝1时,使用“－”号

例 6.44] 比较以 AREA1和 AREA2单元开始的 30个字节数据是否相同,如果相同则 AREA3单元内容为 0,不同则为 0FFH,程序段如下：

```
   L1：  OV     SI,OFFSET AREA1
      MOV     DI,OFFSET AREA2
      MOV     BX,OFFSET AREA3
      MOV     CX,30
      CLD
      REPZ    CMPSB
      JNZ     L1
      MOV     [BX],0
      JMP     L2
   L1：MOV     [BX],0FFH
   L2：HLT
```

3. 串扫描指令 SCAS (Scan Byte or Word String)

指令格式：CAS 目的操作数

```
        SCASB                          ;字节串传送
```

SCASW ;字串传送

指令功能：(AL) － [ES (0) ＋DI]，DI=DI±1 (字节操作)；

(AX) － [ES (0) ＋DI]，DI=DI±2 (字操作)

若 DF＝0时，使用“＋”号；若 DF＝1时，使用“－”号

例 6.45] 设 ES段中自 BLOCK单元存有 30个字符数，要求查找其中是否有空格符，若有则将第一个被找到的字符在 ES段中的偏移量送到 CX中，否则 CX值为 0。

```
L1:  OV    DI, OFFSET BLOCK
     MOV   CX, 30
     MOV   AL, 20H
     CLD
     REPNZ SCASB
     JZ    L1
     MOV   CX, 0
     JMP   L2
L1:  DEC   DI
     MOV   CX, DI
L2:  HLT
```

4. 从串中取指令 LODS (Load Byte or Word String)

指令格式：ODS 源操作数

LODSB ;字节串传送

LODSW ;字串传送

指令功能：[DS (0) ＋SI] →AL, SI=SI±1 (字节操作)；

[DS (0) ＋SI] →AX, SI=SI±2 (字操作)

若 DF＝0时，使用“＋”号；若 DF＝1时，使用“－”号

5. 存入串指令 STOS (Store Byte or Word String)

指令格式：TOS 目的操作数

STOSB ;字节串传送

STOSW ;字串传送

指令功能：(AL)→ [ES (0) ＋DI]，DI=DI±1 (字节操作)；

(AX)→ [ES (0) ＋DI]，DI=DI±2 (字操作)

若 DF＝0时，使用“＋”号；若 DF＝1时，使用“－”号

例 6.46] 设内存缓冲区中有一长度为 300的数据块，起始地址为 BUFFER，要求把这一数据区中的正数存放在 STR1中，负数存放在 STR2中。

```
L1:  EA    SI, BUFFER
     LEA   DI, STR1
     LEA   BX, STR2
     MOV   CX, 300
     CLD
```

```
L1: LODSB
    TEST    AL, 80H
    JNZ     L2
    STOSB
    JMP     L3
L2: XCHG    BX, DI
    STOSB
    XCHG    BX, DI
L3: DEC     CX
    JNZ     L1
    HLT
```

6.3.5 控制转移类指令

在 8086系统中,指令执行的顺序由代码段寄存器 CS和指令指针 IP的内容决定。在程序顺序执行时,CPU执行完一条指令后 IP自动增量执行下一条指令;但是在程序非顺序执行时,则由控制转移指令改变代码段寄存器 CS和指令指针 IP的内容,CPU根据新的 CS和 IP的值执行指令,从而实现程序转移。8086有 5种控制转移指令:无条件转移指令、条件转移指令、循环指令、子程序及中断指令和处理机控制指令。

1. 无条件转移指令

无条件跳转指令 JMP(Jump),共有 5种格式:

① 段内直接短转移:

指令格式:JMP SHORT 目标地址

指令功能 转移 IP地址由当前 IP加 8位位移量形成,只能在段内 -128B~127B范围内转移。

② 段内直接近转移:

指令格式:JMP NEAR PTR 目标地址

指令功能 转移的 IP值由当前 IP加 16位位移量形成,可以转移到代码段内任何位置。

③ 段内间接转移:

指令格式:JMP WORD PTR 目标地址

指令功能 转移的 IP值位于一通用寄存器中或由寻址方式确定的有效地址中。

④ 段间直接转移:

指令格式:JMP FAR PTR 目标地址

指令功能 转移时 CS和 IP值是目标地址所在的段地址和段内偏移量。

⑤ 段间间接转移:

指令格式:JMP DWORD PTR 目标地址

指令功能 转移地址是存储器中的一个双字,高位字是转移地址的 CS值,低位字是转移地址的 IP值。

例 6.47] 转移指令举例。

```
JMP  SHORT  PTR  L1      ;段内短转移
JMP  NEAR   PTR  L2      ;段内直接转移
JMP  WORD   PTR  BX      ;段内间接转移
JMP  FAR    PTR  L3      ;段间直接转移
JMP  DWORD  PTR  [BX]    ;段间间接转移
JMP  DWORD  PTR  [BX+SI] ;段间间接转移
```

2. 条件转移指令

条件转移指令根据上一条指令所设置的标志位来判别测试条件,满足测试条件则转移到由指令指出的转向地址处执行程序,不满足测试条件则顺序执行下一条指令。

(1) 单个标志位的条件转移指令 (如表 6-1所示)

表 6-1 单个标志位的条件转移指令

标志	指令助记符	测试条件	指令功能
ZF	JZ/JE	ZF=1	结果为 0或相等则转移
	JNZ/JNE	ZF=0	结果不为 0或不相等则转移
CF	JC	CF=1	有进位 (或借位)则转移
	JNC	CF=0	无进位 (或借位)则转移
SF	JS	SF=1	结果为负则转移
	JNS	SF=0	结果为正则转移
OF	JO	JO=1	结果有溢出则转移
	JNO	JO=0	结果无溢出则转移
PF	JP/JPE	JP=1	结果中 1的个数为偶数则转移
	JNP/JPO	JP=0	结果中 1的个数为奇数则转移

(2) 判断无符号数大小条件转移指令 (如表 6-2所示)

表 6-2 判断无符号数大小条件转移指令

指令助记符	指令功能
JA/JNBE	大于/不小于等于则转移
JAE/JNB	大于等于/不小于则转移
JB/JNAE	小于/不大于等于则转移
JBE/JNA	小于等于/不大于则转移

(3) 判断有符号数大小条件转移指令 (如表 6-3所示)

表 6-3 判断有符号数大小条件转移指令

指令助记符	指令功能
JG/JNLE	大于/不小于等于则转移
JGE/JNL	大于等于/不小于则转移
JL/JNGE	小于/不大于等于则转移
JLE/JNG	小于等于/不大于则转移

⑷ CX条件转移指令 JCXZ

指令格式：JCXZ　　目标标号

指令功能：当 CX为 0时转移，CX非 0时不转移。

JCXZ指令相当于：MP　　CX, 0

JZ　　标号

3. 循环指令

8086的循环指令主要包括 LOOP、LOOPE/LOOPZ和 LOOPNE/LOOPNZ。

⑴ 指令格式：LOOP　　标号

指令功能：循环次数（CX）－1→CX，若 CX＝0则退出循环，若 CX≠0则循环。LOOP指令相当于：

DEC　　CX

JNZ　　标号

⑵ 指令格式：LOOPE/LOOPZ　　标号

指令功能：循环次数（CX）－1→CX，若 CX＝0则退出循环，若 CX≠0且 ZF＝1（结果为0或相等时）则循环。

⑶ 指令格式：LOOPNE/LOOPNZ　　标号

指令功能：循环次数（CX）－1→CX，若 CX＝0则退出循环，若 CX≠0且 ZF＝0（结果不为 0或不相等时）则循环。

例 6.48　自 ARRAY开始的内存缓冲区中，有 80个 16位有符号数，找出最大值，存放在MAX存储单元中。

```
L1:  OV    BX, OFFSET ARRAY
     MOV   DI, OFFSET  MAX
     MOV   AX, [BX]
     INC   BX
     INC   BX
     MOV   CX, 79
L1:  CMP   AX, [BX]
     JG    L2
     MOV   AX, [BX]
L2:  INC   BX
     INC   BX
     LOOP  L1
     MOV   [DI], AX
     HLT
```

4. 子程序及中断指令

⑴ 子程序调用指令 CALL

CALL指令用于暂停正在执行的主程序，转去执行相应的子程序，子程序执行完后再返回到主程序中，所以要把 CALL指令的下一条指令的 CS和 IP值入栈保护。

① 段内直接调用

指令格式:CALL 直接地址 或 CALL NEAR PTR 标号

指令功能:首先将断点的 IP值压入堆栈,再将子程序的地址偏移量加到当前 IP上,然后根据 IP转到相应子程序执行。

② 段内间接调用

指令格式:CALL 寄存器 或 CALL WORD PTR 存储器

指令功能:首先将断点的 IP值压入堆栈,再将子程序的地址偏移量送入 IP,然后根据 IP转到相应子程序执行。

③ 段间直接调用

指令格式:CALL FAR PTR 标号

指令功能:首先将断点的 CS值压入堆栈,并将子程序的段地址送入 CS;再把断点的 IP值压入堆栈,把子程序的地址偏移量送入 IP,然后根据 CS:IP值转到相应子程序执行。

④ 段间间接调用

指令格式:CALL DWORD PTR 存储器

指令功能:首先将断点的 CS值压入堆栈,并将指令中指定的双字存储器的第二个字的内容送入 CS;再把断点的 IP值压入堆栈,然后把双字存储器的第一个字的内容送入 IP,最后根据 CS:IP值转到相应子程序执行。

[例 6.49] 各种子程序调用指令的功能。

```
CALL    NEAR PTR L1          ;段内直接调用
CALL    1000H                ;段内直接调用
CALL    BX                   ;段内间接调用
CALL    WORD PTR [BX]        ;段内间接调用
CALL    FAR PTR L2           ;段间直接调用
CALL    DWORD PTR [BX]       ;段间间接调用
CALL    DWORD PTR [BX+SI]    ;段间间接调用
```

(2) 子程序返回指令 RET

RET与 CALL相对应,通常作为一个子程序的最后一条指令,用以返回到调用这个子程序的主程序断点处继续执行。

① 直接返回指令

指令格式:RET

指令功能:若是段内的 RET指令,只返回主程序断点处的 IP值,即从堆栈中弹出一个字送 IP;若是段间的 RET指令,则要返回主程序断点处的 CS和 IP值,即从堆栈中弹出一个字送 IP,再从堆栈中弹出一个字送 CS。

② 带立即数返回指令。

指令格式:RET n

指令功能:先执行与 RET相同的操作,再修改 SP,使 SP+n→SP,n为一个十六进制的立即数,通常是偶数,表示返回时从堆栈中舍弃的字节数。

(3) 中断指令

① 中断调用指令 INT

指令格式：INT　n；n为中断类型号，取值范围为 0～255。

指令功能：把 PSW、CS、IP寄存器内容依次压入堆栈，并根据中断类型号从中断向量表中取出连续 4个字节的内容赋予 IP和 CS，转到中断服务程序执行。

② 溢出中断指令 INTO

指令格式：INTO

指令功能：若 OF＝1则启动一个中断类型号为 4的中断过程，否则不中断。

③ 中断返回指令 IRET

指令格式：IRET

指令功能：放在中断服务程序的最后，用于返回主程序，同时从堆栈中恢复 IP、CS、PSW寄存器的内容，返回原断点执行程序。

6.3.6　处理机控制指令

处理机控制指令用来控制处理机的某些功能，包括调整状态标志位、使 8086 CPU与外部事件同步及空操作指令。该类指令无操作数。

1.标志处理指令

① CLC (Clear Carry Flag)：进位位清零，(CF)←0

② CMC (Complement Carry Flag)：进位位求反，(CF)←$\overline{(CF)}$

③ STC (Set Carry Flag)：进位位置位，(CF)←1

④ CLD (Clear Direction Flag)：方向标志清零，(DF)←0

⑤ STD (Set Direction Flag)：方向标志置位，(DF)←1

⑥ CLI (Clear Interrupt Enable Flag)：中断标志清零，(IF)←0

⑦ STI (Set Interrupt Enable Flag)：中断标志置位，(IF)←1

2. 其他处理机控制指令

① HLT (Processor Halt)：暂停，使 8086 CPU进入暂停状态，除非复位或有中断发生才退出暂停状态。

② ESC (Processor Escape)：ESC又称换码指令、交权指令，把控制权交给协处理器，即把存储单元的内容送到数据总线，由协处理器获得该内容。

③ WAIT (Processor Wait)：等待，检测$\overline{TEST}$状态，若$\overline{TEST}$无效（高电平），则等待，否则执行 WAIT指令后的下一条指令。

④ LOCK：总线封锁，LOCK可加在任何指令前作为前缀，使得处理机在执行该指令期间保持总线锁定信号$\overline{LOCK}$有效，把总线封锁，使别的主设备不能控制总线，以便实现多处理机对资源共享的要求。

⑤ NOP (No Operation)：空操作，NOP使 CPU不执行任何操作，除非响应外中断。放在程序中有两个作用：一是让它占有一定的存储单元，以便以后用其他指令代替；二是可以起到延时的作用。

6.4 汇编语言程序设计基础

汇编语言程序的语句除前面介绍的机器指令外，还有伪指令，并且汇编语言源程序要想正确执行，还要遵守一些规则和约定，如语句格式、程序格式、参数表示、符号定义、内存分配等，本节将对这些汇编语言程序设计的基础知识进行介绍。

6.4.1 伪指令

单纯由机器指令不能形成完整程序，需要一些辅助语句来组织指令和数据，这些辅助语句就是伪指令。伪指令语句是说明性语句，告诉汇编程序如何工作，主要完成数据定义、分配存储区、指示程序结束等功能。伪指令不像机器指令，它没有对应的机器码，汇编后不产生目标代码，只是在对程序汇编时起作用。

1. 数据定义伪指令

数据定义伪指令包括 DB、DW、DD、DQ、DT，用于定义变量及分配存储区。

指令格式：[变量名] 数据定义伪指令，操作数项

指令功能：用于定义变量的类型、给存储器赋初值或给变量分配存储单元。

注意事项：

① 方括号中的变量名为任选项，变量名后面不跟冒号。

② DB定义字节类型变量，DW定义字类型变量，DD定义双字类型变量，DQ定义 4字类型变量，DT定义 10字类型变量。

③ 操作数的值不能超出相应数据类型限定的取值范围。

④ 操作数项可以包括多个数据，它们之间用逗号隔开。操作数项可以是常数表达式、地址表达式（仅适用 DW、DD）、字符串（超过 2个字符仅用 DB）、问号（只分配存储单元，而不赋值）、重复子句 DUP。

【例 6.50】 下面是不正确的定义：

```
BUF1: DB  90H, 80H      ;变量不能有冒号
BUF2  DB  1000          ;操作数项超过了字节数的取值范围
BUF3  DD  'HELLO'       ;超过 2个字符的字符串只能用 DB定义
BUF4  DB  BLOCK         ;地址表达式只能用 DW或 DD定义
```

如将 DB改为 DW，则操作数项表示取 BLOCK的偏移地址；如将 DB改为 DD，则操作数项表示取 BLOCK的偏移地址和段地址，且第一个字为偏移地址，第二个字为段地址。

【例 6.51】 画出下列存储单元分配示意图，如图 6－4－1所示。

① DATA1 DB 20, 20H, 'ABC', 2 DUP (?, 2 DUP (0))

② DATA2 DW 'DE', 2 DUP (3, ?)

③ DATA3 DD ?, 5*10, －9

2. 赋值伪指令

指令格式：名字 EQU 表达式

指令功能：将表达式的值赋予一个名字，以后可以用这个名字来代替对应的表达式。

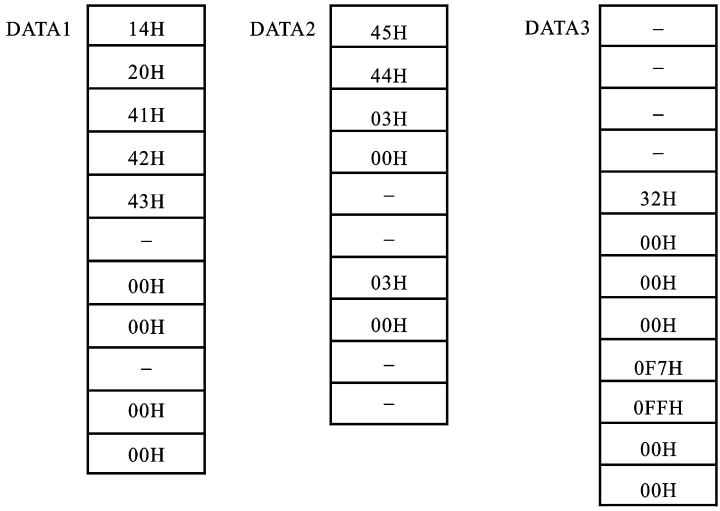


图 6－4－1　例 6.51中指令执行情况

注意事项：

① 表达式可以是一个常数、符号、数值表达式或地址表达式。

② 已赋值的名字可以在以后的赋值语句中引用。如下面两条语句执行后 TAB2的值为 6。

```
TAB1    EQU    9
TAB2    EQU    TAB1－3
```

③ EQU伪指令的功能类似于等号“＝”的功能，区别在于由 EQU 赋值的名字不可重复赋值，而由“＝”赋值的名字可以重复赋值。如：

```
STR＝1
STR＝STR＋1
```

3. 段定义伪指令

⑴ 完整段定义伪指令

指令格式：段名　　SEGMENT

　　…

　　段名　　ENDS

指令功能：SEGMENT和 ENDS成对使用，用来定义一个段的开始和结束。

注意事项：

① SEGMENT和 ENDS前面的段名必须一致。

② SEGMEN和 ENDS之间的语句是定义段的内容。

③ 可以定义代码段、数据段、附加数据段和堆栈段，代码段主要是程序指令和某些伪指令；数据段和附加数据段用于定义数据和存储单元；堆栈段为堆栈操作预留出存储空间。

⑵ 指定段寄存器伪指令

指令格式：ASSUME　　段寄存器名：段名［,段寄存器名：段名］

指令功能：由于段名是用户定义的，所以要指明哪个段名对应哪个段寄存器。

注意事项：段寄存器名必须是 CS、DS、ES、SS中的一个，而段名则必须是由 SEGMENT定义的段名，并且用 SEGMENT定义了几个段，ASSUME 伪指令就需要指明几个段。

4. 过程定义伪指令

过程定义伪指令用来定义一个子程序，子程序又称过程，在主程序中由 CALL 指令调用，调用结束将返回到主程序中 CALL指令的下一条指令继续执行，而子程序中必须有一条返回指令 RET。

指令格式：过程名　PROC　类型

…

过程名　ENDP

指令功能：PROC和 ENDP成对使用，用来定义一个过程的开始和结束。

注意事项：

① PROC和 ENDP前面的过程名必须一致。

② PROC和 ENDP之间的语句是定义过程的内容。

③ 过程的类型有 NEAR和 FAR两种，在本段内调用的过程是 NEAR过程，可在不同段间调用的过程是 FAR过程，定义时若不指定类型，缺省为 NEAR类型。

5. 模块定义伪指令

在汇编语言中每一个独立的源程序称为一个模块，在源程序的开始可以用 NAME或 TITLE伪指令为模块命名，而源程序结束使用 END伪指令。

指令格式：AME　模块名

TITLE　模块名

END　[标号]

指令功能：

① NAME伪指令可以缺省，如果缺省 NAME指令，汇编程序以 TITLE指令中前 6个字符作为模块名。

② TITLE伪指令用于给程序一个标题，列表文件中每一页的第一行都会显示这个标题，它是用户任意选定的字符串，但是字符的个数不能超过 60。TITLE指令也可以缺省，如果 NAME和 TITLE都缺省，则以源文件名作为模块名。

③ END伪指令中的标号指出程序开始执行的第一条指令的地址，END不能缺省。

6. 偏移地址设置伪指令

指令格式：ORG n

指令功能：为指令或数据设置由 n开始的偏移地址。

注意事项：n的取值范围是 0～65 535。

7. 地址计数器伪指令

指令格式：$

指令功能：在汇编程序内，为了指示下一个数据或指令在相应段中的偏移量，汇编程序使用了一个当前位置计数器 $。

[例 6.52]　执行下面程序段后，(CX)=12H。

```
STR1  DW  'AB'
STR2  DB  16 DUP(?)
CNT   EQU  $-STR1
MOV   CX, CNT
```

6.4.2 汇编语言语句格式

在汇编语言源程序中每个语句可由 4项组成,格式如下:

[名字项]　操作项　操作数项　[注释项]

1. 名字项

(1) 名字项的组成规则

① 组成名字的字符可以是:A～Z、a～z、0～9、?、·、@、$、_

② 名字中不能以数字开头;问号本身不能单独作为名字;如果用到“·”则必须是第一个字符;名字的最大长度为 31,若超过则后续字符无效。

(2) 名字项的构成

名字项可以是标号和变量。标号在代码段中定义,后跟冒号,用以表示转向地址;变量在除代码段以外的其他段中定义,后不跟冒号。

2. 操作项

操作项可以是指令、伪指令的助记符。对于指令,汇编程序将其翻译为机器语言;对于伪指令,汇编程序将根据其所要求的功能进行处理。

3. 操作数项

操作数项可以是常量、变量、表达式和标号。常量就是指令中出现的一些固定值,可分为数值常量和字符串常量;变量是存放在存储单元或寄存器中的数据;表达式是由常数、变量通过操作运算符连接而成的,这些操作运算符主要分为:算术运算符、逻辑运算符和关系运算符等;标号则表示机器指令的符号地址。

(1) 算术运算符

算术运算符有:+、-、*、/、MOD(取余)

[例 6.53]　把首地址为 ARRAY的字数组的第 5个字传送到 AX寄存器中。指令如下:

```
MOV    BX,OFFSET  ARRAY
MOV    AX, [BX + (5-1)*2]
```

(2) 逻辑运算符

逻辑运算符有:AND、OR、NOT、XOR

[例 6.54]　`AND  AX,35H AND 0FH`

第一个 AND是逻辑与指令,第二个 AND是逻辑运算符。

(3) 关系运算符

关系运算符有:EQ(等于)、NE(不等于)、LT(小于)、GT(大于)、LE(小于等于)、GE(大于等于)

参加运算的必须是两个数值或同一段内的两个存储单元地址。计算结果应为逻辑值,当关系式成立时,结果为真,用全 1表示;当关系式不成立时,结果为假,用 0表示。

例 6.55] MOV AX,5 GT 2

因为 5>2关系式成立,所以结果为真,即 (AX) =0FFFFH。

(4) 分析运算符

分析运算符有 :TYPE、LENGTH、SIZE、OFFSET、SEG

① TYPE 指令

指令格式 :TYPE 变量或标号

指令功能 :如果是变量,则汇编程序将回送该变量的类型值 :DB为 1, DW 为 2, DD为 4, DQ为 8, DT为 10如果是标号,则汇编程序将回送代表该标号类型的数值 :NEAR为 -1, FAR为 -2。

② LENGTH 指令

指令格式 :LENGTH 变量

指令功能 :对于变量中使用 DUP的情况,则返回外层 DUP的给定值 如果没有使用 DUP,则返回值是 1。

③ SIZE 指令

指令格式 :SIZE 变量

指令功能 :汇编程序将返回分配给该变量的字节数,此值是 LENGTH 和 TYPE 的乘积。

例 6.56] 分析运算符功能举例。

```
ONE     DB 'GOOD'
TWO     DW  30 DUP (?)
THREE   DD  1,2
MOV     AH, TYPE  ONE       ; (AH) =1
MOV     AL, TYPE  TWO       ; (AL) =2
MOV     BH, TYPE  THREE     ; (BH) =4
MOV     BL, LENGTH  ONE     ; (BL) =1
MOV     CH, LENGTH  TWO     ; (CH) =30
MOV     CL, LENGTH  THREE   ; (CL) =1
MOV     DH, SIZE ONE        ; (DH) =1
MOV     DL, SIZE  TWO       ; (DL) =60
MOV     DI, SIZE  THREE     ; (DI) =4
```

④ OFFSET指令

指令格式 :OFFSET 变量或标号

指令功能 :回送变量或标号的偏移地址值。

⑤ SEG指令

指令格式 :SEG 变量或标号

指令功能 :回送变量或标号的段地址值。

例 6.57] 设符号地址 BLOCK所在的段地址为 1000H,偏移地址为 2000H,则执行下面指令:

```
MOV     SI, OFFSET  BLOCK ; (SI) =2000H
MOV     DI, SEG     BLOCK ; (DI) =1000H
```

(5) 属性运算符

属性运算符有：PTR、THIS、SHORT和段操作符。

① PTR属性运算符

指令格式：类型 PTR 存储器

指令功能：运算符 PTR用于指定存储器操作数的类型，类型可以是 BYTE、WORD、DWORD、NEAR、FAR。

例 6.58] 如下面是使用 PTR运算符的例子：

INC BYTE PTR [BX]

在这条语句中操作数是存储单元，如果只使用 [BX]，则汇编该语句时不能确定存储单元是字节单元还是字单元，所以必须使用 PTR来说明。

② THIS属性运算符

指令格式：THIS 属性或类型

指令功能：建立一个指定类型 (BYTE、WORD、DWORD)或指定距离 (NEAR、FAR)的地址操作数，该操作数的段地址和偏移地址与下一个存储单元地址相同。

例 6.59] 使用 THIS运算符的例子。

DATA1 EQU THIS BYTE

DATA2 DW 30 DUP (0)

在这两条语句中的 DATA1和 DATA2实际上表示同一个数据区地址，由 30个字组成，只是DATA1的类型为字节，而 DATA2的类型为字。

③ SHORT属性运算符

指令格式：SHORT 标号

指令功能：用来指定 JMP指令中转向地址的属性，即指出转向地址是在下一条指令地址的－128～＋127个字节的范围内。

④ 段操作符

指令格式：段寄存器：存储器

指令功能：冒号跟在段寄存器名之后，给一个存储器操作数指定段属性，也称为段超越。

例 6.60] 取出 ES：段内偏移地址由 BX和 SI指定，存储单元内容送 DI的指令为：

MOV DI,ES:[BX+SI]

(6) 运算符的优先级

① ( )，[]，LENGTH、SIZE

② 冒号：

③ PTR、OFFSET、SEG、TYPE、THIS

④ ＊、／、MOD、SHL、SHR

⑤ ＋、－

⑥ EQ、NE、LT、LE、GT、GE

⑦ NOT

⑧ AND

⑨ OR、XOR

⑩ SHORT

4. 注释项

注释项是一个任选字段,在汇编语言语句的最后,它必须以分号开始,如果注释的内容超出一行,则第二行要以分号开始。注释项的内容不影响程序的功能,也不出现在汇编后的机器代码中,只是提高程序的可读性。

6.4.3 汇编语言程序框架

1. 汇编语言程序框架格式

(1) 汇编语言程序框架(格式一):

```
DATA    SEGMENT ;定义数据段
        …
DATA    ENDS
EDATA   SEGMENT ;定义附加数据段
        …
EDATA   ENDS
STACK   SEGMENT ;定义堆栈段
        …
STACK   ENDS
CODE    SEGMENT
        ASSUME    CS:CODE, DS:DATA, ES:EDATA, SS:STACK
START:  MOV       AX, DATA ;给 DS段赋值
        MOV       DS, AX
        MOV       AX, EDATA ;给 ES段赋值
        MOV       ES, AX
        MOV       AX, STACK ;给 SS段赋值
        MOV       SS, AX
        …
        MOV       AH, 4CH ;返回 DOS
        INT       21H
CODE    ENDS
        END START
```

(2) 汇编语言程序框架(格式二):

```
DATA    SEGMENT ;定义数据段
        …
DATA    ENDS
EDATA   SEGMENT ;定义附加数据段
        …
EDATA   ENDS
```

```
STACK   SEGMENT ;定义堆栈段
        …
STACK   ENDS
CODE    SEGMENT
        ASSUME    CS:CODE, DS:DATA, ES:EDATA, SS:STACK
MAIN    PROC      FAR
START:  PUSH      DS ;另一种返回 DOS的方式
        XOR       AX, AX
        PUSH      AX
        MOV       AX, DATA ;给 DS段赋值
        MOV       DS, AX
        MOV       AX, EDATA ;给 ES段赋值
        MOV       ES, AX
        MOV       AX, STACK ;给 SS段赋值
        MOV       SS, AX
        …
        RET
        MAIN      ENDP
CODE    ENDS
        END       START
```

2. 汇编语言程序框架格式说明

① 任何一个汇编语言的源程序,至少应含有一个代码段,其他段视需要而定。

② ASSUME伪指令只是说明了汇编程序中段寄存器与逻辑段之间的关系,并没有给段寄存器赋予实际的初值,所以要在代码段的开始处给各段寄存器赋值(代码段 CS由系统自动赋值),使用了这样的语句:

```
MOV   AX, DATA
MOV   DS, AX
MOV   AX, EDATA
MOV   ES, AX
MOV   AX, STACK
MOV   SS, AX
```

③ 在 IBM-PC DOS环境下运行汇编语言程序,为了使程序执行结束后正常返回 DOS系统,要在代码段中使用如下的指令:

格式一中使用:

```
MOV   AH, 4CH
INT   21H
```

这两条语句是利用 DOS系统功能调用(调用号为 4CH),执行后将由系统结束程序并返回到DOS状态下。

格式二中使用：

```
MAIN    PROC   FAR
START:  PUSH   DS
        XOR    AX, AX
        PUSH   AX
        …
        RET
        MAIN   ENDP
```

为了使用过程编写程序，并能自动返回 DOS，所以要在过程段的开始处，先将 DS入栈，然后再将段内偏移量 0入栈，在过程结束时，执行 RET指令后，就会结束程序并返回 DOS系统。

例 6.61】 把内存中以 BUFF1开始的 20个字母“R”传送到以 BUFF2开始的单元中，写出完整的汇编语言程序。

格式一：

```
DATA    SEGMENT
BUFF1   DB   20 DUP ('R')
DATA    ENDS
EDATA   SEGMENT
BUFF2   DB   20 DUP (?)
EDATA   ENDS
CODE    SEGMENT
        ASSUME     CS:CODE, DS:DATA, ES:EDATA
START:  MOV        AX, DATA
        MOV        DS, AX
        MOV        AX, EDATA
        MOV        ES, AX
        MOV        SI, OFFSET   BUFF1
        MOV        DI, OFFSET   BUFF2
        CLD
        MOV        CX, 20
        REP        MOVSB
        MOV        AH, 4CH
        INT        21H
CODE    ENDS
        END        START
```

格式二：

```
DATA    SEGMENT
BUFF1   DB   20 DUP ('R')
DATA    ENDS
```

```
EDATA   SEGMENT
BUFF2   DB   20 DUP (?)
EDATA   ENDS
CODE    SEGMENT
        ASSUME    CS:CODE, DS:DATA, ES:EDATA
MAIN    PROC      FAR
START:  PUSH      DS
        XOR       AX, AX
        PUSH      AX
        MOV       AX, DATA
        MOV       DS, AX
        MOV       AX, EDATA
        MOV       ES, AX
        MOV       SI, OFFSET  BUFF1
        MOV       DI, OFFSET  BUFF2
        CLD
        MOV       CX, 20
        REP       MOVSB
        RET
MAIN    ENDP
CODE    ENDS
        END       START
```

6.4.4 汇编语言上机过程

在计算机上运行汇编语言程序步骤如下：

① 用编辑程序建立.ASM源文件。

② 用汇编程序把.ASM文件转换成.OBJ文件。

③ 用连接程序把.OBJ文件转换成.EXE文件。

④ 用 DEBUG调试或在 DOS下执行文件。

1. 建立.ASM文件

编辑一个汇编语言源程序，可以使用各种文本编辑软件，如 EDIT、CCED、WPS、TURBO C、TURBO PASCAL等。建立.ASM文件的方法是：

设存放汇编程序的目录是：C:\MASM，文件名为 LIANXI，则在 DOS提示符下键入：C:\MASM>ED LIANXI.ASM (回车)。参考程序如下 (求内存中两个字单元 AREA1和 AREA2内容之和，结果送 AREA3)：

```
DATA    SEGMENT
AREA1   DW      1234H
AREA2   DW      5678H
```

```
AREA3   DW      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     AX, AREA1
        ADD     AX, AREA2
        MOV     AREA3, AX
        MOV     AH, 4CH
        INT     21H
CODE    ENDS
        END     START
```

2. 用汇编程序产生.OBJ文件

用MASM汇编程序对刚建立的LIANXI.ASM文件进行汇编,产生LIANXI.OBJ目标文件,方法是键入:

C:\MASM>MASM LIANXI;(回车)

也可以在文件名后不加";",回车后按提示进行操作。

若在汇编过程中有错误,再用编辑文件进行修改,修改后再重新汇编,直至没有错误为止。此汇编过程是查找源文件中的语法错误,汇编后输出文件有3个:第一个是OBJ目标文件;第二个是LST列表文件,该文件给出源程序和汇编后的机器码;第三个是CRF索引文件。其中后两个文件是可选的,如不需要可直接按回车,不输入文件名。

3. 用连接程序产生.EXE文件

用LINK连接程序对刚建立的XIANXI.OBJ目标文件进行连接,产生XIANXI.EXE可执行文件,方法是键入:

C:\MASM>LINK LIANXI;(回车)

也可以在文件名后不加";",回车后按提示进行操作。

连接程序可以将若干个目标模块连同子程序库的库文件LIB连接在一起。连接后给出的无堆栈段的警告错误(No Stack Segment),并不影响形成可执行文件,也不影响文件的执行。连接后输出文件有两个:第一个是EXE可执行文件,第二个是MAP连接映像文件。

4. 程序的执行

当形成可执行文件后,就可以在DOS下直接输入文件名运行程序。方法是键入:

C:\MASM> LIANXI.EXE (回车)

程序运行结束返回DOS,如果程序中使用了系统调用,就可直接在显示器上把结果显示出来,否则需要使用DEBUG调试程序。

5. DEBUG的运行

在DOS下,键入:C:\MASM>DEBUG LIANXI.EXE (回车)

DEBUG运行后,在屏幕上出现DEBUG的状态提示符短划线"-"。

DEBUG的主要命令有：

⑴ 检查修改寄存器内容命令

格式：-r或-r <寄存器名 > 或-rf

功能：用于显示所有寄存器内容或显示修改指定寄存器内容，rf是显示修改标志寄存器内容。其中，标志寄存器状态如表 6－4所示。

表 6－4 各标志寄存器状态

状态 \ 标志	OF	DF	IF	SF	ZF	AF	PF	CF
0	NV	UP	EI	PL	NZ	NA	PO	NC
1	OV	DN	DI	NG	ZR	AC	PE	CY

⑵ 显示存储单元的命令

格式：-d [地址 1 [地址 2]]

功能：显示指定地址开始的 80H个字节单元内容或显示指定范围内的内容。

⑶ 修改存储单元内容的命令

格式：-e < 地址 > [内容列表]

功能：显示指定地址的内容，并可修改。某字节修改完成或不修改按空格键，整体修改完成或不再修改按回车键。若给出内容列表，则可实现批量修改。

⑷ 填充命令

格式：-f<地址范围 > <内容列表 >

功能：用内容列表来填充地址范围。若列表内容大于地址范围，多余部分忽略；若列表内容小于地址范围，则重复填入。

⑸ 汇编命令

格式：-a [地址]

功能：输入汇编指令，汇编成机器码存入指定单元。

⑹ 反汇编命令

格式：-u [地址 1 [地址 2]]

功能：反汇编出汇编指令的机器码。

⑺ 跟踪命令

格式：-t/p [=地址] 或-t[=地址] [n]

功能：用于从指定地址执行一条或 n条指令，执行后显示所有寄存器的内容。 tp命令的区别是：t命令碰到 CALL指令时会进入子程序，而 p命令会当作普通命令处理。

⑻ 运行命令

格式：-g [=地址] [,地址 [,地址]]

功能：从第一个地址处开始连续执行指令，后面的地址是断点地址。执行后显示所有寄存器的内容。

⑼ 退出命令

格式：-q

功能：结束 DEBUG，返回 DOS。

6.5 汇编语言程序设计

6.5.1 DOS系统功能调用

DOS为用户提供了很多功能调用，即功能子程序，并将这些功能子程序按顺序编号，这个编号称为功能号，通过功能号调用系统提供的相应功能。主要有基本输入输出、磁盘读写控制、文件操作及目录管理以及内存管理等功能。

1. DOS功能调用方式

一般进行 DOS功能调用要做 3方面的工作：

(1) 设置入口参数

DOS功能调用一般都是通过 DL/DX寄存器传送入口参数的，但也有一些功能调用不需要设置入口参数。

(2) 设置功能号

将所需要调用的子程序的功能号送入 AH寄存器。

(3) 执行软中断指令 INT 21H

该指令将程序控制自动转向相应子程序的入口，并执行功能。

2. 常用 DOS功能调用

(1) 带显示的键盘输入 (1号调用)

入口参数：无

调用方式：
```
MOV     AH,01H
INT     21H
```
出口参数：AL中为输入字符 ASCII码。

功能：等待从键盘输入一个字符，并将输入字符的 ASCII码送入 AL寄存器，且在屏幕上显示输入的字符。若输入字符为 Ctrl+Break组合键时，则中断程序执行，返回 DOS。

(2) 显示字符 (2号调用)

入口参数：DL寄存器的内容为要显示字符的 ASCII码。

调用方式：
```
MOV     DL,要显示字符的 ASCII码
MOV     AH,02H
INT     21H
```
出口参数：无

功能：将 DL寄存器中的字符送显示器输出。

(3) 不带显示的键盘输入 (7，8号调用)

操作：同 1号调用

功能：与 1号调用不同的是，键入的字符不在屏幕上显示。且 7号调用对键入的字符不做 Ctrl+Break检查。

(4) 字符串显示 (9号调用)

入口参数 :DX寄存器的内容,为要显示字符串的首地址。

调用方式 : EA　　DX,要显示字符串的首地址

　　MOV　　AH,09H

　　INT　　21H

出口参数 :无

功能 :显示以“$ ”为结束标志的字符串,且字符串应在数据段中。

(5) 字符串输入 (10号调用)

入口参数 :DX寄存器的内容,为缓冲区的首地址。

调用方式 :LEA　　DX,从键盘接收字符的输入缓冲区首地址

　　MOV　　AH,0AH

　　INT　　21H

出口参数 :无

功能 :从键盘输入字符串到指定缓冲区,直到输入回车符为止。

说明 :

① 缓冲区的第一个字节存放缓冲区能接收的字符的个数,其值的范围为 1～255,不能为 0。

② 缓冲区的第二个字节存放实际输入的字符数 (不含回车键),该值由 DOS返回时自动填入。

③ 从缓冲区的第三个字节开始存放输入的字符,若实际输入的字符多于定义的字节数,则多余字符被略去,且响铃报警。

④ 缓冲区一定要定义在当前数据段中。

例 6.62　从键盘输入一个字符,并在显示器上输出,按下 Ctrl+Break组合键时结束,可使用下面两种方法。

方法一 :使用 1号功能调用。

```
MOV　　AH,01H
INT　　21H
```

方法二 :使用 8号和 2号功能调用。

```
MOV　　AH,08H
INT　　21H
MOV　　DL,AL
MOV　　AH,2
INT　　21H
```

例 6.63　使用 9号和 10号功能调用实现人机对话。

```
DATA　　SEGMENT
DAT1　　DB '5+3=','$'
DAT2　　DB '6-2=','$'
BUF1　　DB 5;定义缓冲区 1
　　DB ?
```

```
        DB   5 DUP (?)
BUF2    DB   5;定义缓冲区 2
        DB   ?
        DB   5 DUP (?)
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     DX, OFFSET  DAT1;显示 DAT1信息
        MOV     AH, 09H
        INT     21H
        MOV     DX, OFFSET  BUF1;键盘输入回答 DAT1信息
        MOV     AH, 0AH
        INT     21H
        MOV     DL, 0AH       输出换行符
        MOV     AH, 2
        INT     21H
        MOV     DX, OFFSET  DAT2;显示 DAT2信息
        MOV     AH, 09H
        INT     21H
        MOV     DX, OFFSET  BUF2;键盘输入回答 DAT2信息
        MOV     AH, 0AH
        INT     21H
        MOV     AH, 4CH
        INT     21H
CODE    ENDS
        END     START
```

6.5.2 程序设计结构及举例

用汇编语言编写的程序称为汇编语言源程序,主要有顺序结构程序、分支结构程序、循环结构程序和子程序。

1. 顺序结构程序

按指令书写的先后顺序执行的程序称为顺序结构程序,在程序中没有转移、调用等指令。

例 6.64 求两个字节数之差,并将结果显示出来。

分析:本程序实现两个字节数相减,并将减后的结果显示在显示器上,只需要顺序执行下列指令序列即可。

```
DATA    SEGMENT
```

```
ADR1    DB   68H
ADR2    DB   32H
DATA    ENDS
STACK   SEGMENT
        DB   100 DUP (?)
STACK   ENDS
CODE    SEGMENT
        ASSUME    CS:CODE, DS:DATA ;SS:STACK
START:  MOV       AX, DATA
        MOV       DS, AX
        MOV       AX, STACK
        MOV       SS, AX
        MOV       AL, ADR1
        SUB       AL, ADR2
        MOV       AH, 0
        PUSH      AX ;将结果入栈保存,注意栈指令对字操作
        MOV       CL, 4
        ROL       AL, CL
        AND       AL, 0FH ;屏蔽高 4位
        ADD       AL, 30H ;计算 ASCII码
        MOV       DL, AL;输出高位
        MOV       AH, 2
        INT       21H
        POP       AX
        AND       AL, 0FH
        ADD       AL, 30H
        MOV       DL, AL;输出低位
        MOV       AH, 2
        INT       21H
        MOV       AH, 4CH
        INT       21H
CODE    ENDS
        END       START
```

2. 分支结构程序

分支结构是根据某一判断的不同结果执行不同的程序段的程序设计方法。典型的分支结构流程如图 6-5-1所示。

[例 6.65] 设内存中自 BUFFER单元存有 10个字节的数,要求查找是否有 50H这个数,若有显示 FOUND,如果没有则显示 NOT FOUND,并返回 DOS状态。

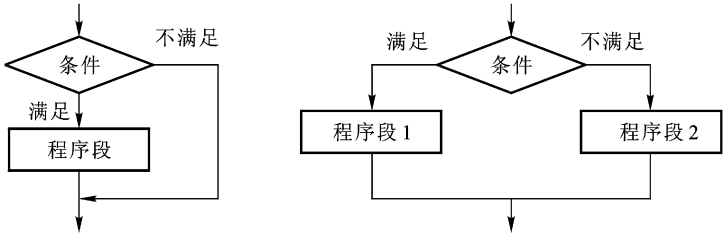


图 6-5-1 分支结构

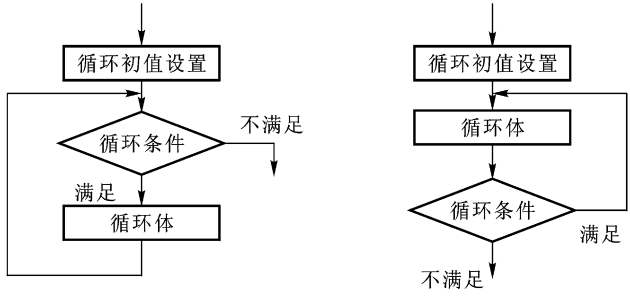
分析:本程序是典型的分支程序,用重复串扫描指令 REPNZ SCASB 查找 50H 这个数,执行操作为当 AL-[DI]结果不为 0且 CX 值不为 0时重复查找。若 AL-[DI]结果为 0,则说明源串中有 50H 这个数,则转到标号 FOUND去执行,在显示器上显示字符串 FOUND;若循环次数已经为 0时,结果仍不为 0,则表示源串中没有要找的 50H 这个数,则顺序执行程序,在显示器上显示字符串 NOT FOUND。

```
DATA    SEGMENT
FSUB    DB  'FOUND','$'
NSUB    DB  'NOT FOUND','$'
DATA    ENDS
EDATA   SEGMENT
BUFFER  DB  12H,34H,56H,78H,90H,50H,77H,88H,99H
EDATA   ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA,ES:EDATA
START:  MOV     AX,DATA
        MOV     DS,AX
        MOV     AX,EDATA
        MOV     ES,AX
        MOV     DI,OFFSET  BUFFER
        MOV     CX,10
        MOV     AL,50H
        REPNZ   SCASB;结果不为 0(没找到)则重复搜索
        JZ      FOUND;找到 50H,转到标号 FOUND
        MOV     AX,DATA
        MOV     DS,AX
        MOV     DX,OFFSET  NSUB;没找到显示 NOT FOUND
        MOV     AH,09H
        INT     21H
```

```
        JMP     EXIT
FOUND:  MOV     AX, DATA ;找到显示 FOUND
        MOV     DS, AX
        MOV     DX, OFFSET FSUB
        MOV     AH, 09H
        INT     21H
EXIT:   MOV     AH, 4CH
        INT     21H
CODE    ENDS
        END     START
```

3. 循环结构程序

循环结构是按给定的条件重复执行一系列指令，直到循环条件不满足为止。循环程序结构有两种，一种是先判断后工作（While循环），另一种是先工作后判断（Until循环），如图 6－5－2 (a)、图 6－5－2 (b) 所示。



(a) 先判断后工作方式　　(b) 先工作后判断方式

图 6－5－2　循环结构

例 6.66】　数组 A包含 30个互不相等的整数，数组 B包含 40个互不相等的整数，将即在数组 A中又在数组 B中出现的整数存放在数组 C中。

分析：本程序是典型的循环程序，且为二重循环。设外层循环次数为 30，内层循环次数为 40，首先让数组 A中第一个数和数组 B中的所有数相比较，若相等则存入数组 C中，若不相等则让数组 A中下一个数和数组 B中的所有数相比较，依此类推，直到都比较完为止。

```
DATA    SEGMENT
  A     DW   30 DUP (?)
  B     DW   40 DUP (?)
  C     DW   30 DUP (?)
DATA    ENDS
CODE    SEGMENT
        ASSUME    CS:CODE, DS:DATA
```

```
MAIN    PROC    FAR
START:  PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        MOV     SI, 0
        MOV     BX, 0
        MOV     CX, 30 ;外层循环次数
NEXT1:  PUSH    CX ;保存外层循环次数
        MOV     DI, 0
        MOV     CX, 40 ;内层循环次数
        MOV     AX, A [SI]
NEXT2:  CMP     B [DI], AX ;比较两数组中的数
        JNE     L1;不相等,转 L1取数组 B中下一个数
        MOV     C [BX] , AX ;相等则存入数组 C中
        ADD     BX, 2
        JMP     L2
L1:     ADD     DI, 2 ;取数组 B中下一个数
        LOOP    NEXT2
L2:     ADD     SI, 2 ;取数组 A中下一个数
        POP     CX ;恢复外层循环次数
        LOOP    NEXT1
        RET
MAIN    ENDP
CODE    ENDS
        END     START
```

4. 子程序结构

把经常使用的程序段编写成子程序,可以使程序可读性增强并趋于程序设计模块化。子程序的调用和返回可用指令 CALL和 RET实现,CALL指令在主程序中,而 RET指令则在被调用子程序的末尾。另外由于子程序执行时要用到某些寄存器,而主程序也可能正在使用这些寄存器,所以要考虑现场信息的保护。例如,主程序中正在使用 AX、BX、CX 寄存器,子程序需要使用 BX、CX 寄存器,这时就要将 BX、CX 的内容压入堆栈,待子程序返回后,再从堆栈中恢复 BX、CX 的内容。

方法一 在主程序中进行信息保护。

```
PUSH    BX
PUSH    CX
CALL    SUB1
```

```
    POP     CX
    POP     BX
```

方法二 在子程序中进行信息保护。

```
SUB1  PROC  NEAR
      PUSH  BX
      PUSH  CX
      …             ;子程序功能段
      POP   CX
      POP   BX
      RET
SUB1  ENDP
```

[例 6.67] 实现将 10个 8位的无符号数按递减次序排序,并将结果显示出来。

分析:本程序采用冒泡算法,共有两个循环体。内循环进行两两相邻比较,如果符合递减次序则不交换,否则就相互交换。外循环设置一个交换标志位 (BL),每次进入外循环就设置为 0,而内循环每作一次交换就将该标志设置为 1,在每次内循环结束后就测试交换标志,如果该位为 1,则再一次进入外循环;如果该位不为 1,则说明上一轮比较已经不再引起交换操作,数组已经排序完毕,这样就立即结束外循环。当实现排序后将排好的结果调用显示子程序,将结果显示在显示器上。

```
DATA    SEGMENT
ARRAY   B       05H,78H,0F8H,7BH,00H,8CH,20H,0A0H,80H,60H
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
MAIN    PROC    FAR
START:  PUSH    DS
        XOR     AX,AX
        PUSH    AX
        MOV     AX,DATA
        MOV     DS,AX
  AB1:  MOV     SI,OFFSET ARRAY
        MOV     BL,0;交换标志 BL置 0
        MOV     CX,9
AGAIN:  MOV     AL,[SI]
        INC     SI
        CMP     AL,[SI] ;两两相比较
        JNC     CD1;若前一个数大于等于后一个数则转到 CD1去执行
        MOV     AH,[SI] ;若前一个数小于后一个数则顺序执行程序来实现交换
        MOV     [SI],AL
```

```
        DEC     SI
        MOV     [SI], AH
        INC     SI
        MOV     BL, 1
CD1:    MLOOP   AGAIN
        DEC     BL
        JZ      AB1;BL值减 1后为 0说明排序没完成,再一次进入外循环
        MOV     SI, OFFSET ARRAY;BL值减 1后不为 0则排序已完成需显示
        MOV     DI, 10
LP11:   MOV     BL, [SI]
        CALL    DISPL;调用显示子程序
        INC     SI
        DEC     DI
        JNZ     LP11
        RET
MAIN    ENDP
DISPL   PROC    NEAR;显示子程序
        MOV     DH, 2;显示两位十六进制数
LOP2:   MOV     CL, 4
        ROR     BL, CL
        MOV     DL, BL
        AND     DL, 0FH
        ADD     DL, 30H
        CMP     DL, 3AH 判断是否为 A～F之间的数
        JB      DIP;0～9之间的数直接显示
        ADD     DL, 7;A～F之间的数 ASCII值加 7
DIP:    MOV     AH, 2;显示字符
        INT     21H
        DEC     DH
        JNZ     LOP2
        MOV     DL, 20H;显示空格符
        MOV     AH, 2
        INT     21H
        RET
DISPL   ENDP
CODE    ENDS
        END     START
```

习 题

1. 解释下列名词

(1) 指令 (2) 指令系统 (3) 寻址方式

2. 8086的寻址方式有哪几类,用哪一种寻址方式的指令执行速度最快?

3. 直接寻址方式中,一般只指出操作数的偏移地址,那么,段地址如何确定?如果要用某个段寄存器指出段地址,指令中应如何表示?

4. 说明下列指令分别用了哪些寻址方式。

(1) ADD AL,34H

(2) CMP [SI+3],BL

(3) MOV [BX+SI+2],AX

(4) INC WORD PTR [BX]

(5) MOV ARRAY[SI],AX

(6) MOV ES:[120H],AL

5. 说明下列指令是否正确。

(1) MOV AX,NUMBER

(2) ADD AX,DL

(3) INC CX

(4) DEC [SI+2]

(5) ROL WORD PTR [BX]

(6) PUSH WORD PTR [SI+3]

6. 指出下列指令的错误。

(1) MOV BL,CX

(2) MOV [BX],[DI]

(3) MOV AX,[BX][BP]

(4) MOV AX,COUNT[BX][SI],ES:DX

(5) MOV BYTE PTR[SI],2000

(6) MOV BX,OFFSET BLOCK[BX]

(7) MOV CS,DX

7. 设有关寄存器及存储单元的内容如下:(DS)=2000H,(BX)=0100H,(SI)=0002H,(20100)=12H,(20101)=34H,(20102)=56H,(20103)=78H,(21200)=2AH,(21201)=4CH,(21202)=0B7H,(21203)=65H。

试指出下列各条指令单独执行完后 AX寄存器的内容。

(1) MOV AX,1234H (2) MOV AX,BX

(3) MOV AX,[1200H] (4) MOV AX,[BX]

(5) MOV AX,1100[BX] (6) MOV AX,[BX][SI]

(7) MOV AX,1100[BX][SI]

8. 用加法指令设计一个简单程序,实现 2个 10位十进制数的相加,结果放在被加数单元。

9. 为什么用增量或减量指令设计程序时,在这类指令后面不用进位标志作为判断依据?

10. 用普通运算指令执行 BCD码运算时,为什么要进行十进制调整,具体讲,在对 BCD码的加、减、乘、除运算时,程序段的什么位置上必须加上十进制调整指令?

11. 在串操作指令使用时，特别要注意 SI、DI这两个寄存器及方向标志 DF。试具体就指令 MOVSB／MOVSW、CMPSB／CMPSW、SCASB／SCASW、LODSB／LODSW、STOSB／STOSW 列表说明与 SI、DI及 DF的关系。

12. 用串操作指令设计实现如下功能的程序段，首先将 300个数从 1000H处移到 2000H处，再从中检索等于 AL中字符的单元，并将此单元的值换成空格符。

13. 设当前 SS=2010H，SP=FE00H，BX=3457H，计算当前栈顶地址为多少，当执行 PUSH BX后，栈顶地址和栈顶两个字节的内容分别是什么？

14. 下列程序段完成什么工作？

```
DATX1  DB    300 DUP (?)
DATX2  DB    100 DUP (?)
       MOV   CX, 100
       MOV   BX, 200
       MOV   SI, 0
       MOV   DI, 0
NEXT:  MOV   AL, DATX1 [BX] [SI]
       MOV   DATX2 [DI], AL
       INC   SI
       INC   DI
       LOOP  NEXT
```

15. 执行下列指令后，AX寄存器中的内容是什么？

```
TAB    DW   12H, 34H, 56H, 78H, 9AH
ARR    DW   3
       MOV  BX, OFFSET TABLE
       ADD  BX, ENTRY
       MOV  AX, [BX]
```

16. 用移位指令实现将 AL寄存器内容乘以 12。

17. 假定 AX和 BX中的内容为带符号数，CX和 DX中的内容为无符号数，试用比较指令和条件转移指令实现以下判断。

(1) 若 DX的值超过 CX的值，则转去执行 L1。

(2) 若 BX的值大于 AX的值，则转去执行 L2。

(3) CX中的值为 0吗？若是则转去执行 L3。

(4) BX的值与 AX的值进行比较，会产生溢出吗？若溢出转 L4。

(5) 若 BX的值小于 AX的值，则转去执行 L5。

(6) 若 DX的值低于 CX的值，则转去执行 L6。

18. 假如在程序的括号中分别填入指令：

(1) LOOP L1　(2) LOOPNZ L1　(3) LOOPZ L1

说明在 3种情况下，当程序执行后，AX、BX、CX、DX这 4个寄存器的内容分别是什么？

```
L1:  OV    AX, 01
     MOV   BX, 02
     MOV   DX, 03
     MOV   CX, 04
L1:  INC   AX
```

```
ADD     BX, AX
SHR     DX, 1
(       )
HLT
```

19. 编写程序，从长度为 60的无符号字数组中找出最小的数，存于变量 MIN中。
20. 编写程序，比较两个字符串是否相同，若相同，显示 YES,否则显示 NO。
21. 编写程序，将从键盘输入的大写字母转换为小写字母。
22. 编写程序，统计数据区中 0、正数、负数的个数，结果存放在 C1、C1、C2中。